ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΑΤΡΩΝ
UNIVERSITY OF PATRAS

ΣΧΟΛΗ ΑΝΘΡΩΠΙΣΤΙΚΩΝ ΚΑΙ ΚΟΙΝΩΝΙΚΩΝ ΕΠΙΣΤΗΜΩΝ

ΤΜΗΜΑ ΜΟΥΣΕΙΟΛΟΓΙΑΣ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

ΘΕΜΑ :

«ΜΕΛΕΤΗ ΚΑΙ ΚΑΤΑΣΚΕΥΗ ΗΛΕΚΤΡΟΝΙΚΟΥ ΠΑΙΧΝΙΔΙΟΥ ΓΙΑ ΤΗΝ ΨΥΧΑΓΩΓΙΑ ΤΩΝ ΠΑΙΔΙΩΝ ΣΕ ΞΕΝΟΔΟΧΕΙΑΚΕΣ ΜΟΝΑΔΕΣ »

ΔΗΜΗΤΡΗΣ ΚΑΤΣΙΦΑΡΑΚΗΣ
ΕΥΑΓΓΕΛΙΑ ΤΣΙΛΙΓΙΑΝΝΗ

ΕΠΙΒΛΕΠΩΝ ΚΑΘΗΓΗΤΗΣ : ΠΑΠΑΔΟΠΟΥΛΟΣ ΔΗΜΗΤΡΙΟΣ

ΠΑΤΡΑ 2020

ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΑΤΡΩΝ
UNIVERSITY OF PATRAS

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

ΜΕΛΕΤΗ ΚΑΙ ΚΑΤΑΣΚΕΥΗ ΗΛΕΚΤΡΟΝΙΚΟΥ ΠΑΙΧΝΙΔΙΟΥ ΓΙΑ ΤΗΝ ΨΥΧΑΓΩΓΙΑ ΤΩΝ ΠΑΙΔΙΩΝ ΣΕ ΞΕΝΟΔΟΧΕΙΑΚΕΣ ΜΟΝΑΔΕΣ

ΟΝΟΜΑΤΕΠΩΝΥΜΟ ΣΠΟΥΔΑΣΤΗ/ΤΩΝ

ΚΑΤΣΙΦΑΡΑΚΗΣ ΔΗΜΗΤΡΙΟΣ

ΤΣΙΛΙΓΙΑΝΝΗ ΕΥΑΓΓΕΛΙΑ

ΕΠΟΠΤΕΥΩΝ ΚΑΘΗΓΗΤΗΣ

ΠΑΠΑΔΟΠΟΥΛΟΣ ΔΗΜΗΤΡΙΟΣ

ΠΑΤΡΑ - 2020

TITLE

Design and construction of an electronic game for the entertainment of children in hotel units

STUDENT NAME/S

KATSIFARAKIS DIMITRIOS

TSILIGIANNI EVANGELIA

SUPERVISING PROFESSOR

PAPADOPOULOS DIMITRIOS

PATRAS - 2020

ΠΡΟΛΟΓΟΣ

Η παρούσα Πτυχιακή εργασία με τίτλο " Μελέτη και κατασκευή ηλεκτρονικού παιχνιδιού για την ψυχαγωγία των παιδιών σε ξενοδοχειακές μονάδες" εκπονήθηκε στα πλαίσια της ολοκλήρωσης των προϋποθέσεων, για την λήψη του πτυχίου μας από το Ίδρυμα του πανεπιστημίου Πατρών (πρώην Δ.Ο.Ε.Π & Τ.Μ του Τ.Ε.Ι Δυτικής Ελλάδας), με έδρα τον Πύργο του Ν. Ηλείας. Η ανάληψη της εργασίας ορίστηκε τον Δεκέμβριο του 2018 και η ολοκλήρωση της πραγματοποιήθηκε εντός των προβλεπόμενων ορίων, τον Δεκέμβριο του 2020.

Η εργασία πραγματοποιήθηκε υπό την επίβλεψη του κ. Παπαδόπουλου Δημήτρη, Καθηγητή του τμήματος Διοίκηση Οικονομίας Επικοινωνίας Πολιτιστικών και Τουριστικών μονάδων.

Ο σκοπός μας στην εν λόγω Πτυχιακή εργασία, ήταν όσο το δυνατόν να πραγματοποιήσουμε σωστή και σαφή ανάλυση του θέματος μας στο θεωρητικό κομμάτι και την δημιουργία ενός ευχάριστου παιχνιδιού για την ψυχαγωγία των παιδιών.

Θα θέλαμε να ευχαριστήσουμε τον επιβλέποντα κ. Παπαδόπουλο Δημήτρη για τις χρήσιμες ιδέες του που μας βοήθησαν για το καλύτερο αποτέλεσμα της πτυχιακής.

Τέλος, θα θέλαμε να αναφέρουμε ότι σε αυτά τα δύο χρόνια ολοκλήρωσης της Πτυχιακής εργασίας μας συνέβησαν, το 2019 ο γάμος μας και το 2020 ο ερχομός του μωρού μας, Μιχαήλ, τώρα πια τέλος του 2020 φτάνει στην ολοκλήρωση του ένα μεγάλο κεφάλαιο της ζωής μας, οι σπουδές μας.

ΠΕΡΙΛΗΨΗ

Η Πτυχιακή Εργασία με τίτλο ¨Μελέτη και κατασκευή ηλεκτρονικού παιχνιδιού για την ψυχαγωγία των παιδιών σε ξενοδοχειακές μονάδες¨ έχει ως αντικείμενο την σχεδίαση, ανάπτυξη και υλοποίηση ενός παιχνιδιού για την ψυχαγωγία των παιδιών κατά τη διάρκεια των διακοπών τους στη χώρα μας. Στόχος της παρούσας Πτυχιακής Εργασίας είναι η δημιουργία ενός παιχνιδιού το οποίο θα σχετίζεται με τρεις από τους πιο σημαντικούς Άθλους του Ηρακλή ώστε το εν λόγω παιχνίδι εκτός από διασκέδαση να προσφέρει και γνώσεις σχετικά με την Ελληνική μυθολογία. Πιο αναλυτικά, τα παιδιά που θα έρθουν σε επαφή με το συγκεκριμένο ηλεκτρονικό παιχνίδι θα έχουν την ευκαιρία να γνωρίσουν τον Ηρακλή μέσα από τους πιο διάσημους Άθλους του όπως το Λιοντάρι της Νεμέας, τον Ταύρο της Κρήτης και την Λερναία Ύδρας, συνδυάζοντας την διασκέδαση και την ψυχαγωγία. Η υλοποίηση του παιχνιδιού βασίστηκε στη λογική του παιχνιδιού Super Mario Bros, με κατάλληλες τροποποιήσεις και προσαρμογές ώστε να εξυπηρετεί και τους σκοπούς της μάθησης.

Λέξεις Κλειδιά

Κατασκευή ηλεκτρονικού παιχνιδιού, Ηλεκτρονικό παιχνίδι, Άθλοι του Ηρακλή, Ψυχαγωγία, Τουρισμός

Abstract

The thesis entitled “Study and construction of an electronic game for children entertainment on hotel units”, haw as its object the design, development and implementation of a game for the entertainment of kids during their holidays in our country. The aim of this thesis is to create a game which will be related to three of the most famous labors of Hercules, so that this game in addition to entertainment, also offers knowledge about Greek mythology. In more detail, the children who come in contact with this electronic game will have the opportunity to get to know Hercules through his most famous labors as the “Nemea Lion”, “Taurus of Crete” and “Lernaean Hydra”, combining fun and entertainment. The implementation of the game was based on the logic of the game “Super Mario Bros” with appropriate modifications and adaptations to serve the learning proposes.

Keywords

Game development, electronic game, Labors of Hercules, entertainment, tourism

ΠΕΡΙΕΧΟΜΕΝΑ

1. Τουρισμός

Στο εισαγωγικό αυτό κεφάλαιο γίνεται μία σύντομη παρουσίαση της έννοια του τουρισμού. Συγκεκριμένα στην ενότητα 1.1 δίνεται ο ορισμός του τουρισμού, ενώ στην ενότητα 1.2 γίνεται μία ιστορική αναδρομή. Στη ενότητα 1.3 παρουσιάζονται τα σημαντικότερα ήδη και μορφές του τουρισμού, ενώ στην ενότητα 1.4 γίνεται μία σύντομη αναφορά στην θετική επίδραση που είχε ανάπτυξη της τεχνολογίας και των τηλεπικοινωνιών στον τουρισμό.

1.1 Ορισμός

Ο όρος τουρισμός έχει αγγλική προέλευση από τον αντίστοιχο όρο «touring» που με τη σειρά του έχει τις ρίζες του στην γαλλική γλώσσα και τη λέξη «tour» που ερμηνεύεται ως περιοδεία ή ταξίδι. Ανατρέχοντας κάποιος στην βιβλιογραφία, θα διαπιστώσει ότι υπάρχει μία πληθώρα ορισμών για την έννοια του τουρισμού, το οποίο αναδεικνύει και την πολυπλοκότητά ως φαινόμενο. Ο τουρισμός, σύμφωνα με την Διεθνή Ένωση Επιστημόνων Τουρισμού (International Association of Scientific Experts on Tourism – AIEST), αποτελεί το σύνολο των φαινομένων και των σχέσεων που προκύπτουν από το ταξίδι και τη διαμονή μη μόνιμων κατοίκων σε έναν προορισμό, εφόσον δεν οδηγούν σε μόνιμη διαμονή στον προορισμό και δεν συνδέονται με κάποια κερδοσκοπική δραστηριότητα. Σύμφωνα με τον Παγκόσμιο Οργανισμό Τουρισμού (ΠΟΤ), ο τουρισμός περιλαμβάνει τις δραστηριότητες των ανθρώπων που ταξιδεύουν και διαμένουν σε προορισμούς και περιοχές πέραν αυτών που αποτελούν το συνηθισμένο περιβάλλον τους και για χρονικό διάστημα που δεν υπερβαίνει τον έναν χρόνο και έχει ως στόχο την αναψυχή, την ικανοποίηση των επαγγελματικών τους αναγκών κ.α.

1.2 Ιστορική αναδρομή

Η ανάγκη του ανθρώπου για εξερεύνηση και ανακάλυψη νέων τόπων καθώς και η απόκτηση νέων εμπειριών ξεφεύγοντας από το γνωστό και καθημερινό είναι γνωστή και αποτελεί μέρος της ανθρώπινης συμπεριφοράς. Η ανάγκη αυτή ξεκινά από την αρχαιότητα συνεχίζεται στον μεσαίωνα και την αναγέννηση και φτάνει ως τις μέρες μας. Σε καθ' ένα από τα στάδια της εξέλιξης του τουρισμού σημαντικό ρόλο παίζουν η επικρατούσα τεχνολογία και τα υπάρχοντα μεταφορικά μέσα, οι επικρατούσες συνθήκες (πόλεμοι, δικτατορικά καθεστώτα) καθώς και η επικοινωνία. (Λαγός, 2003)

Κατά την αρχαιότητα, η έννοια του τουρισμού συνδέθηκε με την ανάγκη των ανθρώπων για ανακάλυψη νέων, ανεξερεύνητων περιοχών προκειμένου να επεκτείνουν τις εμπορικές τους δραστηριότητες. Τα πρώτα ταξίδια χρονολογούνται στα μέσα της τέταρτης χιλιετίας με τους Σουμέριους να είναι οι πρώτοι που τα πραγματοποιούν. Το παράδειγμα των Σουμέριων θα ακολουθήσουν και άλλοι λαοί όπως οι Αιγύπτιοι και οι Κρήτες κατά τη διάρκεια της Μινωικής περιόδου. Έπειτα θα ακολουθήσουν οι Ρωμαίοι και οι Βυζαντινοί κατά την ακμή των αυτοκρατοριών τους. Οι πρώτοι ταξιδιώτες ήταν συνήθως έμποροι ή επιστήμονες και συνδύαζαν τα ταξίδια τους με το επάγγελμά τους. Καθοριστικό ρόλο για την ανάπτυξη της πρώιμης αυτής μορφής του τουρισμού έπαιξε η ανάπτυξη της σφηνοειδούς γραφής, του

χρήματος και της ρόδας από τους Σουμέριους δίνοντάς τους τον τίτλο των ιδρυτών του τουρισμού.

Κατά τον Μεσαίωνα οι μετακινήσεις των ανθρώπων αυξάνονται, παρά το γεγονός της ύπαρξης πειρατείας και ληστειών που έθεταν σε κίνδυνο τους ταξιδιώτες. Τα ταξίδια πραγματοποιούνταν εκτός από εμπορικούς σκοπούς και για εκπαιδευτικούς ή θρησκευτικούς λόγους, ενώ αυτή την περίοδο ανακαλύφθηκαν η Κίνα και η Αμερική. Την περίοδο της Αναγέννησης υπήρξε στην Ευρώπη ανάπτυξη των οργανωμένων ταξιδιών, τα οποία αφορούσαν κατά κύριο λόγο αριστοκράτες προκειμένου να διευρύνουν τους πνευματικούς τους ορίζοντες και να γίνουν δεκτοί στις αυλές των βασιλιάδων. Αυτό είχε ως αποτέλεσμα να αρχίσουν να αναπτύσσονται δομές του τουρισμού όπως ξενοδοχεία, δρόμοι και μεταφορικά μέσα. Έτσι αρχίζει σιγά σιγά η ανάπτυξη τουριστικών πακέτων με κάποιες ιδιαίτερες παροχές όπως αυτή του ξεναγού. Παράλληλα, άρχισαν και οι μεγάλες εξερευνήσεις στην Αμερική στην Ασία, την Αφρική και την Αυστραλία.

Τον 19ο αιώνα η εμφάνιση του σιδηροδρόμου οδήγησε στην μαζικοποίηση του τουρισμού. Το 1841, στην Αγγλία ο Thomas Cook οργάνωσε το πρώτο ταξίδι με οδηγούς ενώ το 1845 δημιούργησε το ταξιδιωτικό γραφείο "Thomas Cook & Son". Τα χρόνια που ακολούθησαν, ο Thomas Cook θα ιδρύσει υποκαταστήματα σε 68 χώρες ανά τον κόσμο και θα γίνει ο πρώτος επαγγελματίας τουριστικός πράκτορας, οργανώνοντας ταξίδια με χαμηλό κόστος προσελκύοντας πελάτες που προέρχονταν κυρίως από τη μεσαία τάξη. Επιχειρηματίες στην Ευρώπη και στην Αμερική μιμήθηκαν το επιχείρημα του Cook ιδρύοντας μεγάλα τουριστικά πρακτορεία κάνοντας τα οργανωμένα ταξίδια πολύ δημοφιλείς.

Σε αντίθεση με τον 19^{ο} αιώνα όπου ο τουρισμός γνώρισε άνθιση και έγινε προσιτός για το ευρύ κοινό, ο 20^{ος} αιώνας αποδείχθηκε ιδιαίτερα δύσκολος για το κλάδο. Η οικονομική κρίση κατά την περίοδο 1905 – 1914, ο Α΄ Παγκόσμιος πόλεμος 1914- 1918, ο πληθωρισμός και η οικονομική κρίση του 1929 οδήγησαν στην καταστροφή της αριστοκρατικής και της μεσαίας τάξης, με αποτέλεσμα ο τουρισμός να δεχθεί ισχυρό πλήγμα. Μετά από μια μακρά περίοδο ύφεσης, ο τουρισμός άρχισε να αναπτύσσεται και να εξελίσσεται. Σημαντικό ρόλο σε αυτό έπαιξαν αφενός το τέλος του Παγκοσμίου πολέμου και αφετέρου η είσοδος του αεροπλάνου και του αυτοκινήτου στις μετακινήσεις, οι οποίες μέχρι τότε γινόντουσαν κυρίως με τρένα και ατμόπλοια. Τα παραπάνω, σε συνδυασμό με την κατασκευή οδικών δικτύων οδήγησαν στην ανάπτυξη του εσωτερικού κυρίως τουρισμού.

Τις δεκαετίες που ακολούθησαν η εξέλιξη τόσο των αεροπλάνων, όσο και των υπολοίπων μέσων μεταφοράς οδήγησε σε αλματώδη ανάπτυξη του τουριστικού προϊόντος. Πολλές περιοχές αναπτύχθηκαν και στήριξαν αυτήν τους την ανάπτυξη στον τουρισμό εκμεταλλευόμενοι τον φυσικό και πολιτιστικό πλούτο της περιοχής τους. Η ανέγερση ξενοδοχειακών μονάδων και η ανάπτυξη οδικών αξόνων οδήγησαν επίσης προς αυτή την κατεύθυνση. Σχεδόν όλα τα ανεπτυγμένα κράτη προωθούν τον τουρισμό τους ως οικονομική δραστηριότητα διαμορφώνοντας το κατάλληλο θεσμικό πλαίσιο, το οποίο θα οργανώσει και θα ενισχύσει το τουριστικό τους προϊόν. Ειδικότερα, μετά το 1950 ο τουρισμός αναπτύσσεται με αλματώδεις ρυθμούς και επηρεάζει σημαντικά την οικονομική ανάπτυξη των χωρών οι οποίες επενδύουν σ' αυτόν. Για την ραγδαία αυτή ανάπτυξη του τουρισμού σημαντικό ρόλο έπαιξαν η πολιτική και οικονομική σταθερότητα των ανεπτυγμένων χωρών, η ανάπτυξη του

«κοινωνικού κράτους», η βελτίωση των μεταφορικών μέσων και υποδομών, καθώς και η δημιουργία τουριστικών γραφείων και πρακτορείων.

Την τελευταία δεκαετία, ο τουρισμός έχει αναβαθμιστεί ακόμα περισσότερο με την χρήση της τεχνολογίας. Συγκεκριμένα, η ανάπτυξη και διάδοση του διαδικτύου στο σύνολο σχεδόν του πληθυσμού των αναπτυγμένων κρατών οδήγησε στην ανάπτυξη εφαρμογών οι οποίες διευκολύνουν τους επίδοξους ταξιδιώτες στην οργάνωση του ταξιδιού τους. Εφαρμογές αναζήτησης ξενοδοχείων βάσει των κριτηρίων των ταξιδιωτών, αναζήτησης πτήσεων και άλλων μεταφορικών μέσων, εφαρμογές αξιολόγησης καταστημάτων εστίασης καθώς και η χρήση του πλαστικού χρήματος έναντι του παραδοσιακού συναλλάγματος, έχουν δώσει τεράστια ώθηση στο τουριστικό προϊόν βελτιώνοντας ταυτόχρονα την εμπειρία των τουριστών.

1.3 Είδη και μορφές τουρισμού

Ο τουρισμός σαν προϊόν, διακρίνεται σε διάφορα είδη και μορφές που διαμορφώνονται με βάση τα γεωγραφικά και τα κοινωνικοοικονομικά χαρακτηρίστηκα καθώς και από την συμπεριφορά των τουριστών. Η διάκριση αυτή είναι αναγκαία προκειμένου να διευκολυνθεί η ανάλυσή του και να επιτευχθεί ο βέλτιστος συνδυασμός τουριστικής αγοράς και τουριστικού προϊόντος, με στόχο τόσο τη βελτίωση της εμπειρίας των τουριστών όσο και την αύξηση των κερδών των τουριστικών περιοχών. Οι πιο βασικές μορφές τουρισμού είναι οι ακόλουθες:

- Μαζικός και ατομικός τουρισμός
- Εγχώριος και διεθνής τουρισμός
- Εποχιακός και συνεχής τουρισμός
- Στατικός και κινητικός τουρισμός
- Νεανικός και 3^{ης} ηλικίας τουρισμός
- Κοινωνικός τουρισμός
- Εναλλακτικός τουρισμός

1.4 Η επίδραση της τεχνολογίας στον τουρισμό

Τα τελευταία χρόνια η συμμετοχή της τεχνολογίας και των νέων μορφών επικοινωνίας κερδίζει συνεχώς έδαφος σε όλους τους τομείς και τους κλάδους παραγωγής με τον τουρισμό να μην αποτελεί εξαίρεση. Η εξέλιξη αυτή, άλλαξε άρδην το τοπίο στις επιχειρήσεις, καθώς έδωσαν τα κατάλληλα εργαλεία ώστε όχι μόνο να επεκτείνουν τις δραστηριότητές τους αλλά και να βελτιώσουν την ποιότητα των υπηρεσιών που παρέχουν. Δεν είναι τυχαίο πως σήμερα, ακόμα και οι πιο μικρές ξενοδοχειακές μονάδες διαθέτουν έναν τουλάχιστον ηλεκτρονικό υπολογιστή με μια τυπική σύνδεση στο ίντερνετ ώστε να οργανώσουν και να διαφημίσουν την επιχείρησή τους.

Η ανάπτυξη της πληροφορικής και των τηλεπικοινωνίων έχουν αλλάξει τον τρόπο με τον οποίο ένας τουρίστας οργανώνει και βιώνει τις διακοπές του. Η λογική του τουριστικού πρακτορείου με ολοκληρωμένα πακέτα διακοπών έχει αρχίσει σταδιακά να εγκαταλείπεται. Η

διάδοση του ίντερνετ, έχει ως αποτέλεσμα ο καθένας να έχει εύκολη πρόσβαση στην πληροφορία. Διαδικασίες που πριν λίγες δεκαετίες ήταν δύσκολες όπως η αναζήτηση ξενοδοχείου, αεροπορικών εισιτηρίων, τουριστικών προορισμών και σημείων ενδιαφέροντος τώρα γίνονται μέσα σε λίγα λεπτά.

Την αρχή έκαναν οι αεροπορικές εταιρίες, οι οποίες στην προσπάθειά τους να αυξήσουν την πληρότητα των πτήσεων τους, εφάρμοσαν την τακτική των δημοπρασιών μέσω του διαδικτύου. Βλέποντας το καταναλωτικό κοινό να ανταποκρίνεται θετικά, προχώρησαν σε διεύρυνση των απευθείας πωλήσεων και σε άλλες λειτουργίες τους. Το αποτέλεσμα ήταν ευεργετικό για τον τουρισμό, καθώς όχι μόνο μειώθηκε σημαντικά το κόστος μεταφοράς με αεροπλάνο αλλά αυτοματοποιήθηκε και η διαδικασία αναζήτησης και αγοράς αεροπορικών εισιτηρίων που μέχρι τότε ήταν μια αρκετά χρονοβόρα και δαπανηρή διαδικασία. Η μείωση του κόστους των μεταφορών, ήταν ιδιαίτερα καθοριστικός παράγοντας στην ανάπτυξη του τουριστικού προϊόντος. Οι μεγάλες ξενοδοχειακές επιχειρήσεις από την άλλη, βλέποντας την αποδοχή των διαδικτυακών υπηρεσιών από το κοινό, ακολούθησαν την ίδια τακτική παρέχοντας υπηρεσίες ηλεκτρονικής κράτησης δωματίων. Η τακτική αυτή υιοθετήθηκε σιγά σιγά και από άλλες μικρότερες ξενοδοχειακές μονάδες, οι οποίες είδαν με αυτό τον τρόπο τον τζίρο τους να αυξάνεται.

Σε όλα αυτά ήρθε να προστεθεί και η εμφάνιση των smartphones, τα οποία άλλαξαν τον τρόπου που ζούμε. Όπως είναι φυσικό, ο τουρισμός δεν θα μπορούσε να μην επηρεαστεί από ένα τέτοιο γεγονός. Τα τελευταία χρόνια έχουν αναπτυχθεί πολλές εφαρμογές προσανατολισμένες στο να κάνουν την εμπειρία του τουρίστα όσο το δυνατόν καλύτερη. Έτσι, σήμερα συναντάει κανείς εφαρμογές αναζήτησης ξενοδοχείων, αεροπορικών πτήσεων κλπ οι οποίες δίνουν τη δυνατότητα στον καθένα να έχει πρόσβαση σε όλες τις πληροφορίες, ώστε να οργανώσει το ταξίδι του σύμφωνα με τις δικές του ανάγκες και επιθυμίες. Επιπλέον, κάθε χρήστης μπορεί να μοιραστεί τις εμπειρίες του από έναν προορισμό, σε ειδικές εφαρμογές και μέσα κοινωνικής δικτύωσης. Εφαρμογές τύπου Trip Advisor, επιτρέπουν στους τουρίστες να μοιραστούν τις εμπειρίες τους για αξιοθέατα, ξενοδοχεία, εστιατόρια και άλλα, αφήνοντας παράλληλα μία κριτική για τους χώρους που επισκέφτηκαν. Οι πληροφορίες αυτές θα χρησιμοποιηθούν από άλλους τουρίστες σε μεταγενέστερο χρόνο, οι οποίοι θα μπορέσουν να αποφύγουν τυχόν κακοτοπιές βελτιώνοντας σημαντικά την ποιότητα των διακοπών τους. Επιπλέον, οι τουριστικές επιχειρήσεις παίρνουν ανατροφοδότηση για την ποιότητα των υπηρεσιών που παρέχουν, βοηθώντας τες στο να γίνονται καλύτερες. Τέλος, εφαρμογές όπως η Airbnb, δίνουν τη δυνατότητα τους τουρίστες να επιλέξουν ακόμα πιο οικονομικά καταλύματα για τις διακοπές τους, συμβάλλοντας στην αύξηση του αριθμού των τουριστών. Ως αποτέλεσμα, ο τελικός κερδισμένος είναι ο πελάτης – τουρίστας ο οποίος έχοντας όλη αυτή την πληροφορία μπορεί να οργανώσει τις ιδανικές διακοπές γι’ αυτόν.

2. Ο Τουρισμός στην Ελλάδα

Στο κεφάλαιο αυτό παρουσιάζεται και αναλύεται το τουριστικό προϊόν της Ελλάδας καθώς και πως αυτό επηρεάζει την εγχώρια οικονομία και ανάπτυξή της. Συγκεκριμένα στην ενότητα 2.1 γίνεται μία ιστορική αναδρομή του τουρισμού στην χώρας μας και πως αυτός εξελίχθηκε στο πέρασμα των χρόνων. Στην ενότητα 2.2 παρουσιάζεται ο τουρισμός που υπάρχει σήμερα στην χώρα ενώ στην ενότητα 2.3 δίνονται κάποια αριθμητικά στοιχεία τα οποία αποδεικνύουν τις ευεργετικές συνέπειες που έχει ο τουρισμός για την χώρα μας.

2.1 Ιστορική αναδρομή

Τα ταξίδια προς την Ελλάδα έχουν τις ρίζες τους βαθιά στους αιώνες, όταν επιστήμονες της εποχής, προερχόμενοι κυρίως από την Βρετανία, ερχόντουσαν στην χώρα μας για εκπαιδευτικούς σκοπούς. Τα γραπτά των αρχαίων ιστορικών και φιλοσόφων οδήγησαν περιοχές με ιδιαίτερη αρχαιολογική και ιστορική αξία όπως η Αθήνα, η αρχαία Ολυμπία οι Δελφοί και οι Μυκήνες να δέχονται ταξιδιώτες με τα ανάλογα ενδιαφέροντα.

Μετά την περίοδο της τουρκοκρατίας και την απελευθέρωση του ελληνικού κράτους, γεννήθηκε η ιδέα της αναβίωσης της αρχαίας Ελλάδας, προσελκύοντας πολλούς επιχειρηματίες και διπλωμάτες από ολόκληρη την Ευρώπη. Το σημείο όμως που θεωρείται η αρχή της προόδου για τον ελληνικό τουρισμό είναι η αναβίωση των Ολυμπιακών Αγώνων (1896) καθώς την περίοδο εκείνη έκαναν την εμφάνισή τους τα πρώτα ξενοδοχεία σε μεγάλες πόλεις τις Ελλάδας όπως η Αθήνα, η Κέρκυρα και το Ναύπλιο. Επίσης, εκείνη την περίοδο άρχισαν να πρωτοεμφανίζονται στην Ελλάδα τα πρώτα τουριστικά επαγγέλματα όπως ταξιδιωτικά γραφεία και πρακτορεία. Παρ΄ όλα αυτά, στην οποία ανάπτυξη του τουρισμού, το κράτος δεν είχε καμία συμμετοχή, με Έλληνες και ξένους επιχειρηματίες να προσπαθούν να σηκώσουν από το μηδέν το τουριστικό προϊόν της χώρας (Δρίτσας, 2003).

Ο νόμος 4377 του 1929 έδωσε μία ώθηση στον ελληνικό τουρισμό δίνοντας σημασία στην βελτίωση των υποδομών. Το γεγονός αυτό, οδήγησε στην ανέγερση κάποιων από τα καλύτερα και πολυτελέστερα ξενοδοχεία της Αθήνας. Παράλληλα οι ελληνικές τράπεζες άρχισαν να χρηματοδοτούν την τουριστική αγορά δίνοντας δάνεια για την δημιουργία ξενοδοχειακών μονάδων με κάποιες διευκολύνσεις. Σημειώνεται, ότι την περίοδο εκείνη οι περιοχές με την μεγαλύτερη τουριστική ανάπτυξη ήταν αυτές που ήταν εύκολα προσβάσιμες με τον σιδηρόδρομο, όπως η Θεσσαλονίκη. Σ΄ αυτό το σημείο αξίζει να τονιστεί ότι την περίοδο εκείνη δεν είχε αναπτυχθεί ακόμα ο θαλάσσιος τουρισμός, καθώς δεν υπήρχαν ξενοδοχειακές μονάδες σε παραλιακές περιοχές παρά μόνο σε περιοχές με έντονο αρχαιολογικό και ιστορικό ενδιαφέρον.

Σημαντικός σταθμός στην ιστορία του ελληνικού τουρισμού, ήταν η ίδρυση το Ελληνικού Οργανισμού Τουρισμού (Ε.Ο.Τ) από τον Ελευθέριο Βενιζέλο το 1929. Ο Ε.Ο.Τ. πριν την κατάργηση του το 1936 από τον Ι. Μεταξά, ο οποίος τον αντικατέστησε με το Υφυπουργείο Τύπου και Τουρισμού, οργάνωσε τα πρώτα διαφημίστηκα προγράμματα και εξέδωσε έντυπα και αφίσες και μάλιστα σε δύο ξένες γλώσσες. Επίσης τοποθέτησε τουριστικά περίπτερα σε

αρχαιολογικούς χώρους και θέσπισε τις προδιαγραφές που θα έπρεπε να πληρούν τα ξενοδοχεία.

Το έργο του Ε.Ο.Τ. ήρθε να συνεχίσει το Υπουργείο Τύπου και Τουρισμού από το 1936 και έπειτα, το οποίο ανέλαβε την προώθηση του ελληνικού τουριστικού προϊόντος. Οι συνθήκες όμως που επικρατούσαν, όπως ο Παγκόσμιος Πόλεμος και ο εμφύλιος, αποτέλεσαν σημαντικό τροχοπέδη στην προσπάθεια αυτή. Η επανίδρυση του Ε.Ο.Τ. το 1950 σήμανε και την έναρξη της τουριστικής ανασυγκρότησης της χωράς, με βασικό σκοπό την προσέλκυση ξένων τουριστών. Στα χρόνια που ακολούθησαν, έγιναν σημαντικά έργα υποδομής στην χώρα, όπως η ανέγερση εκατοντάδων ξενοδοχειακών μονάδων, οδικών σταθμών, οργανωμένων ακτών κλπ. Παράλληλα, μέσα από διάφορες δράσεις αναδείχθηκαν τα ελληνικά νησιά και ο θαλάσσιος τουρισμός που μέχρι τότε ήταν ανύπαρκτος. Μία από αυτές τις δράσεις ήταν και η εξασφάλιση παραγωγών του Χόλυγουντ όπως το «Το παιδί και το δελφίνι» και «Τα κανόνια του Ναβαρόνε». Όλα αυτά είχαν ως αποτέλεσμα την ραγδαία ανάπτυξη του τουρισμού στην Ελλάδα, η οποία πολλαπλασίασε τις αφίξεις ξένων τουριστών.

Η ακμή του τουριστικού προϊόντος στην Ελλάδα συνεχίστηκε και κατά τη δεκαετία του 70 και του 80, αυξάνοντας τόσο το πλήθος των ξενοδοχείων όσο και το πλήθος των κατοίκων που είχαν ως κύρια ασχολία τους τον τουρισμό. Το κράτος σε αυτή την περίπτωση, βλέποντας τα πολλαπλά οικονομικά οφέλη, στάθηκε αρωγός επιδοτώντας την κατασκευή κλινών σε ορισμένες περιοχές της επικράτειας.

Με το πέρασμα των χρόνων όμως εκτός από το μέγεθος του τουρισμού, άλλαξε και η μορφή του καθώς σταδιακά ο τουρισμός «πολιτισμού» έδινε την θέση του στον «θαλάσσιο» τουρισμό με αποτέλεσμα η χώρα να θεωρείται σχεδόν αποκλειστικά προορισμός ήλιου και θάλασσας. Έτσι, από το 1985 άλλαξε η τουριστική πολιτική της χώρας προς την κατεύθυνση τη αναβάθμισης της ποιότητας του τουρισμού. Επιχειρήθηκε δηλαδή η σταδιακή απεξάρτηση της Ελλάδας από τον μαζικό τουρισμό και η αξιοποίηση των πλεονεκτημάτων της χώρας για την ανάπτυξη άλλων μορφών τουρισμού όπως ο «θαλάσσιος» τουρισμός. Ταυτόχρονα και δεδομένου ότι η συγκεκριμένη μορφή τουρισμού περιοριζόταν μόνο κατά τους θερινούς μήνες, έγινε προσπάθεια επιμήκυνσης της τουριστικής περιόδου.

Στις μέρες μας, το τουριστικό προϊόν που έχει να αναδείξει η Ελλάδα διαθέτει τόσο τους φυσικούς όσο και τους πολιτιστικούς πόρους καθώς και τις απαραίτητες υποδομές και αποτελεί μία σημαντική τουριστική δύναμη. Ολοένα και περισσότεροι άνθρωποι στην χώρα μας ασχολούνται με τον τουρισμό, ενώ η ανάπτυξη της τεχνολογίας οδήγησε και σε αυτή την περίπτωση στην βελτίωση της ποιότητας των παρεχόμενων υπηρεσιών. Τέλος, οι εναλλακτικές μορφές τουρισμού που αναπτυχθήκαν στη χώρα τα τελευταία χρόνια, όπως ο αγροτουρισμός, ο συνεδριακός και θρησκευτικός τουρισμός είχαν ως αποτέλεσμα όλο και περισσότερες μη παραθαλάσσιες περιοχές, να ασχοληθούν με τον τουρισμό ενισχύοντας το τουριστικό προϊόν της Ελλάδας.

2.2 Ο τουρισμός στην Ελλάδα σήμερα

Η Ελλάδα εδώ και αρκετές δεκαετίες αποτελεί έναν από τους πιο δημοφιλείς τουριστικούς προορισμούς, ιδιαίτερα κατά τους θερινούς μήνες. Οι πλούσιοι φυσικοί πόροι και το μεσογειακό κλίμα που επικρατεί, σε συνδυασμό με τον αρχαιολογικό και ιστορικό της

πλούτο, προσελκύει όλο και περισσότερους τουρίστες απ' όλο τον κόσμο. Η Ελλάδα διαθέσει περισσότερα από 16.000 χιλιόμετρα ακτογραμμής και πάνω από 6.000 νησιά και βραχονησίδες. Οι καθαρές παραλίες, τα επιβλητικά βουνά, η πλούσια ιστορία, η μεσογειακή κουζίνα και η ξακουστή ελληνική φιλοξενία, είναι κάποιες από τις εμπειρίες που αναζητούν και απολαμβάνουν οι τουρίστες που επιλέγουν την Ελλάδα ως τον επόμενο προορισμό τους.

Σε όλα αυτά έρχεται να προστεθεί και η ολοένα αναβαθμισμένες υπηρεσίες και υποδομές. Οι Ολυμπιακοί Αγώνες που πραγματοποιήθηκαν στον Ελλάδα το 2004, εκτός από την διαφήμισή της σε παγκόσμιο επίπεδο, είχε και σαν συνέπεια την ανάπτυξη και αναβάθμιση των υποδομών τους καθώς η διεξαγωγής τους σηματοδότησε την έναρξη μιας σειράς σημαντικών έργων υποδομής που θα δώσουν επιπλέον ώθηση στον ελληνικό τουρισμό. Ο Διεθνής Αερολιμένας Αθηνών έχει αναγνωριστεί ως ένα από τα καλύτερα αεροδρόμια του κόσμου, προσελκύοντας νέες αεροπορικές εταιρείας και ενισχύοντας τον ρόλο της Ελλάδας στον τουριστικό χάρτη. Επίσης, η ολοκλήρωση σημαντικών οδικών υποδομών όπως η Εγνατία Οδός, που συνδέει την Τουρκία με την Ηγουμενίτσα αποτελεί ένα από τα σημαντικότερα έργα υποδομής στην Ευρώπη. Σ' αυτό το έργο έρχεται να προστεθεί και η Ιόνια Οδός που συνδέει τα λιμάνια της Πάτρα και της Ηγουμενίτσα. Το Μετρό και η Αττική Οδός, άλλαξαν τον τρόπο μετακίνησης των πολιτών και των τουριστών στην Αθήνα συμβάλλοντας σημαντικά στην βελτίωση των αστικών μεταφορών. Τέλος, τα τελευταία χρόνια επιχειρείται σημαντική αναβάθμιση στα περισσότερα αεροδρόμια της Ελλάδος ώστε να βελτιώσουν την ποιότητα των υπηρεσιών τους και να προσελκύσουν περισσότερες αεροπορικές εταιρίες, το οποίο θα αυξήσει και τον αριθμό των τουριστών.

2.3 Ο ελληνικός τουρισμός με αριθμούς

Η τουριστική δραστηριότητα επηρεάζει σε μεγάλο βαθμό τόσο την οικονομία και την κοινωνία, όσο και τον πολιτισμό και το φυσικό περιβάλλον. Είναι προφανές ότι όσο μεγαλύτερη είναι η δραστηριότητα αυτή, τόσο μεγαλύτερος θα είναι ο αντίκτυπος στις ανωτέρω κατηγορίες. Ο τουρισμός για την Ελλάδα παίζει καθοριστικό ρόλο στην οικονομική ζωή, καθώς αποτελεί έναν από τους σημαντικότερους πυλώνες οικονομικής ανάπτυξης. Συντελεί στη δημιουργία νέων θέσεων εργασίας, στην καταπολέμηση της ανεργίας και τη μείωση της φτώχειας συμβάλλοντας στην οικονομική ανάπτυξη της χώρας.

2.3.1 Συμμετοχή του τουρισμού στο ΑΕΠ

Η σημασία του τουρισμού για την οικονομία και την ανάπτυξη της Ελλάδας αποτυπώνεται από τη συνεισφορά του στο Ακαθάριστο Εγχώριο Προϊόν (ΑΕΠ) της χώρας. Ο τουρισμός αποτελεί διαχρονικά έναν πολλαπλασιαστή ισχύος της ελληνικής οικονομίας καθώς με το πέρασμα των χρόνων, αποτελεί τον κυριότερο εκφραστή στης αύξησης του ΑΕΠ. Από το 2,9% το 1960, έφτασε να αποτελεί το 30,9% του ΑΕΠ της χώρας το 2018 αποτελώντας τον σημαντικότερο παράγοντα ανάπτυξης. Σε αντίθεση με άλλους κλάδους παραγωγής, ο τουρισμός διατήρησε και ενίσχυσε όλα αυτά τα χρόνια την δυναμική του. Την χρονική περίοδο 1960 έως 1980 συνέβαλε κατά 14,8% στη διαμόρφωση της μεταβολής του ΑΕΠ, ακολουθώντας τον κλάδο της μεταποίησης με 20,7%, του αγροτικού τομέα με 25,4% και της

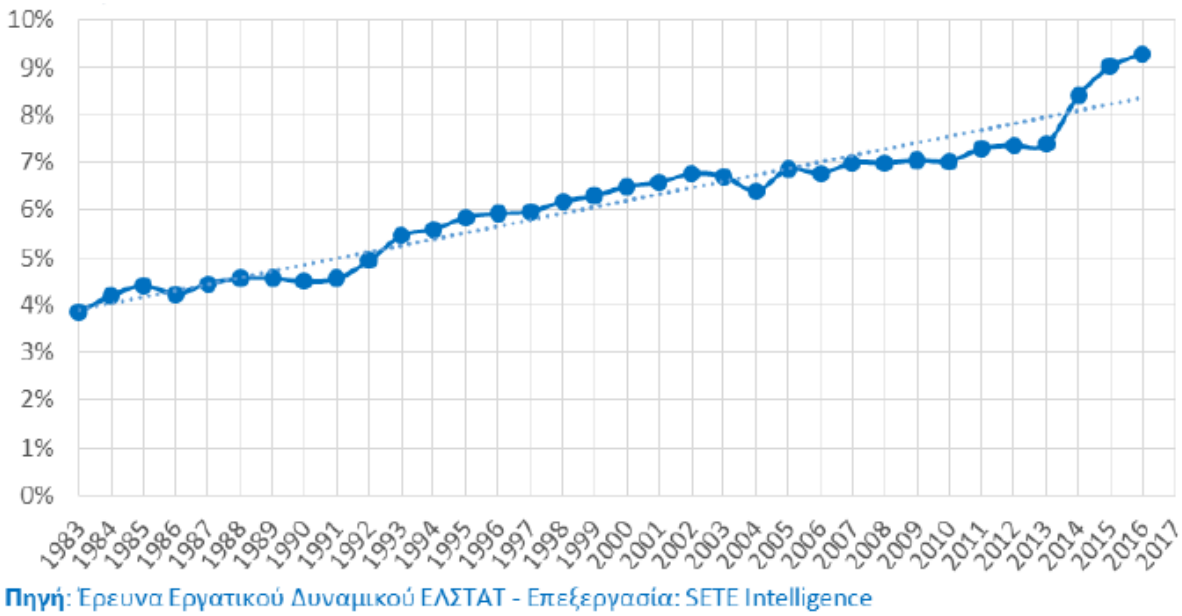
δευτερογενούς παραγωγής με 31,4% (Παυλόπουλος, 1999). Η ανοδική πορεία συνεχίστηκε και τη δεκαετία του 1990 φτάνοντας στο 18,4% το 1997. Αντίθετα τη δεκαετία του 2000 ο τουρισμός θα παρουσιάσει πολλές αυξομειώσεις και η συμμετοχή του στο ΑΕΠ θα κυμανθεί από 15,9% το 2003 και το 2009 έως 17,8% το 2016. Η μεγάλη ανάπτυξη του τουρισμού έρχεται την τελευταία δεκαετία και το 2017 καθώς η συμμετοχή του τουρισμού στο ΑΕΠ της χώρας εκτινάσσεται από το 18,6% του 2016 στο 27,3%. Η δυναμική του τουρισμού συνεχίζεται και το 2018 συμβάλλοντας κατά 30,9% στο ΑΕΠ της Ελλάδας.

Πίνακας 1: Τουρισμός και ΑΕΠ 2010 έως 2018 (ΣΕΤΕ)

Έτος	Συμμετοχή Τουρισμού στο ΑΕΠ (%)	Έσοδα (δις Ευρώ)
2010	16%	9,6
2011	15,8%	10,5
2012	16,4%	10,02
2013	16,3%	11,7
2014	17,3%	13
2015	18,5%	13,6
2016	18,6%	12,7
2017	27,3%	14,2
2018	30,9%	15,6

2.3.2 Απασχόληση

Σημαντική είναι και η επίδραση του τουρισμού και στην απασχόληση και αυτό γιατί ο τουρισμός βασίζεται κυρίως στην παροχή υπηρεσιών, τις οποίες προσφέρει το ανθρώπινο δυναμικό. Ως εκ τούτου η αύξηση του τουριστικού προϊόντος συνεπάγεται με την αύξηση των θέσεων εργασίας στον τουρισμό. Τα τελευταία χρόνια, η οικονομική κρίση και η αύξηση της ανεργίας οδηγούν όλο και περισσότερους στην εποχιακή εργασία σε κάποιο τουριστικό μέρος. Παρότι είναι δύσκολο να υπολογιστεί η απασχόληση στον τουριστικό κλάδο, καθώς υπάρχουν πολλά επαγγέλματα που επηρεάζονται έμμεσα από αυτόν, ο τουρισμός είναι ο τρίτος σε σειρά μεγαλύτερος ¨εργοδότης¨ της χώρας μετά το εμπόριο και τον πρωτογενή τομέα, ενώ σύμφωνα με τα στοιχεία της ΕΛΣΤΑΤ, από το 1986 έως και το 2016 η απασχόληση στον τουρισμό έχει υπερδιπλασιαστεί.



Εικόνα 1: Έρευνα Εργατικού Δυναμικού (SETE)

Στο παραπάνω διάγραμμα παρατηρούμε ότι με εξαίρεση τις χρονιές 1991 και 2004 που υπήρξε μία μικρή πτώση, η απασχόληση στον τουρισμού αυξάνεται σταδιακά από το 1983 μέχρι και το 2017. Το 2018, οπού η συμβολή του τουρισμού στο ΑΕΠ φτάνει το 30,9% τα στοιχεία της ΕΛΣΤΑΤ μας δείχνουν ότι η συμμετοχή των απασχολουμένων στον τουρισμό επί του συνόλου των απασχολουμένων φτάνει στο 9,97%. Η αύξηση και η κατανομή της απασχόλησης στον τουρισμό τα τελευταία χρόνια φαίνεται στον ακόλουθο πίνακα

Πίνακας 2: Πλήθος απασχολούμενων 2016 έως 2018 (ΕΛΣΤΑΤ)

Υποτομείς της τουριστικής βιομηχανίας	**(σε χιλιάδες)**		
	2016	2017	2018
Απασχολούμενοι στα τουριστικά καταλύματα	79,6	87,3	95,4
Απασχολούμενοι σε υπηρεσίες εστίασης	261,5	263,2	266,2
Απασχολούμενοι σε ταξιδιωτικά πρακτορεία, σε άλλες υπηρεσίες κρατήσεων και σε συναφείς δραστηριότητες	20,9	20,1	14,1

Στον παραπάνω πίνακα παρατηρούμε ότι η απασχόληση τα τελευταία χρόνια αυξάνεται συνεχώς ενώ ο κλάδος του τουρισμού που παρέχει τις περισσότερες θέσεις εργασίας είναι η εστίαση φτάνοντας τον αριθμό ρεκόρ το 2018 με 266.200 θέσεις εργασίας. Ακολουθεί ο

κλάδος των τουριστικών καταλυμάτων με 95.400 θέσεις εργασίας και ο κλάδος των τουριστικών πρακτορείων με 14.100 θέσεις εργασίας. Στον πίνακα που ακολουθεί παρουσιάζεται η συμμετοχή του τουρισμού στην απασχόληση την περίοδο 2010 – 2018

Πίνακας 3: Τουρισμός και Απασχόληση 2010 έως 2018 (ΣΕΤΕ)

Έτος	Συνολική Συμμετοχή Τουρισμού στην Απασχόληση	Συνολική Απασχόληση
2010	17,8%	786.000
2011	17.6%	720.600
2012	18.3%	688.800
2013	18.2%	657.100
2014	17.3%	699.000
2015	23.1%	821.900
2016	23.4%	860.315
2017	24.8%	934.500
2018	25.9%	988.600

3. Ψυχαγωγία στις ξενοδοχειακές μονάδες (animation)

Στο κεφάλαιο αυτό γίνεται μία αναφορά για τους τρόπους διασκέδασης και ψυχαγωγίας των τουριστών εντός των ξενοδοχειακών μονάδων. Συγκεκριμένα, στην ενότητα 3.1 δίνεται ο ορισμός του animation ενώ στην ενότητα 3.2 τονίζεται η αναγκαιότητα της ύπαρξης του animation τόσο από την πλευρά των ξενοδοχειακών μονάδων, όσο και από τη πλευρά των τουριστών. Τέλος, στην ενότητα 3.3 παρουσιάζονται τα διάφορα είδη animation που εφαρμόζοντας στις ξενοδοχειακές μονάδες.

3.1 Ορισμός

Ο όρος animation προέρχεται από την ελληνική λέξη ¨άνεμος¨, με την έννοια του ψυχικού ανέμου, ο οποίος ενθαρρύνει και παροτρύνει σε δράση και ζωή. Από τον παραπάνω όρο προκύπτουν και οι λατινικές λέξεις animus (πνοή ζωής) και anima (ψυχή), από τις οποίες προέρχεται το ρήμα ¨animare¨ που σημαίνει εμψυχώνω. Αν και ο όρος animation έχει συνδεθεί με την κίνηση εικόνων που συναντάμε σε παιχνίδια και κινούμενα σχέδια, στο τουριστικό τομέα ο όρος animation σημαίνει ¨άνεμος συναισθημάτων στις διακοπές¨. Στην ελληνική γλώσσα ως animation μεταφράζεται η ψυχαγωγία, το οποίο όμως περιορίζει την πραγματική διάσταση, καθώς σχετίζεται και με τις έννοιες της άθλησης, της ξεκούρασης, της διασκέδασης και της ενθάρρυνσης.

3.2 Αναγκαιότητα του animation

Τα τελευταία χρόνια, όλο και περισσότεροι τουρίστες εκτός από τις παραδοσιακές τουριστικές υπηρεσίες όπως η διαμονή, το ταξίδι, η περιποίηση αλλά και τις εμπειρίες που προσφέρει ο κάθε προορισμός όπως θάλασσα, ήλιος, αρχαίος πολιτισμός κλπ, έχουν ανάγκη και από μία ακόμα υπηρεσία, την “διαμόρφωση διακοπών”, το οποίο επιτελεί το animation. Αυτός είναι και ο βασικός λόγος που όλο και περισσότερα ξενοδοχεία αναβαθμίζουν συνεχώς τις υπηρεσίες ψυχαγωγίας που παρέχουν στους πελάτες τους κατά τη διάρκεια των διακοπών τους. Η σημασία του animation περνάει σε άλλη διάσταση για περιπτώσεις ξενοδοχείων όπου στηρίζονται αποκλειστικά σε all inclusive πακέτα. Η συγκεκριμένη μορφή τουρισμού, είναι ιδιαίτερα διαδεδομένη στον ελληνικό τουρισμό καθώς μεγάλος αριθμός ξένων επισκεπτών, επιλέγει ξενοδοχεία αυτού του τύπου για να περάσει τις διακοπές του. Η ιδιαιτερότητα αυτής της μορφής τουρισμού είναι ότι οι τουρίστες αγοράζουν ολόκληρο το πακέτων των διακοπών τους, το οποίο περιλαμβάνει τόσο το ταξίδι και τη διαμονή, όσο και οτιδήποτε καταναλωθεί εντός του ξενοδοχείου. Αυτό οδηγεί πολλούς τουρίστες, να περνάνε τον περισσότερο χρόνο τους στο ξενοδοχείο, κάνοντας ακόμα πιο σημαντική τη χρήση του animation.

Η ψυχαγωγία σε πολλές περιπτώσεις έχει ευεργετικές επιπτώσεις για τα ίδια τα ξενοδοχεία καθώς αυξάνει τις πωλήσεις. Η θετική επίδραση που ασκεί στους τουρίστες το animation, ανεξάρτητα από την κοινωνική ή οικονομική τάξη στην οποία ανήκουν, τους οδηγεί σε κατανάλωση περισσότερων ειδών εντός του ξενοδοχείου συμβάλλοντας σημαντικά στην αύξηση του τζίρου της επιχείρησης. Ταυτόχρονα, η εικόνα ενός ξενοδοχείου που περιέχει οργανωμένες εγκαταστάσεις ψυχαγωγίας για τους πελάτες του, αναβαθμίζεται τόσο απέναντι στους πελάτες όσο και απέναντι στα τουριστικά γραφεία. Για τον λόγο αυτό, τα τελευταία χρόνια οι ξενοδοχειακές μονάδες, ανεξάρτητα από το μέγεθός τους, προσπαθούν να

αναπτύξουν τόσο τις υπηρεσίες που προσφέρουν στον τομέα της άθλησης και της ψυχαγωγίας όσο και τις εγκαταστάσεις στις οποίες αυτές λαμβάνουν χώρα. Έτσι, κάθε χρόνο τα ξενοδοχεία προσλαμβάνουν εξειδικευμένο προσωπικό το οποίο θα αναλάβει το κομμάτι του animation. Έχει παρατηρηθεί ότι σε περιπτώσεις όπου η ψυχαγωγία των πελατών μιας ξενοδοχειακής μονάδας ήταν ικανοποιητική, το ξενοδοχείο αποκτούσε πλεονεκτική θέση έναντι των ανταγωνιστών του. Τέλος, τα σχόλια που αφήνουν για το ξενοδοχείο οι πελάτες σε διάφορα ηλεκτρονικά μέσα και μέσα κοινωνικής δικτύωσης, λαμβάνονται υπόψη από νέους επίδοξους ταξιδιώτες δίνοντας ακόμα μεγαλύτερο πλεονέκτημα σε ξενοδοχεία που επενδύουν στο animation και στην ψυχαγωγία.

Από την άλλη πλευρά, το animation παίζει σημαντικό ρόλο και για την ψυχολογία του πελάτη, βελτιώνοντας σημαντικά την εμπειρία των διακοπών του. Ο σύγχρονος τρόπος ζωής, η έντονη επαγγελματική δραστηριότητα, η αύξηση του άγχους των ανθρώπων στις μεγαλουπόλεις δημιουργούν ολοένα και αυξανόμενη ανάγκη στον άνθρωπο για ψυχαγωγία. Έτσι, πολλοί αποφασίζουν να περάσουν τις διακοπές τους σε κάποιον προορισμό προκειμένου να ξεκουραστούν σωματικά και πνευματικά και να αποκτήσουν νέες εμπειρίες. Η ύπαρξη δραστηριοτήτων από τα ξενοδοχεία είναι προς αυτή την κατεύθυνση και σκοπό έχει να χαλαρώσει και να ξεκουράσει τους τουρίστες κάνοντάς τους να ξεχάσουν, έστω και προσωρινά, τα προβλήματα και τις έγνοιες της καθημερινότητάς τους. Μέσω του animation, ο τουρίστας έχει τη δυνατότητα να αναπτύξει την κοινωνικότητά του, καθώς έρχεται σε επαφή με άλλους ανθρώπου που έχουν την ίδια ανάγκη για ψυχαγωγία και χαλάρωση μ΄ αυτόν, την κινητικότητά του μέσα από αθλητικές δραστηριότητες, τη δημιουργικότητά του, ενώ ταυτόχρονα ζει την περιπέτεια αλλά και την ηρεμία που τον βοηθάνε να καταπολεμήσει το άγχος.

Εν κατακλείδι, το ρόλος του animation είναι υψίστης σημασίας για μία ξενοδοχειακή μονάδα, ενώ για πολλούς αποτελεί και το σημαντικότερο στοιχείο του τουριστικού προϊόντος, διαμορφώνοντας πολλές φορές τη συνολική ποιοτική του αξία. Οργανωμένες δράσεις μεταξύ των τουριστικών επιχειρήσεων σε τοπικό επίπεδο με στόχο την ενημέρωση και αναβάθμιση των υπηρεσιών τους θα έχει σημαντικά οφέλη για την περιοχή, καθώς ο τουρίστας που επιστρέφει ευχαριστημένος και πλήρης από τις διακοπές του, θα αποτελέσει σημαντική διαφήμιση γι’ αυτούς.

3.3 Είδη animation

Το animation στις ξενοδοχειακές μονάδες καλύπτει σε πολλές περιπτώσεις μία μεγάλη γκάμα δραστηριοτήτων, κάθε μία από τις οποίες μπορεί να περιλαμβάνει μία ή και περισσότερες από τις ακόλουθες κατηγορίες όπως κίνηση, δημιουργικότητα, εκπαίδευση, κοινωνικότητα, περιπέτεια, αυτοσυγκέντρωση. Το animation με βάση τη φύση του και τις δεξιότητες που απαιτεί μπορεί να χωριστεί σε δύο μεγάλες κατηγορίες, τις ψυχαγωγικές δραστηριότητες και τις αθλητικές δραστηριότητες.

3.3.1 Ψυχαγωγικές δραστηριότητες

Στις ψυχαγωγικές δραστηριότητες οι συμμετέχοντες περνούν τον χρόνο τους αναπτύσσοντας τη δημιουργικότητά τους, μαθαίνοντας νέα πράγματα ενώ έρχονται σε επαφή με άλλους ανθρώπους διαφορετικής κουλτούρας και ιδιοσυγκρασίας. Οι δραστηριότητες αυτές μπορεί να περιέχουν στοιχεία περιπέτειας εκπαίδευσης αθλητισμού ή αυτοσυγκέντρωσης και έχουν ως στόχο τόσο την χαλάρωση των συμμετεχόντων όσο και την επιμόρφωσή τους. Οι δραστηριότητες αυτές ανάλογα με τα χαρακτηριστικά μπορεί να ανήκουν σε μία ή και περισσότερες από τις ακόλουθες κατηγορίες:

- Κοινωνικές δραστηριότητες

Οι δραστηριότητες αυτές προάγουν την γνωριμία μεταξύ των τουριστών, διευκολύνοντας τις μεταξύ τους μελλοντικές επαφές καλλιεργώντας ένα κλίμα γιορτής. Κύριοι εκπρόσωποι αυτού του τύπου δραστηριοτήτων, είναι η οργάνωση εκδηλώσεων αποδοχής ή αποχαιρετισμού πελατών, εορτασμοί επετείων, γενεθλίων, χοροί, πάρτι, παρακολούθηση αθλητικών οργανώσεων κλπ.

- Δημιουργικές δραστηριότητες

Οι δραστηριότητες αυτές, έχουν ως στόχο να αναπτύξουν τη φαντασία των συμμετεχόντων γεμίζοντας δημιουργικά τον ελεύθερο χρόνο των διακοπών τους. Τέτοιες δραστηριότητες μπορεί να είναι η ζωγραφική, η φωτογραφία, η κεραμική, διάφορες κατασκευές αλλά και συμμετοχή σε θεατρικές παραστάσεις.

- Εκπαιδευτικές δραστηριότητες

Οι εκπαιδευτικές δραστηριότητες στοχεύουν στην απόκτηση νέων γνώσεων των τουριστών σχετικά με τον πολιτισμό, του τρόπου ζωής των ντόπιων καθώς και γνωριμία με τα ήθη και έθιμα της περιοχής. Η δραστηριότητες αυτές επιτυγχάνονται με επισκέψεις σε μουσεία, ιστορικά μνημεία και χώρους ή ακόμα και με παρακολούθηση διαλέξεων.

- Δραστηριότητες περιπέτειας

Δραστηριότητες αυτής της κατηγορίας μπορεί να είναι διανυκτέρευση στο ύπαιθρο, σαφάρι στα βουνά, ορειβασία, καταδύσεις κλπ. Στόχος τους είναι κάνουν τα βιώματα των συμμετεχόντων ακόμα πιο έντονα γνωρίζοντας παράλληλα τις φυσικές ομορφιές της περιοχής.

- Δραστηριότητας αυτοσυγκέντρωσης

Εκτός από την περιπέτεια, την εκπαίδευση και την κοινωνικοποίηση οι τουρίστες έχουν την ανάγκη για χαλάρωση και αυτοσυγκέντρωση. Έτσι, οι animateurs διοργανώνουν τέτοιες δραστηριότητες οι οποίες περιλαμβάνουν βραδιές κλασσικής μουσικής, ποίησης κλπ.

3.3.2 Αθλητικές δραστηριότητες

Πέρα από τις ψυχαγωγικές δραστηριότητες, πολλοί τουρίστες προτιμούν να ασχοληθούν με αθλητικές δραστηριότητες, είτε επειδή θέλουν να ακολουθήσουν την καθημερινή τους αθλητική ρουτίνα και στην περίοδο των διακοπών, είτε γιατί βρίσκουν την ευκαιρία να

έρθουν σε επαφή με νέα αθλήματα και προκλήσεις. Ο αθλητισμός προσφέρει υγεία, ευεξία, προσωπική ικανοποίηση ενώ αυξάνει και την αυτοπεποίθηση των συμμετεχόντων. Τα τελευταία χρόνια, όλο και περισσότερες ξενοδοχειακές μονάδες προσπαθούν να δημιουργήσουν και να βελτιώσουν τις αθλητικές τους εγκαταστάσεις, γνωρίζοντας ότι με αυτόν τον τρόπο θα προσελκύσουν περισσότερους τουρίστες. Έτσι, στα περισσότερα ξενοδοχεία πλέον ο τουρίστας θα βρει γυμναστήριο, spa, γήπεδα ποδοσφαίρου, μπάσκετ, τένις κλπ. Επίσης, πολλά ξενοδοχεία επενδύουν και σε θαλάσσια αθλήματα όπως θαλάσσιο σκι, ιστιοσανίδα, καταδύσεις, κωπηλασία και άλλα.

4. Ηλεκτρονικά Παιχνίδια

Με τον όρο ηλεκτρονικό παιχνίδι αναφερόμαστε σε οποιοδήποτε παιχνίδι παίζεται με τη χρήση κατάλληλης ηλεκτρονικής συσκευής. Η συσκευή αυτή μπορεί να είναι ένας ηλεκτρονικός υπολογιστής, μία κονσόλα παιχνιδιών ή ακόμα και ένα κινητό τηλέφωνο μέσω της ο οποίας ο χρήστης αλληλοεπιδρά με το παιχνίδι. Στο κεφάλαιο αυτό και συγκεκριμένα στην ενότητα 4.1 θα γίνει μια ιστορική αναδρομή στα ηλεκτρονικά παιχνίδια ενώ στην ενότητα 4.2 θα γίνει μία αναφορά στις κατηγορίες των ηλεκτρονικών παιχνιδιών. Τέλος, στην ενότητα 4.3 θα παρουσιαστούν τα εργαλεία που χρησιμοποιούνται για την ανάπτυξη ηλεκτρονικών παιχνιδιών.

4.1 Ιστορική αναδρομή

Η ιστορία των ηλεκτρονικών παιχνιδιών ξεκινάει στα τέλη της δεκαετίας του 40 όταν ο Thomas T. Goldsmith Jr. και Estle Ray Mann δημιούργησαν το πρώτο ηλεκτρονικό παιχνίδι. Πρόκειται για έναν εξομοιωτή πυραύλων ο οποίος εμφανίζονταν με μία οθόνη CRT και ο χρήστης μπορούσε να την κατευθύνει χρησιμοποιώντας ένα αναλογικό κύκλωμα. Τρία χρόνια αργότερα, το 1950 δημιουργείται από τον Σάρλι Αντάμα ένα παιχνίδι με τίτλο Bouncing Ball. Μέσα στην ίδια δεκαετία δημιουργήθηκε από τον Αλεξάντερ Σάφτο Ντάγκλας ένα παιχνίδι, για τον Αυτόματο Υπολογιστή του John von Neumann, με το όνομα OXO με θέμα την τρίλιζα αλλά με τα γράμματα Ο ή Χ στο οποίο αντίπαλος του παίκτη ήταν ο ίδιος ο υπολογιστής. Στα τέλη της δεκαετίας τους 1950 ο William Higinbotham δημιούργησε ένα διαδραστικό παιχνίδι με το όνομα “Tennis for two” το οποίο αφορούσε το άθλημα του τένις. Το παιχνίδι παίζονταν με δύο παίκτες, κάθε ένας από τους οποίους έχει ένα χειριστήριο και πατώντας τα πλήκτρα προσπαθούσε να ρίξει το μπαλάκι στην πλευρά του αντιπάλου του..

Το 1961 δημιουργείται από τον φοιτητή Steve Russell, Αμερικανό φοιτητή του M.I.T, το παιχνίδι Spacewar. Πρόκειται για ένα παιχνίδι με πρωταγωνιστή ένα διαστημικό πύραυλο ο οποίος εξουδετερώνει ότι εμφανίζεται μπροστά του. Το παιχνίδι παιζόταν στον υπολογιστή DEC PDP-1 και χρησιμοποιούσε για οθόνη έναν παλμογράφο. Το παιχνίδι αυτό είναι σημαντικό κομμάτι της ιστορίας των ηλεκτρονικών παιχνιδιών καθώς έχει σαφώς καλύτερα γραφικά ενώ έχει και καλύτερο εξομοιωτή σε σχέση με τα προηγούμενα παιχνίδια. Λίγα χρόνια αργότερα, το 1966 ένας άλλον Αμερικανός φοιτητής, ο Ralph Baer, έχει την ιδέα του in-house game play, παιχνίδια δηλαδή που μπορούν οι χρήστες να τα παίζουν στο σπίτι τους με τη χρήση κονσόλας και τηλεόρασης. Έτσι, έναν χρόνο αργότερα κατασκευάζει το Brown Box, την πρώτη οικιακή παιχνιδομηχανή.

Το 1970 εμφανίζεται το παιχνίδι Pong από την εταιρία βιοντεοπαιχνιδιών Atari των Nolan Bushnell και Al Alcorn. Το Pong, το οποίο όπως και το Tennis for two βασίζονταν στο άθλημα του τένις, γνώρισε ιδιαίτερη επιτυχία. Πρόκειται για το πρώτο παιχνίδι το οποίο ελέγχεται εξολοκλήρου από τον υπολογιστή εγκαινιάζοντας την εποχή των Computer Control Games ενώ από πλευράς υλικού χαρακτηρίζεται από το γεγονός ότι όλα τα μέρη του, εκτός από τα joysticks, είναι ηλεκτρονικά. Δύο χρόνια αργότερα το 1972, η ATARI δοκιμάζει για πρώτη φορά παιχνίδια με τη μορφή των Arcades, δηλαδή με μηχανή παιχνιδιών με κερματοδέκτη στην οποία ο χρήστης εισήγατε το κέρμα και έπαιζε ένα συγκεκριμένο παιχνίδι. Το εγχείρημά τους είχε τεράστια αποδοχή από το κοινό εγκαινιάζοντας την εποχή

των παιχνιδιών Arcades. Το 1977, η ίδια εταιρία παρουσιάζει την πρώτη ολοκληρωμένη κονσόλα με την ονομασία Atari 2600, η οποία μεταξύ άλλων προσφέρει παιχνίδια σε έγχρωμη μορφή και χειριστήρια για τους παίκτες. Τέλος, το 1978 δημιουργήθηκε από την γιαπωνέζικη εταιρία Taito το Space Invaders, ένα από τα δημοφιλή παιχνίδια όλων των εποχών.

Η εξέλιξη της τεχνολογίας και η αποδοχή από το ευρύ κοινό των πρώτων ηλεκτρονικών παιχνιδιών, οδήγησε σε μεγαλύτερη ανάπτυξη του κλάδου την δεκαετία του 80. Έτσι, το 1986 δημιουργείται από την εταιρία The Learning Company, το πρώτο εκπαιδευτικό παιχνίδια για παιδιά και ενήλικες με το όνομα Reader Rabbit, εισάγοντας με αυτό τον τρόπο την μάθηση μέσα από ηλεκτρονικά παιχνίδια. Την ίδια χρονιά κυκλοφόρησε το παιχνίδι Legend of Zelda από την εταιρία Shigeru Miyamoto, το πρώτο παιχνίδι ρόλων (role playing game), όπου ο παίκτης μπορούσε να διαλέξει έναν χαρακτήρα και να τον εξελίξει κατά τη διάρκεια του παιχνιδιού. Η επιτυχία των παιχνιδιών οδήγησε την εταιρία Nintendo, το 1989 να κάνει ένα βήμα παραπάνω κατασκευάζοντας την πρώτη κονσόλα χεριού με την ονομασία Game Boy.

Την δεκαετία του 90, οι παιχνιδομηχανές περνούν στην επόμενη γενιά, μεγαλώνοντας ακόμα περισσότερο την βιομηχανία παιχνιδιών. Οι τρεις μεγάλες εταιρίες της εποχής Sony, SEGA και Nintendo λανσάρουν τις νέες τους κονσόλες στην αγορά προκαλώντας ντελίριο στους πελάτες τους. Την αρχή έκανε η εταιρία SEGA με το Sega Saturn to 1994 και ακολούθησε η Sony με το πρώτο PlayStation, ενώ δύο χρόνια αργότερα η Nintendo παρουσίασε το Nintendo 64.

Η έλευση της νέας χιλιετίας σήμανε και την γιγάντωση της βιομηχανίας των παιχνιδιών. Αναβαθμισμένες κονσόλες όπως PlayStation 2, Xbox και Nintendo, νέα παιχνίδια με εξελιγμένα γραφικά καθώς και η εισαγωγή των Η/Υ σε όλο και περισσότερα σπίτια συντέλεσαν σ΄ αυτή τη γιγάντωση. Ταυτόχρονα η ανάπτυξη του διαδικτύου, οδήγησε σε δημιουργία παιχνιδιών multi-player, δηλαδή σε συνεύρεση πολλών παικτών στο διαδίκτυο ώστε να παίξουν το ίδιο παιχνίδι ξεφεύγοντας από το παραδοσιακό single- player παιχνίδι, στο οποίο ένας παίκτης παίζει το παιχνίδι μόνος του τοπικά στην κονσόλα ή στον υπολογιστή του. Όλα αυτά οδήγησαν σε εξελιγμένα παιχνίδια, με εντυπωσιακά γραφικά και δυνατότητες τα οποία σε πολλές περιπτώσεις εφαρμόζουν και τεχνητή νοημοσύνη.

4.2 Κατηγορίες ηλεκτρονικών παιχνιδιών

Η ανάπτυξη της τεχνολογίας και των τηλεπικοινωνιών έχει δώσει τεράστια ώθηση στην βιομηχανία παιχνιδιών, καθώς τα παιχνίδια που αναπτύσσονται ενσωματώνουν εφαρμογές της τεχνητής νοημοσύνης ενώ δίνει τη δυνατότητα σε πολλούς παίκτες να παίζουν το ίδιο παιχνίδι μέσω του ίντερνετ. Κάποιες από τις πιο βασικές κατηγορίες ηλεκτρονικών παιχνιδιών είναι οι ακόλουθες.

- **Παιχνίδια πρώτου προσώπου (First Person Shooters)**

Τα παιχνίδια αυτά, όπως φανερώνει και η ονομασία τους, είναι οπτικής πρώτου προσώπου. Αυτό σημαίνει ότι ο παίκτης δεν βλέπει ολόκληρο τον χαρακτήρα του αλλά μόνο το χέρι του δίνοντάς του την ψευδαίσθηση ότι είναι και ο ίδιος μέρος του παιχνιδιού. Το Shooters σημαίνει ότι ο χαρακτήρας κρατάει κάποιο όπλο με το οποίο προσπαθεί να εξολοθρεύσει τους εχθρούς του. Τα παιχνίδια αυτά χρησιμοποιούν 3D τεχνολογία για να δίνει στον παίκτη την ψευδαίσθηση του πραγματικού χώρου ενώ παίζονται συνήθως σε Η/Υ. Το συγκεκριμένο είδος παιχνιδιού είναι ιδιαίτερα δημοφιλές καθώς επιτρέπει πολλούς παίκτες να παίζουν στο ίδιο σενάριο ενώ ταυτόχρονα μπορούν να μιλάνε και μεταξύ τους με τη χρήση μικροφώνου κάνοντάς το ακόμα πιο ρεαλιστικό. Τα πιο χαρακτηριστικά παιχνίδια της συγκεκριμένης κατηγορίας είναι το Counter – Strike, το Call of Duty και το Halo.

- **Παιχνίδια περιπέτειας (Adventure Games)**

Τα παιχνίδια αυτής της κατηγορίας ακολουθούν μία ιστορία στην οποία ο παίκτης είναι ο πρωταγωνιστής και έχει ως στόχο την επίλυση διάφορων γρίφων. Τα πρώτα παιχνίδια είχαν ως βάση το κείμενο, ενώ στη συνέχει άρχισε ολοένα να εισάγεται και το γραφικό περιβάλλον. Στη σημερινή τους μορφή, τα παιχνίδια αυτά βασίζονται στο ¨point-n-click¨ κατά το οποίο ο χρήστης κάνει click με το ποντίκι του Η/Υ προκειμένου να πυροδοτήσει μία ενέργεια στον χαρακτήρα του, ενώ το σενάριο τους είναι βασισμένο στην επιστημονική φαντασία. Το πρώτο ηλεκτρονικό παιχνίδι της συγκεκριμένης κατηγορίας αναπτύχθηκε το 1970 με τον τίτλο “Colossal Cave Adventures”.

- **Αθλητικά παιχνίδια (Sport Game)**

Τα παιχνίδια αυτής κατηγορίας σχετίζονται με τον αθλητισμό και τα σπορ, όπως είναι το ποδόσφαιρο, το μπάσκετ, το τένις και ο μηχανοκίνητος αθλητισμός. Τα περισσότερα από τα παιχνίδια αυτής της κατηγορίας βγάζουν κάθε χρόνο νέες εκδώσεις προκειμένου να συμβαδίζουν με τις κανονικούς αθλητικούς ομίλους. Έτσι, οι εταιρίες που δραστηριοποιούνται σ’ αυτό το είδος παιχνιδιών, ξοδεύουν τεράστια ποσά κάθε χρόνο προκειμένου να εξασφαλίσουν τα δικαιώματα των ομάδων και των παικτών ώστε να μπορούν να τους ενσωματώσουν στα παιχνίδια τους, κάνοντας την εμπειρία του χρήστη ακόμα πιο ρεαλιστική. Τα παιχνίδια αυτά δίνουν στους χρήστες τη δυνατότητα να παίξουν είτε μόνοι τους είτε με φίλους τους είτε ακόμα να παίξουν και διαδικτυακά. Χαρακτηριστικά παιχνίδια της συγκεκριμένης κατηγορίας είναι το FIFA, το NBA και η Formula 1.

- **Παιχνίδια στρατηγικής (Real Time Strategy)**

Στα παιχνίδια στρατηγικής, όπως φανερώνει ο τίτλος τους ο παίκτης εφαρμόζει μία στρατηγική και επιμέρους τακτικές προκειμένου να φτάσει στον στόχο του. Στα παιχνίδια αυτής της κατηγορίας σημαντικό ρόλο παίζουν η σωστή σκέψη, η ταχύτητα και η οξυδέρκεια του παίκτη. Το πιο κοινό σενάριο αυτού του είδους παιχνιδιών είναι η δημιουργία, ανάπτυξη και επέκταση αυτοκρατοριών μεταξύ των παικτών με τελικό στόχο την επικράτηση τους έναντι των υπολοίπων. Τα παιχνίδια αυτά παίζονται συνήθως σε γύρους και κάθε γύρος τελειώνει με την τελική μάχη μεταξύ των παικτών. Χαρακτηριστικά παιχνίδια στρατηγικής είναι Age of mythology και το Civilization.

- **Παιχνίδια προσομοίωσης (Simulations Games)**

Ένα παιχνίδι προσομοίωσης έχει ως στόχο την αναπαραγωγή διάφορων δραστηριοτήτων του πραγματικού κόσμου σε μορφή παιχνιδιού. Το πιο χαρακτηριστικό παιχνίδι της συγκεκριμένης κατηγορίας είναι το Flight Simulator κατά το οποίο ο παίκτης μπαίνει στην θέση του πιλότου και πιλοτάρει ένα αεροσκάφος.

- **Παιχνίδια υπόδησης ρόλων (Massive multiplayer online role playing game)**

Βασικό χαρακτηριστικό αυτής της κατηγορίας παιχνιδιών είναι ο μεγάλος αριθμός ταυτόχρονων παικτών που συμμετέχουν σε έναν μεμονωμένο παιχνίδι. Κάθε παίκτης στην αρχή του παιχνιδιού διαλέγει και από έναν χαρακτήρα τον όποιο προσπαθεί να τον εξελίξει κατά τη διάρκεια του παιχνιδιού, προσθέτοντας επιπλέον χαρακτηριστικά και δυνατότητες σ΄ αυτόν. Η μεγάλη απήχηση αυτών των παιχνιδιών είναι ότι μπορούν να καλύψουν διαφορετικές κατηγορίες παικτών. Έτσι κάποιοι από τους πιο φανατικούς παίκτες (hardcore gamers) σε πολλές περιπτώσεις είναι διατεθειμένοι να πληρώσουν, προκειμένου να εξελίξουν τον χαρακτήρα τους ή ακόμα και να αγοράσουν έτοιμο χαρακτήρα από κάποιον άλλον παίκτη. Χαρακτηριστικά παιχνίδια της συγκεκριμένης κατηγορίας είναι το DOTA, το Need for Speed World και το Star Wars Galaxies.

- **Παιχνίδια λογικής**

Τα παιχνίδια αυτά έχουν ως στόχο να κάνουν τον παίκτη να σκεφτεί και να χρησιμοποιήσει την φαντασία του. Ο παίκτης, καλείται να επιλύσει λογικά πάζλ, να περιηγηθεί σε λαβύρινθους και να λύσει διάφορα λογικά προβλήματα. Τα παιχνίδια αυτά, τις περισσότερες φορές συνδυάζονται και με τα παιχνίδια περιπέτειας. Χαρακτηριστικός εκπρόσωπος της συγκεκριμένης κατηγορίας είναι το Tetris.

4.3 Μηχανές ανάπτυξης παιχνιδιού (Game engines)

Ιστορικά η ανάπτυξη ηλεκτρονικών παιχνιδιών γινόταν από τους προγραμματιστές παιχνιδιών, οι οποίοι έκαναν τα πάντα, από την συγγραφή του κώδικα μέχρι τη σχεδίαση των γραφικών και των εικόνων, τον χειρισμό του παιχνιδιού, την ηχοληψία το animation και οτιδήποτε άλλο απαιτούνταν. Παρ΄ όλα αυτά, όσο η βιομηχανία ηλεκτρονικών παιχνιδιών μεγάλωνε έγινε διαχωρισμός των ρόλων και έτσι στην δημιουργία ενός παιχνιδιού εκτός από τους προγραμματιστές συμμετέχουν επιπλέον Computer Graphics (CG) animators, παραγωγοί μουσικής, σεναριογράφοι, ηθοποιοί κοκ.

Προκειμένου να γίνει η σύνθεση των διαφορετικών αυτών κομματιών, χρειάζεται ειδικό λογισμικό γνωστό και ως μηχανή ανάπτυξης παιχνιδιού. Μία μηχανή ανάπτυξης παιχνιδιού (game engine) δεν είναι τίποτα άλλο από ένα εξειδικευμένο λογισμικό που χρησιμοποιείται για την ανάπτυξη ενός ηλεκτρονικού παιχνιδιού. Η βασική λειτουργικότητα που παρέχει μία μηχανή ανάπτυξης παιχνιδιού περιλαμβάνει το renderer για δισδιάστατα (2D) ή τρισδιάστατα (3D) γραφικά, έναν μηχανισμό εντοπισμού συγκρούσεων μεταξύ των αντικειμένων του παιχνιδιού, το animation, το γραφικό σκηνικό (scene graph), τον ήχο, τη διαχείριση μνήμης, νήματα (threading) ενώ επιπλέον μπορεί να ενσωματώνει δικτύωση και τεχνητή νοημοσύνη.

Αυτά τα εργαλεία παρέχονται σε ένα ολοκληρωμένο περιβάλλον ανάπτυξης προκειμένου να διευκολυνθεί η ανάπτυξη ενός παιχνιδιού. Ένα άλλο πλεονέκτημα από τη χρήση των game engines είναι ότι μικρές αλλαγές στον πηγαίο κώδικα επιτρέπουν ένα παιχνίδι να μπορεί να εκτελεστεί σε διάφορες πλατφόρμες είτε αυτές είναι κονσόλες παιχνιδιών είτε Η/Υ.

Καθώς η τεχνολογία αναπτύσσεται, νέες συσκευές όπως τα κινητά τηλέφωνα (smartphones) είναι ικανές να υποστηρίξουν την αναπαραγωγή παιχνιδιών κάτι που οδηγεί της μηχανές παιχνιδιών να κινηθούν και προς αυτή την κατεύθυνση. Επιπλέον όσο αυτές εξελίσσονται και γίνονται πιο φιλικές προς χρήση τόσο αυξάνεται και η χρήσης τους σε παιχνίδια σοβαρού σκοπού στον τομέα της ιατρικής, της εκπαίδευσης καθώς και σε στρατιωτικές εφαρμογές. Τέλος, όλες σχεδόν οι μηχανές παιχνιδιών βασίζονται πάνω σε κάποια γλώσσα προγραμματισμού υψηλού επιπέδου, όπως για παράδειγμα η java, η C# και η Python αυξάνοντας κατά πολύ την παραγωγικότητα των δημιουργών. Κάποιες από τις πιο σημαντικές μηχανές ανάπτυξης παιχνιδιών είναι οι ακόλουθες:

- Unreal Engine

Η Unreal Engine είναι μία από πιο δημοφιλείς μηχανές αναζήτησης η οποία αναπτύχθηκε από την εταιρία ανάπτυξης παιχνιδιών Epic Games το 1998. Αν και αρχικά σχεδιάστηκε για παιχνίδια πρώτου προσώπου (First-person shooter), χρησιμοποιήθηκε με επιτυχία για την δημιουργία και άλλων τύπων παιχνιδιών όπως παιχνιδιών υπόδησης ρόλου (Massive multiplayer online role playing game) και παιχνιδιών στρατηγικής (Real time strategy). Η Unreal Engine έχει αναπτυχθεί με την γλώσσα προγραμματισμού C++ παρέχοντας υψηλό βαθμό φορητότητας και υποστηρίζοντας ένα μεγάλο εύρος από πλατφόρμες. Η πιο πρόσφατη έκδοση είναι η Unreal Engine 4 η οποία διατέθηκε για πρώτη φορά το 2014 ενώ από το 2015 έχει γίνει ανοιχτού κώδικα (open source), επιτρέποντας στον οποιονδήποτε να τη χρησιμοποιήσει δωρεάν. Τέλος αξίζει να σημειωθεί ότι τον Δεκέμβριο του ίδιου έτους η Unreal Engine βραβεύτηκε από την Guinness World Record ως η πιο επιτυχημένη μηχανή παιχνιδιού.

- Unity

Η Unity είναι επίσης μία πολύ διαδεδομένη game engine, η οποία αναπτύχθηκε από την εταιρία Unity Technologies και παρουσιάστηκε για πρώτη φορά τον Ιούνιο του 2015 στο παγκόσμιο συνέδριο προγραμματισμός της Apple ως η αποκλειστική game engine του Mac OS X. Μέχρι το 2018, η Unity είχε επεκταθεί τόσο πολύ που υποστήριζε περισσότερες από 25 πλατφόρμες παιχνιδιών, κάποιες από τις οποίες είναι Windows, Mac, iPhone, PlayStation και Xbox. Η συγκεκριμένη game engine χρησιμοποιείται για τη δημιουργία δισδιάστατων (2D) και τρισδιάστατων (3D) παιχνιδιών και παιχνιδιών εικονικής πραγματικότητας (virtual reality) ενώ η χρήση της είναι ευρεία και σε παιχνίδια προσομοίωσης. Η Unity δεν έχει υιοθετηθεί μόνο από εταιρίες ανάπτυξης παιχνιδιών αλλά από εταιρίες που δραστηριοποιούνται σε άλλους τομείς όπως στον κινηματογράφος, στη μηχανική στην αρχιτεκτονική και αλλού. Σε σχέση με την Unreal Engine, η Unity χρησιμοποιείται κυρίως από πολλούς προγραμματιστές για την ανάπτυξη παιχνιδιών σε κινητές συσκευές σε αντίθεση με την Unreal Engine, η οποία είναι πιο διαδεδομένη στην ανάπτυξη παιχνιδιών για κονσόλες

και Η/Υ. Κάποια από τα πιο γνωστά παιχνίδια τα οποία φτιάχτηκαν σε Unity είναι το Super Mario Run, το Angry Birds και το Pokémon GO.

- Godot

Η Godot είναι μία ανοιχτού κώδικα game engine και χρησιμοποιείται για τη δημιουργία 2D και 3D παιχνιδιών τα οποία μπορούν να εκτελεστούν σε διάφορες πλατφόρμες όπως Windows, Linux, macOS, Android κλπ. Πριν την μετατροπή της σε λογισμικό ανοιχτού κώδικα, η Godot χρησιμοποιήθηκε από διάφορες εταιρίες ανάπτυξης παιχνιδιών στην Λατινική Αμερική. Η συγκεκριμένη game engine επιτρέπει τους δημιουργούς παιχνιδιών να αναπτύξουν παιχνίδια από το μηδέν χρησιμοποιώντας την C++ ή την C#, ενώ και η ίδια διαθέτει μία δικιά της υψηλού επιπέδου γλώσσα προγραμματισμού παρόμοια με την Python.

- Amazon Lumberyard

Η Amazon Lumberyard είναι μία cross-platform game engine, η οποία διατίθεται δωρεάν και βασίζεται στην game engine CryEngine. Ένα από τα χαρακτηριστικά της είναι ότι ενσωματώνεται με τα Amazon Web Services επιτρέποντας έτσι τους δημιουργούν παιχνιδιών να αναπτύξουν ή να κάνουν host τα παιχνίδια τους στους servers της Amazon. Επίσης τους δίνεται η δυνατότητα για δημιουργία παιχνιδιών livestreaming μέσω της υπηρεσίας Twitch. Ο πηγαίος κώδικας της μηχανής είναι διαθέσιμος στους τελικούς χρήστες με τον περιορισμό να μην τον χρησιμοποιήσουν για την δημιουργία άλλων game engines. Κάποιοι από τους πιο διαδεδομένους τίτλους παιχνιδιών που δημιουργήθηκαν με τη συγκεκριμένη game engine είναι το Breakaway και το Star Citizen.

5. Η Python και η βιβλιοθήκη pygame

Στην ενότητα 4.3 είδαμε τις βασικές και πιο διαδεδομένες game engines οι οποίες χρησιμοποιούνται για τη δημιουργία cross-platform ηλεκτρονικών παιχνιδιών. Παρ΄ όλα αυτά, η χρήση μίας game engine για την δημιουργία ενός παιχνιδιού δεν είναι μονόδρομος. Μία άλλη λύση για τη δημιουργία ηλεκτρονικών παιχνιδιών είναι η γλώσσα προγραμματισμού Python και η βιβλιοθήκη pygame. Στο κεφάλαιο αυτό και συγκεκριμένα στην ενότητα 5.1 θα γίνει μία παρουσίαση της γλώσσα προγραμματισμού Python, ενώ στην ενότητα 5.2 θα παρουσιαστούν οι βασικές ιδιότητες και τα χαρακτηριστικά της βιβλιοθήκης pygame η οποία και θα χρησιμοποιηθεί στην παρούσα Πτυχιακή Εργασία για την ανάπτυξη του παιχνιδιού “Super Hercules”. Τέλος στην ενότητα 5.3 θα παρουσιαστεί η πιο απλή δομή παιχνιδιού με την pygame.

5.1 Η γλώσσα προγραμματισμού Python

Η Python δημιουργήθηκε από τον Ολλανδό μαθηματικό Guido van Rossum και κυκλοφόρησε για πρώτη φορά το 1991. Αρχικά, η Python χρησιμοποιήθηκε ως γλώσσα σεναρίων για το κατανεμημένο λειτουργικό σύστημα Amoeba. Το όνομά της προέρχεται από τη δημοφιλή κωμική σειρά Monty Python’s, που γνώρισε τεράστια επιτυχία τη δεκαετία του ‘70. Είναι μία γλώσσα προγραμματισμού υψηλού επιπέδου, διερμηνευόμενη κάτι που σημαίνει ότι σε αντίθεση με άλλες γλώσσες προγραμματισμού όπως η C ή η C++ εκτελείται εντολή προς εντολή και δεν μεταφράζεται ολόκληρη. Όπως αναφέρθηκε και παραπάνω, η πρώτη έκδοση της Python κυκλοφόρησε το 1991. Ακολούθησε η Python 2.0 η οποία κυκλοφόρησε στις 16 Οκτωβρίου 2000. Η τρέχουσα έκδοση της Python είναι η Python 3.x η οποία κυκλοφόρησε στις 3 Δεκεμβρίου του 2008 και χρησιμοποιείται μέχρι και σήμερα.

Ένα από τα κύρια χαρακτηριστικά της Python, είναι η απλότητα και η ευκολία χρήσης της, το οποίο επιτρέπει τον προγραμματιστή να αφιερώσει περισσότερο χρόνο στην επίλυση του προβλήματος παρά στις ιδιοτροπίες της γλώσσας. Το απλό συντακτικό της, επιτρέπει τους προγραμματιστές να αναπτύξουν τα προγράμματα τους σε λιγότερες γραμμές κώδικα και σε σημαντικά λιγότερο χρόνο σε σχέση με άλλες γλώσσες προγραμματισμού όπως η C και η java. Επίσης, υποστηρίζει τόσο τον διαδικαστικό (procedural programming) όσο και τον αντικειμενοστραφές προγραμματισμό (object oriented programming). Όλα αυτά οδήγησαν την Python στο να γίνει σταδιακά η πιο διαδεδομένη γλώσσα προγραμματισμού παγκοσμίως. Η μεγάλη αποδοχή της από την κοινότητα των προγραμματιστών είχε ως αποτέλεσμα να αναπτυχθεί μία πληθώρα βιβλιοθηκών (modules), τα οποία επιτρέπουν τους χρήστες να την χρησιμοποιήσουν σε διάφορες εφαρμογές όπως παραθυρικές εφαρμογές, εφαρμογές στον ιστό (web applications), εφαρμογές τεχνητής νοημοσύνης και μηχανικής μάθησης, ανάπτυξη παιχνιδιών κοκ.

Εκτός όμως από τα πολλά πλεονεκτήματα που προσφέρει στους προγραμματιστές, η Python έχει και κάποιες αδυναμίες που την καθιστούν ακατάλληλη για κάποιες εφαρμογές. Το βασικό μειονέκτημα της Python είναι ο χρόνος εκτέλεσης των προγραμμάτων, καθώς παρατηρείται μία καθυστέρηση στην εκτέλεση σε σχέση με άλλες γλώσσες προγραμματισμού όπως η C και η C++ που είναι μεταγλωττιζόμενες (compiled). Το γεγονός αυτό οφείλεται, στο ότι ένα πρόγραμμα Python δεν μεταγλωττίζεται σε δυαδικό κώδικα μηχανής που

εκτελείται άμεσα από τον επεξεργαστή, καθώς όπως αναφέρθηκε και παραπάνω, η Python είναι μία διερμηνευόμενη γλώσσα. Έτσι, η χρήση της Python δεν ενδείκνυται όταν ο χρόνος απόκρισης του αναπτυσσόμενου συστήματος είναι καθοριστικής σημασίας.

Όσον αφορά την ανάπτυξη ηλεκτρονικών παιχνιδιών η Python έχει πολλά να δώσει, καθώς διαθέτει μία πληθώρα από frameworks και βιβλιοθήκες (modules) που δίνουν τα κατάλληλα εργαλεία σε όποιον θέλει να αναπτύξει τα δικά του παιχνίδια. Κάποιες από τις σημαντικότερες βιβλιοθήκες της python είναι:

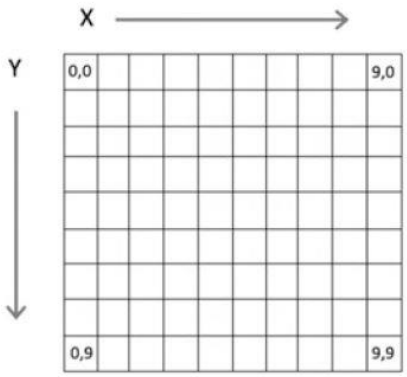
- Arcade. Χρησιμοποιείται για τη δημιουργία 2D παιχνιδιών
- pyglet. Είναι μία παραθυρική πολυμεσική βιβλιοθήκη της Python η οποία μπορεί να χρησιμοποιηθεί και την δημιουργία παιχνιδιών
- Cocos2d. Είναι ένα framework για τη δημιουργία 2D παιχνιδιών με τη χρήση και της βιβλιοθήκης pyglet.
- pygame. Είναι η πιο διαδεδομένη βιβλιοθήκη για ανάπτυξη παιχνιδιών με την Python καθώς επιτρέπει τον χρήστη να δημιουργήσει διαφόρων τύπων παιχνίδια.

5.2 Η βιβλιοθήκη pygame της Python

Η pygame είναι μία ανοιχτού κώδικά (Open Source), cross-platform βιβλιοθήκη της Python, η οποία χρησιμοποιείται για την ανάπτυξη ηλεκτρονικών παιχνιδιών. Η ανάπτυξή της ξεκίνησε το τον Οκτώβριο του 2000 και η πρώτη της έκδοση (pygame 1.0) βγήκε σε κυκλοφορία έξι μήνες αργότερα. Η βιβλιοθήκη pygame έχει κατασκευαστεί πάνω στην βιβλιοθήκη SDL (Simple Directmedia Layer). Η SDL είναι μία cross-platform βιβλιοθήκη η οποία παρέχει πρόσβαση σε ήχο, πληκτρολόγιο, ποντίκι, joystick και κάρτα γραφικών μέσω του OpenGL και του Direct3D. Η pygame ουσιαστικά "αγκαλιάζει" τη βιβλιοθήκη SDL παρέχοντας επιπλέον λειτουργικότητα, όπως διανύσματα, εντοπισμός συγκρούσεων αντικειμένων, 2D γραφικά, κάμερα, διαχείριση pixel κλπ. τα οποία κάνουν τη δημιουργία των παιχνιδιών ευκολότερη.

5.2.1 Επιφάνεια Απεικόνισης (Display Surface)

Η Display Surface είναι ίσως το πιο σημαντικό κομμάτι στην ανάπτυξη της εφαρμογής καθώς είναι το κύριο παράθυρο του παιχνιδιού, πάνω στο οποίο τοποθετούνται όλα τα αντικείμενα, τα οποία αλληλοεπιδρούν μεταξύ τους. Αποτελεί δηλαδή, ένα λευκό κομμάτι χαρτιού πάνω στο οποίο ο δημιουργός μπορεί να ζωγραφίσει τα γραφικά στοιχεία του παιχνιδιού. Η επιφάνεια αυτή, αποτελείται από εικονοστοιχεία (pixels), τα οποία αριθμούνται από το 0,0 το οποίο αντιστοιχεί στην πάνω αριστερή κορυφή της επιφάνειας μέχρι και την κάτω δεξιά κορυφή που αντιστοιχεί στις διαστάσεις που έχει δώσει ο δημιουργός στο παράθυρο. Η εικόνα 2 δείχνει τον τρόπο με τον οποίο αριθμούνται τα pixels του Surface. Χρησιμοποιώντας αυτές τις συντεταγμένες, ο δημιουργός μπορεί να τοποθετήσει εικόνες ή ακόμα και να ζωγραφίσει διάφορα αντικείμενα σε όποιο σημείο της επιφάνειας επιθυμεί.



Εικόνα 2: Display Surface μεγέθους 9x9 (Advanced Guide to Python 3 Programming)

Για τη δημιουργία του Surface χρησιμοποιείται η μέθοδος `pygame.display.set_mode()` η οποία παίρνει ως παράμετρο τις διαστάσεις της επιφάνειας. Για παράδειγμα η εντολή

```
display_surface = pygame.display.set_mode((800, 600))
```

δημιουργεί ένα παράθυρο 800 pixel πλάτους και 600 pixel ύψους ενώ αναφερόμαστε σε αυτό με την μεταβλητή `display_surface`. Από τη στιγμή που ορίστηκε το παράθυρο, ο δημιουργός μπορεί να βάλει χρώμα παρασκηνίου (background) με την εντολή

`display_surface.fill((255, 255, 255))` το οποίο γεμίζει το παράθυρο με το λευκό χρώμα. Σ΄ αυτό το σημείο θα πρέπει να τονιστεί, ότι τα χρώματα στην pygame ορίζονται με την κωδικοποίηση RGB (Red, Green, Blue) και κάθε πεδίο παίρνει τιμές από 0 έως και 255.

Όλα τα στοιχεία τα οποία έχουμε εισάγει στο παράθυρο (background χρώμα, σχήματα, εικόνες) δεν θα εμφανιστούν στην οθόνη αν πρώτα δεν εκτελεστούν οι μέθοδοι `update()` ή `flip()`. Η πρώτη μέθοδος καλείται ως `pygame.display.update()` και ¨ζωγραφίζει¨ εκ νέου όλα τα αντικείμενα στην οθόνη, ενώ παίρνει και μία προαιρετική παράμετρο ώστε να κάνει update ένα συγκεκριμένο αντικείμενο πάνω στην οθόνη. Αντίστοιχη λειτουργία επιτελεί και η μέθοδος `flip()`, η οποία καλείται ως `pygame.display.flip()`, με τη διαφορά όμως ότι κάνει update ολόκληρο το παράθυρο και ότι αυτό περιέχει.

Τέλος προκειμένου να δοθεί τίτλος στο συγκεκριμένο παράθυρό χρησιμοποιείται η μέθοδος `pygame.display.set_caption('Super Hercules')`, η οποία παίρνει ως παράμετρο ένα αλφαριθμητικό το οποίο αντιστοιχεί στον τίτλο του παραθύρου.

5.2.2 Συμβάντα (Events)

Προκειμένου ο χρήστης του παιχνιδιού να μπορέσει να αλληλεπιδράσει με αυτό, η pygame χρησιμοποιεί ένα event loop με το οποίο μπορεί και ¨συλλέγει¨ τις ενέργειες που κάνει ο χρήστης, χρησιμοποιώντας το ποντίκι, το πληκτρολόγιο ή οποιαδήποτε άλλη συσκευή που του επιτρέπει να αλληλεπιδράσει με το περιβάλλον του παιχνιδιού. Μια τέτοια ενέργεια για παράδειγμα, θα μπορούσε να είναι το πάτημα εντός πλήκτρου του πληκτρολογίου ή ακόμα

και η κίνηση του ποντικού μέσα στο παράθυρο του παιχνιδιού. Όλες αυτές οι ενέργειες του χρήστη αντιπροσωπεύονται ως ένα συμβάν (event).

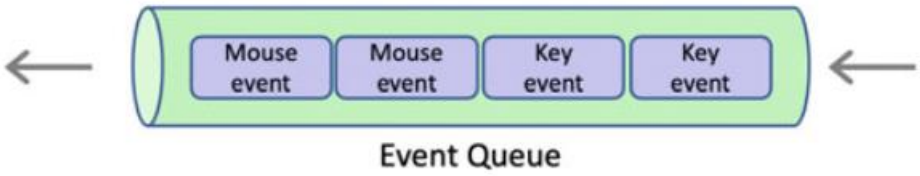
Κάθε event που δημιουργείται από τον χρήστη περιέχει πληροφορία που σχετίζεται με αυτόν, όπως για παράδειγμα ο τύπος του event. Για παράδειγμα, πιέζοντας ένα πλήκτρο από το πληκτρολόγιο θα έχει ως αποτέλεσμα την δημιουργία ενός event τύπου KEYDOWN ενώ όταν το πλήκτρο απελευθερωθεί θα δημιουργηθεί ένα event τύπου KEYUP. Ομοίως, χρησιμοποιώντας το ποντίκι ο χρήστης δημιουργεί events τύπου MOUSEMOTION καθώς επίσης και MOUSEBUTTONDOWN και MOUSEBUTTONUP αν πατήσει και στη συνέχεια απελευθερώσει ένα πλήκτρο του ποντικού. Τέλος, αν ο χρήστης επιλέξει να κλείσει το παράθυρό πατώντας το αντίστοιχο κουμπί πάνω δεξιά θα παραχθεί ένα event τύπου QUIT. Αυτός ο μηχανισμός επιτρέπει στον δημιουργό του παιχνιδιού να καθορίσει πια event τον ενδιαφέρουν και ποια όχι.

Κάθε τύπος event περιέχει πληροφορίες που σχετίζονται με αυτό. Για παράδειγμα, ένα event τύπου Key περιέχει πληροφορία για το πιο πλήκτρο πίεσε ή απελευθέρωσε ο χρήστης ενώ ένα event τύπου Mouse περιέχει πληροφορίες για τη θέση του ποντικού στην οθόνη κλπ. Έτσι, αν θέλουμε να ελέγξουμε αν ο χρήστης πίεσε το πλήκτρο space θα γράψουμε τον ακόλουθο κώδικα:

```
# Έλεγχος αν πατήθηκε κάποιο πλήκτρο του πληκτρολογίου
if event.type == pygame.KEYDOWN:
    # Έλεγχος αν το πλήκτρο που πατήθηκε ήταν το space
    if event.key == pygame.K_SPACE:
        print('space')
```

Κάθε πλήκτρο αναπαρίσταται από την pygame ως μία σταθερά η οποία ξεκινάει με το αλφαριθμητικό 'K_' ακολουθούμενο από το όνομα του πλήκτρου, για παράδειγμα η σταθερά K_TAB αντιστοιχεί στο πλήκτρο Tab και η K_LEFT αντιστοιχεί στο αριστερό βελάκι.

Τα events αποθηκεύονται σε μία ακολουθία event (Event Queue). Ένα Event Queue χρησιμοποιείται για να συλλέξει όλα τα events που συμβαίνουν κατά την εκτέλεση του παιχνιδιού. Για παράδειγμα, ας υποθέσουμε ότι ένας χρήστης σε κάποιο σημείο της εκτέλεσης του παιχνιδιού πατάει διπλό κλικ από το ποντίκι και ένα πλήκτρο από το πληκτρολόγιο δύο φορές, πριν όμως το πρόγραμμα προλάβει να επεξεργαστεί τα πρώτα events. Η εικόνα 3 δείχνει το περιεχόμενο της Event Queue



Εικόνα 3: Event Queue (Advanced Guide to Python 3 Programming)

Η εφαρμογή με αυτό τον τρόπο μπορεί να πάρει τα events από το Event Queue με τη σειρά με την οποία παρήχθησαν και να τα επεξεργαστεί, ενώ ταυτόχρονα εισέρχονται και άλλα events στην ουρά, καθώς το παιχνίδι συνεχίζει να εκτελείται. Ένα σημαντικό πλεονέκτημα αυτής της προσέγγισης, είναι ότι δεν χάνεται κανένα event καθώς όλα τα events του χρήστη καταγράφονται στην ουρά και εκτελούνται με τη σειρά. Η μέθοδος `pygame.event.get()` διαβάζει όλα τα event από την Event Queue, απομακρύνοντάς τα στη συνέχεια από την ουρά. Η μέθοδος επιστρέφει μία λίστα (List) με τα events τα οποία επεξεργάζονται με τη σειρά. Στον παρακάτω κώδικα για παράδειγμα, παίρνουμε όλα τα events από την Event Queue και τυπώνουμε το κατάλληλο μήνυμα ανάλογα με τον τύπο του event.

```
for event in pygame.event.get():
     if event.type == pygame.QUIT():
          print('Received Quit Event')
     elif event.type == pygame.MOUSEBUTTONDOWN:
          print('Received Mouse Event')
     elif event.type == pygame.KEYDOWN:
          print('Received Keydown Event')
```

5.2.3 Εικόνες (Images)

Η pygame περιέχει μεθόδους με τις οποίες μπορούμε να φορτώσουμε εικόνες στο παιχνίδι προκειμένου να το κάνουμε πιο ρεαλιστικό. Όταν εισάγεται μία εικόνα στην pygame, αντιμετωπίζεται ως αντικείμενο Surface. Αυτό δίνει μεγάλη ευελιξία στον δημιουργό του παιχνιδιού, καθώς μπορεί να την χειριστεί με διάφορους τρόπους. Οι εικόνες μπορεί να είναι τύπου JPEG, PNG, BMP κτλ. Μία εικόνα μπορεί να φορτωθεί ως ακολούθως

```
image_surface = pygame.image.load(filename).convert()
```

Με αυτό τον τρόπο εισάγουμε μία εικόνα στο παιχνίδι σε μορφή surface. Η μέθοδος convert() χρησιμοποιείται για να μετατρέψει το format των pixel σε αυτό που χρησιμοποιείται από το Surface. Σε αντίθετη περίπτωση, αυτό θα γίνεται κάθε φορά που η συγκεκριμένη εικόνα εμφανίζεται στο παράθυρο δημιουργώντας όμως σημαντικές καθυστερήσεις καταστρέφοντας ουσιαστικά το παιχνίδι.

Μετά τη δημιουργία του surface που περιέχει την εικόνα που έχει εισαχθεί από τον δημιουργό, το surface μπορεί να τοποθετηθεί πάνω σε κάποιο άλλο surface, όπως για παράδειγμα στο κεντρικό παράθυρο. Για να γίνει αυτό χρησιμοποιείται η μέθοδος `Surface.blit()`. Η μέθοδος `blit()` ζωγραφίσει ένα surface, πάνω σε κάποιο άλλο surface και στο σημείο που δίνεται ως παράμετρος. Για παράδειγμα η εντολή

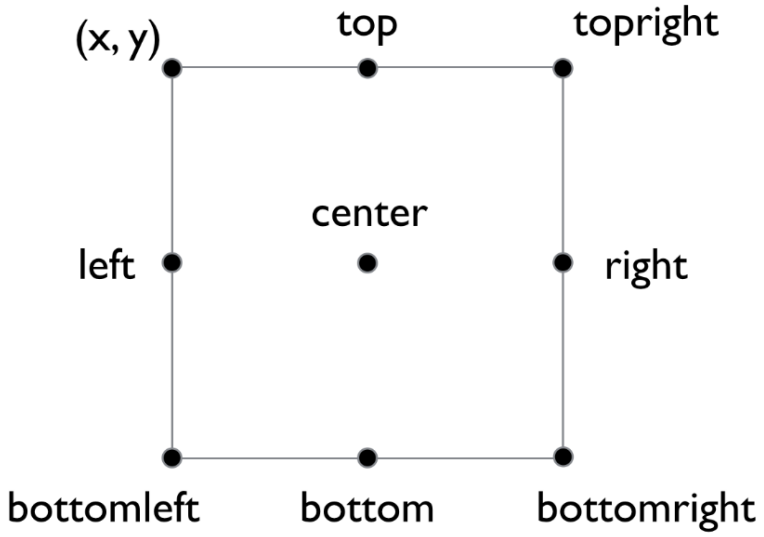
```
display_surface.blit(image_surface, (x, y))
```

θα τοποθετήσει την εικόνα στο surface display_surface στις συντεταγμένες x, y.

Το επόμενο βήμα είναι να καθορίσουμε το Rect της εικόνας. Ο όρος Rect, αποτελεί τη συντομογραφία της λέξης Rectangle (ορθογώνιο). Τα Rectangles χρησιμοποιούνται στην pygame ώστε να παρακολουθούμε τις συντεταγμένες και τις κινήσεις της κάθε εικόνας. Η εντολή `get_rect()` υπολογίζει και επιστρέφει το ορθογώνιο το οποίο περικλείει την συγκεκριμένη εικόνα. Ο παρακάτω κώδικας παίρνει το Rect της εικόνας και την τοποθετεί στο κέντρο της οθόνης.

```
rect = image_surface.get_rect()
rect.center = (WIDTH/2, HEIGHT/2)
pygame.display.update()
```

Μπορούμε να αναφερθούμε σε σημεία του Rect ώστε να δώσουμε κατάλληλες τιμές και να τοποθετηθεί η εικόνα στο σημείο που θέλουμε. Η εικόνα 4 παρουσιάζει τα σημεία αυτά.



Εικόνα 4: Σημεία αναφοράς του Rect (https://kidscancode.org/)

Μπορούμε να χρησιμοποιήσουμε το Rect όχι μόνο για να τοποθετήσουμε όπου θέλουμε την εικόνα μέσα στην οθόνη αλλά και να την κινήσουμε με βάση τις ενέργειες του χρήστη.

```
for event in pygame.event.get():
        if event.type == pygame.KEYDOWN:
            if event.key == pygame.K_LEFT:
                rect.x -= 5
            if event.key == pygame.K_RIGHT:
                rect.x += 5
```

Ο παραπάνω κώδικας μετακινεί την εικόνα 5 pixels αριστερά αν ο χρήστης πατήσει το αριστερό βελάκι, ενώ μετακινεί την εικόνα 5 pixels δεξιά όταν χρήστης πατήσει το δεξί βελάκι. Παρατηρούμε, ότι κατά την αριστερή κίνηση αφαιρούμε 5 pixels από την τρέχουσα θέση της εικόνας, ενώ κατά τη δεξιά κίνηση προσθέτουμε 5 pixels. Αυτό γίνεται γιατί όπως

αναφέραμε και στην υποενότητα 5.2.1, η αρχή των αξόνων x, y είναι η πάνω αριστερή γωνία του παραθύρου και αρχικοποιείται στις συντεταγμένες 0,0 ενώ ο άξονας των x είναι ο οριζόντιος άξονας και ο άξονας των y είναι ο κάθετος άξονας. Μία κίνηση της εικόνας κατά τον άξονα των x (αριστερά – δεξιά από την τρέχουσα θέση του) γίνεται προσθέτοντας pixel (δεξιά κίνηση) ή αφαιρώντας pixels (αριστερή κίνηση). Με παρόμοιο τρόπο γίνεται η κίνηση κατά τον y άξονα (κινήσεις πάνω και κάτω από την τρέχουσα θέση). Έτσι για να κινηθεί η εικόνα προς τα πάνω αφαιρούμε pixels, ενώ για να κινηθεί η εικόνα προς τα κάτω προσθέτουμε pixels. Ακολουθεί ο κώδικας

```
for event in pygame.event.get():
        if event.type == pygame.KEYDOWN:
            if event.key == pygame.K_UP:
                rect.y -= 5
            if event.key == pygame.K_DOWN:
                rect.y += 5
```

Το λεπτό σημείο που πρέπει να επισημανθεί, είναι ότι ανάλογα με τον άξονα πάνω στον οποίο γίνεται η κίνηση θα πρέπει να προστεθούν ή να αφαιρεθούν τα ανάλογα pixels τροποποιώντας την τιμή του Rect.x και του Rect.y αντίστοιχα.

5.2.4 Animation

Το animation είναι η ταχεία προβολή μιας σειράς εικόνων έτσι ώστε να δίνουν την ψευδαίσθηση της κίνησης. Ουσιαστικά πρόκειται για μία οφθαλμαπάτη της κίνησης και ο λόγος που συμβαίνει αυτό είναι εξ΄ αιτίας του φαινομένου της διατήρησης της εικόνας στο μάτι για 1/12 του δευτερολέπτου. Η ίδια τεχνική χρησιμοποιείται και στα παιχνίδια, προκειμένου να δημιουργεί την ψευδαίσθηση της κίνησης και να κάνει το παιχνίδι πιο ρεαλιστικό. Για να δημιουργία animation στα παιχνίδια χρησιμοποιείται μια σειρά εικόνων που ονομάζεται sprite sheet. Τα sprite sheet περιέχουν εικόνες με μικρές διαφορές μεταξύ τους, εμφανίζοντάς τες όμως με μεγάλη ταχύτητα στη σωστή σειρά, μας δίνουν την ψευδαίσθηση ότι ο χαρακτήρας κινείται, κάνει άλματα κλπ. Η πιο συνηθισμένη πρακτική είναι να εξάγουμε τις εικόνες που μας ενδιαφέρουν από ένα sprite sheet και να τις εισάγουμε σε μία δομή δεδομένων. Έτσι κάθε φορά που ο χαρακτήρας θα κάνει μία ενέργεια, θα υπάρχει η εναλλαγή των κατάλληλων εικόνων δίνοντας την αίσθηση της κίνησης. Στην εικόνα 5 δίνεται ένα παράδειγμα sprite sheet όπου δείχνει τον χαρακτήρα να χαιρετάει.

Εικόνα 5: Παράδειγμα sprite sheet (http://openbookproject.net/)

5.2.5 Αναγνώριση συγκρούσεων (Collisions detection)

Ένα από τα πιο σημαντικά κομμάτια στη δημιουργία των παιχνιδιών είναι και ο μηχανισμός αναγνώρισης συγκρούσεων (collisions detection). Μπορούμε να προσθέσουμε αυτόν τον μηχανισμό στα παιχνίδια, χρησιμοποιώντας και σε αυτή την περίπτωση τα Rect της pygame καθώς όπως αναφέρθηκε και στην προηγούμενη υποενότητα τα Rect αντιπροσωπεύουν τα αντικείμενα και τις συντεταγμένες τους. Η pygame μας δίνει αρκετούς τρόπους για τον εντοπισμό συγκρούσεων μεταξύ των αντικειμένων. Αυτό που γίνεται σε αυτή την περίπτωση είναι ένας έλεγχος για το εάν ένα Rect (Rectangle) ήρθε σε επαφή με κάποιο άλλο οπότε και έχουμε σύγκρουση. Η ακόλουθη εντολή είναι μία από τις πολλές που χρησιμοποιούνται για τον εντοπισμό συγκρούσεων.

```
pygame.sprite.spritecollide(player, enemy)
```

Η μέθοδος αυτή παίρνει ως παραμέτρους τα δύο αντικείμενα που θέλουμε να ελέγξουμε την ύπαρξη σύγκρουσης και επιστρέφει `True` αν υπάρχει σύγκρουση ή `False` αν δεν υπάρχει.

5.2.6 Game Loop

Η ¨καρδιά¨ του κάθε παιχνιδιού είναι μία επανάληψη (loop) με την ονομασία Game Loop. Αυτό το loop εκτελείται συνεχώς καθ΄ όλη την διάρκεια εκτέλεσης του παιχνιδιού, εκτελώντας όλες τις ενέργειες που απαιτούνται για να λειτουργήσει το παιχνίδι. Σε κάθε loop εκτελούνται κάποια βήματα, τα οποία ομαδοποιούνται σε τρεις κατηγορίες που είναι οι ακόλουθες:

- Χειρισμός Συμβάντων (Handle Events)

Όπως είδαμε και στην υποενότητα 5.2.2 τα events που παράγονται από το λειτουργικό σύστημα ανάλογα με τις επιλογές του χρήστη, ¨συλλαμβάνονται¨ από την pygame. Μία από τις ενέργειες που γίνονται σε κάθε loop είναι να εξετάζονται τα events που προκάλεσε ο χρήστης και να εκτελούνται οι αντίστοιχες ενέργειες όπως για παράδειγμα η παύση του παιχνιδιού, ο τερματισμός του παιχνιδιού, η μετακίνηση του χαρακτήρα κλπ.

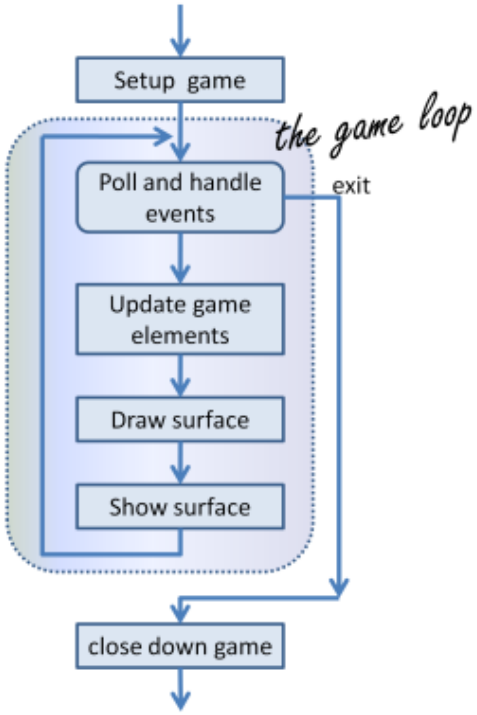
- Ενημέρωση παιχνιδιού (Update Game)

Το update του παιχνιδιού αναφέρεται στις διάφορες αλλαγές που συμβαίνουν στο παιχνίδι, είτε από το πέρασμα του χρόνου όπως για παράδειγμα η μείωση κατά μίας μονάδας του ρολογιού αντίστροφης μέτρησης του παιχνιδιού, είτε από ενέργειες του χρήση που επηρεάζουν την κίνηση του χαρακτήρα. Όλες αυτές οι αλλαγές γίνονται με τη μέθοδο update.

- Εμφάνιση αντικειμένων (Draw)

Σε αυτό το βήμα γίνεται η απεικόνιση όλων των αντικειμένων στην οθόνη. Όπως αναφέρθηκε και στην υποενότητα 5.2.4, τα πάντα σε ένα παιχνίδι είναι κινούμενες εικόνες που αποτυπώνονται ξανά και ξανά πάνω στην οθόνη. Έτσι, background εικόνες, χαρακτήρες, μενού ή οτιδήποτε άλλο ο χρήστης πρέπει να δει στην οθόνη θα πρέπει να είναι τοποθετημένα στην σωστή θέση.

Το Game Loop εκτελεί συνεχώς τις παραπάνω ενέργειες εξασφαλίζοντας την ομαλή λειτουργία του παιχνιδιού και τερματίζει την εκτέλεσή του όταν ο χρήστης κλείσει το παιχνίδι. Η εικόνα 6 δείχνει σχηματικά τις ενέργειες που εκτελεί το Game Loop.



Εικόνα 6: Ενέργειες που εκτελούνται στο Game Loop (http://openbookproject.net/)

5.3 Hello World Game

Έχοντας αναφερθεί στα σημαντικότερα κομμάτια της βιβλιοθήκης pygame θα δούμε σε μορφή κώδικα ένα απλό παιχνίδι που δεν κάνει τίποτα άλλο από το να τυπώνει κάποια μηνύματα στην οθόνη

```
import pygame
def main():
    print('Έναρξη παιχνιδιού')
    print('Αρχικοποίηση της pygame')
    pygame.init()
    print('Δημιουργία παραθύρου HelloWorldGame')
    pygame.display.set_mode((400, 400))
    pygame.display.set_caption('Hello World')
    print('Ανανέωση στοιχείων ')
    pygame.display.update()
```

```
    print('Έναρξη του Game Loop')
    running = True
    while running:
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                print('Λήφθηκε Quit Event:', event)
                running = False
    print('Game Over')
    pygame.quit()

if __name__ == '__main__':
    main()
```

Στον παραπάνω κώδικα, το πρώτο πράγμα που πραγματοποιείται η εισαγωγή της βιβλιοθήκης pygame με την εντολή `import`.

Στη συνέχεια εκτελείται η μέθοδος main το πρώτο πράγμα που εκτελείται σ΄ αυτήν είναι η μέθοδος `pygame.init()`. Αυτή η μέθοδος είναι υποχρεωτική και πρέπει να εκτελείται στην αρχή πριν από οτιδήποτε άλλο, καθώς αρχικοποιεί το περιβάλλον της pygame.

Το επόμενο βήμα είναι η δημιουργία του παραθύρου με τη μέθοδο `pygame.display.set_mode()`. Όπως αναφέρθηκε και παραπάνω η μέθοδος αυτή παίρνει ως όρισμα μία πλειάδα (tuple) με τις διαστάσεις του παραθύρου όπου σε αυτή την περίπτωση είναι 400 pixel πλάτος και 400 pixel ύψος. Η μέθοδος αυτή επιστρέφει το παράθυρο πάνω στο οποίο μπορούν να τοποθετηθούν αργότερα τα αντικείμενα του παιχνιδιού. Έπειτα, εισάγουμε όνομα στο παράθυρο με την εντολή pygame`.display.set_caption('Hello World').`

Στη συνέχεια εμφανίζουμε τα πάντα στην οθόνη. Αυτή τη στιγμή το μόνο που έχει δημιουργηθεί είναι ένα άδειο παράθυρο με τον τίτλο του. Η εμφάνιση στην οθόνη γίνεται με την εντολή `pygame.display.update()`.

Το επόμενο βήμα, είναι η είσοδος στο Game Loop. Το Game Loop στη συγκεκριμένη περίπτωση εκτελείται συνέχεια λαμβάνοντας τα events με τη μέθοδο `pygame.event.get()` και ελέγχει αν το event είναι το `pygame.QUIT`, που σημαίνει ότι ο χρήστης πάτησε το κουμπί για το κλείσιμο του παραθύρου. Όταν συμβεί αυτό αλλάζει την τιμή του flag σε `False` και το Loop Game τερματίζει.

Το τελευταίο που εκτελείται είναι ο τερματισμός της pygame με την εντολή `pygame.quit()`.

6. Το παιχνίδι Super Hercules

Στο κεφάλαιο αυτό, αναλύεται το παιχνίδι που αναπτύχθηκε στην Παρούσα Πτυχιακή εργασία με στόχο τόσο την γνωριμία του παίκτη με την ελληνική μυθολογία και τους άθλους του Ηρακλή, όσο και την ψυχαγωγία του κατά τη διάρκεια των διακοπών του και την παραμονή του στα ελληνικά ξενοδοχεία. Έτσι, στην ενότητα 6.1 περιγράφεται το παιχνίδι και οι στόχοι για τους οποίους φτιάχτηκε ενώ στην ενότητα 6.2 γίνεται μία αναφορά στο γραφικό περιβάλλον που χρησιμοποιείται. Τέλος στην ενότητα 6.3 παρουσιάζονται οι χαρακτήρες του παιχνιδιού σε μία προσπάθεια οπτικοποίησης των άθλων κάνοντας τη διαδικασία της μάθησης πιο συναρπαστική για τον παίκτη.

6.1 Περιγραφή του παιχνιδιού

Στα πλαίσια της παρούσας Πτυχιακής Εργασίας, έχει αναπτυχθεί ένα ηλεκτρονικό παιχνίδι σχετικά με την ελληνική μυθολογία και συγκεκριμένα με τους Άθλους του Ηρακλή. Σκοπός του εν λόγω λογισμικού είναι να γνωρίσουν τα μικρά παιδιά κάποιους από τους πιο γνωστούς Άθλους του Ηρακλή, συνδυάζοντας τη γνώση και την ψυχαγωγία με ηλεκτρονικά παιχνίδια. Για τον λόγο αυτό, έχει επιλεγεί το πολύ γνωστό παιχνίδι Super Mario Bros, το οποίο αποτέλεσε τη βάση, ενώ τροποποιήθηκε κατάλληλα προκειμένου να ικανοποιήσει τους σκοπούς της μάθησης. Το παιχνίδι αυτό αναπτύχθηκε με την βιβλιοθήκη pygame της γλώσσας προγραμματισμού Python και εκτελείται σε επιτραπέζιους ηλεκτρονικούς υπολογιστές. Πρόκειται για παιχνίδι δύο διαστάσεων (2D), ενώ προκειμένου να γίνει πιο ρεαλιστικό, εκτός από τα γραφικά και τις εικόνες, ενσωματώθηκαν επίσης μουσική και ήχοι για κάθε ενέργεια του Ηρακλή.

Το παιχνίδι που υλοποιήθηκε, αποτελείται από τρεις κόσμους (level), κάθε ένας από τους οποίους σχετίζεται με έναν άθλο. Πριν από την έναρξη του παιχνιδιού, ο παίκτης έχει τηη ευκαιρία να γνωρίσει τον ήρωα του παιχνιδιού, τον μυθικό Ηρακλή καθώς και για το πως ξεκίνησαν οι άθλοι του. Έπειτα και πριν από την έναρξη του κάθε κόσμου , ο παίκτης μαθαίνει για έναν άθλο του Ηρακλή, τον οποίο στη συνέχεια πρέπει να βγάλει εις πέρας. Συνολικά, έχουν υλοποιηθεί τρεις κόσμοι, κάθε ένας από τους οποίους σχετίζεται με έναν άθλο του Ηρακλή, ενώ οι άθλοι που επιλέχθηκαν είναι ¨Ο ταύρος της Κρήτης¨, ¨Το λιοντάρι της Νεμέας¨ και η ¨Λερναία Ύδρα¨.

Ο πρώτος κόσμος αφορά τον Ταύρο της Κρήτης. Πρόκειται για τον έβδομο στη σειρά άθλο του Ηρακλή, κατά τον οποίο σύμφωνα με την μυθολογία, έπιασε τον άγριο ταύρο της Κρήτης και τον πήγε ζωντανό στον Ευρυσθέα. Αφότου μάθει κάποια πράγματα για τον συγκεκριμένο άθλο, ο παίκτης καλείται να πραγματοποιήσει και αυτός τον συγκεκριμένο άθλο οδηγώντας τον Ηρακλή στο κάστρο. Για να το πετύχει αυτό, θα πρέπει να αποφύγει ή να εξουδετερώσει τους όλους τους ταύρους που θα βρεθούν στον δρόμο του. Αν τα καταφέρει και τους εξουδετερώσει, τότε οδηγείται στο κάστρο, το οποίο αποτελεί και εφαλτήριο για τον επόμενο άθλο.

Ο δεύτερος κόσμος αφορά το Λιοντάρι της Νεμέας. Η θανάτωση του λιονταριού ήταν ο πρώτος από τους δώδεκα άθλους που ο Ευρυσθέας ανέθεσε στον Ηρακλή. Ο παίκτης αφού μάθει για τον τρόπο με τον οποίο ο Ηρακλής κατάφερε και θανάτωσε το θρυλικό λιοντάρι καλείται να πράξει το ίδιο στο ηλεκτρονικό παιχνίδι. Στόχος του και σε αυτή την περίπτωση, είναι να οδηγήσει τον Ηρακλή στο ασφαλές κάστρο. Για να το καταφέρει, αυτή τη φορά θα πρέπει να αποφύγει και να εξουδετερώσει κάθε λιοντάρι που θα βρεθεί στον δρόμο του. Αν τα καταφέρει, θα βρεθεί στο κάστρο και από κει στον επόμενο κόσμο του παιχνιδιού.

Ο τρίτος και τελευταίος κόσμος έχει να κάνει την Λερναία Ύδρα, ίσως τον πιο δημοφιλή άθλο Ηρακλή. Η θανάτωση της Λερναίας Ύδρας αποτελεί τον δεύτερο σε σειρά κατόρθωμα του μυθικού ήρωα. Πρόκειται για ένα τέρας με εννιά κεφάλια, το οποίο ζούσε στη λίμνη Λέρνη, και σκορπούσε τον τρόμο στους κατοίκους της περιοχής. Οι παίκτες έχουν την ευκαιρία να μάθουν για την Λερναία Ύδρα και για το πως ο Ηρακλής κατάφερε να τη θανατώσει. Στη συνέχεια, θα πρέπει να κάνουν το ίδιο και να νικήσουν και αυτοί την Λερναία Ύδρα που υπάρχει στον τρίτο και τελευταίο κόσμο του παιχνιδιού. Εάν τα καταφέρουν, θα έχουν καταφέρει να ολοκληρώσουν το παιχνίδι έχοντας έρθει σε επαφή και έχοντας μάθει έναν από τους πιο γνωστούς ήρωες της ελληνικής μυθολογίας.

Ο παίκτης προκειμένου να ολοκληρώσει το παιχνίδι έχει τρεις ζωές, τις οποίες πρέπει να διαφυλάξει καθώς κάθε επαφή με κάποιον από τους εχθρικούς χαρακτήρες είτε του αφαιρεί δύναμη είτε του αφαιρεί μία ζωή. Όπως αναφέρθηκε και παραπάνω, το παιχνίδι που αναπτύχθηκε βασίστηκε στην φιλοσοφία του γνωστού παιχνιδιού Super Mario Bros. Κατά συνέπεια, υιοθετήθηκαν αρκετά στοιχεία του συγκεκριμένου παιχνιδιού, τα οποία προσαρμόστηκαν κατάλληλα. Έτσι, ο χαρακτήρας του Ηρακλή μπορεί να τρέξει, τα κάνει άλματα ενώ μπορεί να γίνει μεγάλος και να πυροβολεί τους αντιπάλους του. Τέλος, υπάρχει αντίστροφη μέτρηση και χρόνος μέσα στον οποίο ο παίκτης πρέπει να ολοκληρώσει τον άθλο.

6.2 Γραφικό περιβάλλον παιχνιδιού

Όπως αναφέρθηκε και στην ενότητα 6.1, το παιχνίδι που αναπτύχθηκε βασίζεται στο γνωστό ηλεκτρονικό παιχνίδι Super Mario Bros και αποτελείται από τρεις κόσμους. Κάθε κόσμος αποτελείται από μία εικόνα παρασκηνίου (background image) μέσα στην οποία τοποθετούνται και αλληλεπιδρούν μεταξύ τους άλλες εικόνες. Οι εικόνες παρασκηνίου που χρησιμοποιούνται είναι αυτές που χρησιμοποιούνται για παιχνίδι Super Mario Bros και συγκεκριμένα, είναι οι background εικόνες του level 1, level 2 και level 4 του πρώτου κόσμου του ανωτέρω παιχνιδιού.

Εικόνα 7: Εικόνα παρασκηνίου για το level 1 του παιχνιδιού (https://www.spriters-resource.com/)

Πέρα από τις εικόνες background, το παιχνίδι που αναπτύχθηκε κάνει χρήση και άλλων εικόνων του Super Mario Bros όπως είναι τα τούβλα (bricks), τα νομίσματα (coins), τα

σκαλοπάτια (steps), οι εχθροί (enemies) κοκ. Παρατηρούμε, ότι τα στοιχεία αυτά δεν είναι ενσωματωμένα στην εικόνα του background αλλά τοποθετούνται από τον δημιουργό του παιχνιδιού σε σημεία της επιλογής του. Ο τρόπος με τον οποίο τοποθετούνται τα ανωτέρω στοιχεία πάνω στην εικόνα του background θα αναλυθεί στην ενότητα 7.1.

Τέλος, για τις εικόνες των αλφαριθμητικών που χρησιμοποιούνται για την αναγραφή των πληροφοριών του παιχνιδιού στο πάνω μέρος της οθόνης χρησιμοποιήθηκαν αυτές του παιχνιδιού Super Mario Bros.

Εικόνα 8: Sprite sheet αλφαριθμητικών χαρακτήρων παιχνιδιού (https://www.spriters-resource.com/)

6.3 Χαρακτήρες παιχνιδιού

6.3.1 Ηρακλής

Όπως φανερώνει τόσο ο τίτλος, όσο και η φιλοσοφία του παιχνιδιού, ο κεντρικός χαρακτήρας του παιχνιδιού είναι ο Ηρακλής. Γιος του Δία και της Αλκμήνης, ο Ηρακλής γεννήθηκε στη Θήβα και θεωρείται ο μέγιστος των Ελλήνων ηρώων της μυθολογίας. Ο χαρακτήρας του Ηρακλή αποτελεί και τον χαρακτήρα, τον οποίο χειρίζεται ο παίκτης κατά τη διάρκεια του παιχνιδιού. Ο βασικός στόχος είναι, μέσα από τις δοκιμασίες που εμφανίζονται σε κάθε επίπεδο του παιχνιδιού, ο παίκτης να καταλάβει τη δύναμη και το θάρρος που διέκρινε τον συγκεκριμένο ήρωα. Για το λόγο αυτό, στο παιχνίδι που αναπτύχθηκε, τα χαρακτηρίστηκα του Ηρακλή (ταχύτητα, άλμα κλπ) είναι καλύτερα από αυτά των εχθρών του, ώστε να κατανοεί ο παίκτης την ανωτερότητα του ήρωά του.

Εικόνα 9: Χαρακτήρας Ηρακλή (https://www.spriters-resource.com/)

Όσον αφορά το παιχνίδι, ο χαρακτήρας του Ηρακλή όπως αναφέρθηκε και παραπάνω είναι αυτός τον οποίον χειρίζεται ο παίκτης. Ο χαρακτήρας μπορεί να περπατήσει και να τρέξει και προς τις δύο κατευθύνσεις ενώ μπορεί να κάνει άλματα και να σκύψει. Τέλος, ο χαρακτήρας του Ηρακλή μπορεί να μεγαλώσει και να έχει μεγαλύτερη δύναμη σπάζοντας τα εμπόδια που εμφανίζονται στον δρόμο του, ενώ όντας σε αυτή την κατάσταση μπορεί να κερδίσει και το πυροβόλο, κάτι που τον κάνει ακόμα πιο ισχυρό.

6.3.2 Ταύρος της Κρήτης

Πρόκειται για έναν ρωμαλέο και δυνατό ταύρο, με μαύρο τρίχωμα, ο οποίος είχε γίνει ο φόβος και το τρόμος της Κρήτης κατά τη Μινωική εποχή. Σύμφωνα με τη μυθολογία, ο ταύρος σκότωνε όποιο ζωντανό πλάσμα έβρισκε μπροστά του, είτε αυτό ήταν κάποιο άλλο ζώο, είτε ήταν άνθρωπος. Στο παιχνίδι που αναπτύχθηκε γίνεται μία προσπάθεια να τονιστεί το μέγεθος και η δύναμη του ταύρου της Κρήτης.

Εικόνα 10: Χαρακτήρας Ταύρος της Κρήτης (https://www.spriters-resource.com/)

Ο ταύρος εμφανίζεται στο πρώτο level του παιχνιδιού και έχει ως στόχο αφενός ο παίκτης να ενημερωθεί για τον συγκεκριμένο άθλο και αφετέρου για την δυσκολία του εγχειρήματος αυτού από τον Ηρακλή. Οι ταύροι έχουν ως στόχο να εμποδίζουν τον Ηρακλή να φτάσει στον στόχο του που δεν είναι άλλος από το να φτάσει στο Κάστρο και να ολοκληρώσει το level. Οι ταύροι κινούνται αριστερά και δεξιά ενώ μόλις έρθουν σε επαφή με κάποιο άλλο αντικείμενο, όπως είναι σωλήνες (pipes), άλλοι ταύροι κλπ αλλάζουν την κατεύθυνσή τους.

6.3.3 Το λιοντάρι της Νεμέας

Ο συγκεκριμένος άθλος αφορά ένα τρομερό λιοντάρι το οποίο ζούσε στα βουνά της Νεμέας και κατασπάραζε ανθρώπους και ζώα. Το λιοντάρι έμενε σε μια σπηλιά στην πιο ψηλή κορυφή του βουνού. Η δύναμη του λιονταριού αποτυπώνεται στον χαρακτήρα του στο παιχνίδι, προκειμένου και σε αυτή την περίπτωση, ο παίκτης να ¨βιώσει¨ μέσα από το παιχνίδι τη δυσκολία που αντιμετώπισε ο Ηρακλής μέχρι να το κατατροπώσει.

Εικόνα 11: Χαρακτήρας το Λιοντάρι της Νεμέας (https://www.spriters-resource.com/)

Το λιοντάρι εμφανίζεται στο δεύτερο level του παιχνιδιού, πριν από το οποίο ο παίκτης μαθαίνει για το Λιοντάρι της Νεμέας και το πως ο Ηρακλής κατάφερε και το νίκησε. Και σε αυτή την περίπτωση, γίνεται μία προσπάθεια να καταλάβει ο παίκτης, παίζοντας το παιχνίδι, τη δυσκολία και τους κινδύνους που έπρεπε να ξεπεράσει ο Ηρακλής προκειμένου να ολοκληρώσει έναν ακόμη άθλο. Τα λιοντάρια που εμφανίζονται στοχεύουν στο να πλήξουν τον ήρωα του παιχνιδιού είτε άμεσα, είτε έμμεσα οδηγώντας τον σε κάποιο μοιραίο λάθος το οποίο θα του στερήσει μία από τις ζωές του. Τα λιοντάρια, κινούνται επίσης κατά τον οριζόντιο άξονα, αριστερά και δεξιά ενώ όταν έρθουν σε επαφή με κάποιο αντικείμενο εκτός του Ηρακλή αλλάζουν την κατεύθυνσή τους.

6.3.4 Η Λερναία Ύδρα

Η Λερναία Ύδρα είναι ένα πιο τρομακτικά πλάσματα της ελληνικής μυθολογίας. Ζούσε στη λίμνη κοντά στο Άργος που λεγόταν Λέρνα. Η Λερναία Ύδρα είχε εννιά κεφάλια και κάθε βράδυ έμπαινε στον κάμπο καταστρέφοντας τις σοδιές και τα ζώα των χωρικών οι οποίοι δεν τολμούσαν να αντιμετωπίσουν το επικίνδυνο αυτό πλάσμα. Σύμφωνα με τον μύθο, αν κάποιος έκοβε ένα κεφάλι της Λερναίας Ύδρας στη θέση του φύτρωναν άλλα δύο. Αυτό έκανε ακόμα πιο τρομακτική την Λερναία Ύδρα, την οποία κλήθηκε να αντιμετωπίσει ο Ηρακλής στον δεύτερο άθλο του. Στο παιχνίδι που αναπτύχθηκε, η Λερναία Ύδρα εμφανίζεται στο τρίτο και τελευταίο επίπεδο ενώ το μέγεθος και η δυνατότητές της, δίνουν στον παίκτη να καταλάβει τόσο πόσο δύσκολη ήταν η αποστολή του Ηρακλή.

Εικόνα 12: Χαρακτήρας της Λερναίας Ύδρας (https://www.spriters-resource.com/)

Σε αντίθεση με τον Ταύρο της Κρήτης που συναντάμε στο επίπεδο ένα και το Λιοντάρι της Νεμέας του επιπέδου δύο, η Λερναία Ύδρα εκτός από κινήσεις στον οριζόντιο άξονα (κινήσεις αριστερά δεξιά) κάνει κινήσεις στον κατακόρυφο άξονα (κινήσεις πάνω κάτω). Επίσης, είναι ο μόνος εχθρικός χαρακτήρας ο οποίος ρίχνει πυρά προκειμένου να πλήξει τον Ηρακλή, ενώ δεν εμφανίζεται πολλές αλλά μόνο μία στο τέλος επιπέδου 3 όντας το τελευταίο μεγάλο εμπόδιο του παίκτη για την ολοκλήρωση του παιχνιδιού.

7. Υλοποίηση παιχνιδιού Super Hercules

Η δημιουργία ηλεκτρονικών παιχνιδιών είναι μια κοπιαστική, χρονοβόρα και επίπονη διαδικασία. Για το λόγο αυτό, στη βιομηχανία ηλεκτρονικών παιχνιδιών ολόκληρες ομάδες ατόμων διαφορετικών ειδικοτήτων συνεργάζονται προκειμένου να πετύχουν υψηλής ποιότητας αποτέλεσμα ώστε να κερδίσουν τον ενδιαφέρον των καταναλωτών. Στο κεφάλαιο , θα γίνει μία αναφορά στα βασικά σημεία της δημιουργίας του παιχνιδιού Super Hercules. Έτσι, στην ενότητα 7.1 θα γίνει μία αναφορά στο πως δημιουργήθηκαν τα επίπεδα του παιχνιδιού και πως τοποθετήθηκαν τα αντικείμενα (χαρακτήρες, εμπόδια κλπ), πάνω σε αυτά. Στην ενότητα 7.2 παρουσιάζεται ο τρόπος με τον οποίον δημιουργήθηκε ο Ηρακλής και πως πραγματοποιεί τις κινήσεις του, ενώ στην ενότητα 7.3 παρουσιάζονται οι ανωτέρω διαδικασίες για τη δημιουργία των εχθρικών χαρακτήρων του Ηρακλή. Τέλος, στη ενότητα 7.4 θα δούμε πως επιτυγχάνεται η καταστροφή εχθρικών χαρακτήρων από τον Ηρακλή.

7.1 Στάδια παιχνιδιού

Όπως όλα τα ηλεκτρονικά παιχνίδια, έτσι το παιχνίδι Super Hercules, που αναπτύχθηκε στα πλαίσια της παρούσας Πτυχιακής Εργασίας, αποτελείται από διάφορα στάδια, τα οποία εκτελούνται το ένα μετά το άλλο ώστε να εξασφαλίζεται μια συνεχή ροή δίνοντας στον χρήστη την εικόνα της ομαλής συνέχειας. Το παιχνίδι Super Hercules αποτελείται συνολικά από δώδεκα στάδια κάθε ένα από τα οποία έχει συγκεκριμένο σκοπό και λειτουργία. Σε κάθε στάδιο, εκτός των άλλων, ορίζεται το επόμενο στάδιο ενώ σημειώνεται και το προηγούμενο. Επίσης, κάθε στάδιο ¨περνάει¨ στο επόμενο, διάφορα στοιχεία όπως τις ζωές του παίκτη, τα σκορ και άλλα. Τα στάδια από τα οποία αποτελείται το παιχνίδι είναι τα ακόλουθα:

- Βασικό μενού (main menu). Αποτελεί την αρχική οθόνη του παιχνιδιού και εκτελείται μέχρι ο χρήστης να πιέσει ένα από τα πλήκτρα A και S, τα οποία είναι και τα βασικά πλήκτρα του παιχνιδιού. Στο στάδιο αυτό, απεικονίζεται εκτός από το πρώτο επίπεδο, ο Ηρακλής, το σκορ, το καλύτερο σκορ που έχει πετύχει ο παίκτης, το λογότυπο του παιχνιδιού καθώς και πληροφορίες για το παιχνίδι στο επάνω μέρος της οθόνης.
- Ιστορία (Story). Μετά το βασικό μενού ακολουθεί το στάδιο της Ιστορίας. Αποτελείται από δύο εικόνες, που διαδέχονται η μία την άλλη, οι οποίες περιγράφουν με λίγα λόγια την ιστορία του Ηρακλή και το πως ξεκίνησαν οι άθλοι του. Το στάδιο αυτό διαρκεί αρκετό χρόνο ώστε να ο χρήστης να προλάβει να διαβάσει όλη την ιστορία και να μάθει κάποια βασικά πράγματα για τον μυθικό Ηρακλή. Τέλος, το στάδιο αυτό, έχει ¨ντυθεί¨ και από κατάλληλη μουσική υπόκρουση βάζοντας τον χρήστη στο κλίμα γι’ αυτό που θα ακολουθήσει.

- Ταύρος της Κρήτης (Labor Bull). Αφότου ολοκληρωθεί η ιστορία για τον Ηρακλή και τους άθλους του, περνάμε στον πρώτο άθλο που πραγματεύεται το παιχνίδι. Το στάδιο αυτό αποτελείται από δύο εικόνες, οι οποίες δίνουν πληροφορίες για τον Ταύρο της Κρήτης. Έτσι, ο χρήστης έχει τη δυνατότητα να μάθει βασικές γνώσεις για το συγκεκριμένο άθλο πριν προχωρήσει στο πρώτο επίπεδο του παιχνιδιού. Το στάδιο αυτό, όπως και το προηγούμενο διαρκεί αρκετά δευτερόλεπτα, ώστε ο χρήστης να μπορέσει να το διαβάσει, ενώ και σε αυτό το στάδιο υπάρχει ανάλογη μουσική υπόκρουση.
- Επίπεδο 1 (Level 1). Έχοντας μάθει ο χρήστης βασικές γνώσεις για τον άθλο του Ταύρου της Κρήτης, το επόμενο βήμα είναι να τον ¨αντιμετωπίσει¨ και στο παιχνίδι. Το στάδιο αυτό, αποτελεί και τον πρώτο γύρο του παιχνιδιού, στον οποίο ο χρήστης προσπαθεί να αποφύγει ή να εξουδετερώσει τους ταύρους που εμφανίζονται. Εκτός από την εικόνα παρασκηνίου έχουν τοποθετηθεί οι χαρακτήρες (Ηρακλής, ταύροι) καθώς και διάφορα άλλα αντικείμενα όπως τούβλα (bricks), κέρματα (coins) καθώς και στοιχεία τα οποία δίνουν δύναμη και παραπάνω δυνατότητες στον Ηρακλή. Τέλος, προκειμένου να διατηρηθεί ο χαρακτήρας του παιχνιδιού, χρησιμοποιείται τόσο μουσική όσο και διάφοροι ήχοι.
- Λιοντάρι της Νεμέας (Labor Lion). Μετά την επιτυχή ολοκλήρωση το πρώτου επιπέδου, ο χρήστης μαθαίνει για το Λιοντάρι της Νεμέας. Όπως και το επίπεδο Ταύρος της Κρήτης (Labor Bull) έτσι και το συγκεκριμένο επίπεδο, αποτελείται από δύο εικόνες, οι οποίες εναλλάσσουν η μία την άλλη και περιγράφουν τον εν λόγο άθλο.
- Επίπεδο 2 (Level 2). Το επίπεδο αυτό ακολουθεί το επίπεδο Λιοντάρι της Νεμέας (Labor Lion) και αποτελεί τον δεύτερο γύρο του παιχνιδιού. Στόχος του χρήστη στο συγκεκριμένο επίπεδο είναι η αποφυγή ή εξουδετέρωση των λιονταριών που εμφανίζονται, με στόχο να οδηγηθεί ο Ηρακλής στο Κάστρο και στο επόμενο επίπεδο του παιχνιδιού. Και σε αυτό το επίπεδο έχουν τοποθετηθεί σε κατάλληλα σημεία τόσο οι χαρακτήρες όσο και τα άλλα αντικείμενα, τα οποία βοηθούν τον παίκτη να ολοκληρώσει τον γύρο, ενώ η μουσική και οι ήχοι που έχουν ενσωματωθεί βοηθούν στο να γίνει το παιχνίδι πιο ρεαλιστικό.
- Λερναία Ύδρα (Labor Label). Στο στάδιο αυτό, ο χρήστης μαθαίνει για την Λερναία Ύδρα και πως ο Ηρακλής χρησιμοποιώντας την ευστροφία του κατάφερε να την εξουδετερώσει. Αποτελείται από δύο εικόνες οι οποίες εξιστορούν με λίγα λόγια τον συγκεκριμένο άθλο προετοιμάζοντας τον χρήστη για τον επόμενο γύρο του παιχνιδιού. Η μουσική κάνει την εμφάνισή της και σ’ αυτόν τον γύρο, ενώ ο χρήστης έχει ικανοποιητικό χρόνο για να διαβάσει την ιστορία.
- Επίπεδο 3 (Level 3). Αποτελεί τον τελευταίο γύρο του παιχνιδιού, στο οποίο ο χρήστης έρχεται σε επαφή με την Λερναία Ύδρα. Για να φτάσει να αντιμετωπίσει όμως το θρυλικό τέρας, θα πρέπει πρώτα να αποφύγει τις παγίδες που του έχει στήσει σε όλο το μήκος της διαδρομής. Η ιστορία στο επίπεδο αυτό διαδραματίζεται μέσα στο κάστρο, ενώ η μουσική που χρησιμοποιείται δίνει τον ανάλογο δραματικό τόνο. Σε περίπτωση που ο χρήστης νικήσει την Λερναία Ύδρα, τότε ολοκληρώνει το παιχνίδι.

- Τέλος Ιστορίας (Game end). Η κατάσταση αυτή εμφανίζεται σε περίπτωση που ο χρήστης ολοκληρώσει με επιτυχία όλες τις πίστες του παιχνιδιού, αποτελείται από μία εικόνα που συγχαίρει τον χρήστη για την επιτυχία του. Στη συνέχεια, ο χρήστης οδηγείται στην αρχική οθόνη απ' όπου μπορεί να παίξει εκ νέου το παιχνίδι.
- Τέλος Χρόνου (Time Out). Ο χρήστης έχει στη διάθεσή του συγκεκριμένο χρόνο για την ολοκλήρωση ενός επιπέδου. Όταν ο χρόνος αυτός παρέλθει και ο χρήστης δεν έχει καταφέρει να ολοκληρώσει το επίπεδο, τότε λαμβάνει χώρα η εν λόγω κατάσταση. Η οθόνη της κατάστασης αυτής, ενημερώνει τον χρήστη για το τέλος του χρόνου αφαιρώντας από αυτόν μία ζωή. Διαρκεί λίγα μόλις δευτερόλεπτα και στη συνέχεια αν ο χρήστης έχει και άλλη ζωή επιστρέφει στο παιχνίδι. Σε διαφορετική περίπτωση, το παιχνίδι ολοκληρώνεται.
- Τέλος Παιχνιδιού (Game Over). Η κατάσταση αυτή σηματοδοτεί το τέλος του παιχνιδιού καθώς ο παίκτης δεν κατάφερε να το ολοκληρώσει. Κάθε παίκτης έχει στη διάθεσή του τρεις ζωές προκειμένου να ολοκληρώσει το παιχνίδι. Σημειώνεται, ότι κατά τη διάρκεια του παιχνιδιού μπορεί να κερδίσει και άλλες ζωές. Κάθε φορά που ο παίκτης έρχεται σε επαφή με κάποιον εχθρικό χαρακτήρα ή πέφτει στο κενό χάνει μία ζωή. Αν ο παίκτης χάσει όλες τις ζωές του, τότε λαμβάνει χώρα η συγκεκριμένη κατάσταση, η οποία ενημερώνει τον χρήστη για το τέλος του παιχνιδιού ενώ ο έλεγχος πηγαίνει στην αρχικό μενού, απ' όπου ο χρήστης μπορεί να ξεκινήσει ένα νέο παιχνίδι.

7.2 Δημιουργία των κόσμων

Στην ενότητα αυτή θα αναλυθεί ο τρόπος με τον οποίο δημιουργήθηκαν τα τρία επίπεδα του παιχνιδιού, κάθε ένα από τα οποία αντιπροσωπεύει και έναν άθλο του Ηρακλή. Στην ενότητα 6.2 αναφέρθηκε ότι ως εικόνες παρασκηνίου (background images), των τριών επιπέδων, χρησιμοποιήθηκαν αυτές του γνωστού και επιτυχημένου παιχνιδιού Super Mario Bros. Μία εικόνα παρασκηνίου δεν είναι τίποτε άλλο παρά μία εικόνα στην οποία τοποθετούμε πάνω τα διάφορα αντικείμενα, που στην περίπτωσή μας είναι ο χαρακτήρας του Ηρακλή, οι ταύροι, τα λιοντάρια, η λερναία Ύδρα καθώς και μία σειρά από άλλα αντικείμενα όπως τούβλα, εμπόδια, κέρματα τα οποία εμπλουτίζουν το παιχνίδι. Χάριν παραδείγματος, θα χρησιμοποιήσουμε παρακάτω τις εντολές για τη δημιουργία του πρώτου επιπέδου του παιχνιδιού.

Το πρώτο βήμα για την δημιουργία ενός γύρου του παιχνιδιού είναι να εισάγουμε την εικόνα υποβάθρου (background image). Η διαδικασία αυτή γίνεται κατά την αρχικοποίηση του γύρου με τη συνάρτηση setup_background() που ορίζεται ως ακολούθως.

```
    def setup_background(self):
        self.background = setup.GFX['level_1']
        self.back_rect = self.background.get_rect()
        self.background = pg.transform.scale(self.background,
(int(self.back_rect.width*c.BACKGROUND_MULTIPLER),
int(self.back_rect.height*c.BACKGROUND_MULTIPLER)))
```

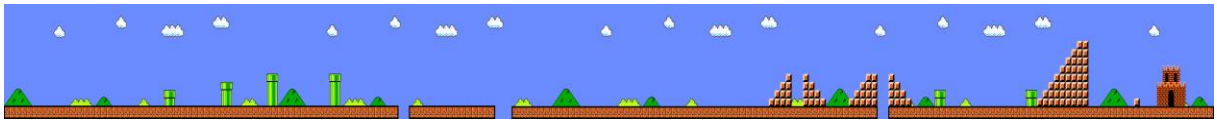
```
self.back_rect = self.background.get_rect()
```

Σ' αυτό το σημείο πρέπει να τονιστεί, ότι η εικόνα που εισάγεται πολλαπλασιάζεται κατά μήκος και κατά πλάτος με το μέγεθος `c.BACKGROUND_MULTIPLIER`, το οποίο αντιστοιχεί στην τιμή 2,679. Ο λόγος που γίνεται αυτό είναι, ότι η εικόνα έχει μικρές διαστάσεις οπότε προκειμένου να βελτιώσουμε την εμπειρία του χρήστη την μεγεθύνουμε.

Το επόμενο βήμα είναι να ορίσουμε τις περιοχές (έδαφος) πάνω στις οποίες μπορούν να κινηθούν οι χαρακτήρες. Για να το πετύχουμε αυτό, δημιουργούμε ορθογώνια (Rectangles – Rect) πάνω στα οποία τοποθετούμε τους χαρακτήρες. Έτσι αν ένας χαρακτήρας βγει από αυτές τις περιοχές τότε θεωρείται ότι βρίσκεται στο κενό, οπότε και πέφτει μέσα σ' αυτό. Ο ορισμός του εδάφους γίνεται κατά την αρχικοποίηση με τη συνάρτηση setup_ground() και ορίζεται ως εξής:

```
def setup_ground(self):
    ground_rect1 = collider.Collider(0, c.GROUND_HEIGHT,    2953, 60)
    ground_rect2 = collider.Collider(3048, c.GROUND_HEIGHT,  635, 60)
    ground_rect3 = collider.Collider(3819, c.GROUND_HEIGHT, 2735, 60)
    ground_rect4 = collider.Collider(6647, c.GROUND_HEIGHT, 2300, 60)
```

Παρατηρούμε ότι δημιουργούμε τέσσερα ορθογώνια τα οποία ορίζουν το έδαφος. Η εντολή collider.Collider() έχει τέσσερα ορίσματα τα οποία είναι κατά σειρά το σημείο x, το σημείο y, το πλάτος του Rectangle και το ύψος. Αξίζει να σημειωθεί ότι οι τιμές αυτές είναι πολλαπλασιασμένες με τον αριθμό 2,679 με τον οποίο πολλαπλασιάστηκε και η εικόνα υποβάθρου. Έτσι, αν θέλουμε να βρούμε τις συντεταγμένες αυτές πάνω στην εικόνα θα πρέπει να τις διαιρέσουμε με τον αριθμό 2,679. Μετά την εκτέλεση του παραπάνω κώδικα, το έδαφος του επιπέδου θα έχει την ακόλουθη μορφή.

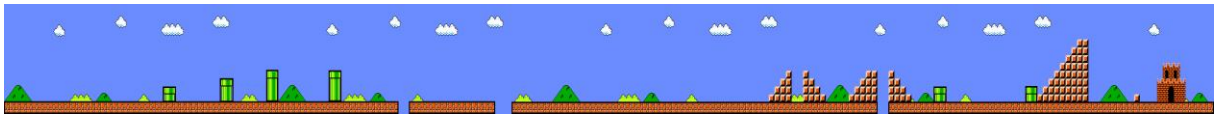


Εικόνα 13: Εικόνα υποβάθρου μετά την εισαγωγή των rectangle για τον ορισμό του εδάφους (https://www.spriters-resource.com/)

Το επόμενο βήμα να ορίσουμε τα Rectangles για κάθε σωλήνα (pipe) που υπάρχει, προκειμένου οι χαρακτήρες να αλληλεπιδρούν με αυτό σαν να είναι εμπόδιο και να μην περνάνε από μέσα. Αυτό επιτυγχάνεται κατά την αρχικοποίηση του επιπέδου με τη μέθοδο setup_pipes() που έχει την ακόλουθη υλοποίηση

```
def setup_pipes(self):
        pipe1 = collider.Collider(1202, 452, 83, 82)
        pipe2 = collider.Collider(1631, 409, 83, 140)
        pipe3 = collider.Collider(1973, 366, 83, 170)
        pipe4 = collider.Collider(2445, 366, 83, 170)
        pipe5 = collider.Collider(6989, 452, 83, 82)
        pipe6 = collider.Collider(7675, 452, 83, 82)
```

Παρατηρούμε ότι δημιουργούμε έξι Rectangles όσες και οι σωλήνες της εικόνας, ενώ και σε αυτή την περίπτωση αν θέλουμε να βρούμε τις πραγματικές συντεταγμένες πάνω στην εικόνα, διαιρούμε όλα τα ορίσματα με τον αριθμό 2,679. Η εικόνα του background έχει πλέον την ακόλουθη μορφή,



Εικόνα 13: Εικόνα υποβάθρου μετά την εισαγωγή των rectangle για τον ορισμό των σωλήνων (pipes) (https://www.spriters-resource.com/)

Παρατηρώντας πιο προσεκτικά την εικόνα, βλέπει κανείς και τα διάφορα σκαλοπάτια (steps) τα οποία πρέπει επίσης να οριστούν ως Rectangles, προκειμένου οι χαρακτήρες να μην περνάνε από μέσα. Και αυτή η διαδικασία γίνεται κατά την αρχικοποίηση του επιπέδου με τη μέθοδο setup_steps() που για το πρώτο επίπεδο υλοποιείται ως ακολούθως

```
def setup_steps(self):
        step1 = collider.Collider(5745, 495, 40, 44)
        step2 = collider.Collider(5788, 452, 40, 44)
        step3 = collider.Collider(5831, 409, 40, 44)
        step4 = collider.Collider(5874, 366, 40, 176)
        step5 = collider.Collider(6001, 366, 40, 176)
        step6 = collider.Collider(6044, 408, 40, 40)
        step7 = collider.Collider(6087, 452, 40, 40)
        step8 = collider.Collider(6130, 495, 40, 40)
        step9 = collider.Collider(6345, 495, 40, 40)
        step10 = collider.Collider(6388, 452, 40, 40)
        step11 = collider.Collider(6431, 409, 40, 40)
        step12 = collider.Collider(6474, 366, 40, 40)
        step13 = collider.Collider(6517, 366, 40, 176)
        step14 = collider.Collider(6644, 366, 40, 176)
        step15 = collider.Collider(6687, 408, 40, 40)
        step16 = collider.Collider(6728, 452, 40, 40)
        step17 = collider.Collider(6771, 495, 40, 40)
        step18 = collider.Collider(7760, 495, 40, 40)
```

```
step19 = collider.Collider(7803, 452, 40, 40)
step20 = collider.Collider(7845, 409, 40, 40)
step21 = collider.Collider(7888, 366, 40, 40)
step22 = collider.Collider(7931, 323, 40, 40)
step23 = collider.Collider(7974, 280, 40, 40)
step24 = collider.Collider(8017, 237, 40, 40)
step25 = collider.Collider(8060, 194, 40, 40)
step26 = collider.Collider(8103, 194, 40, 360)
step27 = collider.Collider(8488, 495, 40, 40)
```

Μετά την εκτέλεση της παραπάνω μεθόδου, η εικόνα του background όσον αφορά τα Rectangle έχει την ακόλουθη μορφή.

Εικόνα 14: Εικόνα υποβάθρου μετά την εισαγωγή των rectangle για τον ορισμό των σκαλοπατιών (steps) (https://www.spriters-resource.com/)

Έχοντας ορίσει όλα τα απαραίτητα Rectangles που σχετίζονται με το έδαφος και να εμπόδια της εικόνας, μπορούμε να προχωρήσουμε στην εισαγωγή άλλων αντικειμένων όπως τα τούβλα (bricks). Αυτή η διαδικασία γίνεται κατά την αρχικοποίηση του επιπέδου με τη μέθοδο setup_bricks(). Η φιλοσοφία που ακολουθείται είναι η ίδια που ακολουθήθηκε και προηγούμενος. Επιλέγουμε το σημείο που θέλουμε να εισάγουμε ένα τούβλο (συντεταγμένες x και y) και τις πολλαπλασιάζουμε με τον πολλαπλασιαστή 2,679. Οι δύο τιμές που προκύπτουν είναι τα ορίσματα της μεθόδου setup_bricks(). Για παράδειγμα αν θέλουμε να εισάγουμε ένα τουβλάκι στις συντεταγμένες της εικόνας 320, 136 τότε ως ορίσματα της μεθόδου θα εισάγουμε τα 858 (320 x 2,679 = 858) και 365 (136 x 2,679 = 365) οπότε έχουμε την εντολή `brick1 = bricks.Brick(858, 365)` όπου Brick είναι η κλάση την οποία έχουμε υλοποιήσει και δημιουργεί αντικείμενα αυτού του τύπου.

Με τον ίδιο ακριβός τρόπο δημιουργούμε και άλλα αντικείμενα όπως coin boxes και coins. Πλέον, το επίπεδο είναι έτοιμο και το μόνο που μένει είναι η εισαγωγή των χαρακτήρων που θα αναλυθεί στις επόμενες ενότητες.

7.3 Δημιουργία και κίνηση χαρακτήρα Ηρακλή

Έχοντας δημιουργήσει τον ¨κόσμο¨ του παιχνιδιού, τοποθετώντας τα διάφορα στοιχεία στις επιθυμητές θέσεις, σειρά παίρνει η αρχικοποίηση του Ηρακλή, που είναι και ο κεντρικός χαρακτήρας του παιχνιδιού και αυτός που χειρίζεται ο εκάστοτε παίκτης. Και αυτή η διαδικασία γίνεται κατά την αρχικοποίηση του επιπέδου, με τη συνάρτηση setup_hercules(), η οποία έχει την ακόλουθη υλοποίηση

```
def setup_hercules(self):
```

```
self.hercules = hercules.Hercules()
self.hercules.rect.x = self.viewport.x + 110
self.hercules.rect.bottom = c.GROUND_HEIGHT
```

Στο παραπάνω κώδικα, το πρώτο που γίνεται είναι να δημιουργηθεί ένα αντικείμενο τύπου Hercules, το οποίο είναι ο χαρακτήρας του Ηρακλή. Στη συνέχεια τοποθετούμε τον χαρακτήρα, με τη χρήση του Rectangle, 110px δεξιά του viewport κατά τον άξονα των x και 538px κατά τον άξονα των y, ώστε να δίνεται η ψευδαίσθηση ότι ο χαρακτήρας πατάει στο έδαφος που δημιουργήσαμε προηγουμένως και πάλι με την χρήση του Rectangle.

Ο Ηρακλής είναι ο μόνος χαρακτήρας μπορεί να χειριστεί ο παίκτης του παιχνιδιού. Κάθε μία ενέργεια που μπορεί να κάνει ο Ηρακλής είναι και μία κατάσταση, στην οποία και βρίσκεται όσο εκτελεί τη συγκεκριμένη ενέργεια. Κάθε φορά που ο παίκτης πιέζει ένα πλήκτρο, η κατάσταση στην οποία βρίσκεται ο χαρακτήρας αλλάζει, ανάλογα με το πλήκτρο που πατήθηκε. Επιπλέον, εξετάζονται διάφορα σενάρια για το αν ο χαρακτήρας είναι μικρός ή μεγάλος, αν μπορεί να πυροβολήσει, ο προσανατολισμός του κοκ. Οι καταστάσεις στις οποίες μπορεί να βρεθεί ο Ηρακλής είναι οι ακόλουθες:

- STAND. Είναι η κατάσταση στην οποία ο Ηρακλής είναι σταθερός και αποτελεί την αρχική κατάσταση του χαρακτήρα όταν δημιουργείται.
- WALK. Είναι η κατάσταση στην οποία ο χαρακτήρας κινείται προς μία κατεύθυνση. Η κίνηση του χαρακτήρα επιτυγχάνεται με το αριστερό και δεξί βελάκι, μετακινώντας τον χαρακτήρα προς τα αριστερά ή τα δεξιά αντίστοιχα. Και στις δύο περιπτώσεις, υπάρχει μία εναλλαγή των εικόνων ώστε να δίνει στον χρήστη την ψευδαίσθηση της κίνησης, ενώ παράλληλα κινείται και η background εικόνα προκειμένου να δίνεται η αίσθηση της προόδου μέσα στο επίπεδο. Η ταχύτητα κίνησης του χαρακτήρα είναι σταθερή, ενώ πατώντας ο χρήστης το πλήκτρο S, ο χαρακτήρας κινείται πιο γρήγορα.
- JUMP. Είναι η κατάσταση στην οποία ο χαρακτήρας εκτελεί άλμα προς μία κατεύθυνση. Για να πραγματοποιηθεί αυτό, ο χρήστης πρέπει να πατήσει το πλήκτρο A. Σε αυτή την περίπτωση πραγματοποιείται άλμα επιτόπου. Σε διαφορετική περίπτωση, αν ο χρήστης πιέσει εκτός από το πλήκτρο A και το δεξί ή αριστερό βελάκι τότε ο χαρακτήρας θα εκτελέσει το άλμα και θα προσγειωθεί σε κάποιο σημείο δεξιά ή αριστερά του αρχικού σημείου. Όταν ο χαρακτήρας φτάσει το ανώτατο σημείο άλματος, τότε η κατάσταση του χαρακτήρα αλλάζει σε FALL και ξεκινάει η ομαλή του πτώση στο έδαφος. Σημειώνεται ότι ο ύψος του άλματος εξαρτάται από την ταχύτητα που κινείται ο χαρακτήρας (μεγαλύτερη ταχύτητα μεγαλύτερο άλμα) καθώς και από το μέγεθος του χαρακτήρα (μεγάλος χαρακτήρας μεγαλύτερο άλμα). Και σε αυτή την περίπτωση κατά την εκτέλεση του άλματος γίνεται εναλλαγή των εικόνων για να γίνει το παιχνίδι πιο ρεαλιστικό.
- DEATH_JUMP. Στην κατάσταση αυτή, ο παίκτης έχει χάσει μία ζωή είτε λόγω έλλειψης χρόνου είτε λόγω επαφής του με κάποιον εχθρό. Στην περίπτωση αυτή, επιλέγεται το κατάλληλο frame και ο χαρακτήρας μεταβαίνει σε κατάσταση

DEATH_JUMP κατά την οποία κάνει ένα άλμα και στη συνέχεια εξαφανίζεται από την οθόνη προκειμένου να ξεκινήσει το παιχνίδι από την αρχή.

- SMALL_TO_BIG. Πρόκειται για την κατάσταση στην οποία ο Ηρακλής έχει κερδίσει κάποιο powerup (αντικείμενο που του δίνει δύναμη). Σε αυτή τη περίπτωση, γίνεται εναλλαγή στα frames ώστε να χρησιμοποιούνται αυτά στα οποία ο Ηρακλής είναι μεγάλος. Κατά τη διάρκεια της μετατροπής, ο χαρακτήρας είναι άτρωτος και δεν μπορεί να πληγεί από κάποιον εχθρό, ενώ στη συνέχεια μεταβαίνει σε κατάσταση WALK.
- BIG_TO_FIRE. Είναι η κατάσταση στην οποία ο Ηρακλής κερδίζει το δεύτερο powerup και αποκτά τη δυνατότητα να πυροβολεί τους αντιπάλους του. Από το σημείο αυτό και έπειτα, ο χρήστης πατώντας το πλήκτρο S, μπορεί να πυροβολεί, ενώ υπάρχει περιορισμός στο να ρίχνει δύο σφαίρες κάθε φορά και όχι περισσότερες.
- BIG_TO_SMALL. Σε αντίθεση με τις δύο τελευταίες καταστάσεις, στην περίπτωση αυτή, ο χαρακτήρας χάνει δύναμη ύστερα από επαφή του με κάποιον εχθρό. Στην περίπτωση αυτή, ο χαρακτήρας μικραίνει και γίνεται πιο ευάλωτος ενώ χάνει και την ικανότητά του να πυροβολεί. Κατά την μετατροπή του από μεγάλο σε μικρό αλλάζουν και πάλι τα frames που χρησιμοποιούνται, ενώ σ' αυτό το διάστημα ο χαρακτήρας είναι άτρωτος και δεν μπορεί να πληγεί περεταίρω από τους εχθρούς του.
- FLAGPOLE. Ο Ηρακλής περιέρχεται στην συγκεκριμένη κατάσταση όταν κάνει άλμα στον ιστό της σημαίας που σηματοδοτεί και το τέλος του συγκεκριμένου επιπέδου. Στην συνέχεια, ο χαρακτήρας κατεβάζει την σημαία ενώ ταυτόχρονα υπάρχει εναλλαγή των frames προκειμένου να το κάνουν πιο ρεαλιστικό. Αξίζει να σημειωθεί ότι αυτό γίνεται αυτόματα και δεν απαιτείται η ενέργεια του χρήστη.
- BOTTOM_OF_POLE. Μόλις ο Ηρακλής κατεβάσει τη σημαία, τότε περιέρχεται στην συγκεκριμένη κατάσταση.
- WALKING_TO_CASTLE. Η κατάσταση αυτή ακολουθεί την κατάσταση BOTTOM_OF_POLE και οδηγεί τον Ηρακλή στο κάστρο και σηματοδοτεί το τέλος του επιπέδου. Σημειώνεται, ότι και σε αυτή την περίπτωση, η κίνηση είναι αυτόματη και δεν απαιτείται ενέργεια του χρήστη.

7.4 Δημιουργία και τοποθέτηση εχθρικών χαρακτήρων

Μετά και την τοποθέτηση του κεντρικού χαρακτήρα του παιχνιδιού, σειρά παίρνουν οι εχθρικοί χαρακτήρες κάθε ένας από τους οποίους αναφέρεται και σε έναν άθλο. Στην περίπτωση αυτή όμως τα πράγματα είναι διαφορετικά. Επειδή, οι χαρακτήρες δεν είναι σταθεροί αλλά κινούνται ανεξάρτητα στον χώρο, η τοποθέτησή τους στον ¨κόσμο¨ του παιχνιδιού βασίζεται στη θέση του Ηρακλή στο παιχνίδι.

Αρχικά, το πρώτο που γίνεται είναι η δημιουργία αντικειμένων του εχθρού. Στο παιχνίδι που αναπτύχθηκε υπάρχουν συνολικά πέντε ήδη εχθρών. Έτσι, έχουμε τον εχθρικό χαρακτήρα Bull όπου εμφανίζεται στο πρώτο επίπεδο, το οποίο και ασχολείται με τον Ταύρο της Κρήτης, τον χαρακτήρα Lion που εμφανίζεται στο δεύτερο επίπεδο και αφορά το Λιοντάρι της Νεμέας και τον χαρακτήρα Hydra, που εμφανίζεται στο τρίτο και τελευταίο επίπεδο που αναφέρεται στην Λερναία Ύδρα. Οι άλλοι δύο εχθρικοί χαρακτήρες είναι η Fire (φλόγα) που

δημιουργεί η Λερναία Ύδρα καθώς και τα Firestick, τα οποία αποτελούν τα κύρια εμπόδια στο δρόμο του Ηρακλή προς στη Λερναία Ύδρα κατά το τρίτο επίπεδο.

Έτσι, κατά την αρχικοποίηση του κάθε επιπέδου, εκτελείται η συνάρτηση setup_enemies() η οποία δημιουργεί τους αντίστοιχους εχθρικούς χαρακτήρες. Η υλοποίηση της συγκεκριμένης συνάρτησης για το πρώτο επίπεδο του παιχνιδιού είναι η ακόλουθη.

```
def setup_enemies(self):
    bull0 = enemies.Bull()
    bull1 = enemies.Bull()
    bull2 = enemies.Bull()
    bull3 = enemies.Bull()
    bull4 = enemies.Bull(193)
    bull5 = enemies.Bull(193)
    bull6 = enemies.Bull()
    bull8 = enemies.Bull()
    bull9 = enemies.Bull()
    bull10 = enemies.Bull()
    bull12 = enemies.Bull()
    bull14 = enemies.Bull()
    bull16 = enemies.Bull()

    enemy_group1 = pg.sprite.Group(bull0)
    enemy_group2 = pg.sprite.Group(bull1)
    enemy_group3 = pg.sprite.Group(bull2, bull3)
    enemy_group4 = pg.sprite.Group(bull4, bull5)
    enemy_group5 = pg.sprite.Group(bull6)
    enemy_group6 = pg.sprite.Group(bull16)
    enemy_group7 = pg.sprite.Group(bull8, bull9)
    enemy_group8 = pg.sprite.Group(bull10)
    enemy_group9 = pg.sprite.Group(bull12)
    enemy_group10 = pg.sprite.Group(bull14)

    self.enemy_group_list = [enemy_group1, enemy_group2,
                             enemy_group3, enemy_group4,
```

```
                                    enemy_group5, enemy_group6,
                                    enemy_group7, enemy_group8,
                                    enemy_group9, enemy_group10]
```

Από τον παραπάνω κώδικα, παρατηρούμε ότι έχουν κατασκευαστεί συνολικά δεκαέξι αντικείμενα τύπου Bull, τα οποία έχουν τοποθετηθεί σε δομές sprite.Group(). Η δομή sprite.Group() βοηθάει στο να έχουν όλα τα αντικείμενα της δομής την ίδια συμπεριφορά. Έπειτα, όλα τα sprite.Group() αποθηκεύονται σε μία δομή λίστας. Όπως παρατηρείται, σε κανένα από τα δεκαέξι αντικείμενα τύπου Bull που δημιουργήθηκαν δεν έχουν οριστεί οι συντεταγμένες στις οποίες θα εμφανιστούν. Στόχος μας είναι τα εχθρικά αντικείμενα να τοποθετηθούν μέσα στο παιχνίδι την κατάλληλη χρονική στιγμή, όταν δηλαδή πλησιάζει ο Ηρακλής. Για τον λόγο αυτό δημιουργούμε τα αντικείμενα τύπου Checkpoint, τα οποία δεν είναι τίποτα άλλο από σημεία πάνω στο παιχνίδι, αόρατα, τα οποία προκαλούν ένα συμβάν μόλις ο Ηρακλής περάσει από αυτά. Ένα από αυτά είναι και η τοποθέτηση των εχθρικών αντικειμένων στον κόσμο του παιχνιδιού.

Έτσι, κατά την αρχικοποίηση του επιπέδου, εκτελείται η μέθοδος setup_checkpoints() στην οποία ορίζουμε τα συγκεκριμένα σημεία. Η υλοποίηση της μεθόδου για το πρώτο επίπεδο του παιχνιδιού είναι η ακόλουθη,

```
    def setup_checkpoints(self):
        check1 = checkpoint.Checkpoint(510, "1")
        check2 = checkpoint.Checkpoint(1400, '2')
        check3 = checkpoint.Checkpoint(1740, '3')
        check4 = checkpoint.Checkpoint(3080, '4')
        check5 = checkpoint.Checkpoint(3750, '5')
        check6 = checkpoint.Checkpoint(4150, '6')
        check7 = checkpoint.Checkpoint(4470, '7')
        check8 = checkpoint.Checkpoint(4950, '8')
        check9 = checkpoint.Checkpoint(5100, '9')
        check10 = checkpoint.Checkpoint(6800, '10')
        check11 = checkpoint.Checkpoint(8504, '11', 5, 6)
        check12 = checkpoint.Checkpoint(8775, '12')
```

Αξίζει να σημειωθεί, ότι δεν είναι όλα τα checkpoint για τη τοποθέτηση εχθρικών χαρακτήρων καθώς κάποια από αυτά σηματοδοτούν το τέλος του συγκεκριμένου επιπέδου κλπ. Το πρώτο όρισμα της μεθόδου είναι η συντεταγμένη x του checkpoint πολλαπλασιασμένη με τον αριθμό 2,679 με τον οποίο πολλαπλασιάστηκαν οι διαστάσεις του background, ενώ το δεύτερο όρισμα είναι το όνομα του checkpoint.

Το επόμενο βήμα είναι η δημιουργία ενός μηχανισμού το οποίο θα τοποθετεί τους εχθρούς μόλις ο Ηρακλής περάσει αυτά τα σημεία. Αυτό γίνεται με τη μέθοδο check_points_check() η υλοποίηση της οποίας στο πρώτο επίπεδο του παιχνιδιού είναι η ακόλουθη,

```
    def check_points_check(self):
     checkpoint = pg.sprite.spritecollideany(self.hercules,
     self.check_point_group)
        # Αν έχει περάσει από checkpoint
        if checkpoint:
            checkpoint.kill()
             #Τα checkpoint 1 εως 10 πυροδοτούν την δημιουργία
των enemies που
          #έχουμε ορίσει παραπάνω. Οπότε μόλις περάσουμε ένα
          από αυτά τα       #checkpoints δημιουργούνται τα
          αντίστοιχα enemies
            for i in range(1,11):
                if checkpoint.name == str(i):
                    for index, enemy in
enumerate(self.enemy_group_list[i -1]):
                        enemy.rect.x = self.viewport.right +
(index * 60)

self.enemy_group.add(self.enemy_group_list[i-1])
```

Μετά την τοποθέτησή τους στον ¨κόσμο¨ του παιχνιδιού, τα εχθρικά αντικείμενα κινούνται αυτόνομα. Εξ' ορισμού, η κίνησή τους είναι προς τα αριστερά, ενώ όταν έρθουν σε επαφή με άλλο αντικείμενο, εκτός του χαρακτήρα Ηρακλή, όπως είναι σωλήνες (pipes), σκαλάκια (steps) ή άλλοι εχθροί, κινούνται προς την αντίθετη κατεύθυνση. Η βασική τους κίνηση είναι κατά τον άξονα των x (αριστερά, δεξιά) ενώ μόνο ο χαρακτήρας Hydra (Λερναία Ύδρα) μπορεί να κάνει και άλματα. Προκειμένου να γίνει το παιχνίδι πιο ρεαλιστικό, κατά την κίνηση των εχθρικών χαρακτήρων υπάρχει εναλλαγή μεταξύ των αντίστοιχων frames ώστε να φαίνεται η κίνηση των άκρων τους. Επίσης, όπως αναφέρθηκε και παραπάνω, ο χαρακτήρας της Hydra (Λερναία Ύδρα) είναι ο μόνος χαρακτήρας ο οποίος πυροβολεί τον Ηρακλή με αντικείμενα Fire (φωτιά). Η κίνηση αυτή, όπως και όλες οι κινήσεις του συγκεκριμένου χαρακτήρα, γίνεται σε τυχαίο χρόνο. Τέλος, υπάρχει και το εχθρικό αντικείμενο Firestick, το οποίο εμφανίζεται στο τελευταίο επίπεδο. Πρόκειται για μία δέσμη φωτιάς η οποία κινείται κυκλικά και στόχος του είναι να πλήξει τον Ηρακλή στην προσπάθειά του να φτάσει στην Λερναία Ύδρα.

7.5 Καταστροφή εχθρικών χαρακτήρων

Στις προηγούμενες ενότητες, παρουσιάστηκε ο τρόπος με τον οποίον δημιουργείται ένας ¨κόσμος¨ του παιχνιδιού. Έτσι, αφού έχουν εισαχθεί όλα τα στοιχεία που απαρτίζουν το παιχνίδι, το επόμενο βήμα είναι ο τρόπος με τον οποίο αλληλοεπιδρούν μεταξύ τους σύμφωνα και με τις κινήσεις που επιλέγει ο χρήστης για τον χαρακτήρα του. Το πιο σημαντικό κομμάτι, που αφορά την εξέλιξη του παιχνιδιού, είναι ο εντοπισμός και αναγνώριση των συγκρούσεων. Στην ενότητα 5.2.5 παρουσιάστηκε ο τρόπος με τον οποίο αναγνωρίζονται οι συγκρούσεις. Δεν είναι όμως όλες οι συγκρούσεις μεταξύ των αντικειμένων ίδιες, Για παράδειγμα, άλλη συμπεριφορά επιθυμούμε όταν συγκρούεται ένας εχθρικός χαρακτήρας σε έναν σωλήνα (pipe) και άλλη όταν συγκρούεται ένα εχθρικός χαρακτήρας με τον χαρακτήρα του Ηρακλή. Στην πρώτη περίπτωση, η επιθυμητή ενέργεια είναι η αλλαγή κατεύθυνσης του χαρακτήρα, ενώ στη δεύτερη πρέπει να εξετάσουμε περαιτέρω την σύγκρουση ώστε να εντοπίσουμε σε ποιον άξονα (x ή y) προκλήθηκε προκειμένου να αποφανθούμε ποιος από τους δύο χαρακτήρες νικήθηκε.

Έτσι, μία από τις ενέργειες που εκτελούνται στο Game Loop, είναι ο εντοπισμός και η εξέταση των συγκρούσεων του χαρακτήρα Ηρακλή με τους εχθρικούς χαρακτήρες. Η ενέργεια αυτή γίνεται τόσο κατά τον άξονα των x, όσο και κατά τον άξονα των y. Η παραπάνω ενέργειες γίνονται με την χρήστη των συναρτήσεων check_hercules_x_collisions() και check_hercules_y_collisions(). Κάθε μία από αυτές, εξετάζει τις συγκρούσεις του Ηρακλή με όλα τα υπόλοιπα στοιχεία του παιχνιδιού όπως, το έδαφος, οι σωλήνες (pipes), οι εχθρικοί χαρακτήρες κλπ και στην συνέχεια εξετάζει το είδος της σύγκρουσης εξάγοντας το αντίστοιχο αποτέλεσμα. Αν από τον παραπάνω έλεγχο προκύπτει ότι ένας εχθρικός χαρακτήρας έχει νικηθεί από τον Ηρακλή, τότε καταστρέφεται και χάνεται από το παιχνίδι. Αντίθετα, αν ο Ηρακλής είναι αυτός που νικήθηκε από κάποια σύγκρουση, τότε ο παίκτης χάνει μία ζωή και γίνεται επανέναρξη του παιχνιδιού.

8. ΕΠΙΛΟΓΟΣ

Στο παρόν κεφάλαιο γίνεται μία σύνοψη, παρουσιάζεται το τελικό αποτέλεσμα του παιχνιδιού ενώ τέλος προτείνονται τρόπο επέκτασης του ηλεκτρονικού παιχνιδιού που αναπτύχθηκε.

8.1 Σύνοψη

Η παρούσα Πτυχιακή Εργασία είχε ως αντικείμενο την δημιουργία ενός ηλεκτρονικού παιχνιδιού για μικρά παιδιά, με το οποίο μπορούν να παίξουν στο ξενοδοχείο όπου διαμένουν κατά τη διάρκεια των διακοπών τους. Όπως αναφέρθηκε και στα προηγούμενα κεφάλαια, στις δύσκολες εποχές που διανύουμε, με τον σύγχρονο τρόπο ζωής και τις συνεχείς υποχρεώσεις, η ανάγκη για διακοπές είναι επιτακτική για όλους. Κατά τη διάρκεια των διακοπών, οι άνθρωποι χαλαρώνουν, περνούν χρόνο με τον εαυτό τους και την οικογένειά τους, βιώνουν νέες εμπειρίες ενώ έρχονται σε επαφή με άλλους ανθρώπους διαφορετικού πολιτισμού και κουλτούρας. Η ψυχαγωγία είναι αναπόσπαστο κομμάτι των διακοπών και οι ξενοδοχειακές μονάδες επενδύουν όλο και περισσότερο σε αυτό το κομμάτι γνωρίζοντας ότι βελτιώνοντας τη εμπειρία του τουρίστα θα αυξήσουν τα κέρδη και θα νικήσουν τον ανταγωνισμό.

Τα ηλεκτρονικά παιχνίδια αποτελούν ένα σημαντικό κομμάτι της ψυχαγωγίας κατά την περίοδο των καλοκαιρινών διακοπών ενώ με φαντασία και δημιουργικότητα μπορούν να μετατραπούν σε σημαντικά μέσα μάθησης. Το ηλεκτρονικό παιχνίδι που αναπτύχθηκε, είναι προς αυτή την κατεύθυνση. Στόχος, ήταν τα μικρά παιδιά να έρθουν σε επαφή και να γνωρίσουν την ελληνική μυθολογία, τον ήρωα Ηρακλή και τους Άθλους του. Έτσι, ένα από τα πιο γνωστά και επιτυχημένα παιχνίδια, το Super Mario Bros, τροποποιήθηκε κατάλληλα ενώ επεκτάθηκε η λειτουργικότητά του για να εξυπηρετήσει τους σκοπούς της μάθησης και της ψυχαγωγίας. Μέσω αυτού, μικροί και μεγάλοι που μπαίνουν στον πειρασμό να παίξουν το παιχνίδι, μαθαίνουν για τους πιο ξακουστούς Άθλους του ήρωα Ηρακλή, όπως είναι ο Ταύρος της Κρήτης, το Λιοντάρι της Νεμέας και η Λερναία Ύδρα. Έτσι, φεύγοντας από την Ελλάδα, εκτός από τον ήλιο, τη θάλασσα και την ελληνική φιλοξενία παίρνουν μαζί τους και ένα κομμάτι της ελληνικής μυθολογίας, που ίσως για κάποιους από αυτούς να αποτελέσει το έναυσμα για να διδαχθούν την ελληνική ιστορία και τον ελληνικό πολιτισμό.

8.2 Παρουσίαση τελικού αποτελέσματος

Στα προηγούμενα κεφάλαια αναλύθηκαν τόσο η βιβλιοθήκη pygame της γλώσσας προγραμματισμού Python, που χρησιμοποιήθηκε για την ανάπτυξη του παιχνιδιού, όσο και ο τρόπος με τον οποίο αναπτύχθηκε το εν λόγο παιχνίδι προκειμένου να εξυπηρετεί τους σκοπούς της μάθησης και την ψυχαγωγίας. Το αποτέλεσμα είναι ένα ηλεκτρονικό παιχνίδι, το οποίο συνδυάζει εικόνες, animation, μουσική και ήχους προκειμένου να γίνει όσο το δυνατόν πιο ¨ζωντανό¨ και να κρατήσει το ενδιαφέρον του χρήστη.

Εικόνα 15: Εικόνα από το πρώτο επίπεδο της εφαρμογής(screenshot εφαρμογής)

Εικόνα 16: Εικόνα από το πρώτο επίπεδο της εφαρμογής(screenshot εφαρμογής)

Εικόνα 17: Εικόνα από το δεύτερο επίπεδο της εφαρμογής(screenshot εφαρμογής)

Εικόνα 18: Εικόνα από το τρίτο επίπεδο της εφαρμογής(screenshot εφαρμογής)

8.3 Επεκτάσεις

Στα πλαίσια της παρούσας Πτυχιακής Εργασίας αναπτύχθηκε, με βάση το παιχνίδι Super Mario Bros, ένα ηλεκτρονικό παιχνίδι με κεντρικό ήρωα τον μυθικό Ηρακλή. Το παιχνίδι απευθύνεται σε μικρά παιδιά και παρουσιάζει τρεις από τους πιο γνωστούς άθλους του Ηρακλή προκειμένου να έρθουν σε επαφή με εάν από τους πιο σημαντικούς ήρωες της ελληνικής μυθολογίας. Για το σκοπό αυτό αναπτύχθηκαν τρεις κόσμοι (level) κάθε ένας από τους οποίους ασχολείται με έναν άθλο. Το παρόν παιχνίδι μπορεί να επεκταθεί με την προσθήκη επιπλέον κόσμων, έτσι ώστε τα παιδιά που έρχονται σε επαφή μαζί του να γνωρίσουν μέσα από αυτό και τους δώδεκα άθλους του Ηρακλή και να έρθουν πιο κοντά στην ελληνική μυθολογία. Επιπλέον, σαν σημείο επέκτασης μπορεί να θεωρηθεί η βελτίωση των γραφικών και των κινήσεων τόσο του χαρακτήρα του Ηρακλή όσο και των εχθρικών χαρακτήρων.

ΒΙΒΛΙΟΓΡΑΦΙΑ

Λαγός, Δ. (2005). Τουριστική Οικονομική. Αθήνα: Εκδόσεις Κριτική

Βαρβαρέσος, Σ. (2000). Τουρισμός Έννοιες, Μεγέθη, Δομές. Η ελληνική Πραγματικότητα. 2η έκδοση. Αθήνα: Εκδόσεις Προπομπός

Ακριβός Χ, Σαλεσιώτης Μ (2007), Τουρισμός, Εισαγωγικές Έννοιες, Τουριστική Συνείδηση, Τουριστική Συμπεριφορά, Αθήνα: Εκδόσεις Interbooks

Παυλόπουλος, Π. (1999). Θέματα οικονομικής και τουριστικής πολιτικής. Αθήνα: Εκδόσεις: Ινστιτούτο Τουριστικών Ερευνών και Προβλέψεων(1999)

Μοίρα – Μυλωνοπούλου Π., Κοινωνιολογία και Ψυχολογία του Τουρισμού , Α' Εκδόση, Οργανισμός Εκδόσεως Διδακτικών Βιβλίων, Αθήνα 2003.

Λαλούμης, Δ. (1997). Ψυχαγωγία και άθληση πελατών ξενοδοχείων. Αθήνα: Εκδόσεις Έλλην.

Λαλούμης, Δ. (1999), Ξενοδοχειακή Ψυχαγωγία και Άθληση, Αθήνα: Εκδόσεις Σταμούλης.

Μάντζιος, Ν., & Γλυνιά, Ε. (2005). Η αγορά της άθλησης και ψυχαγωγίας (animation) σε ξενοδοχειακές μονάδες στην Ελλάδα: Προβλήματα και προοπτικές. Αναζητήσεις στην Φυσική Αγωγή και τον Αθλητισμό, 3 (1), 64 – 76

Dristas, M, (2003). "Tourism in Greece during the 20th century: A way to what sort of develppment?" , L. Tissot (ed), "Development of a tourist Industry in the 19th and 20th centuries: Technology, Politics and Economy, International Perspectives, NEUCHATEL, p.p. 187-210

www.egnatia.eu, Η σημασία του έργου ' ΕΓΝΑΤΙΑ ΟΔΟΣ'. Online. Διαθέσιμο:

https://www.ellinikifoni.gr/egnatiaodos.htm

visitgreece,gr, Ελληνικά νησιά. Online. Διαθέσιμο:

http://www.visitgreece.gr/el/greek_islands

Fraport Greece, Τα σχέδια της Fraport Greece για τη νέα εποχή των 14 αεροδρομίων. Online. Διαθέσιμο:

https://www.fraport-greece.com/ell/kentro-typou/nea/ta-sxedia-tis-fraport-greece-gia-ti-nea-epoxi-ton-14-aerodromon

TITAN Greece, «ΙΟΝΙΑ ΟΔΟΣ». Online. Διαθέσιμο:

https://www.titan.gr/el/proionta-kai-yphresies/endeiktika-erga/ergo/ionia-odos

ΑΤΤΙΚΟ ΜΕΤΡΟ Α.Ε, " Το βασικό έργο". Online. Διαθέσιμο:

https://www.ametro.gr/?page_id=112

Αττική Οδός, « ΑΤΤΙΚΗ ΟΔΟΣ». Online. Διαθέσιμο:

https://www.csweek.gr/member.html?id=86

TravelDailyNews, Τα καλύτερα αεροδρόμια στον κόσμο για το 2020. Online. Διαθέσιμο:

https://traveldailynews.gr/news/article/72324 11 Μαίου 2020

Καθ.Δρ.h.c.,Νίκος Γ. Ηγουμενάκης. " Τα οφέλη για τον τουρισμό από τη διοργάνωση των Ολυμπιακών αγώνων του 2004". Online. Διαθέσιμο:

http://pacific.jour.auth.gr/olympics2/Data/manuscripts/Tb1.pdf

Hunt, J (2019), Advanced Guide to Python 3 Programming. Switzerland: Εκδόσεις: Springer

Ίκκος, Ά. Η συμβολή του Τουρισμού στην ελληνική οικονομία το 2017. Online. Διαθέσιμο:

https://sete.gr/media/10888/2018_symvolhtourismou-2017.pdf 22 Απριλίου 2020

Παπαδημητράκης, Γ. Ανάπτυξη ενός νέου μοντέλου all inclusive για την ελληνική τουριστική βιομηχανία. Online. Διαθέσιμο:

http://www.taxhorizon.club 25 Απριλίου 2020

Μπενάκη, Β. Ελληνικό Σύστημα Στατιστικών Τουρισμού. Online. Διαθέσιμο:

https://www.statistics.gr/documents/20181/16797907/ELSTAT+visit_Tourism+statistics+in+Greece.pdf/ec9dea18-ce0a-a862-89c0-bbcc4509a1a8?t=1568718444000 3 Μαΐου 2020

Σύνδεσμος Ελληνικώ Τπυριστικών Επιχειρίσεων (ΣΕΤΕ). Online. Διαθέσιμο:

https://sete.gr/el/ 15 Απριλίου 2020

Παγκόσμιος Οργανισμός Τουρισμού. Online. Διαθέσιμο:

https://www.unwto.org/ 16 Απριλίου 2020

Εισαγωγή στην Ελληνική Μυθολογία – Ηρακλής. Online. Διαθέσιμο:

https://sites.google.com/site/hellasmythology/heroes/hercules 5 Απριλίου 2020

The Game Console. Online, Διαθέσιμο:

http://www.thegameconsole.com/sega.html 24 Απριλίου 2020

Επίσημη Σελίδα της γλώσσα προγραμματισμού Python. Online. Διαθέσιμο:

https://www.python.org/ 18 Μαρτίου 2020

Επίσημη Σελίδα της βιβλιοθήκης pygame. Online. Διαθέσιμο:

https://www.pygame.org/news 12 Μαρτίου 2020

Wentworth P, Elkner J, Downey A, Meyers C. PyGame. Online. Διαθέσιμο:

http://openbookproject.net/thinkcs/python/english3e/pygame.html 15 Μαρτίου 2020

Kids Can Code. Online. Διαθέσιμο:

https://kidscancode.org/blog/ 25 Μαΐου 2020

Video Game Music. Online. Διαθέσιμο:

https://downloads.khinsider.com/ 29 Απριλίου 2020

The spriters Resource. Online. Διαθέσιμο:

https://www.spriters-resource.com/ 1 Απριλίου 2020

Sachin K. Python Game Development: Build 11 Total Games. Online. Διαθέσιμο:

https://www.udemy.com/course/python-game-development 24 Απριλίου 2020

Κωδικός Python “ superHercules”

```
# Έναρξη παιχνιδιού
if __name__ == '__main__':
    main()
    pg.quit()
    sys.exit()
#Η κλάση αυτή αναφέρεται στα τούλβα bricks τα οποία μπορούν
# να καταστραφούν από τον Ηρακλή
class Brick(pg.sprite.Sprite):
    def __init__(self, x, y, contents=None, powerup_group=None,
name='brick'):
        # Αρχικοποίηση αντικειμένων
        pg.sprite.Sprite.__init__(self)
        self.sprite_sheet = setup.GFX['tile_set']

        self.frames = []
        self.frame_index = 0
        self.setup_frames()
        self.image = self.frames[self.frame_index]
        self.rect = self.image.get_rect()
        self.rect.x = x
        self.rect.y = y
        self.mask = pg.mask.from_surface(self.image)
        self.bumped_up = False
        self.rest_height = y
        self.state = c.RESTING
        self.y_vel = 0
        self.gravity = 1.2
        self.name = name
        self.contents = contents
        self.setup_contents()
        self.group = powerup_group
        self.powerup_in_box = True

    # Εξαγωγή εικόνων από το sprite sheet
    def get_image(self, x, y, width, height):
        image = pg.Surface([width, height]).convert()
        rect = image.get_rect()

        image.blit(self.sprite_sheet, (0, 0), (x, y, width, height))
        image.set_colorkey(c.BLACK)
        image =
pg.transform.scale(image,(int(rect.width*c.BRICK_SIZE_MULTIPLIER),int(rect.
height*c.BRICK_SIZE_MULTIPLIER)))
        return image

    # εισαγωγή των frames σε μία λίστα
    def setup_frames(self):
        self.frames.append(self.get_image(16, 0, 16, 16))
        self.frames.append(self.get_image(432, 0, 16, 16))

    # Εισάγει 6 coins σε ένα brick αν το contents είναι ίσο με 6coins
```

```
    def setup_contents(self):
        if self.contents == '6coins':
            self.coin_total = 6
        else:
            self.coin_total = 0

    # update τα bricks
    def update(self):
        self.handle_states()

    # Καθορίζει τη συμπεριφορά του brick με βάση την κατάστασή του
    def handle_states(self):
        if self.state == c.RESTING:
            self.resting()
        elif self.state == c.BUMPED:
            self.bumped()
        elif self.state == c.OPENED:
            self.opened()

    # Ανοίγει το brick όταν ο Ηρακλής πάρει και και 6 coins
    def resting(self):
        if self.contents == '6coins':
            if self.coin_total == 0:
                self.state == c.OPENED

    # Μετακίνηση των bricks όταν ο Ηρακλής τα χτυπάει
    def bumped(self):
        self.rect.y += self.y_vel
        self.y_vel += self.gravity

        if self.rect.y >= (self.rest_height + 5):
            self.rect.y = self.rest_height
            if self.contents == 'star':
                self.state = c.OPENED
            elif self.contents == '6coins':
                if self.coin_total == 0:
                    self.state = c.OPENED
                else:
                    self.state = c.RESTING
            else:
                self.state = c.RESTING


    def start_bump(self, score_group):
        self.y_vel = -6

        if self.contents == '6coins':
            setup.SFX['coin'].play()

            if self.coin_total > 0:
                self.group.add(coin.Coin(self.rect.centerx, self.rect.y,
score_group))
                self.coin_total -= 1
                if self.coin_total == 0:
                    self.frame_index = 1
                    self.image = self.frames[self.frame_index]
        elif self.contents == 'star':
            setup.SFX['powerup_appears'].play()
            self.frame_index = 1
            self.image = self.frames[self.frame_index]
```

```
        self.state = c.BUMPED

    # Το brick είναι ανοιχτό
    def opened(self):
        self.frame_index = 1
        self.image = self.frames[self.frame_index]

        if self.contents == 'star' and self.powerup_in_box:
            self.group.add(powerups.Star(self.rect.centerx,
self.rest_height))
            self.powerup_in_box = False


# Η κλαση αντιπροσωπεύει τα κομμάτια των bricks όταν αυτά σπάνε
class BrickPiece(pg.sprite.Sprite):
    def __init__(self, x, y, xvel, yvel):
        super(BrickPiece, self).__init__()
        self.sprite_sheet = setup.GFX['item_objects']
        self.setup_frames()
        self.frame_index = 0
        self.image = self.frames[self.frame_index]
        self.rect = self.image.get_rect()
        self.rect.x = x
        self.rect.y = y
        self.x_vel = xvel
        self.y_vel = yvel
        self.gravity = .8

    # Δημιουργία λίστας με τις εικόνες (frames)
    def setup_frames(self):
        self.frames = []

        image = self.get_image(68, 20, 8, 8)
        reversed_image = pg.transform.flip(image, True, False)

        self.frames.append(image)
        self.frames.append(reversed_image)

    # Εξαγωγή της εικόνας από το sprite sheet
    def get_image(self, x, y, width, height):
        image = pg.Surface([width, height]).convert()
        rect = image.get_rect()

        image.blit(self.sprite_sheet, (0, 0), (x, y, width, height))
        image.set_colorkey(c.BLACK)
        image =
pg.transform.scale(image,(int(rect.width*c.BRICK_SIZE_MULTIPLIER),int(rect.
height*c.BRICK_SIZE_MULTIPLIER)))
        return image

    # Κάνει update τα κομμάτια των bricks
    def update(self):
        self.rect.x += self.x_vel
        self.rect.y += self.y_vel
        self.y_vel += self.gravity
        self.check_if_off_screen()

    # Απομακρύνει τα κομμάτια από την οθόνη
    def check_if_off_screen(self):
        if self.rect.y > c.SCREEN_HEIGHT:
            self.kill()
```

```
# Η κλάση αυτή αναπαριστά την σημαία στο κάστρο
class Flag(pg.sprite.Sprite):
    def __init__(self, x, y):
        # Αρχικοποίηση αντικειμένου
        super(Flag, self).__init__()
        self.sprite_sheet = setup.GFX['item_objects']
        self.image = self.get_image(129, 2, 14, 14)
        self.rect = self.image.get_rect()
        self.rect.x = x
        self.rect.y = y
        self.state = 'rising'
        self.y_vel = -2
        self.target_height = y

    # Εξαγωγή εικόνας από τον sprite sheet
    def get_image(self, x, y, width, height):

        image = pg.Surface([width, height])
        rect = image.get_rect()

        image.blit(self.sprite_sheet, (0, 0), (x, y, width, height))
        image.set_colorkey(c.BLACK)
        image =
pg.transform.scale(image,(int(rect.width*c.SIZE_MULTIPLIER),int(rect.height
*c.SIZE_MULTIPLIER)))
        return image

    # Update τη θέση της σημαίας
    def update(self, *args):
        if self.state == 'rising':
            self.rising()
        elif self.state == 'resting':
            self.resting()

    # Κατέβασμα της σημαίας από τον Ηρακλη
    def rising(self):
        self.rect.y += self.y_vel
        if self.rect.bottom <= self.target_height:
            self.state = 'resting'

    def resting(self):
        pass
# Η κλάση αυτή σχετίζεται με τα checkpoints. Τα checkpoints είναι σημεία
πάνω στην background εικόνα
# τα οποία χρησιμοποιούνται ως trigger για τη δημιουργία εχθρών κλπ
class Checkpoint(pg.sprite.Sprite):
    def __init__(self, x, name, y=0, width=10, height=600):
        super(Checkpoint, self).__init__()
        self.image = pg.Surface((width, height))
        self.image.fill(c.BLACK)
        self.rect = self.image.get_rect()
        self.rect.x = x
        self.rect.y = y
        self.name = name
# Η κλάση αυτή αναπαριστά τα coins (νομίσματα)  τα οποία είναι μέσα σε
τούβλα
class Coin(pg.sprite.Sprite):
    # Αρχικοποίηση αντικειμένου
    def __init__(self, x, y, score_group):
        pg.sprite.Sprite.__init__(self)
        self.sprite_sheet = setup.GFX['item_objects']
```

```
        self.frames = []
        self.frame_index = 0
        self.animation_timer = 0
        self.state = c.SPIN
        self.setup_frames()
        self.image = self.frames[self.frame_index]
        self.rect = self.image.get_rect()
        self.rect.centerx = x
        self.rect.bottom = y - 5
        self.gravity = 1
        self.y_vel = -15
        self.initial_height = self.rect.bottom - 5
        self.score_group = score_group

    # Η μέθοδος εξάγει τις εικόνες από το sprite sheet που ορίζεται κατά
την αρχικοποίηση
    def get_image(self, x, y, width, height):
        image = pg.Surface([width, height]).convert()
        rect = image.get_rect()

        image.blit(self.sprite_sheet, (0, 0), (x, y, width, height))
        image.set_colorkey(c.BLACK)


        image = pg.transform.scale(image,
(int(rect.width*c.SIZE_MULTIPLIER),int(rect.height*c.SIZE_MULTIPLIER)))
        return image

    # Δημιουργία λίστας με τις εικόνες (frames)
    def setup_frames(self):
        self.frames.append(self.get_image(52, 113, 8, 14))
        self.frames.append(self.get_image(4, 113, 8, 14))
        self.frames.append(self.get_image(20, 113, 8, 14))
        self.frames.append(self.get_image(36, 113, 8, 14))

    # Update της κατάστασης των coins
    def update(self, game_info, viewport):
        self.current_time = game_info[c.CURRENT_TIME]
        self.viewport = viewport
        if self.state == c.SPIN:
            self.spinning()

    # Η μέθοδος αυτή εκτελείται όταν το coin είναι σε κατάσταση spinning
και εναλλάσει τις
    #εικόνες ώστε να δίνει την ψευδαίσθηση της κίνησης
    def spinning(self):
        self.image = self.frames[self.frame_index]
        self.rect.y += self.y_vel
        self.y_vel += self.gravity

        if (self.current_time - self.animation_timer) > 80:
            if self.frame_index < 3:
                self.frame_index += 1
            else:
                self.frame_index = 0

            self.animation_timer = self.current_time

        if self.rect.bottom > self.initial_height:
            self.kill()
            self.score_group.append(score.Score(self.rect.centerx -
```

```
self.viewport.x, self.rect.y, 200))


# Η κλάση αυτή αντιπροσωπευει τα coins τα οποία στέκονται εκτός των bricks
class StaticCoin(pg.sprite.Sprite):
    # Αρχικοποίηση
    def __init__(self, x, y):
        pg.sprite.Sprite.__init__(self)
        self.frame_index = 0
        self.frames = []
        self.load_frames()
        self.image = self.frames[self.frame_index]
        self.rect = self.image.get_rect()
        self.rect.x = x
        self.rect.y = y
        self.animation_timer = 0

    # Εισαγωγή των εικόνων (frames) σε μία λίστα
    def load_frames(self):
        self.sprite_sheet = setup.GFX["item_objects"]
        self.frames.append(self.get_image(3, 98, 9, 13))
        self.frames.append(self.get_image(19, 98, 9, 13))
        self.frames.append(self.get_image(35, 98, 9, 13))
        self.frames.append(self.get_image(51, 98, 9, 13))

    # Εξαγωγή των εικόνων από το sprite sheet
    def get_image(self, x, y, width, height):
        image = pg.Surface([width, height]).convert()
        rect = image.get_rect()

        image.blit(self.sprite_sheet, (0, 0), (x, y, width, height))
        image.set_colorkey(c.BLACK)

        image =
pg.transform.scale(image,(int(rect.width*c.BRICK_SIZE_MULTIPLIER),int(rect.
height*c.BRICK_SIZE_MULTIPLIER)))
        return image

    # Update στην κατάσταση των coins και εναλλαγή των εικόνων ώστε να
δίνει την ψευδαίσθηση
    # της κίνησης
    def update(self, game_info):
        self.current_time = game_info[c.CURRENT_TIME]

        time_list = [375, 125, 125, 125]
        if self.animation_timer == 0:
            self.animation_timer = self.current_time
        elif (self.current_time - self.animation_timer) >
time_list[self.frame_index]:
            self.frame_index += 1
            if self.frame_index == 4:
                self.frame_index = 0
            self.animation_timer = self.current_time

        self.image = self.frames[self.frame_index]
# Η κλάση αυτή αναπαριστά το box που περιέχει ένα coin
class Coin_box(pg.sprite.Sprite):
    # Αρχικοποίηση
    def __init__(self, x, y, contents='coin', group=None):
        pg.sprite.Sprite.__init__(self)
        self.sprite_sheet = setup.GFX['tile_set']
```

```
        self.frames = []
        self.setup_frames()
        self.frame_index = 0
        self.image = self.frames[self.frame_index]
        self.rect = self.image.get_rect()
        self.rect.x = x
        self.rect.y = y
        self.mask = pg.mask.from_surface(self.image)
        self.animation_timer = 0
        self.first_half = True
        self.state = c.RESTING
        self.rest_height = y
        self.gravity = 1.2
        self.y_vel = 0
        self.contents = contents
        self.group = group

    # Εξαγωγή εικόνων από το sprite sheet
    def get_image(self, x, y, width, height):
        image = pg.Surface([width, height]).convert()
        rect = image.get_rect()

        image.blit(self.sprite_sheet, (0, 0), (x, y, width, height))
        image.set_colorkey(c.BLACK)

        image =
pg.transform.scale(image,(int(rect.width*c.BRICK_SIZE_MULTIPLIER),int(rect.
height*c.BRICK_SIZE_MULTIPLIER)))
        return image

    # Εισαγωγή των εικόνων (frames) σε λίστα
    def setup_frames(self):
        self.frames.append(
            self.get_image(384, 0, 16, 16))
        self.frames.append(
            self.get_image(400, 0, 16, 16))
        self.frames.append(
            self.get_image(416, 0, 16, 16))
        self.frames.append(
            self.get_image(432, 0, 16, 16))

    # update της κατάστασης των coins με τη χρήση της μεθόδου handle_states
    def update(self, game_info):
        self.current_time = game_info[c.CURRENT_TIME]
        self.handle_states()

    # Εκτέλεση της ενέργειας με βάση την κατάσταση στην οποία βρίσκεται το
coin
    def handle_states(self):
        if self.state == c.RESTING:
            self.resting()
        elif self.state == c.BUMPED:
            self.bumped()
        elif self.state == c.OPENED:
            self.opened()

    # Εναλλαγή των εικόνων όταν η κατάσταση του coin είναι resting
    def resting(self):
        if self.first_half:
            if self.frame_index == 0:
                if (self.current_time - self.animation_timer) > 375:
```

```
                    self.frame_index += 1
                    self.animation_timer = self.current_time
            elif self.frame_index < 2:
                if (self.current_time - self.animation_timer) > 125:
                    self.frame_index += 1
                    self.animation_timer = self.current_time
            elif self.frame_index == 2:
                if (self.current_time - self.animation_timer) > 125:
                    self.frame_index -= 1
                    self.first_half = False
                    self.animation_timer = self.current_time
        else:
            if self.frame_index == 1:
                if (self.current_time - self.animation_timer) > 125:
                    self.frame_index -= 1
                    self.first_half = True
                    self.animation_timer = self.current_time

        self.image = self.frames[self.frame_index]

    # Εκτελείται όταν ο Ηρακλής "ανοίξει¨ ένα coin box
    def bumped(self):

        self.rect.y += self.y_vel
        self.y_vel += self.gravity

        if self.rect.y > self.rest_height + 5:
            self.rect.y = self.rest_height
            self.state = c.OPENED
            if self.contents == 'mushroom':
                self.group.add(powerups.Milk(self.rect.centerx,
self.rect.y))
            elif self.contents == 'fireflower':
                self.group.add(powerups.FireApple(self.rect.centerx,
self.rect.y))
            elif self.contents == '1up_mushroom':
                self.group.add(powerups.LifeMilk(self.rect.centerx,
self.rect.y))


        self.frame_index = 3
        self.image = self.frames[self.frame_index]

    # Ενέργειες που εκτελούνται όταν ο Ηρακλής χτυπάει τα coins
    def start_bump(self, score_group):
        self.y_vel = -6
        self.state = c.BUMPED

        if self.contents == 'coin':
            self.group.add(coin.Coin(self.rect.centerx,
self.rect.y,score_group))
            setup.SFX['coin'].play()
        else:
            setup.SFX['powerup_appears'].play()


    def opened(self):
        pass
# Η κλάση αυτή σχετίζεται με τους Colliders τα οποία μας βοηθούν να
αναγνωρίζουμε τις
# συγκρούσεις μετακύ των αντικειμένων
```

```
class Collider(pg.sprite.Sprite):
    def __init__(self, x, y, width, height, name='collider'):
        pg.sprite.Sprite.__init__(self)
        self.image = pg.Surface((width, height)).convert()
        #self.image.fill(c.RED)
        self.rect = self.image.get_rect()
        self.rect.x = x
        self.rect.y = y
        self.state = None
import pygame as pg
from .. import setup
from .. import constants as c
import math


# Υπερκλάση για όλους τους enemies
class Enemy(pg.sprite.Sprite):
    def __init__(self):
        pg.sprite.Sprite.__init__(self)

    # Αρχικοποίηση μεταβλητών για τους enemies
    def setup_enemy(self, x, y, direction, name, setup_frames, sheet):
        self.sprite_sheet = setup.GFX[sheet]
        self.frames = []
        self.frame_index = 0
        self.animate_timer = 0
        self.death_timer = 0
        self.gravity = 1.5
        self.state = c.WALK

        self.name = name
        self.direction = direction
        setup_frames()

        self.image = self.frames[self.frame_index]
        self.rect = self.image.get_rect()
        self.rect.x = x
        self.rect.bottom = y
        self.set_velocity()

    # Καθορίζει την ταχύτητα των εχθρών με βάση την κατεύθυνσή τους
    def set_velocity(self):
        if self.direction == c.LEFT:
            self.x_vel = -2
        else:
            self.x_vel = 2

        self.y_vel = 0

    # Εξάγει τα frames από το sprite sheet
    def get_image(self, x, y, width, height, color_key, size):
        image = pg.Surface([width, height]).convert()
        rect = image.get_rect()
        image.blit(self.sprite_sheet, (0, 0), (x, y, width, height))
        image.set_colorkey(color_key)
        image = pg.transform.scale(image, (int(rect.width * size),
int(rect.height * size)))
        return image

    # Συμπεριφορά των εχθρών ανάλογα με την κατάσταση στην οποία βρίσκονται
    def handle_state(self):
```

```
        if self.state == c.WALK or self.state == c.FLY:
            self.walking()
        elif self.state == c.FALL:
            self.falling()
        elif self.state == c.JUMPED_ON:
            self.jumped_on()
        elif self.state == c.SHELL_SLIDE:
            self.shell_sliding()
        elif self.state == c.DEATH_JUMP:
            self.death_jumping()

    # Προκαθορισμένη κατάσταση στην οποία τα enemies περπατάνε
    def walking(self):
        # Κάθε 125ms γίνεται εναλλαγή των εικόνων
        if (self.current_time - self.animate_timer) > 125:
            if self.frame_index == 0:
                self.frame_index += 1
            elif self.frame_index == 1:
                self.frame_index = 0
            # ενημέρωση animate_timer
            self.animate_timer = self.current_time

    # Πτώση
    def falling(self):
        if self.y_vel < 10:
            self.y_vel += self.gravity

    def jumped_on(self):
        """Placeholder for when the enemy is stomped on"""
        pass

    # Animation όταν ο εχθρός πεθαίνει
    def death_jumping(self):
        self.rect.y += self.y_vel
        self.rect.x += self.x_vel
        self.y_vel += self.gravity

        if self.rect.y > 600:
            self.kill()

    # Εναλλαγή του enemy σε κατάσταση DEATH JUMP με αλλαγή frame
    def start_death_jump(self, direction):
        self.y_vel = -8
        if direction == c.RIGHT:
            self.x_vel = 2
        else:
            self.x_vel = -2
        self.gravity = .5
        self.frame_index = 3
        self.image = self.frames[self.frame_index]
        self.state = c.DEATH_JUMP

    # Animation εναλλαγή μεταξύ των εικόνων
    def animation(self):
        self.image = self.frames[self.frame_index]

    # Update των enemies
    def update(self, game_info, *args):
        self.current_time = game_info[c.CURRENT_TIME]
        self.handle_state()
        self.animation()
```

```
class Lion(Enemy):
    def __init__(self, y=c.GROUND_HEIGHT, x=0, direction=c.LEFT,
name='lion'):
        super().__init__()
        self.setup_enemy(x, y, direction, name, self.setup_frames,
"FireLion")

    # Εισάγουμε στην λίστα των frames τις εικόνες για το animation
    def setup_frames(self):
        # Λιοντάρι σταθερό left direction [0]
        self.frames.append(
            self.get_image(288, 191, 64, 61, (10, 114, 61), 0.8))
        # Λιοντάρι κάνει βήμα left direction [1]
        self.frames.append(
            self.get_image(208, 179, 60, 73, (10, 114, 61), 0.8))
        # Λιοντάρι λιώμα [2]
        self.frames.append(
            self.get_image(731, 292, 63, 62, (10, 114, 61), 0.8))
        # Λιοντάρι σταθερό right direction [3]
        self.frames.append(pg.transform.flip(self.frames[0], True, False))
        # Λιοντάρι κάνει βήμα right direction [4]
        self.frames.append(pg.transform.flip(self.frames[1], True, False))

    # Εκτελείται όταν ο εχθρός περπάει (state == WALK) και χρησιμοποιείται
για να εναλλάσει
    # τα frame δημιουργώντας το animation
    def walking(self):
        if (self.current_time - self.animate_timer) > 125:
            if self.direction == c.RIGHT:
                if self.frame_index == 0 or self.frame_index == 1:
                    self.frame_index = 3
                elif self.frame_index == 3:
                    self.frame_index += 1
                elif self.frame_index == 4:
                    self.frame_index = 3

            else:
                if self.frame_index == 3 or self.frame_index == 4:
                    self.frame_index = 0
                elif self.frame_index == 0:
                    self.frame_index += 1
                elif self.frame_index == 1:
                    self.frame_index = 0

            self.animate_timer = self.current_time

    # Εκτελείται όταν ο Ηρακλής πλακώνει το λιοντάρι από πάνω
    def jumped_on(self):
        self.frame_index = 2
        # μετά από 1 sec το συγκεκριμένο αντικείμενο εξαφανίζεται από την
οθόνη
        if (self.current_time - self.death_timer) > 1000:
            self.kill()

class Bull(Enemy):
    def __init__(self, y=c.GROUND_HEIGHT, x=0, direction=c.LEFT,
name='bull'):
        super().__init__()
        self.setup_enemy(x, y, direction, name, self.setup_frames, "bull")
```

```
    # Εισάγουμε στην λίστα των frames τις εικόνες για το animation
    def setup_frames(self):
        # Ταύρος right direction [0]
        self.frames.append(self.get_image(256, 21, 136, 75, c.LIGHT_GREEN,
c.BULL_SIZE_MULTIPLIER))
        # Ταύρος right direction [1]
        self.frames.append(self.get_image(396, 24, 138, 81, c.LIGHT_GREEN,
c.BULL_SIZE_MULTIPLIER))
        # Ταύρος right direction [2]
        self.frames.append(self.get_image(537, 25, 128, 82, c.LIGHT_GREEN,
c.BULL_SIZE_MULTIPLIER))
        # Ταύρος right direction [3]
        self.frames.append(self.get_image(669, 21, 128, 83, c.LIGHT_GREEN,
c.BULL_SIZE_MULTIPLIER))
        # Ταύρος νεκρός right direction [4]
        self.frames.append(self.get_image(625, 172, 137, 51, c.LIGHT_GREEN,
c.BULL_SIZE_MULTIPLIER))
        # Ταύρος left direction [5]
        self.frames.append(pg.transform.flip(self.frames[0], True, False))
        # Ταύρος left direction [6]
        self.frames.append(pg.transform.flip(self.frames[1], True, False))
        # Ταύρος left direction [7]
        self.frames.append(pg.transform.flip(self.frames[2], True, False))
        # Ταύρος left direction [8]
        self.frames.append(pg.transform.flip(self.frames[3], True, False))
        # Ταύρος νεκρός left direction [9]
        self.frames.append(pg.transform.flip(self.frames[4], True, False))

    # Εκτελείται όταν ο εχθρός περπάει (state == WALK) και χρησιμοποιείται
για να εναλλάσει
    # τα frame δημιουργώντας το animation
    def walking(self):
        if (self.current_time - self.animate_timer) > 125:
            if self.direction == c.RIGHT:
                if self.frame_index >= 3:
                    self.frame_index = 0
                self.frame_index += 1
            else:
                if self.frame_index <= 5 or self.frame_index == 8:
                    self.frame_index = 5
                self.frame_index += 1

            self.animate_timer = self.current_time

    # Εκτελείται όταν ο Ηρακλής πλακώνει το λιοντάρι από πάνω
    def jumped_on(self):
        if self.direction == c.RIGHT:
            self.frame_index = 4
        else:
            self.frame_index = 9
        # μετά από 1 sec το συγκεκριμένο αντικείμενο εξαφανίζεται από την
οθόνη
        if (self.current_time - self.death_timer) > 1000:
            self.kill()

# Η κλάση σχετίζεται με την λερναία ύδρα
class Hydra(Enemy):
    def __init__(self,level, start_position, end_position,
y=c.GROUND_HEIGHT, x=0, direction=c.LEFT, name=c.HYDRA):
        Enemy.__init__(self)
```

```
        self.setup_enemy(x, y, direction, name, self.setup_frames, "hydra")
        self.x_vel = 0
        self.level = level
        self.start_position = start_position
        self.end_position = end_position
        self.gravity = 0.3
        self.fire_timer = 0
        self.jump_timer = 0
        self.lives = 10

    def setup_frames(self):
        # Προανατολισμός frame προς τα δεξιά
        self.frames.append(self.get_image(5, 303, 72, 86, c.BLACK,
c.HYDRA_MULTIPLIER))
        self.frames.append(self.get_image(145, 303, 72, 86, c.BLACK,
c.HYDRA_MULTIPLIER))
        self.frames.append(self.get_image(290, 301, 66, 88, c.BLACK,
c.HYDRA_MULTIPLIER))
        self.frames.append(self.get_image(431, 303, 72, 84, c.BLACK,
c.HYDRA_MULTIPLIER))

        # Προανατολισμός frame προς τα αριστερά
        self.frames.append(pg.transform.flip(self.frames[0], True, False))
        self.frames.append(pg.transform.flip(self.frames[1], True, False))
        self.frames.append(pg.transform.flip(self.frames[2], True, False))
        self.frames.append(pg.transform.flip(self.frames[3], True, False))


    def walking(self):
        if (self.current_time - self.animate_timer) > 250:
            if self.direction == c.RIGHT:
                self.frame_index = 0
                self.frame_index += 1
                if self.frame_index >= 3:
                    self.frame_index = 0
            else:
                self.frame_index = 4
                self.frame_index += 1
                if self.frame_index >= 7:
                    self.frame_index = 4
            self.animate_timer = self.current_time

        self.shoot_fire()
        if self.should_jump():
            self.y_vel = -7

    def falling(self):
        if self.y_vel < 7:
            self.y_vel += self.gravity
        self.shoot_fire()

    def should_jump(self):
        if (self.current_time - self.jump_timer) > 2500:
            self.jump_timer = self.current_time
            return True
        return False

    def shoot_fire(self):
        if (self.current_time - self.fire_timer) > 3000:
            self.fire_timer = self.current_time
            self.level.enemy_group.add(Fire(self.rect.x, self.rect.bottom-
```

```
20, self.direction))


class Fire(Enemy):

    def __init__(self, x, y, direction, name=c.FIRE):
        Enemy.__init__(self)
        self.setup_enemy(x, y, direction, name, self.setup_frames,
"smb_enemies_sheet")

        self.state = c.FLY
        self.x_vel = 5 if self.direction == c.RIGHT else -5

    def setup_frames(self):
        self.frames.append(self.get_image(101, 253, 23, 8, c.BLACK,
c.SIZE_MULTIPLIER))
        self.frames.append(self.get_image(131, 253, 23, 8, c.BLACK,
c.SIZE_MULTIPLIER))
        # right images
        self.frames.append(pg.transform.flip(self.frames[0], True, False))
        self.frames.append(pg.transform.flip(self.frames[1], True, False))

    def check_x_collisions(self, level):
        sprite_group = pg.sprite.Group(
                                        level.brick_group, level.box_group)
        sprite = pg.sprite.spritecollideany(self, sprite_group)
        if sprite:
            self.kill()

    def start_death_jump(self, direction):
        self.kill()


# Η κλάση αφορά το firestick το οποίο περιστρέφεται στην τελευταία πίστα.
class FireStick(pg.sprite.Sprite):
    def __init__(self, center_x, center_y, radius, name=c.FIRESTICK):

        pg.sprite.Sprite.__init__(self)

        self.frames = []
        self.frame_index = 0
        self.animate_timer = 0
        self.name = name
        self.sprite_sheet = setup.GFX["item_objects"]
        self.load_frames()
        self.animate_timer = 0
        self.image = self.frames[self.frame_index]
        self.rect = self.image.get_rect()
        self.rect.x = center_x - radius
        self.rect.y = center_y
        self.center_x = center_x
        self.center_y = center_y
        self.radius = radius
        self.angle = 0

    def load_frames(self):
        self.frames.append(self.get_image(96, 144, 8, 8))
        self.frames.append(self.get_image(104, 144, 8, 8))
        self.frames.append(self.get_image(96, 152, 8, 8))
        self.frames.append(self.get_image(104, 152, 8, 8))
```

```
    # Εξάγει τα frames από το sprite sheet
    def get_image(self, x, y, width, height):
        image = pg.Surface([width, height]).convert()
        rect = image.get_rect()
        image.blit(self.sprite_sheet, (0, 0), (x, y, width, height))
        image.set_colorkey(c.BLACK)
        image = pg.transform.scale(image, (int(rect.width *
c.BRICK_SIZE_MULTIPLIER),
                                           int(rect.height *
c.BRICK_SIZE_MULTIPLIER)))
        return image

    def update(self, game_info):
        self.current_time = game_info[c.CURRENT_TIME]
        if (self.current_time - self.animate_timer) > 100:
            if self.frame_index < 3:
                self.frame_index += 1
            else:
                self.frame_index = 0
            self.animate_timer = self.current_time
        self.image = self.frames[self.frame_index]

        self.angle += 1
        if self.angle == 360:
            self.angle = 0
        radian = math.radians(self.angle)
        self.rect.x = self.center_x + math.sin(radian) * self.radius
        self.rect.y = self.center_y + math.cos(radian) * self.radius


# class Lion(Enemy):
#     def __init__(self, y=c.GROUND_HEIGHT, x=0, direction=c.LEFT,
name='lion'):
#         super().__init__()
#         self.setup_enemy(x, y, direction, name, self.setup_frames,
"lion")
#
#     # Εισάγουμε στην λίστα των frames τις εικόνες για το animation
#     def setup_frames(self):
#         # Λιοντάρι σταθερό left direction [0]
#         self.frames.append(
#             self.get_image(30, 60, 14, 16, c.WHITE, c.SIZE_MULTIPLIER))
#         # Λιοντάρι κάνει βήμα left direction [1]
#         self.frames.append(
#             self.get_image(30, 90, 15, 16, c.WHITE, c.SIZE_MULTIPLIER))
#         # Λιοντάρι λιώμα [2]
#         self.frames.append(
#             self.get_image(240, 242, 16, 11, c.WHITE, c.SIZE_MULTIPLIER))
#         # Λιοντάρι σταθερό right direction [3]
#         self.frames.append(pg.transform.flip(self.frames[0], True,
False))
#         # Λιοντάρι κάνει βήμα right direction [4]
#         self.frames.append(pg.transform.flip(self.frames[1], True,
False))
#
#     # Εκτελείται όταν ο εχθρός περπάει (state == WALK) και
χρησιμοποιείται για να εναλλάσει
#     # τα frame δημιουργώντας το animation
#     def walking(self):
#         if (self.current_time - self.animate_timer) > 125:
#             if self.direction == c.RIGHT:
```

```
#                 if self.frame_index == 0 or self.frame_index == 1:
#                     self.frame_index = 3
#                 elif self.frame_index == 3:
#                     self.frame_index += 1
#                 elif self.frame_index == 4:
#                     self.frame_index = 3
#
#             else:
#                 if self.frame_index == 3 or self.frame_index == 4:
#                     self.frame_index = 0
#                 elif self.frame_index == 0:
#                     self.frame_index += 1
#                 elif self.frame_index == 1:
#                     self.frame_index = 0
#
#             self.animate_timer = self.current_time
#
#     # Εκτελείται όταν ο Ηρακλής πλακώνει το λιοντάρι από πάνω
#     def jumped_on(self):
#         self.frame_index = 2
#         # μετά από 1 sec το συγκεκριμένο αντικείμενο εξαφανίζεται από την
οθόνη
#         if (self.current_time - self.death_timer) > 1000:
#             self.kill()
import pygame as pg
from .. import setup
from .. import constants as c

# Η κλάση αυτή αντιπροσωπεύει την σημαία στην κορυφή του ιστού
class Flag(pg.sprite.Sprite):
    # Αρχικοποίηση
    def __init__(self, x, y):
        super(Flag, self).__init__()
        self.sprite_sheet = setup.GFX['item_objects']
        self.setup_images()
        self.image = self.frames[0]
        self.rect = self.image.get_rect()
        self.rect.right = x
        self.rect.y = y
        self.state = c.TOP_OF_POLE

    # Προσθήκη εικόνας (frame) στη λίστα
    def setup_images(self):
        self.frames = []

        self.frames.append(self.get_image(128, 32, 16, 16))

    # Εξάγει τη σημαία από τη sprite sheet
    def get_image(self, x, y, width, height):
        image = pg.Surface([width, height])
        rect = image.get_rect()

        image.blit(self.sprite_sheet, (0, 0), (x, y, width, height))
        image.set_colorkey(c.BLACK)
        image = pg.transform.scale(image,
(int(rect.width*c.BRICK_SIZE_MULTIPLIER),int(rect.height*c.BRICK_SIZE_MULTI
PLIER)))
        return image

    # Κάνει update την κατάσταση της σημαίας
    def update(self, *args):
```

```
        self.handle_state()

    # Καθορίζουν την συμπεριφορά της σημαίας με βάση την κατάσταση της
σημαίας
    def handle_state(self):
        if self.state == c.TOP_OF_POLE:
            self.image = self.frames[0]
        elif self.state == c.SLIDE_DOWN:
            self.sliding_down()
        elif self.state == c.BOTTOM_OF_POLE:
            self.image = self.frames[0]

    # Ο Ηρακλής κατεβάζει τη σημαία στο τέλος του level
    def sliding_down(self):
        self.y_vel = 5
        self.rect.y += self.y_vel

        if self.rect.bottom >= 485:
            self.state = c.BOTTOM_OF_POLE

# Η κλάση αυτή αντιπροσωπεύει τον ιστό της σημαίας
class Pole(pg.sprite.Sprite):
    def __init__(self, x, y):
        super(Pole, self).__init__()
        self.sprite_sheet = setup.GFX['tile_set']
        self.setup_frames()
        self.image = self.frames[0]
        self.rect = self.image.get_rect()
        self.rect.x = x
        self.rect.y = y

    # Εισαγωγή εικόνας (frames) στη λίστα
    def setup_frames(self):
        self.frames = []

        self.frames.append(self.get_image(263, 144, 2, 16))

    # Εξάγει την εικόνα από το sprite sheet
    def get_image(self, x, y, width, height):
        image = pg.Surface([width, height])
        rect = image.get_rect()

        image.blit(self.sprite_sheet, (0, 0), (x, y, width, height))
        image.set_colorkey(c.BLACK)
        image =
pg.transform.scale(image,(int(rect.width*c.BRICK_SIZE_MULTIPLIER),int(rect.
height*c.BRICK_SIZE_MULTIPLIER)))
        return image


    def update(self, *args):
        pass

# Η κλάση αυτή αντιπροσωπεύει το τέλος του ιστού
class Finial(pg.sprite.Sprite):
    def __init__(self, x, y):
        super(Finial, self).__init__()
        self.sprite_sheet = setup.GFX['tile_set']
        self.setup_frames()
        self.image = self.frames[0]
        self.rect = self.image.get_rect()
```

```
        self.rect.centerx = x
        self.rect.bottom = y

    # Εισαγωγή εικόνας (frame) στη λίστα
    def setup_frames(self):
        self.frames = []

        self.frames.append( self.get_image(228, 120, 8, 8))

    # Εξάγει την εικόνα απο το sprite sheet
    def get_image(self, x, y, width, height):
        image = pg.Surface([width, height])
        rect = image.get_rect()

        image.blit(self.sprite_sheet, (0, 0), (x, y, width, height))
        image.set_colorkey(c.BLACK)
        image = pg.transform.scale(image,
(int(rect.width*c.SIZE_MULTIPLIER),int(rect.height*c.SIZE_MULTIPLIER)))
        return image


    def update(self, *args):
        pass
# Κλάση η οποία αντιπροσωπεύει το coin που λαμπιρίζει δίπλα στο πλήθος
κερδισμένων coin
class Coin(pg.sprite.Sprite):
    def __init__(self, x, y):
        super().__init__()
        self.sprite_sheet = setup.GFX['item_objects']
        self.create_frames()
        self.image = self.frames[0]
        self.rect = self.image.get_rect()
        self.rect.x = x
        self.rect.y = y
        self.timer = 0
        self.first_half = True
        self.frame_index = 0

    # Τις εικόνες για το coin τις αποθηκεύει στη λίστα self.frames
    def create_frames(self):
        self.frames = []
        self.frame_index = 0

        self.frames.append(self.get_image(1, 160, 5, 8))
        self.frames.append(self.get_image(9, 160, 5, 8))
        self.frames.append(self.get_image(17, 160, 5, 8))

    # Εξάγει τις εικόνες από το αρχείο item_objects
    def get_image(self, x, y, width, height):

        image = pg.Surface([width, height])
        rect = image.get_rect()

        image.blit(self.sprite_sheet, (0, 0), (x, y, width, height))
        image.set_colorkey(c.BLACK)
        image =
pg.transform.scale(image,(int(rect.width*c.BRICK_SIZE_MULTIPLIER),int(rect.
height*c.BRICK_SIZE_MULTIPLIER)))
        return image

    # Εναλλάσει τις 3 εικόνες του coin ώστε να δημιουργείται η ψευδαίσθηση
```

```
της κίνησης
    def update(self, current_time):
        if self.first_half:
            if self.frame_index == 0:
                if (current_time - self.timer) > 375:
                    self.frame_index += 1
                    self.timer = current_time
            elif self.frame_index < 2:
                if (current_time - self.timer) > 125:
                    self.frame_index += 1
                    self.timer = current_time
            elif self.frame_index == 2:
                if (current_time - self.timer) > 125:
                    self.frame_index -= 1
                    self.first_half = False
                    self.timer = current_time
        else:
            if self.frame_index == 1:
                if (current_time - self.timer) > 125:
                    self.frame_index -= 1
                    self.first_half = True
                    self.timer = current_time

        self.image = self.frames[self.frame_index]
import pygame as pg
from .. import setup, tools
from .. import constants as c
from . import powerups


# Κλάση για το Ηρακλή
class Hercules(pg.sprite.Sprite):
    def __init__(self):
        pg.sprite.Sprite.__init__(self)
        self.sprite_sheet = setup.GFX['hercules_sheet']
        # Αρχικοποίηση χρονομέτρων
        self.setup_timers()
        # Αρχικοποίηση καταστάσων
        self.setup_state_booleans()
        # Αρχικοποίηση μεταβλητών σχετικά με τις κινήσεις του Ηρακλή
        self.setup_forces()
        # Αρχικοποίηση μεταβλητών
        self.setup_counters()
        # Εισαγωγή εικόνων από το sprite sheet
        self.load_images_from_sheet()
        # Ορισμός αρχικής κατάστασης Ηρακλή
        self.state = c.WALK
        # Ορισμός αρχικής εικόνας
        self.image = self.right_frames[self.frame_index]
        # Ορισμός του rect για την αρχική εικόνα
        self.rect = self.image.get_rect()
        # Ορισμός της μάσκας για τον εντοπισμό των συγκρούσεων
        self.mask = pg.mask.from_surface(self.image)

        self.key_timer = 0

    # Μέθοδος που αρχικοποιεί τα χρονόμετρα που χρησιμοποιούνται για το
animation
    def setup_timers(self):
        self.walking_timer = 0
        self.invincible_animation_timer = 0
```

```
        self.invincible_start_timer = 0
        self.fire_transition_timer = 0
        self.death_timer = 0
        self.transition_timer = 0
        self.last_fireball_time = 0
        self.hurt_invisible_timer = 0
        self.hurt_invisible_timer2 = 0
        self.flag_pole_timer = 0

    # Μέθοδος που αρχικοποιεί τις καταστάσεις που βρίσκεται ο Ηρακλής
    def setup_state_booleans(self):
        self.facing_right = True
        self.allow_jump = True
        self.dead = False
        self.invincible = False
        self.big = False
        self.fire = False
        self.allow_fireball = True
        self.in_transition_state = False
        self.hurt_invincible = False
        self.in_castle = False
        self.crouching = False
        self.losing_invincibility = False

    # Μέθοδος που αρχικοποιεί τις μεταβλητές που σχετίζονται με τις
κινήσεις του Ηρακλή
    def setup_forces(self):
        self.x_vel = 0
        self.y_vel = 0
        self.max_x_vel = c.MAX_WALK_SPEED
        self.max_y_vel = c.MAX_Y_VEL
        self.x_accel = c.WALK_ACCEL
        self.jump_vel = c.JUMP_VEL
        self.gravity = c.GRAVITY

    # Μέθοδος που αρχικοποιεί μεταβλητές
    def setup_counters(self):
        self.frame_index = 0
        self.invincible_index = 0
        self.fire_transition_index = 0
        self.fireball_count = 0
        self.flag_pole_right = 0

    # Εξαγωγή των εικόνων από το αρχείο
    def load_images_from_sheet(self):
        # Αποθηκεύουμε τα frames του χαρακτήρα ανάλογα με τον
προσανατολισμό
        self.right_frames = []
        self.left_frames = []
        # Αποθηκεύουμε τα frames του μικρού χαρακτήρα ανάλογα με τον
προσανατολισμό
        self.right_small_normal_frames = []
        self.left_small_normal_frames = []

        # Αποθηκεύουμε τα frames του μεγάλου χαρακτήρα ανάλογα με τον
προσανατολισμό
        self.right_big_normal_frames = []
        self.left_big_normal_frames = []

        self.right_fire_frames = []
        self.left_fire_frames = []
```

```
        # Εικόνες από τον μικρό χαρακτήρα του Ηρακλή
        self.right_small_normal_frames.append(
            self.get_image(96, 406, 46, 58, c.SMALL_HERCULES))  # Right [0]
        self.right_small_normal_frames.append(
            self.get_image(303, 406, 54, 58, c.SMALL_HERCULES))  # Right
walking 1 [1]
        self.right_small_normal_frames.append(
            self.get_image(481, 405, 47, 58, c.SMALL_HERCULES))  # Right
walking 2 [2]
        self.right_small_normal_frames.append(
            self.get_image(580, 405, 58, 58, c.SMALL_HERCULES))  # Right
walking 3 [3]
        self.right_small_normal_frames.append(
            self.get_image(417, 602, 59, 66, c.SMALL_HERCULES))  # Right
jump [4]
        self.right_small_normal_frames.append(
            self.get_image(130, 32, 14, 16, c.SMALL_HERCULES))  # Right
skid [5] ΔΕΝ ΤΟ ΑΛΛΑΞΑ
        self.right_small_normal_frames.append(
            self.get_image(745, 1966, 79, 29, c.SMALL_HERCULES))  # Death
frame [6]
        self.right_small_normal_frames.append(
            self.get_image(912, 2132, 40, 70, c.SMALL_HERCULES))  #
Transition small to big [7]
        self.right_small_normal_frames.append(
            self.get_image(882, 2067, 70, 32, c.SMALL_HERCULES))  #
Transition big to small [8]
        self.right_small_normal_frames.append(
            self.get_image(1385, 2145, 50, 58, c.SMALL_HERCULES))  # Frame
1 of flag pole Slide [9]
        self.right_small_normal_frames.append(
            self.get_image(1438, 2145, 47, 58, c.SMALL_HERCULES))  # Frame
2 of flag pole slide [10]

        # Εικόνες για τον μεγάλο Ηρακλή
        self.right_big_normal_frames.append(
            self.get_image(96, 406, 46, 58, c.BIG_HERCULES))  # Right
standing [0]
        self.right_big_normal_frames.append(
            self.get_image(303, 406, 54, 58, c.BIG_HERCULES))  # Right
walking 1 [1]
        self.right_big_normal_frames.append(
            self.get_image(481, 405, 47, 58, c.BIG_HERCULES))  # Right
walking 2 [2]
        self.right_big_normal_frames.append(
            self.get_image(580, 405, 58, 58, c.BIG_HERCULES))  # Right
walking 3 [3]
        self.right_big_normal_frames.append(
            self.get_image(417, 602, 59, 66, c.BIG_HERCULES))  # Right jump
[4]
        self.right_big_normal_frames.append(
            self.get_image(385, 1274, 64, 60, c.BIG_HERCULES))  # Right
fire [5]
        self.right_big_normal_frames.append(
            self.get_image(912, 2132, 40, 70, c.BIG_HERCULES))  # Right get
power to fire [6]
        self.right_big_normal_frames.append(
            self.get_image(380, 517, 53, 40, c.BIG_HERCULES))  # Right
crouching [7]
        self.right_big_normal_frames.append(
```

```
            self.get_image(882, 2067, 70, 32, c.BIG_HERCULES))  #
Transition big to small [8]
        self.right_big_normal_frames.append(
            self.get_image(1385, 2145, 50, 58, c.BIG_HERCULES))  # Frame 1
of flag pole slide [9]
        self.right_big_normal_frames.append(
            self.get_image(1438, 2145, 47, 58, c.BIG_HERCULES))  # Frame 2
of flag pole slide [10]

        self.right_fire_frames = self.right_big_normal_frames

        # Οι εικόνες του Ηρακλή που κοιτούν προς τα αριστερά είναι οι ίδιες
        # με αυτές που κοιτούν προς τα δεξιά, περιστρεφόμενες κάθετα
        # Για τον μικρό χαρακτήρα του Ηρακλή
        for frame in self.right_small_normal_frames:
            new_image = pg.transform.flip(frame, True, False)
            self.left_small_normal_frames.append(new_image)

        # Για τον μεγάλο χαρακτήρα του Ηρακλή
        for frame in self.right_big_normal_frames:
            new_image = pg.transform.flip(frame, True, False)
            self.left_big_normal_frames.append(new_image)

        self.left_fire_frames = self.left_big_normal_frames

        self.fire_frames = [self.right_fire_frames, self.left_fire_frames]

        # Όλα τα frames για τον μικρό χαρακτήρα του Ηρακλή
        self.normal_small_frames = [self.right_small_normal_frames,
                                    self.left_small_normal_frames]

        # Όλα τα frames για τον μεγάλο χαρακτήρα του Ηρακλή
        self.normal_big_frames = [self.right_big_normal_frames,
                                  self.left_big_normal_frames]

        self.invincible_small_frames_list = [self.normal_small_frames]

        self.invincible_big_frames_list = [self.normal_big_frames]

        # Αποθήκευση όλων των frames
        self.all_images = [self.right_big_normal_frames,
                           self.right_small_normal_frames,
                           self.left_big_normal_frames,
                           self.left_small_normal_frames
                           ]
        # Αποθήκευση των δεξιών frames του μικρού χαρακτήρα
        self.right_frames = self.normal_small_frames[0]
        # Αποθήκευση των αριστερών frames του μικρού χαρακτήρα
        self.left_frames = self.normal_small_frames[1]

    # Εγαγωγή των frames από το sheet
    def get_image(self, x, y, width, height, size):
        image = pg.Surface([width, height])
        rect = image.get_rect()

        image.blit(self.sprite_sheet, (0, 0), (x, y, width, height))
        image.set_colorkey((173, 214, 214))
        image = pg.transform.scale(image, (int(rect.width * size),
int(rect.height * size)))
        return image
```

```
    # Κάνει update την κατάσταση του Ηρακλή και το animation για κάθε frame
    def update(self, keys, game_info, fire_group):
        self.current_time = game_info[c.CURRENT_TIME]
        self.handle_state(keys, fire_group)
        self.check_for_special_state()
        self.animation()

    # Εκτελείται η αντίστοιχη ενέργεια (μέθοδος) ανάλογα με την κατάσταση
του Ηρακλή
    def handle_state(self, keys, fire_group):
        if self.state == c.STAND:
            self.standing(keys, fire_group)
        elif self.state == c.WALK:
            self.walking(keys, fire_group)
        elif self.state == c.JUMP:
            self.jumping(keys, fire_group)
        elif self.state == c.FALL:
            self.falling(keys, fire_group)
        elif self.state == c.DEATH_JUMP:
            self.jumping_to_death()
        elif self.state == c.SMALL_TO_BIG:
            self.changing_to_big()
        elif self.state == c.BIG_TO_FIRE:
            self.changing_to_fire()
        elif self.state == c.BIG_TO_SMALL:
            self.changing_to_small()
        elif self.state == c.FLAGPOLE:
            self.flag_pole_sliding()
        elif self.state == c.BOTTOM_OF_POLE:
            self.sitting_at_bottom_of_pole()
        elif self.state == c.WALKING_TO_CASTLE:
            self.walking_to_castle()
        elif self.state == c.END_OF_LEVEL_FALL:
            self.falling_at_end_of_level()

    # Η μέθοδος εκτελείται όταν ο Ηρακλής είναι σταθερός
    def standing(self, keys, fire_group):
        self.check_to_allow_jump(keys)
        self.check_to_allow_fireball(keys)

        # Καθορίζεται το frame που θα εμφανιστεί και η ταχυτήτα κίνησης
        # και στους 2 άξονες στο 0
        self.frame_index = 0
        self.x_vel = 0
        self.y_vel = 0

        # Αν πατηθεί ο χαρακτήρας s εξετέσεται αν ο Ηρακλής μπορεί
        # να πυροβολήσει
        if keys[tools.keybinding['action']]:
            if self.fire and self.allow_fireball:
                self.shoot_fireball(fire_group)

        # Αν πατηθεί το κάτω βελάκι
        if keys[tools.keybinding['down']]:
            self.crouching = True

        # Αν πατηθεί το αριστερό βαλάκι
        if keys[tools.keybinding['left']]:
            self.facing_right = False
            self.get_out_of_crouch()
            self.state = c.WALK
```

```
        # Αν πατηθεί το δεξί βελάκι
        elif keys[tools.keybinding['right']]:
            self.facing_right = True
            self.get_out_of_crouch()
            self.state = c.WALK
        # Αν πατηθει ο χαρακτήρας a
        elif keys[tools.keybinding['jump']]:
            if self.allow_jump:
                if self.big:
                    setup.SFX['big_jump'].play()
                else:
                    setup.SFX['small_jump'].play()
                self.state = c.JUMP
                self.y_vel = c.JUMP_VEL
        else:
            self.state = c.STAND
        # Αν δεν είναι πατημένο το κάτω βελάκι
        if not keys[tools.keybinding['down']]:
            self.get_out_of_crouch()

    # Βγάζει τον χαρακτήρα από την γονατιστή θέση
    def get_out_of_crouch(self):
        # Παίρνει τις θέσεις του rect ώστε ο χαρακτήρας να τοποθετηθεί στην
ίδια θέση αλλά όρθιος
        bottom = self.rect.bottom
        left = self.rect.x
        # Επιλογή frame ανάλογα με τον προσανατολισμό του χαρακτήρα
        if self.facing_right:
            self.image = self.right_frames[0]
        else:
            self.image = self.left_frames[0]
        self.rect = self.image.get_rect()
        # Ο χαρακτήρας τοποθετείται στην ίδια θέση με πριν αλλά με άλλο
frame
        self.rect.bottom = bottom
        self.rect.x = left
        self.crouching = False

    # Έλεγχος για το αν μπορεί να πηδήξει ο χαρακτήρας
    def check_to_allow_jump(self, keys):
        # Αν δεν έχει πατηθεί το πλήκτο a τότε ο χαρακτήρας μπορεί να
πηδήξει
        if not keys[tools.keybinding['jump']]:
            self.allow_jump = True

    # Έλεγχος για το αν μπορεί να πυροβολήσει ο χαρακτήρας
    def check_to_allow_fireball(self, keys):
        # Αν δεν έχει πατηθεί το πλήκτο s τότε ο χαρακτήρας μπορεί να
πυροβολήσει
        if not keys[tools.keybinding['action']]:
            self.allow_fireball = True

    # Ο Ηρακλής πυροβολέι επιτρέποντας μέχρι δύο σφαίρες κάθε φορά
    def shoot_fireball(self, powerup_group):
        setup.SFX['fireball'].play()
        self.fireball_count = self.count_number_of_fireballs(powerup_group)

        if (self.current_time - self.last_fireball_time) > 200:
            if self.fireball_count < 2:
                self.allow_fireball = False
                powerup_group.add(
```

```
                    powerups.FireBall(self.rect.right, self.rect.y,
self.facing_right))
                self.last_fireball_time = self.current_time

                self.frame_index = 0
                if self.facing_right:
                    self.image = self.right_frames[self.frame_index]
                else:
                    self.image = self.left_frames[self.frame_index]

    def count_number_of_fireballs(self, powerup_group):
        fireball_list = []

        for powerup in powerup_group:
            if powerup.name == c.FIREBALL:
                fireball_list.append(powerup)

        return len(fireball_list)

    # Εκτελείται όταν ο Ηρακλής κινείται (WALKING) αλλάζοντας τα frames και
    # αλλάζει την κατάσταση του Ηρακλή ανάλογα με τις εντολές του χρήστη
    def walking(self, keys, fire_group):

        self.check_to_allow_jump(keys)
        self.check_to_allow_fireball(keys)
        # Ο χαρακτήρας από σταθερός αρχίζει να κινείται
        if self.frame_index == 0:
            self.frame_index += 1
            self.walking_timer = self.current_time
        else:
            # Ο χαρακτήρας βρίσκεται σε κίνηση.
            # Γίνεται εναλλαγή των frames 1,2,3 μετά από κάποιον χρόνο
            if (self.current_time - self.walking_timer >
                    self.calculate_animation_speed()):
                if self.frame_index < 3:
                    self.frame_index += 1
                else:
                    self.frame_index = 1

                self.walking_timer = self.current_time
        # Όταν ο χαρακτήρας κινείται και πατηθεί το πλήκτρο s, ο χαρακτήρας
επιταχ. και πυροβολάει αν μπορεί
        if keys[tools.keybinding['action']]:
            self.max_x_vel = c.MAX_RUN_SPEED
            self.x_accel = c.RUN_ACCEL
            if self.fire and self.allow_fireball:
                self.shoot_fireball(fire_group)
        else:
            self.max_x_vel = c.MAX_WALK_SPEED
            self.x_accel = c.WALK_ACCEL

        # Όταν ο χαρακτήτας κινείται και πατηθεί το πλήκτρο a, τότε ο
χαρακτήρας κάνει το jump
        # Εξετάζεται η ταχύτητα του χαρακτήρα και προσαρμόζεται το άλμα
        # Η κατάσταση αλλάζει σε JUMP
        if keys[tools.keybinding['jump']]:
            if self.allow_jump:
                if self.big:
                    setup.SFX['big_jump'].play()
                else:
                    setup.SFX['small_jump'].play()
```

```
                self.state = c.JUMP
                if self.x_vel > 4.5 or self.x_vel < -4.5:
                    self.y_vel = c.JUMP_VEL - .5
                else:
                    self.y_vel = c.JUMP_VEL
        # Όταν ο χαρακτήρας κινείται και πατηθεί το αριστερό βελάκι
        if keys[tools.keybinding['left']]:
            self.get_out_of_crouch()
            self.facing_right = False
            if self.x_vel > 0:
                self.frame_index = 1  # 5
                self.x_accel = c.SMALL_TURNAROUND
            else:
                self.x_accel = c.WALK_ACCEL

            if self.x_vel > (self.max_x_vel * -1):
                self.x_vel -= self.x_accel
                if self.x_vel > -0.5:
                    self.x_vel = -0.5
            elif self.x_vel < (self.max_x_vel * -1):
                self.x_vel += self.x_accel
        # Όταν ο χαρακτήρας κινείται και πατηθεί το δεξί βελάκι
        elif keys[tools.keybinding['right']]:
            self.get_out_of_crouch()
            self.facing_right = True
            if self.x_vel < 0:
                self.frame_index = 1  # 5
                self.x_accel = c.SMALL_TURNAROUND
            else:
                self.x_accel = c.WALK_ACCEL

            if self.x_vel < self.max_x_vel:
                self.x_vel += self.x_accel
                if self.x_vel < 0.5:
                    self.x_vel = 0.5
            elif self.x_vel > self.max_x_vel:
                self.x_vel -= self.x_accel

        else:
            if self.facing_right:
                if self.x_vel > 0:
                    self.x_vel -= self.x_accel
                else:
                    self.x_vel = 0
                    self.state = c.STAND
            else:
                if self.x_vel < 0:
                    self.x_vel += self.x_accel
                else:
                    self.x_vel = 0
                    self.state = c.STAND

    # Υπολογισμός της ταχύτητας του animation, η οποία γίνεται με βάση την
ταχύτητα του Ηρακλή
    def calculate_animation_speed(self):
        if self.x_vel == 0:
            animation_speed = 130
        elif self.x_vel > 0:
            animation_speed = 130 - (self.x_vel * (13))
        else:
            animation_speed = 130 - (self.x_vel * (13) * -1)
```

```
        return animation_speed

    # Εκτελείται όταν ο Ηρακλής πηδάει
    def jumping(self, keys, fire_group):
        self.allow_jump = False
        self.frame_index = 4
        self.gravity = c.JUMP_GRAVITY
        self.y_vel += self.gravity
        self.check_to_allow_fireball(keys)

        # Όταν τελειώσει το jump αλλάζει η κατάσταση και γίνεται FALL
        if self.y_vel >= 0 and self.y_vel < self.max_y_vel:
            self.gravity = c.GRAVITY
            self.state = c.FALL

        if keys[tools.keybinding['left']]:
            if self.x_vel > (self.max_x_vel * - 1):
                self.x_vel -= self.x_accel

        elif keys[tools.keybinding['right']]:
            if self.x_vel < self.max_x_vel:
                self.x_vel += self.x_accel

        if not keys[tools.keybinding['jump']]:
            self.gravity = c.GRAVITY
            self.state = c.FALL

        if keys[tools.keybinding['action']]:
            if self.fire and self.allow_fireball:
                self.shoot_fireball(fire_group)

    # Εκτελείται όταν ο Ηρακλής πέφτει μετά από jump
    def falling(self, keys, fire_group):

        self.check_to_allow_fireball(keys)
        if self.y_vel < c.MAX_Y_VEL:
            self.y_vel += self.gravity

        if keys[tools.keybinding['left']]:
            if self.x_vel > (self.max_x_vel * - 1):
                self.x_vel -= self.x_accel

        elif keys[tools.keybinding['right']]:
            if self.x_vel < self.max_x_vel:
                self.x_vel += self.x_accel

        if keys[tools.keybinding['action']]:
            if self.fire and self.allow_fireball:
                self.shoot_fireball(fire_group)

    # Εκτελείται όταν ο Ηρακλής πεθαίνει και εκτελείται το DEATH JUMP
    def jumping_to_death(self):
        if self.death_timer == 0:
            self.death_timer = self.current_time
        elif (self.current_time - self.death_timer) > 500:
            self.rect.y += self.y_vel
            self.y_vel += self.gravity

    def start_death_jump(self, game_info):
        self.dead = True
```

```
        game_info[c.HERCULES_DEAD] = True
        self.y_vel = -11
        self.gravity = .5
        self.frame_index = 6
        self.image = self.right_frames[self.frame_index]
        self.state = c.DEATH_JUMP
        self.in_transition_state = True

    # Αλλάζει το μέγεθος του Ηρακλή από μικρό σε μεγάλο
    def changing_to_big(self):

        self.in_transition_state = True

        if self.transition_timer == 0:
            self.transition_timer = self.current_time
        elif self.timer_between_these_two_times(135, 885):
            self.set_hercules_to_powerup_image()

        elif self.timer_between_these_two_times(885, 950):
            self.set_hercules_to_big_image()
            self.state = c.WALK
            self.in_transition_state = False
            self.transition_timer = 0
            self.become_big()

    # Εξετάζει αν ο χρόνος είναι μέσα στα όρια του start_time και του
end_time
    def timer_between_these_two_times(self, start_time, end_time):
        if (self.current_time - self.transition_timer) >= start_time \
                and (self.current_time - self.transition_timer) < end_time:
            return True

    # Εκτελείται κατά τη διάρκεια της μετατροπής του Ηρακλή από μικρό σε
μεγάλο
    # και εμφανίζει την εικόνα με τον Ηρακλή να παίρνει δύναμη
    def set_hercules_to_powerup_image(self):

        if self.facing_right:
            self.image = self.normal_small_frames[1][7]
        else:
            self.image = self.normal_small_frames[0][7]
        bottom = self.rect.bottom
        centerx = self.rect.centerx
        self.rect = self.image.get_rect()
        self.rect.bottom = bottom
        self.rect.centerx = centerx

    # Εκτελείται κατά τη διάρκεια της μετατροπής του Ηρακλή από μικρό σε
μεγάλο
    # και εμγανίζει τον μεγάλο Ηρακλή
    def set_hercules_to_big_image(self):
        if self.facing_right:
            self.image = self.normal_big_frames[1][7]
        else:
            self.image = self.normal_big_frames[0][7]
        bottom = self.rect.bottom
        centerx = self.rect.centerx
        self.rect = self.image.get_rect()
        self.rect.bottom = bottom
        self.rect.centerx = centerx
```

```
    # Εκτελείται όταν ο Ηρακλής γίνει μεγάλος. Αλλάζει τις δομές ώστε να
διαχειρίζεται τα ανάλογα (μεγάλα)
    # frames και τοποθετεί τον χαρακτήρα στο ίδιο σημείο μετά την μετατροπή
του σε μεγάλο
    def become_big(self):
        self.big = True
        self.right_frames = self.right_big_normal_frames
        self.left_frames = self.left_big_normal_frames
        bottom = self.rect.bottom
        left = self.rect.x
        image = self.right_frames[0]
        self.rect = image.get_rect()
        self.rect.bottom = bottom
        self.rect.x = left

    # Εκτελείται όταν ο Ηρακλής μπορεί να πυροβολίσει
    def changing_to_fire(self):

        self.in_transition_state = True

        if self.facing_right:
            frames = self.left_big_normal_frames
        else:
            frames = self.right_big_normal_frames

        if self.fire_transition_timer == 0:
            self.fire_transition_timer = self.current_time
        elif (self.current_time - self.fire_transition_timer) > 65 and (
                    self.current_time - self.fire_transition_timer) < 130:
            self.image = frames[0]
        elif (self.current_time - self.fire_transition_timer) < 975:
            self.image = frames[6]

        elif (self.current_time - self.fire_transition_timer) < 1040:
            self.image = frames[0]
            self.fire = True
            self.in_transition_state = False
            self.state = c.WALK
            self.transition_timer = 0

    # Εκτελείται όταν ο Ηρακλής χάνει δύναμη και από μεγάλος γίνεται μικρός
    def changing_to_small(self):

        self.in_transition_state = True
        self.hurt_invincible = True
        self.state = c.BIG_TO_SMALL

        if self.facing_right:
            frames = [self.right_big_normal_frames[0],
                      self.right_big_normal_frames[8],
                      self.right_small_normal_frames[8]
                      ]
        else:
            frames = [self.left_big_normal_frames[0],
                      self.left_big_normal_frames[8],
                      self.left_small_normal_frames[8]
                      ]

        if self.transition_timer == 0:
            self.transition_timer = self.current_time
        elif (self.current_time - self.transition_timer) < 265:
```

```
            self.image = frames[0]
            self.hurt_invincible_check()
            self.adjust_rect()
        elif (self.current_time - self.transition_timer) < 330:
            self.image = frames[1]
            self.hurt_invincible_check()
            self.adjust_rect()
        elif (self.current_time - self.transition_timer) < 395:
            self.image = frames[2]
            self.hurt_invincible_check()
            self.adjust_rect()
        elif (self.current_time - self.transition_timer) < 460:
            self.image = frames[1]
            self.hurt_invincible_check()
            self.adjust_rect()
        elif (self.current_time - self.transition_timer) < 525:
            self.image = frames[2]
            self.hurt_invincible_check()
            self.adjust_rect()
        elif (self.current_time - self.transition_timer) < 590:
            self.image = frames[1]
            self.hurt_invincible_check()
            self.adjust_rect()
        elif (self.current_time - self.transition_timer) < 655:
            self.image = frames[2]
            self.hurt_invincible_check()
            self.adjust_rect()
        elif (self.current_time - self.transition_timer) < 720:
            self.image = frames[1]
            self.hurt_invincible_check()
            self.adjust_rect()
        elif (self.current_time - self.transition_timer) < 785:
            self.image = frames[2]
            self.hurt_invincible_check()
            self.adjust_rect()
        elif (self.current_time - self.transition_timer) < 850:
            self.image = frames[1]
            self.hurt_invincible_check()
            self.adjust_rect()
        elif (self.current_time - self.transition_timer) < 915:
            self.image = frames[2]
            self.adjust_rect()
            self.in_transition_state = False
            self.state = c.WALK
            self.big = False
            self.transition_timer = 0
            self.hurt_invisible_timer = 0
            self.become_small()

    # Εκτελείται όταν ο Ηρακλής αλλάζει μέγεθος εξασφαλίζοντας ότι
    # το νέο frame θα εμφανιστεί στο ίδιο frame με το παλιό
    def adjust_rect(self):
        x = self.rect.x
        bottom = self.rect.bottom
        self.rect = self.image.get_rect()
        self.rect.x = x
        self.rect.bottom = bottom

    # Εκτελείται όταν ο Ηρακλής γίνει απο μεγάλος μικρος
    # καθορίζει τις δομες με τα frames που θα χρησμοποιηθουν
    # και εξασφαλίζει ότι ο χαρακτήρας θα εμφανιστεί στο ίδιο σημείο
```

```
    # μετά την μετατροπή
    def become_small(self):
        self.big = False
        self.right_frames = self.right_small_normal_frames
        self.left_frames = self.left_small_normal_frames
        bottom = self.rect.bottom
        left = self.rect.x
        image = self.right_frames[0]
        self.rect = image.get_rect()
        self.rect.bottom = bottom
        self.rect.x = left

    # Εκτελείται όταν ο Ηρακλής κατεβαίνει από το flag pole
    # στο τέλος της πίστας
    def flag_pole_sliding(self):
        self.state = c.FLAGPOLE
        self.in_transition_state = True
        self.x_vel = 0
        self.y_vel = 0

        if self.flag_pole_timer == 0:
            self.flag_pole_timer = self.current_time
        elif self.rect.bottom < 493:
            if (self.current_time - self.flag_pole_timer) < 65:
                self.image = self.right_frames[9]
            elif (self.current_time - self.flag_pole_timer) < 130:
                self.image = self.right_frames[10]
            elif (self.current_time - self.flag_pole_timer) >= 130:
                self.flag_pole_timer = self.current_time

            self.rect.right = self.flag_pole_right
            self.y_vel = 5
            self.rect.y += self.y_vel

            if self.rect.bottom >= 488:
                self.flag_pole_timer = self.current_time

        elif self.rect.bottom >= 493:
            self.image = self.right_frames[10]

    # Εκτελείται όταν ο Ηρακλής έχει κατέβει από το flag pole
    def sitting_at_bottom_of_pole(self):
        if self.flag_pole_timer == 0:
            self.flag_pole_timer = self.current_time
            self.image = self.left_frames[10]
        elif (self.current_time - self.flag_pole_timer) < 210:
            self.image = self.left_frames[10]
        else:
            self.in_transition_state = False
            if self.rect.bottom < 485:
                self.state = c.END_OF_LEVEL_FALL
            else:
                self.state = c.WALKING_TO_CASTLE

    # Θέτει την κατάσταση του Ηρακλή σε BOTTOM_OF_POLE
    def set_state_to_bottom_of_pole(self):
        self.image = self.left_frames[9]
        right = self.rect.right
        self.rect.x = right
        if self.big:
            self.rect.x -= 10
```

```
        self.flag_pole_timer = 0
        self.state = c.BOTTOM_OF_POLE

    # Εκτελείται στο τέλος της πίστας και μετακινεί τον Ηρακλή μέσα στο
κάστρο.
    def walking_to_castle(self):
        self.max_x_vel = 5
        self.x_accel = c.WALK_ACCEL

        if self.x_vel < self.max_x_vel:
            self.x_vel += self.x_accel

        if (self.walking_timer == 0 or (self.current_time -
self.walking_timer) > 200):
            self.walking_timer = self.current_time

        elif (self.current_time - self.walking_timer) > \
                self.calculate_animation_speed():
            if self.frame_index < 3:
                self.frame_index += 1
            else:
                self.frame_index = 1
            self.walking_timer = self.current_time

    def falling_at_end_of_level(self, *args):
        self.y_vel += c.GRAVITY

    # Εξατάζει αν ο Ηρακλής είναι σε κάποια "ειδική" κατάσταση όπως
    # άτροτος, on fire ή lose power ή είναι γονατιστός
    def check_for_special_state(self):
        self.check_if_invincible()
        self.check_if_fire()
        self.check_if_hurt_invincible()
        self.check_if_crouching()

    # Εξετάζει αν ο Ηρακλής είναι άτροτος
    # επίσης εναλλάσει τα frames ανάμεσα στον μεγάλο και στο μικρό Ηρακλή
    def check_if_invincible(self):
        if self.invincible:
            if ((self.current_time - self.invincible_start_timer) < 10000):
                self.losing_invincibility = False
                self.change_frame_list(30)
            elif ((self.current_time - self.invincible_start_timer) <
12000):
                self.losing_invincibility = True
                self.change_frame_list(100)
            else:
                self.losing_invincibility = False
                print("Invicinble: ", self.losing_invincibility)
                self.invincible = False
        else:
            if self.big:
                self.right_frames = self.right_big_normal_frames
                self.left_frames = self.left_big_normal_frames
            else:
                self.right_frames = self.invincible_small_frames_list[0][0]
                self.left_frames = self.invincible_small_frames_list[0][1]

    def change_frame_list(self, frame_switch_speed):
        if (self.current_time - self.invincible_animation_timer) >
frame_switch_speed:
```

```
            if self.invincible_index <
(len(self.invincible_small_frames_list) - 1):
                self.invincible_index += 1
            else:
                self.invincible_index = 0

            if self.big:
                frames =
self.invincible_big_frames_list[self.invincible_index]
            else:
                frames =
self.invincible_small_frames_list[self.invincible_index]

            self.right_frames = frames[0]
            self.left_frames = frames[1]

            self.invincible_animation_timer = self.current_time

    # Ελεγχει αν ο Ηρακλής μπορεί να πυροβολήσει
    def check_if_fire(self):
        if self.fire and self.invincible == False:
            self.right_frames = self.fire_frames[0]
            self.left_frames = self.fire_frames[1]

    # Εξετάζει αν  ο Ηρακλής είναι προσωρινά άτροτος μετά από χτύπημα που
δέχθηκε
    def check_if_hurt_invincible(self):

        if self.hurt_invincible and self.state != c.BIG_TO_SMALL:
            if self.hurt_invisible_timer2 == 0:
                self.hurt_invisible_timer2 = self.current_time
            elif (self.current_time - self.hurt_invisible_timer2) < 2000:
                self.hurt_invincible_check()
            else:
                self.hurt_invincible = False
                self.hurt_invisible_timer = 0
                self.hurt_invisible_timer2 = 0
                for frames in self.all_images:
                    for image in frames:
                        image.set_alpha(255)

    def hurt_invincible_check(self):
        if self.hurt_invisible_timer == 0:
            self.hurt_invisible_timer = self.current_time
        elif (self.current_time - self.hurt_invisible_timer) < 35:
            self.image.set_alpha(0)
        elif (self.current_time - self.hurt_invisible_timer) < 70:
            self.image.set_alpha(255)
            self.hurt_invisible_timer = self.current_time

    # Ελέγχει αν Ηρακλής είναι σκυφτός
    def check_if_crouching(self):

        if self.crouching and self.big:
            bottom = self.rect.bottom
            left = self.rect.x
            if self.facing_right:
                self.image = self.right_frames[7]
            else:
                self.image = self.left_frames[7]
            self.rect = self.image.get_rect()
```

```
            self.rect.bottom = bottom
            self.rect.x = left

    # Προσαρμώσει την εικόνα του Ηρακλή ανάλγα με την κατάσταση στην οποία
βρίσκεται
    def animation(self):
        if self.state == c.DEATH_JUMP \
                or self.state == c.SMALL_TO_BIG \
                or self.state == c.BIG_TO_FIRE \
                or self.state == c.BIG_TO_SMALL \
                or self.state == c.FLAGPOLE \
                or self.state == c.BOTTOM_OF_POLE \
                or self.crouching:
            pass
        elif self.facing_right:
            self.image = self.right_frames[self.frame_index]
        else:
            self.image = self.left_frames[self.frame_index]
# Υπερκλάση για όλους τους χαρακτήρες που χρησιμοποιούνται στο overhead
info
class Character(pg.sprite.Sprite):
    def __init__(self, image):
        super().__init__()
        # Κάθε χαρακτήρας έχει ένα img και ένα rect
        self.image = image
        self.rect = self.image.get_rect()

# κλάση η οποία χρησιμοποιείται για την απεικόνιση των στοιχείων
# στο επάνω μέρος της οθόνης όπως πχ score, coins, time
class OverheadInfo(object):
    def __init__(self, game_info, state):
        self.sprite_sheet = setup.GFX['text_images']
        self.coin_total = game_info[c.COIN_TOTAL]
        self.time = 401
        self.current_time = 0
        self.total_lives = game_info[c.LIVES]
        self.top_score = game_info[c.TOP_SCORE]
        self.state = state
        self.world3 = False
        self.special_state = None
        self.game_info = game_info
        self.previous = self.game_info[c.PREV_LEVEL]
        self.create_image_dict()
        self.create_score_group()
        self.check_label()
        self.create_info_labels()
        self.create_load_screen_labels()
        self.create_countdown_clock()
        self.create_coin_counter()
        self.create_flashing_coin()
        self.create_hercules_image()
        self.create_game_over_label()
        self.create_time_out_label()
        self.create_main_menu_labels()

    # Δημιουργία λεξικού το οποίο αντιστοιχίζει όους του χαρακτήρες και τα
γράμματα
    # με τις αντίστοιχες εικόνες οι οποίες εξάγονται από το αρχείο
text_images.png
    def create_image_dict(self):
        self.image_dict = {}
```

```
        image_list = []

        image_list.append(self.get_image(3, 230, 7, 7))
        image_list.append(self.get_image(12, 230, 7, 7))
        image_list.append(self.get_image(19, 230, 7, 7))
        image_list.append(self.get_image(27, 230, 7, 7))
        image_list.append(self.get_image(35, 230, 7, 7))
        image_list.append(self.get_image(43, 230, 7, 7))
        image_list.append(self.get_image(51, 230, 7, 7))
        image_list.append(self.get_image(59, 230, 7, 7))
        image_list.append(self.get_image(67, 230, 7, 7))
        image_list.append(self.get_image(75, 230, 7, 7))

        image_list.append(self.get_image(83, 230, 7, 7))
        image_list.append(self.get_image(91, 230, 7, 7))
        image_list.append(self.get_image(99, 230, 7, 7))
        image_list.append(self.get_image(107, 230, 7, 7))
        image_list.append(self.get_image(115, 230, 7, 7))
        image_list.append(self.get_image(123, 230, 7, 7))
        image_list.append(self.get_image(3, 238, 7, 7))
        image_list.append(self.get_image(11, 238, 7, 7))
        image_list.append(self.get_image(20, 238, 7, 7))
        image_list.append(self.get_image(27, 238, 7, 7))
        image_list.append(self.get_image(35, 238, 7, 7))
        image_list.append(self.get_image(44, 238, 7, 7))
        image_list.append(self.get_image(51, 238, 7, 7))
        image_list.append(self.get_image(59, 238, 7, 7))
        image_list.append(self.get_image(67, 238, 7, 7))
        image_list.append(self.get_image(75, 238, 7, 7))
        image_list.append(self.get_image(83, 238, 7, 7))
        image_list.append(self.get_image(91, 238, 7, 7))
        image_list.append(self.get_image(99, 238, 7, 7))
        image_list.append(self.get_image(108, 238, 7, 7))
        image_list.append(self.get_image(115, 238, 7, 7))
        image_list.append(self.get_image(123, 238, 7, 7))
        image_list.append(self.get_image(3, 246, 7, 7))
        image_list.append(self.get_image(11, 246, 7, 7))
        image_list.append(self.get_image(20, 246, 7, 7))
        image_list.append(self.get_image(27, 246, 7, 7))
        image_list.append(self.get_image(48, 248, 7, 7))

        image_list.append(self.get_image(68, 249, 6, 2))
        image_list.append(self.get_image(75, 247, 6, 6))

        character_string = '0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ -*'

        for character, image in zip(character_string, image_list):
            self.image_dict[character] = image

    # Εξαγωγή των εικόνων από το sprite sheet
    # x-cord, y-cord, width, height
    def get_image(self, x, y, width, height):
        # Δημιουργία εικόνας
        image = pg.Surface([width, height])
        rect = image.get_rect()

        image.blit(self.sprite_sheet, (0, 0), (x, y, width, height))
        image.set_colorkey((92, 148, 252))
        image =
pg.transform.scale(image,(int(rect.width*2.9),int(rect.height*2.9)))
```

```
        return image

    # Δημιουργεί το σκορ που εμφανίζεται επάνω και αριστερά στην οθόνη
    # Αρχικο σκορ είναι το 000000
    def create_score_group(self):
        self.score_images = []
        # δομή, string, x-coord, y-coord
        self.create_label(self.score_images, '000000', 75, 55)


    def check_label(self):
        if self.previous == c.LABOUR_BULL or self.previous == None:
            self.labour = '1-1'
        elif self.previous == c.LABOUR_LION:
            self.labour = '1-2'
        elif self.previous == c.LABOUR_HYDRA:
            self.labour = '1-3'

    # Δημιουργεί τα labels για κάθε πληροφορία που εμφανίζεται στο επάνω
μέρος της οθόνης
    def create_info_labels(self):

        self.hercules_label = []
        self.world_label = []
        self.time_label = []
        self.stage_label = []

        self.create_label(self.hercules_label, 'HERCULES', 75, 30)
        self.create_label(self.world_label, 'LABOR', 440, 30)
        self.create_label(self.time_label, 'TIME', 625, 30)
        self.create_label(self.stage_label, self.labour, 472, 55)

        self.label_list = [self.hercules_label,
                           self.world_label,
                           self.time_label,
                           self.stage_label]

    # Δημιουργεί τα labels τα οποία χρησιμοποιούνται για τις πληροφορίες
που εμφανίζονται
    # στο κεντρο της οθόνης πριν ξεκινήσει η πίστα
    def create_load_screen_labels(self):
        world_label = []
        number_label = []

        self.create_label(world_label, 'LABOUR', 280, 200)
        self.create_label(number_label, self.labour, 430, 200)

        self.center_labels = [world_label, number_label]

    # Δημιουργεί την αντίστροφη μέτρηση που εμφανίζεται πάνω δεξιά
    def create_countdown_clock(self):
        self.count_down_images = []
        self.create_label(self.count_down_images, str(self.time), 645, 55)

    # Για κάθε γράμμα του string εισάγει στην λίστα την αντίστοιχη image
    def create_label(self, label_list, string, x, y):

        for letter in string:
            label_list.append(Character(self.image_dict[letter]))

        # Καλεί τη μέθοδο για να καθορίζει το που θα τοποθετηθούν τα
```

```
στοιχεία στην οθόνη
        self.set_label_rects(label_list, x, y)

    # Καθορίζει τη τοποθεσία που θα εμφανιστεί ο κάθε χαρακτήρας στην οθόνη
    def set_label_rects(self, label_list, x, y):
        for i, letter in enumerate(label_list):
            letter.rect.x = x + ((letter.rect.width + 3) * i)
            letter.rect.y = y
            if letter.image == self.image_dict['-']:
                letter.rect.y += 7
                letter.rect.x += 2

    # Δημιουργεί το πλήθος των νομισμάτων που εμφανίζονται στο πάνω
(κεντρικό) μέρος της οθόνης
    def create_coin_counter(self):
        self.coin_count_images = []
        self.create_label(self.coin_count_images, '*00', 300, 55)

    # Δημιουργεί το coin που εμφανίζεται δίπλα στο πλήθος των κερδισμένων
coin
    def create_flashing_coin(self):
        """Creates the flashing coin next to the coin total"""
        self.flashing_coin = flashing_coin.Coin(280, 53)

    # Παίρνουμε την εικόνα του Hercules
    def create_hercules_image(self):
        # Παίρνουμε το x από το αρχείο text_images.png
        self.life_times_image = self.get_image(75, 247, 6, 6)
        self.life_times_rect = self.life_times_image.get_rect(center=(378,
295))
        self.life_total_label = []
        self.create_label(self.life_total_label, str(self.total_lives),
                          450, 285)
        # Εξάγουμε τον Hercules από το αρχείο hercules_sheet
        self.sprite_sheet = setup.GFX['hercules_sheet']
        self.hercules_image = self.get_image(96, 132, 48, 62)
        self.hercules_image.set_colorkey((173, 214, 214))
        self.hercules_rect = self.hercules_image.get_rect()
        self.hercules_image = pg.transform.scale(self.hercules_image,

(int(self.hercules_rect.width * c.HERCULES_SIZE_MULTIPLIER),
                                 int(self.hercules_rect.height *
c.HERCULES_SIZE_MULTIPLIER)))

        self.hercules_rect = self.hercules_image.get_rect(center=(320,
290))

    # Δημιουργία των label για την GAME OVER screen
    def create_game_over_label(self):
        game_label = []
        over_label = []

        self.create_label(game_label, 'GAME', 280, 300)
        self.create_label(over_label, 'OVER', 400, 300)

        self.game_over_label = [game_label, over_label]

    # Δημιουργία των label για την TIME OUT screen
    def create_time_out_label(self):
        """Create the label for the time out screen"""
        time_out_label = []
```

```
        self.create_label(time_out_label, 'TIME OUT', 290, 310)
        self.time_out_label = [time_out_label]

    # Δημιουργία των labels για την MAIN MENU screen
    def create_main_menu_labels(self):

        player_one_game = []
        player_two_game = []
        top = []
        top_score = []

        self.create_label(player_one_game, '1 PLAYER GAME', 272, 360)
        self.create_label(top, 'TOP - ', 290, 465)
        self.create_label(top_score, '000000', 400, 465)

        self.main_menu_labels = [player_one_game, player_two_game,
                                 top, top_score]

    # Κάνει update τις πληροφορίες που εμφανίζονται στο overhead
    def update(self, level_info, hercules=None):
        self.hercules = hercules
        self.handle_level_state(level_info)

    # Κάνει update με βάση την κατάσταση στην οποία βρίσκεται το παιχνίδι
    def handle_level_state(self, level_info):
        # Το παιχνίδι είναι σε κατάσταση Main Menu
        if self.state == c.MAIN_MENU:
            self.score = level_info[c.SCORE]
            self.update_score_images(self.score_images, self.score)
            self.update_score_images(self.main_menu_labels[3],
self.top_score)
            self.update_coin_total(level_info)
            self.flashing_coin.update(level_info[c.CURRENT_TIME])
        # Το παιχνίδι είναι σε κατάσταση LOAD
        elif self.state == c.LOAD_SCREEN:
            self.score = level_info[c.SCORE]
            self.update_score_images(self.score_images, self.score)
            self.update_coin_total(level_info)
        # Το παιχνίδι είναι σε κατάσταση LEVEL δηλαδή ο χρήστης παίζει
        elif self.state == c.LEVEL:
            self.score = level_info[c.SCORE]
            self.update_score_images(self.score_images, self.score)
            if level_info[c.LEVEL_STATE] != c.FROZEN \
                    and self.hercules.state != c.WALKING_TO_CASTLE \
                    and self.hercules.state != c.END_OF_LEVEL_FALL \
                    and not self.hercules.dead:
                self.update_count_down_clock(level_info)
            self.update_coin_total(level_info)
            self.flashing_coin.update(level_info[c.CURRENT_TIME])
        # Το παιχνίδι είναι σε κατάσταση TIME OUT, δηλαδή τελείωσε ο χρόνος
        elif self.state == c.TIME_OUT:
            self.score = level_info[c.SCORE]
            self.update_score_images(self.score_images, self.score)
            self.update_coin_total(level_info)
        # Το παιχνίδι είναι σε κατάσταση GAME OVER
        elif self.state == c.GAME_OVER:
            self.score = level_info[c.SCORE]
            self.update_score_images(self.score_images, self.score)
            self.update_coin_total(level_info)
        # Το παιχνίδι είναι σε κατάσταση FAST COUNT DOWN
```

```
        elif self.state == c.FAST_COUNT_DOWN:
            level_info[c.SCORE] += 50
            self.score = level_info[c.SCORE]
            self.update_count_down_clock(level_info)
            self.update_score_images(self.score_images, self.score)
            self.update_coin_total(level_info)
            self.flashing_coin.update(level_info[c.CURRENT_TIME])
            if self.time == 0:
                self.state = c.END_OF_LEVEL
        # Το παιχνίδι είναι σε κατάσταση END OF LEVEL, δηλαδή ο παίκτης
πέρασε με επιτυχία το level
        elif self.state == c.END_OF_LEVEL:
            self.flashing_coin.update(level_info[c.CURRENT_TIME])

    # Κάνει update τους εικόνες των αριθμών των score οι οποίες θα
εμφανιστούν στην οθόνη
    def update_score_images(self, images, score):
        index = len(images) - 1

        for digit in reversed(str(score)):
            rect = images[index].rect
            images[index] = Character(self.image_dict[digit])
            images[index].rect = rect
            index -= 1

    # Μέθοδος που κάνει update το ρολόι του παιχνιδιού
    def update_count_down_clock(self, level_info):
        if self.state == c.FAST_COUNT_DOWN:
            self.time -= 1

        elif (level_info[c.CURRENT_TIME] - self.current_time) > 400:
            self.current_time = level_info[c.CURRENT_TIME]
            self.time -= 1
        self.count_down_images = []
        self.create_label(self.count_down_images, str(self.time), 645, 55)
        if len(self.count_down_images) < 2:
            for i in range(2):
                self.count_down_images.insert(0,
Character(self.image_dict['0']))
            self.set_label_rects(self.count_down_images, 645, 55)
        elif len(self.count_down_images) < 3:
            self.count_down_images.insert(0,
Character(self.image_dict['0']))
            self.set_label_rects(self.count_down_images, 645, 55)

    # Κάνει update τις εικόνες που δείχνουν το πλήθος των coins
    def update_coin_total(self, level_info):
        self.coin_total = level_info[c.COIN_TOTAL]

        coin_string = str(self.coin_total)
        if len(coin_string) < 2:
            coin_string = '*0' + coin_string
        elif len(coin_string) > 2:
            coin_string = '*00'
        else:
            coin_string = '*' + coin_string

        x = self.coin_count_images[0].rect.x
        y = self.coin_count_images[0].rect.y

        self.coin_count_images = []
```

```
        self.create_label(self.coin_count_images, coin_string, x, y)

    # Εμφάνιση των στοιχείων overhead πάνω στην εικόνα
    def draw(self, surface):
        if self.state == c.MAIN_MENU:
            self.draw_main_menu_info(surface)
        elif self.state == c.LOAD_SCREEN:
            self.draw_loading_screen_info(surface)
        elif self.state == c.LEVEL:
            self.draw_level_screen_info(surface)
        elif self.state == c.GAME_OVER:
            self.draw_game_over_screen_info(surface)
        elif self.state == c.FAST_COUNT_DOWN:
            self.draw_level_screen_info(surface)
        elif self.state == c.END_OF_LEVEL:
            self.draw_level_screen_info(surface)
        elif self.state == c.TIME_OUT:
            self.draw_time_out_screen_info(surface)
        else:
            pass


    # Αν το state είναι σε MAIN MENU τότε το ζωγραφίζει στην οθόνη
    def draw_main_menu_info(self, surface):
        for info in self.score_images:
            surface.blit(info.image, info.rect)

        for label in self.main_menu_labels:
            for letter in label:
                surface.blit(letter.image, letter.rect)

        for character in self.coin_count_images:
            surface.blit(character.image, character.rect)

        for label in self.label_list:
            for letter in label:
                surface.blit(letter.image, letter.rect)

        surface.blit(self.flashing_coin.image, self.flashing_coin.rect)

    # Αν το state είναι σε LOAD (αναμονή έναρξης) τότε το ζωγραφίζει στην
οθόνη
    def draw_loading_screen_info(self, surface):
        for info in self.score_images:
            surface.blit(info.image, info.rect)

        for word in self.center_labels:
            for letter in word:
                surface.blit(letter.image, letter.rect)

        for word in self.life_total_label:
            surface.blit(word.image, word.rect)

        surface.blit(self.hercules_image, self.hercules_rect)
        surface.blit(self.life_times_image, self.life_times_rect)

        for character in self.coin_count_images:
            surface.blit(character.image, character.rect)

        for label in self.label_list:
```

```
            for letter in label:
                surface.blit(letter.image, letter.rect)

        surface.blit(self.flashing_coin.image, self.flashing_coin.rect)

    # Αν το state είναι σε LEVEL (game play) τότε ζωγραφίζει στην οθόνη τις
πληροφορίες
    def draw_level_screen_info(self, surface):
        for info in self.score_images:
            surface.blit(info.image, info.rect)

        for digit in self.count_down_images:
                surface.blit(digit.image, digit.rect)

        for character in self.coin_count_images:
            surface.blit(character.image, character.rect)

        for label in self.label_list:
            for letter in label:
                surface.blit(letter.image, letter.rect)

        surface.blit(self.flashing_coin.image, self.flashing_coin.rect)

    # Αν το state είναι σε GAME OVER τότε το ζωγραφίζει στην οθόνη
    def draw_game_over_screen_info(self, surface):
        for info in self.score_images:
            surface.blit(info.image, info.rect)

        for word in self.game_over_label:
            for letter in word:
                surface.blit(letter.image, letter.rect)

        for character in self.coin_count_images:
            surface.blit(character.image, character.rect)

        for label in self.label_list:
            for letter in label:
                surface.blit(letter.image, letter.rect)

        surface.blit(self.flashing_coin.image, self.flashing_coin.rect)

    # Αν το state είναι σε TIME OUT τότε το ζωγραφίζει στην οθόνη
    def draw_time_out_screen_info(self, surface):
        for info in self.score_images:
            surface.blit(info.image, info.rect)

        for word in self.time_out_label:
            for letter in word:
                surface.blit(letter.image, letter.rect)

        for character in self.coin_count_images:
            surface.blit(character.image, character.rect)

        for label in self.label_list:
            for letter in label:
                surface.blit(letter.image, letter.rect)

        surface.blit(self.flashing_coin.image, self.flashing_coin.rect)
# Κλάση η οποία αντιπροσωπεύει όλα τα αντικείμενα τα οποία
# δίνουν ενέργεια στον επιπρόσθετες δυνάμεις στον Ηρακλή
class Powerup(pg.sprite.Sprite):
```

```
    def __init__(self, x, y):
        super(Powerup, self).__init__()


    def setup_powerup(self, x, y, name, setup_frames, sprite_sheet):
        self.sprite_sheet = setup.GFX[sprite_sheet]
        self.frames = []
        self.frame_index = 0
        setup_frames()
        self.image = self.frames[self.frame_index]
        self.rect = self.image.get_rect()
        self.rect.centerx = x
        self.rect.y = y
        self.state = c.REVEAL
        self.y_vel = -1
        self.x_vel = 0
        self.direction = c.RIGHT
        self.box_height = y
        self.gravity = 1
        self.max_y_vel = 8
        self.animate_timer = 0
        self.name = name

    # Εξαγωγή εικόνας από το sprite sheet
    def get_image(self, x, y, width, height, sprite_sheet):
        image = pg.Surface([width, height]).convert()
        rect = image.get_rect()

        image.blit(self.sprite_sheet, (0, 0), (x, y, width, height))
        if sprite_sheet == 'powerup':
            image.set_colorkey((112,146,190))
            print('test')
        else:
            image.set_colorkey(c.BLACK)
        image =
pg.transform.scale(image,(int(rect.width*c.SIZE_MULTIPLIER),int(rect.height
*c.SIZE_MULTIPLIER)))
        return image


    def update(self, game_info, *args):
        self.current_time = game_info[c.CURRENT_TIME]
        self.handle_state()


    def handle_state(self):
        pass

    # Βγάζει το powerup από το coin box
    def revealing(self, *args):
        self.rect.y += self.y_vel

        if self.rect.bottom <= self.box_height:
            self.rect.bottom = self.box_height
            self.y_vel = 0
            self.state = c.SLIDE

    # Μετακίνηση του powerup στο έδαφος
    def sliding(self):
        if self.direction == c.RIGHT:
            self.x_vel = 3
```

```
        else:
            self.x_vel = -3


    def falling(self):
        if self.y_vel < self.max_y_vel:
            self.y_vel += self.gravity

# Powerup Milk το οποίο κάνει τον Ηρακλή μεγάλο
class Milk(Powerup):
    def __init__(self, x, y, name='mushroom'):
        super(Milk, self).__init__(x, y)
        self.setup_powerup(x, y, name, self.setup_frames, 'powerup')

    # Εισαγωγή εικόνας (frame) στη λίστα
    def setup_frames(self):
        self.frames.append(self.get_image(32, 74, 15, 16, 'powerup'))

    # Εκτέλεση της ενέργειας με βάση την κατάσταση που βρίσκεται
    def handle_state(self):
        if self.state == c.REVEAL:
            self.revealing()
        elif self.state == c.SLIDE:
            self.sliding()
        elif self.state == c.FALL:
            self.falling()

# LifeMilk
class LifeMilk(Milk):
    def __init__(self, x, y, name='1up_mushroom'):
        super(LifeMilk, self).__init__(x, y)
        self.setup_powerup(x, y, name, self.setup_frames, 'powerup')

    def setup_frames(self):
        self.frames.append(self.get_image(53, 74, 16, 16,
self.sprite_sheet))


# Powerup που επιτρέπει τον Ηρακλή να πυροβολεί
class FireApple(Powerup):
    def __init__(self, x, y, name=c.FIREFLOWER):
        super(FireApple, self).__init__(x, y)
        self.setup_powerup(x, y, name, self.setup_frames, 'powerup')

    # Εισαγωγή εικόνων (frame) στη λίστα
    def setup_frames(self):
        self.frames.append(
            self.get_image(72, 181, 18, 19, 'powerup'))
        self.frames.append(
            self.get_image(93, 181, 18, 19, 'powerup'))
        self.frames.append(
            self.get_image(114, 181, 18, 19, 'powerup'))
        self.frames.append(
            self.get_image(156, 181, 18, 19, 'powerup'))

    # Ελέγχει την συμπεριφορά του αντικειμένου με βάση την κατάστασή του
    def handle_state(self):
        if self.state == c.REVEAL:
            self.revealing()
        elif self.state == c.RESTING:
            self.resting()
```

```
    # Animation του apple όταν εμφανίζεται
    def revealing(self):
        self.rect.y += self.y_vel

        if self.rect.bottom <= self.box_height:
            self.rect.bottom = self.box_height
            self.state = c.RESTING

        self.animation()

    # Εκτελείται όταν το apple είναι ανοιχτλο
    def resting(self):
        self.animation()

    # Εναλλαγή εικόνων ώστε να δημιουργείται η ψευδαίσθηση της κίνησης
    def animation(self):
        if (self.current_time - self.animate_timer) > 30:
            if self.frame_index < 3:
                self.frame_index += 1
            else:
                self.frame_index = 0

            self.image = self.frames[self.frame_index]
            self.animate_timer = self.current_time

# Η κλάση Star αντιπροσωπεύει το powerup Star το οποίο κάνει τον Ηρακλή
άτρωτο
class Star(Powerup):
    def __init__(self, x, y, name='star'):
        super(Star, self).__init__(x, y)
        self.setup_powerup(x, y, name, self.setup_frames, 'item_objects')
        self.animate_timer = 0
        self.rect.y += 1
        self.gravity = .4

    # Εισαγωγή εικόνων (frames) στη λίστα
    def setup_frames(self):
        self.frames.append(self.get_image(1, 48, 15, 16, 'item_objects'))
        self.frames.append(self.get_image(17, 48, 15, 16, 'item_objects'))
        self.frames.append(self.get_image(33, 48, 15, 16, 'item_objects'))
        self.frames.append(self.get_image(49, 48, 15, 16, 'item_objects'))

    # Ελέγχει τη συμπεριφορά του αντικειμένου με βάση την κατάστασή του
    def handle_state(self):
        if self.state == c.REVEAL:
            self.revealing()
        elif self.state == c.BOUNCE:
            self.bouncing()

    # Αποκάλυψη του star από το brick
    def revealing(self):
        self.rect.y += self.y_vel

        if self.rect.bottom <= self.box_height:
            self.rect.bottom = self.box_height
            self.start_bounce(-2)
            self.state = c.BOUNCE

        self.animation()
```

```
    # Εναλλαγή εικόνων για τη δημιουργία του animation
    def animation(self):
        if (self.current_time - self.animate_timer) > 30:
            if self.frame_index < 3:
                self.frame_index += 1
            else:
                self.frame_index = 0
            self.animate_timer = self.current_time
            self.image = self.frames[self.frame_index]


    def start_bounce(self, vel):
        self.y_vel = vel

    # Εκτελείται όταν το star κινείται στο χώρο
    def bouncing(self):
        self.animation()

        if self.direction == c.LEFT:
            self.x_vel = -5
        else:
            self.x_vel = 5


# Κλάση που ναπιπροσωπεύει τις σφαίρες που πυρβολεί ο Ηρακλής
class FireBall(pg.sprite.Sprite):
    def __init__(self, x, y, facing_right, name=c.FIREBALL):
        super(FireBall, self).__init__()
        self.sprite_sheet = setup.GFX['item_objects']
        self.setup_frames()
        if facing_right:
            self.direction = c.RIGHT
            self.x_vel = 12
        else:
            self.direction = c.LEFT
            self.x_vel = -12
        self.y_vel = 10
        self.gravity = .9
        self.frame_index = 0
        self.animation_timer = 0
        self.state = c.FLYING
        self.image = self.frames[self.frame_index]
        self.rect = self.image.get_rect()
        self.rect.right = x
        self.rect.y = y
        self.name = name

    # Εισάγει τις εικόνες στη λίστα
    def setup_frames(self):
        self.frames = []

        self.frames.append(
            self.get_image(96, 144, 8, 8)) #Frame 1 of flying
        self.frames.append(
            self.get_image(104, 144, 8, 8))  #Frame 2 of Flying
        self.frames.append(
            self.get_image(96, 152, 8, 8))   #Frame 3 of Flying
        self.frames.append(
            self.get_image(104, 152, 8, 8))  #Frame 4 of flying
        self.frames.append(
            self.get_image(112, 144, 16, 16))    #frame 1 of exploding
```

```
        self.frames.append(
            self.get_image(112, 160, 16, 16))  #frame 2 of exploding
        self.frames.append(
            self.get_image(112, 176, 16, 16))  #frame 3 of exploding

    # Εξαγωγή των εικόνων από το sprite sheet
    def get_image(self, x, y, width, height):
        image = pg.Surface([width, height]).convert()
        rect = image.get_rect()

        image.blit(self.sprite_sheet, (0, 0), (x, y, width, height))
        image.set_colorkey(c.BLACK)
        image = pg.transform.scale(image,
(int(rect.width*c.SIZE_MULTIPLIER),int(rect.height*c.SIZE_MULTIPLIER)))
        return image

    # Κάνει update τη συμπεριφορά του fireball
    def update(self, game_info, viewport):
        self.current_time = game_info[c.CURRENT_TIME]
        self.handle_state()
        self.check_if_off_screen(viewport)

    # Ελέγχει τη συμπεριφορά τοτ αντικειμένου ανάλογα με την κατάστασή του
    def handle_state(self):
        if self.state == c.FLYING:
            self.animation()
        elif self.state == c.BOUNCING:
            self.animation()
        elif self.state == c.EXPLODING:
            self.animation()

    # Εναλλαγή εικόνων για το animation
    def animation(self):
        if self.state == c.FLYING or self.state == c.BOUNCING:
            if (self.current_time - self.animation_timer) > 200:
                if self.frame_index < 3:
                    self.frame_index += 1
                else:
                    self.frame_index = 0
                self.animation_timer = self.current_time
                self.image = self.frames[self.frame_index]


        elif self.state == c.EXPLODING:
            if (self.current_time - self.animation_timer) > 50:
                if self.frame_index < 6:
                    self.frame_index += 1
                    self.image = self.frames[self.frame_index]
                    self.animation_timer = self.current_time
                else:
                    self.kill()

    # Εναλλαγή κατάστασης του αντικειμένου σε κατάσταση EXPLODING
    def explode_transition(self):
        self.frame_index = 4
        centerx = self.rect.centerx
        self.image = self.frames[self.frame_index]
        self.rect.centerx = centerx
        self.state = c.EXPLODING

    # Εξαφάνιση εικόνας αντικειμένου από την οθόνη
```

```
    def check_if_off_screen(self, viewport):
        if (self.rect.x > viewport.right) or (self.rect.y >
viewport.bottom) \
            or (self.rect.right < viewport.x):
            self.kill()
# Η κλάση αυτή αντιπροσωπεύει το ψηφίο
class Digit(pg.sprite.Sprite):
    def __init__(self, image):
        super(Digit, self).__init__()
        self.image = image
        self.rect = image.get_rect()

# Κλάση που αντιπροσωπεύει το score που εμφανίζεται και εξαφανίζεται
σταδιακά
class Score(object):
    def __init__(self, x, y, score, flag_pole=False):
        self.x = x
        self.y = y
        if flag_pole:
            self.y_vel = -4
        else:
            self.y_vel = -3
        self.sprite_sheet = setup.GFX['item_objects']
        self.create_image_dict()
        self.score_string = str(score)
        self.create_digit_list()
        self.flag_pole_score = flag_pole

    # Δημιουργία ενός λεξικού που αντιστοιχεί τα  ψηφία 0-9 με τις
αντίστοιχες εικόνες
    def create_image_dict(self):
        self.image_dict = {}

        image0 = self.get_image(1, 168, 3, 8)
        image1 = self.get_image(5, 168, 3, 8)
        image2 = self.get_image(8, 168, 4, 8)
        image4 = self.get_image(12, 168, 4, 8)
        image5 = self.get_image(16, 168, 5, 8)
        image8 = self.get_image(20, 168, 4, 8)
        image9 = self.get_image(32, 168, 5, 8)
        image10 = self.get_image(37, 168, 6, 8)
        image11 = self.get_image(43, 168, 5, 8)

        self.image_dict['0'] = image0
        self.image_dict['1'] = image1
        self.image_dict['2'] = image2
        self.image_dict['4'] = image4
        self.image_dict['5'] = image5
        self.image_dict['8'] = image8
        self.image_dict['3'] = image9
        self.image_dict['7'] = image10
        self.image_dict['9'] = image11

    # Εξαγωγή εικόνας από το sprite sheet
    def get_image(self, x, y, width, height):
        image = pg.Surface([width, height]).convert()
        rect = image.get_rect()

        image.blit(self.sprite_sheet, (0, 0), (x, y, width, height))
        image.set_colorkey(c.BLACK)
        image =
```

```
pg.transform.scale(image,(int(rect.width*c.BRICK_SIZE_MULTIPLIER),int(rect.
height*c.BRICK_SIZE_MULTIPLIER)))
        return image

    # δημιουργεί μια σειρά εικόνων με ψηφία, με βάση μια συμβολοσειρά
    def create_digit_list(self):
        self.digit_list = []
        self.digit_group = pg.sprite.Group()

        for digit in self.score_string:
            self.digit_list.append(Digit(self.image_dict[digit]))

        self.set_rects_for_images()

    # Δημιουργεί τα rects για κάθε εικόνα με ψηφία
    def set_rects_for_images(self):
        for i, digit in enumerate(self.digit_list):
            digit.rect = digit.image.get_rect()
            digit.rect.x = self.x + (i * 10)
            digit.rect.y = self.y

    # Κάνει update στο score
    def update(self, score_list, level_info):
        for number in self.digit_list:
            number.rect.y += self.y_vel

        if score_list:
            self.check_to_delete_floating_scores(score_list, level_info)

        if self.flag_pole_score:
            if self.digit_list[0].rect.y <= 120:
                self.y_vel = 0

    # "Ζωγραφίζει¨ τις εικόνες του score στην οθόνη
    def draw(self, screen):
        for digit in self.digit_list:
            screen.blit(digit.image, digit.rect)

    def check_to_delete_floating_scores(self, score_list, level_info):
        for i, score in enumerate(score_list):
            if int(score.score_string) == 1000:
                if (score.y - score.digit_list[0].rect.y) > 130:
                    score_list.pop(i)

            else:
                if (score.y - score.digit_list[0].rect.y) > 75:
                    score_list.pop(i)
from __future__ import division


import pygame as pg
from .. import setup, tools
from .. import constants as c
from .. import game_sound
from .. components import hercules
from .. components import collider
from .. components import bricks
from .. components import coin_box
from .. components import enemies
from .. components import checkpoint
from .. components import flagpole
```

```
from .. components import info
from .. components import score
from .. components import castle_flag

# Η κλάση αντιπροσωπεύει το πρώτο level (πίστα) του παιχνιδιού
class Level1(tools._State):
    def __init__(self):
        tools._State.__init__(self)

    # Εκτελείται για την αρχικοποίηση
    def startup(self, current_time, persist):
        self.game_info = persist
        self.persist = self.game_info
        self.game_info[c.CURRENT_TIME] = current_time
        self.game_info[c.LEVEL_STATE] = c.NOT_FROZEN
        self.game_info[c.HERCULES_DEAD] = False

        self.state = c.NOT_FROZEN
        self.death_timer = 0
        self.flag_timer = 0
        self.flag_score = None
        self.flag_score_total = 0

        self.moving_score_list = []
        self.overhead_info_display = info.OverheadInfo(self.game_info,
c.LEVEL)
        self.sound_manager = game_sound.Sound(self.overhead_info_display)

        self.setup_background()
        self.setup_ground()
        self.setup_pipes()
        self.setup_steps()
        self.setup_bricks()
        self.setup_coin_boxes()
        self.setup_flag_pole()
        self.setup_enemies()
        self.setup_hercules()
        self.setup_checkpoints()
        self.setup_spritegroups()

    # Ορίζει την εικόνα του background και το viewport
    def setup_background(self):
        self.background = setup.GFX['level_1']
        self.back_rect = self.background.get_rect()
        self.background = pg.transform.scale(self.background,

(int(self.back_rect.width*c.BACKGROUND_MULTIPLER),

int(self.back_rect.height*c.BACKGROUND_MULTIPLER)))
        self.back_rect = self.background.get_rect()
        width = self.back_rect.width
        height = self.back_rect.height

        self.level = pg.Surface((width, height)).convert()
        self.level_rect = self.level.get_rect()
        self.viewport =
setup.SCREEN.get_rect(bottom=self.level_rect.bottom)
        self.viewport.x = self.game_info[c.CAMERA_START_X]

    # Δημιουργεί διάφανα rectangles προκειμένου τα αντικείμενα (Ηρακλής και
enemies) να
```

```
    # κοινούνται πάνω σε αυτό
    def setup_ground(self):
        ground_rect1 = collider.Collider(0, c.GROUND_HEIGHT,    2953, 60)
        ground_rect2 = collider.Collider(3048, c.GROUND_HEIGHT,  635, 60)
        ground_rect3 = collider.Collider(3819, c.GROUND_HEIGHT, 2735, 60)
        ground_rect4 = collider.Collider(6647, c.GROUND_HEIGHT, 2300, 60)

        self.ground_group = pg.sprite.Group(
                                                ground_rect1,ground_rect2,
                                                ground_rect3,ground_rect4
                                            )


    # "Σημειώνει" τους σωλήνες που υπάρχουν πάνω στο έδαφος ώστε να μπορούν
    # να αλληλεπιδρούν τα αντικείμενα (Ηρακλής και enemies με αυτά)
    def setup_pipes(self):
        pipe1 = collider.Collider(1202, 452, 83, 82)
        pipe2 = collider.Collider(1631, 409, 83, 140)
        pipe3 = collider.Collider(1973, 366, 83, 170)
        pipe4 = collider.Collider(2445, 366, 83, 170)
        pipe5 = collider.Collider(6989, 452, 83, 82)
        pipe6 = collider.Collider(7675, 452, 83, 82)

        self.pipe_group = pg.sprite.Group(pipe1, pipe2, pipe3, pipe4,
pipe5, pipe6)

    # "Σημειώνει" τσ εμπόδια (τουβλάκια) που υπάρχουν πάνω στο έδαφος ώστε
να μπορούν
    # να αλληλεπιδρούν τα αντικείμενα (Ηρακλής και enemies με αυτά)
    def setup_steps(self):
        step1 = collider.Collider(5745, 495, 40, 44)
        step2 = collider.Collider(5788, 452, 40, 44)
        step3 = collider.Collider(5831, 409, 40, 44)
        step4 = collider.Collider(5874, 366, 40, 176)

        step5 = collider.Collider(6001, 366, 40, 176)
        step6 = collider.Collider(6044, 408, 40, 40)
        step7 = collider.Collider(6087, 452, 40, 40)
        step8 = collider.Collider(6130, 495, 40, 40)

        step9 = collider.Collider(6345, 495, 40, 40)
        step10 = collider.Collider(6388, 452, 40, 40)
        step11 = collider.Collider(6431, 409, 40, 40)
        step12 = collider.Collider(6474, 366, 40, 40)
        step13 = collider.Collider(6517, 366, 40, 176)

        step14 = collider.Collider(6644, 366, 40, 176)
        step15 = collider.Collider(6687, 408, 40, 40)
        step16 = collider.Collider(6728, 452, 40, 40)
        step17 = collider.Collider(6771, 495, 40, 40)

        step18 = collider.Collider(7760, 495, 40, 40)
        step19 = collider.Collider(7803, 452, 40, 40)
        step20 = collider.Collider(7845, 409, 40, 40)
        step21 = collider.Collider(7888, 366, 40, 40)
        step22 = collider.Collider(7931, 323, 40, 40)
        step23 = collider.Collider(7974, 280, 40, 40)
        step24 = collider.Collider(8017, 237, 40, 40)
        step25 = collider.Collider(8060, 194, 40, 40)
        step26 = collider.Collider(8103, 194, 40, 360)
```

```
        step27 = collider.Collider(8488, 495, 40, 40)

        self.step_group = pg.sprite.Group(step1,  step2,
                                          step3,  step4,
                                          step5,  step6,
                                          step7,  step8,
                                          step9,  step10,
                                          step11, step12,
                                          step13, step14,
                                          step15, step16,
                                          step17, step18,
                                          step19, step20,
                                          step21, step22,
                                          step23, step24,
                                          step25, step26,
                                          step27)

    # Δημιουργεί όλα τα breakable τούβλα της συγκεκριμένης πίστας
    def setup_bricks(self):

        self.coin_group = pg.sprite.Group()
        self.powerup_group = pg.sprite.Group()
        self.brick_pieces_group = pg.sprite.Group()

        brick1  = bricks.Brick(858,  365)
        brick2  = bricks.Brick(944,  365)
        brick3  = bricks.Brick(1030, 365)
        brick4  = bricks.Brick(3299, 365)
        brick5  = bricks.Brick(3385, 365)
        brick6  = bricks.Brick(3430, 193)
        brick7  = bricks.Brick(3473, 193)
        brick8  = bricks.Brick(3516, 193)
        brick9  = bricks.Brick(3559, 193)
        brick10 = bricks.Brick(3602, 193)
        brick11 = bricks.Brick(3645, 193)
        brick12 = bricks.Brick(3688, 193)
        brick13 = bricks.Brick(3731, 193)
        brick14 = bricks.Brick(3901, 193)
        brick15 = bricks.Brick(3944, 193)
        brick16 = bricks.Brick(3987, 193)
        brick17 = bricks.Brick(4030, 365, c.SIXCOINS, self.coin_group)
        brick18 = bricks.Brick(4287, 365)
        brick19 = bricks.Brick(4330, 365, c.STAR, self.powerup_group)
        brick20 = bricks.Brick(5058, 365)
        brick21 = bricks.Brick(5187, 193)
        brick22 = bricks.Brick(5230, 193)
        brick23 = bricks.Brick(5273, 193)
        brick24 = bricks.Brick(5488, 193)
        brick25 = bricks.Brick(5574, 193)
        brick26 = bricks.Brick(5617, 193)
        brick27 = bricks.Brick(5531, 365)
        brick28 = bricks.Brick(5574, 365)
        brick29 = bricks.Brick(7202, 365)
        brick30 = bricks.Brick(7245, 365)
        brick31 = bricks.Brick(7331, 365)

        self.brick_group = pg.sprite.Group(brick1,  brick2,
                                           brick3,  brick4,
                                           brick5,  brick6,
                                           brick7,  brick8,
                                           brick9,  brick10,
```

```
                                            brick11, brick12,
                                            brick13, brick14,
                                            brick15, brick16,
                                            brick17, brick18,
                                            brick19, brick20,
                                            brick21, brick22,
                                            brick23, brick24,
                                            brick25, brick26,
                                            brick27, brick28,
                                            brick29, brick30,
                                            brick31)

    # Δημιουργεί όλα τα coin boxes που μπορεί να περιέχουν και mushroom ή
flower
    def setup_coin_boxes(self):
        coin_box1  = coin_box.Coin_box(685, 365, c.COIN, self.coin_group)
        coin_box2  = coin_box.Coin_box(901, 365, c.MUSHROOM,
self.powerup_group)
        coin_box3  = coin_box.Coin_box(987, 365, c.COIN, self.coin_group)
        coin_box4  = coin_box.Coin_box(943, 193, c.COIN, self.coin_group)
        coin_box5  = coin_box.Coin_box(3342, 365, c.MUSHROOM,
self.powerup_group)
        coin_box6  = coin_box.Coin_box(4030, 193, c.COIN, self.coin_group)
        coin_box7  = coin_box.Coin_box(4544, 365, c.COIN, self.coin_group)
        coin_box8  = coin_box.Coin_box(4672, 365, c.COIN, self.coin_group)
        coin_box9  = coin_box.Coin_box(4672, 193, c.MUSHROOM,
self.powerup_group)
        coin_box10 = coin_box.Coin_box(4800, 365, c.COIN, self.coin_group)
        coin_box11 = coin_box.Coin_box(5531, 193, c.COIN, self.coin_group)
        coin_box12 = coin_box.Coin_box(7288, 365, c.COIN, self.coin_group)

        self.coin_box_group = pg.sprite.Group(coin_box1,  coin_box2,
                                              coin_box3,  coin_box4,
                                              coin_box5,  coin_box6,
                                              coin_box7,  coin_box8,
                                              coin_box9,  coin_box10,
                                              coin_box11, coin_box12)

    # Δημιουγεί τον ιστό και την σημεία πάνω στον ιστό στο τέλος της πίστας
    def setup_flag_pole(self):
        self.flag = flagpole.Flag(8505, 100)

        pole0 = flagpole.Pole(8505, 97)
        pole1 = flagpole.Pole(8505, 137)
        pole2 = flagpole.Pole(8505, 177)
        pole3 = flagpole.Pole(8505, 217)
        pole4 = flagpole.Pole(8505, 257)
        pole5 = flagpole.Pole(8505, 297)
        pole6 = flagpole.Pole(8505, 337)
        pole7 = flagpole.Pole(8505, 377)
        pole8 = flagpole.Pole(8505, 417)
        pole9 = flagpole.Pole(8505, 450)

        finial = flagpole.Finial(8507, 97)

        self.flag_pole_group = pg.sprite.Group(self.flag,
                                               finial,
                                               pole0,
                                               pole1,
                                               pole2,
                                               pole3,
```

```
                                                pole4,
                                                pole5,
                                                pole6,
                                                pole7,
                                                pole8,
                                                pole9)

    # Δημιουργεί όλους τους enemies και τους διατηρεί σε μία λίστα
    def setup_enemies(self):
        bull0 = enemies.Bull()
        bull1 = enemies.Bull()
        bull2 = enemies.Bull()
        bull3 = enemies.Bull()
        bull4 = enemies.Bull(193)
        bull5 = enemies.Bull(193)
        bull6 = enemies.Bull()
        bull8 = enemies.Bull()
        bull9 = enemies.Bull()
        bull10 = enemies.Bull()
        bull12 = enemies.Bull()
        bull14 = enemies.Bull()
        bull16 = enemies.Bull()


        enemy_group1 = pg.sprite.Group(bull0)
        enemy_group2 = pg.sprite.Group(bull1)
        enemy_group3 = pg.sprite.Group(bull2, bull3)
        enemy_group4 = pg.sprite.Group(bull4, bull5)
        enemy_group5 = pg.sprite.Group(bull6)
        enemy_group6 = pg.sprite.Group(bull16)
        enemy_group7 = pg.sprite.Group(bull8, bull9)
        enemy_group8 = pg.sprite.Group(bull10)
        enemy_group9 = pg.sprite.Group(bull12)
        enemy_group10 = pg.sprite.Group(bull14)


        self.enemy_group_list = [enemy_group1,
                                 enemy_group2,
                                 enemy_group3,
                                 enemy_group4,
                                 enemy_group5,
                                 enemy_group6,
                                 enemy_group7,
                                 enemy_group8,
                                 enemy_group9,
                                 enemy_group10]

    # Δημιουργεί τον χαρακτήρα του Ηρακλή και τον τοποθετεί στην αρχή του
level 1
    def setup_hercules(self):
        self.hercules = hercules.Hercules()
        self.hercules.rect.x = self.viewport.x + 110
        self.hercules.rect.bottom = c.GROUND_HEIGHT

    # Δημιουργεί invisible checkpoints με τα οποία όταν ο Ηρακλής έρθει σε
επαφή θα
    # προκαλέσει τη δημιουργία εχθρών από την self.enemy_group_list
    def setup_checkpoints(self):
        check1 = checkpoint.Checkpoint(510, "1")
        check2 = checkpoint.Checkpoint(1400, '2')
        check3 = checkpoint.Checkpoint(1740, '3')
```

```
        check4 = checkpoint.Checkpoint(3080, '4')
        check5 = checkpoint.Checkpoint(3750, '5')
        check6 = checkpoint.Checkpoint(4150, '6')
        check7 = checkpoint.Checkpoint(4470, '7')
        check8 = checkpoint.Checkpoint(4950, '8')
        check9 = checkpoint.Checkpoint(5100, '9')
        check10 = checkpoint.Checkpoint(6800, '10')
        check11 = checkpoint.Checkpoint(8504, '11', 5, 6)
        check12 = checkpoint.Checkpoint(8775, '12')
        check13 = checkpoint.Checkpoint(2740, 'secret_mushroom', 360, 40,
12)

        self.check_point_group = pg.sprite.Group(check1, check2, check3,
                                                 check4, check5, check6,
                                                 check7, check8, check9,
                                                 check10, check11, check12,
                                                 check13)

    # Δημιουργούμε sprite groups για να τα ομαδοποιήσουμε
    # και να μπορούμε να τα χειριστούμε με μεγαλύτερη ευκολία
    def setup_spritegroups(self):
        self.sprites_about_to_die_group = pg.sprite.Group()
        self.shell_group = pg.sprite.Group()
        self.enemy_group = pg.sprite.Group()

        self.ground_step_pipe_group =
pg.sprite.Group(self.ground_group,self.pipe_group,self.step_group)

        self.hercules_and_enemy_group = pg.sprite.Group(self.hercules,
self.enemy_group)

    # Κάνει update ολόκληρο το level. Καλειται από την tools.Control κλάση
    def update(self, surface, keys, current_time):
        self.game_info[c.CURRENT_TIME] = self.current_time = current_time
        self.handle_states(keys)
        self.check_if_time_out()
        self.blit_everything(surface)
        self.sound_manager.update(self.game_info, self.hercules)


    # Εκτελείται η αντίστοιχη λειτουργία ανάλογα με την κατάσταση του level
    def handle_states(self, keys):
        if self.state == c.FROZEN:
            self.update_during_transition_state(keys)
        elif self.state == c.NOT_FROZEN:
            self.update_all_sprites(keys)
        elif self.state == c.IN_CASTLE:
            self.update_while_in_castle()
        elif self.state == c.FLAG_AND_FIREWORKS:
            self.update_flag_and_fireworks()

    # Κάνει upadate τον Ηρακλή καθώς αυτός βρίσκεται σε transition
κατάσταση
    # γίνεται μεγάλος΄η μικρός ή πεθαίνει ή αποκτά πυροβόλο
    def update_during_transition_state(self, keys):

        self.hercules.update(keys, self.game_info, self.powerup_group)

        for score in self.moving_score_list:
            score.update(self.moving_score_list, self.game_info)
        if self.flag_score:
```

```
            self.flag_score.update(None, self.game_info)
            self.check_to_add_flag_score()
        # Κάνει update τα coins και την σημαία στο τέλος του level
        self.coin_box_group.update(self.game_info)
        self.flag_pole_group.update(self.game_info)
        # Εξετάζει μήπως ο Ηρακλής δεν είναι πλέον σε κατάσταση transition
        self.check_if_hercules_in_transition_state()
        # Εξετάζει αν ο Ηρακλής έχει κατεβάσει τη σημαία
        self.check_flag()
        # Εξετάζει αν ο Ηρακλής έχει σκοτωθεί
        self.check_for_hercules_death()
        # Τυπωνει τις overhead πληροφορίες στην οθόνη
        self.overhead_info_display.update(self.game_info, self.hercules)

    # Εξετάζει αν ο Ηρακλής είναι σε κατάσταση transition και θέτει την
κατάσταση του level σε FROZEN
    # Διαφορετικά θέτει την κατάσταση του level σε NOT FROZEN
    def check_if_hercules_in_transition_state(self):
        if self.hercules.in_transition_state:
            self.game_info[c.LEVEL_STATE] = self.state = c.FROZEN
        elif self.hercules.in_transition_state == False:
            if self.state == c.FROZEN:
                self.game_info[c.LEVEL_STATE] = self.state = c.NOT_FROZEN

    # Το level είναι σε κατάσταση NOT FROZEN οπότε όλα τα sprites
    # ανανεώνονται μαζί και η θέση τους
    def update_all_sprites(self, keys):
        self.hercules.update(keys, self.game_info, self.powerup_group)
        for score in self.moving_score_list:
            score.update(self.moving_score_list, self.game_info)
        if self.flag_score:
            self.flag_score.update(None, self.game_info)
            self.check_to_add_flag_score()
        self.flag_pole_group.update()
        # Ελεγχος αν ο Ηρακλής περασε από τα checkpoints
        self.check_points_check()
        self.enemy_group.update(self.game_info)
        self.sprites_about_to_die_group.update(self.game_info,
self.viewport)
        self.shell_group.update(self.game_info)
        self.brick_group.update()
        self.coin_box_group.update(self.game_info)
        self.powerup_group.update(self.game_info, self.viewport)
        self.coin_group.update(self.game_info, self.viewport)
        self.brick_pieces_group.update()
        self.adjust_sprite_positions()
        self.check_if_hercules_in_transition_state()
        self.check_for_hercules_death()
        self.update_viewport()
        self.overhead_info_display.update(self.game_info, self.hercules)

    # Εξετάζει αν ο Ηρακλής έχει περάσει από κάποιο checkpoint ώστε να
    # δημιουργηθούν οι enemies
    def check_points_check(self):
        checkpoint = pg.sprite.spritecollideany(self.hercules,
self.check_point_group)

        # Αν έχει περάσει από checkpoint
        if checkpoint:
            checkpoint.kill()
            # Τα checkpoint 1 εως 10 πυροδοτούν την δημιουργία των enemies
```

```
που έχουμε
            # ορίσει παραπάνω. Οπότε μόλις περάσουμε ένα από αυτά τα
checkpoints δημιουργούνται
            # τα αντίστοιχα enemies
            for i in range(1,11):
                if checkpoint.name == str(i):
                    for index, enemy in enumerate(self.enemy_group_list[i -
1]):
                        enemy.rect.x = self.viewport.right + (index * 60)
                    self.enemy_group.add(self.enemy_group_list[i-1])

            # Το checkpoint 11 αφορά τον ιστό στο τέλος του level
            if checkpoint.name == '11':
                self.hercules.state = c.FLAGPOLE
                self.hercules.invincible = False
                self.hercules.flag_pole_right = checkpoint.rect.right
                if self.hercules.rect.bottom < self.flag.rect.y:
                    self.hercules.rect.bottom = self.flag.rect.y
                self.flag.state = c.SLIDE_DOWN
                # Εμφανίζει το σκορ που κέρδισε ο Ηρακλής στον ιστό
                self.create_flag_points()

            # Το checkpoint 12 σημαίνει ότι ο Ηρακλής έφτασε στο κάστρο και
            # έχει ολοκληρωθεί το level
            elif checkpoint.name == '12':
                self.state = c.IN_CASTLE
                self.hercules.kill()
                self.hercules.state == c.STAND
                self.hercules.in_castle = True
                self.overhead_info_display.state = c.FAST_COUNT_DOWN

            # Σε διαφορετική περίπτωση σχετίζεται με το κρυφή live
            elif checkpoint.name == 'secret_mushroom' and
self.hercules.y_vel < 0:
                mushroom_box = coin_box.Coin_box(checkpoint.rect.x,
                                                 checkpoint.rect.bottom - 40,
                                                 '1up_mushroom',
                                                 self.powerup_group)
                mushroom_box.start_bump(self.moving_score_list)
                self.coin_box_group.add(mushroom_box)

                self.hercules.y_vel = 7
                self.hercules.rect.y = mushroom_box.rect.bottom
                self.hercules.state = c.FALL

            self.hercules_and_enemy_group.add(self.enemy_group)

    # Δημιουργεί τους πόντους που εμφανίζνται όταν ο Ηρακλής πηδάει στον
ιστό
    # Οι πόντοι εξαρτώνται από το πόσο ψυλά φτάνει στον ιστό
    def create_flag_points(self):
        x = 8518
        y = c.GROUND_HEIGHT - 60
        hervcules_bottom = self.hercules.rect.bottom

        if hervcules_bottom > (c.GROUND_HEIGHT - 40 - 40):
            self.flag_score = score.Score(x, y, 100, True)
            self.flag_score_total = 100
        elif hervcules_bottom > (c.GROUND_HEIGHT - 40 - 160):
            self.flag_score = score.Score(x, y, 400, True)
            self.flag_score_total = 400
```

```
        elif hervcules_bottom > (c.GROUND_HEIGHT - 40 - 240):
            self.flag_score = score.Score(x, y, 800, True)
            self.flag_score_total = 800
        elif hervcules_bottom > (c.GROUND_HEIGHT - 40 - 360):
            self.flag_score = score.Score(x, y, 2000, True)
            self.flag_score_total = 2000
        else:
            self.flag_score = score.Score(x, y, 5000, True)
            self.flag_score_total = 5000

    def adjust_sprite_positions(self):
        self.adjust_hercules_position()
        self.adjust_enemy_position()
        self.adjust_powerup_position()

    # Προσαρμόζει την θέση του Ηρακλή ανάλογα με την ταχύτητά του και τις
πιθανές συγκρούσεις
    def adjust_hercules_position(self):
        self.last_x_position = self.hercules.rect.right
        self.hercules.rect.x += round(self.hercules.x_vel)
        self.check_hercules_x_collisions()

        if self.hercules.in_transition_state == False:
            self.hercules.rect.y += round(self.hercules.y_vel)
            self.check_hercules_y_collisions()

        if self.hercules.rect.x < (self.viewport.x + 5):
            self.hercules.rect.x = (self.viewport.x + 5)


    # Εξετάζει αν ο Ηρακλής συγκρούεται με ενα αντικείμενο κατά την
οριζόντια κίνηση
    def check_hercules_x_collisions(self):
        collider = pg.sprite.spritecollideany(self.hercules,
self.ground_step_pipe_group)
        coin_box = pg.sprite.spritecollideany(self.hercules,
self.coin_box_group)
        brick = pg.sprite.spritecollideany(self.hercules, self.brick_group)
        enemy = pg.sprite.spritecollideany(self.hercules, self.enemy_group)
        powerup = pg.sprite.spritecollideany(self.hercules,
self.powerup_group)

        if coin_box:
            self.adjust_hercules_for_x_collisions(coin_box)

        elif brick:
            self.adjust_hercules_for_x_collisions(brick)

        elif collider:
            self.adjust_hercules_for_x_collisions(collider)

        # Αν έρθει σε επαφή με εχθρό
        elif enemy:
            # Ο Ηρακλής είναι σε κατάσταση ανίκητος (μετά από αστέρι)
            if self.hercules.invincible:
                setup.SFX['kick'].play()
                self.game_info[c.SCORE] += 100
                self.moving_score_list.append(
                    score.Score(self.hercules.rect.right - self.viewport.x,
                                self.hercules.rect.y, 100))
                enemy.kill()
```

```
                enemy.start_death_jump(c.RIGHT)
                self.sprites_about_to_die_group.add(enemy)
            # Ο Ηρακλής είναι μεγαλος οπότε μικραίνει
            elif self.hercules.big:
                setup.SFX['pipe'].play()
                self.hercules.fire = False
                self.hercules.y_vel = -1
                self.hercules.state = c.BIG_TO_SMALL
                self.convert_fireflowers_to_mushrooms()
            elif self.hercules.hurt_invincible:
                pass
            # Ο Ηρακλής είναι μικρός οπότε χάνει μία ζωή
            else:
                self.hercules.start_death_jump(self.game_info)
                self.state = c.FROZEN

        # Ο Ηρακλής έρχεται σε επαφή με κάποιο powerup
        elif powerup:
            # Αστέρι
            if powerup.name == c.STAR:
                self.game_info[c.SCORE] += 1000

                self.moving_score_list.append(
                    score.Score(self.hercules.rect.centerx -
self.viewport.x,
                                self.hercules.rect.y, 1000))
                self.hercules.invincible = True
                self.hercules.invincible_start_timer = self.current_time
            # Μανιτάρι που μεγαλώνει
            elif powerup.name == c.MUSHROOM:
                setup.SFX['powerup'].play()
                self.game_info[c.SCORE] += 1000
                self.moving_score_list.append(
                    score.Score(self.hercules.rect.centerx -
self.viewport.x,
                                self.hercules.rect.y - 20, 1000))

                self.hercules.y_vel = -1
                self.hercules.state = c.SMALL_TO_BIG
                self.hercules.in_transition_state = True
                self.convert_mushrooms_to_fireflowers()
            # Μαντιτάρι που κερδίζει ζωή
            elif powerup.name == c.LIFE_MUSHROOM:
                self.moving_score_list.append(
                    score.Score(powerup.rect.right - self.viewport.x,
                                powerup.rect.y,
                                c.ONEUP))

                self.game_info[c.LIVES] += 1
                setup.SFX['one_up'].play()
            # Fire flower που του δίνει πυρομαχικά
            elif powerup.name == c.FIREFLOWER:
                setup.SFX['powerup'].play()
                self.game_info[c.SCORE] += 1000
                self.moving_score_list.append(
                    score.Score(self.hercules.rect.centerx -
self.viewport.x,
                                self.hercules.rect.y, 1000))

                if self.hercules.big and self.hercules.fire == False:
                    self.hercules.state = c.BIG_TO_FIRE
```

```
                    self.hercules.in_transition_state = True
                elif self.hercules.big == False:
                    self.hercules.state = c.SMALL_TO_BIG
                    self.hercules.in_transition_state = True
                    self.convert_mushrooms_to_fireflowers()

            if powerup.name != c.FIREBALL:
                powerup.kill()

    # Όταν ο Ηρακλής γίνει μεγάλος τότε μετατρέπει όλα τα mushrooms σε
fireflowers
    def convert_mushrooms_to_fireflowers(self):
        for brick in self.brick_group:
            if brick.contents == c.MUSHROOM:
                brick.contents = c.FIREFLOWER
        for coin_box in self.coin_box_group:
            if coin_box.contents == c.MUSHROOM:
                coin_box.contents = c.FIREFLOWER

    # Όταν ο Ηρακλής γίνει μικρός τότε μετατρέπει όλα τα fireflowers σε
mushroom
    def convert_fireflowers_to_mushrooms(self):
        for brick in self.brick_group:
            if brick.contents == c.FIREFLOWER:
                brick.contents = c.MUSHROOM
        for coin_box in self.coin_box_group:
            if coin_box.contents == c.FIREFLOWER:
                coin_box.contents = c.MUSHROOM

    # Τοποθετεί τον Ηρακλή δίπλα στον collider
    def adjust_hercules_for_x_collisions(self, collider):
        if self.hercules.rect.x < collider.rect.x:
            self.hercules.rect.right = collider.rect.left
        else:
            self.hercules.rect.left = collider.rect.right

        self.hercules.x_vel = 0


    # Εξετάζει αν ο Ηρακλής έρχεται σε επαφή με άλλα αντικείμενα κατά την
κίνηση
    # στον κάθετο άξονα
    def check_hercules_y_collisions(self):

        ground_step_or_pipe = pg.sprite.spritecollideany(self.hercules,
self.ground_step_pipe_group)
        enemy = pg.sprite.spritecollideany(self.hercules, self.enemy_group)
        brick = pg.sprite.spritecollideany(self.hercules, self.brick_group)
        coin_box = pg.sprite.spritecollideany(self.hercules,
self.coin_box_group)
        powerup = pg.sprite.spritecollideany(self.hercules,
self.powerup_group)

        brick, coin_box = self.prevent_collision_conflict(brick, coin_box)

        if coin_box:
            self.adjust_hercules_for_y_coin_box_collisions(coin_box)

        elif brick:
            self.adjust_hercules_for_y_brick_collisions(brick)
```

```
        elif ground_step_or_pipe:

self.adjust_hercules_for_y_ground_pipe_collisions(ground_step_or_pipe)

        elif enemy:
            if self.hercules.invincible:
                setup.SFX['kick'].play()
                enemy.kill()
                self.sprites_about_to_die_group.add(enemy)
                enemy.start_death_jump(c.RIGHT)
            else:
                self.adjust_hercules_for_y_enemy_collisions(enemy)

        elif powerup:
            if powerup.name == c.STAR:
                setup.SFX['powerup'].play()
                powerup.kill()
                self.hercules.invincible = True
                self.hercules.invincible_start_timer = self.current_time

        self.test_if_hercules_is_falling()


    def prevent_collision_conflict(self, obstacle1, obstacle2):
        if obstacle1 and obstacle2:
            obstacle1_distance = self.hercules.rect.centerx -
obstacle1.rect.centerx
            if obstacle1_distance < 0:
                obstacle1_distance *= -1
            obstacle2_distance = self.hercules.rect.centerx -
obstacle2.rect.centerx
            if obstacle2_distance < 0:
                obstacle2_distance *= -1

            if obstacle1_distance < obstacle2_distance:
                obstacle2 = False
            else:
                obstacle1 = False

        return obstacle1, obstacle2

    # Σύγκρουση του Ηρακλή με τα coin boxes κατά τον y άξονα. Αν το coin
box είναι σε κατασταση Resting και
    # περιέχει κάποιο coin τότε αυξάνεται το πληθος των coin κατά 1 και το
box παίρνει την κατάσταση OPENED
    def adjust_hercules_for_y_coin_box_collisions(self, coin_box):
        if self.hercules.rect.y > coin_box.rect.y:
            if coin_box.state == c.RESTING:
                if coin_box.contents == c.COIN:
                    self.game_info[c.SCORE] += 200
                    coin_box.start_bump(self.moving_score_list)
                    self.game_info[c.COIN_TOTAL] += 1
                else:
                    coin_box.start_bump(self.moving_score_list)

            elif coin_box.state == c.OPENED:
                pass
            setup.SFX['bump'].play()
            self.hercules.y_vel = 7
            self.hercules.rect.y = coin_box.rect.bottom
            self.hercules.state = c.FALL
```

```
        else:
            self.hercules.y_vel = 0
            self.hercules.rect.bottom = coin_box.rect.top
            self.hercules.state = c.WALK

    # Εκτελείται όταν ο Ηρακλής συγκρούεται με ένα brick (τούβλο)
    def adjust_hercules_for_y_brick_collisions(self, brick):
        if self.hercules.rect.y > brick.rect.y:
            if brick.state == c.RESTING:
                if self.hercules.big and brick.contents is None:
                    setup.SFX['brick_smash'].play()
                    self.check_if_enemy_on_brick(brick)
                    brick.kill()
                    self.brick_pieces_group.add(
                        bricks.BrickPiece(brick.rect.x,
                                          brick.rect.y -
(brick.rect.height/2),
                                          -2, -12),
                        bricks.BrickPiece(brick.rect.right,
                                          brick.rect.y -
(brick.rect.height/2),
                                          2, -12),
                        bricks.BrickPiece(brick.rect.x,
                                          brick.rect.y,
                                          -2, -6),
                        bricks.BrickPiece(brick.rect.right,
                                          brick.rect.y,
                                          2, -6))
                else:
                    setup.SFX['bump'].play()
                    if brick.coin_total > 0:
                        self.game_info[c.COIN_TOTAL] += 1
                        self.game_info[c.SCORE] += 200
                    self.check_if_enemy_on_brick(brick)
                    brick.start_bump(self.moving_score_list)
            elif brick.state == c.OPENED:
                setup.SFX['bump'].play()
            self.hercules.y_vel = 7
            self.hercules.rect.y = brick.rect.bottom
            self.hercules.state = c.FALL

        else:
            self.hercules.y_vel = 0
            self.hercules.rect.bottom = brick.rect.top
            self.hercules.state = c.WALK

    # Αν υπάρχει κάποιος εχθρός ο οποίος είναι πάνω στο brick το οποίο
σπάει ή κουνάει
    # ο Ηρακλής τότε ο εχθρός πεθαίνει
    def check_if_enemy_on_brick(self, brick):
        brick.rect.y -= 5

        enemy = pg.sprite.spritecollideany(brick, self.enemy_group)

        if enemy:
            setup.SFX['kick'].play()
            self.game_info[c.SCORE] += 100
            self.moving_score_list.append(
                score.Score(enemy.rect.centerx - self.viewport.x,
                            enemy.rect.y,
                            100))
```

```
            enemy.kill()
            self.sprites_about_to_die_group.add(enemy)
            if self.hercules.rect.centerx > brick.rect.centerx:
                enemy.start_death_jump('right')
            else:
                enemy.start_death_jump('left')

        brick.rect.y += 5


    # Ο Ηρακλής έρχεται σε επαφή με σωλήνα
    def adjust_hercules_for_y_ground_pipe_collisions(self, collider):

        if collider.rect.bottom > self.hercules.rect.bottom:
            self.hercules.y_vel = 0
            self.hercules.rect.bottom = collider.rect.top
            if self.hercules.state == c.END_OF_LEVEL_FALL:
                self.hercules.state = c.WALKING_TO_CASTLE
            else:
                self.hercules.state = c.WALK
        elif collider.rect.top < self.hercules.rect.top:
            self.hercules.y_vel = 7
            self.hercules.rect.top = collider.rect.bottom
            self.hercules.state = c.FALL


    def test_if_hercules_is_falling(self):
        self.hercules.rect.y += 1
        test_collide_group = pg.sprite.Group(self.ground_step_pipe_group,
                                             self.brick_group,
                                             self.coin_box_group)


        if pg.sprite.spritecollideany(self.hercules, test_collide_group) is
None:
            if self.hercules.state != c.JUMP \
                and self.hercules.state != c.DEATH_JUMP \
                and self.hercules.state != c.SMALL_TO_BIG \
                and self.hercules.state != c.BIG_TO_FIRE \
                and self.hercules.state != c.BIG_TO_SMALL \
                and self.hercules.state != c.FLAGPOLE \
                and self.hercules.state != c.WALKING_TO_CASTLE \
                and self.hercules.state != c.END_OF_LEVEL_FALL:
                self.hercules.state = c.FALL
            elif self.hercules.state == c.WALKING_TO_CASTLE or \
                self.hercules.state == c.END_OF_LEVEL_FALL:
                self.hercules.state = c.END_OF_LEVEL_FALL

        self.hercules.rect.y -= 1

    # Ο Ηρακλής έρχεται σε επαφή με τον εχθρό κατά τον άξονα των y
    def adjust_hercules_for_y_enemy_collisions(self, enemy):

        if self.hercules.y_vel > 0:
            setup.SFX['stomp'].play()
            self.game_info[c.SCORE] += 100
            self.moving_score_list.append(
                score.Score(enemy.rect.centerx - self.viewport.x,
                            enemy.rect.y, 100))
            enemy.state = c.JUMPED_ON
            enemy.kill()
```

```
            if enemy.name == c.BULL or enemy.name == c.LION:
                enemy.death_timer = self.current_time
                self.sprites_about_to_die_group.add(enemy)
            elif enemy.name == c.KOOPA:
                self.shell_group.add(enemy)

            self.hercules.rect.bottom = enemy.rect.top
            self.hercules.state = c.JUMP
            self.hercules.y_vel = -7


    # Μετακινεί ολους τους εχθρούς κατά τον άξονα X και Y και εξετάζει αν
έγιναν συγκρούσεις μετά την κίνηση
    # του εχθρού
    def adjust_enemy_position(self):

        for enemy in self.enemy_group:
            enemy.rect.x += enemy.x_vel
            self.check_enemy_x_collisions(enemy)

            enemy.rect.y += enemy.y_vel
            self.check_enemy_y_collisions(enemy)
            self.delete_if_off_screen(enemy)

    # Εξετάζει τις συγκρούσεις των εχθρών μεταξύ τους καθως και με σωλήνες
η steps
    def check_enemy_x_collisions(self, enemy):

        enemy.kill()

        collider = pg.sprite.spritecollideany(enemy,
self.ground_step_pipe_group)
        enemy_collider = pg.sprite.spritecollideany(enemy,
self.enemy_group)

        # Συγκρουση του εχθρού με έναν σωλήνα ή step οπότε αλλάζει
κατεύθυνση και
        # πηγαίνει προς την αντίθετη πλευρά
        if collider:
            if enemy.direction == c.RIGHT:
                enemy.rect.right = collider.rect.left
                enemy.direction = c.LEFT
                enemy.x_vel = -2
            elif enemy.direction == c.LEFT:
                enemy.rect.left = collider.rect.right
                enemy.direction = c.RIGHT
                enemy.x_vel = 2

        # σύγκρουση του εχθρού με κάποιον άλλον εχθρό οπότε και οι δύο
εχθροί θα κινηθούν προς την
        # αντίθετη κατεύθυνση
        elif enemy_collider:
            if enemy.direction == c.RIGHT:
                enemy.rect.right = enemy_collider.rect.left
                enemy.direction = c.LEFT
                enemy_collider.direction = c.RIGHT
                enemy.x_vel = -2
                enemy_collider.x_vel = 2
            elif enemy.direction == c.LEFT:
                enemy.rect.left = enemy_collider.rect.right
                enemy.direction = c.RIGHT
```

```
                enemy_collider.direction = c.LEFT
                enemy.x_vel = 2
                enemy_collider.x_vel = -2

        self.enemy_group.add(enemy)
        self.hercules_and_enemy_group.add(self.enemy_group)

    # Ελέγχει τις συγκρούσεις των εχθρών ως προς τον άξονα των y
    def check_enemy_y_collisions(self, enemy):

        collider = pg.sprite.spritecollideany(enemy,
self.ground_step_pipe_group)
        brick = pg.sprite.spritecollideany(enemy, self.brick_group)
        coin_box = pg.sprite.spritecollideany(enemy, self.coin_box_group)

        if collider:
            if enemy.rect.bottom > collider.rect.bottom:
                enemy.y_vel = 7
                enemy.rect.top = collider.rect.bottom
                enemy.state = c.FALL
            elif enemy.rect.bottom < collider.rect.bottom:

                enemy.y_vel = 0
                enemy.rect.bottom = collider.rect.top
                enemy.state = c.WALK

        elif brick:
            if brick.state == c.BUMPED:
                enemy.kill()
                self.sprites_about_to_die_group.add(enemy)
                if self.hercules.rect.centerx > brick.rect.centerx:
                    enemy.start_death_jump('right')
                else:
                    enemy.start_death_jump('left')

            elif enemy.rect.x > brick.rect.x:
                enemy.y_vel = 7
                enemy.rect.top = brick.rect.bottom
                enemy.state = c.FALL
            else:
                enemy.y_vel = 0
                enemy.rect.bottom = brick.rect.top
                enemy.state = c.WALK

        elif coin_box:
            if coin_box.state == c.BUMPED:
                self.game_info[c.SCORE] += 100
                self.moving_score_list.append(
                    score.Score(enemy.rect.centerx - self.viewport.x,
                                enemy.rect.y, 100))
                enemy.kill()
                self.sprites_about_to_die_group.add(enemy)
                if self.hercules.rect.centerx > coin_box.rect.centerx:
                    enemy.start_death_jump('right')
                else:
                    enemy.start_death_jump('left')

            elif enemy.rect.x > coin_box.rect.x:
                enemy.y_vel = 7
                enemy.rect.top = coin_box.rect.bottom
                enemy.state = c.FALL
```

```
            else:
                enemy.y_vel = 0
                enemy.rect.bottom = coin_box.rect.top
                enemy.state = c.WALK


        else:
            enemy.rect.y += 1
            test_group = pg.sprite.Group(self.ground_step_pipe_group,
                                         self.coin_box_group,
                                         self.brick_group)
            if pg.sprite.spritecollideany(enemy, test_group) is None:
                if enemy.state != c.JUMP:
                    enemy.state = c.FALL

            enemy.rect.y -= 1

    # Μετακινεί μανιτάρια, αστέρια και τις fireballs
    def adjust_powerup_position(self):
        for powerup in self.powerup_group:
            if powerup.name == c.MUSHROOM:
                self.adjust_mushroom_position(powerup)
            elif powerup.name == c.STAR:
                self.adjust_star_position(powerup)
            elif powerup.name == c.FIREBALL:
                self.adjust_fireball_position(powerup)
            elif powerup.name == '1up_mushroom':
                self.adjust_mushroom_position(powerup)

    # Εμφανίζει, μετακινεί το μανιτάρι και εξετάζει τις συγκρούσεις στους
άξονες x, y
    def adjust_mushroom_position(self, mushroom):
        if mushroom.state != c.REVEAL:
            mushroom.rect.x += mushroom.x_vel
            self.check_mushroom_x_collisions(mushroom)

            mushroom.rect.y += mushroom.y_vel
            self.check_mushroom_y_collisions(mushroom)
            self.delete_if_off_screen(mushroom)

    # Εξετάζει τις συγκρούσεις των μανιταριών με τα άλλα αντικείμενα  στον
άξονα των x
    def check_mushroom_x_collisions(self, mushroom):

        collider = pg.sprite.spritecollideany(mushroom,
self.ground_step_pipe_group)
        brick = pg.sprite.spritecollideany(mushroom, self.brick_group)
        coin_box = pg.sprite.spritecollideany(mushroom,
self.coin_box_group)

        # Σύγκρουση με σωλήνα ή step
        if collider:
            self.adjust_mushroom_for_collision_x(mushroom, collider)
        # Σύγκρουση με brick
        elif brick:
            self.adjust_mushroom_for_collision_x(mushroom, brick)
        # Σύγκρουση με coin box
        elif coin_box:
            self.adjust_mushroom_for_collision_x(mushroom, coin_box)

    # Εξετάζει τις συγκρούσεις του μανιταριού στον άξονα των y
```

```
    def check_mushroom_y_collisions(self, mushroom):
        collider = pg.sprite.spritecollideany(mushroom,
self.ground_step_pipe_group)
        brick = pg.sprite.spritecollideany(mushroom, self.brick_group)
        coin_box = pg.sprite.spritecollideany(mushroom,
self.coin_box_group)


        if collider:
            self.adjust_mushroom_for_collision_y(mushroom, collider)
        elif brick:
            self.adjust_mushroom_for_collision_y(mushroom, brick)
        elif coin_box:
            self.adjust_mushroom_for_collision_y(mushroom, coin_box)
        else:
            self.check_if_falling(mushroom, self.ground_step_pipe_group)
            self.check_if_falling(mushroom, self.brick_group)
            self.check_if_falling(mushroom, self.coin_box_group)

    # Αλλάζει την κατεύθυνση του μανιταριού κατά τη σύγκρουση με κάποιο
άλλο αντικείμενο
    def adjust_mushroom_for_collision_x(self, item, collider):
        if item.rect.x < collider.rect.x:
            item.rect.right = collider.rect.x
            item.direction = c.LEFT
        else:
            item.rect.x = collider.rect.right
            item.direction = c.RIGHT

    # Αλλάζει την κατάσταση του μανιταριού σε SLIDE
    def adjust_mushroom_for_collision_y(self, item, collider):
        item.rect.bottom = collider.rect.y
        item.state = c.SLIDE
        item.y_vel = 0

    # Μετακινεί το αστέρι που κάνει τον Ηρακλή άτρωτος και εξετάζει για
συγκρούσεις
    def adjust_star_position(self, star):

        if star.state == c.BOUNCE:
            star.rect.x += star.x_vel
            self.check_mushroom_x_collisions(star)
            star.rect.y += star.y_vel
            self.check_star_y_collisions(star)
            star.y_vel += star.gravity
            self.delete_if_off_screen(star)

    # Σύγκρουση ου ατεριού με αλλα αντικείμενα
    def check_star_y_collisions(self, star):
        collider = pg.sprite.spritecollideany(star,
self.ground_step_pipe_group)
        brick = pg.sprite.spritecollideany(star, self.brick_group)
        coin_box = pg.sprite.spritecollideany(star, self.coin_box_group)

        if collider:
            self.adjust_star_for_collision_y(star, collider)
        elif brick:
            self.adjust_star_for_collision_y(star, brick)
        elif coin_box:
            self.adjust_star_for_collision_y(star, coin_box)
```

```
    def adjust_star_for_collision_y(self, star, collider):
        if star.rect.y > collider.rect.y:
            star.rect.y = collider.rect.bottom
            star.y_vel = 0
        else:
            star.rect.bottom = collider.rect.top
            star.start_bounce(-8)

    # Μετακινεί τις fireball και ελέγχει για συγκρούσεις
    def adjust_fireball_position(self, fireball):
        if fireball.state == c.FLYING:
            fireball.rect.x += fireball.x_vel
            self.check_fireball_x_collisions(fireball)
            fireball.rect.y += fireball.y_vel
            self.check_fireball_y_collisions(fireball)
        elif fireball.state == c.BOUNCING:
            fireball.rect.x += fireball.x_vel
            self.check_fireball_x_collisions(fireball)
            fireball.rect.y += fireball.y_vel
            self.check_fireball_y_collisions(fireball)
            fireball.y_vel += fireball.gravity
        self.delete_if_off_screen(fireball)

    # Κάνει την fireball να αναπηδά θέτοντάς την σε κατάσταση BOUNCING και
αρχικοποιώντας
    # το y_val και το x_vel
    def bounce_fireball(self, fireball):
        fireball.y_vel = -8
        if fireball.direction == c.RIGHT:
            fireball.x_vel = 15
        else:
            fireball.x_vel = -15

        if fireball in self.powerup_group:
            fireball.state = c.BOUNCING

    # Ελέγχει αν η fireball συγκρούστηκε με κάποιο αντικείμενο
    def check_fireball_x_collisions(self, fireball):
        collide_group = pg.sprite.Group(self.ground_group,
                                        self.pipe_group,
                                        self.step_group,
                                        self.coin_box_group,
                                        self.brick_group)

        collider = pg.sprite.spritecollideany(fireball, collide_group)
        # Αν η fireball χτυπήσει σε step, σωλήνα, coin box και brick τότε
εξαφανίζεται
        if collider:
            fireball.kill()
            self.sprites_about_to_die_group.add(fireball)
            fireball.explode_transition()


    # Ελέγχει αν υπάρχει σύγκρουση των fireball με αντικείμενο στον άξονα
των y
    def check_fireball_y_collisions(self, fireball):
        collide_group = pg.sprite.Group(self.ground_group,
                                        self.pipe_group,
                                        self.step_group,
                                        self.coin_box_group,
```

```
                                        self.brick_group)

        collider = pg.sprite.spritecollideany(fireball, collide_group)
        enemy = pg.sprite.spritecollideany(fireball, self.enemy_group)
        shell = pg.sprite.spritecollideany(fireball, self.shell_group)

        # Αν η fireball συγκρουστεί με το έδαφος τότε αναπηδά
        if collider and (fireball in self.powerup_group):
            fireball.rect.bottom = collider.rect.y
            self.bounce_fireball(fireball)

        # Αν η fireball συγκρουστεί με έναν εχθρό τότε τον σκοτώνει
        elif enemy:
            self.fireball_kill(fireball, enemy)

        elif shell:
            self.fireball_kill(fireball, shell)

    # Σκοτώνει τον εχθρό αν συγκρουστεί μαζί του μία fireball
    def fireball_kill(self, fireball, enemy):
        setup.SFX['kick'].play()
        self.game_info[c.SCORE] += 100
        self.moving_score_list.append(
            score.Score(enemy.rect.centerx - self.viewport.x,
                        enemy.rect.y,100))
        fireball.kill()
        enemy.kill()
        self.sprites_about_to_die_group.add(enemy, fireball)
        enemy.start_death_jump(fireball.direction)
        fireball.explode_transition()


    def check_if_falling(self, sprite, sprite_group):
        sprite.rect.y += 1

        if pg.sprite.spritecollideany(sprite, sprite_group) is None:
            if sprite.state != c.JUMP:
                sprite.state = c.FALL

        sprite.rect.y -= 1

    # Αφαιρεί ένα αντικείμενο αν φύγει από την οθόνη
    def delete_if_off_screen(self, enemy):

        if enemy.rect.x < (self.viewport.x - 300):
            enemy.kill()

        elif enemy.rect.y > (self.viewport.bottom):
            enemy.kill()

        elif enemy.state == c.SHELL_SLIDE:
            if enemy.rect.x > (self.viewport.right + 500):
                enemy.kill()

    # Προσαρμώσει την κατάσταση του Ηρακλή όταν αυτός κατεβάσει τη σημαία
στο τέλος του level
    def check_flag(self):
        if (self.flag.state == c.BOTTOM_OF_POLE and self.hercules.state ==
c.FLAGPOLE):
            self.hercules.set_state_to_bottom_of_pole()
```

```
    # Προσθέτει το flag score στο συνολικό score
    def check_to_add_flag_score(self):
        if self.flag_score.y_vel == 0:
            self.game_info[c.SCORE] += self.flag_score_total
            self.flag_score_total = 0

    # Εξετάζει αν ο Ηρακλής έχει σκοτωθεί (οπότε το frame έχει φύγει από
την οθόνη)
    # και κάνει restart το level
    def check_for_hercules_death(self):
        if self.hercules.rect.y > c.SCREEN_HEIGHT and not
self.hercules.in_castle:
            self.hercules.dead = True
            self.hercules.x_vel = 0
            self.state = c.FROZEN
            self.game_info[c.HERCULES_DEAD] = True

        if self.hercules.dead:
            self.play_death_song()


    def play_death_song(self):
        if self.death_timer == 0:
            self.death_timer = self.current_time
        elif (self.current_time - self.death_timer) > 3000:
            self.set_game_info_values()
            self.done = True

    # Αρχικοποιεί τις τιμές του παιχνιδιού όταν ο παίκτης χάσει όλες του
τις ζωές
    def set_game_info_values(self):
        if self.game_info[c.SCORE] > self.persist[c.TOP_SCORE]:
            self.persist[c.TOP_SCORE] = self.game_info[c.SCORE]
        if self.hercules.dead:
            self.persist[c.LIVES] -= 1

        if self.persist[c.LIVES] == 0:
            self.next = c.GAME_OVER
            self.game_info[c.CAMERA_START_X] = 0
        elif self.hercules.dead == False:
            self.next = c.MAIN_MENU
            self.game_info[c.CAMERA_START_X] = 0
        elif self.overhead_info_display.time == 0:
            self.next = c.TIME_OUT
        else:
            if self.hercules.rect.x > 3670 \
                    and self.game_info[c.CAMERA_START_X] == 0:
                self.game_info[c.CAMERA_START_X] = 3440
            self.next = c.LOAD_SCREEN

    # Εξετάζει αν έχει εξαντληθεί ο χρόνος και ο παίκτης δεν έχει
ολοκληρώσει την πίστα
    def check_if_time_out(self):
        if self.overhead_info_display.time <= 0 \
                and not self.hercules.dead \
                and not self.hercules.in_castle:
            self.state = c.FROZEN
            self.hercules.start_death_jump(self.game_info)

    # Αλλάζει το viewport
    def update_viewport(self):
```

```
        third = self.viewport.x + self.viewport.w//3
        hercules_center = self.hercules.rect.centerx
        hercules_right = self.hercules.rect.right

        if self.hercules.x_vel > 0 and hercules_center >= third:
            mult = 0.7 if hercules_right < self.viewport.centerx else 1.2
            new = self.viewport.x + mult * self.hercules.x_vel
            highest = self.level_rect.w - self.viewport.w
            self.viewport.x = min(highest, new)

    # Κάνει update όταν ο Ηρακλής έχει ολοκληρώσει την πίστα
    def update_while_in_castle(self):
        for score in self.moving_score_list:
            score.update(self.moving_score_list, self.game_info)
        self.overhead_info_display.update(self.game_info)

        if self.overhead_info_display.state == c.END_OF_LEVEL:
            self.state = c.FLAG_AND_FIREWORKS
            self.flag_pole_group.add(castle_flag.Flag(8745, 322))


    def update_flag_and_fireworks(self):
        for score in self.moving_score_list:
            score.update(self.moving_score_list, self.game_info)
        self.overhead_info_display.update(self.game_info)
        self.flag_pole_group.update()

        self.end_game()

    # Τέλος της συγκεκριμένης πίστας
    def end_game(self):
        if self.flag_timer == 0:
            self.flag_timer = self.current_time
        elif (self.current_time - self.flag_timer) > 2000:
            self.set_game_info_values()
            self.next = c.LABOUR_LION
            self.sound_manager.stop_music()
            self.done = True

    # Τοποθετεί όλατα τα αντικείμενα στην οθόνη
    def blit_everything(self, surface):
        self.level.blit(self.background, self.viewport, self.viewport)
        if self.flag_score:
            self.flag_score.draw(self.level)
        self.powerup_group.draw(self.level)
        self.coin_group.draw(self.level)
        self.brick_group.draw(self.level)
        self.coin_box_group.draw(self.level)
        self.sprites_about_to_die_group.draw(self.level)
        self.shell_group.draw(self.level)
        #self.check_point_group.draw(self.level)
        self.brick_pieces_group.draw(self.level)
        self.flag_pole_group.draw(self.level)
        self.hercules_and_enemy_group.draw(self.level)

        surface.blit(self.level, (0,0), self.viewport)
        self.overhead_info_display.draw(surface)
        for score in self.moving_score_list:
            score.draw(surface)
from __future__ import division
```

```
import pygame as pg
from .. import setup, tools
from .. import constants as c
from .. import game_sound
from ..components import hercules
from ..components import collider
from ..components import bricks
from ..components import coin_box
from ..components import coin
from ..components import enemies
from ..components import checkpoint
from ..components import flagpole
from ..components import info
from ..components import score
from ..components import castle_flag


# Η κλάση αντιπροσωπεύει το πρώτο level (πίστα) του παιχνιδιού
class Level2(tools._State):
    def __init__(self):
        tools._State.__init__(self)

    # Εκτελείται για την αρχικοποίηση
    def startup(self, current_time, persist):
        self.game_info = persist
        self.persist = self.game_info
        self.game_info[c.CURRENT_TIME] = current_time
        self.game_info[c.LEVEL_STATE] = c.NOT_FROZEN
        self.game_info[c.HERCULES_DEAD] = False

        self.state = c.NOT_FROZEN
        self.death_timer = 0
        self.flag_timer = 0
        self.flag_score = None
        self.flag_score_total = 0

        self.moving_score_list = []
        self.overhead_info_display = info.OverheadInfo(self.game_info,
c.LEVEL)
        self.sound_manager = game_sound.Sound(self.overhead_info_display)

        self.next = self.set_next_state()

        self.setup_background()
        self.setup_ground()
        self.setup_pipes()
        self.setup_steps()
        self.setup_bricks()
        self.setup_coin_boxes()
        self.setup_static_coins()
        self.setup_flag_pole()
        self.setup_enemies()
        self.setup_hercules()
        self.setup_checkpoints()
        self.setup_spritegroups()

    # Καθορίζεται το επόμενο state ως το LEVEL 1
    def set_next_state(self):
        return c.LEVEL3

    # Ορίζει την εικόνα του background και το viewport
```

```
    def setup_background(self):
        self.background = setup.GFX['level_2']
        self.back_rect = self.background.get_rect()
        self.background = pg.transform.scale(self.background,
                                             (int(self.back_rect.width *
c.BACKGROUND_MULTIPLER),
                                              int(self.back_rect.height *
c.BACKGROUND_MULTIPLER)))
        self.back_rect = self.background.get_rect()
        width = self.back_rect.width
        height = self.back_rect.height

        self.level = pg.Surface((width, height)).convert()
        self.level_rect = self.level.get_rect()
        self.viewport =
setup.SCREEN.get_rect(bottom=self.level_rect.bottom)
        self.viewport.x = self.game_info[c.CAMERA_START_X]

    # Δημιουργεί διάφανα rectangles προκειμένου τα αντικείμενα (Ηρακλής και
enemies) να
    # κοινούνται πάνω σε αυτό
    def setup_ground(self):
        ground_rect1 = collider.Collider(0, 538, 686, 60)
        ground_rect2 = collider.Collider(774, 496, 166, 36)
        ground_rect3 = collider.Collider(1031, 368, 340, 36)
        ground_rect4 = collider.Collider(1074, 406, 252, 194)
        ground_rect5 = collider.Collider(1118, 196, 208, 36)
        ground_rect6 = collider.Collider(1374, 496, 126, 36)
        ground_rect7 = collider.Collider(1504, 325, 208, 36)
        ground_rect8 = collider.Collider(1544, 360, 126, 240)
        ground_rect9 = collider.Collider(1716, 153, 296, 36)
        ground_rect10 = collider.Collider(1760, 190, 208, 410)
        ground_rect11 = collider.Collider(2146, 538, 166, 36)
        ground_rect12 = collider.Collider(2534, 538, 208, 36)
        ground_rect13 = collider.Collider(2574, 198, 166, 36)
        ground_rect14 = collider.Collider(2790, 538, 208, 36)
        ground_rect15 = collider.Collider(3002, 366, 126, 36)
        ground_rect16 = collider.Collider(3044, 406, 40, 194)
        ground_rect17 = collider.Collider(3262, 238, 252, 36)
        ground_rect18 = collider.Collider(3304, 274, 166, 326)
        ground_rect19 = collider.Collider(4204, 452, 166, 36)
        ground_rect20 = collider.Collider(4246, 488, 84, 112)
        ground_rect21 = collider.Collider(4462, 282, 338, 36)
        ground_rect22 = collider.Collider(4502, 318, 252, 282)
        ground_rect23 = collider.Collider(4847, 538, 126, 60)
        ground_rect24 = collider.Collider(4976, 368, 166, 36)
        ground_rect25 = collider.Collider(5018, 406, 84, 194)
        ground_rect26 = collider.Collider(5232, 368, 166, 36)
        ground_rect27 = collider.Collider(5274, 406, 84, 194)
        ground_rect28 = collider.Collider(5532, 538, 1508, 60)


        self.ground_group = pg.sprite.Group(ground_rect1, ground_rect2,
ground_rect3,
                                            ground_rect4, ground_rect5,
ground_rect6,
                                            ground_rect7, ground_rect8,
ground_rect9,
                                            ground_rect10, ground_rect11,
ground_rect12,
                                            ground_rect13, ground_rect14,
```

```
ground_rect15,
                                                ground_rect16, ground_rect17,
ground_rect18,
                                                ground_rect19, ground_rect20,
ground_rect21,
                                                ground_rect22, ground_rect23,
ground_rect24,
                                                ground_rect25, ground_rect26,
ground_rect27,
                                                ground_rect28)

    # "Σημειώνει" τους σωλήνες που υπάρχουν πάνω στο έδαφος ώστε να μπορούν
    # να αλληλεπιδρούν τα αντικείμενα (Ηρακλής και enemies με αυτά)
    def setup_pipes(self):
       pass

    # "Σημειώνει" τσ εμπόδια (τουβλάκια) που υπάρχουν πάνω στο έδαφος ώστε
να μπορούν
    # να αλληλεπιδρούν τα αντικείμενα (Ηρακλής και enemies με αυτά)
    def setup_steps(self):
        step1 = collider.Collider(5917, 366, 80, 176)
        step2 = collider.Collider(6002, 280, 80, 264)
        step3 = collider.Collider(6090, 194, 80, 352)
        step4 = collider.Collider(6516, 495, 40, 44)

        self.step_group = pg.sprite.Group(step1, step2, step3, step4)

    # Δημιουργεί όλα τα breakable τούβλα της συγκεκριμένης πίστας
    def setup_bricks(self):


        self.powerup_group = pg.sprite.Group()
        self.brick_pieces_group = pg.sprite.Group()

        brick1 = bricks.Brick(3750, 365)
        brick2 = bricks.Brick(3782, 365)


        self.brick_group = pg.sprite.Group(brick1, brick2)

    # Δημιουργεί όλα τα coin boxes που μπορεί να περιέχουν και mushroom ή
flower
    def setup_coin_boxes(self):
        coin_box1 = coin_box.Coin_box(522, 365, c.MUSHROOM,
self.powerup_group)
        self.coin_box_group = pg.sprite.Group(coin_box1)

    def setup_static_coins(self):

        static_coin1 = coin.StaticCoin(1166, 156)
        static_coin2 = coin.StaticCoin(1208, 156)
        static_coin3 = coin.StaticCoin(1250, 156)
        static_coin4 = coin.StaticCoin(1594, 70)
        static_coin5 = coin.StaticCoin(1636, 70)
        static_coin6 = coin.StaticCoin(2152, 240)
        static_coin7 = coin.StaticCoin(2194, 240)
        static_coin8 = coin.StaticCoin(2580, 156)
        static_coin9 = coin.StaticCoin(2622, 156)
        static_coin10 = coin.StaticCoin(2664, 156)
        static_coin11 = coin.StaticCoin(2706, 156)
        static_coin12 = coin.StaticCoin(3652, 198)
```

```
        static_coin13 = coin.StaticCoin(3694, 198)
        static_coin14 = coin.StaticCoin(3996, 156)
        static_coin15 = coin.StaticCoin(4038, 156)
        static_coin16 = coin.StaticCoin(4166, 156)
        static_coin17 = coin.StaticCoin(4208, 156)
        static_coin18 = coin.StaticCoin(4852, 496)
        static_coin19 = coin.StaticCoin(4894, 496)
        static_coin20 = coin.StaticCoin(4936, 496)
        static_coin21 = coin.StaticCoin(5154, 198)
        static_coin22 = coin.StaticCoin(5196, 198)


        self.coin_group = pg.sprite.Group(static_coin1, static_coin2,
static_coin3, static_coin4,
                                          static_coin5, static_coin6,
static_coin7, static_coin8,
                                          static_coin9, static_coin10,
static_coin11, static_coin12,
                                          static_coin13, static_coin14,
static_coin15, static_coin16,
                                          static_coin17, static_coin18,
static_coin19, static_coin20,
                                          static_coin21, static_coin22)


    # Δημιουγεί τον ιστό και την σημεία πάνω στον ιστό στο τέλος της πίστας
    def setup_flag_pole(self):
        self.flag = flagpole.Flag(6531, 116)

        pole0 = flagpole.Pole(8505, 97)
        pole1 = flagpole.Pole(8505, 137)
        pole2 = flagpole.Pole(8505, 177)
        pole3 = flagpole.Pole(8505, 217)
        pole4 = flagpole.Pole(8505, 257)
        pole5 = flagpole.Pole(8505, 297)
        pole6 = flagpole.Pole(8505, 337)
        pole7 = flagpole.Pole(8505, 377)
        pole8 = flagpole.Pole(8505, 417)
        pole9 = flagpole.Pole(8505, 450)

        finial = flagpole.Finial(8497, 97)

        self.flag_pole_group = pg.sprite.Group(self.flag, finial, pole0,
                                               pole1, pole2, pole3,
                                               pole4, pole5, pole6,
                                               pole7, pole8, pole9)

    # Δημιουργεί όλους τους enemies και τους διατηρεί σε μία λίστα
    def setup_enemies(self):
        lion0 = enemies.Lion(x = 1284, y = 196)
        lion1 = enemies.Lion(x = 1886, y = 152)
        lion2 = enemies.Lion(x = 1946, y = 152)
        lion3 = enemies.Lion(x = 3160, y = 220)
        lion4 = enemies.Lion(x = 3458, y = 238)
        lion5 = enemies.Lion(x = 4740, y = 282)
        lion6 = enemies.Lion(x = 4880, y = 260)
        lion7 = enemies.Lion(x = 5836, y = 538)


        enemy_group0 = pg.sprite.Group(lion0)
        enemy_group1 = pg.sprite.Group(lion1, lion2)
```

```
        enemy_group2 = pg.sprite.Group(lion3)
        enemy_group3 = pg.sprite.Group(lion4)
        enemy_group4 = pg.sprite.Group(lion5)
        enemy_group5 = pg.sprite.Group(lion6)
        enemy_group6 = pg.sprite.Group(lion7)

        self.enemy_group_list = [enemy_group0, enemy_group1,
                                 enemy_group2, enemy_group3,
                                 enemy_group4, enemy_group5,
                                 enemy_group6]

    # Δημιουργεί τον χαρακτήρα του Ηρακλή και τον τοποθετεί στην αρχή του
level 2
    def setup_hercules(self):
        self.hercules = hercules.Hercules()
        self.hercules.rect.x = self.viewport.x + 250
        self.hercules.rect.bottom = c.GROUND_HEIGHT

    # Δημιουργεί invisible checkpoints με τα οποία όταν ο Ηρακλής έρθει σε
επαφή θα
    # προκαλέσει τη δημιουργία εχθρών από την self.enemy_group_list
    def setup_checkpoints(self):
        check1 = checkpoint.Checkpoint(690, "1")
        check2 = checkpoint.Checkpoint(1292, '2')
        check3 = checkpoint.Checkpoint(2300, '3')
        check4 = checkpoint.Checkpoint(2870, '4')
        check5 = checkpoint.Checkpoint(4154, '5')
        check6 = checkpoint.Checkpoint(4466, '6')
        check7 = checkpoint.Checkpoint(5246, '7')
        check8 = checkpoint.Checkpoint(6536, '8')
        check9 = checkpoint.Checkpoint(6853, '9')


        self.check_point_group = pg.sprite.Group(check1, check2, check3,
                                                 check4, check5, check6,
                                                 check7, check8, check9)

    # Δημιουργούμε sprite groups για να τα ομαδοποιήσουμε
    # και να μπορούμε να τα χειριστούμε με μεγαλύτερη ευκολία
    def setup_spritegroups(self):
        self.sprites_about_to_die_group = pg.sprite.Group()
        self.shell_group = pg.sprite.Group()
        self.enemy_group = pg.sprite.Group()

        self.ground_step_pipe_group = pg.sprite.Group(self.ground_group,
self.step_group)

        self.hercules_and_enemy_group = pg.sprite.Group(self.hercules,
self.enemy_group)

    # Κάνει update ολόκληρο το level. Καλειται από την tools.Control κλάση
    def update(self, surface, keys, current_time):
        self.game_info[c.CURRENT_TIME] = self.current_time = current_time
        self.handle_states(keys)
        self.check_if_time_out()
        self.blit_everything(surface)
        self.sound_manager.update(self.game_info, self.hercules)

    # Εκτελείται η αντίστοιχη λειτουργία ανάλογα με την κατάσταση του level
    def handle_states(self, keys):
        if self.state == c.FROZEN:
```

```
            self.update_during_transition_state(keys)
        elif self.state == c.NOT_FROZEN:
            self.update_all_sprites(keys)
        elif self.state == c.IN_CASTLE:
            self.update_while_in_castle()
        elif self.state == c.FLAG_AND_FIREWORKS:
            self.update_flag_and_fireworks()

    # Κάνει upadate τον Ηρακλή καθώς αυτός βρίσκεται σε transition
κατάσταση
    # γίνεται μεγάλος΄η μικρός ή πεθαίνει ή αποκτά πυροβόλο
    def update_during_transition_state(self, keys):

        self.hercules.update(keys, self.game_info, self.powerup_group)

        for score in self.moving_score_list:
            score.update(self.moving_score_list, self.game_info)
        if self.flag_score:
            self.flag_score.update(None, self.game_info)
            self.check_to_add_flag_score()
        # Κάνει update τα coins και την σημαία στο τέλος του level
        self.coin_box_group.update(self.game_info)
        self.flag_pole_group.update(self.game_info)
        # Εξετάζει μήπως ο Ηρακλής δεν είναι πλέον σε κατάσταση transition
        self.check_if_hercules_in_transition_state()
        # Εξετάζει αν ο Ηρακλής έχει κατεβάσει τη σημαία
        self.check_flag()
        # Εξετάζει αν ο Ηρακλής έχει σκοτωθεί
        self.check_for_hercules_death()
        # Τυπωνει τις overhead πληροφορίες στην οθόνη
        self.overhead_info_display.update(self.game_info, self.hercules)

    # Εξετάζει αν ο Ηρακλής είναι σε κατάσταση transition και θέτει την
κατάσταση του level σε FROZEN
    # Διαφορετικά θέτει την κατάσταση του level σε NOT FROZEN
    def check_if_hercules_in_transition_state(self):
        if self.hercules.in_transition_state:
            self.game_info[c.LEVEL_STATE] = self.state = c.FROZEN
        elif self.hercules.in_transition_state == False:
            if self.state == c.FROZEN:
                self.game_info[c.LEVEL_STATE] = self.state = c.NOT_FROZEN

    # Το level είναι σε κατάσταση NOT FROZEN οπότε όλα τα sprites
    # ανανεώνονται μαζί και η θέση τους
    def update_all_sprites(self, keys):
        self.hercules.update(keys, self.game_info, self.powerup_group)
        for score in self.moving_score_list:
            score.update(self.moving_score_list, self.game_info)
        if self.flag_score:
            self.flag_score.update(None, self.game_info)
            self.check_to_add_flag_score()
        self.flag_pole_group.update()
        # Ελεγχος αν ο Ηρακλής περασε από τα checkpoints
        self.check_points_check()
        self.enemy_group.update(self.game_info)
        self.sprites_about_to_die_group.update(self.game_info,
self.viewport)
        self.shell_group.update(self.game_info)
        self.brick_group.update()
        self.coin_box_group.update(self.game_info)
        self.powerup_group.update(self.game_info, self.viewport)
```

```
        self.coin_group.update(self.game_info)
        self.brick_pieces_group.update()
        self.adjust_sprite_positions()
        self.check_if_hercules_in_transition_state()
        self.check_for_hercules_death()
        self.update_viewport()
        self.overhead_info_display.update(self.game_info, self.hercules)

    # Εξετάζει αν ο Ηρακλής έχει περάσει από κάποιο checkpoint ώστε να
    # δημιουργηθούν οι enemies
    def check_points_check(self):
        checkpoint = pg.sprite.spritecollideany(self.hercules,
self.check_point_group)

        # Αν έχει περάσει από checkpoint
        if checkpoint:
            checkpoint.kill()
            # Τα checkpoint 1 εως 10 πυροδοτούν την δημιουργία των enemies
που έχουμε
            # ορίσει παραπάνω. Οπότε μόλις περάσουμε ένα από αυτά τα
checkpoints δημιουργούνται
            # τα αντίστοιχα enemies
            for i in range(1, 8):
                if checkpoint.name == str(i):
                    for index, enemy in enumerate(self.enemy_group_list[i -
1]):
                        enemy.rect.x = self.viewport.right + (index * 60)
                    self.enemy_group.add(self.enemy_group_list[i - 1])

            # Το checkpoint 8 αφορά τον ιστό στο τέλος του level
            if checkpoint.name == '8':
                self.hercules.state = c.FLAGPOLE
                self.hercules.invincible = False
                self.hercules.flag_pole_right = checkpoint.rect.right
                if self.hercules.rect.bottom < self.flag.rect.y:
                    self.hercules.rect.bottom = self.flag.rect.y
                self.flag.state = c.SLIDE_DOWN
                # Εμφανίζει το σκορ που κέρδισε ο Ηρακλής στον ιστό
                self.create_flag_points()

            # Το checkpoint 12 σημαίνει ότι ο Ηρακλής έφτασε στο κάστρο και
            # έχει ολοκληρωθεί το level
            elif checkpoint.name == '9':
                self.state = c.IN_CASTLE
                self.hercules.kill()
                self.hercules.state == c.STAND
                self.hercules.in_castle = True
                self.overhead_info_display.state = c.FAST_COUNT_DOWN

            # Σε διαφορετική περίπτωση σχετίζεται με το κρυφή live
            elif checkpoint.name == 'secret_mushroom' and
self.hercules.y_vel < 0:
                mushroom_box = coin_box.Coin_box(checkpoint.rect.x,
                                                 checkpoint.rect.bottom -
40,
                                                 '1up_mushroom',
                                                 self.powerup_group)
                mushroom_box.start_bump(self.moving_score_list)
                self.coin_box_group.add(mushroom_box)

                self.hercules.y_vel = 7
```

```
                self.hercules.rect.y = mushroom_box.rect.bottom
                self.hercules.state = c.FALL

            self.hercules_and_enemy_group.add(self.enemy_group)

    # Δημιουργεί τους πόντους που εμφανίζνται όταν ο Ηρακλής πηδάει στον ιστό
    # Οι πόντοι εξαρτώνται από το πόσο ψυλά φτάνει στον ιστό
    def create_flag_points(self):

        x = 8518
        y = c.GROUND_HEIGHT - 60
        hercules_bottom = self.hercules.rect.bottom

        if hercules_bottom > (c.GROUND_HEIGHT - 40 - 40):
            self.flag_score = score.Score(x, y, 100, True)
            self.flag_score_total = 100
        elif hercules_bottom > (c.GROUND_HEIGHT - 40 - 160):
            self.flag_score = score.Score(x, y, 400, True)
            self.flag_score_total = 400
        elif hercules_bottom > (c.GROUND_HEIGHT - 40 - 240):
            self.flag_score = score.Score(x, y, 800, True)
            self.flag_score_total = 800
        elif hercules_bottom > (c.GROUND_HEIGHT - 40 - 360):
            self.flag_score = score.Score(x, y, 2000, True)
            self.flag_score_total = 2000
        else:
            self.flag_score = score.Score(x, y, 5000, True)
            self.flag_score_total = 5000

    def adjust_sprite_positions(self):
        self.adjust_hercules_position()
        self.adjust_enemy_position()
        self.adjust_powerup_position()

    # Προσαρμόζει την θέση του Ηρακλή ανάλογα με την ταχύτητά του και τις πιθανές συγκρούσεις
    def adjust_hercules_position(self):
        self.last_x_position = self.hercules.rect.right
        self.hercules.rect.x += round(self.hercules.x_vel)
        self.check_hercules_x_collisions()

        if self.hercules.in_transition_state == False:
            self.hercules.rect.y += round(self.hercules.y_vel)
            self.check_hercules_y_collisions()

        if self.hercules.rect.x < (self.viewport.x + 5):
            self.hercules.rect.x = (self.viewport.x + 5)

    # Εξετάζει αν ο Ηρακλής συγκρούεται με ενα αντικείμενο κατά την οριζόντια κίνηση
    def check_hercules_x_collisions(self):
        collider = pg.sprite.spritecollideany(self.hercules, self.ground_step_pipe_group)
        coin_box = pg.sprite.spritecollideany(self.hercules, self.coin_box_group)
        brick = pg.sprite.spritecollideany(self.hercules, self.brick_group)
        coin = pg.sprite.spritecollideany(self.hercules, self.coin_group)
        enemy = pg.sprite.spritecollideany(self.hercules, self.enemy_group)
        powerup = pg.sprite.spritecollideany(self.hercules, self.powerup_group)
```

```
        if coin_box:
            self.adjust_hercules_for_x_collisions(coin_box)

        elif coin:
            setup.SFX['coin'].play()
            self.game_info[c.SCORE] += 100
            self.game_info[c.COIN_TOTAL] += 1

self.moving_score_list.append(score.Score(self.hercules.rect.centerx -
self.viewport.x, self.hercules.rect.y, 100))
            coin.kill()

        elif brick:
            self.adjust_hercules_for_x_collisions(brick)

        elif collider:
            self.adjust_hercules_for_x_collisions(collider)

        # Αν έρθει σε επαφή με εχθρό
        elif enemy:
            # Ο Ηρακλής είναι σε κατάσταση ανίκητος (μετά από αστέρι)
            if self.hercules.invincible:
                setup.SFX['kick'].play()
                self.game_info[c.SCORE] += 100
                self.moving_score_list.append(
                    score.Score(self.hercules.rect.right - self.viewport.x,
                                self.hercules.rect.y, 100))
                enemy.kill()
                enemy.start_death_jump(c.RIGHT)
                self.sprites_about_to_die_group.add(enemy)
            # Ο Ηρακλής είναι μεγαλος οπότε μικραίνει
            elif self.hercules.big:
                setup.SFX['pipe'].play()
                self.hercules.fire = False
                self.hercules.y_vel = -1
                self.hercules.state = c.BIG_TO_SMALL
                self.convert_fireflowers_to_mushrooms()
            elif self.hercules.hurt_invincible:
                pass
            # Ο Ηρακλής είναι μικρός οπότε χάνει μία ζωή
            else:
                self.hercules.start_death_jump(self.game_info)
                self.state = c.FROZEN
        # Ο Ηρακλής έρχεται σε επαφή με κάποιο powerup
        elif powerup:
            # Αστέρι
            if powerup.name == c.STAR:
                self.game_info[c.SCORE] += 1000

                self.moving_score_list.append(
                    score.Score(self.hercules.rect.centerx -
self.viewport.x,
                                self.hercules.rect.y, 1000))
                self.hercules.invincible = True
                self.hercules.invincible_start_timer = self.current_time
            # Μανιτάρι που μεγαλώνει
            elif powerup.name == c.MUSHROOM:
                setup.SFX['powerup'].play()
                self.game_info[c.SCORE] += 1000
                self.moving_score_list.append(
```

```
                    score.Score(self.hercules.rect.centerx -
self.viewport.x,
                                self.hercules.rect.y - 20, 1000))

                self.hercules.y_vel = -1
                self.hercules.state = c.SMALL_TO_BIG
                self.hercules.in_transition_state = True
                self.convert_mushrooms_to_fireflowers()
            # Μαντιτάρι που κερδίζει ζωή
            elif powerup.name == c.LIFE_MUSHROOM:
                self.moving_score_list.append(
                    score.Score(powerup.rect.right - self.viewport.x,
                                powerup.rect.y,
                                c.ONEUP))

                self.game_info[c.LIVES] += 1
                setup.SFX['one_up'].play()
            # Fire flower που του δίνει πυρομαχικά
            elif powerup.name == c.FIREFLOWER:
                setup.SFX['powerup'].play()
                self.game_info[c.SCORE] += 1000
                self.moving_score_list.append(
                    score.Score(self.hercules.rect.centerx -
self.viewport.x,
                                self.hercules.rect.y, 1000))

                if self.hercules.big and self.hercules.fire == False:
                    self.hercules.state = c.BIG_TO_FIRE
                    self.hercules.in_transition_state = True
                elif self.hercules.big == False:
                    self.hercules.state = c.SMALL_TO_BIG
                    self.hercules.in_transition_state = True
                    self.convert_mushrooms_to_fireflowers()

            if powerup.name != c.FIREBALL:
                powerup.kill()

    # Όταν ο Ηρακλής γίνει μεγάλος τότε μετατρέπει όλα τα mushrooms σε
fireflowers
    def convert_mushrooms_to_fireflowers(self):
        for brick in self.brick_group:
            if brick.contents == c.MUSHROOM:
                brick.contents = c.FIREFLOWER
        for coin_box in self.coin_box_group:
            if coin_box.contents == c.MUSHROOM:
                coin_box.contents = c.FIREFLOWER

    # Όταν ο Ηρακλής γίνει μικρός τότε μετατρέπει όλα τα fireflowers σε
mushroom
    def convert_fireflowers_to_mushrooms(self):
        for brick in self.brick_group:
            if brick.contents == c.FIREFLOWER:
                brick.contents = c.MUSHROOM
        for coin_box in self.coin_box_group:
            if coin_box.contents == c.FIREFLOWER:
                coin_box.contents = c.MUSHROOM

    # Τοποθετεί τον Ηρακλή δίπλα στον collider
    def adjust_hercules_for_x_collisions(self, collider):
        if self.hercules.rect.x < collider.rect.x:
            self.hercules.rect.right = collider.rect.left
```

```
        else:
            self.hercules.rect.left = collider.rect.right

        self.hercules.x_vel = 0


    # Εξετάζει αν ο Ηρακλής έρχεται σε επαφή με άλλα αντικείμενα κατά την
κίνηση
    # στον κάθετο άξονα
    def check_hercules_y_collisions(self):

        ground_step_or_pipe = pg.sprite.spritecollideany(self.hercules,
self.ground_step_pipe_group)
        enemy = pg.sprite.spritecollideany(self.hercules, self.enemy_group)
        brick = pg.sprite.spritecollideany(self.hercules, self.brick_group)
        coin_box = pg.sprite.spritecollideany(self.hercules,
self.coin_box_group)
        powerup = pg.sprite.spritecollideany(self.hercules,
self.powerup_group)

        brick, coin_box = self.prevent_collision_conflict(brick, coin_box)

        if coin_box:
            self.adjust_hercules_for_y_coin_box_collisions(coin_box)

        elif brick:
            self.adjust_hercules_for_y_brick_collisions(brick)

        elif ground_step_or_pipe:

self.adjust_hercules_for_y_ground_pipe_collisions(ground_step_or_pipe)

        elif enemy:
            if self.hercules.invincible:
                setup.SFX['kick'].play()
                enemy.kill()
                self.sprites_about_to_die_group.add(enemy)
                enemy.start_death_jump(c.RIGHT)
            else:
                self.adjust_hercules_for_y_enemy_collisions(enemy)

        elif powerup:
            if powerup.name == c.STAR:
                setup.SFX['powerup'].play()
                powerup.kill()
                self.hercules.invincible = True
                self.hercules.invincible_start_timer = self.current_time

        self.test_if_hercules_is_falling()

    def prevent_collision_conflict(self, obstacle1, obstacle2):
        if obstacle1 and obstacle2:
            obstacle1_distance = self.hercules.rect.centerx -
obstacle1.rect.centerx
            if obstacle1_distance < 0:
                obstacle1_distance *= -1
            obstacle2_distance = self.hercules.rect.centerx -
obstacle2.rect.centerx
            if obstacle2_distance < 0:
                obstacle2_distance *= -1
```

```
            if obstacle1_distance < obstacle2_distance:
                obstacle2 = False
            else:
                obstacle1 = False

        return obstacle1, obstacle2

    # Σύγκρουση του Ηρακλή με τα coin boxes κατά τον y άξονα. Αν το coin
box είναι σε κατασταση Resting και
    # περιέχει κάποιο coin τότε αυξάνεται το πληθος των coin κατά 1 και το
box παίρνει την κατάσταση OPENED
    def adjust_hercules_for_y_coin_box_collisions(self, coin_box):
        if self.hercules.rect.y > coin_box.rect.y:
            if coin_box.state == c.RESTING:
                if coin_box.contents == c.COIN:
                    self.game_info[c.SCORE] += 200
                    coin_box.start_bump(self.moving_score_list)
                    self.game_info[c.COIN_TOTAL] += 1
                else:
                    coin_box.start_bump(self.moving_score_list)

            elif coin_box.state == c.OPENED:
                pass
            setup.SFX['bump'].play()
            self.hercules.y_vel = 7
            self.hercules.rect.y = coin_box.rect.bottom
            self.hercules.state = c.FALL
        else:
            self.hercules.y_vel = 0
            self.hercules.rect.bottom = coin_box.rect.top
            self.hercules.state = c.WALK

    # Εκτελείται όταν ο Ηρακλής συγκρούεται με ένα brick (τούβλο)
    def adjust_hercules_for_y_brick_collisions(self, brick):
        if self.hercules.rect.y > brick.rect.y:
            if brick.state == c.RESTING:
                if self.hercules.big and brick.contents is None:
                    setup.SFX['brick_smash'].play()
                    self.check_if_enemy_on_brick(brick)
                    brick.kill()
                    self.brick_pieces_group.add(
                        bricks.BrickPiece(brick.rect.x,
                                          brick.rect.y - (brick.rect.height
/ 2),
                                          -2, -12),
                        bricks.BrickPiece(brick.rect.right,
                                          brick.rect.y - (brick.rect.height
/ 2),
                                          2, -12),
                        bricks.BrickPiece(brick.rect.x,
                                          brick.rect.y,
                                          -2, -6),
                        bricks.BrickPiece(brick.rect.right,
                                          brick.rect.y,
                                          2, -6))
                else:
                    setup.SFX['bump'].play()
                    if brick.coin_total > 0:
                        self.game_info[c.COIN_TOTAL] += 1
                        self.game_info[c.SCORE] += 200
                    self.check_if_enemy_on_brick(brick)
```

```
                    brick.start_bump(self.moving_score_list)
            elif brick.state == c.OPENED:
                setup.SFX['bump'].play()
            self.hercules.y_vel = 7
            self.hercules.rect.y = brick.rect.bottom
            self.hercules.state = c.FALL

        else:
            self.hercules.y_vel = 0
            self.hercules.rect.bottom = brick.rect.top
            self.hercules.state = c.WALK

    # Αν υπάρχει κάποιος εχθρός ο οποίος είναι πάνω στο brick το οποίο
σπάει ή κουνάει
    # ο Ηρακλής τότε ο εχθρός πεθαίνει
    def check_if_enemy_on_brick(self, brick):
        brick.rect.y -= 5

        enemy = pg.sprite.spritecollideany(brick, self.enemy_group)

        if enemy:
            setup.SFX['kick'].play()
            self.game_info[c.SCORE] += 100
            self.moving_score_list.append(
                score.Score(enemy.rect.centerx - self.viewport.x,
                            enemy.rect.y,
                            100))
            enemy.kill()
            self.sprites_about_to_die_group.add(enemy)
            if self.hercules.rect.centerx > brick.rect.centerx:
                enemy.start_death_jump('right')
            else:
                enemy.start_death_jump('left')

        brick.rect.y += 5

    # Ο Ηρακλής έρχεται σε επαφή με σωλήνα
    def adjust_hercules_for_y_ground_pipe_collisions(self, collider):

        if collider.rect.bottom > self.hercules.rect.bottom:
            self.hercules.y_vel = 0
            self.hercules.rect.bottom = collider.rect.top
            if self.hercules.state == c.END_OF_LEVEL_FALL:
                self.hercules.state = c.WALKING_TO_CASTLE
            else:
                self.hercules.state = c.WALK
        elif collider.rect.top < self.hercules.rect.top:
            self.hercules.y_vel = 7
            self.hercules.rect.top = collider.rect.bottom
            self.hercules.state = c.FALL

    def test_if_hercules_is_falling(self):
        self.hercules.rect.y += 1
        test_collide_group = pg.sprite.Group(self.ground_step_pipe_group,
                                             self.brick_group,
self.coin_group,
                                             self.coin_box_group)

        if pg.sprite.spritecollideany(self.hercules, test_collide_group) is
None:
            if self.hercules.state != c.JUMP \
```

```
                    and self.hercules.state != c.DEATH_JUMP \
                    and self.hercules.state != c.SMALL_TO_BIG \
                    and self.hercules.state != c.BIG_TO_FIRE \
                    and self.hercules.state != c.BIG_TO_SMALL \
                    and self.hercules.state != c.FLAGPOLE \
                    and self.hercules.state != c.WALKING_TO_CASTLE \
                    and self.hercules.state != c.END_OF_LEVEL_FALL:
                self.hercules.state = c.FALL
            elif self.hercules.state == c.WALKING_TO_CASTLE or \
                    self.hercules.state == c.END_OF_LEVEL_FALL:
                self.hercules.state = c.END_OF_LEVEL_FALL

        self.hercules.rect.y -= 1

    # Ο Ηρακλής έρχεται σε επαφή με τον εχθρό κατά τον άξονα των y
    def adjust_hercules_for_y_enemy_collisions(self, enemy):

        if self.hercules.y_vel > 0:
            setup.SFX['stomp'].play()
            self.game_info[c.SCORE] += 100
            self.moving_score_list.append(
                score.Score(enemy.rect.centerx - self.viewport.x,
                            enemy.rect.y, 100))
            enemy.state = c.JUMPED_ON
            enemy.kill()
            if enemy.name == c.GOOMBA or enemy.name == c.LION:
                enemy.death_timer = self.current_time
                self.sprites_about_to_die_group.add(enemy)
            elif enemy.name == c.KOOPA:
                self.shell_group.add(enemy)

            self.hercules.rect.bottom = enemy.rect.top
            self.hercules.state = c.JUMP
            self.hercules.y_vel = -7


    # Μετακινεί ολους τους εχθρούς κατά τον άξονα Χ και Υ και εξετάζει αν
έγιναν συγκρούσεις μετά την κίνηση
    # του εχθρού
    def adjust_enemy_position(self):

        for enemy in self.enemy_group:
            enemy.rect.x += enemy.x_vel
            self.check_enemy_x_collisions(enemy)

            enemy.rect.y += enemy.y_vel
            self.check_enemy_y_collisions(enemy)
            self.delete_if_off_screen(enemy)

    # Εξετάζει τις συγκρούσεις των εχθρών μεταξύ τους καθως και με σωλήνες
η steps
    def check_enemy_x_collisions(self, enemy):
        enemy.kill()
        collider = pg.sprite.spritecollideany(enemy,
self.ground_step_pipe_group)
        enemy_collider = pg.sprite.spritecollideany(enemy,
self.enemy_group)

        # Συγκρουση του εχθρού με έναν σωλήνα ή step οπότε αλλάζει
κατεύθυνση και
        # πηγαίνει προς την αντίθετη πλευρά
```

```
        if collider:
            if enemy.direction == c.RIGHT:
                enemy.rect.right = collider.rect.left
                enemy.direction = c.LEFT
                enemy.x_vel = -2
            elif enemy.direction == c.LEFT:
                enemy.rect.left = collider.rect.right
                enemy.direction = c.RIGHT
                enemy.x_vel = 2

        # σύγκρουση του εχθρού με κάποιον άλλον εχθρό οπότε και οι δύο
εχθροί θα κινηθούν προς την
        # αντίθετη κατεύθυνση
        elif enemy_collider:
            if enemy.direction == c.RIGHT:
                enemy.rect.right = enemy_collider.rect.left
                enemy.direction = c.LEFT
                enemy_collider.direction = c.RIGHT
                enemy.x_vel = -2
                enemy_collider.x_vel = 2
            elif enemy.direction == c.LEFT:
                enemy.rect.left = enemy_collider.rect.right
                enemy.direction = c.RIGHT
                enemy_collider.direction = c.LEFT
                enemy.x_vel = 2
                enemy_collider.x_vel = -2

        self.enemy_group.add(enemy)
        self.hercules_and_enemy_group.add(self.enemy_group)

    # Ελέγχει τις συγκρούσεις των εχθρών ως προς τον άξονα των y
    def check_enemy_y_collisions(self, enemy):

        collider = pg.sprite.spritecollideany(enemy,
self.ground_step_pipe_group)
        brick = pg.sprite.spritecollideany(enemy, self.brick_group)
        coin_box = pg.sprite.spritecollideany(enemy, self.coin_box_group)

        if collider:
            if enemy.rect.bottom > collider.rect.bottom:
                enemy.y_vel = 7
                enemy.rect.top = collider.rect.bottom
                enemy.state = c.FALL
            elif enemy.rect.bottom < collider.rect.bottom:

                enemy.y_vel = 0
                enemy.rect.bottom = collider.rect.top
                enemy.state = c.WALK

        elif brick:
            if brick.state == c.BUMPED:
                enemy.kill()
                self.sprites_about_to_die_group.add(enemy)
                if self.hercules.rect.centerx > brick.rect.centerx:
                    enemy.start_death_jump('right')
                else:
                    enemy.start_death_jump('left')

            elif enemy.rect.x > brick.rect.x:
                enemy.y_vel = 7
                enemy.rect.top = brick.rect.bottom
```

```
                enemy.state = c.FALL
            else:
                enemy.y_vel = 0
                enemy.rect.bottom = brick.rect.top
                enemy.state = c.WALK

        elif coin_box:
            if coin_box.state == c.BUMPED:
                self.game_info[c.SCORE] += 100
                self.moving_score_list.append(
                    score.Score(enemy.rect.centerx - self.viewport.x,
                                enemy.rect.y, 100))
                enemy.kill()
                self.sprites_about_to_die_group.add(enemy)
                if self.hercules.rect.centerx > coin_box.rect.centerx:
                    enemy.start_death_jump('right')
                else:
                    enemy.start_death_jump('left')

            elif enemy.rect.x > coin_box.rect.x:
                enemy.y_vel = 7
                enemy.rect.top = coin_box.rect.bottom
                enemy.state = c.FALL
            else:
                enemy.y_vel = 0
                enemy.rect.bottom = coin_box.rect.top
                enemy.state = c.WALK


        else:
            enemy.rect.y += 1
            test_group = pg.sprite.Group(self.ground_step_pipe_group,
                                         self.coin_box_group,
                                         self.brick_group)
            if pg.sprite.spritecollideany(enemy, test_group) is None:
                if enemy.state != c.JUMP:
                    enemy.state = c.FALL

            enemy.rect.y -= 1



    # Μετακινεί μανιτάρια, αστέρια και τις fireballs
    def adjust_powerup_position(self):
        for powerup in self.powerup_group:
            if powerup.name == c.MUSHROOM:
                self.adjust_mushroom_position(powerup)
            elif powerup.name == c.STAR:
                self.adjust_star_position(powerup)
            elif powerup.name == c.FIREBALL:
                self.adjust_fireball_position(powerup)
            elif powerup.name == '1up_mushroom':
                self.adjust_mushroom_position(powerup)

    # Εμφανίζει, μετακινεί το μανιτάρι και εξετάζει τις συγκρούσεις στους
άξονες x, y
    def adjust_mushroom_position(self, mushroom):
        if mushroom.state != c.REVEAL:
            mushroom.rect.x += mushroom.x_vel
            self.check_mushroom_x_collisions(mushroom)
```

```
            mushroom.rect.y += mushroom.y_vel
            self.check_mushroom_y_collisions(mushroom)
            self.delete_if_off_screen(mushroom)

    # Εξετάζει τις συγκρούσεις των μανιταριών με τα άλλα αντικείμενα  στον
άξονα των x
    def check_mushroom_x_collisions(self, mushroom):

        collider = pg.sprite.spritecollideany(mushroom,
self.ground_step_pipe_group)
        brick = pg.sprite.spritecollideany(mushroom, self.brick_group)
        coin_box = pg.sprite.spritecollideany(mushroom,
self.coin_box_group)

        # Σύγκρουση με σωλήνα ή step
        if collider:
            self.adjust_mushroom_for_collision_x(mushroom, collider)
        # Σύγκρουση με brick
        elif brick:
            self.adjust_mushroom_for_collision_x(mushroom, brick)
        # Σύγκρουση με coin box
        elif coin_box:
            self.adjust_mushroom_for_collision_x(mushroom, coin_box)

    # Εξετάζει τις συγκρούσεις του μανιταριού στον άξονα των y
    def check_mushroom_y_collisions(self, mushroom):
        collider = pg.sprite.spritecollideany(mushroom,
self.ground_step_pipe_group)
        brick = pg.sprite.spritecollideany(mushroom, self.brick_group)
        coin_box = pg.sprite.spritecollideany(mushroom,
self.coin_box_group)

        if collider:
            self.adjust_mushroom_for_collision_y(mushroom, collider)
        elif brick:
            self.adjust_mushroom_for_collision_y(mushroom, brick)
        elif coin_box:
            self.adjust_mushroom_for_collision_y(mushroom, coin_box)
        else:
            self.check_if_falling(mushroom, self.ground_step_pipe_group)
            self.check_if_falling(mushroom, self.brick_group)
            self.check_if_falling(mushroom, self.coin_box_group)

    # Αλλάζει την κατεύθυνση του μανιταριού κατά τη σύγκρουση με κάποιο
άλλο αντικείμενο
    def adjust_mushroom_for_collision_x(self, item, collider):
        if item.rect.x < collider.rect.x:
            item.rect.right = collider.rect.x
            item.direction = c.LEFT
        else:
            item.rect.x = collider.rect.right
            item.direction = c.RIGHT

    # Αλλάζει την κατάσταση του μανιταριού σε SLIDE
    def adjust_mushroom_for_collision_y(self, item, collider):
        item.rect.bottom = collider.rect.y
        item.state = c.SLIDE
        item.y_vel = 0

    # Μετακινεί το αστέρι που κάνει τον Ηρακλή άτρωτος και εξετάζει για
συγκρούσεις
```

```
    def adjust_star_position(self, star):

        if star.state == c.BOUNCE:
            star.rect.x += star.x_vel
            self.check_mushroom_x_collisions(star)
            star.rect.y += star.y_vel
            self.check_star_y_collisions(star)
            star.y_vel += star.gravity
            self.delete_if_off_screen(star)

    # Σύγκρουση ου ατεριού με αλλα αντικείμενα
    def check_star_y_collisions(self, star):
        collider = pg.sprite.spritecollideany(star,
self.ground_step_pipe_group)
        brick = pg.sprite.spritecollideany(star, self.brick_group)
        coin_box = pg.sprite.spritecollideany(star, self.coin_box_group)

        if collider:
            self.adjust_star_for_collision_y(star, collider)
        elif brick:
            self.adjust_star_for_collision_y(star, brick)
        elif coin_box:
            self.adjust_star_for_collision_y(star, coin_box)

    def adjust_star_for_collision_y(self, star, collider):
        if star.rect.y > collider.rect.y:
            star.rect.y = collider.rect.bottom
            star.y_vel = 0
        else:
            star.rect.bottom = collider.rect.top
            star.start_bounce(-8)

    # Μετακινεί τις fireball και ελέγχει για συγκρούσεις
    def adjust_fireball_position(self, fireball):
        if fireball.state == c.FLYING:
            fireball.rect.x += fireball.x_vel
            self.check_fireball_x_collisions(fireball)
            fireball.rect.y += fireball.y_vel
            self.check_fireball_y_collisions(fireball)
        elif fireball.state == c.BOUNCING:
            fireball.rect.x += fireball.x_vel
            self.check_fireball_x_collisions(fireball)
            fireball.rect.y += fireball.y_vel
            self.check_fireball_y_collisions(fireball)
            fireball.y_vel += fireball.gravity
        self.delete_if_off_screen(fireball)

    # Κάνει την fireball να αναπηδά θέτοντάς την σε κατάσταση BOUNCING και
αρχικοποιώντας
    # το y_val και το x_vel
    def bounce_fireball(self, fireball):
        fireball.y_vel = -8
        if fireball.direction == c.RIGHT:
            fireball.x_vel = 15
        else:
            fireball.x_vel = -15

        if fireball in self.powerup_group:
            fireball.state = c.BOUNCING

    # Ελέγχει αν η fireball συγκρούστηκε με κάποιο αντικείμενο
```

```
    def check_fireball_x_collisions(self, fireball):
        collide_group = pg.sprite.Group(self.ground_group,
                                        self.step_group,
                                        self.coin_box_group,
                                        self.brick_group)

        collider = pg.sprite.spritecollideany(fireball, collide_group)
        # Αν η fireball χτυπήσει σε step, σωλήνα, coin box και brick τότε
εξαφανίζεται
        if collider:
            fireball.kill()
            self.sprites_about_to_die_group.add(fireball)
            fireball.explode_transition()

    # Ελέγχει αν υπάρχει σύγκρουση των fireball με αντικείμενο στον άξονα
των y
    def check_fireball_y_collisions(self, fireball):
        collide_group = pg.sprite.Group(self.ground_group,
                                        self.step_group,
                                        self.coin_box_group,
                                        self.brick_group)

        collider = pg.sprite.spritecollideany(fireball, collide_group)
        enemy = pg.sprite.spritecollideany(fireball, self.enemy_group)
        shell = pg.sprite.spritecollideany(fireball, self.shell_group)

        # Αν η fireball συγκρουστεί με το έδαφος τότε αναπηδά
        if collider and (fireball in self.powerup_group):
            fireball.rect.bottom = collider.rect.y
            self.bounce_fireball(fireball)

        # Αν η fireball συγκρουστεί με έναν εχθρό τότε τον σκοτώνει
        elif enemy:
            self.fireball_kill(fireball, enemy)

        elif shell:
            self.fireball_kill(fireball, shell)

    # Σκοτώνει τον εχθρό αν συγκρουστεί μαζί του μία fireball
    def fireball_kill(self, fireball, enemy):
        setup.SFX['kick'].play()
        self.game_info[c.SCORE] += 100
        self.moving_score_list.append(
            score.Score(enemy.rect.centerx - self.viewport.x,
                        enemy.rect.y, 100))
        fireball.kill()
        enemy.kill()
        self.sprites_about_to_die_group.add(enemy, fireball)
        enemy.start_death_jump(fireball.direction)
        fireball.explode_transition()

    def check_if_falling(self, sprite, sprite_group):
        sprite.rect.y += 1

        if pg.sprite.spritecollideany(sprite, sprite_group) is None:
            if sprite.state != c.JUMP:
                sprite.state = c.FALL

        sprite.rect.y -= 1

    # Αφαιρεί ένα αντικείμενο αν φύγει από την οθόνη
```

```
    def delete_if_off_screen(self, enemy):

        if enemy.rect.x < (self.viewport.x - 300):
            enemy.kill()

        elif enemy.rect.y > (self.viewport.bottom):
            enemy.kill()

        elif enemy.state == c.SHELL_SLIDE:
            if enemy.rect.x > (self.viewport.right + 500):
                enemy.kill()

    # Προσαρμώσει την κατάσταση του Ηρακλή όταν αυτός κατεβάσει τη σημαία
στο τέλος του level
    def check_flag(self):
        if (self.flag.state == c.BOTTOM_OF_POLE and self.hercules.state ==
c.FLAGPOLE):
            self.hercules.set_state_to_bottom_of_pole()

    # Προσθέτει το flag score στο συνολικό score
    def check_to_add_flag_score(self):
        if self.flag_score.y_vel == 0:
            self.game_info[c.SCORE] += self.flag_score_total
            self.flag_score_total = 0

    # Εξετάζει αν ο Ηρακλής έχει σκοτωθεί (οπότε το frame έχει φύγει από
την οθόνη)
    # και κάνει restart το level
    def check_for_hercules_death(self):
        if self.hercules.rect.y > c.SCREEN_HEIGHT and not
self.hercules.in_castle:
            self.hercules.dead = True
            self.hercules.x_vel = 0
            self.state = c.FROZEN
            self.game_info[c.HERCULES_DEAD] = True

        if self.hercules.dead:
            self.play_death_song()

    def play_death_song(self):
        if self.death_timer == 0:
            self.death_timer = self.current_time
        elif (self.current_time - self.death_timer) > 3000:
            self.set_game_info_values()
            self.done = True

    # Αρχικοποιεί τις τιμές του παιχνιδιού όταν ο παίκτης χάσει όλες του
τις ζωές
    def set_game_info_values(self):
        if self.game_info[c.SCORE] > self.persist[c.TOP_SCORE]:
            self.persist[c.TOP_SCORE] = self.game_info[c.SCORE]
        if self.hercules.dead:
            self.persist[c.LIVES] -= 1

        if self.persist[c.LIVES] == 0:
            self.next = c.GAME_OVER
            self.game_info[c.CAMERA_START_X] = 0
        elif self.hercules.dead == False:
            self.next = c.MAIN_MENU
            self.game_info[c.CAMERA_START_X] = 0
        elif self.overhead_info_display.time == 0:
```

```
            self.next = c.TIME_OUT
        else:
            if self.hercules.rect.x > 3670 and
self.game_info[c.CAMERA_START_X] == 0:
                self.game_info[c.CAMERA_START_X] = 2600
            self.next = c.LOAD_SCREEN

    # Εξετάζει αν έχει εξαντληθεί ο χρόνος και ο παίκτης δεν έχει
ολοκληρώσει την πίστα
    def check_if_time_out(self):
        if self.overhead_info_display.time <= 0 \
                and not self.hercules.dead \
                and not self.hercules.in_castle:
            self.state = c.FROZEN
            self.hercules.start_death_jump(self.game_info)

    # Αλλάζει το viewport
    def update_viewport(self):
        third = self.viewport.x + self.viewport.w // 3
        hercules_center = self.hercules.rect.centerx
        hercules_right = self.hercules.rect.right

        if self.hercules.x_vel > 0 and hercules_center >= third:
            mult = 0.7 if hercules_right < self.viewport.centerx else 1.2
            new = self.viewport.x + mult * self.hercules.x_vel
            highest = self.level_rect.w - self.viewport.w
            self.viewport.x = min(highest, new)

    # Κάνει update όταν ο Ηρακλής έχει ολοκληρώσει την πίστα
    def update_while_in_castle(self):
        for score in self.moving_score_list:
            score.update(self.moving_score_list, self.game_info)
        self.overhead_info_display.update(self.game_info)

        if self.overhead_info_display.state == c.END_OF_LEVEL:
            self.state = c.FLAG_AND_FIREWORKS
            self.flag_pole_group.add(castle_flag.Flag(8745, 322))

    def update_flag_and_fireworks(self):
        for score in self.moving_score_list:
            score.update(self.moving_score_list, self.game_info)
        self.overhead_info_display.update(self.game_info)
        self.flag_pole_group.update()

        self.end_game()

    # Τέλος της συγκεκριμένης πίστας
    def end_game(self):
        if self.flag_timer == 0:
            self.flag_timer = self.current_time
        elif (self.current_time - self.flag_timer) > 2000:
            self.set_game_info_values()
            self.next = c.LABOUR_HYDRA
            self.sound_manager.stop_music()
            self.done = True

    # Τοποθετεί όλατα τα αντικείμενα στην οθόνη
    def blit_everything(self, surface):
        self.level.blit(self.background, self.viewport, self.viewport)
        if self.flag_score:
            self.flag_score.draw(self.level)
```

```
        self.powerup_group.draw(self.level)
        self.brick_group.draw(self.level)
        self.coin_group.draw(self.level)
        self.coin_box_group.draw(self.level)
        self.sprites_about_to_die_group.draw(self.level)
        self.shell_group.draw(self.level)
        # self.check_point_group.draw(self.level)
        self.brick_pieces_group.draw(self.level)
        self.flag_pole_group.draw(self.level)
        self.hercules_and_enemy_group.draw(self.level)

        surface.blit(self.level, (0, 0), self.viewport)
        self.overhead_info_display.draw(surface)
        for score in self.moving_score_list:
            score.draw(surface)
from __future__ import division

import pygame as pg
from .. import setup, tools
from .. import constants as c
from .. import game_sound
from ..components import hercules
from ..components import collider
from ..components import bricks
from ..components import coin_box
from ..components import coin
from ..components import enemies
from ..components import checkpoint
from ..components import flagpole
from ..components import info
from ..components import score
from ..components import castle_flag


# Η κλάση αντιπροσωπεύει το πρώτο level (πίστα) του παιχνιδιού
class Level3(tools._State):
    def __init__(self):
        tools._State.__init__(self)

    # Εκτελείται για την αρχικοποίηση
    def startup(self, current_time, persist):
        self.game_info = persist
        self.persist = self.game_info
        self.game_info[c.CURRENT_TIME] = current_time
        self.game_info[c.LEVEL_STATE] = c.NOT_FROZEN
        self.game_info[c.HERCULES_DEAD] = False

        self.state = c.NOT_FROZEN
        self.death_timer = 0
        self.flag_timer = 0
        self.flag_score = None
        self.flag_score_total = 0

        self.moving_score_list = []
        self.overhead_info_display = info.OverheadInfo(self.game_info,
c.LEVEL)
        self.overhead_info_display.world3 = True
        self.sound_manager = game_sound.Sound(self.overhead_info_display)

        self.setup_background()
        self.setup_ground()
```

```
        self.setup_pipes()
        self.setup_steps()
        self.setup_bricks()
        self.setup_coin_boxes()
        self.setup_static_coins()
        self.setup_flag_pole()
        self.setup_enemies()
        self.setup_firestick()
        self.setup_hercules()
        self.setup_checkpoints()
        self.setup_spritegroups()

    # Ορίζει την εικόνα του background και το viewport
    def setup_background(self):
        self.background = setup.GFX['level_3']
        self.back_rect = self.background.get_rect()
        self.background = pg.transform.scale(self.background,
                                             (int(self.back_rect.width *
c.BACKGROUND_MULTIPLER),
                                              int(self.back_rect.height *
c.BACKGROUND_MULTIPLER)))
        self.back_rect = self.background.get_rect()
        width = self.back_rect.width
        height = self.back_rect.height

        self.level = pg.Surface((width, height)).convert()
        self.level_rect = self.level.get_rect()
        self.viewport =
setup.SCREEN.get_rect(bottom=self.level_rect.bottom)
        self.viewport.x = self.game_info[c.CAMERA_START_X]

    # Δημιουργεί διάφανα rectangles προκειμένου τα αντικείμενα (Ηρακλής και
enemies) να
    # κοινούνται πάνω σε αυτό
    def setup_ground(self):
        ground_rect1 = collider.Collider(0, 409, 556, 191)
        ground_rect2 = collider.Collider(0, 366, 215, 43)
        ground_rect3 = collider.Collider(0, 323, 172, 43)
        ground_rect4 = collider.Collider(0, 280, 129, 43)
        ground_rect5 = collider.Collider(623, 409, 470, 191)
        ground_rect6 = collider.Collider(1222, 409, 129, 191)
        ground_rect7 = collider.Collider(1480, 409, 2956, 191)
        ground_rect8 = collider.Collider(1480, 366, 1584, 43)
        ground_rect9 = collider.Collider(4436, 537, 1031, 60)
        ground_rect10 = collider.Collider(4952, 409, 170, 130)
        ground_rect11 = collider.Collider(5252, 409, 213, 130)
        ground_rect12 = collider.Collider(6024, 366, 126, 170)
        ground_rect13 = collider.Collider(6024, 537, 820, 60)


        self.ground_group = pg.sprite.Group(ground_rect1, ground_rect2,
ground_rect3,
                                            ground_rect4, ground_rect5,
ground_rect6,
                                            ground_rect7, ground_rect8,
ground_rect9,
                                            ground_rect10, ground_rect11,
ground_rect12,
                                            ground_rect13)

    # "Σημειώνει" τους σωλήνες που υπάρχουν πάνω στο έδαφος ώστε να μπορούν
```

```
    # να αλληλεπιδρούν τα αντικείμενα (Ηρακλής και enemies με αυτά)
    def setup_pipes(self):
        pass

    # "Σημειώνει" τσ εμπόδια (τουβλάκια) που υπάρχουν πάνω στο έδαφος ώστε
να μπορούν
    # να αλληλεπιδρούν τα αντικείμενα (Ηρακλής και enemies με αυτά)
    def setup_steps(self):
        step1 = collider.Collider(0, 63, 1008, 127)
        step2 = collider.Collider(966, 190, 41, 87)
        step3 = collider.Collider(1008, 63, 556, 41)
        step4 = collider.Collider(1566, 63, 1502, 170)
        step5 = collider.Collider(1566, 234, 41, 41)
        step6 = collider.Collider(2080, 234, 41, 43)
        step7 = collider.Collider(2552, 234, 41, 43)
        step8 = collider.Collider(2852, 234, 41, 43)
        step9 = collider.Collider(3066, 63, 3778, 40)
        step10 = collider.Collider(3238, 366, 41, 43)
        step11 = collider.Collider(3409, 104, 41, 86)
        step12 = collider.Collider(3581, 366, 41, 43)
        step13 = collider.Collider(3752, 104, 41, 86)
        step14 = collider.Collider(3924, 366, 41, 43)
        step15 = collider.Collider(3580, 366, 41, 43)
        step16 = collider.Collider(4138, 104, 297, 86)
        step17 = collider.Collider(5252, 104, 213, 86)
        step18 = collider.Collider(6067, 104, 84, 128)
        step19 = collider.Collider(5468, 409, 540, 44)

        self.step_group = pg.sprite.Group(step1, step2, step3, step4,
                                          step5, step6, step7, step8,
                                          step9, step10, step11, step12,
                                          step13, step14, step15, step16,
                                          step17, step18, step19)

    # Δημιουργεί όλα τα breakable τούβλα της συγκεκριμένης πίστας
    def setup_bricks(self):
        self.powerup_group = pg.sprite.Group()
        self.brick_pieces_group = pg.sprite.Group()
        brick1 = bricks.Brick(6492, 492)
        self.brick_group = pg.sprite.Group(brick1)

    # Δημιουργεί όλα τα coin boxes που μπορεί να περιέχουν και mushroom ή
flower
    def setup_coin_boxes(self):
        coin_box1 = coin_box.Coin_box(728, 267, c.MUSHROOM,
self.powerup_group)
        coin_box2 = coin_box.Coin_box(5036, 250, c.MUSHROOM,
self.powerup_group)
        self.coin_box_group = pg.sprite.Group(coin_box1, coin_box2)

    def setup_static_coins(self):
        self.coin_group = pg.sprite.Group()


    # Δημιουγεί τον ιστό και την σημεία πάνω στον ιστό στο τέλος της πίστας
    def setup_flag_pole(self):
        self.flag = flagpole.Flag(6510, 135)


        pole1 = flagpole.Pole(6510, 137)
        pole2 = flagpole.Pole(6510, 177)
```

```
        pole3 = flagpole.Pole(6510, 217)
        pole4 = flagpole.Pole(6510, 257)
        pole5 = flagpole.Pole(6510, 297)
        pole6 = flagpole.Pole(6510, 337)
        pole7 = flagpole.Pole(6510, 377)
        pole8 = flagpole.Pole(6510, 417)
        pole9 = flagpole.Pole(6510, 450)
        finial = flagpole.Finial(6513, 137)
        self.flag_pole_group = pg.sprite.Group(self.flag, finial, pole1,
                                               pole2, pole3, pole4, pole5,
pole6,
                                               pole7, pole8, pole9)

    # Δημιουργεί όλους τους enemies και τους διατηρεί σε μία λίστα
    def setup_enemies(self):
        hydra = enemies.Hydra(self,x = 5880, y = 405, start_position=5910,
end_position=5733)
        enemy_group0 = pg.sprite.Group(hydra)

        self.enemy_group_list = [enemy_group0]


    def setup_firestick(self):
        fire0 = enemies.FireStick(1275, 422, 0)
        fire1 = enemies.FireStick(1275, 422, 21)
        fire2 = enemies.FireStick(1275, 422, 42)
        fire3 = enemies.FireStick(1275, 422, 63)
        fire4 = enemies.FireStick(1275, 422, 84)
        fire5 = enemies.FireStick(1275, 422, 105)
        fire6 = enemies.FireStick(1275, 422, 126)

        fire7 = enemies.FireStick(2091, 249, 0)
        fire8 = enemies.FireStick(2091, 249, 21)
        fire9 = enemies.FireStick(2091, 249, 42)
        fire10 = enemies.FireStick(2091, 249, 63)
        fire11 = enemies.FireStick(2091, 249, 84)
        fire12 = enemies.FireStick(2091, 249, 105)
        fire13 = enemies.FireStick(2091, 249, 126)

        fire14 = enemies.FireStick(2562, 249, 0)
        fire15 = enemies.FireStick(2562, 249, 21)
        fire16 = enemies.FireStick(2562, 249, 42)
        fire17 = enemies.FireStick(2562, 249, 63)
        fire18 = enemies.FireStick(2562, 249, 84)
        fire19 = enemies.FireStick(2562, 249, 105)
        fire20 = enemies.FireStick(2562, 249, 126)

        fire21 = enemies.FireStick(2862, 249, 0)
        fire22 = enemies.FireStick(2862, 249, 21)
        fire23 = enemies.FireStick(2862, 249, 42)
        fire24 = enemies.FireStick(2862, 249, 63)
        fire25 = enemies.FireStick(2862, 249, 84)
        fire26 = enemies.FireStick(2862, 249, 105)
        fire27 = enemies.FireStick(2862, 249, 126)

        fire28 = enemies.FireStick(3250, 380, 0)
        fire29 = enemies.FireStick(3250, 380, 21)
        fire30 = enemies.FireStick(3250, 380, 42)
        fire31 = enemies.FireStick(3250, 380, 63)
        fire32 = enemies.FireStick(3250, 380, 84)
        fire33 = enemies.FireStick(3250, 380, 105)
```

```
        fire34 = enemies.FireStick(3250, 380, 126)

        fire35 = enemies.FireStick(3592, 380, 0)
        fire36 = enemies.FireStick(3592, 380, 21)
        fire37 = enemies.FireStick(3592, 380, 42)
        fire38 = enemies.FireStick(3592, 380, 63)
        fire39 = enemies.FireStick(3592, 380, 84)
        fire40 = enemies.FireStick(3592, 380, 105)
        fire41 = enemies.FireStick(3592, 380, 126)

        fire42 = enemies.FireStick(3762, 163, 0)
        fire43 = enemies.FireStick(3762, 163, 21)
        fire44 = enemies.FireStick(3762, 163, 42)
        fire45 = enemies.FireStick(3762, 163, 63)
        fire46 = enemies.FireStick(3762, 163, 84)
        fire47 = enemies.FireStick(3762, 163, 105)
        fire48 = enemies.FireStick(3762, 163, 126)



        firestick_group0 = pg.sprite.Group(fire0, fire1, fire2, fire3,
fire4, fire5, fire6)
        firestick_group1 = pg.sprite.Group(fire7, fire8, fire9, fire10,
fire11, fire12, fire13)
        firestick_group2 = pg.sprite.Group(fire14, fire15, fire16, fire17,
fire18, fire19, fire20)
        firestick_group3 = pg.sprite.Group(fire21, fire22, fire23, fire24,
fire25, fire26, fire27)
        firestick_group4 = pg.sprite.Group(fire28, fire29, fire30, fire31,
fire32, fire33, fire34)
        firestick_group5 = pg.sprite.Group(fire35, fire36, fire37, fire38,
fire39, fire40, fire41)
        firestick_group6 = pg.sprite.Group(fire42, fire43, fire44, fire45,
fire46, fire47, fire48)

        self.firestick_group_list = [firestick_group0, firestick_group1,
                                     firestick_group2, firestick_group3,
                                     firestick_group4, firestick_group5,
                                     firestick_group6]







    # Δημιουργεί τον χαρακτήρα του Ηρακλή και τον τοποθετεί στην αρχή του
level 1
    def setup_hercules(self):
        self.hercules = hercules.Hercules()
        self.hercules.rect.x = self.viewport.x + 150
        self.hercules.rect.bottom = 409

    # Δημιουργεί invisible checkpoints με τα οποία όταν ο Ηρακλής έρθει σε
επαφή θα
    # προκαλέσει τη δημιουργία εχθρών από την self.enemy_group_list
    def setup_checkpoints(self):
        check1 = checkpoint.Checkpoint(577, "1")
        check2 = checkpoint.Checkpoint(1050, '2')
        check3 = checkpoint.Checkpoint(1700, '3')
        check4 = checkpoint.Checkpoint(2162, '4')
        check5 = checkpoint.Checkpoint(2550, '5')
```

```
        check6 = checkpoint.Checkpoint(2892, '6')
        check7 = checkpoint.Checkpoint(3062, '7')
        check8 = checkpoint.Checkpoint(3900, '8')
        #check9 = checkpoint.Checkpoint(x = 6026, name = '9', y = 324,
width = 40, height = 40)
        check9 = checkpoint.Checkpoint(5090, '9')
        check10 = checkpoint.Checkpoint(6510, '10')
        check11 = checkpoint.Checkpoint(6827, '11')

        self.check_point_group = pg.sprite.Group(check1, check2, check3,
                                                 check4, check5, check6,
                                                 check7, check8, check9,
                                                 check10, check11)

    # Δημιουργούμε sprite groups για να τα ομαδοποιήσουμε
    # και να μπορούμε να τα χειριστούμε με μεγαλύτερη ευκολία
    def setup_spritegroups(self):
        self.sprites_about_to_die_group = pg.sprite.Group()
        self.shell_group = pg.sprite.Group()
        self.enemy_group = pg.sprite.Group()
        self.firestick_group = pg.sprite.Group()
        self.ground_step_pipe_group = pg.sprite.Group(self.ground_group,
self.step_group)

        self.hercules_and_enemy_group = pg.sprite.Group(self.hercules,
self.enemy_group, self.firestick_group)

    # Κάνει update ολόκληρο το level. Καλειται από την tools.Control κλάση
    def update(self, surface, keys, current_time):
        self.game_info[c.CURRENT_TIME] = self.current_time = current_time
        self.handle_states(keys)
        self.check_if_time_out()
        self.blit_everything(surface)
        self.sound_manager.update(self.game_info, self.hercules)

    # Εκτελείται η αντίστοιχη λειτουργία ανάλογα με την κατάσταση του level
    def handle_states(self, keys):
        if self.state == c.FROZEN:
            self.update_during_transition_state(keys)
        elif self.state == c.NOT_FROZEN:
            self.update_all_sprites(keys)
        elif self.state == c.IN_CASTLE:
            self.update_while_in_castle()
        elif self.state == c.FLAG_AND_FIREWORKS:
            self.update_flag_and_fireworks()

    # Κάνει upadate τον Ηρακλή καθώς αυτός βρίσκεται σε transition
κατάσταση
    # γίνεται μεγάλος΄η μικρός ή πεθαίνει ή αποκτά πυροβόλο
    def update_during_transition_state(self, keys):

        self.hercules.update(keys, self.game_info, self.powerup_group)

        for score in self.moving_score_list:
            score.update(self.moving_score_list, self.game_info)
        if self.flag_score:
            self.flag_score.update(None, self.game_info)
            self.check_to_add_flag_score()
        # Κάνει update τα coins και την σημαία στο τέλος του level
        self.coin_box_group.update(self.game_info)
        self.flag_pole_group.update(self.game_info)
```

```
        # Εξετάζει μήπως ο Ηρακλής δεν είναι πλέον σε κατάσταση transition
        self.check_if_hercules_in_transition_state()
        # Εξετάζει αν ο Ηρακλής έχει κατεβάσει τη σημαία
        self.check_flag()
        # Εξετάζει αν ο Ηρακλής έχει σκοτωθεί
        self.check_for_hercules_death()
        # Τυπώνει τις overhead πληροφορίες στην οθόνη
        self.overhead_info_display.update(self.game_info, self.hercules)

    # Εξετάζει αν ο Ηρακλής είναι σε κατάσταση transition και θέτει την
κατάσταση του level σε FROZEN
    # Διαφορετικά θέτει την κατάσταση του level σε NOT FROZEN
    def check_if_hercules_in_transition_state(self):
        if self.hercules.in_transition_state:
            self.game_info[c.LEVEL_STATE] = self.state = c.FROZEN
        elif self.hercules.in_transition_state == False:
            if self.state == c.FROZEN:
                self.game_info[c.LEVEL_STATE] = self.state = c.NOT_FROZEN

    # Το level είναι σε κατάσταση NOT FROZEN οπότε όλα τα sprites
    # ανανεώνονται μαζί και η θέση τους
    def update_all_sprites(self, keys):
        self.hercules.update(keys, self.game_info, self.powerup_group)
        for score in self.moving_score_list:
            score.update(self.moving_score_list, self.game_info)
        if self.flag_score:
            self.flag_score.update(None, self.game_info)
            self.check_to_add_flag_score()
        self.flag_pole_group.update()
        # Ελεγχος αν ο Ηρακλής περασε από τα checkpoints
        self.check_points_check()
        self.enemy_group.update(self.game_info)
        self.firestick_group.update(self.game_info)
        self.sprites_about_to_die_group.update(self.game_info,
self.viewport)
        self.shell_group.update(self.game_info)
        self.brick_group.update()
        self.coin_box_group.update(self.game_info)
        self.powerup_group.update(self.game_info, self.viewport)
        self.coin_group.update(self.game_info)
        self.brick_pieces_group.update()
        self.adjust_sprite_positions()
        self.check_if_hercules_in_transition_state()
        self.check_for_hercules_death()
        self.update_viewport()
        self.overhead_info_display.update(self.game_info, self.hercules)

    # Εξετάζει αν ο Ηρακλής έχει περάσει από κάποιο checkpoint ώστε να
    # δημιουργηθούν οι enemies
    def check_points_check(self):
        checkpoint = pg.sprite.spritecollideany(self.hercules,
self.check_point_group)

        # Αν έχει περάσει από checkpoint
        if checkpoint:
            checkpoint.kill()
            # Τα checkpoint 1 εως 10 πυροδοτούν την δημιουργία των enemies
που έχουμε
            # ορίσει παραπάνω. Οπότε μόλις περάσουμε ένα από αυτά τα
checkpoints δημιουργούνται
            # τα αντίστοιχα enemies
```

```
            for i in range(1, 8):
                if checkpoint.name == str(i):
                    for index, enemy in
enumerate(self.firestick_group_list[i - 1]):
                        enemy.rect.x = self.viewport.right + (index * 60)
                    self.firestick_group.add(self.firestick_group_list[i -
1])
            # Εμφάνιση της Ύδρας
            if checkpoint.name == '9':
                for index, enemy in enumerate(self.enemy_group_list[0]):
                    enemy.rect.x = enemy.start_position
                self.enemy_group.add(self.enemy_group_list[0])

            # Το checkpoint 11 αφορά τον ιστό στο τέλος του level
            if checkpoint.name == '10':
                self.hercules.state = c.FLAGPOLE
                self.hercules.invincible = False
                self.hercules.flag_pole_right = checkpoint.rect.right
                if self.hercules.rect.bottom < self.flag.rect.y:
                    self.hercules.rect.bottom = self.flag.rect.y
                self.flag.state = c.SLIDE_DOWN
                # Εμφανίζει το σκορ που κέρδισε ο Ηρακλής στον ιστό
                self.create_flag_points()

            # Το checkpoint 12 σημαίνει ότι ο Ηρακλής έφτασε στο κάστρο και
            # έχει ολοκληρωθεί το level
            elif checkpoint.name == '11':
                self.state = c.IN_CASTLE
                self.hercules.kill()
                self.hercules.state == c.STAND
                self.hercules.in_castle = True
                self.overhead_info_display.state = c.FAST_COUNT_DOWN

            # Σε διαφορετική περίπτωση σχετίζεται με το κρυφή live
            elif checkpoint.name == 'secret_mushroom' and
self.hercules.y_vel < 0:
                mushroom_box = coin_box.Coin_box(checkpoint.rect.x,
                                                 checkpoint.rect.bottom -
40,
                                                 '1up_mushroom',
                                                 self.powerup_group)
                mushroom_box.start_bump(self.moving_score_list)
                self.coin_box_group.add(mushroom_box)

                self.hercules.y_vel = 7
                self.hercules.rect.y = mushroom_box.rect.bottom
                self.hercules.state = c.FALL

            self.hercules_and_enemy_group.add(self.enemy_group)

    # Δημιουργεί τους πόντους που εμφανίζνται όταν ο Ηρακλής πηδάει στον
ιστό
    # Οι πόντοι εξαρτώνται από το πόσο ψυλά φτάνει στον ιστό
    def create_flag_points(self):

        x = 8518
        y = c.GROUND_HEIGHT - 60
        hercules_bottom = self.hercules.rect.bottom

        if hercules_bottom > (c.GROUND_HEIGHT - 40 - 40):
            self.flag_score = score.Score(x, y, 100, True)
```

```
            self.flag_score_total = 100
        elif hercules_bottom > (c.GROUND_HEIGHT - 40 - 160):
            self.flag_score = score.Score(x, y, 400, True)
            self.flag_score_total = 400
        elif hercules_bottom > (c.GROUND_HEIGHT - 40 - 240):
            self.flag_score = score.Score(x, y, 800, True)
            self.flag_score_total = 800
        elif hercules_bottom > (c.GROUND_HEIGHT - 40 - 360):
            self.flag_score = score.Score(x, y, 2000, True)
            self.flag_score_total = 2000
        else:
            self.flag_score = score.Score(x, y, 5000, True)
            self.flag_score_total = 5000

    def adjust_sprite_positions(self):
        self.adjust_hercules_position()
        self.adjust_enemy_position()
        self.adjust_powerup_position()

    # Προσαρμόζει την θέση του Ηρακλή ανάλογα με την ταχύτητά του και τις
πιθανές συγκρούσεις
    def adjust_hercules_position(self):
        self.last_x_position = self.hercules.rect.right
        self.hercules.rect.x += round(self.hercules.x_vel)
        self.check_hercules_x_collisions()

        if self.hercules.in_transition_state == False:
            self.hercules.rect.y += round(self.hercules.y_vel)
            self.check_hercules_y_collisions()

        if self.hercules.rect.x < (self.viewport.x + 5):
            self.hercules.rect.x = (self.viewport.x + 5)

    # Εξετάζει αν ο Ηρακλής συγκρούεται με ενα αντικείμενο κατά την
οριζόντια κίνηση
    def check_hercules_x_collisions(self):
        collider = pg.sprite.spritecollideany(self.hercules,
self.ground_step_pipe_group)
        coin_box = pg.sprite.spritecollideany(self.hercules,
self.coin_box_group)
        brick = pg.sprite.spritecollideany(self.hercules, self.brick_group)
        coin = pg.sprite.spritecollideany(self.hercules, self.coin_group)
        enemy = pg.sprite.spritecollideany(self.hercules, self.enemy_group)
        powerup = pg.sprite.spritecollideany(self.hercules,
self.powerup_group)
        fire = pg.sprite.spritecollideany(self.hercules,
self.firestick_group)
        if coin_box:
            self.adjust_hercules_for_x_collisions(coin_box)

        elif coin:
            setup.SFX['coin'].play()
            self.game_info[c.SCORE] += 100
            self.game_info[c.COIN_TOTAL] += 1

self.moving_score_list.append(score.Score(self.hercules.rect.centerx -
self.viewport.x, self.hercules.rect.y, 100))
            coin.kill()

        elif brick:
            self.adjust_hercules_for_x_collisions(brick)
```

```
        elif collider:
            self.adjust_hercules_for_x_collisions(collider)

        # Αν έρθει σε επαφή με εχθρό
        elif enemy or fire:
            # Ο Ηρακλής είναι σε κατάσταση ανίκητος (μετά από αστέρι)
            if self.hercules.invincible:
                setup.SFX['kick'].play()
                self.game_info[c.SCORE] += 100
                self.moving_score_list.append(
                    score.Score(self.hercules.rect.right - self.viewport.x,
                                self.hercules.rect.y, 100))
                enemy.kill()
                enemy.start_death_jump(c.RIGHT)
                self.sprites_about_to_die_group.add(enemy)
            # Ο Ηρακλής είναι μεγαλος οπότε μικραίνει
            elif self.hercules.big:
                setup.SFX['pipe'].play()
                self.hercules.fire = False
                self.hercules.y_vel = -1
                self.hercules.state = c.BIG_TO_SMALL
                self.convert_fireflowers_to_mushrooms()
            elif self.hercules.hurt_invincible:
                pass
            # Ο Ηρακλής είναι μικρός οπότε χάνει μία ζωή
            else:
                self.hercules.start_death_jump(self.game_info)
                self.state = c.FROZEN

        # Ο Ηρακλής έρχεται σε επαφή με κάποιο powerup
        elif powerup:
            # Αστέρι
            if powerup.name == c.STAR:
                self.game_info[c.SCORE] += 1000

                self.moving_score_list.append(
                    score.Score(self.hercules.rect.centerx -
self.viewport.x,
                                self.hercules.rect.y, 1000))
                self.hercules.invincible = True
                self.hercules.invincible_start_timer = self.current_time
            # Μανιτάρι που μεγαλώνει
            elif powerup.name == c.MUSHROOM:
                setup.SFX['powerup'].play()
                self.game_info[c.SCORE] += 1000
                self.moving_score_list.append(
                    score.Score(self.hercules.rect.centerx -
self.viewport.x,
                                self.hercules.rect.y - 20, 1000))

                self.hercules.y_vel = -1
                self.hercules.state = c.SMALL_TO_BIG
                self.hercules.in_transition_state = True
                self.convert_mushrooms_to_fireflowers()
            # Μαντιτάρι που κερδίζει ζωή
            elif powerup.name == c.LIFE_MUSHROOM:
                self.moving_score_list.append(
                    score.Score(powerup.rect.right - self.viewport.x,
                                powerup.rect.y,
                                c.ONEUP))
```

```
                self.game_info[c.LIVES] += 1
                setup.SFX['one_up'].play()
            # Fire flower που του δίνει πυρομαχικά
            elif powerup.name == c.FIREFLOWER:
                setup.SFX['powerup'].play()
                self.game_info[c.SCORE] += 1000
                self.moving_score_list.append(
                    score.Score(self.hercules.rect.centerx -
self.viewport.x,
                                self.hercules.rect.y, 1000))

                if self.hercules.big and self.hercules.fire == False:
                    self.hercules.state = c.BIG_TO_FIRE
                    self.hercules.in_transition_state = True
                elif self.hercules.big == False:
                    self.hercules.state = c.SMALL_TO_BIG
                    self.hercules.in_transition_state = True
                    self.convert_mushrooms_to_fireflowers()

            if powerup.name != c.FIREBALL:
                powerup.kill()

    # Όταν ο Ηρακλής γίνει μεγάλος τότε μετατρέπει όλα τα mushrooms σε
fireflowers
    def convert_mushrooms_to_fireflowers(self):
        for brick in self.brick_group:
            if brick.contents == c.MUSHROOM:
                brick.contents = c.FIREFLOWER
        for coin_box in self.coin_box_group:
            if coin_box.contents == c.MUSHROOM:
                coin_box.contents = c.FIREFLOWER

    # Όταν ο Ηρακλής γίνει μικρός τότε μετατρέπει όλα τα fireflowers σε
mushroom
    def convert_fireflowers_to_mushrooms(self):
        for brick in self.brick_group:
            if brick.contents == c.FIREFLOWER:
                brick.contents = c.MUSHROOM
        for coin_box in self.coin_box_group:
            if coin_box.contents == c.FIREFLOWER:
                coin_box.contents = c.MUSHROOM

    # Τοποθετεί τον Ηρακλή δίπλα στον collider
    def adjust_hercules_for_x_collisions(self, collider):
        if self.hercules.rect.x < collider.rect.x:
            self.hercules.rect.right = collider.rect.left
        else:
            self.hercules.rect.left = collider.rect.right

        self.hercules.x_vel = 0



    # Εξετάζει αν ο Ηρακλής έρχεται σε επαφή με άλλα αντικείμενα κατά την
κίνηση
    # στον κάθετο άξονα
    def check_hercules_y_collisions(self):
        ground_step_or_pipe = pg.sprite.spritecollideany(self.hercules,
self.ground_step_pipe_group)
        enemy = pg.sprite.spritecollideany(self.hercules, self.enemy_group)
```

```
        brick = pg.sprite.spritecollideany(self.hercules, self.brick_group)
        coin_box = pg.sprite.spritecollideany(self.hercules,
self.coin_box_group)
        powerup = pg.sprite.spritecollideany(self.hercules,
self.powerup_group)
        brick, coin_box = self.prevent_collision_conflict(brick, coin_box)

        if coin_box:
            self.adjust_hercules_for_y_coin_box_collisions(coin_box)

        elif brick:
            self.adjust_hercules_for_y_brick_collisions(brick)

        elif ground_step_or_pipe:

self.adjust_hercules_for_y_ground_pipe_collisions(ground_step_or_pipe)

        elif enemy and enemy.name != c.FIRE:
            if self.hercules.invincible:
                setup.SFX['kick'].play()
                enemy.kill()
                self.sprites_about_to_die_group.add(enemy)
                enemy.start_death_jump(c.RIGHT)
            else:
                self.adjust_hercules_for_y_enemy_collisions(enemy)
        elif powerup:
            if powerup.name == c.STAR:
                setup.SFX['powerup'].play()
                powerup.kill()
                self.hercules.invincible = True
                self.hercules.invincible_start_timer = self.current_time

        self.test_if_hercules_is_falling()

    def prevent_collision_conflict(self, obstacle1, obstacle2):
        if obstacle1 and obstacle2:
            obstacle1_distance = self.hercules.rect.centerx -
obstacle1.rect.centerx
            if obstacle1_distance < 0:
                obstacle1_distance *= -1
            obstacle2_distance = self.hercules.rect.centerx -
obstacle2.rect.centerx
            if obstacle2_distance < 0:
                obstacle2_distance *= -1

            if obstacle1_distance < obstacle2_distance:
                obstacle2 = False
            else:
                obstacle1 = False

        return obstacle1, obstacle2

    # Σύγκρουση του Ηρακλή με τα coin boxes κατά τον y άξονα. Αν το coin
box είναι σε κατασταση Resting και
    # περιέχει κάποιο coin τότε αυξάνεται το πληθος των coin κατά 1 και το
box παίρνει την κατάσταση OPENED
    def adjust_hercules_for_y_coin_box_collisions(self, coin_box):
        if self.hercules.rect.y > coin_box.rect.y:
            if coin_box.state == c.RESTING:
                if coin_box.contents == c.COIN:
                    self.game_info[c.SCORE] += 200
```

```
                    coin_box.start_bump(self.moving_score_list)
                    self.game_info[c.COIN_TOTAL] += 1
                else:
                    coin_box.start_bump(self.moving_score_list)

            elif coin_box.state == c.OPENED:
                pass
            setup.SFX['bump'].play()
            self.hercules.y_vel = 7
            self.hercules.rect.y = coin_box.rect.bottom
            self.hercules.state = c.FALL
        else:
            self.hercules.y_vel = 0
            self.hercules.rect.bottom = coin_box.rect.top
            self.hercules.state = c.WALK

    # Εκτελείται όταν ο Ηρακλής συγκρούεται με ένα brick (τούβλο)
    def adjust_hercules_for_y_brick_collisions(self, brick):
        if self.hercules.rect.y > brick.rect.y:
            if brick.state == c.RESTING:
                if self.hercules.big and brick.contents is None:
                    setup.SFX['brick_smash'].play()
                    self.check_if_enemy_on_brick(brick)
                    brick.kill()
                    self.brick_pieces_group.add(
                        bricks.BrickPiece(brick.rect.x,
                                          brick.rect.y - (brick.rect.height
/ 2),
                                          -2, -12),
                        bricks.BrickPiece(brick.rect.right,
                                          brick.rect.y - (brick.rect.height
/ 2),
                                          2, -12),
                        bricks.BrickPiece(brick.rect.x,
                                          brick.rect.y,
                                          -2, -6),
                        bricks.BrickPiece(brick.rect.right,
                                          brick.rect.y,
                                          2, -6))
                else:
                    setup.SFX['bump'].play()
                    if brick.coin_total > 0:
                        self.game_info[c.COIN_TOTAL] += 1
                        self.game_info[c.SCORE] += 200
                    self.check_if_enemy_on_brick(brick)
                    brick.start_bump(self.moving_score_list)
            elif brick.state == c.OPENED:
                setup.SFX['bump'].play()
            self.hercules.y_vel = 7
            self.hercules.rect.y = brick.rect.bottom
            self.hercules.state = c.FALL

        else:
            self.hercules.y_vel = 0
            self.hercules.rect.bottom = brick.rect.top
            self.hercules.state = c.WALK

    # Αν υπάρχει κάποιος εχθρός ο οποίος είναι πάνω στο brick το οποίο
σπάει ή κουνάει
    # ο Ηρακλής τότε ο εχθρός πεθαίνει
    def check_if_enemy_on_brick(self, brick):
```

```
        brick.rect.y -= 5

        enemy = pg.sprite.spritecollideany(brick, self.enemy_group)

        if enemy and enemy.name != c.FIRE:
            setup.SFX['kick'].play()
            self.game_info[c.SCORE] += 100
            self.moving_score_list.append(
                score.Score(enemy.rect.centerx - self.viewport.x,
                            enemy.rect.y,
                            100))
            enemy.kill()
            self.sprites_about_to_die_group.add(enemy)
            if self.hercules.rect.centerx > brick.rect.centerx:
                enemy.start_death_jump('right')
            else:
                enemy.start_death_jump('left')

        brick.rect.y += 5

    # Ο Ηρακλής έρχεται σε επαφή με σωλήνα
    def adjust_hercules_for_y_ground_pipe_collisions(self, collider):

        if collider.rect.bottom > self.hercules.rect.bottom:
            self.hercules.y_vel = 0
            self.hercules.rect.bottom = collider.rect.top
            if self.hercules.state == c.END_OF_LEVEL_FALL:
                self.hercules.state = c.WALKING_TO_CASTLE
            else:
                self.hercules.state = c.WALK
        elif collider.rect.top < self.hercules.rect.top:
            self.hercules.y_vel = 7
            self.hercules.rect.top = collider.rect.bottom
            self.hercules.state = c.FALL

    def test_if_hercules_is_falling(self):
        self.hercules.rect.y += 1
        test_collide_group = pg.sprite.Group(self.ground_step_pipe_group,
                                             self.brick_group,
self.coin_group,
                                             self.coin_box_group)

        if pg.sprite.spritecollideany(self.hercules, test_collide_group) is
None:
            if self.hercules.state != c.JUMP \
                    and self.hercules.state != c.DEATH_JUMP \
                    and self.hercules.state != c.SMALL_TO_BIG \
                    and self.hercules.state != c.BIG_TO_FIRE \
                    and self.hercules.state != c.BIG_TO_SMALL \
                    and self.hercules.state != c.FLAGPOLE \
                    and self.hercules.state != c.WALKING_TO_CASTLE \
                    and self.hercules.state != c.END_OF_LEVEL_FALL:
                self.hercules.state = c.FALL
            elif self.hercules.state == c.WALKING_TO_CASTLE or \
                    self.hercules.state == c.END_OF_LEVEL_FALL:
                self.hercules.state = c.END_OF_LEVEL_FALL

        self.hercules.rect.y -= 1

    # Ο Ηρακλής έρχεται σε επαφή με τον εχθρό κατά τον άξονα των y
    def adjust_hercules_for_y_enemy_collisions(self, enemy):
```

```
        if self.hercules.y_vel > 0:
            setup.SFX['stomp'].play()
            self.game_info[c.SCORE] += 100
            self.moving_score_list.append(
                score.Score(enemy.rect.centerx - self.viewport.x,
                            enemy.rect.y, 100))
            enemy.state = c.JUMPED_ON
            enemy.kill()
            if enemy.name == c.LION:
                enemy.death_timer = self.current_time
                self.sprites_about_to_die_group.add(enemy)
            elif enemy.name == c.KOOPA:
                self.shell_group.add(enemy)

            self.hercules.rect.bottom = enemy.rect.top
            self.hercules.state = c.JUMP
            self.hercules.y_vel = -7


    # Μετακινεί ολους τους εχθρούς κατά τον άξονα X και Y και εξετάζει αν
έγιναν συγκρούσεις μετά την κίνηση
    # του εχθρού
    def adjust_enemy_position(self):
        for enemy in self.enemy_group:
            if enemy.name == c.HYDRA:
                if enemy.rect.x >= enemy.start_position:
                    enemy.rect.x -= enemy.x_vel
                elif enemy.rect.x <= enemy.end_position:
                    enemy.rect.x += enemy.x_vel
            enemy.rect.x += enemy.x_vel
            self.check_enemy_x_collisions(enemy)
            if enemy.name != c.FIRE:
                enemy.rect.y += enemy.y_vel
                self.check_enemy_y_collisions(enemy)
                self.delete_if_off_screen(enemy)

    # Εξετάζει τις συγκρούσεις των εχθρών μεταξύ τους καθως και με σωλήνες
η steps
    def check_enemy_x_collisions(self, enemy):
        enemy.kill()

        collider = pg.sprite.spritecollideany(enemy,
self.ground_step_pipe_group)
        enemy_collider = pg.sprite.spritecollideany(enemy,
self.enemy_group)

        # Συγκρουση του εχθρού με έναν σωλήνα ή step οπότε αλλάζει
κατεύθυνση και
        # πηγαίνει προς την αντίθετη πλευρά
        if collider:
            if enemy.direction == c.RIGHT:
                enemy.rect.right = collider.rect.left
                enemy.direction = c.LEFT
                enemy.x_vel = -2
            elif enemy.direction == c.LEFT:
                enemy.rect.left = collider.rect.right
                enemy.direction = c.RIGHT
                enemy.x_vel = 2

        # σύγκρουση του εχθρού με κάποιον άλλον εχθρό οπότε και οι δύο
εχθροί θα κινηθούν προς την
```

```
        # αντίθετη κατεύθυνση
        elif enemy_collider:
            if enemy.direction == c.RIGHT:
                enemy.rect.right = enemy_collider.rect.left
                enemy.direction = c.LEFT
                enemy_collider.direction = c.RIGHT
                enemy.x_vel = -2
                enemy_collider.x_vel = 2
            elif enemy.direction == c.LEFT:
                enemy.rect.left = enemy_collider.rect.right
                enemy.direction = c.RIGHT
                enemy_collider.direction = c.LEFT
                enemy.x_vel = 2
                enemy_collider.x_vel = -2

        self.enemy_group.add(enemy)
        self.hercules_and_enemy_group.add(self.enemy_group)

    # Ελέγχει τις συγκρούσεις των εχθρών ως προς τον άξονα των y
    def check_enemy_y_collisions(self, enemy):
        collider = pg.sprite.spritecollideany(enemy,
self.ground_step_pipe_group)
        brick = pg.sprite.spritecollideany(enemy, self.brick_group)
        coin_box = pg.sprite.spritecollideany(enemy, self.coin_box_group)

        if collider:
            if enemy.rect.bottom > collider.rect.bottom:
                enemy.y_vel = 7
                enemy.rect.top = collider.rect.bottom
                enemy.state = c.FALL
            elif enemy.rect.bottom < collider.rect.bottom:

                enemy.y_vel = 0
                enemy.rect.bottom = collider.rect.top
                enemy.state = c.WALK

        elif brick:
            if brick.state == c.BUMPED:
                enemy.kill()
                self.sprites_about_to_die_group.add(enemy)
                if self.hercules.rect.centerx > brick.rect.centerx:
                    enemy.start_death_jump('right')
                else:
                    enemy.start_death_jump('left')

            elif enemy.rect.x > brick.rect.x:
                enemy.y_vel = 7
                enemy.rect.top = brick.rect.bottom
                enemy.state = c.FALL
            else:
                enemy.y_vel = 0
                enemy.rect.bottom = brick.rect.top
                enemy.state = c.WALK

        elif coin_box:
            if coin_box.state == c.BUMPED:
                self.game_info[c.SCORE] += 100
                self.moving_score_list.append(
                    score.Score(enemy.rect.centerx - self.viewport.x,
                                enemy.rect.y, 100))
                enemy.kill()
```

```
                self.sprites_about_to_die_group.add(enemy)
                if self.hercules.rect.centerx > coin_box.rect.centerx:
                    enemy.start_death_jump('right')
                else:
                    enemy.start_death_jump('left')

            elif enemy.rect.x > coin_box.rect.x:
                enemy.y_vel = 7
                enemy.rect.top = coin_box.rect.bottom
                enemy.state = c.FALL
            else:
                enemy.y_vel = 0
                enemy.rect.bottom = coin_box.rect.top
                enemy.state = c.WALK
        else:
            enemy.rect.y += 1
            test_group = pg.sprite.Group(self.ground_step_pipe_group,
                                         self.coin_box_group,
                                         self.brick_group)
            if pg.sprite.spritecollideany(enemy, test_group) is None:
                if enemy.state != c.JUMP:
                    enemy.state = c.FALL

            enemy.rect.y -= 1


    # Μετακινεί μανιτάρια, αστέρια και τις fireballs
    def adjust_powerup_position(self):
        for powerup in self.powerup_group:
            if powerup.name == c.MUSHROOM:
                self.adjust_mushroom_position(powerup)
            elif powerup.name == c.STAR:
                self.adjust_star_position(powerup)
            elif powerup.name == c.FIREBALL:
                self.adjust_fireball_position(powerup)
            elif powerup.name == '1up_mushroom':
                self.adjust_mushroom_position(powerup)

    # Εμφανίζει, μετακινεί το μανιτάρι και εξετάζει τις συγκρούσεις στους
άξονες x, y
    def adjust_mushroom_position(self, mushroom):
        if mushroom.state != c.REVEAL:
            mushroom.rect.x += mushroom.x_vel
            self.check_mushroom_x_collisions(mushroom)

            mushroom.rect.y += mushroom.y_vel
            self.check_mushroom_y_collisions(mushroom)
            self.delete_if_off_screen(mushroom)

    # Εξετάζει τις συγκρούσεις των μανιταριών με τα άλλα αντικείμενα  στον
άξονα των x
    def check_mushroom_x_collisions(self, mushroom):

        collider = pg.sprite.spritecollideany(mushroom,
self.ground_step_pipe_group)
        brick = pg.sprite.spritecollideany(mushroom, self.brick_group)
        coin_box = pg.sprite.spritecollideany(mushroom,
self.coin_box_group)

        # Σύγκρουση με σωλήνα ή step
        if collider:
```

```
            self.adjust_mushroom_for_collision_x(mushroom, collider)
        # Σύγκρουση με brick
        elif brick:
            self.adjust_mushroom_for_collision_x(mushroom, brick)
        # Σύγκρουση με coin box
        elif coin_box:
            self.adjust_mushroom_for_collision_x(mushroom, coin_box)

    # Εξετάζει τις συγκρούσεις του μανιταριού στον άξονα των y
    def check_mushroom_y_collisions(self, mushroom):
        collider = pg.sprite.spritecollideany(mushroom,
self.ground_step_pipe_group)
        brick = pg.sprite.spritecollideany(mushroom, self.brick_group)
        coin_box = pg.sprite.spritecollideany(mushroom,
self.coin_box_group)

        if collider:
            self.adjust_mushroom_for_collision_y(mushroom, collider)
        elif brick:
            self.adjust_mushroom_for_collision_y(mushroom, brick)
        elif coin_box:
            self.adjust_mushroom_for_collision_y(mushroom, coin_box)
        else:
            self.check_if_falling(mushroom, self.ground_step_pipe_group)
            self.check_if_falling(mushroom, self.brick_group)
            self.check_if_falling(mushroom, self.coin_box_group)

    # Αλλάζει την κατεύθυνση του μανιταριού κατά τη σύγκρουση με κάποιο
άλλο αντικείμενο
    def adjust_mushroom_for_collision_x(self, item, collider):
        if item.rect.x < collider.rect.x:
            item.rect.right = collider.rect.x
            item.direction = c.LEFT
        else:
            item.rect.x = collider.rect.right
            item.direction = c.RIGHT

    # Αλλάζει την κατάσταση του μανιταριού σε SLIDE
    def adjust_mushroom_for_collision_y(self, item, collider):
        item.rect.bottom = collider.rect.y
        item.state = c.SLIDE
        item.y_vel = 0

    # Μετακινεί το αστέρι που κάνει τον Ηρακλή άτρωτος και εξετάζει για
συγκρούσεις
    def adjust_star_position(self, star):

        if star.state == c.BOUNCE:
            star.rect.x += star.x_vel
            self.check_mushroom_x_collisions(star)
            star.rect.y += star.y_vel
            self.check_star_y_collisions(star)
            star.y_vel += star.gravity
            self.delete_if_off_screen(star)

    # Σύγκρουση ου ατεριού με αλλα αντικείμενα
    def check_star_y_collisions(self, star):
        collider = pg.sprite.spritecollideany(star,
self.ground_step_pipe_group)
        brick = pg.sprite.spritecollideany(star, self.brick_group)
        coin_box = pg.sprite.spritecollideany(star, self.coin_box_group)
```

```
        if collider:
            self.adjust_star_for_collision_y(star, collider)
        elif brick:
            self.adjust_star_for_collision_y(star, brick)
        elif coin_box:
            self.adjust_star_for_collision_y(star, coin_box)

    def adjust_star_for_collision_y(self, star, collider):
        if star.rect.y > collider.rect.y:
            star.rect.y = collider.rect.bottom
            star.y_vel = 0
        else:
            star.rect.bottom = collider.rect.top
            star.start_bounce(-8)

    # Μετακινεί τις fireball και ελέγχει για συγκρούσεις
    def adjust_fireball_position(self, fireball):
        if fireball.state == c.FLYING:
            fireball.rect.x += fireball.x_vel
            self.check_fireball_x_collisions(fireball)
            fireball.rect.y += fireball.y_vel
            self.check_fireball_y_collisions(fireball)
        elif fireball.state == c.BOUNCING:
            fireball.rect.x += fireball.x_vel
            self.check_fireball_x_collisions(fireball)
            fireball.rect.y += fireball.y_vel
            self.check_fireball_y_collisions(fireball)
            fireball.y_vel += fireball.gravity
        self.delete_if_off_screen(fireball)

    # Κάνει την fireball να αναπηδά θέτοντάς την σε κατάσταση BOUNCING και
αρχικοποιώντας
    # το y_val και το x_vel
    def bounce_fireball(self, fireball):
        fireball.y_vel = -8
        if fireball.direction == c.RIGHT:
            fireball.x_vel = 15
        else:
            fireball.x_vel = -15

        if fireball in self.powerup_group:
            fireball.state = c.BOUNCING

    # Ελέγχει αν η fireball συγκρούστηκε με κάποιο αντικείμενο
    def check_fireball_x_collisions(self, fireball):
        collide_group = pg.sprite.Group(self.ground_group,
                                        self.step_group,
                                        self.coin_box_group,
                                        self.brick_group)

        collider = pg.sprite.spritecollideany(fireball, collide_group)
        # Αν η fireball χτυπήσει σε step, σωλήνα, coin box και brick τότε
εξαφανίζεται
        if collider:
            fireball.kill()
            self.sprites_about_to_die_group.add(fireball)
            fireball.explode_transition()

    # Ελέγχει αν υπάρχει σύγκρουση των fireball με αντικείμενο στον άξονα
των y
```

```
    def check_fireball_y_collisions(self, fireball):
        collide_group = pg.sprite.Group(self.ground_group,
                                        self.step_group,
                                        self.coin_box_group,
                                        self.brick_group)

        collider = pg.sprite.spritecollideany(fireball, collide_group)
        enemy = pg.sprite.spritecollideany(fireball, self.enemy_group)
        shell = pg.sprite.spritecollideany(fireball, self.shell_group)

        # Αν η fireball συγκρουστεί με το έδαφος τότε αναπηδά
        if collider and (fireball in self.powerup_group):
            fireball.rect.bottom = collider.rect.y
            self.bounce_fireball(fireball)

        # Αν η fireball συγκρουστεί με έναν εχθρό τότε τον σκοτώνει
        elif enemy:
            self.fireball_kill(fireball, enemy)

        elif shell:
            self.fireball_kill(fireball, shell)

    # Σκοτώνει τον εχθρό αν συγκρουστεί μαζί του μία fireball
    def fireball_kill(self, fireball, enemy):
        setup.SFX['kick'].play()
        self.game_info[c.SCORE] += 100
        self.moving_score_list.append(
            score.Score(enemy.rect.centerx - self.viewport.x,
                        enemy.rect.y, 100))

        fireball.kill()
        self.sprites_about_to_die_group.add(fireball)
        fireball.explode_transition()
        if enemy.name == c.HYDRA:
            if enemy.lives == 0:
                enemy.kill()
                self.sprites_about_to_die_group.add(enemy, fireball)
                enemy.start_death_jump(fireball.direction)
            else:
                enemy.lives -= 1



    def check_if_falling(self, sprite, sprite_group):
        sprite.rect.y += 1

        if pg.sprite.spritecollideany(sprite, sprite_group) is None:
            if sprite.state != c.JUMP:
                sprite.state = c.FALL

        sprite.rect.y -= 1

    # Αφαιρεί ένα αντικείμενο αν φύγει από την οθόνη
    def delete_if_off_screen(self, enemy):

        if enemy.rect.x < (self.viewport.x - 300):
            enemy.kill()

        elif enemy.rect.y > (self.viewport.bottom):
            enemy.kill()
```

```
        elif enemy.state == c.SHELL_SLIDE:
            if enemy.rect.x > (self.viewport.right + 500):
                enemy.kill()

    # Προσαρμώσει την κατάσταση του Ηρακλή όταν αυτός κατεβάσει τη σημαία
στο τέλος του level
    def check_flag(self):
        if (self.flag.state == c.BOTTOM_OF_POLE and self.hercules.state ==
c.FLAGPOLE):
            self.hercules.set_state_to_bottom_of_pole()

    # Προσθέτει το flag score στο συνολικό score
    def check_to_add_flag_score(self):
        if self.flag_score.y_vel == 0:
            self.game_info[c.SCORE] += self.flag_score_total
            self.flag_score_total = 0

    # Εξετάζει αν ο Ηρακλής έχει σκοτωθεί (οπότε το frame έχει φύγει από
την οθόνη)
    # και κάνει restart το level
    def check_for_hercules_death(self):
        if self.hercules.rect.y > c.SCREEN_HEIGHT and not
self.hercules.in_castle:
            self.hercules.dead = True
            self.hercules.x_vel = 0
            self.state = c.FROZEN
            self.game_info[c.HERCULES_DEAD] = True

        if self.hercules.dead:
            self.play_death_song()

    def play_death_song(self):
        if self.death_timer == 0:
            self.death_timer = self.current_time
        elif (self.current_time - self.death_timer) > 3000:
            self.set_game_info_values()
            self.done = True

    # Αρχικοποιεί τις τιμές του παιχνιδιού όταν ο παίκτης χάσει όλες του
τις ζωές
    def set_game_info_values(self):
        if self.game_info[c.SCORE] > self.persist[c.TOP_SCORE]:
            self.persist[c.TOP_SCORE] = self.game_info[c.SCORE]
        if self.hercules.dead:
            self.persist[c.LIVES] -= 1

        if self.persist[c.LIVES] == 0:
            self.next = c.GAME_OVER
            self.game_info[c.CAMERA_START_X] = 0
        elif self.hercules.dead == False:
            self.next = c.MAIN_MENU
            self.game_info[c.CAMERA_START_X] = 0
        elif self.overhead_info_display.time == 0:
            self.next = c.TIME_OUT
        else:
            if self.hercules.rect.x > 3670 and
self.game_info[c.CAMERA_START_X] == 0:
                self.game_info[c.CAMERA_START_X] = 2600
            self.next = c.LOAD_SCREEN
```

```
    # Εξετάζει αν έχει εξαντληθεί ο χρόνος και ο παίκτης δεν έχει
ολοκληρώσει την πίστα
    def check_if_time_out(self):
        if self.overhead_info_display.time <= 0 \
                and not self.hercules.dead \
                and not self.hercules.in_castle:
            self.state = c.FROZEN
            self.hercules.start_death_jump(self.game_info)

    # Αλλάζει το viewport
    def update_viewport(self):
        third = self.viewport.x + self.viewport.w // 3
        hercules_center = self.hercules.rect.centerx
        hercules_right = self.hercules.rect.right

        if self.hercules.x_vel > 0 and hercules_center >= third:
            mult = 0.7 if hercules_right < self.viewport.centerx else 1.2
            new = self.viewport.x + mult * self.hercules.x_vel
            highest = self.level_rect.w - self.viewport.w
            self.viewport.x = min(highest, new)

    # Κάνει update όταν ο Ηρακλής έχει ολοκληρώσει την πίστα
    def update_while_in_castle(self):
        for score in self.moving_score_list:
            score.update(self.moving_score_list, self.game_info)
        self.overhead_info_display.update(self.game_info)

        if self.overhead_info_display.state == c.END_OF_LEVEL:
            self.state = c.FLAG_AND_FIREWORKS
            self.flag_pole_group.add(castle_flag.Flag(8745, 322))

    def update_flag_and_fireworks(self):
        for score in self.moving_score_list:
            score.update(self.moving_score_list, self.game_info)
        self.overhead_info_display.update(self.game_info)
        self.flag_pole_group.update()

        self.end_game()

    # Τέλος της συγκεκριμένης πίστας
    def end_game(self):
        if self.flag_timer == 0:
            self.flag_timer = self.current_time
        elif (self.current_time - self.flag_timer) > 2000:
            self.set_game_info_values()
            self.next = c.GAME_END
            self.sound_manager.stop_music()
            self.done = True

    # Τοποθετεί όλατα τα αντικείμενα στην οθόνη
    def blit_everything(self, surface):
        self.level.blit(self.background, self.viewport, self.viewport)
        if self.flag_score:
            self.flag_score.draw(self.level)
        self.powerup_group.draw(self.level)
        self.brick_group.draw(self.level)
        self.firestick_group.draw(self.level)
        self.coin_group.draw(self.level)
        self.coin_box_group.draw(self.level)
        self.sprites_about_to_die_group.draw(self.level)
        self.shell_group.draw(self.level)
```

```
        # self.check_point_group.draw(self.level)
        self.brick_pieces_group.draw(self.level)
        self.flag_pole_group.draw(self.level)
        self.hercules_and_enemy_group.draw(self.level)

        surface.blit(self.level, (0, 0), self.viewport)
        self.overhead_info_display.draw(surface)
        for score in self.moving_score_list:
            score.draw(surface)
class LoadScreen(tools._State):
    def __init__(self):
        tools._State.__init__(self)

    def startup(self, current_time, persist):
        # Start time είναι ο χρόνος που ξεκινάει το συγκεκριμένο state
        self.start_time = current_time
        # Μεταφέρονται οι game_info
        self.persist = persist
        self.previous = self.persist[c.PREV_LEVEL]
        self.game_info = self.persist
        # καθορίζεται η επόμενη state
        self.next = self.set_next_state()
        # Καθορίζονται οι πληροφορίες στο overhead και δημιουργείται ένα
αντικείμενο αυτού του τύπου
        info_state = self.set_overhead_info_state()
        self.overhead_info = info.OverheadInfo(self.game_info, info_state)
        self.sound_manager = game_sound.Sound(self.overhead_info)



    # Καθορίζεται το επόμενο state
    def set_next_state(self):
        if self.previous == c.LABOUR_BULL:
            return c.LEVEL1
        elif self.previous == c.LABOUR_LION:
            return c.LEVEL2
        elif self.previous == c.LABOUR_HYDRA:
            return c.LEVEL3

    # Καθορίζεται το overhead state που θα εμφανιστεί στην οθόνη
    # Στη συγκεκριμένη περίπτωση στο πάνω μέρος θα εμφανίζονται
    # οι πληροφορίες του παιχνιδιού (χρόνος, coins κλπ)
    def set_overhead_info_state(self):
        return c.LOAD_SCREEN

    # Η μέθοδος καλείται από την update μέθοδο της tools.Control
    # Κάνει ανανέωση της Loading Screen
    def update(self, surface, keys, current_time):
        # print('current_time: {}, start_time: {}'.format(current_time,
self.start_time))
        if (current_time - self.start_time) < 400:
            surface.fill(c.BLACK)
            self.overhead_info.update(self.game_info)
            self.overhead_info.draw(surface)

        elif (current_time - self.start_time) < 600:
            surface.fill(c.BLACK)

        elif (current_time - self.start_time) < 635:
            surface.fill((106, 150, 252))
```

```
        # Μόλις περάσουν τα 2635ms τότε το state.done = True (έχει
ολοκληρωθεί)
        # και εκτελείται το επόμενο state (LEVEL) που έχει οριστεί παραπάνω
        else:
            self.done = True




# Η κλάση αυτή σετίζεται με την οθόνη GAME_OVER
class GameOver(LoadScreen):
    def __init__(self):
        super(GameOver, self).__init__()

    # Επικάλυψη της μεθόδου set_next_stage ωστε αυτή τη φορά να οδηγούμαστε
στο MAIN_MENU
    def set_next_state(self):
        return c.MAIN_MENU

    # Επικάλυψη της μεθόδου set_overhead_info σχετικά με τις overhead info
    def set_overhead_info_state(self):
        return c.GAME_OVER

    def update(self, surface, keys, current_time):
        self.current_time = current_time
        self.sound_manager.update(self.persist, None)

        if (self.current_time - self.start_time) < 1000:
            surface.fill(c.BLACK)
            self.overhead_info.update(self.game_info)
            self.overhead_info.draw(surface)
        elif (self.current_time - self.start_time) < 2000:
            surface.fill(c.BLACK)
        elif (self.current_time - self.start_time) < 2350:
            surface.fill((106, 150, 252))
        else:
            self.done = True

# Κλάση που αντιπροσωπεύει το TimeOut όταν δηλαδή έχει τελειώσει ο χρόνος
του χρήστη
class TimeOut(LoadScreen):
    def __init__(self):
        super(TimeOut, self).__init__()

    # Καθορισμός του επόμενου επιπέδου
    def set_next_state(self):
        if self.persist[c.LIVES] == 0:
            return c.GAME_OVER
        else:
            return c.LOAD_SCREEN


    # Καθορισμός του overhead
    def set_overhead_info_state(self):
        return c.TIME_OUT

    def update(self, surface, keys, current_time):
        self.current_time = current_time

        if (self.current_time - self.start_time) < 2400:
            surface.fill(c.BLACK)
```

```
            self.overhead_info.update(self.game_info)
            self.overhead_info.draw(surface)
        else:
            self.done = True
import pygame as pg
from .. import setup, tools
from .. import constants as c
from .. components import info, hercules


class Menu(tools._State):
    def __init__(self):
        # Αρχικοποίηση
        super().__init__()
        persist = {c.COIN_TOTAL: 0,  # Πλήθος από coins
                   c.SCORE: 0,  # score
                   c.LIVES: 3,  # Πλήθος από ζωές
                   c.TOP_SCORE: 0,
                   c.CURRENT_TIME: 0.0,  # Αρχικοποίση της ώρας
                   c.PREV_LEVEL: None,
                   c.LEVEL_STATE: None,
                   c.CAMERA_START_X: 0,
                   c.HERCULES_DEAD: False}
        self.startup(0.0, persist)

    # Εκτελείται κάθε φορά που το state παίρνει την τιμή main_menu
    def startup(self, current_time, persist):
        # Καθορίζουμε ότι η επόμενη state θα είναι η LOAD_SCREEN
        self.next = c.STORY_ONE
        self.persist = persist
        self.game_info = persist
        # Δημιουργία αντικειμένου OverheadInfo για την εμφάνιση στην οθόνη
των
        # στοιχείων (χρόνος, ζωές, coins, κλπ)
        self.overhead_info = info.OverheadInfo(self.game_info, c.MAIN_MENU)

        self.sprite_sheet = setup.GFX['title_screen']
        self.setup_background()
        self.setup_hercules()
        self.setup_cursor()

    # Δημιουργεί το mushroom cursor για τη επιλογή 1 ή 2 πακτών
    def setup_cursor(self):
        self.cursor = pg.sprite.Sprite()

        self.cursor.image, self.cursor.rect = self.get_image(
            24, 160, 8, 8, (220, 358), setup.GFX['item_objects'])
        self.cursor.state = c.PLAYER1

    # Τοποθετεί τον Ηρακλή στην οθόνη
    def setup_hercules(self):
        self.hercules = hercules.Hercules()
        self.hercules.rect.x = 110
        self.hercules.rect.bottom = c.GROUND_HEIGHT

    # Αρχικοποίηση της εικόνας του background
    def setup_background(self):
        # Παίρνουμε την εικόνα και το rect της εικόνας
        self.background = setup.GFX['level_1']
        self.background_rect = self.background.get_rect()
        # Μεγενθύνουμε την εικόνα
```

```
        self.background = pg.transform.scale(self.background,

(int(self.background_rect.width*c.BACKGROUND_MULTIPLER),

int(self.background_rect.height*c.BACKGROUND_MULTIPLER)))

        self.viewport =
setup.SCREEN.get_rect(bottom=setup.SCREEN_RECT.bottom)

        # Περιέχει τη εικόνα και το rect της εικόνας
        self.image_dict = {}
        self.image_dict['GAME_NAME_BOX'] = self.get_image(
            10, 73, 205, 50, (140, 100), setup.GFX['title_logo'])



    # Επιστρέφει την εικόνα και το rect της εικόνας από το αρχείο
title_logo (Τίτλος Παιχνιδιού)
    def get_image(self, x, y, width, height, dest, sprite_sheet):
        image = pg.Surface([width, height])
        rect = image.get_rect()

        image.blit(sprite_sheet, (0, 0), (x, y, width, height))
        if sprite_sheet == setup.GFX['title_logo']:
            # image.set_colorkey((255, 0, 220))
            image = pg.transform.scale(image,
                                       (int(rect.width*c.SIZE_MULTIPLIER),
                                        int(rect.height*c.SIZE_MULTIPLIER)))
        else:
            image.set_colorkey(c.BLACK)
            image = pg.transform.scale(image,
                                       (int(rect.width*3),
                                        int(rect.height*3)))

        rect = image.get_rect()
        rect.x = dest[0]
        rect.y = dest[1]
        return (image, rect)

    # Κάνει update την κατάσταση main_menu
    def update(self, surface, keys, current_time):
        self.current_time = current_time
        self.game_info[c.CURRENT_TIME] = self.current_time
        self.update_cursor(keys)
        self.overhead_info.update(self.game_info)

        #  Εμφάνιση αντικειμένων πάνω στην εικόνα
        surface.blit(self.background, self.viewport, self.viewport)
        surface.blit(self.image_dict['GAME_NAME_BOX'][0],
self.image_dict['GAME_NAME_BOX'][1])
        surface.blit(self.hercules.image, self.hercules.rect)
        surface.blit(self.cursor.image, self.cursor.rect)
        self.overhead_info.draw(surface)

    # Κάνει update την θέση του cursor
    def update_cursor(self, keys):
        input_list = [pg.K_RETURN, pg.K_a, pg.K_s]
        if self.cursor.state == c.PLAYER1:
            self.cursor.rect.y = 358
            if keys[pg.K_DOWN]:
                self.cursor.state = c.PLAYER2
```

```
            for input in input_list:
                if keys[input]:
                    self.reset_game_info()
                    self.done = True
        elif self.cursor.state == c.PLAYER2:
            self.cursor.rect.y = 403
            if keys[pg.K_UP]:
                self.cursor.state = c.PLAYER1

    # Αρχικοποιεί τα δεδομένα της δομής game_info σε περίπτωση Game Over
και restart
    def reset_game_info(self):
        self.game_info[c.COIN_TOTAL] = 0
        self.game_info[c.SCORE] = 0
        self.game_info[c.LIVES] = 3
        self.game_info[c.CURRENT_TIME] = 0.0
        self.game_info[c.LEVEL_STATE] = None

        self.persist = self.game_info
import pygame as pg
from data.states import load_screen
from data import setup, game_sound
from data import constants as c
from data import tools
from data.components import info


# Δημιουργία κλάσης Story για τα κείμενα που αναφέρονται στους άθλους του
Ηρακλή
class Story(tools._State):
    def __init__(self):
        super().__init__()

    def startup(self, current_time, persist):
        # Start time είναι ο χρόνος που ξεκινάει το συγκεκριμένο state
        self.start_time = current_time
        # Μεταφέρονται οι game_info
        self.persist = persist
        self.game_info = self.persist
        self.setup_background()
        # καθορίζεται η επόμενη state
        self.next = self.set_next_state()
        # Καθορίζονται οι πληροφορίες στο overhead και δημιουργείται ένα
αντικείμενο αυτού του τύπου
        info_state = self.set_overhead_info_state()
        self.overhead_info = info.OverheadInfo(self.game_info, info_state)
        # Ήχος
        self.sound_manager = game_sound.Sound(self.overhead_info)


    def set_overhead_info_state(self):
        return c.STORY_TELLING

    # Ορίζεται το επόμενο state του παιχνιδιού
    def set_next_state(self):
        return c.LABOUR_BULL

    # Ορίζεται το background
    def setup_background(self):
        # Παίρνουμε την πρώτη εικόνα του background και το rect της εικόνας
        self.sprite_sheet = setup.GFX['story_1']
```

```
        self.background_img = self.get_image(0, 0, 800, 600)
        self.background_rect = self.background_img.get_rect()
        self.background_rect.x = 0
        self.background_rect.y = 0

        # Παίρνουμε την δεύτερη εικόνα του background και το rect της
εικόνας
        self.sprite_sheet = setup.GFX['story_2']
        self.background_img2 = self.get_image(0, 0, 800, 600)
        self.background_rect2 = self.background_img2.get_rect()
        self.background_rect2.x = 0
        self.background_rect2.y = 0

        # Παίρνουμε την εικόνα logo και το rect της εικόνας
        self.sprite_sheet = setup.GFX['title_logo']
        self.image = self.get_image(10, 73, 205, 50)
        self.image_rect = self.image.get_rect()
        self.image =
pg.transform.scale(self.image,(int(self.image_rect.width * 1.7),
                                       int(self.image_rect.height *
1.7)))
        self.image_rect.x = 230
        self.image_rect.y = 10

    # Εξάγει την εικόνα από το επιλεγμένο αρχείο
    def get_image(self, x, y, width, height):
        image = pg.Surface([width, height]).convert()
        image.blit(self.sprite_sheet, (0, 0), (x, y, width, height))
        return image

    # Εμφανίζει τις εικόνες
    def update(self, surface, keys, current_time):
        self.current_time = current_time

        if (self.current_time - self.start_time) < 30000:
        #if (self.current_time - self.start_time) < 300:
            surface.blit(self.background_img, self.background_rect)
            surface.blit(self.image, self.image_rect)
        #elif (self.current_time - self.start_time) < 300:
        elif (self.current_time - self.start_time) < 60000:
            surface.blit(self.background_img2, self.background_rect)
            surface.blit(self.image, self.image_rect)
        else:
            self.done = True

class BullLabour(Story):
    def __init__(self):
        super().__init__()
        self.setup_background()

    def set_next_state(self):
        self.persist[c.PREV_LEVEL] = c.LABOUR_BULL
        return c.LOAD_SCREEN

    def setup_background(self):
        # Παίρνουμε την πρώτη εικόνα του background και το rect της εικόνας
        self.sprite_sheet = setup.GFX['labour_bull_1']
        self.background_img = self.get_image(0, 0, 800, 600)
        self.background_rect = self.background_img.get_rect()
        self.background_rect.x = 0
        self.background_rect.y = 0
```

```
        # Παίρνουμε την δεύτερη εικόνα του background και το rect της
εικόνας
        self.sprite_sheet = setup.GFX['labour_bull_2']
        self.background_img2 = self.get_image(0, 0, 800, 600)
        self.background_rect2 = self.background_img2.get_rect()
        self.background_rect2.x = 0
        self.background_rect2.y = 0

        # Παίρνουμε την εικόνα logo και το rect της εικόνας
        self.sprite_sheet = setup.GFX['title_logo']
        self.image = self.get_image(10, 73, 205, 50)
        self.image_rect = self.image.get_rect()
        self.image = pg.transform.scale(self.image,
(int(self.image_rect.width * 1.7),

int(self.image_rect.height * 1.7)))
        self.image_rect.x = 230
        self.image_rect.y = 10


class LionLabour(Story):
    def __init__(self):
        super().__init__()
        self.setup_background()

    def set_next_state(self):
        self.persist[c.PREV_LEVEL] = c.LABOUR_LION
        return c.LOAD_SCREEN

    def setup_background(self):
        # Παίρνουμε την πρώτη εικόνα του background και το rect της εικόνας
        self.sprite_sheet = setup.GFX['labour_lion_1']
        self.background_img = self.get_image(0, 0, 800, 600)
        self.background_rect = self.background_img.get_rect()
        self.background_rect.x = 0
        self.background_rect.y = 0

        # Παίρνουμε την δεύτερη εικόνα του background και το rect της
εικόνας
        self.sprite_sheet = setup.GFX['labour_lion_2']
        self.background_img2 = self.get_image(0, 0, 800, 600)
        self.background_rect2 = self.background_img2.get_rect()
        self.background_rect2.x = 0
        self.background_rect2.y = 0

        # Παίρνουμε την εικόνα logo και το rect της εικόνας
        self.sprite_sheet = setup.GFX['title_logo']
        self.image = self.get_image(10, 73, 205, 50)
        self.image_rect = self.image.get_rect()
        self.image = pg.transform.scale(self.image,
(int(self.image_rect.width * 1.7),

int(self.image_rect.height * 1.7)))
        self.image_rect.x = 230
        self.image_rect.y = 10

    # Εμφανίζει τις εικόνες
    def update(self, surface, keys, current_time):
        self.current_time = current_time
        if (self.current_time - self.start_time) < 30000:
```

```
        #if (self.current_time - self.start_time) < 300:
            surface.blit(self.background_img, self.background_rect)
            surface.blit(self.image, self.image_rect)
        #elif (self.current_time - self.start_time) < 300:
        elif (self.current_time - self.start_time) < 60000:
            surface.blit(self.background_img2, self.background_rect)
            surface.blit(self.image, self.image_rect)
        else:
            self.done = True


class HydraLabour(Story):
    def __init__(self):
        super().__init__()
        self.setup_background()

    def set_next_state(self):
        self.persist[c.PREV_LEVEL] = c.LABOUR_HYDRA
        return c.LOAD_SCREEN

    def setup_background(self):
        # Παίρνουμε την πρώτη εικόνα του background και το rect της εικόνας
        self.sprite_sheet = setup.GFX['labour_hydra_1']
        self.background_img = self.get_image(0, 0, 800, 600)
        self.background_rect = self.background_img.get_rect()
        self.background_rect.x = 0
        self.background_rect.y = 0

        # Παίρνουμε την δεύτερη εικόνα του background και το rect της
εικόνας
        self.sprite_sheet = setup.GFX['labour_hydra_2']
        self.background_img2 = self.get_image(0, 0, 800, 600)
        self.background_rect2 = self.background_img2.get_rect()
        self.background_rect2.x = 0
        self.background_rect2.y = 0

        # Παίρνουμε την εικόνα logo και το rect της εικόνας
        self.sprite_sheet = setup.GFX['title_logo']
        self.image = self.get_image(10, 73, 205, 50)
        self.image_rect = self.image.get_rect()
        self.image = pg.transform.scale(self.image,
(int(self.image_rect.width * 1.7),

int(self.image_rect.height * 1.7)))
        self.image_rect.x = 230
        self.image_rect.y = 10

     # Εμφανίζει τις εικόνες
    def update(self, surface, keys, current_time):
        self.current_time = current_time
        if (self.current_time - self.start_time) < 30000:
        #if (self.current_time - self.start_time) < 300:
            surface.blit(self.background_img, self.background_rect)
            surface.blit(self.image, self.image_rect)
        #elif (self.current_time - self.start_time) < 300:
        elif (self.current_time - self.start_time) < 60000:
            surface.blit(self.background_img2, self.background_rect)
            surface.blit(self.image, self.image_rect)
        else:
            self.done = True
```

```
class StoryEnd(Story):
    def __init__(self):
        super().__init__()
        self.setup_background()

    def set_next_state(self):
        return c.MAIN_MENU

    def setup_background(self):
        # Παίρνουμε την πρώτη εικόνα του background και το rect της εικόνας
        self.sprite_sheet = setup.GFX['end_of_story']
        self.background_img = self.get_image(0, 0, 800, 600)
        self.background_rect = self.background_img.get_rect()
        self.background_rect.x = 0
        self.background_rect.y = 0


        # Παίρνουμε την εικόνα logo και το rect της εικόνας
        self.sprite_sheet = setup.GFX['title_logo']
        self.image = self.get_image(10, 73, 205, 50)
        self.image_rect = self.image.get_rect()
        self.image = pg.transform.scale(self.image,
(int(self.image_rect.width * 1.7),

int(self.image_rect.height * 1.7)))
        self.image_rect.x = 230
        self.image_rect.y = 10

        # Εμφανίζει τις εικόνες

    def update(self, surface, keys, current_time):
        self.current_time = current_time
        if (self.current_time - self.start_time) < 10000:
            surface.blit(self.background_img, self.background_rect)
            surface.blit(self.image, self.image_rect)
        else:
            self.done = True
# Διαστάσεις οθόνης
SCREEN_HEIGHT = 600
SCREEN_WIDTH = 800
SCREEN_SIZE = (SCREEN_WIDTH, SCREEN_HEIGHT)

# Τίτλος Παιχνιδιού
ORIGINAL_CAPTION = "Super Hercules"

# Χρώματα σε μορφή RGB
GRAY = (100, 100, 100)
NAVYBLUE = (60, 60, 100)
WHITE = (255, 255, 255)
RED = (255, 0, 0)
GREEN = (0, 255, 0)
FOREST_GREEN = (31, 162, 35)
BLUE = (0, 0, 255)
SKY_BLUE = (39, 145, 251)
YELLOW = (255, 255, 0)
ORANGE = (255, 128, 0)
PURPLE = (255, 0, 255)
CYAN = (0, 255, 255)
BLACK = (0, 0, 0)
NEAR_BLACK = (19, 15, 48)
COMBLUE = (233, 232, 255)
```

```
GOLD = (255, 215, 0)
LIGHT_GREEN = (159, 159, 95)

BGCOLOR = WHITE

HERCULES_SIZE_MULTIPLIER = 0.25
BULL_SIZE_MULTIPLIER = 0.65
HYDRA_MULTIPLIER = 1.3
SIZE_MULTIPLIER = 2.5
BRICK_SIZE_MULTIPLIER = 2.69
BACKGROUND_MULTIPLER = 2.679
GROUND_HEIGHT = SCREEN_HEIGHT - 62

PREV_LEVEL = 'previous level'

# Μέγεθος Ηρακλή
SMALL_HERCULES = 1.2
BIG_HERCULES = 1.5

# Δυνάμεις Ηρακλή
WALK_ACCEL = .15
RUN_ACCEL = 20
SMALL_TURNAROUND = .35

GRAVITY = 1.01
JUMP_GRAVITY = .31
JUMP_VEL = -11
FAST_JUMP_VEL = -12.5
MAX_Y_VEL = 11

MAX_RUN_SPEED = 800
MAX_WALK_SPEED = 6

# Καταστάσεις στις οπίες μπορεί να βρεθεί ο Ηρακλής

STAND = 'standing'
WALK = 'walk'
JUMP = 'jump'
FALL = 'fall'
SMALL_TO_BIG = 'small to big'
BIG_TO_FIRE = 'big to fire'
BIG_TO_SMALL = 'big to small'
FLAGPOLE = 'flag pole'
WALKING_TO_CASTLE = 'walking to castle'
END_OF_LEVEL_FALL = 'end of level fall'

# Lion Stuff

LEFT = 'left'
RIGHT = 'right'
JUMPED_ON = 'jumped on'
DEATH_JUMP = 'death jump'

SHELL_SLIDE = 'shell slide'

# BRICK STATES

RESTING = 'resting'
BUMPED = 'bumped'

# COIN STATES
```

```
OPENED = 'opened'

# MILK STATES

REVEAL = 'reveal'
SLIDE = 'slide'

# COIN STATES

SPIN = 'spin'

# STAR STATES

BOUNCE = 'bounce'

# FIRE STATES

FLYING = 'flying'
BOUNCING = 'bouncing'
EXPLODING = 'exploding'

# Brick and coin box contents

MUSHROOM = 'mushroom'
STAR = 'star'
FIREFLOWER = 'fireflower'
SIXCOINS = '6coins'
COIN = 'coin'
LIFE_MUSHROOM = '1up_mushroom'

FIREBALL = 'fireball'

# LIST of ENEMIES
GOOMBA = 'goomba'
KOOPA = 'koopa'
FLY_COOPA = 'fly_coopa'
BULL = 'bull'
LION = 'lion'
HYDRA = 'hydra'
FIRESTICK = 'firestick'
FIRE = 'fire'

FLY = 'fly'

# LEVEL STATES

FROZEN = 'frozen'
NOT_FROZEN = 'not frozen'
IN_CASTLE = 'in castle'
FLAG_AND_FIREWORKS = 'flag and fireworks'

# FLAG STATE
TOP_OF_POLE = 'top of pole'
SLIDE_DOWN = 'slide down'
BOTTOM_OF_POLE = 'bottom of pole'

# 1UP score
ONEUP = '379'

# MAIN MENU CURSOR STATES
PLAYER1 = '1 player'
```

```
PLAYER2 = '2 player'

# OVERHEAD INFO STATES
MAIN_MENU = 'main menu'
LOAD_SCREEN = 'loading screen'
LEVEL = 'level'
TEST = 'test'
GAME_OVER = 'game over'
FAST_COUNT_DOWN = 'fast count down'
END_OF_LEVEL = 'end of level'

# GAME INFO DICTIONARY KEYS
COIN_TOTAL = 'coin total'
SCORE = 'score'
TOP_SCORE = 'top score'
LIVES = 'lives'
CURRENT_TIME = 'current time'
LEVEL_STATE = 'level state'
CAMERA_START_X = 'camera start x'
HERCULES_DEAD = 'hercules dead'

# STATES FOR ENTIRE GAME
MAIN_MENU = 'main menu'
STORY_ONE = 'story one'
GAME_END = 'game end'
LABOUR_BULL = 'labour bull'
LABOUR_LION = 'labour lion'
LABOUR_HYDRA = 'labour hydra'
LOAD_SCREEN = 'load screen'
TIME_OUT = 'time out'
GAME_OVER = 'game over'
LEVEL1 = 'level1'
LEVEL2 = 'level2'
LEVEL3 = 'level3'

# SOUND STATES
STORY_TELLING = 'story telling'
NORMAL = 'normal'
STAGE_CLEAR = 'stage clear'
WORLD_CLEAR = 'world clear'
TIME_WARNING = 'time warning'
SPED_UP_NORMAL = 'sped up normal'
HERCULES_INVINCIBLE = 'hercules invincible'
# Κλάση που διαχειρίζεται τους ήχους του παιχνιδιού
class Sound(object):
    def __init__(self, overhead_info):
        self.sfx_dict = setup.SFX
        self.music_dict = setup.MUSIC
        self.overhead_info = overhead_info
        self.game_info = overhead_info.game_info
        self.set_music_mixer()


    # Ορίζει τη μουσική για κάθε μέρος του παιχνιδού
    def set_music_mixer(self):
        if self.overhead_info.state == c.STORY_TELLING:
            pg.mixer.music.load(self.music_dict['story_music'])
            pg.mixer.music.play()
            self.state = c.STORY_TELLING
        elif self.overhead_info.world3:
            pg.mixer.music.load(self.music_dict['hydra_castle'])
```

```
            pg.mixer.music.play()
            self.state = c.NORMAL
        elif self.overhead_info.state == c.LEVEL:
            pg.mixer.music.load(self.music_dict['main_theme'])
            pg.mixer.music.play()
            self.state = c.NORMAL

        elif self.overhead_info.state == c.GAME_OVER:
            pg.mixer.music.load(self.music_dict['game_over'])
            pg.mixer.music.play()
            self.state = c.GAME_OVER


    def update(self, game_info, hercules):
        self.game_info = game_info
        self.hercules = hercules
        self.handle_state()

    # Παίζει τον ανάλογο ήχο ανάλογα με την κατάσταση που βρίσεκται ο
Ηρακλής
    def  handle_state(self):
        if self.state == c.NORMAL:
            if self.hercules.dead:
                self.play_music('death', c.HERCULES_DEAD)
            elif self.hercules.invincible \
                    and self.hercules.losing_invincibility == False:
                self.play_music('invincible', c.HERCULES_INVINCIBLE)
            elif self.hercules.state == c.FLAGPOLE:
                self.play_music('flagpole', c.FLAGPOLE)
            elif self.overhead_info.time == 100:
                self.play_music('out_of_time', c.TIME_WARNING)


        elif self.state == c.FLAGPOLE:
            if self.hercules.state == c.WALKING_TO_CASTLE:
                self.play_music('stage_clear', c.STAGE_CLEAR)

        elif self.state == c.STAGE_CLEAR:
            if self.hercules.in_castle:
                self.sfx_dict['count_down'].play()
                self.state = c.FAST_COUNT_DOWN

        elif self.state == c.FAST_COUNT_DOWN:
            if self.overhead_info.time == 0:
                self.sfx_dict['count_down'].stop()
                self.state = c.WORLD_CLEAR

        elif self.state == c. TIME_WARNING:
            if pg.mixer.music.get_busy() == 0:
                self.play_music('main_theme_sped_up', c.SPED_UP_NORMAL)
            elif self.hercules.dead:
                self.play_music('death', c.HERCULES_DEAD)

        elif self.state == c.SPED_UP_NORMAL:
            if self.hercules.dead:
                self.play_music('death', c.HERCULES_DEAD)
            elif self.hercules.state == c.FLAGPOLE:
                self.play_music('flagpole', c.FLAGPOLE)

        elif self.state == c.HERCULES_INVINCIBLE:
            if (self.hercules.current_time -
```

```
self.hercules.invincible_start_timer) > 11000:
                self.play_music('main_theme', c.NORMAL)
            elif self.hercules.dead:
                self.play_music('death', c.HERCULES_DEAD)


        elif self.state == c.WORLD_CLEAR:
            pass
        elif self.state == c.HERCULES_DEAD:
            pass
        elif self.state == c.GAME_OVER:
            pass
        elif self.state == c.STORY_TELLING:
            self.play_music('story_music', c.STORY_TELLING)

    # Παίζει τη μουσική
    def play_music(self, key, state):
        pg.mixer.music.load(self.music_dict[key])
        pg.mixer.music.play()
        self.state = state

    def stop_music(self):
        pg.mixer.music.stop()
def main():
    # Δημιουργία αντικ. Control
    run_it = tools.Control(setup.ORIGINAL_CAPTION)
    # Δημιουργία λεξικού που περιέχει τα states του παιχνιδιού
    state_dict = {c.MAIN_MENU: main_menu.Menu(),
                  c.STORY_ONE: story.Story(),
                  c.LABOUR_BULL: story.BullLabour(),
                  c.LABOUR_LION: story.LionLabour(),
                  c.LABOUR_HYDRA: story.HydraLabour(),
                  c.LOAD_SCREEN: load_screen.LoadScreen(),
                  c.TIME_OUT: load_screen.TimeOut(),
                  c.GAME_OVER: load_screen.GameOver(),
                  c.GAME_END: story.StoryEnd(),
                  c.LEVEL1: level1.Level1(),
                  c.LEVEL2: level2.Level2(),
                  c.LEVEL3: level3.Level3()}

    # Η Εκτλεση ξεκινάει με το main_menu
    run_it.setup_states(state_dict, c.MAIN_MENU)
    run_it.main()
ORIGINAL_CAPTION = c.ORIGINAL_CAPTION


os.environ['SDL_VIDEO_CENTERED'] = '1'
pg.init()
pg.event.set_allowed([pg.KEYDOWN, pg.KEYUP, pg.QUIT])
pg.display.set_caption(c.ORIGINAL_CAPTION)
SCREEN = pg.display.set_mode(c.SCREEN_SIZE)
SCREEN_RECT = SCREEN.get_rect()


FONTS = tools.load_all_fonts(os.path.join("resources","fonts"))
MUSIC = tools.load_all_music(os.path.join("resources","music"))
GFX   = tools.load_all_gfx(os.path.join("resources","graphics"))
SFX   = tools.load_all_sfx(os.path.join("resources","sound"))
keybinding = {
    'action':pg.K_s,
    'jump':pg.K_a,
```

```
    'left':pg.K_LEFT,
    'right':pg.K_RIGHT,
    'down':pg.K_DOWN
}

class Control(object):
    # Η κλαση Control περιέχει το game loop καθώς και τη μέθοδο event_loop
η οποία
    # περνάει events στην κλάση State

    def __init__(self, caption):
        self.screen = pg.display.get_surface()
        self.done = False
        self.clock = pg.time.Clock()
        self.caption = caption
        self.fps = 60
        self.show_fps = False
        self.current_time = 0.0
        self.keys = pg.key.get_pressed()
        self.state_dict = {}
        self.state_name = None
        self.state = None

    def setup_states(self, state_dict, start_state):
        self.state_dict = state_dict
        self.state_name = start_state
        self.state = self.state_dict[self.state_name]

    def update(self):
        # Ο χρόνος που πέρασε από την αρχή της εκτέλεσης
        self.current_time = pg.time.get_ticks()
        if self.state.quit:
            self.done = True
        # Όταν ολοκληρωθεί ένα state πηγαίνουμε στο άλλο
        elif self.state.done:
            self.flip_state()
        # Εκτελείται η update μέθοδος της κατάστασης state
        self.state.update(self.screen, self.keys, self.current_time)

    # Αλλαγή κατάστασης (state)
    def flip_state(self):
        print(self.state.next)
        previous, self.state_name = self.state_name, self.state.next
        persist = self.state.cleanup()
        self.state = self.state_dict[self.state_name]
        self.state.startup(self.current_time, persist)
        self.state.previous = previous

    # Loop το οποίο χειρίζεται τα events από το πληκρολόγια
    def event_loop(self):
        for event in pg.event.get():
            # Τερματισμός παιχνιδιού
            if event.type == pg.QUIT:
                self.done = True
            # Πάτημα πλήκτρων
            elif event.type == pg.KEYDOWN:
                self.keys = pg.key.get_pressed()
                self.toggle_show_fps(event.key)
            # Απελευθέρωση πλήκτρων
            elif event.type == pg.KEYUP:
                self.keys = pg.key.get_pressed()
```

```
            self.state.get_event(event)


    def toggle_show_fps(self, key):
        if key == pg.K_F5:
            self.show_fps = not self.show_fps
            if not self.show_fps:
                pg.display.set_caption(self.caption)

    # Main Loop για ολόκληρο το πρόγραμμα
    def main(self):
        while not self.done:
            self.event_loop()
            self.update()
            pg.display.update()
            self.clock.tick(self.fps)
            if self.show_fps:
                fps = self.clock.get_fps()
                with_fps = "{} - {:.2f} FPS".format(self.caption, fps)
                pg.display.set_caption(with_fps)


class _State(object):
    def __init__(self):
        self.start_time = 0.0
        self.current_time = 0.0
        self.done = False
        self.quit = False
        self.next = None
        self.previous = None
        self.persist = {}

    def get_event(self, event):
        pass

    def startup(self, current_time, persistant):
        self.persist = persistant
        self.start_time = current_time

    def cleanup(self):
        self.done = False
        return self.persist

    def update(self, surface, keys, current_time):
        pass



def load_all_gfx(directory, colorkey=(255,0,255), accept=('.png', 'jpg',
'bmp')):
    graphics = {}
    for pic in os.listdir(directory):
        name, ext = os.path.splitext(pic)
        if ext.lower() in accept:
            img = pg.image.load(os.path.join(directory, pic))
            if img.get_alpha():
                img = img.convert_alpha()
            else:
                img = img.convert()
                img.set_colorkey(colorkey)
            graphics[name]=img
```

```
    return graphics


def load_all_music(directory, accept=('.wav', '.mp3', '.ogg', '.mdi')):
    songs = {}
    for song in os.listdir(directory):
        name,ext = os.path.splitext(song)
        if ext.lower() in accept:
            songs[name] = os.path.join(directory, song)
    return songs


def load_all_fonts(directory, accept=('.ttf')):
    return load_all_music(directory, accept)


def load_all_sfx(directory, accept=('.wav','.mpe','.ogg','.mdi')):
    effects = {}
    for fx in os.listdir(directory):
        name, ext = os.path.splitext(fx)
        if ext.lower() in accept:
            effects[name] = pg.mixer.Sound(os.path.join(directory, fx))
    return effects
```