ΤΕΧΝΟΛΟΓΙΚΟ ΕΚΠΑΙΔΕΥΤΙΚΟ ΙΔΡΥΜΑ ΔΥΤΙΚΗΣ ΕΛΛΑΔΑΣ
ΣΧΟΛΗ ΔΙΟΙΚΗΣΗΣ ΚΑΙ ΟΙΚΟΝΟΜΙΑΣ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΜΜΕ

**ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ**

# ΔΗΜΙΟΥΡΓΙΑ ΔΙΑΔΡΑΣΤΙΚΟΥ ΠΑΙΧΝΙΔΙΟΥ ΜΕ ΧΡΗΣΗ ΤΗΣ ΣΧΕΔΙΑΣΤΙΚΗΣ ΠΛΑΤΦΟΡΜΑΣ UNITY

**Σπουδαστής :Οικονόμος Σωτήριος**

**ΕΠΟΠΤΕΥΩΝ ΚΑΘΗΓΗΤΗΣ: ΚΟΥΤΡΑΣ ΑΘΑΝΑΣΙΟΣ**

**Πύργος, 2018**

# ΠΙΣΤΟΠΟΙΗΣΗ

Πιστοποιείται ότι η πτυχιακή εργασία με θέμα:

# «ΔΗΜΙΟΥΡΓΙΑ ΔΙΑΔΡΑΣΤΙΚΟΥ ΠΑΙΧΝΙΔΙΟΥ ΜΕ ΧΡΗΣΗ ΤΗΣ ΣΧΕΔΙΑΣΤΙΚΗΣ ΠΛΑΤΦΟΡΜΑΣ UNITY»

του φοιτητή του τμήματος ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΜΜΕ

**ΣΩΤΗΡΙΟΥ ΟΙΚΟΝΟΜΟΥ**

παρουσιάστηκε δημόσια και εξετάσθηκε στο Τμήμα ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΜΜΕ στις

_____ / _____ / _____

Ο ΕΠΙΒΛΕΠΩΝ

ΚΟΥΤΡΑΣ ΑΘΑΝΑΣΙΟΣ

Ο ΠΡΟΕΔΡΟΣ ΤΟΥ ΤΜΗΜΑΤΟΣ

Δρ. ΚΟΥΓΙΑΣ ΙΩΑΝΝΗΣ

# ΥΠΕΥΘΥΝΗ ΔΗΛΩΣΗ ΠΕΡΙ ΜΗ ΛΟΓΟΚΛΟΠΗΣ

Βεβαιώνω ότι είμαι συγγραφέας αυτής της εργασίας και ότι κάθε βοήθεια την οποία είχα για την προετοιμασία της, είναι πλήρως αναγνωρισμένη και αναφέρεται στην εργασία. Επίσης, έχω αναφέρει τις όποιες πηγές από τις οποίες έκανα χρήση δεδομένων, ιδεών ή λέξεων, είτε αυτές αναφέρονται ακριβώς είτε παραφρασμένες. Ακόμα δηλώνω ότι αυτή η γραπτή εργασία προετοιμάστηκε από εμένα προσωπικά και αποκλειστικά και ειδικά για την συγκεκριμένη πτυχιακή εργασία και ότι θα αναλάβω πλήρως τις συνέπειες εάν η εργασία αυτή αποδειχθεί ότι δεν μου ανήκει.

| **ΟΝΟΜΑΤΕΠΩΝΥΜΟ ΣΠΟΥΔΑΣΤΗ** | **ΑΜ** | **ΥΠΟΓΡΑΦΗ** |
|---|---|---|
| ΟΙΚΟΝΟΜΟΣ ΣΩΤΗΡΙΟΣ | |  |

# Ευχαριστίες

Θέλω να ευχαριστήσω αρχικά τους καθηγητές μου και ιδιαίτερα τον κ. Αθανάσιο Κούτρα για την εξαιρετική συνεργασία που είχαμε στην εκπόνηση της πτυχιακής εργασίας.

Θα ήθελα επίσης να ευχαριστήσω την οικογένεια μου για την απεριόριστη στήριξη που μου πρόσφερε σε όλη τη διάρκεια της φοιτητικής μου ζωής.

# Πρόλογος

Το παρόν κείμενο αποτελεί την πτυχιακή εργασία του φοιτητή Σωτήριου Οικονόμου ως προαπαιτούμενο για την απόκτηση προπτυχιακού τίτλου στο τμήμα Πληροφορικής και ΜΜΕ του Τεχνολογικού Εκπαιδευτικού Ιδρύματος Δυτικής Ελλάδας.

Στόχος της συγκεκριμένης εργασίας είναι η μελέτη της δημιουργίας μιας διαδραστικής εφαρμογής για κινητά με λειτουργικό σύστημα android χρησιμοποιώντας το εργαλείο Unity. Σε αυτή την εργασία παρουσιάζεται το εργαλείο καθώς και η μεθοδολογία που ακολούθησα μαζί με το τελικό αποτέλεσμα.

Καλή ανάγνωση

# Περίληψη

Στην πτυχιακή εργασία έγινε μια θεωρητική προσέγγιση και ανάλυση του gaming και των παιχνιδιών γενικότερα μέσα από την δημιουργία μιας διαδραστικής εφαρμογής.

Το gaming, εκτός από μέσω διασκέδασης, αποτελεί και ένα εργαλείο εκμάθησης. Οι μορφές παιχνιδιού που συναντάμε σήμερα, τόσο σε φυσική όσο και σε ηλεκτρονική μορφή, είναι αμέτρητες. Με την ραγδαία αύξηση της τεχνολογίας, όλο και περισσότεροι άνθρωποι ανά τον κόσμο αποκτούν πρόσβαση σε έξυπνες ηλεκτρονικές συσκευές. Έτσι το mobile gaming παρουσιάζει και αυτό μια ραγδαία αύξηση.

Για την δημιουργία ενός διαδραστικού παιχνιδιού για κινητά υπάρχουν αρκετά εργαλεία όπως το Unity και η Unreal Engine. Φυσικά υπάρχει πάντα η δυνατότητα από τον προγραμματιστή να χρησιμοποιήσει απλά τις γνώσεις του σε μία γλώσσα προγραμματισμού όπως η Java ή η JavaScript για να προγραμματίσει ένα παιχνίδι.

Στα πλαίσια της εργασίας υλοποίησα μία εφαρμογή που ονομάζεται Infinite Runner και αποτελεί ένα παιχνίδι που σκοπός του χρήστη είναι να αντέξει όση περισσότερη ώρα μπορεί μαζεύοντας περισσότερους πόντους. Για την δημιουργία της χρησιμοποιήθηκε το εργαλείο Unity όπου μέσα από αυτό, χρησιμοποιώντας την γλώσσα προγραμματισμού C# και ακολουθώντας μια MVC αρχιτεκτονική χτίστηκε όλο το περιβάλλον.

Συμπερασματικά, ο χώρος του gaming και ειδικότερα η δημιουργία ενός παιχνιδιού αποτελούν ένα συναρπαστικό κομμάτι με τεράστιες προοπτικές ανάπτυξης και εφαρμογής.

## Abstract

The thesis has been a theoretical approach and analysis of gaming and games in general through the creation of an interactive application.

Gaming industry, besides entertainment, can also has learning capabilities. Games today are countless. With the rapid increase in technology, more and more people around the world are getting access to smart electronic devices. This results in a rapid increase in mobile gaming as well.

In order to create an interactive mobile game, there are several tools such as Unity and Unreal Engine. Of course, there is always the possibility for the developer to simply use his knowledge in a programming language like Java or JavaScript to create a game.

In the context of this thesis, I have created an application called Infinite Runner. It is a game that the user's purpose is to last as long as possible and collect more points. To create it, I used the Unity software, using the C # programming language and an MVC architecture.

In conclusion, the gaming area and, in particular, the creation of a game is exciting with enormous prospects of growth and application.

## Λέξεις Κλειδιά

Παιχνίδια, Android, Unity, Spreadsheets, Photoshop, Δημιουργία διαδραστικού παιχνιδιού, δημιουργία εφαρμογής, C#, Game Development

*Για τον πατέρα μου…*

# Περιεχόμενα

# Εικόνες

# Πίνακες

# Κεφάλαιο 1 - Θεωρία και Ιστορία του Gaming

Το gaming μπορεί να θεωρείται από πολλούς ως μια πρόσφατη «ανακάλυψη» που εμφανίστηκε με την ραγδαία ανάπτυξη της τεχνολογίας και απευθύνεται κυρίως σε άτομα που βρίσκονται σε νεαρή ηλικία. Στην πραγματικότητα όμως αποτελεί έναν επιστημονικό κλάδο με σπουδαίους θεωρητικούς. Όπως βέβαια θα δούμε παρακάτω το game theory δεν ταυτίζεται απόλυτα με την σημερινή έννοια του όρου gaming, όμως η υπάρχει αλληλένδετη σχέση μεταξύ τους. Αν και στην συγκεκριμένη εργασία θα ασχοληθώ με την δημιουργία διαδραστικού παιχνιδιού μέσω του προγράμματος Unity, όπως μαρτυρά και ο τίτλος, είναι απαραίτητο να γίνει μια αναφορά γενικότερα στην επιστήμη του gaming ώστε να γίνουν πιο κατανοητοί οι στόχοι της εν λόγω εργασίας.

## 1.1 Τι είναι η επιστήμη του Gaming

Μπορεί σήμερα το gaming να αποτελεί κυρίως έναν δημοφιλή τρόπο διασκέδασης, όμως στην πραγματικότητα το game theory έχει και διαφορετικές εφαρμογές τόσο στην επιστήμη της οικονομίας και των μαθηματικών όσο και στις κοινωνικές επιστήμες.

Το game theory αποτελεί με απλά λόγια την διαδικασία λήψης στρατηγικών αποφάσεων με σκοπό την επίλυση ενός προβλήματος ή ενός διλήμματος. Στην οικονομία χρησιμοποιείται κυρίως στην οργάνωση της βιομηχανίας και στον σχεδιασμό μηχανισμών. Εφαρμογές επίσης βρίσκει και σε κοινωνικές επιστήμες όπως η ανθρωπολογία, η κοινωνιολογία, η ψυχολογία κ.λπ. όπου υφίσταται κυρίως με την μορφή κοινωνικών διλημμάτων με σκοπό την ανάλυση της συμπεριφοράς του ατόμου σε μια κοινωνική κατάσταση (χαρακτηριστικό παράδειγμα τέτοιων παιχνιδιών είναι το «δίλλημα του φυλακισμένου[1]» όπου μπορεί να βρει εφαρμογή σε πολλές επιστήμες).

## 1.2 Ιστορία του Gaming

Το παιχνίδι υφίσταται, υπό διάφορες μορφές, εκατοντάδες χρόνια τώρα. Χαρακτηριστικό παράδειγμα είναι τα ζάρια αλλά και το ζατρίκιο, αντίστοιχο του σημερινού παιχνιδιού σκάκι, που έπαιζαν οι αρχαίοι Έλληνες. Από την αρχαιότητα, λοιπόν, μέχρι σήμερα το παιχνίδι το συναντάμε σε πολλές μορφές. Στην συγκεκριμένη εργασία θα αφιερώσω περισσότερο χρόνο στο ηλεκτρονικό παιχνίδι (Video Game) το οποίο είναι ουσιαστικά και το αντικείμενο μας.

[1] Βλ. Παράρτημα Α

Η πρώτη εμφάνιση των video games γίνεται το 1947 όταν ο Thomas T. Goldsmith Jr και ο Estle Ray Mann δημιούργησαν το πρώτο παιχνίδι βασισμένο σε μια συσκευή καθοδικού σωλήνα και ήταν εμπνευσμένο από τους πυραύλους του δευτέρου παγκοσμίου πολέμου.

Αφού έγινε λοιπόν η αρχή, θα πρέπει να μεταφερθούμε 19 χρόνια μετά για να συναντήσουμε την επόμενη επιτυχή απόπειρα δημιουργίας ενός video game. Το 1966, λοιπόν, η μελέτη του Ralph Baer για νέους τρόπους απέδωσε καρπούς και έτσι δημιουργήθηκε η πρώτη συσκευή που μπορούσε να συνδεθεί στην τηλεόραση και να παίξει παιχνίδια. Το όνομα της ήταν Odyssey.

Από εκεί και πέρα η πρόοδος των video games ήταν ραγδαία. Το 1971 κάνει την εμφάνιση της στο πανεπιστήμιο του Στάνφορντ η πρώτη παιχνιδομηχανή με κέρματα, το Galaxy Game ενώ λίγο αργότερα βγήκε το Computer Space, το πρώτο video game που βγήκε σε μαζική παραγωγή. Οι δημιουργοί του δεύτερου ήταν και εκείνη που το 1972 δημιούργησαν την Atari.

Το 1976 και ενώ τα arcade παιχνίδια πολλαπλασιάζονταν με ταχύ ρυθμό έχουμε την πρώτη οικιακή συσκευή από την Coleco, την Telstar που έγινε μαζική παραγωγή. Το παιχνίδι που έπαιζε λεγόταν Pong (σε διάφορες παραλλαγές). Το 1978 ξεκινάει η χρυσή εποχή των arcade games με το παγκοσμίως γνωστό Space Invaders να κάνει την εμφάνιση του, ενώ την επόμενη χρονιά η Atari θα δημιουργήσει το Asteroids.

Η δεκαετία του 1980 θα ξεκινήσει με ένα ορόσημο για την βιομηχανία ηλεκτρονικών παιχνιδιών αφού θα κάνει την εμφάνιση του το δημοφιλέστερο παιχνίδι όλων των εποχών, το Pac Man. Την επόμενη χρονιά εμφανίστηκε το Donkey Kong με την βιομηχανία να καταγράφει ένα τζίρο 4.9 δις δολάρια. Την ίδια χρονιά άρχισαν να πολλαπλασιάζονται οι οικιακοί υπολογιστές κάτι που είχε ως αποτέλεσμα την ραγδαία αύξηση των video games. Επίσης το 1980 είχαμε την πρώτη φορητή κονσόλα από την Nintendo, το Game & Watch.

Από το 1983 έως το 1995 έχουμε την Τρίτη γενιά κονσολών. Χαρακτηριστικοί εκπρόσωποι είναι το Commodore 64, NES, Sega Max 3 ενώ έκαναν την εμφάνισή τους παιχνίδια όπως το Zelda, Dragon Quest, Final Fantasy αλλά και το Sweet Home, το πρώτο Horror Game από την Capcom.

Στην συνέχεια, αρχίζουν την εμφάνιση τους κονσόλες όπως το Mega Drive και το SNES ως πρώτοι εκπρόσωποι της νέας γενιάς. Παιχνίδια όπως το Sonic και το Mario, που ακόμα και σήμερα έχουν φανατικό κοινό έκαναν την εμφάνισή τους.

Η πραγματική επανάσταση στο gaming, όμως έγινε τα επόμενα χρόνια. Τα πρώτα παιχνίδια με 3D γραφικά κάνουν την εμφάνιση τους. Ο ερχομός των κονσολών PlayStation αλλάζει για πάντα τα video games αφού αποτελεί την πιο επιτυχημένη σειρά κονσολών μέχρι και σήμερα. Ταυτόχρονα και ενώ οι κονσόλες συνέχισαν να ανθίζουν το PC gaming είχε επίσης ραγδαία αύξηση με παιχνίδια όπως Grand Theft Auto, Red Alert, Doom να γίνονται ανάρπαστα.

Μπαίνοντας πια στην νέα χιλιετία η αύξηση είναι τεράστια τόσο στην παραγωγή κονσολών όσο και παιχνιδιών. Ιδιαίτερη αναφορά αξίζουν κονσόλες όπως το PlayStation 2, PSP, 3 και 4, τα Xbox, 360 και One καθώς και τα Nintendo Wii, Switch αλλά και η σειρά GameBoy.

Κλείνοντας, ιδιαίτερη αναφορά πρέπει να γίνει στα mobile games που έκαναν την εμφάνιση τους από το 1997. Αρχικά με το γνωστό φιδάκι και το tetris αλλά και με πολλά ακόμα παιχνίδια να κάνουν την εμφάνιση τους το mobile gaming άρχισε να γίνεται θεσμός και απογειώθηκε με την εμφάνιση των πρώτων έξυπνων κινητών. Σήμερα αποτελούν μία άκρως επικερδή αγορά με τζίρο δισεκατομμυρίων δολαρίων κάθε χρόνο.

# Κεφάλαιο 2 – Εισαγωγή στο Unity

## 2.1 Εισαγωγή

Το Unity είναι ένα cross-platform πρόγραμμα παραγωγής 2D και 3D παιχνιδιών. Αν και αρχικά ανακοινώθηκε μόνο για το OS X σήμερα είναι διαθέσιμο για πάνω από 15 λειτουργικά συστήματα και πλατφόρμες και αποτελεί το βασικό Κιτ Ανάπτυξης Λογισμικού (SDK) της παιχνιδομηχανής Wii U. Από το 2006 έως σήμερα 6 εκδόσεις του προγράμματος έχουν διατεθεί, ενώ πολύ γνωστοί τίτλοι παιχνιδιών (Angry Birds, Slender κ.α.) έχουν δημιουργηθεί με την βοήθεια του Unity.

## 2.2 Βασικές λειτουργίες του Unity

Το Unity διαθετει πολλα χαρακτηριστικά κάτι που το φέρνει στις πρώτες θέσεις μεταξύ των προγραμμάτων της κατηγορίας του. Συγκεκριμένα το Unity συνδέεται με τις διεπαφές προγραμματισμού εφαρμογών (API) Direct3D για Windows και XBOX360, OpenGL για Mac και Windows, OpenGL ES για Android και iOS καθώς και με τα APIs των εκάστωτε παιχνιδομηχανών. Μεγάλη καινοτομία είναι η παραμετροποίηση της συμπίεσης υφής και της ανάλυσης για κάθε πλατφορμα καθώς και η υποστήριξη για Bump, Reflection και Parallax Mapping καθως και για δυναμική σκίαση.

Οι προγραμματιστές μπορούν να δημιουργήσουν τις εφαρμογές τους χρησιμοποιώντας Javascript, C# καθώς και Boo. Ακόμα έχουν την δυνατότητα μέσα απο το ίδιο project να αναπτύξουν πολλές εκδόσεις του ίδιου παιχνιδιού για διαφορετικές πλατφορμες.


**Εικόνα 2.1 Το λογότυπο του Unity**

## 2.3 Γνωριμία με το περιβάλλον διεπαφής

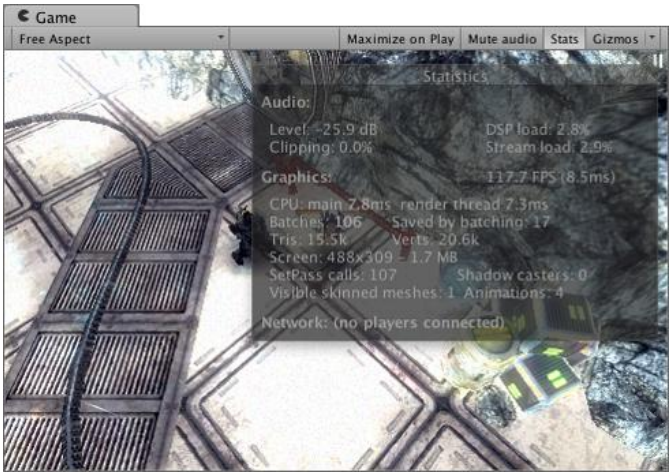
Το περιβάλλον διεπαφής του Unity είναι σχεδιασμένο για να εξυπηρετεί τόσο τον απλό όσο και τον προχωρημένο χρήστη. Η αρχική οθόνη (μετά την οθόνη δημιουργίας project) αποτελείται από την γραμμή μενού, τα εργαλεία μετακίνησης, περιστροφής και εστίασης των αντικειμένων καθώς και τα εργαλεία εκκίνησης, debugging της εφαρμογής και επιλογής layout και τέλος την κύρια οθόνη.

Η κύρια οθόνη είναι οργανωμένη με καρτέλες (tabs) προσφέροντας πιο εύκολη περιήγηση στους χρήστες. Οι κύριες καρτέλες είναι η Σκηνή (Scene), το Παιχνίδι (Game), η Ιεραρχία (Hierarchy), το Project και τέλος το Inspector.

Η Σκηνή χρησιμοποιείται για να τοποθετηθούν τα αντικείμενα που χρειάζονται, όπως για παράδειγμα οι κάμερες, ο παίχτης, οι αντίπαλοι κ.ο.κ. Οι περιήγηση στην σκηνή γίνεται είτε με συντομεύσεις του πληκτρολογίου, είτε με τα εργαλεία που αναφέρθηκαν στην αρχή της ενότητας.

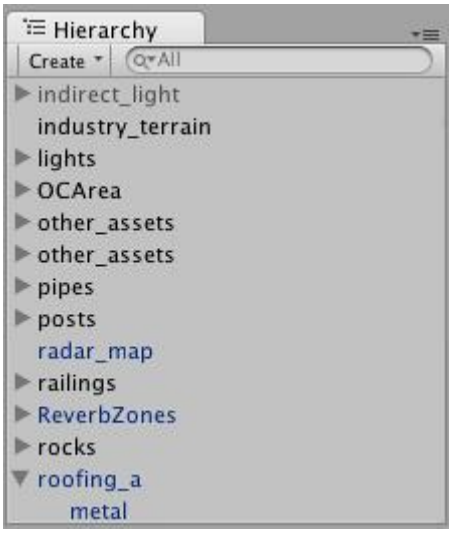

**Εικόνα 2.2 - Η καρτέλα Σκηνή**

Η καρτέλα Παιχνίδι είναι η τελική απεικόνιση του παιχνιδιού η οποία προκύπτει από το render που κάνουν οι κάμερες στην σκηνή. Η χρήση κάμερας είναι απαραίτητη για να μπορέσει να υπάρξει απεικόνιση του παιχνιδιού. Στην καρτέλα Game δίνεται η επιλογή για αλλαγή του aspect ratio, μεγιστοποίηση κατά την εκτέλεση, σίγαση, στατιστικά και η εμφάνιση ή όχι των αξόνων (Gismos).




**Εικόνα 2.3 - Η καρτέλα Game**

Η καρτέλα Hierarchy περιλαμβάνει όλα τα αντικείμενα που βρίσκονται στην σκηνή. Αυτά μπορεί να είναι κάμερες, αντικείμενα, textures κ.ο.κ.

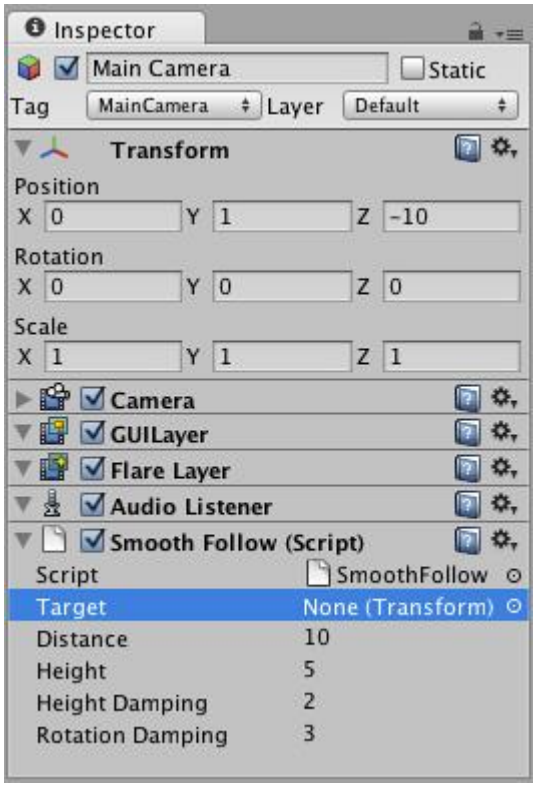



**Εικόνα 2.4 – Καρτέλα Hierarchy**

Η καρτέλα Project περιέχει όλα τα αρχεία και τους φακέλους που έχουν δημιουργηθεί για το συγκεκριμένο project. Χωρίζεται σε δύο οθόνες: η δεξιά περιέχει τα αρχεία ή και τους φακέλους ενώ η αριστερή περιέχει την δομή των φακέλων του project.


**Εικόνα 2.5 - Καρτέλα Project**

Η καρτέλα Inspector, τέλος, περιέχει τις ιδιότητες και τα επιμέρους στοιχεία (ήχους, φώτα κλπ.) των αντικειμένων που συνθέτουν την σκηνή. Κάθε ιδιότητα μπορεί να αλλάξει σε αυτή την καρτέλα ακόμη και τιμές μεταβλητών στα script (αλλαγές στις γραμμές του script ωστόσο χρειάζεται να γίνουν με επεξεργασία του script σε κάποιον editor – το unity προσφέρει το MonoDevelop).




**Εικόνα 2.6 – Η καρτέλα Inspector**

# Κεφάλαιο 3 - Δημιουργία Διαδραστικού Παιχνιδιού

## 3.1 Εισαγωγή

Το Unity προσφέρει όπως είδαμε αρκετά εργαλεία για την δημιουργία ενός ικανοποιητικού διαδραστικού παιχνιδιού. Για την συγκεκριμένη εργασία επέλεξα να δημιουργήσω ένα παιχνίδι κατηγορίας infinite runner για το λειτουργικό σύστημα Android (αν και όπως ανέφερα παραπάνω μπορεί να γίνει compile και σε άλλες πλατφόρμες).Το project δημιουργήθηκε με την γλώσσα προγραμματισμού C# και περιέχει 4 σκηνές τις οποίες θα τις αναλύσουμε επιμέρους παρακάτω. Αυτές είναι: η Start Scene, η Main Scene, η GameOver Scene και η Info Scene.

## 3.2 Start Scene

Η Start Scene (το όνομα αρχείου στο project είναι startScreenScene.unity) είναι η αρχική οθόνη που εμφανίζεται με την εκκίνηση της εφαρμογής. Περιέχει το κουμπί εκκίνησης της εφαρμογής, το λογότυπο, το κουμπί μετάβασης στην οθόνη πληροφοριών (info scene), το background και το κουμπί εξόδου.

Πριν ξεκινήσω να αναλύω τον κώδικα είναι πολύ βασικό να εξηγήσω κάποια κομμάτια του που είναι απαραίτητα για την λειτουργία όλων των script του project.

```
1.  using UnityEngine;
2.  using System.Collections;
```

**Πίνακας 3.1 – Βασικά namespaces**

Η λειτουργία των παραπάνω namespaces είναι απλή αλλά εξαιρετικά σημαντική για την λειτουργία του εκάστοτε κώδικα. Η πρώτη γραμμή είναι υπεύθυνη για την κλήση όλων των συναρτήσεων και των κλάσεων του Unity. Παρόμοια είναι και η χρησιμότητα της δεύτερης εντολής όπου με την σειρά της καλεί όλες τις συναρτήσεις της C#. Έτσι λοιπόν γίνεται κατανοητό ότι χωρίς αυτές τις εντολές δεν μπορεί να λειτουργήσει σωστά το project.

```
1.  public class ScriptName: MonoBehaviour;
```

**Πίνακας 3.2 – Monobehaviour**

Μία πολλή βασική κλάση είναι το *Monobehaviour*. Είναι απαραίτητο σε όλα τα script ώστε να μπορούν να λειτουργήσουν τα αντικείμενα στην σκηνή. Στην JavaScript κάθε script προέρχεται από το *Monobehaviour* αλλά στην C# είναι απαραίτητη η δήλωση του στο όνομα του script.

### 3.2.1 Κουμπιά εκκίνησης, εξόδου, info και λογότυπο

Αφού λοιπόν όρισα κάποια βασικά αλλά απαραίτητα κομμάτια κώδικα μπορώ να προχωρήσω στο κυρίως κομμάτι. Αν και το Unity προσφέρει την δυνατότητα δημιουργίας buttons από το interface, προτίμησα να τα δημιουργήσω μέσω του

κώδικα. Έτσι είχα την ελευθερία να δώσω στο εκάστοτε κουμπί την εντολή που ήταν απαραίτητη καθώς και να παραμετροποιήσω καλύτερα το μέγεθος.

Πριν φτάσω στο σημείο να δημιουργήσω τα κουμπιά θα πρέπει να ορίσω κάποιες άλλες παραμέτρους όπως τα skin για κάθε στοιχείο, τις γραμματοσειρές και το κουμπί εξόδου στην συσκευή (όχι το κουμπί που θα δημιουργηθεί στην οθόνη αλλά το φυσικό κουμπί επιστροφής που έχουν όλες οι συσκευές με λειτουργικό android).

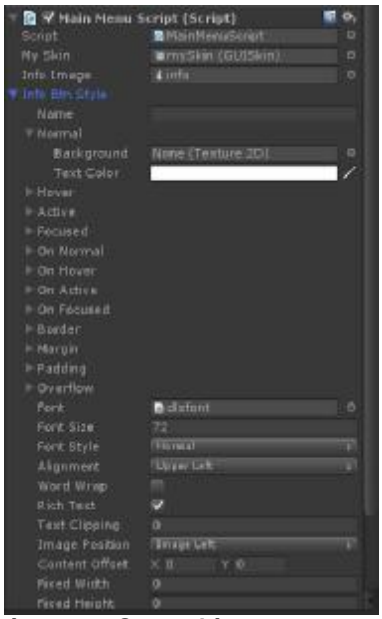
Το skin χρησιμεύει στο να δοθούν επιμέρους ιδιότητες σε κείμενα, κουμπιά κλπ. Αν και η δήλωση του skin γίνεται στο script, οι ιδιότητες συνήθως καθορίζονται στον inspector που είδαμε παραπάνω, χωρίς αυτό να σημαίνει ότι δεν μπορούν να γίνουν από τον κώδικα.

```
1.  public GUISkin mySkin;
2.  public Texture2D infoImage;
3.  public GUIStyle infoBtnStyle;
```

**Πίνακας 3.3 – Skin και Textures**

Εκτός από την δήλωση του skin στην πρώτη εντολή υπάρχει ακόμα η μεταβλητή *Texture2D* και η μεταβλητή *GUIStyle*. Η πρώτη είναι απαραίτητη για την εισαγωγή κάποιου γραφικού στοιχείου σε ένα κουμπί (και όχι μόνο) ενώ η δεύτερη βοηθά στο να δώσουμε ένα διαφορετικό style σε ένα κείμενο, κουμπί κλπ. από το γενικό που ορίζουμε μέσω του skin που είδαμε παραπάνω.

Αν λοιπόν κοιτάξουμε στον inspector θα παρατηρήσουμε τα παρακάτω.




**Εικόνα 3.1 – Οι εντολές στον Inspector**

Φαίνεται λοιπόν πως οι εντολές δημιούργησαν δύο πεδία και ένα Dropdown μενού. Στο πεδίο *mySkin* μπορούμε να εισάγουμε το skin μας που μπορούμε να δημιουργήσουμε και να παραμετροποιήσουμε από την καρτέλα Project και τον Inspector (με δεξί κλικ – διαφορετικά από το μενού του Unity). Το δεύτερο πεδίο είναι το Info Image όπου μπορώ να εισάγω την εικόνα που θέλω να εμφανίζεται στο

κουμπί και τέλος έχουμε το *Info Btn Style* όπου μπορώ να αλλάξω το style για το αντικείμενο που θα καθορίσω στην συνέχεια.

Αφού καθόρισα στο script κάποιες βασικές μεταβλητές, μπορώ να προχωρήσω στην πρώτη συνάρτηση. Η *Update* είναι μια πολύ συνηθισμένη συνάρτηση στο Unity καθώς δίνει την δυνατότητα να καλείτε το περιεχόμενο της σε κάθε frame.

```
1.  void Update() {
2.      if (Input.GetKeyDown(KeyCode.Escape)) {
3.          Application.Quit();
4.      }
5.      mySkin.label.fontSize = Screen.width / 10;
6.      mySkin.button.fontSize = Screen.width / 20;
7.      infoBtnStyle.fontSize = Screen.width / 15;
8.  }
```

**Πίνακας 3.4 – Η συνάρτηση Update**

Στην συγκεκριμένη συνάρτηση βλέπουμε αρχικά την πρώτη εντολή στις γραμμές 2, 3, 4 όπου έχουμε μια if για να δηλώσω ότι αν πατηθεί το κουμπί επιστροφής της συσκευής τότε η εφαρμογή να κλείσει. Στις υπόλοιπες γραμμές του κώδικα έχουν οριστεί τα μεγέθη των γραμματοσειρών για τα κουμπιά και τα κείμενα στην συγκεκριμένη σκηνή. Αυτή η δυνατότητα φυσικά δίνεται και από τα skin και τα styles που είδαμε παραπάνω. Όμως στον inspector τα μεγέθη είναι καθορισμένα σε pixels (px) κάτι κάνει την εφαρμογή να μην είναι responsive, κάτι εξαιρετικά βασικό αφού οι συσκευές δεν έχουν τις ίδιες οθόνες. Έτσι λοιπόν βλέπουμε για παράδειγμα στην γραμμή 5, ότι το font size για τα label της σκηνής που έχουν το skin mySkin θα είναι το πλάτος της οθόνης δια δέκα. Έτσι το μέγεθος της γραμματοσειράς καθορίζεται δυναμικά ανάλογα την οθόνη.

Κλείνοντας την συνάρτηση Update περνάω στο τελευταίο αλλά και πιο σημαντικό κομμάτι του script, την συνάρτηση OnGUI. Η συγκεκριμένη συνάρτηση επιτρέπει να δημιουργήσω ότι θέλω να δείξω στην οθόνη (το όνομα σημαίνει On Graphics User Interface). Στην συγκεκριμένη σκηνή θέλω να δημιουργήσω το logo, τα κουμπιά Start και Exit καθώς και το κουμπί Info.

```
1.  void OnGUI() {
2.      GUI.skin = mySkin;
3.      GUI.Label(new Rect(0, 0, Screen.width,

    Screen.height), "Crazy \nRunner");
4.      if (GUI.Button(new Rect((Screen.width -

    Screen.width * 0.5 f) - (Screen.width / 3) / 2, (Screen.height -

    Screen.height / 3) - (Screen.height / 7), Screen.width / 3,

    Screen.height / 7), "Start")) {
5.          Application.LoadLevel(1);
6.      }
7.      if (GUI.Button(new Rect((Screen.width -

    Screen.width * 0.5 f) - (Screen.width / 3) / 2, (Screen.height -

    Screen.height / 3) - (Screen.height / 7) * 0.02 f, Screen.width / 3,

    Screen.height / 7), "Exit")) {
8.          Application.Quit();
9.      }
10.     if (GUI.Button(new Rect(20, 20,

    Screen.width / 3, Screen.height / 7), "i", infoBtnStyle)) {
11.         Application.LoadLevel(4);
12.     }
13. }
```

**Πίνακας 3.5 – Η συνάρτηση OnGUI**

Ξεκινώντας την συνάρτηση δηλώνω το skin που θα χρησιμοποιήσω. Στην συνέχεια όπως φαίνεται στην τέταρτη σειρά ξεκινάω να φτιάξω το logo το οποίο δεν αποτελείται από κάποιο texture αλλά από κείμενο. Για να γράψουμε ένα κείμενο χρησιμοποιώ την εντολή GUI.Label. Για να δημιουργηθεί το κείμενο με την συγκεκριμένη εντολή χρειάζεται η δημιουργία ενός ορθογωνίου. Έτσι η σύνταξη της εντολής είναι οι συντεταγμένες της πάνω αριστερής γωνίας του ορθογωνίου καθώς και το πλάτος και ύψος του. Τέλος μετά την δημιουργία του ορθογωνίου θα πρέπει να καθορίσω και το κείμενο.

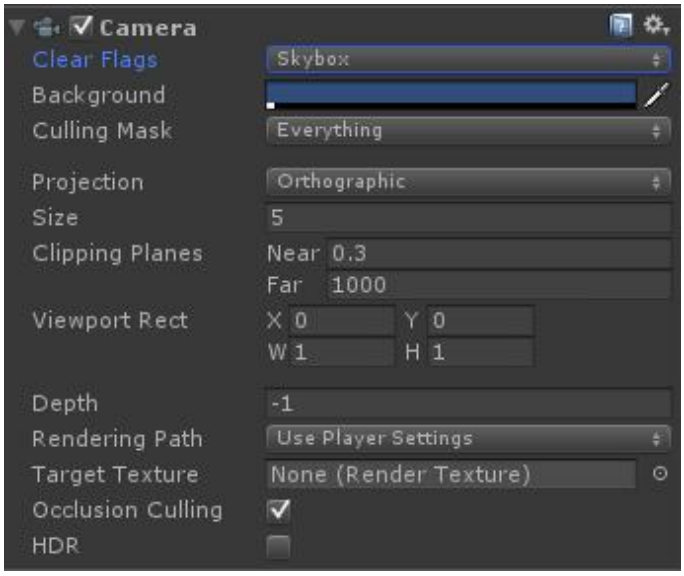
Με την ίδια λογική δημιουργούνται και τα κουμπιά που έχω στην σκηνη με την διαφορά ότι εδω χρειάζεται η εντολή GUI.Button. Φυσικά μόνη της η εντολή αυτή θα δημιουργήσει απλά ένα κουμπί χωρις κάποια χρήση. Για να ορίσω κάποια ενέργεια στο κουμπί θα πρέπει να τοποθετήσω την εντολή μέσα σε μια if όπως φαίνεται και στον πίνακα 3.5. Έτσι έχω τρία διαφορετικά κουμπιά με τρεις διαφορετικές εντολές. Το πρώτο κουμπί είναι το κουμπι Start με την εντολή `Application.LoadLevel(1)` η οποία στέλνει τον χρήστη στην σκηνή 1 (η σειρά των σκηνών καθορίζεται στα build settings όπως θα δούμε παρακάτω). Το δεύτερο κουμπί είναι το Exit με την εντολή `Application.Quit()` με την οποία κλείνει η εφαρμογή και τέλος έχω το τρίτο κουμπί για την σελίδα info με εντολή `Application.LoadLevel(4).` Εδώ αν και είχα σκοπό να χρησιμοποιήσω texture όπως είδαμε παραπάνω τελικά αποφάσισα να χρησιμοποιήσω απλά label όπως και στα άλλα κουμπιά ώστε να υπάρχει μεγαλύτερη ομοιομορφία.

Τελειώνοντας το script έχω δημιουργήσει το βασικό μενού. Φυσικά το script δεν μπορεί να λειτουργήσει απλά με μια αποθήκευση. Χρειάζεται να μπει σε κάποιο αντικείμενο στην σκηνή μας. Ήρθε η ώρα λοιπόν να δημιουργήσω την κάμερα και το background.

### 3.2.2 Κάμερα και Background

Όλα τα παραπάνω δεν θα είχαν κανένα νόημα (καθώς και όσα θα ακολουθήσουν στην συνέχεια) αν στην σκηνή δεν υπήρχε μια κάμερα. Αποτελεί ουσιαστικά το «μάτι» μας στην σκηνή, επιτρέποντας μας να βλέπουμε όλα όσα διαδραματίζονται. Μαζί με την κάμερα απαραίτητο είναι και το Background, όχι για λειτουργικούς λόγους, αφού η εφαρμογή θα μπορεί να εκτελεστεί και χωρίς αυτό, αλλά για αισθητικούς.

Το πρώτο πράγμα που ορίζω σε μία νέα σκηνή είναι η κάμερα. Η δημιουργία της κάμερας γίνεται μέσω του μενού (GameObject > Create Other > Camera). Στην συνέχεια μπορώ να παραμετροποιήσω τον τρόπο προβολής, το μέγεθος κ.α από τον Inspector.




**Εικόνα 3.2 – Η κάμερα στον Inspector**

Στη παραπάνω εικόνα φαίνονται οι επιλογές[2] που μπορώ να παραμετροποιήσω στην κάμερα της σκηνής μας. Στην δική μου περίπτωση χρησιμοποιώ μόνο μία κάμερα (αφού έχω 2D εφαρμογή) οπότε με ενδιαφέρει κυρίως ο τύπος προβολής (Projection) και το μέγεθος (Size). Στο πρώτο επιλέγω ορθογραφική προβολή και στο μέγεθος της προβολής επιλέγω το 5 μετά από σχετικές δοκιμές. Αν η εφαρμογή ήταν 3D τότε στην προβολή θα έπρεπε να χρησιμοποιήσω την Perspective. Θα πρέπει να αναφέρω επίσης την επιλογή Culling Mask που δίνει την δυνατότητα να επιλέξω τα layers που θα γίνουν render από την κάμερα.

Εκτός από την κάμερα, όπως ανέφερα και παραπάνω, θα χρησιμοποιήσω ένα background. Για να μπορέσω να ορίσω μια εικόνα σαν φόντο θα πρέπει πρώτα να την τοποθετήσω σε ένα αντικείμενο. Συγκεκριμένα θα δημιουργήσω ένα Quad (GameObject > Create Other > Quad) και πάνω σε αυτό θα σύρω την εικόνα που θέλω να χρησιμοποιήσω για background. Φυσικά θα πρέπει να προσαρμόσω το μέγεθος και την θέση από τον Inspector για να έχω το μέγεθος που θέλω. Μπορώ επίσης μέσω του shader να αλλάξω τον τρόπο που γίνεται το Render του φόντου δίνοντας μια διαφορετική αίσθηση χρώματος, φωτισμού κλπ., κάτι όμως που στην προκειμένη περίπτωση δεν με ενδιαφέρει.

Πολύ σημαντικό είναι να ορίσω τις ρυθμίσεις της εικόνας που εισάγω καθώς διαφορετικά μπορεί να εμφανίζεται αλλοιωμένη. Όπως φαίνεται στην παρακάτω εικόνα είναι εξαιρετικά σημαντικό να ορίσω το μέγιστο μέγεθος της εικόνας ώστε να μην υπάρξουν προβλήματα με την ποιότητα

[2] Βλ. Παράρτημα Β


**Εικόνα 3.3 – Ρυθμίσεις εισαγόμενης εικόνας**

Με την οριστικοποίηση του background ουσιαστικά ολοκληρώνεται η πρώτη σκηνή της εφαρμογής. Στην συνέχεια θα προχωρήσω στην δημιουργία της σκηνής για την οθόνη των Credits. Παρακάτω μπορείτε να δείτε το τελικό αποτέλεσμα της πρώτης σκηνής αφού εκτελέσω την εφαρμογή σε μια συσκευή android.


**Εικόνα 3.4 – Το τελικό αποτέλεσμα της αρχικής σκηνής**

## 3.3 Info Scene

Η Info Scene (infoScene.unity στο project) είναι η σκηνή για την δημιουργία της οθόνης των πληροφοριών. Η πρόσβαση σε αυτή γίνεται μέσω της αρχικής σκηνής, πατώντας το κουμπί info που ανέλυσα παραπάνω. Περιέχει το λογότυπο, το κείμενο για τις πληροφορίες καθώς και ένα πλήκτρο επιστροφής στην αρχική οθόνη.

### 3.3.1 Ανάλυση των περιεχομένων

Η συγκεκριμένη σκηνή παρουσιάζει αρκετές ομοιότητες με την αρχική σκηνή που ανέλυσα παραπάνω αφού και αυτή αποτελείται από ένα script, μια κάμερα και ένα φόντο.

Όπως και στην προηγούμενη σκηνή, λοιπόν, θα πρέπει αρχικά να ορίσω στο script κάποιες παραμέτρους όπως τα skins και τα μεγέθη των γραμματοσειρών.

```
1.  public GUISkin mySkin;
2.  public GUIStyle infoStyle;
3.  public GUIStyle infoStyle2;
4.  public GUIStyle infoStyle3;
5.  void Update() {
6.      mySkin.label.fontSize = Screen.width / 10;
7.      infoStyle.fontSize = Screen.width / 25;
8.      infoStyle3.fontSize = Screen.width / 10;
9.      infoStyle2.fontSize = Screen.width / 50;
10.     mySkin.button.fontSize = Screen.width / 25;
11. }
```

**Πίνακας 3.6 – Τα skins και τα μεγέθη των fonts**

Στον παραπάνω πίνακα φαίνεται λοιπόν την δήλωση των skins που θα χρησιμοποιήσουμε. Απαραίτητη προϋπόθεση για να λειτουργήσει σωστά το script είναι η χρήση των namespaces καθώς της Monobehaviour class που είδαμε εκτεταμένα στην παράγραφο 3.2. Βλέπουμε στις 4 πρώτες γραμμές του κώδικα την δήλωση του skin και τριών styles που είναι απαραίτητα για να παραμετροποιήσω τα επιμέρους στοιχεία της σκηνής. Στην συνάρτηση Update, τέλος, έχω τα μεγέθη των γραμματοσειρών που υπολογίζονται πάντα με βάση το μέγεθος της οθόνης ώστε να προσαρμόζονται στην εκάστοτε συσκευή.

Στην συνέχεια ακολουθεί η συνάρτηση OnGUI με την οποία θα δημιουργήσω τα επιμέρους στοιχεία της σκηνής.

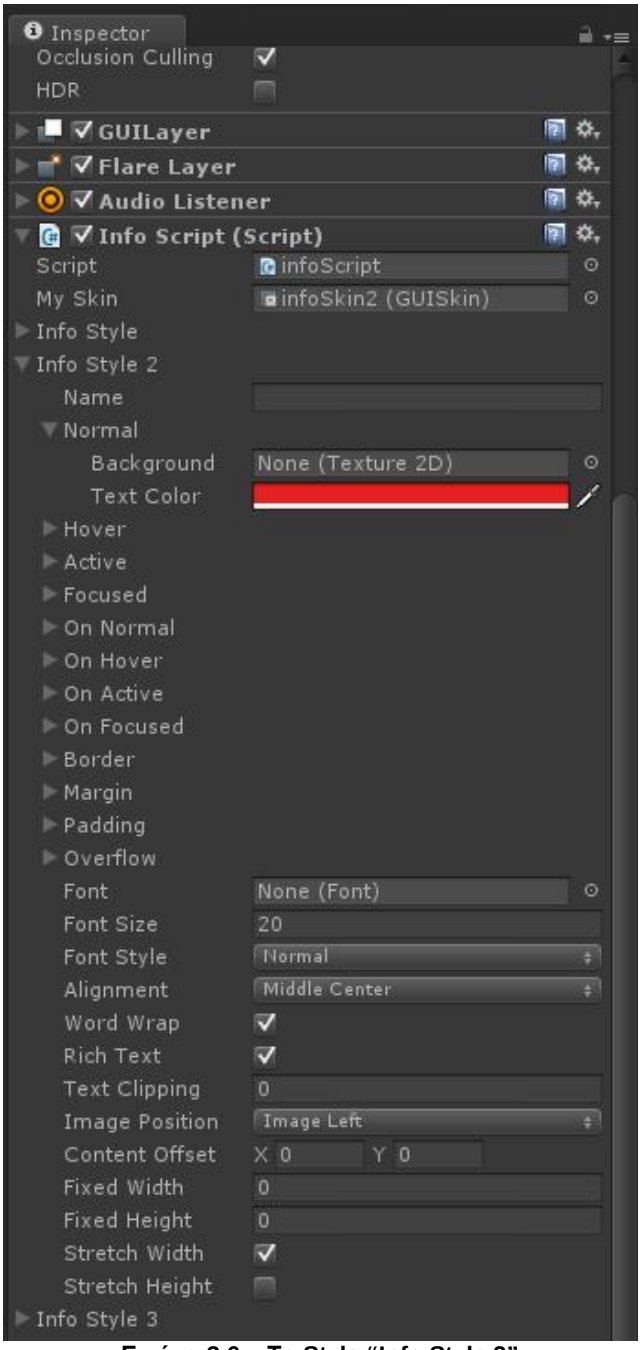
```
1.  void OnGUI() {
2.  GUI.skin = mySkin;
3.
4.  GUI.Label(new Rect((Screen.width - Screen.width * 0.5 f) -
     (Screen.width / 3) / 2, (Screen.height– Screen.height / 2 f) -
     (Screen.height / 7), Screen.width / 3, Screen.height / 7), "Info & Credits", in
    foStyle);
5.
6.  GUI.Label(new Rect((Screen.width - Screen.width * 0.63 f) -
     (Screen.width / 3) / 2,(Screen.height– Screen.height / 4) -
     (Screen.height / 7), Screen.width / 1.7 f, Screen.height / 7), "This app was cr
    eated as part of my
7.  thesis during my studies at the Informatics and Mass Media department of the Tec
    hnological Educational Institute of Western Greece ", infoStyle2);
8.
9.  GUI.Label(new Rect(0, 0, Screen.width, Screen.height), "Crazy \nRunner", infoSty
    le3);
10.
11. if (GUI.Button(new Rect((Screen.width - Screen.width * 0.5 f) -
     (Screen.width / 2.5 f) / 2, (Screen.height) -
     (Screen.height / 7), Screen.width / 2.5 f, Screen.height / 7), "Main Menu")) {

12. Application.LoadLevel(0);
13. }
14. }
```

**Πίνακας 3.7 – Η συνάρτηση OnGui στο Script της Info Scene**

Η συγκεκριμένη συνάρτηση δεν διαφέρει, ως προς την δομή, της από εκείνη που είδαμε στην παράγραφο 3.2.1. Έτσι λοιπόν έχουμε αρχικά την δήλωση του Skin και στην συνέχεια τρεις GUI.Label για την δημιουργία του λογότυπου, του τίτλου της σελίδας και του κειμένου. Εξαιρετικά σημαντική είναι η δήλωση των συντεταγμένων με χρήση του μεγέθους της οθόνης αλλά κυρίως των Style που δήλωσα παραπάνω. Αν δώ τον Inspector μετά την χρήση των Styles θα αντικρίσω την παρακάτω εικόνα.


**Εικόνα 3.5 – Ο Inspector για την σκηνή Info**

Στο Skin δημιούργησα το infoSkin2 το οποίο περιέχει όλες τις βασικές πληροφορίες για τα στοιχεία της σκηνής. Παρόλα αυτά επειδή πρέπει να αλλάξω κάποια επιμέρους στοιχεία θα χρησιμοποιήσω τα Styles που όρισα παραπάνω. Έτσι για να αλλάξωτο χρώμα και το Text Wrap στο κείμενο των πληροφοριών θα χρησιμοποιήσω το Info Style 2 σύμφωνα με το script.




**Εικόνα 3.6 – Το Style "Info Style 2"**

Τέλος θα πρέπει να αναφέρω την κάμερα και το Background που βρίσκονται στην σκηνή και τα οποία ανέλυσα διεξοδικά στη παράγραφο 3.2.2. Παρακάτω μπορείτε να δείτε το τελικό αποτέλεσμα της σκηνής αφού εκτελέσουμε την εφαρμογή σε μια συσκευή android.


**Εικόνα 3.7 – Το τελικό αποτέλεσμα της αρχικής σκηνής**

## 3.4 Main Scene (ή σκηνή παιχνιδιού)

Η κύρια σκηνή (ή σκηνή παιχνιδιού) είναι το βασικότερο κομμάτι της εφαρμογής μας αφού περιέχει ουσιαστικά όλα τα στοιχεία του παιχνιδιού. Θα εξετάσουμε λοιπόν όλα τα επιμέρους στοιχεία της σκηνής όπως ο χαρακτήρας, ο ήχος, οι πλατφόρμες και ο τρόπος που δημιουργούνται κλπ.

### 3.4.1 Χαρακτήρας ή παίχτης (character/player)

Ο χαρακτήρας του παιχνιδιού μας είναι ένα βασικό αλλά πολύπλοκο στοιχείο της εφαρμογής. Για να φτάσει στην τελική του μορφή χρειάζεται να δημιουργήσω την κίνηση, τα animations της κίνησης αλλά και την ίδια τη μορφή του χαρακτήρα. Στην συγκεκριμένη ενότητα, λοιπόν θα αναλύσω όλα τα στοιχεία που συνθέτουν τον χαρακτήρα μας σε τρείς υποενότητες: τα spritesheets, το animation και τα scripts μαζί με τα στοιχεία στον inspector.

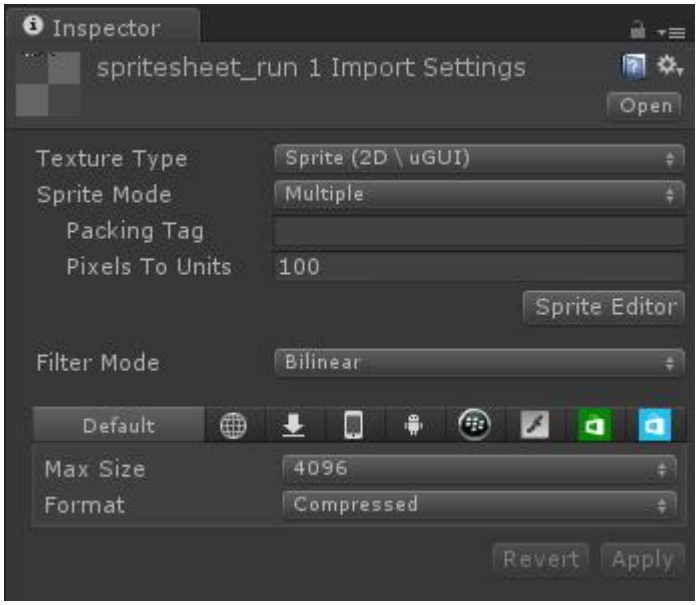
#### 3.4.1.1 Spritesheets

Για να μπορέσει να πάρει μορφή ο χαρακτήρας θα πρέπει πρώτα να την σχεδιάσω. Αυτή η διαδικασία θα πρέπει να γίνει με κάποιο εξωτερικό πρόγραμμα. Για το συγκεκριμένο project επέλεξα να χρησιμοποιήσω ένα σχέδιο που υπήρχε ήδη online και να το επεξεργαστώ χρησιμοποιώντας το πρόγραμμα Illustrator αφού στην αρχική του μορφή δεν ήταν Vector γραφικό.

Όταν σχεδιάζω την μορφή του χαρακτήρα θα πρέπει να το κάνω με σκοπό να προσομοιώσω μια κίνησή του όπως για παράδειγμα το τρέξιμο, το άλμα κτλ διαφορετικά θα έχω ένα ακίνητο αντικείμενο που απλα αλλάζει θέση στον χώρο. Για να εισάγω λοιπόν τα γραφικά με την κίνηση δημιουργώ ένα spritesheet. Ουσιαστικά πρόκειται για μία εικόνα που περιέχει διαδοχικά καρέ του γραφικού ή της μορφής που θέλω να προσομοιώσω την κίνηση.

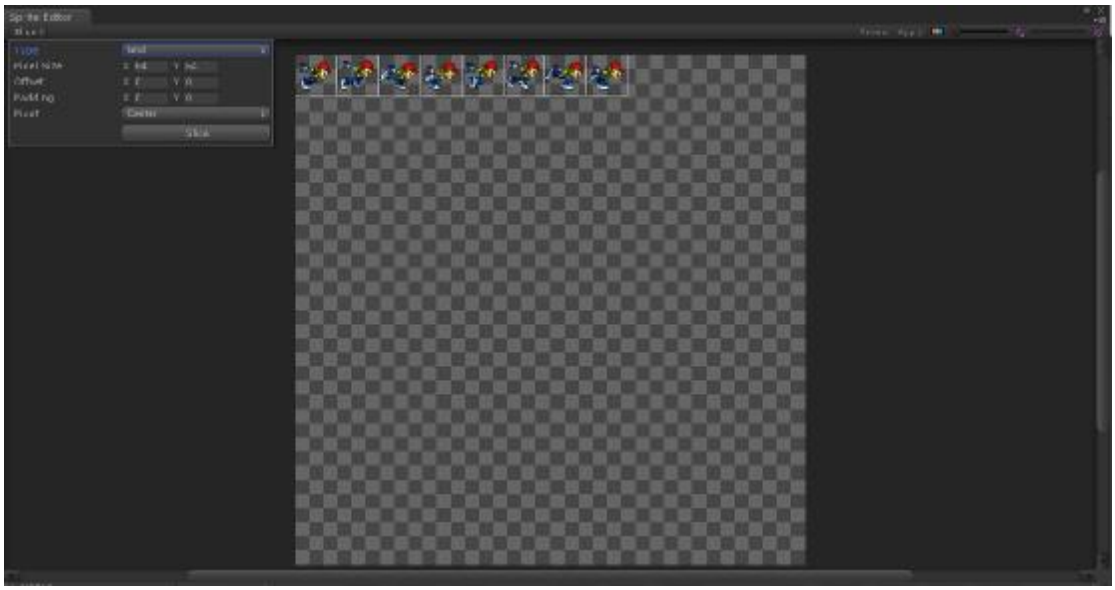
**Εικόνα 3.8 – Το Spritesheet για την κίνηση του τρεξίματος**

Για τον συγκεκριμένο χαρακτήρα αποφάσισα να χρησιμοποιήσω δύο spritesheets για την κίνηση του τρεξίματος και για την κίνηση του άλματος. Για να τα εισάγω αρκεί ένα δεξί κλικ στην καρτέλα Project και η επιλογή “Import New Asset”. Αφού επιλέξω την εικόνα είναι απαραίτητο να δηλώσω στο Unity ότι πρόκειται για Sprite και περιέχει πολλές εικόνες. Αυτή τη διαδικασία την κάνω μέσω του Inspector επιλέγοντας ως Texture Type το Sprite και αλλάζοντας το Sprite Mode σε Multiple. Όπως ανέλυσα στην ενότητα 3.2.2 είναι επίσης πολύ σημαντικό να ορίσω το μέγιστο μέγεθος της εικόνας.




**Εικόνα 3.9 – Οι επιλογές για το Spritesheet στον Inspector**

Με την αλλαγή που έκανα στην επιλογή Sprite Mode παρατηρώ την επιλογή “Sprite Editor” που εμφανίζεται. Η συγκεκριμένη λειτουργία δίνει την δυνατότητα να απομονώσω τα επιμέρους στοιχεία του Spritesheet που εισήγαγα ώστε να μπορέσω να ορίσω το κάθε καρέ της κίνησης παρακάτω. Πολύ χρήσιμο είναι κατά την δημιουργία του Spritesheet στο εξωτερικό πρόγραμμα να τοποθετήσω το κάθε καρέ σε ίση απόσταση με το δίπλα δημιουργώντας ένα είδους πλέγματος (grid). Αυτό θα βοηθήσει να πετύχω το καλύτερο δυνατό «κόψιμο» στον Sprite Editor.



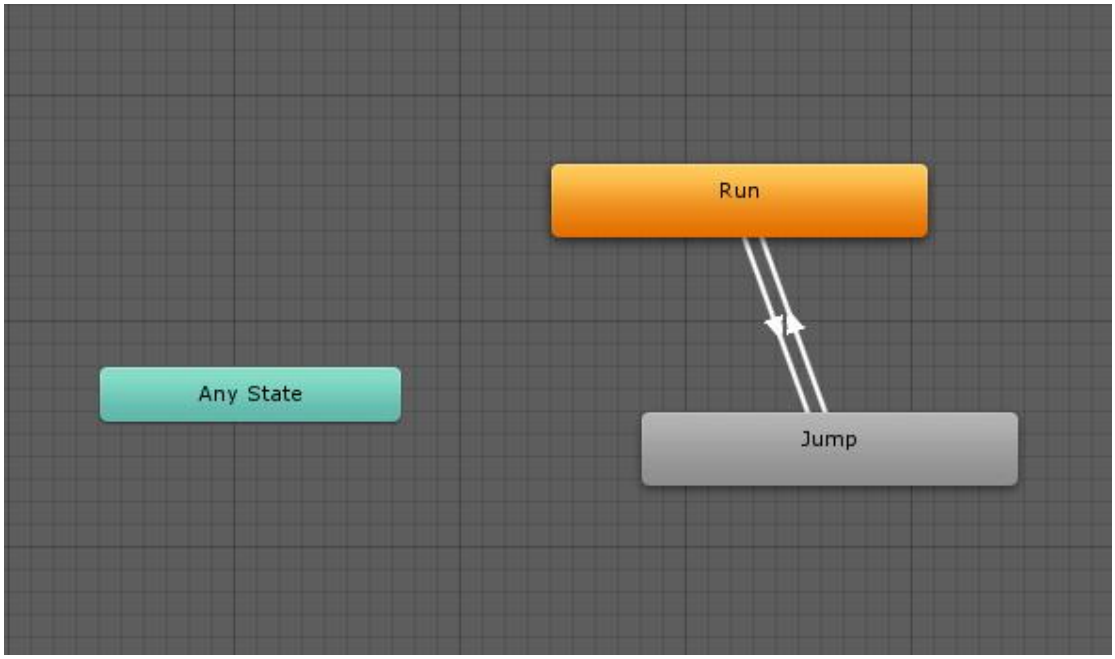
**Εικόνα 3.10 – Ο Sprite Editor**

Αφού λοιπόν ολοκληρώσω και αυτή την διαδικασία τότε τα sprites είναι έτοιμα για το επόμενο βήμα, το animation.

### 3.4.1.2 Animation

Το Animation είναι υπεύθυνο για την κίνηση του χαρακτήρα. Για να μπορέσει να αποκτήσει κίνηση ένα οποιοδήποτε αντικείμενο στο Unity χρειάζεται να έχει έναν Animator Component. Με τη σειρά του, θα πρέπει να αναφέρεται σε έναν Animation Controller ο οποίος περιέχει τις διαφορετικές ομάδες κινήσεων (τρέξιμο, άλμα κλπ.) καθώς και τις συνδέσεις μεταξύ τους.

Για τον συγκεκριμένο χαρακτήρα θα χρειαστώ δύο animations, ένα για την κίνηση του άλματος καθώς και ένα για την κίνηση του τρεξίματος. Αρχικά για να δημιουργήσω την κίνηση του άλματος θα πρέπει στον φάκελο Animations που βρίσκεται στα Assets να δημιουργήσω το Jump Animation και να το προσθέσω στον Animator. Στην συνέχεια θα πρέπει να κάνω το ίδιο και για το Run Animation. Στον Animator θα πρέπει να υπάρχει η ακόλουθη εικόνα.




**Εικόνα 3.11 – Ο Animator**

Όπως παρατηρείται ανάμεσα στις δύο κινήσεις υπάρχουν κάποιες συνδέσεις. Αυτές ορίζουν πως και πότε θα γίνει η μετάβαση από την μία κίνηση στην άλλη..

Στην καρτέλα animation θα δούμε πως υπάρχει ένα timeline. Εκεί θα πρέπει για κάθε κίνηση να εισάγω τα καρέ που έχουμε από τα Spritesheets που ανέφερα παραπάνω καθώς και να ορίσουμε και τα FPS (Frames Per Second) της κίνησης. Για την κίνηση του τρεξίματος τα frames ήταν οκτώ ενώ για το άλμα ήταν τέσσερα. Έπειτα από πειραματισμούς με την κίνηση και συνυπολογίζοντας την ταχύτητα της κάμερας, κατέληξα στο επιθυμητό framerate που εκφράζεται από τον παρακάτω μαθηματικό τύπο.

$$framerate = 2 * \sum_{i=1}^{n} i$$

Έτσι κατέληξα σε 8 fps για το άλμα και σε 16 fps για το τρέξιμο. Τα fps ή samples όπως αναφέρονται στο Unity ορίζονται επίσης στην καρτέλα animation. Μία ολοκληρωμένη κίνηση λοιπόν απεικονίζεται παρακάτω.


**Εικόνα 3.12 –Η κίνηση του Τρεξίματος**

### 3.4.1.3 Κίνηση στο χώρο

Στο προηγούμενο κεφάλαιο είδαμε πως γίνεται το animation του χαρακτήρα. Αυτό όμως δίνει την ψευδαίσθηση της κίνησης και δεν κινεί ουσιαστικά το αντικείμενο του χαρακτήρα στον χώρο. Για να γίνει αυτό θα πρέπει να δημιουργήσω και το αντίστοιχο αρχείο κώδικα (script) που θα είναι υπεύθυνο για αυτή την δουλειά.

Δημιουργώ λοιπόν δύο νέα scripts, το PlatformerCharacter2D και το Platformer2DUserControl. Το πρώτο περιέχει τις μεθόδους που αποτελούν την κίνηση και περιγράφουν πως αυτή θα εκτελεστεί και το δεύτερο είναι υπεύθυνο για την εκτέλεση τους. Θα μπορούσα να έχώ και ένα αρχείο που θα εκτελούσε τις συναρτήσεις κώδικα (functions) ωστόσο ακολουθώ μια MVC αρχιτεκτονική σε όλο το φάσμα της πτυχιακής εργασίας.

Στο script PlatformerCharacter2D, αρχικά, θα πρέπει να οριστικοποιήσω κάποιες global μεταβλητές που θα είναι διαθέσιμες σε όλες τις functions του script μας. Αυτές αφορούν κυρίως τον έλεγχο εδάφους (αν, δηλαδή, ο παίχτης βρίσκεται στο έδαφος ή είναι στον αέρα), την πλευρά που κοιτάζει, την μέγιστη ταχύτητα και την μέγιστη δύναμη κατά το άλμα, το animation κ.α. Κάποιες από τις μεταβλητές, όπως φαίνεται στον παρακάτω πίνακα, έχουν την σημείωση SerializeField δίπλα από τον τύπο τους. Αυτό σημαίνει πως είναι δυνατόν ο ορισμός της τιμής τους να γίνει από τον Inspector (βλ. Κεφάλαιο 2.3 – Γνωριμία με το περιβάλλον διεπαφής)

```
1.  bool facingRight = true;
2.  [SerializeField] public float maxSpeed = 10 f;
3.  [SerializeField] float jumpForce = 400 f;
4.  [SerializeField] bool airControl = false;
5.  [SerializeField] LayerMask whatIsGround;
6.  Transform groundCheck;
7.  float groundedRadius = .2 f;
8.  bool grounded = false;
9.  Transform ceilingCheck;
10. float ceilingRadius = .01 f;
```

```
11. Animator anim;
12. bool doubleJump = false;
13. HUDScript hudscore;
```

**Πίνακας 3.8 – Οι μεταβλητές του script PlatformerCharacter2D**

Στην συνέχεια θα πρέπει να αρχικοποιήσω κάποιες από τις μεταβλητές που όρισα παραπάνω. Για να το πετύχω αυτό θα δημιουργήσω την function Awake η οποία καλείται όταν εκτελείται και το script. Εκεί θα ορίσω τα αντικείμενα που είναι υπεύθυνα για το ground και ceiling check, τον animator και το αντικείμενο (component) του σκορ.

```
1.  void Awake() {
2.      // Setting up references.
3.      groundCheck = transform.Find("GroundCheck");
4.      ceilingCheck = transform.Find("CeilingCheck");
5.      anim = GetComponent < Animator > ();
6.      hudscore = GameObject.Find("Main Camera").GetComponent < HUDScript > ();
7.  }
```

**Πίνακας 3.9 – Η function Awake του script PlatformerCharacter2D**

Έπειτα και αφού τελειώσω με τις μεταβλητές ήρθε η ώρα να ορίσω τις δύο βασικότερες function του script, την FixedUpdate και την Move.

Η πρώτη εκτελείται σε κάθε frame ενώ η δεύτερη είναι υπεύθυνη για την κίνηση του χαρακτήρα. Στη FixedUpdate θα γίνει ο έλεγχος για το αν ο χαρακτήρας μας βρίσκεται στο έδαφος η όχι ώστε να περάσω το αποτέλεσμα, μαζί με την ταχύτητα της οριζόντιας κίνησης, στον animator. Στην συνέχεια αν ο χρήστης είναι στο έδαφος θα πρέπει να του ακυρώσω την δυνατότητα για διπλό άλμα αφού αυτό είναι που μπορεί να γίνει μόνο όταν ο χαρακτήρας είναι ήδη στον αέρα. Τέλος για να υπάρχει μία αυξανόμενη δυσκολία στο παιχνίδι, γίνεται ένας έλεγχος όπου όσο αυξάνεται το σκορ τόσο αυξάνεται και η ταχύτητα του παίχτη.

Στη function Move ουσιαστικά θα περάσω στον animator τις μεταβλητές για τις δύο βασικές κινήσεις, το τρέξιμο και το άλμα. Αρχικά θα περάσω στον animator τη μεταβλητή της κίνησης ώστε να επαναλάβει τα καρέ στην σωστή ταχύτητα και στην συνέχεια θα μετακινήσω τον χαρακτήρα μας προς τα δεξιά. Τέλος γίνεται ο έλεγχος για το αν επιτρέπεται να κάνει άλμα ο παίχτης ώστε να μετακινηθεί ο χαρακτήρας με μια συγκεκριμένη δύναμη που θα του ασκηθεί προς τα πάνω.

```
1.  void FixedUpdate() {
2.
3.  // The player is grounded if a circlecast to the groundcheck position hits anyth
    ing designated as ground
4.
    grounded = Physics2D.OverlapCircle(groundCheck.position, groundedRadius, whatIs
    Ground);
5.      anim.SetBool("Ground", grounded); // Set the vertical animation
6.      anim.SetFloat("vSpeed", rigidbody2D.velocity.y);
7.
        if (grounded) doubleJump = false;
8.
        if (hudscore.playerScore >= 100 && hudscore.playerScore < 110)
            maxSpeed = 10.5 f;
9.      if (hudscore.playerScore >= 170 && hudscore.playerScore < 180)
            maxSpeed = 11.5 f;
10.     if (hudscore.playerScore >= 250 && hudscore.playerScore < 260)
            maxSpeed = 12.5 f;
11.     if (hudscore.playerScore >= 300 && hudscore.playerScore < 310)
            maxSpeed = 13 f;
12. }

13.
14.
```

```
15. public void Move(float move, bool jump) {
        //only control the player if grounded or airControl is turned on
16.     if (grounded || airControl) {
    // The Speed animator parameter is set to the absolute value of the horizontal i
    nput.
17.         anim.SetFloat("Speed", Mathf.Abs(move)); // Move the character
18.         rigidbody2D.velocity = new Vector2(move * maxSpeed, rigidbody2D.velocity
    .y)
19.
        }
        // If the player should jump...
20.     if ((grounded || !doubleJump) && jump) { // Add a vertical force to the play
    er.
21.         anim.SetBool("Ground", false);
22.         rigidbody2D.velocity = new Vector2(rigidbody2D.velocity.x, 0);
23.         rigidbody2D.AddForce(new Vector2(0 f, jumpForce));
24.         if (!grounded) doubleJump = true;
25.     }
26. }
```

**Πίνακας 3.10 – Οι function FixedUpdate και Move του script PlatformerCharacter2D**

Με αυτές τις δύο function ολοκληρώνεται και το script PlatformerCharacter2D. Για να δω όπως το αποτέλεσμα χρειάζομαι και ένα ακόμα script, το Platformer2DUserControl. Το συγκεκριμένο είναι πολύ πιο απλό σε δομή και ουσιαστικά αποτελείται από τρείς function, την Awake, την Update και την FixedUpdate. Στην πρώτη θα επιλέξω τον χαρακτήρα, στην δεύτερη θα ορίσω ότι με το πάτημα ενός κουμπιού (ή με το πάτημα της οθόνης στην συγκεκριμένη περίπτωση) πρέπει να γίνει το άλμα και στη τρίτη θα περάσω στην function Move, που είχα ορίσει στο προηγούμενο script, τις παραμέτρους που χρειάζεται για να εκτελέσει την κίνηση και θα αρχικοποιήσω ξανά την μεταβλητή για το άλμα.

```
1.  using UnityEngine;

2.  [RequireComponent(typeof(PlatformerCharacter2D))]
    public class Platformer2DUserControl: MonoBehaviour {
3.
        private PlatformerCharacter2D character;
4.      public bool jump;
5.      void Awake() {
6.          character = GetComponent < PlatformerCharacter2D > ();
7.      }
8.      void Update() {
9.          if (Input.GetMouseButtonDown(0)) jump = true;
10.     }
11.     void FixedUpdate() {

            // Pass all parameters to the character control script.
12.         character.Move(1, jump);

            // Reset the jump input once it has been used.
13.         jump = false;
14.     }
15. }
```

**Πίνακας 3.11 –Το script Platformer2DUserControl**

#### 3.4.1.4 Inspector Panel

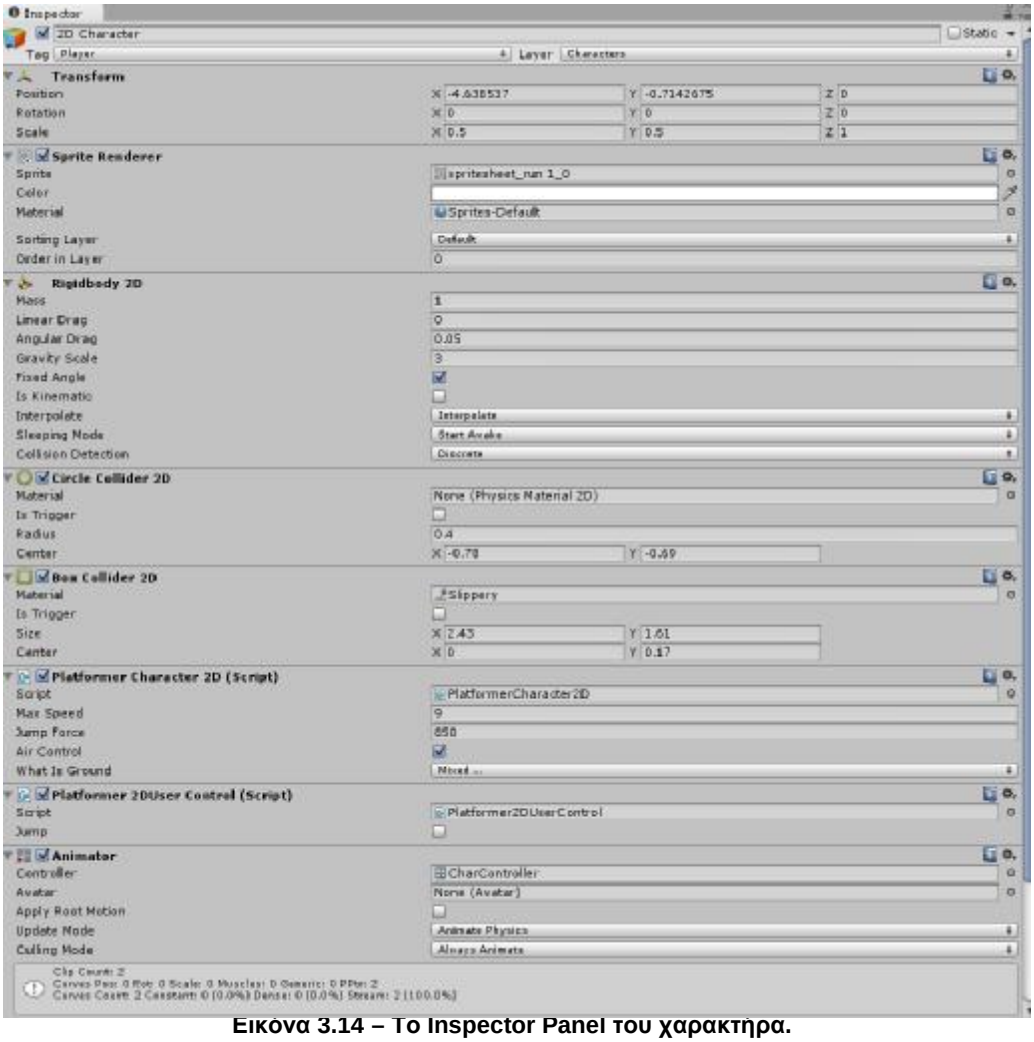
Ορίζοντας την κίνηση για τον χαρακτήρα θα δούμε πράγματι το αντικείμενο να κινείται στο χώρο. Όμως ακόμα δεν μπορεί να αλληλοεπιδράσει με τα αντικείμενα γύρω του. Έτσι αντί να πατάει στις πλατφόρμες που θα διαμορφώσουμε στην συνέχεια θα περνάει από μέσα τους και θα χάνεται από το κάδρο. Έτσι θα πρέπει να του ορίσω μερικά ακόμα στοιχεία μέσω του inspector panel.

Αρχικά θα πρέπει να ορίσω κάποιους colliders. Οι colliders είναι σχήματα τα οποία δίνουν τα φυσικά όρια του χαρακτήρα και θα του δώσουν την δυνατότητα να έχει επαφή με άλλα αντικείμενα.


**Εικόνα 3.13 – Ο χαρακτήρας με την προσθήκη colliders**

Όπως φαίνεται ο βασικός collider καλύπτει το μεγαλύτερο μέρος του χαρακτήρα, ενώ τα δύο μικρότερα εξυπηρετούν στο να μην κολλάει ο χαρακτήρας σε γωνίες.

Από την στιγμή που ο χαρακτήρας μας είναι ένα 2D αντικείμενο τότε θα πρέπει να ορίσω και κάποια χαρακτηριστικά όπως η μάζα, το πόσο τον επηρεάζει η βαρύτητα και κατά πόσο η γωνία του αντικειμένου είναι σταθερή. Αφού εισάγω τις απαραίτητες τιμές φτάνουμε στην ολοκλήρωση του χαρακτήρα. Το Inspector Panel θα πρέπει να έχει την ακόλουθη μορφή.




**Εικόνα 3.14 – Το Inspector Panel του χαρακτήρα.**

### 3.4.2 Κάμερες και αντικείμενα

Αφού λοιπόν ολοκληρώθηκε η δημιουργία του χαρακτήρα μας, θα πρέπει να δημιουργήσω τα υπόλοιπα στοιχεία της σκηνής όπως την κεντρική κάμερα, την κάμερα του background, τις πλατφόρμες που θα πατάει ο χαρακτήρας και τα αντικείμενα που θα δίνουν τους πόντους.

#### 3.4.2.1 Background και κάμερα

Για το background (φόντο) θα χρειαστεί να εισάγω ένα γραφικό και μία κάμερα. Η εικόνα θα πρέπει να είναι εμφωλευμένη στην κάμερα ώστε να είναι πάντα στο κάδρο και να μην ξεφεύγει.

Αρχικά λοιπόν έχω δημιουργήσει μία κάμερα με ορθογραφική προβολή με μέγεθος αρκετό ώστε να χωράει οριζόντια την εικόνα. Στην συνέχεια δημιούργησα ένα quad αντικείμενο πάνω στο οποίο θα βάλω την εικόνα. Με αυτό το βήμα θα μπορούσε να ολοκληρωθεί το συγκεκριμένο κομμάτι. Παρόλα αυτά επειδή θέλω να δώσω μια κίνηση στην εικόνα, δημιούργησα το παρακάτω script για την εικόνα.

```
1.  using UnityEngine;
2.  using System.Collections;
3.  public class ScrollScript: MonoBehaviour {
4.      public float speed = 0;
5.      void Update() {
6.          renderer.material.mainTextureOffset = new Vector2((Time.time * speed) %
    1, 0 f);
7.      }
8.  }
```

**Πίνακας 3.12 –Το script ScrollScript**

Αυτό που ουσιαστικά κάνει το συγκεκριμένο script είναι να μεταβάλει το offset της εικόνας προς τα δεξιά με μία σταθερή ταχύτητα. Έτσι δίνεται η αίσθηση της κίνησης.

#### 3.4.2.2 Κεντρική πλατφόρμα

Όταν ξεκινάει το παιχνίδι, ο χαρακτήρας θα πρέπει να έχει να σταθερό σημείο για να ξεκινήσει. Έτσι δημιούργησα μια σταθερή πλατφόρμα πάνω στην οποία ο χαρακτήρας θα μπορεί να «πατήσει» για να ξεκινήσει την κίνηση του.

Για να δημιουργηθεί η σταθερή πλατφόρμα θα πρέπει πρώτα να δημιουργήσω κάποια prefabs. Τα prefabs είναι επαναχρησιμοποιούμενα αντικείμενα που μπορεί να είναι μεμονωμένα ή ομάδες πολλών. Έτσι δημιούργησα ένα quad αντικείμενο στο οποίο έβαλα την εκάστοτε εικόνα δημιουργώντας στο σύνολο τρία τέτοια αντικείμενα. Στην συνέχεια ομαδοποίησα τα τρία αντικείμενα κάτω από ένα κενό αντικείμενο και από αυτό δημιούργησα το prefab για την πλατφόρμα. Στα επιμέρους αντικείμενα που δημιουργήσαμε είναι απαραίτητο να εισάγω και ένα collider ώστε να αλληλοεπιδρά με τον χαρακτήρα και να μην περνά μέσα του. Το τελικό αποτέλεσμα αφού τοποθέτησα τρία τέτοια αντικείμενα στην σειρά είναι το παρακάτω.


**Εικόνα 3.15 – Το αντικείμενο largeGround.**

### 3.4.2.3 Κεντρική κάμερα και γεννήτριες αντικειμένων

Το τελευταίο μεγάλο κομμάτι της κύριας σκηνής είναι η κεντρική κάμερα, που ακολουθεί την κίνηση του παίχτη και οι γεννήτριες αντικειμένων (spawners) που δημιουργούν νέα αντικείμενα όπως πλατφόρμες και νομίσματα.

Η κεντρική κάμερα είναι αυτή που ουσιαστικά μας δείχνει ότι έχουμε δημιουργήσει. Η κεντρική κάμερα όπως και η κάμερα του background θα έχει ορθογραφική προβολή. Όμως θα χρειαστεί κάποια επιπλέον scripts. Αρχικά δημιούργησα το script για να ακολουθεί η κάμερα την κίνηση του παίχτη και να μην μένει σε ένα σταθερό σημείο.

```
1.  using UnityEngine;
2.  using System.Collections;
3.  public class CameraRunnerScript: MonoBehaviour {
4.      public Transform player;
5.      void Update() {
6.          transform.position = new Vector3(player.position.x + 6, 0, -10);
7.      }
8.  }
```

**Πίνακας 3.13 –Το script CameraRunner**

Όπως φαίνεται είναι ένα εξαιρετικά απλό στην υλοποίηση script. Δέχεται σαν παράμετρο το αντικείμενο του παίχτη και με βάση την θέση του αλλάζει την θέση της κάμερας στην αντίστοιχη κατεύθυνση.

Αφού δημιούργησα το script για την κίνηση της κάμερας, χρειάζεται και άλλο ένα για να απεικονίζεται το score.

```
1.  public class HUDScript: MonoBehaviour {
2.      public GUISkin scoreSkin;
3.      public float playerScore = 0;
4.      PlatformerCharacter2D platformer; // Update is called once per frame
5.      void Update() { //playerScore += Time.deltaTime;
6.          scoreSkin.label.fontSize = Screen.width / 25;
7.      }
8.      public void increaseScore(int amount) {
9.          playerScore += amount;
10.     }
11.     void OnDisable() {
12.         PlayerPrefs.SetInt("Score", (int)(playerScore * 100));
13.     }
14.     void OnGUI() {
15.         GUI.skin = scoreSkin;
16.         GUI.Label(new Rect(10, 10, Screen.width, 160), "Score: " + (int)(playerS
    core * 100));
17.     }
18. }
```

**Πίνακας 3.14 –Το script HUDScript**

Όπως φαίνεται στον παρακάτω πίνακα, στην function Update ορίζω το μέγεθος της γραμματοσειράς με βάση το μέγεθος της οθόνης (η γραμματοσειρά και το μέγεθος ορίζονται από τα skins), στη function increaseScore αυξάνω το score του παίχτη (θα χρησιμοποιηθεί από τα αντικείμενα που θα ορίσουμε παρακάτω), στην OnDisable ορίζω το τελικό σκορ του παίχτη όταν χάσει και τέλος στην OnGUI απεικονίζω το σκορ του παίχτη στην οθόνη.

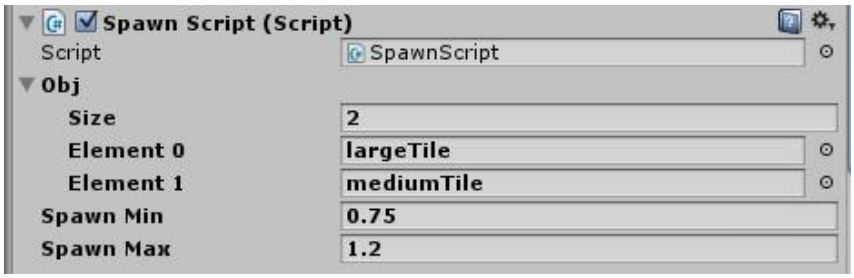
Με αυτά τα δύο script ολοκληρώνεται και η κεντρική κάμερα. Στην συνέχεια θα πρέπει να δημιουργήσω τις πλατφόρμες, τα νομίσματα καθώς και το τρόπο με τον οποίο ο παίχτης θα χάνει το παιχνίδι.

Για τις πλατφόρμες θα χρησιμοποιήσω τα prefabs που είχα δημιουργήσει νωρίτερα για την κεντρική πλατφόρμα. Όμως αφού το παιχνίδι δεν αποτελείται από πολλά επίπεδα αλλά ουσιαστικά τελειώνει μόνο όταν χάσει ο παίχτης, θα χρειαστεί να δημιουργήσω μια γεννήτρια που θα παράγει συνεχώς πλατφόρμες με τυχαία συχνότητα. Για να μην είναι πολύ δύσκολο για τον χρήστη αποφάσισα να δημιουργήσω δύο spawners που θα γεννούν πλατφόρμες σε δύο διαφορετικά ύψη. Έτσι δημιούργησα ένα quad αντικείμενο με μοναδικό περιεχόμενο το script που αναδημιουργεί πλατφόρμες.

```
1.  using UnityEngine;
2.  using System.Collections;
3.  public class SpawnScript: MonoBehaviour {
4.      public GameObject[] obj;
5.      public float SpawnMin = 1 f;
6.      public float SpawnMax = 2 f; // Use this for initialization
7.      void Start() {
8.          Spawn();
9.      }
10.     void Spawn() {
11.         Instantiate(obj[Random.Range(0, obj.GetLength(0))], transform.position,
    Quaternion.identity);
12.         Invoke("Spawn", Random.Range(SpawnMin, SpawnMax));
13.     }
14. }
```

**Πίνακας 3.15 –Το script SpawnScript**

Το script ουσιαστικά δημιουργεί ένα νέο αντικείμενο και τις μεταβλητές για την μέγιστη και ελάχιστη συχνότητα για την δημιουργία μιας πλατφόρμας. Η function Start ξεκινά την function Spawn που με την σειρά της φτιάχνει ένα νέο αντικείμενο από τα αντικείμενα που θα ορίσουμε στον Inspector (βλ. εικόνα 3.12) σε μία τυχαία στιγμή με βάση την συχνότητα που ορίσαμε.




**Εικόνα 3.16 – Οι παράμετροι του Script στον Inspector**

Ακριβώς με τον ίδιο τρόπο δημιούργησα και τα νομίσματα που δίνουν πόντους στον παίχτη. Η μόνη διαφορά είναι πως το prefab του εκάστοτε νομίσματος περιλαμβάνει και ένα script που σκοπό έχει να δίνει διαφορετικούς πόντους ανάλογα με την αξία που έχει το κάθε νόμισμα.

```
1.  using UnityEngine;
2.  using System.Collections;
3.  public class SilverCoinScript: MonoBehaviour {
4.      HUDScript hud;
5.      void OnTriggerEnter2D(Collider2D other) {
6.          if (other.tag == "Player") {
7.              hud = GameObject.Find("Main Camera").GetComponent < HUDScript > ();

8.              hud.increaseScore(5);
9.              Destroy(this.gameObject);
10.         }
11.     }
12. }
```

**Πίνακας 3.16 –Το script CoinScript**

Το script αποτελείται από μόνο μία function όπου ορίζει ότι στην επαφή του νομίσματος με ένα άλλο αντικείμενο[3] θα πρέπει να αυξηθεί το score ανάλογα με την αξία που ορίζω και παράλληλα να καταστραφεί το συγκεκριμένο αντικείμενο.

Τέλος, όταν ο παίχτης δεν καταφέρνει τα ακουμπήσει σωστά πάνω σε μία πλατφόρμα και πέσει στο κενό θα πρέπει να χάνει. Για να το πετύχω αυτό δημιούργησα ένα αντικείμενο κάτω από την κάμερα, ώστε να μην φαίνεται στο κάδρο μας, που όταν έρθει σε επαφή με οποιοδήποτε αντικείμενο το καταστρέφει.

```
1.  using UnityEngine;
2.  using System.Collections;
3.  public class DestroyerScript: MonoBehaviour {
4.      void OnTriggerEnter2D(Collider2D other) {
5.          if (other.tag == "Player") {
6.              Application.LoadLevel(2);
7.              return;
8.          }
9.          if (other.gameObject.transform.parent) {
10.             Destroy(other.gameObject.transform.parent.gameObject);
11.         } else {
12.             Destroy(other.gameObject);
13.         }
14.     }
15. }
```

**Πίνακας 3.17 –Το script DestroyerScript**

Στο παραπάνω script υπάρχουν δύο διαφορετικές περιπτώσεις. Η πρώτη είναι ότι το αντικείμενο που έρχεται σε επαφή είναι ο παίχτης οπότε και θα φορτώσει την σκηνή Game Over και η δεύτερη περίπτωση είναι να είναι οποιοδήποτε αντικείμενο οπότε και το καταστρέφει. Ο λόγος που γίνεται αυτό είναι γιατί αυτό το αντικείμενο θα πρέπει να τοποθετηθεί και δεύτερη φορά στην σκηνή, κάθετα, πίσω από τον παίχτη και έξω από το κάδρο για να καταστρέφει όσα αντικείμενα βγαίνουν εκτός σκηνής (πλατφόρμες, νομίσματα). Αυτό γίνεται ώστε να μην γεμίζει η εφαρμογή μας από

[3] Αν και το άλλο αντικείμενο που μπορεί να ακουμπήσει το νόμισμα είναι μόνο ο παίχτης, θα πρέπει να γίνεται ένας έλεγχος για να αποφευχθεί να τρέξει ο κώδικας σε αντίθετη περίπτωση.

αντικείμενα και χρειάζεται συνεχώς περισσότερη μνήμη και μεγαλύτερη υπολογιστική ισχύ από την συσκευή.

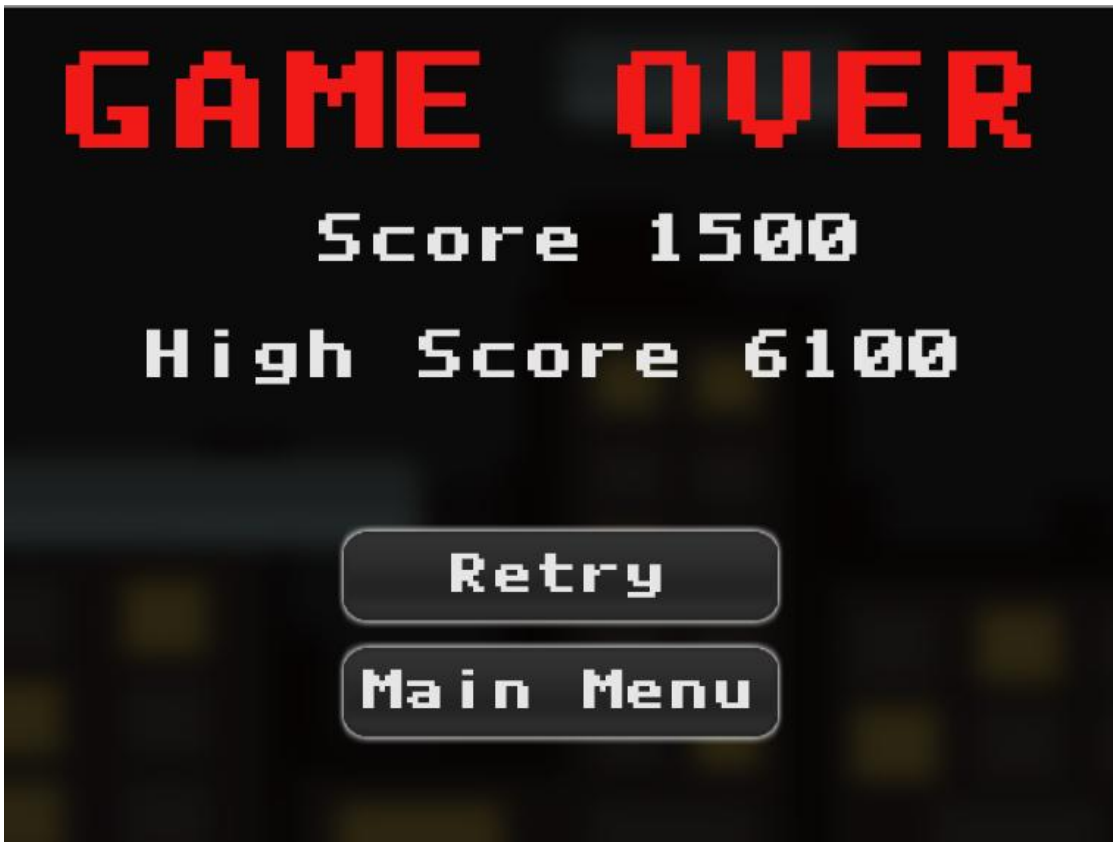
#### 3.4.2.4 Μουσική

Για να ολοκληρώσω την σκηνή έχω προσθέσει και μία μουσική κατά την διάρκεια του παιχνιδιού. Για να το πετύχω αυτό δημιούργησα ένα κενό αντικείμενο με μία πηγή ήχου στην οποία φόρτωσα το αρχείου ήχου που έχω επιλέξει.

Με αυτό ολοκληρώνεται και η κεντρική σκηνή του παιχνιδιού. Στην συνέχεια θα δημιουργήσω την σκηνή Game Over που αποτελεί και το τελευταίο κομμάτι της εφαρμογής.

## 3.5 Game Over Scene

Όταν ο χαρακτήρας δεν καταφέρνει να μείνει πάνω στις πλατφόρμες τότε το παιχνίδι τελειώνει. Για να καταλάβει λοιπόν ο χρήστης πως το παιχνίδι τελείωσε και να ξεκινήσει ξανά από την αρχή τότε πρέπει να ανακατευθυνθεί σε μία νέα σκήνη όπου εμφανίζω ένα μήνυμα που τον ενημερώνει, το σκορ και την καλύτερη επίδοση του από την πρώτη φορά που έπαιξε, και τις επιλογές για να προσπαθήσει ξανά ή να επιστρέψει στο κεντρικό μενού.




**Εικόνα 3.17 – Η σκηνή Game Over**

Η υλοποίηση της σκηνής είναι αρκετά απλή αφού αποτελείται από μια κάμερα με ένα αντικείμενο για το background εμφωλευμένο σε αυτήν. Τα επιμέρους στοιχεία της σκηνής εμφανίζονται στην οθόνη λόγω του script που είναι συνδεδεμένο με την κάμερα.

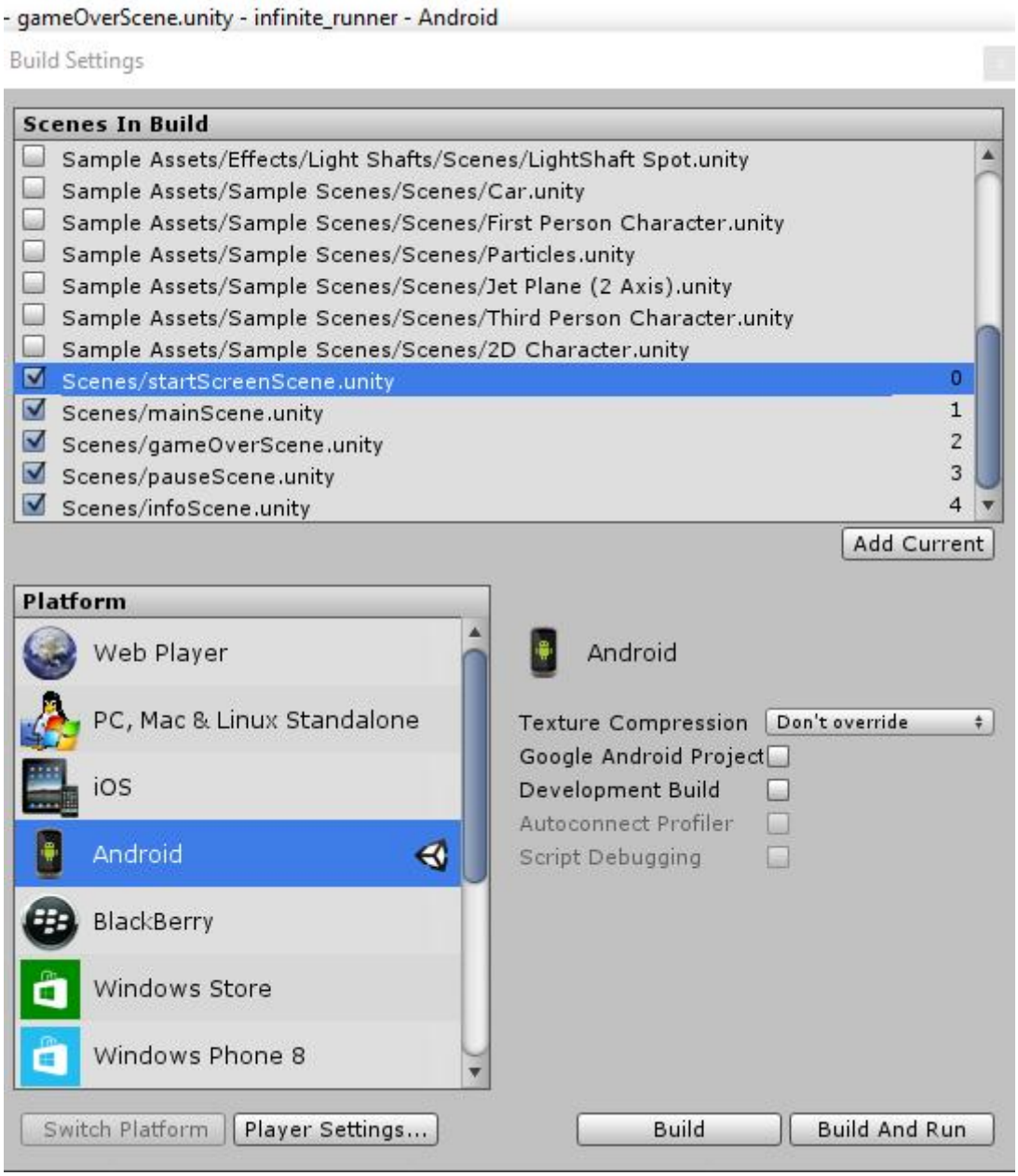
```
1.  using UnityEngine;
2.  using System.Collections;
3.  public class GameOverScript: MonoBehaviour {
4.      int score = 0;
5.      int highscore = 0;
6.      public GUISkin gameOverSkin;
7.      public GUIStyle myStyle;
8.      void Start() {
9.          score = PlayerPrefs.GetInt("Score");
10.         highscore = PlayerPrefs.GetInt("highscore");
11.         if (score > highscore) {
12.             highscore = score;
13.             PlayerPrefs.SetInt("highscore", score);
14.         }
15.     }
16.     void Update() {
17.         if (Input.GetKeyDown(KeyCode.Escape)) Application.Quit();
18.         myStyle.fontSize = Screen.width / 10;
19.         gameOverSkin.label.fontSize = Screen.width / 20;
20.         gameOverSkin.button.fontSize = Screen.width / 25;
21.     }
22.     void OnGUI() {
23.         GUI.skin = gameOverSkin;
24.         GUI.Label(new Rect(0, 0, Screen.width, Screen.height), "GAME OVER", myStyle);
25.         GUI.Label(new Rect((Screen.width - Screen.width * 0.5 f) - (Screen.width / 2.3 f) / 2, (Screen.height - Screen.height / 1.5 f) - (Screen.height / 7), Screen.width, 100), "Score " + score);
26.         GUI.Label(new Rect((Screen.width - Screen.width * 0.5 f) - (Screen.width / 1.35 f) / 2, (Screen.height - Screen.height / 1.5 f) - (Screen.height / 7) * 0.02 f, Screen.width, 100), "High Score " + highscore);
27.         if (GUI.Button(new Rect((Screen.width - Screen.width * 0.5 f) - (Screen.width / 2.5 f) / 2, (Screen.height - Screen.height / 4) - (Screen.height / 7), Screen.width / 2.5 f, Screen.height / 7), "Retry")) {
28.             Application.LoadLevel(1);
29.         }
30.         if (GUI.Button(new Rect((Screen.width - Screen.width * 0.5 f) - (Screen.width / 2.5 f) / 2, (Screen.height - Screen.height / 4) - (Screen.height / 7) * 0.02 f, Screen.width / 2.5 f, Screen.height / 7), "Main Menu")) {
31.             Application.LoadLevel(0);
32.         }
33.     }
34. }
```

**Πίνακας 3.18 –Το script GameOverScript**

Αποτελείται από τρεις βασικές function, την Start, την Update και την OnGUI. Η πρώτη αναλαμβάνει να φέρει το σκορ από το τελευταίο παιχνίδι, το καλύτερο σκορ του χρήστη καθώς και αν το τελευταίο σκορ είναι μεγαλύτερο από το καλύτερο σκορ τότε να ενημερώσει την σκηνή του τελευταίου. Η Update είναι αυτή που επιτρέπει στον χρήστη να βγει τελείως από το παιχνίδι πατώντας το κουμπί πίσω στην συσκευή του και ενημερώνει τα μεγέθη των γραμματοσειρών σε συνάρτηση με το μέγεθος της οθόνης. Τέλος η OnGUI είναι υπεύθυνη για να «τυπώσει» στη σκηνή το μήνυμα τέλους, τα σκορ και τα κουμπιά καθώς και να αναθέσει τις ενέργειες τους.

## 3.6 Δημιουργία APK αρχείου για την εφαρμογή

Αφού ολοκλήρωσα την εφαρμογή, είναι απαραίτητο να κάνω build σε ένα αρχείο APK ώστε να είναι εκτελέσιμη από κινητά με λειτουργικό σύστημα Android. Την διαδικασία αυτή την αναλαμβάνει το ίδιο το Unity. Μοναδική προϋπόθεση είναι να υπάρχει στον υπολογιστή το SDK του Android.




**Εικόνα 3.18 – Τα Build Settings**

Από εδώ, λοιπόν, πρέπει να επιλέξω τις σκηνές που θέλω να χτίσει το Unity με την σωστή σειρά. Επίσης από τα Player Settings έχω την δυνατότητα να επιλέξω το εικονίδιο, την splash screen και άλλες ρυθμίσεις τόσο για την εφαρμογή μου όσο και για την συσκευή.

# Κεφάλαιο 4 – Συμπεράσματα

## 4.1 Ευρήματα

Τα παιχνίδια, σήμερα, αποτελούν ένα αναπόσπαστο κομμάτι της καθημερινότητας εκατομμυρίων ανθρώπων στον κόσμος και αποτελούν μια βιομηχανία με πολύ μεγάλο τζίρο στην παγκόσμια οικονομία. Με την τεχνολογία να είναι πιο προσβάσιμη στις μέρες μας από μεγαλύτερο κομμάτι του παγκόσμιου πληθυσμού φαίνεται πως τα ηλεκτρονικά παιχνίδια κερδίζουν την μερίδα του λέοντος τόσο στην προτίμηση όσο και στο οικονομικό κομμάτι. Η ανάπτυξη, λοιπόν, μιας τέτοιας εφαρμογής μπορεί να φέρει στην εταιρία αναγνωσιμότητα και κέρδος. Δεν είναι τυχαίο πως εταιρίες που παραδοσιακά δραστηριοποιούνται στον χώρο των συμβατικών παιχνιδιών (πχ Hasbro) έχουν στρέψει την προσοχή τους σήμερα στην ανάπτυξη εφαρμογών παιχνιδιών.

Όποιο και να είναι το κίνητρο οι δυνατότητες που προσφέρουν τα παιχνίδια και ιδιαίτερα οι εφαρμογές είναι απεριόριστες. Με τα Android κινητά να κατέχουν το μεγαλύτερο κομμάτι της πίτας της αγοράς ένας προγραμματιστής μπορεί να φτιάξει αυτό που πάντα ήθελε και να το διανείμει σε ολόκληρο τον κόσμος σε μερικές ώρες.

## 4.2 Προβλήματα – Προκλήσεις

Κατά την διάρκεια της εκπόνησης της συγκεκριμένης εργασίας ήρθα αντιμέτωπος με αρκετά προβλήματα αλλά και προκλήσεις. Το βασικότερο πρόβλημα θεωρώ πως ήταν ο χρόνος που έπρεπε να δαπανηθεί για την υλοποίηση της. Επίσης βασικό πρόβλημα ήταν η εύρεση κατάλληλων υλικών που θα μπορούσαν να χρησιμοποιηθούν ελεύθερα για την δημιουργία της εφαρμογής. Τέλος ήταν μεγάλη πρόκληση η υλοποίηση της εφαρμογής λόγω της μικρής κοινότητας developers για Unity.

## 4.3 Μελλοντική δουλειά – Προτάσεις

Στην συγκεκριμένη εργασία θα ήταν εξαιρετική πρόκληση η επαναδημιουργία της με καλύτερα γραφικά και με εφαρμογές UX (User eXperience). Επίσης θα ήταν ενδιαφέρον η δημιουργία ενός παιχνιδιού με ιστορία και επίπεδα βασισμένο σε αυτή την πτυχιακή

# Παράρτημα Α

## Το δίλημμα του φυλακισμένου

Το δίλημμα του φυλακισμένου αποτελεί ένα χαρακτηριστικό πρόβλημα της θεωρίας παιγνίων που αναλύσαμε εκτενώς στο Κεφάλαιο 1: θεωρία και ιστορία του gaming. Σκοπός είναι η ανάλυση των στρατηγικών επιλογών των «λογικά» σκεπτόμενων ατόμων που συμμετέχουν στο πείραμα.

Στο εν λόγω, λοιπόν, πρόβλημα δύο άτομα (εφεξής θα αναφέρονται ως Α και Β) συλλαμβάνονται από την αστυνομία ως ύποπτοι διάπραξης κάποιων εγκλημάτων. Μη έχοντας αρκετά στοιχεία εναντίων τους η αστυνομία αποφασίζει να τους βάλει σε ξεχωριστά δωμάτια, αποτρέποντας τους από το να έχουν οποιαδήποτε επικοινωνία μεταξύ τους. Ο εισαγγελέας επισκέπτεται και τους δύο ξεχωριστά και κάνει στον καθένα τις παρακάτω προτάσεις:

- Αν καταθέσει εναντίων του άλλου (και ο άλλος δεν μιλήσει) τότε θα αφεθεί ελεύθερος με τον άλλο κατηγορούμενο να τιμωρείται με 12 χρόνια φυλάκιση.
- Αν δεν μιλήσει ούτε ο ένας ούτε ο άλλος τότε και οι δύο θα τιμωρηθούν με 1 χρόνο φυλάκιση και οι δύο για ήσσονος σημασίας αδικήματα για τα οποία η αστυνομία έχει αποδείξεις
- Αν μιλήσουν και οι δύο τότε θα τιμωρηθούν με 4 χρόνια έκαστος

| | **Αν μιλήσει ο Β** | **Αν δεν μιλήσει ο Β** |
|---|---|---|
| **Αν μιλήσει ο Α** | 4 χρόνια έκαστος | Ελεύθερος ο Α, 12 χρόνια ο Β |
| **Αν δεν μιλήσει ο Α** | Ελεύθερος ο Β, 12 χρόνια ο Α | 1 χρόνο έκαστος |

**Πίνακας Α.1 - Το δίλημμα του φυλακισμένου**

Η θεωρία παιγνίων λοιπόν ρωτά ποια θα ήταν η αναμενόμενη ορθολογικά βέλτιστη επιλογή για τον καθένα;

Αρχικά, με τον όρο «ορθολογικά» εννοείται η πιο συμφέρουσα λύση για το κάθε άτομο, η επιλογή, δηλαδή, με το μεγαλύτερο δυνατό όφελος και την μικρότερη δυνατή απώλεια.

Έτσι λοιπόν το κάθε άτομο, επειδή ως «λογικά σκεπτόμενο» δεν μπορεί να εμπιστευτεί ότι το άλλο άτομο δεν θα μιλήσει, επιλέγει να μιλήσει ο ίδιος. Έτσι σύμφωνα με την θεωρία παιγνίων η κατάληξη του εν λόγω προβλήματος είναι να τιμωρηθούν με 4 χρόνια έκαστος.

Ίσως να περίμενε κανείς πως οι κατηγορούμενοι θα επέλεγαν την λύση της επιλογής και θα τιμωρούνταν μόνο με 1 έτος φυλάκισης ο καθένας. Όμως σκοπός αυτού του προβλήματος, όπως αναφέρεται παραπάνω, είναι η ανάλυση των στρατηγικών επιλογών. Έτσι το αποτέλεσμα του ανταγωνισμού επικρατεί του αποτελέσματος της συνεργασίας.

# Παράρτημα Β

## Οι επιλογές της κάμερας

Στον παρακάτω πίνακα μπορείτε να δείτε αναλυτικά όλες τις επιλογές της κάμερας.

| Ιδιότητα | Επεξήγηση |
|---|---|
| **Clear Flags** | Ορίζει ποια σημεία της εικόνας δεν θα απεικονιστούν. Ιδιαίτερα χρήσιμο στην χρήση πολλαπλών καμερών |
| **Background** | Το χρώμα που θα εμφανιστεί αφού απεικονιστούν όλα τα στοιχεία της σκηνής. |
| **Culling Mask** | Ορίζει ποια layers θα απεικονιστούν. |
| **Projection** | Αλλαγή του τύπου προβολής |
| **Perspective** | Προοπτική προβολή |
| **Orthographic** | Ορθογραφική προβολή |
| **Size (when Orthographic is selected)** | Το μέγεθος της προβολής (με επιλεγμένη την ορθογραφική προβολή). |
| **Field of view (when Perspective is selected)** | Το πλάτος της γωνίας θέασης της κάμερας |
| **Clipping Planes** | Αποστάσεις από την αρχή και το τέλος του Rendering |

| | |
|---|---|
| **Near** | Το κοντινότερο σημείο ως προς της κάμερα απ' όπου θα αρχίσουν να δημιουργούνται τα αντικείμενα. |
| **Far** | Το πιο μακρινό σημείο ως προς της κάμερα απ' όπου θα αρχίσουν να δημιουργούνται τα αντικείμενα. |
| **Normalized View Port Rect** | Ορίζει το σημείο στην οθόνη που θα δημιουργηθεί η κάμερα |
| **X** | Το αρχικό σημείο στον οριζόντιο άξονα που θα δημιουργηθεί η κάμερα. |
| **Y** | Το αρχικό σημείο στον κάθετο άξονα που θα δημιουργηθεί η κάμερα. |
| **W (Width)** | Το πλάτος της κάμερας |
| **H (Height)** | Το ύψος της κάμερας.. |
| **Depth** | Η θέση της κάμερας στην σειρά δημιουργίας |
| **Rendering Path** | Επιλογές για τη μέθοδο Render της κάμερας. |
| **Use Player Settings** | Η κάμερα θα χρησιμοποιήσει την μέθοδο που ορίζεται από τα Player Settings |
| **Vertex Lit** | Όλα τα αντικείμενα θα γίνουν render σαν Vertex-Lit αντικείμενα. |
| **Forward** | Όλα τα αντικείμενα θα γίνουν render με ένα πέρασμα ανά υλικό |

| | |
|---|---|
| **Deferred Lighting** | Όλα τα αντικείμενα θα γίνουν render χωρίς φωτισμό αρχικά και στην συνέχεια ο φωτισμός όλων των αντικειμένων θα γίνει render στο τέλος |
| **Target Texture** | Αναφορά σε ένα Render Texture που θα περιέχει την προβολή της κάμερας. Η χρήση του απενεργοποιεί την δυνατότητα της κάμερας να κάνει render την σκηνή |
| **HDR** | Ενεργοποιεί το High Dynamic Range rendering για την κάμερα. |

**Πίνακας Β.1 – Οι επιλογές της κάμερας**

# Αναφορές

Finnegan T. (2013) *Unity Game Development by Example,* Packt Publishing, Available at: https://www.packtpub.com (Accessed: 2015)

Aung Sithu Kyaw, Peters C., Thet Naing Swe (2013) *Unity 4.x Game AI Programming,* Packt Publishing, Available at: https://www.packtpub.com (Accessed: 2015)

Calabrese D. (2014) *Unity 2D Game Development*, Packt Publishing Available at: https://www.packtpub.com/game-development/unity-2d-game-development (Accessed: 2015)

Unity, *Unity Manual*, Available at: https://docs.unity3d.com/Manual/index.html (Accessed: 2015)