ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΛΟΠΟΝΝΗΣΟΥ

ΣΧΟΛΗ ΜΗΧΑΝΙΚΩΝ

ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ

ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

«Σχεδιασμός και ανάπτυξη διαδραστικού παιχνιδιού για Nonograms με χρήση του HTML5 Canvas»

Μπογιατζάκης Κωνσταντίνος, ΑΜ: 14886

Φοιτητής Τμήματος Μηχανικών Πληροφορικης Τ.Ε

ΠΑΤΡΑ ΑΠΡΙΛΙΟΣ 2022

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή

Πάτρα, 19/4/2022

ΕΠΙΤΡΟΠΗ ΑΞΙΟΛΟΓΗΣΗΣ

1. Ονοματεπώνυμο, Υπογραφή

2. Ονοματεπώνυμο, Υπογραφή

3. Ονοματεπώνυμο, Υπογραφή

Υπεύθυνη Δήλωση Φοιτητή

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. Επίσης βεβαιώνω ότι αυτή η εργασία προετοιμάστηκε από εμένα προσωπικά ειδικά για τη συγκεκριμένη εργασία. Η έγκριση της διπλωματικής εργασίας από το Τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών του Πανεπιστημίου Πελοποννήσου δεν υποδηλώνει απαραιτήτως και αποδοχή των απόψεων του συγγραφέα εκ μέρους του Τμήματος. Η παρούσα εργασία αποτελεί πνευματική ιδιοκτησία του φοιτητη **Μπογιατζάκη Κωνσταντίνου** που την εκπόνησε. Στο πλαίσιο της πολιτικής ανοικτής πρόσβασης ο συγγραφέας/δημιουργός εκχωρεί στο Πανεπιστήμιο Πελοποννήσου, μη αποκλειστική άδεια χρήσης του δικαιώματος αναπαραγωγής, προσαρμογής, δημόσιου δανεισμού, παρουσίασης στο κοινό και ψηφιακής διάχυσής τους διεθνώς, σε ηλεκτρονική μορφή και σε οποιοδήποτε μέσο, για διδακτικούς και ερευνητικούς σκοπούς, άνευ ανταλλάγματος και για όλο το χρόνο διάρκειας των δικαιωμάτων πνευματικής ιδιοκτησίας. Η ανοικτή πρόσβαση στο πλήρες κείμενο για μελέτη και ανάγνωση δεν σημαίνει καθ' οιονδήποτε τρόπο παραχώρηση δικαιωμάτων διανοητικής ιδιοκτησίας του συγγραφέα/δημιουργού ούτε επιτρέπει την αναπαραγωγή, αναδημοσίευση, αντιγραφή, αποθήκευση, πώληση, εμπορική χρήση, μετάδοση, διανομή, έκδοση, εκτέλεση, «μεταφόρτωση» (downloading), «ανάρτηση» (uploading), μετάφραση, τροποποίηση με οποιονδήποτε τρόπο, τμηματικά ή περιληπτικά της εργασίας, χωρίς τη ρητή προηγούμενη έγγραφη συναίνεση του συγγραφέα/δημιουργού. Ο συγγραφέας/δημιουργός διατηρεί το σύνολο των ηθικών και περιουσιακών του δικαιωμάτων.

Περιεχόμενα

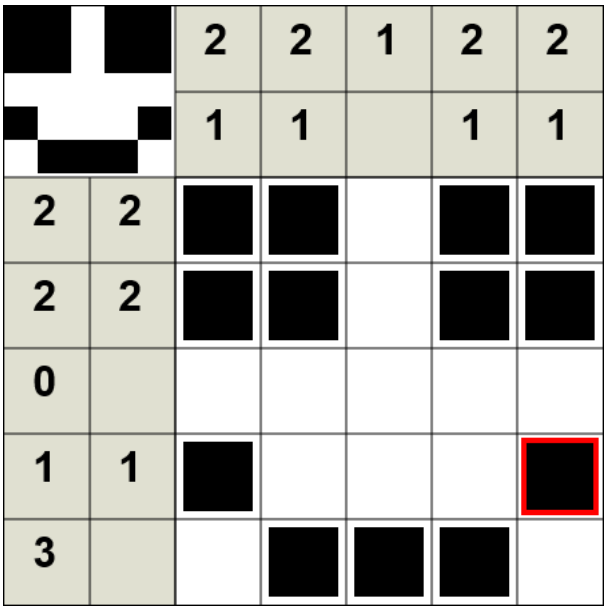
1. Εισαγωγή

Το θέμα της παρούσας πτυχιακής είναι «η δημιουργία του παιχνιδιού Nonogram με την χρήση του canvas api της html5». Το παιχνίδι θα το υλοποιήσουμε με javascript. Οι τεχνολογίες που θα χρησιμοποιήσουμε είναι html5, css3, javascript ecma 6, jquery, node.js, express και socket.io.

1.1 Τι είναι το Nonogram;

Το nonogram (στα ελληνικά εικονόσταυρα, άλλες ονομασίες picross , griddlers) είναι ένα puzzle game εικόνων στο οποίο ο παίκτης πρέπει να συμπληρώσει με χρωματιστά κελιά ένα πλέγμα κελιών βασιζόμενος στους αριθμούς που βρίσκονται αριστερά από το πλέγμα , που μας δείχνουν τον αριθμό των κελιών που πρέπει να χρωματιστούν για κάθε γραμμή και στους αριθμούς που βρίσκονται πάνω από το πλέγμα, που μας δείχνουν τον αριθμό των κελιών που πρέπει να χρωματιστούν για κάθε στήλη.

Πίστα nonogram smile



1.2 Ιστορία του Nononogram

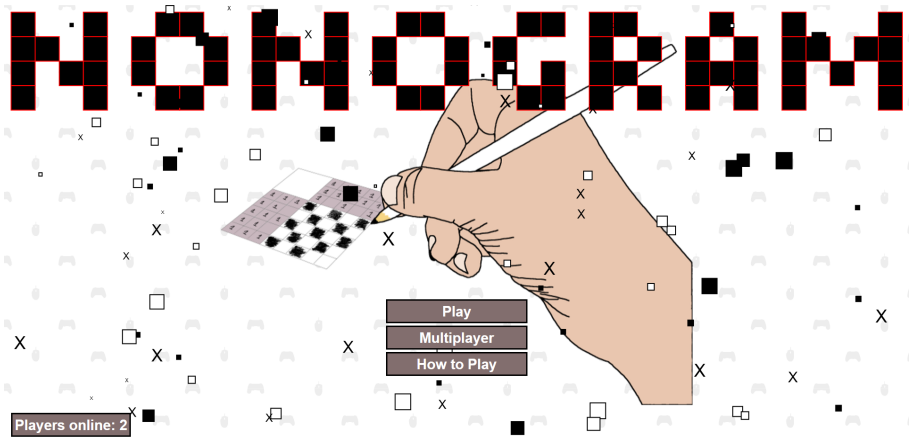
Το 1987, η Non Ishida, μία Γιαπωνέζα graphics editor, κέρδισε στο Τόκυο έναν διαγωνισμό στον οποίο σχεδίασε εικόνες πλέγματος χρησιμοποιώντας είτε ανοιχτά ή κλειστά τα φώτα από ουρανοξύστες. Αυτό την οδήγησε στην ιδέα για ένα puzzle βασισμένο στην ιδέα του να γεμίζεις ορισμένα τετράγωνα σε ένα πλέγμα. Τυχαία, ένας Γιαπωνέζος επαγγελματίας δημιουργός puzzle ο Tetsuya Nishio εφηύρε τα ίδια puzzle εντελώς ανεξάρτητα και τα δημοσίευσε σε ένα διαφορετικό περιοδικό.

https://en.wikipedia.org/wiki/Nonogram#History

1.3 Τι θα φτιάξουμε

Το παιχνίδι που θα φτιάξουμε θα αποτελείται από μία ιστοσελίδα στην οποία θα έχουμε ένα κεντρικό μενού στο οποίο θα επιλέγει ο χρήστης αν θέλει να παίξει μόνος του τις πίστες που έχουμε ή να παίξει μαζί με έναν άλλο χρήστη κάποιες πίστες και να δει τους κανόνες του παιχνιδιού. Θα έχει την εξής μορφή:

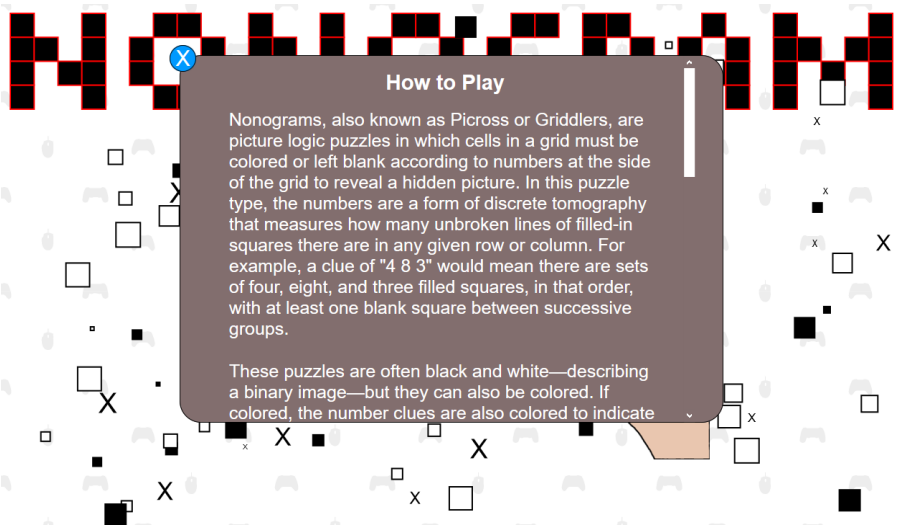
Η κεντρική σελίδα που θα έχει 3 βασικές επιλογές.



Το single-player που θα έχει 3 επίπεδα δυσκολίας και πίστες ανά επίπεδο.

To multiplayer

To howtoplay



1.4 Δημιουργία βασικών αρχείων και server

Το παιχνίδι μας θα αποτελείται από δύο βασικούς φακέλους: των **server** και των **client**.

Ο φάκελος του server θα περιέχει τα αρχεία που χρειαζόμαστε προφανώς για των server μας και για το multiplayer κομμάτι του παιχνιδιού και ο φάκελος client θα περιέχει όλο το υπόλοιπο παιχνίδι.

Θα χρησιμοποιήσουμε node.js για το back-end που σημαίνει ότι ο κώδικας που θα γράψουμε θα είναι σε javascript. Η υλοποίηση του server που θα φτιάξουμε θα γίνει με express.js και socket.io.

Ξεκινάμε κάνοντας εγκατάσταση το node.js από το επίσημο site. Ανάλογα με το λογισμικό που έχουμε κάνουμε την αντίστοιχη εγκατάσταση.

https://nodejs.org/en/download/

Ελέγχουμε ότι έχουμε το nodes.js γράφοντας στο **terminal**,

node -v

αυτή η εντολή μας δείχνει ποια έκδοση του node έχουμε εγκαταστήσει. Η έκδοση του node που χρησιμοποίησα είναι η 8.

Δημιουργούμε τον φάκελο μας με όνομα Nonogram Game ο οποίος θα περιέχει 2 φακέλους τον server και τον client.

- **Nonogram Game**
    - **server**

- **client**

Στην συνέχεια μέσα στον φάκελο server δημιουργούμε το node project μας με την εντολή

npm init -y

η παράμετρος -y που έχουμε βάλει μας επιτρέπει να δημιουργήσουμε το project χωρίς να χρειαστεί να δηλώσουμε κάποια βασικά στοιχεία όπως όνομα. Στην συνέχεια εγκαθιστούμε τα modules του express.js και του socket.io και είμαστε έτοιμοι να φτιάξουμε των server μας.

npm install express socket.io

Πληροφορίες για το express https://expressjs.com/ και για το socket.io https://socket.io/ .

Θα ξεκινήσουμε από των φάκελο server δημιουργώντας τον server μας.

- **server**
  - **server.js**

Οι εκδόσεις του node.js και των modules express και socket.io που χρησιμοποιήσαμε στην εργασία είναι οι: node.js 8.11.1, express 4.17.3 και socket.io 2.3.0.

Προσθέτουμε των κώδικα που χρειαζόμαστε για τον server, προσθέτουμε τα modules μας, δημιουργούμε το app που θα διαχειρίζεται τον server μέσο του express, δίνουμε το path του αρχείου που θα εμφανίζεται σαν αρχική σελίδα και τέλος δημιουργούμε την σύνδεση του socket.io στον server μας.

```
//modules
const http = require('http');
const express = require('express');
const socketio = require('socket.io');
const app = express();
const clientPath = __dirname + '/../client';
console.log('Serving static from ' + clientPath);
app.use(express.static(clientPath));
const server = http.createServer(app);
const io = socketio(server);

//server connection
```

```
io.on('connection', (sock) => {
    //when a user connects to the server
    console.log('Someone connected');
    sock.emit('message', 'Hi you are connected');

    sock.on('disconnect', () => {
            console.log('user disconnect');
    });
});

server.on('error', (err) => {
    console.log('Server error : ' + err);
});


server.listen(8080, () => {
    console.log('Nonogram game started on 8080');
});
```

Θα προσθέσουμε κι άλλα αρχεία και κώδικα στον φάκελο server για το multiplayer κομμάτι όταν φτάσουμε στο κεφάλαιο του multiplayer.

Ο server μας είναι έτοιμος, για να τον ξεκινήσουμε γράφουμε στο terminal την εντολή για να τρέξει ο server και η σελίδα μας είναι η **localhost:8080**.

npm run start

Στην συνέχεια θα ξεκινήσουμε να φτιάχνουμε των κώδικα για το singleplayer.

2.Δημιουργία βασικών αρχείων του παιχνιδιού

Το παιχνίδι θα παίζεται από ένα άτομο (singleplayer) αλλά και με δύο άτομα (multiplayer, co-op).Θα ξεκινήσουμε φτιάχνοντας τα αρχεία για το singleplayer.

Η δομή των αρχείων που θα χρησιμοποιήσουμε για το singleplayer κομμάτι είναι οι εξής. Στην συνέχεια θα προσθέσουμε τα υπόλοιπα αρχεία.

- **client**

index.hmtl

- **js**
  - **jquery-3.1.1..js**
  - **nonogram.js**
  - **nonogram.drawing.js**
  -
- **css**
  - **home.css**
- **img**

Όλες οι εικόνες που θα χρησιμοποιήσουμε στο παιχνίδι μας βρίσκονται εδώ

https://github.com/DinosMpo/Nonogram-Complete-Steps/tree/main/client/img

Θα χρησιμοποιήσουμε jquery για να μας βοηθήσει να γράψουμε λιγότερο κώδικα και να κάνουμε την δουλειά μας πιο εύκολη. Κατεβάζουμε την jquery από εδώ https://jquery.com/download/ και τοποθετούμε το script tag στο **index.html**.

Στο αρχείο **index.html** θα έχουμε για αρχή μόνο τον canvas μας για να δούμε πως θα είναι το παιχνίδι και στην συνέχεια θα προσθέσουμε κι άλλες λειτουργίες όπως το μενού, τις πίστες, τις οδηγίες του παιχνιδιού και τα εργαλεία για το παιχνίδι.

```
<!DOCTYPE html>
<html>
<head>
   <meta charset="utf-8">
   <meta name="viewport" content="width=device-width, initial-scale=1">
   <title>Nonogram Game</title>
   <link rel="stylesheet" type="text/css" href="css/home.css">
</head>
<body>
   <canvas id="canvas"></canvas>
```

```
    <script src="js/jquery-3.1.1.js"></script>
    <script src="js/nonogram.js"></script>
    <script src="js/nonogram.drawing.js"></script>
</body>
</html>
```

2.1 Δημιουργία Nonogram Object

Στο αρχείο **nonogram.js** θα έχουμε το object Nonogram στο οποίο θα βασίζεται το παιχνίδι μας και στο **nonogram.drawing.js** θα έχουμε τις συναρτήσεις που θα ζωγραφίζουν στον canva μας ανάλογα με το πλέγμα της πίστας (grid) που έχουμε και με το τι έχει επιλέξει ο χρήστης.

```
function Nonogram(levelGrid) { … };
```

Έχουμε σαν παράμετρο το levelGrid που είναι η πίστα του nonogram και σε αυτό βασίζεται το μεγαλύτερο μέρος του παιχνιδιού. Το grid είναι ένας δισδιάστατος πίνακας στον οποίο κάθε πίνακας αντιστοιχεί σε μία γραμμή και κάθε στοιχείο μέσα στον πίνακα είναι μία στήλη

π.χ [[1,0,1,0,1], [0,1,0,1,0], [1,0,1,0,1], [0,1,0,1,0], [1,0,1,0,1]]

Κάθε 1 αντιστοιχεί στα χρωματισμένα κουτάκια τις πίστας (στην συγκεκριμένη περίπτωση το χρώμα είναι μαύρο) και κάθε 0 μας δείχνει ότι το κουτάκι πρέπει να παραμείνει κενό.

Φτιάχνουμε μία μεταβλητή για να αποθηκεύσουμε το grid, θα την ονομάσουμε

```
this.levelGrid = levelGrid;
```

1. Για αρχή πρέπει να βρούμε το μέγεθος του παραθύρου για να ξέρουμε πόσο χώρο έχουμε για να ζωγραφίσουμε το παιχνίδι μας. Για να το κάνουμε αυτό παίρνουμε από το window object τις ιδιότητες innerWidth και innerHeight.

```
let windowWidth = window.innerWidth;
let windowHeight = window.innerHeight;
```

Πρέπει να βρούμε ποιο από τα 2 (windowWidth ή windowHeight) είναι μικρότερο για να μπορέσει να χωρέσει το παιχνίδι μας στην οθόνη. Θα χρειαστούμε μία μεταβλητή για να αποθηκεύσουμε το μέγεθος του

παραθύρου. Στην περίπτωση που το πλάτος είναι μεγαλύτερο από το ύψος τότε αφήνουμε ένα κενό στο κάτω μέρος αφαιρώντας -50 από το size. Αυτό το κάνουμε για να μπορέσουμε να χωρέσουν στην συνέχεια τα εργαλεία του παιχνιδιού.

```
let size;

if(windowWidth > windowHeight) {
        size = windowHeight - 50;
}else{
        size = windowWidth;
}
```

Στην συνέχεια πρέπει να βρούμε πόσες γραμμές και στήλες έχει η πίστα μας για το πλέγμα μας (το grid) και πόσα είναι τα χρωματιστά κουτάκια.

2. Φτιάχνουμε μία μεταβλητή την rowNumbers η οποία θα είναι ένας πίνακας από πίνακες και θα περιέχει τους αριθμούς που αντιπροσωπεύουν τα κουτάκια που περιέχει κάθε γραμμή.

```
this.rowNumbers = [];
for(let i=0;i<this.levelGrid.length;i++) {
        this.rowNumbers[i] = []; //ένας πίνακας για κάθε γραμμή
        this.rowNumbers[i][0] = 0; //δίνουμε μία αρχική τιμή στον πίνακα
}
```

Για να πάρουμε τους αριθμούς του πλέγματος για κάθε γραμμή πρέπει να περάσουμε από κάθε πίνακα (ανά γραμμή) και να ελέγξουμε κάθε στοιχείο του πίνακα (κάθε στήλη).

```
for(let row = 0; row < this.levelGrid.length; row ++) { //Για κάθε γραμμή
  let counter = 0; //μετράει πόσοι 1 είναι συνεχόμενοι
  let depth = 0; // μετράει το βάθος των συνεχόμενων αριθμών
  for(let column = 0; column < this.levelGrid[row].length; column++) {
    if(this.levelGrid[row][column] == 1) {
        counter += 1; //αυξάνεται κατά 1 όταν υπάρχει 1
        this.rowNumbers[row][depth] = counter;//Αποθηκεύουμε το βάθος
    }else{
```

```
    if(counter != 0) {
            this.rowNumbers[row][depth] = counter;
            counter = 0; //μηδενίζουμε τον counter
            depth++; // και το βάθος μεγαλώνει
    }
   }
  }
}
```

Παράδειγμα: Άμα είχαμε αυτό το grid τότε η μεταβλητή rowNumbers θα είχε αυτή την μορφή:

grid: [[1,1,1,1,1],[1,0,0,0,0],[1,1,1,1,1],[0,0,0,0,1],[1,1,1,1,1]]

rowNumbers [[5],[1],[5],[1],[5]].

Παρόμοια υλοποίηση κάνουμε και για τους αριθμούς για κάθε στήλη με την μόνη διαφορά να είναι ότι παίρνουμε κάθε στήλη και ελέγχουμε κάθε γραμμή.

```
this.columnNumbers = [];
for(let i=0;i<this.levelGrid[0].length;i++) {
  this.columnNumbers[i] = [];
  this.columnNumbers[i][0] = 0;
}
for(let column=0; column <this.levelGrid[0].length; column ++) {
  let counter = 0; //μετράει πόσοι 1 είναι συνεχόμενοι
  let depth = 0; // μετράει το βάθος των συνεχόμενων αριθμών
  for(let row =0; row <this.levelGrid.length; row++) {
   if(this.levelGrid[row][column]==1) {
    counter += 1; //αυξάνεται κατά 1 όταν υπάρχει 1
    this.columnNumbers[column][depth] = counter;//αποθηκεύουμε το counter
   }else{
    if(counter != 0) {
            this.columnNumbers[column][depth]= counter;
            counter = 0; //μηδενίζουμε τον counter
            depth++; // και το βάθος μεγαλώνει
          }
   }
```

```
  }
}
```

Έχουμε βρεί τους αριθμούς που είναι για κάθε γραμμή και στήλη και πρέπει να βρούμε τον χώρο που θα πιάνουν οι αριθμοί πάνω στον canva. Για να το κάνουμε αυτό πρέπει να βρουμε την γραμμή και την στήλη με το μεγαλύτερο μήκος. Φτιάχνουμε 2 μεταβλητές στις οποίες θα αποθηκεύσουμε τον μεγαλύτερο αριθμό αριθμών που υπάρχει σε κάθε γραμμή και στήλη.

```
this.maxRowNumberSize = 0;
this.maxColumnNumberSize = 0;
for (let i =0; i < this.rowNumbers.length; i ++) {
  if (this.maxRowNumberSize < this.rowNumbers[i].length) {
    this.maxRowNumberSize = this.rowNumbers[i].length;
  }
}

for (let i =0; i < this.columnNumbers.length; i ++) {

  if(this.maxColumnNumberSize < this.columnNumbers[i].length) {
    this.maxColumnNumberSize = this.columnNumbers[i].length;
  }
}
```

2.2 Το μέγεθος των κελιών του Nonogram

Για να βρούμε το μέγεθος των κελιών πρέπει να βρούμε τον μέγιστο αριθμό των κελιών που έχουμε σε μία γραμμή ή στήλη και να τον διαιρέσουμε με το μέγεθος του παραθύρου μας. Φτιάχνουμε μία μεταβλητή την maxSize στην οποία θα αποθηκεύεται ο μέγιστος αριθμός των κελιών που υπάρχει σε μια γραμμή ή σε μια στήλη. Για να βρούμε το maxSize συγκρίνουμε το maxRowNumberSize με το maxColumnNumberSize και όποιο είναι πιο μεγάλο το προσθέτουμε με το πλήθος των κελιών του grid μας δηλαδή με τον αριθμό των κελιών κάθε γραμμής ή στήλης (levelGrid.length). Στις πίστες που θα φτιάξουμε οι γραμμές και οι στήλες έχουν πάντα το ίδιο μήκος (5x5, 10x10, 15x15) και γι αυτό δεν χρειάζεται να βρούμε αν οι γραμμές ή οι στήλες είναι μεγαλύτερες.

```
let maxSize;
```

```
if(this.maxRowNumberSize > this.maxColumnNumberSize) {
        maxSize = this.maxRowNumberSize + this.levelGrid.length;
}else{
        maxSize = this.maxColumnNumberSize + this.levelGrid.length;
}
```

Στο object Nonogram δημιουργούμε μία ιδιότητα την **blockSize** για να αποθηκεύσουμε το μέγεθος που θα έχουν τα κελιά μας.

```
this.blockSize = 0;
this.blockSize = Math.floor((size / maxSize) - 1);
```

Για να έχουμε ακέραιο αποτέλεσμα χρησιμοποιούμε το Math.floor για να στρογγυλοποιήσουμε προς τα κάτω και το -1 το αφαιρούμε για την συνάρτηση drawGrid με την οποία θα ζωγραφίζεται το πλέγμα της πίστας μας που θα εξηγήσω στην συνέχεια.

2.3 Το μέγεθος του Nonogram

Με όσα έχουμε βρεί ως τώρα μπορούμε να βγάλουμε το πλάτος και το ύψος του Nonogram μας και κατά συνέπεια του canvas στον οποίο θα ζωγραφίζουμε. Στο object Nonogram δημιουργούμε 2 ιδιότητες.

```
this.width = 0;
this.height = 0;
```

Για να βρούμε το πλάτος προσθέτουμε το πλήθος των στηλών μιας γραμμής (κάθε γραμμή έχει τον ίδιο αριθμό στηλών) με τον μέγιστο αριθμό των κελιών που χρειαζόμαστε για να δείξουμε τους αριθμούς κάθε γραμμής και αυτό το αποτέλεσμα το πολλαπλασιάζουμε με το μέγεθος του κελιού. Ίδια λογική ακολουθούμε και για το ύψος με μόνη διαφορά ότι αντί για το πλήθος των στηλών μιας γραμμής χρησιμοποιούμε το πλήθος των γραμμών

```
this.width = (this.levelGrid[0].length + this.maxRowNumberSize) * this.blockSize;
this.height = (this.levelGrid.length + this.maxColumnNumberSize) * this.blockSize;
```

Στο επόμενο κεφάλαιο θα ξεκινήσουμε να χειριζόμαστε τον canva και να ζωγραφίζουμε σε αυτόν το nonogram μας.

3.Αναπαράσταση του παιχνιδιού στον canvas

Για να διαχειριστούμε τον canva πρέπει να πάρουμε το object του canvas μας. Φτιάχνουμε ένα καινούργιο αρχείο μέσα στον φάκελο js με όνομα **home.js** και το τοποθετούμε στο index.html μετά το script του nonogram.js. Δημιουργούμε μία μεταβλητή με το όνομα canvas στην οποία θα έχουμε τον canvas μας τον οποίο τον βρίσκουμε με το id 'canvas' που του έχουμε δώσει. Για να μπορέσουμε να ζωγραφίσουμε στον canva πρέπει να πάρουμε το περιεχόμενο του, εμεις θα χρησιμοποιήσουμε το 2d περιεχόμενο του.

```
"use strict"; // το use strict μας επιτρέπει να μην μπορούμε να χρησιμοποιούμε μεταβλητές που δεν έχουμε δηλώσει
//παίρνουμε το object του canvas
let canvas = document.getElementById("canvas");
//παίρνουμε το context του canvas που μας επιτρέπει να ζωγραφίσουμε στον canva
let ctx = canvas.getContext("2d");
```

Τώρα μπορούμε να χρησιμοποιήσουμε το object Nonogram και να ξεκινήσουμε να φτιάχνουμε μία πίστα.

```
let nonogram = new Nonogram( [      [1,0,1,0,1],
                                    [0,1,0,1,0],
                                    [1,0,1,0,1],
                                    [0,1,0,1,0],
                                    [1,0,1,0,1]     ] );
```

Δίνουμε στον canvas μας το πλάτος και το ύψος της πίστας μας

```
canvas.width = nonogram.width;
canvas.height = nonogram.height;
```

Όπως είδαμε πριν το Nonogram object έχει κάποιες μεταβλητές όπως το **rowNumbers**, **columnNumbers**, **blockSize**, **width** και **height**. Θα κάνουμε έναν έλεγχο για να δούμε ότι έχουν σωστές τιμές αυτές οι μεταβλητές, κάνοντας console.log τις μεταβλητές.

```
console.log("rowNumbers: " + nonogram.rowNumbers);
console.log("columnNumbers: "+ nonogram.columnNumbers);
console.log("blockSize: " + nonogram.blockSize);
console.log("width: " + nonogram.width);
```

```
console.log("height: " + nonogram.height);
```

Έχουμε φτιάξει την πίστα και τώρα μπορούμε να την ζωγραφίσουμε.

3.1 Ζωγραφίζοντας την πίστα μας στον canvas

Στο αρχείο **nonogram.drawing.js** θα έχουμε των κώδικα που χρειαζόμαστε για να ζωγραφίσουμε στον canvas την πίστα μας.

Θα ξεκινήσουμε ζωγραφίζοντας το grid μας χωρίς τους αριθμούς. Φτιάχνουμε μία μέθοδο με το όνομα drawGrid.

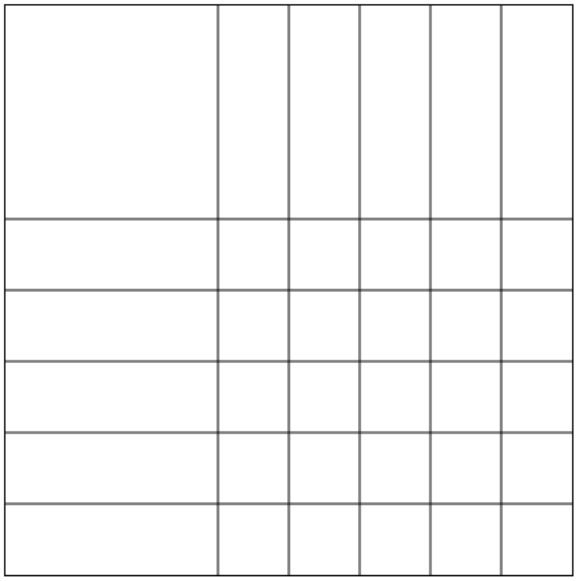
```
Nonogram.prototype.drawGrid = function() {
}
```

1. Το πρώτο πράγμα που θα κάνουμε είναι να φτιάξουμε ένα άσπρο background για την πίστα μας

```
ctx.fillStyle = "white";
ctx.fillRect(0, 0, this.width, this.height);
```

2. Ορίζουμε το μέγεθος της γραμμής και φτιάχνουμε τις γραμμές του πλέγματος για κάθε στήλη και για κάθε γραμμή.

```
ctx.strokeStyle = "black";
ctx.beginPath();
ctx.lineWidth = 1; //Το μέγεθος της γραμμής
for(var i = (this.maxColumnNumberSize ) * this.blockSize; i < this.height; i += this.blockSize ) {
  ctx.moveTo(0,i);
  ctx.lineTo(this.width,i);
}
for( var y = (this.maxRowNumberSize ) * this.blockSize; y < this.width; y += this.blockSize ) {
  ctx.moveTo(y,0);
  ctx.lineTo(y, this.height);
}
```

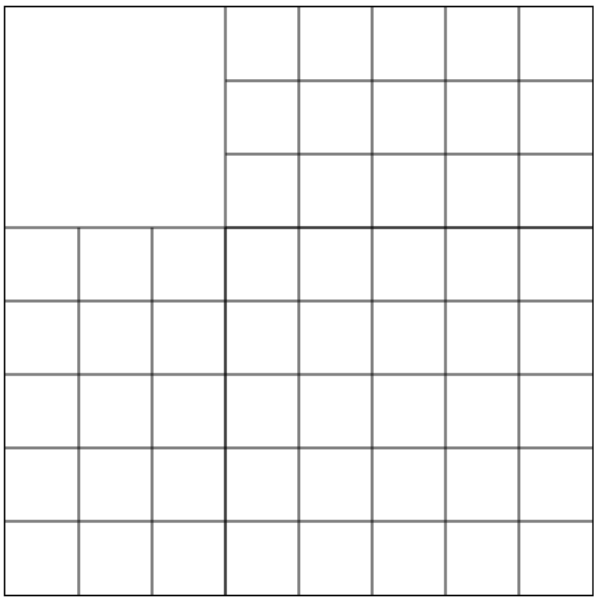


3. Φτιάχνουμε τις γραμμές για τους αριθμούς που αντιπροσωπεύουν τα κελιά που πρέπει να χρωματιστούν για κάθε γραμμή και στήλη.

```
for ( let i = 0; i < this.maxColumnNumberSize; i++ ) {
  ctx.moveTo((this.maxRowNumberSize) * this.blockSize,(i+1)*this.blockSize);
  ctx.lineTo(this.width, (i+1)*this.blockSize);
}

for ( let i = 0; i < this.maxRowNumberSize; i++ ) {
  ctx.moveTo( (i+1)*this.blockSize, (this.maxColumnNumberSize) * this.blockSize);
  ctx.lineTo( (i+1)*this.blockSize, this.height);
}
```

Ζωγραφίζουμε και το περίγραμμα της πίστας μας και ορίσαμε όλα τα σημεία που θέλουμε να γίνουν οι γραμμές και στο τέλος τις ζωγραφίζουμε όλες μαζί.

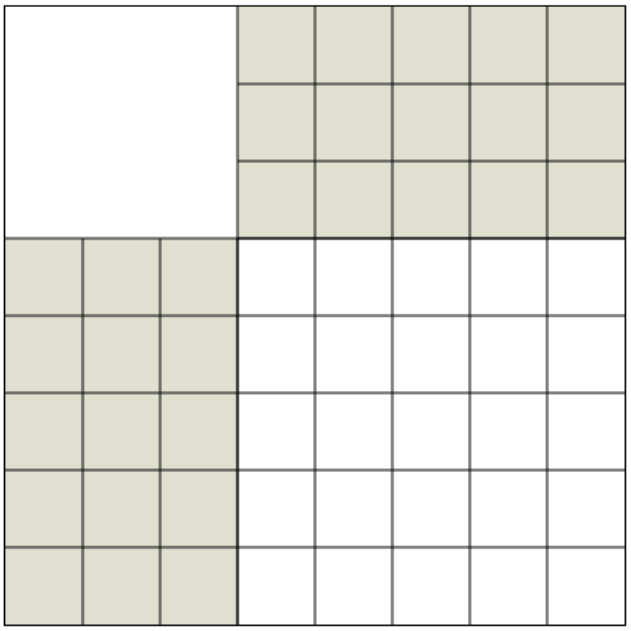
```
ctx.fillStyle = 'black';
ctx.lineWidth = 1;
ctx.strokeRect(0,0,this.width,this.height);
ctx.stroke();
ctx.closePath();
```



Στο αρχείο **home.js** καλούμε την συνάρτηση drawGrid μετά την μεταβλητή nonogram και έχουμε το εξής αποτέλεσμα, μας μένει μόνο να ζωγραφίσουμε ένα background για τα κελιά των αριθμών και να βάλουμε και τους αριθμούς.

4. Για το χρώμα των κελιών τοποθετούμε αυτόν των κώδικα ακριβώς μετά των κώδικα του βήματος 1. Αυτό το κάνουμε για να μην καλυφθούν οι γραμμές του πλέγματος. Ο canvas είναι σαν ένας πραγματικός canvas, ανάλογα με την σειρά που γράφουμε των κώδικα για να ζωγραφίσουμε, έτσι ζωγραφίζεται και στον canva το ένα πάνω στο άλλο.

```
ctx.fillStyle = "#e0e0d1";
ctx.fillRect(0, this.maxColumnNumberSize * this.blockSize, this.maxRowNumberSize *
this.blockSize, this.height-(this.maxColumnNumberSize * this.blockSize));
ctx.fillRect(this.maxRowNumberSize * this.blockSize, 0, this.width-(this.maxRowNumberSize *
this.blockSize), this.maxColumnNumberSize * this.blockSize);
ctx.fillStyle = "black";
```



5. Για τους αριθμούς δημιουργούμε στο object Nonogram μας, 2 μεταβλητές που θα είναι πίνακες, για τους αριθμούς κάθε γραμμής και κάθε στήλης αντίστοιχα.

```
this.rowNumbersGrid = [];
this.columnNumbersGrid = [];
```

Στην συνέχεια θα φτιάξουμε ένα object που θα αντιπροσωπεύει τους αριθμούς των κελιών. Αυτό το object θα κρατάει τις εξής πληροφορίες: πλάτος, ύψος, τις συντεταγμένες x και y του κελιού και τον αριθμό του κελιού.

Στο αρχείο **nonogram.js** φτιάχνουμε το object και τοποθετούμε των κώδικα στην αρχή του αρχείου.

```
function NumberCell(w, h, x, y, number) {
  this.w = w;
  this.h = h;
  this.x = x;
  this.y = y;
  this.number = number;
}
```

Τώρα μπορούμε να τελειώσουμε την υλοποίηση για τους αριθμούς κάθε γραμμής και στήλης. Στο αρχείο **nonogram.js** στο object Nonogram στο τέλος δημιουργούμε τα κελιά των αριθμών.

```
for (var i = 0; i < this.rowNumbers.length; i ++) {
  for ( var y = 0; y < this.rowNumbers[i].length; y ++) {
    this.rowNumbersGrid.push(new NumberCell(
      this.blockSize,
      this.blockSize,
      (y * this.blockSize),
      ((this.maxColumnNumberSize) * this.blockSize) + (i * this.blockSize),
      this.rowNumbers[i][y]));
  }
}
for (var i = 0; i < this.columnNumbers.length; i ++) {
 for ( var y = 0; y < this.columnNumbers[i].length; y ++) {
   this.columnNumbersGrid.push(new NumberCell(
     this.blockSize,
     this.blockSize,
     ((this.maxRowNumberSize) * this.blockSize) + (i * this.blockSize),
     (y * this.blockSize),
     this.columnNumbers[i][y]));

 }
}
```

1. Στο αρχείο **nonogram.drawing.js** δημιουργούμε 2 συναρτήσεις την **drawRowNumbers** και **drawColumnNumbers** για να ζωγραφίσουμε τους αριθμούς κάθε γραμμής και στήλης. Στην συνάρτηση **drawRowNumbers** περνάμε από κάθε γραμμή και στήλη και αποθηκεύουμε ένα object NumberCell που αντιπροσωπεύει έναν αριθμό στην μεταβλητή **rowNumbersGrid** και στην συνέχεια ζωγραφίζουμε τους αριθμούς κάθε γραμμής**.**

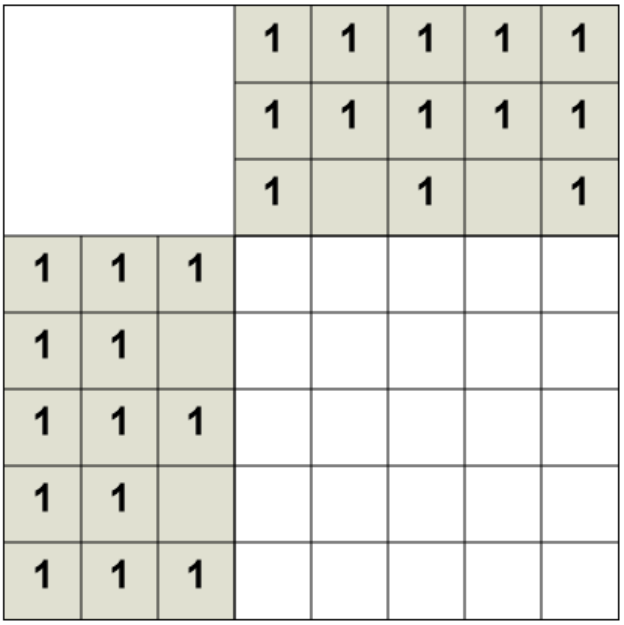
```
Nonogram.prototype.drawRowNumbers = function() {
   ctx.fillStyle = 'black';
```

```
    for (var i = 0; i < this.rowNumbersGrid.length; i ++) {
            ctx.font = "bold " + (this.blockSize / 2) + "px Arial";
            ctx.fillText( this.rowNumbersGrid[i].number,
      (this.rowNumbersGrid[i].x) + (this.blockSize/3),
      (this.rowNumbersGrid[i].y) + ((this.blockSize+8)/2));
    }
}
```

Παρόμοια υλοποίηση κάνουμε και για τους αριθμούς κάθε στήλης με μία μικρή διαφορά στις συντεταγμένες των κελιών.

```
Nonogram.prototype.drawColumnNumbers = function() {
    ctx.fillStyle = 'black';
    for (var i = 0; i < this.columnNumbersGrid.length; i ++) {
            ctx.font = "bold " + (this.blockSize / 2) + "px Arial";
            ctx.fillText(this.columnNumbersGrid[i].number,
      (this.columnNumbersGrid[i].x) + (this.blockSize/3),
      (this.columnNumbersGrid[i].y) + ((this.blockSize+8)/2));
    }
}
```

Καλούμε στο αρχείο **home.js** μετά την συνάρτηση drawGrid, τις συναρτήσεις drawRowNumbers και drawColumnNumbers. Τώρα βρισκόμαστε σε αυτήν την μορφή και έχουμε ζωγραφίσει την πίστα μας.



Στο επόμενο κεφάλαιο θα φτιάξουμε το object που θα αντιπροσωπεύει τα κελιά της πίστας μας.

3.2 Δημιουργία Cell object

Στο αρχείο **nonogram.js** φτιάχνουμε το object για τα κελιά της πίστας και τοποθετούμε των κώδικα στην αρχή του αρχείου.

```
function Cell(w, h, x, y, value) {
        this.w = w;
        this.h = h;
        this.x = x;
        this.y = y;
        this.value = value;
}
```

Η ιδιότητα **value** αντιπροσωπεύει την κατάσταση του κελιού π.χ αν είναι μαύρο.

Όλα τα κελιά θα τα αποθηκεύσουμε σε μία μεταβλητή που θα είναι πίνακας. Αυτήν την μεταβλητή θα την δημιουργήσουμε σαν ιδιότητα στο object Nonogram.

Στο αρχείο **nonogram.js** τοποθετούμε των κώδικα για να δημιουργήσουμε τα κελιά. Για να βρούμε τον χώρο στον οποίο ξεκινάνε τα κελιά της πίστας κάνουμε το εξής: Υπολογίζουμε από πού ξεκινάει η γραμμή και η στήλη των κελιών.

Για να βρούμε από που ξεκινάει η γραμμή πολλαπλασιάζουμε των μέγιστο αριθμό των κελιών για τις στήλες με το μέγεθος του κελιού, αυξάνουμε σε κάθε επανάληψη με το μέγεθος του κελιού και για να βρούμε από που ξεκινάει η στήλη πολλαπλασιάζουμε των μέγιστο αριθμό των κελιών για τις γραμμές με το μέγεθος του κελιού, αυξάνουμε σε κάθε επανάληψη με το μέγεθος του κελιού.

```
this.emptyGrid = [];
for(var i=this.maxColumnNumberSize*this.blockSize; i<this.height; i+=this.blockSize) {
  for(var y=this.maxRowNumberSize*this.blockSize; y<this.width;  y+=this.blockSize) {
    this.emptyGrid.push(new Cell(this.blockSize, this.blockSize, y, i, 0));
  }
}
```

Τώρα είμαστε έτοιμοι να φτιάξουμε τους χειρισμούς για την αλληλεπίδραση του χρήστη με την πίστα.

4.Χειρισμοί του παιχνιδιού

Στον φάκελο **js** φτιάχνουμε έναν καινούργιο φάκελο των contols και φτιάχνουμε το αρχείο **mouse.js** στο οποίο θα έχουμε τους χειρισμούς για την αλληλεπίδραση του χρήστη με ποντίκι και το αρχείο **touch.js** το οποίο θα περιέχει τους χειρισμούς για την αλληλεπίδραση του χρήστη με οθόνη αφής**.** Η υλοποίηση που θα κάνουμε δουλεύει ως εξής: όποτε ο χρήστης θα κάνει click με το ποντίκι πάνω στον canva, αποθηκεύουμε τις συντεταγμένες και ελέγχουμε σε ποιό κελί έχει πατήσει συγκρίνοντας τις συντεταγμένες του κελιού και του ποντικιού. Στο τελευταίο κεφάλαιο της εργασίας θα μιλήσουμε για τα touch controls τα οποία θα αφορούν συσκευές με οθόνη αφής.

Τοποθετούμε τα δύο scripts στο index.html στο τέλος.

```
<script src="js/controls/mouse.js"></script>
<script src="js/controls/touch.js"></script>
```

4.1 Mouse controls

Για τον χειρισμό του ποντικιού θα χρησιμοποιήσουμε το event **mousedown.** Αυτό το event αρχίζει να εκτελείται όταν ο χρήστης πατήσει το αριστερό click του ποντικιού.

```
$(canvas).mousedown(function(event) { ... });
```

Η παράμετρος event αντιπροσωπεύει την αλληλεπίδραση που κάνει ο χρήστης πάνω στον canva που σε αυτήν την περίπτωση είναι το mousedown..

Το πρώτο πράγμα που πρέπει να κάνουμε όταν ενεργοποιηθεί το event είναι να πάρουμε τις συντεταγμένες που πατήθηκε το αριστερό click.

Φτιάχνουμε 2 μεταβλητές: την **startPointMouseX** και **startPointMouseY**. Αυτές οι μεταβλητές θα έχουν τις συντεταγμένες του event. Το event είναι ένα object που έχει κάποιες ιδιότητες 2 από αυτές είναι η offsetX και offsetY. Οι ιδιότητες offsetX και offestY μας δείχνουν τις συντεταγμένες x και y του ποντικιού σε σχέση με το στοιχείο που βρίσκεται το ποντίκι στην σελίδα μας.

```
startPointMouseX = event.offsetX;
startPointMouseY = event.offsetY;
```

Αυτός είναι ο βασικός χειρισμός που θα χρειαστούμε τώρα για την αλληλεπίδραση του χρήστη με τον canva. Σε επόμενα κεφάλαια θα προσθεσουμε κι άλλα events για την αλληλεπίδραση του χρήστη, που θα χρειαστούμε για άλλες λειτουργίες.

4.2 Touch controls

Για τον χειρισμό σε οθόνη αφής δημιουργούμε ένα καινούργιο αρχείο στο φάκελο controls το **touch.js**. Προσθέτουμε το event touchstart το οποίο ενεργοποιείται όταν ο χρήστης πατήσει στην οθόνη.

```
$(canvas).on('touchstart', function(event) { ... };
```

Προσθέτουμε των κώδικα που θα μας δίνει τις συντεταγμένες που έγινε το touch.

```
startPointTouchX = Math.floor(event.touches[0].clientX - ((window.innerWidth - canvas.width) / 2));
startPointTouchY = Math.floor(event.touches[0].clientY - ((window.innerHeight - canvas.height) / 2));
```

Θα μιλήσουμε αναλυτικά για τα touch controls στο κεφάλαιο 13.

5. Ζωγραφίζοντας τα κελιά της πίστας

Τώρα πρέπει να φτιάξουμε την συνάρτηση η οποία θα ζωγραφίζει τα κελιά που επιλέγει ο χρήστης.

1. Στο αρχείο **nonogram.drawing.js** φτιάχνουμε μία συνάρτηση με όνομα **fillCells** η οποία θα περιέχει των κώδικα που χρειαζόμαστε για να ζωγραφίσουμε τα κελιά.

   Τα κελιά θέλουμε να ζωγραφίζονται είτε με μαύρο χρώμα, που σημαίνει ότι ο χρήστης πιστεύει ότι αυτό το κελί είναι σωστό, είτε με ένα X, που σημαίνει ότι ο χρήστης πιστεύει ότι αυτό το κελί δεν πρέπει να είναι μαύρο και είτε με άσπρο χρώμα, που σημαίνει ότι ο χρήστης δεν γνωρίζει αν το κελί αυτό πρέπει να είναι μαύρο ή όχι.

2. Ξεκινάμε φτιάχνοντας την συνάρτηση, στην οποία βάζουμε 2 παραμέτρους, οι οποίες θα αντιπροσωπεύουν τις συντεταγμένες του ποντικιού σε σχέση με τον canva.

```
Nonogram.prototype.fillCels = function(mouseX, mouseY) { ... };
```

3. Για να βρούμε ποιο κελί έχει πατηθεί, ελέγχουμε ένα ένα τα κελιά μας από τον πίνακα **emptyGrid** μέχρι να βρούμε το κελί το οποίο βρίσκεται στις συντεταγμένες που έγινε το click. Για να βρούμε το κελί κάνουμε 4 υπολογισμούς:

   1. αν η συντεταγμένη X του event είναι μεγαλύτερη ή ίση με την συντεταγμένη X του κελιου

   2. αν η συντεταγμένη Y του event είναι μεγαλύτερη ή ίση με την συντεταγμένη Y του κελιου

   3. αν η συντεταγμένη X του event είναι μικρότερη ή ίση με την συντεταγμένη X του κελιου συν το μέγεθος του κελιού

   4. αν η συντεταγμένη Y του event είναι μικρότερη ή ίση με την συντεταγμένη Y του κελιου συν το μέγεθος του κελιού

```
for(var i=0; i<this.emptyGrid.length; i++) {
    if(mouseX >= this.emptyGrid[i].x &&
       mouseY >= this.emptyGrid[i].y &&
       mouseX <= (this.emptyGrid[i].x + this.blockSize) &&
       mouseY <= (this.emptyGrid[i].y + this.blockSize)) {
           ...
    }
}
```

4. Εφόσον βρήκαμε το κελί που πατήθηκε, μπορούμε να αλλάξουμε την κατάσταση του κελιού και το χρώμα του. Την κατάσταση του κελιού την βλέπουμε από την ιδιότητα **value** που έχει το Cell object. Αν το value είναι 0 τότε το κελί πρέπει να είναι άσπρο, αν είναι 1 τότε πρέπει να είναι μαύρο και αν είναι 2 τότε σημαίνει ότι πρέπει να είναι X. Υπάρχει μία ιεραρχία με την κατάσταση του κελιού, άμα είναι 0 και το πατήσει ο χρήστης γίνεται 1, αν είναι 1 γίνεται 2 και αν είναι 2 γίνεται 0.

Για να κάνουμε των κώδικα μας ποιό ευανάγνωστο και για να μην γράφουμε τον ίδιο κώδικα και σε άλλες υλοποιήσεις που θα κάνουμε θα φτιάξουμε στο αρχείο **nonogram.drawing.js** 3 συναρτήσεις την **drawBlackCell**, **drawXCell** και **drawWhiteCell** που θα μας βοηθήσουν να ζωγραφίζουμε στον canva. Περνάμε σαν παράμετρο στις συναρτήσεις το κελί στο οποίο πρέπει να ζωγραφίσουμε. Επίσης φτιάχνουμε 3 μεταβλητές τις drawBlakCellValue, drawWhiteCellValue και drawXCellValue για τις 3 συναρτήσεις που θα έχουν έναν αριθμό που θα είναι σαν όριο που θα προστίθεται στις συντεταγμένες και θα αφαιρείται από το μέγεθος του κελιού. Αυτό το έχω κάνει για να μπορώ να ζωγραφίσω ένα μικρότερο κελί μέσα στο κελί και για να μην ακουμπάει πάνω στις γραμμές του πλέγματος.

```
//Ζωγραφίζει το κελί μαύρο
let drawBlackCellValue = 3;
Nonogram.prototype.drawBlackCell = function(cell) {
   ctx.fillStyle = 'black';
   ctx.fillRect(cell.x + drawBlackCellValue, cell.y + drawBlackCellValue,      cell.w -
(drawBlackCellValue * 2), cell.h - (drawBlackCellValue * 2));
};

//Ζωγραφίζει το κελί άσπρο
let drawWhiteCellValue = 2;
Nonogram.prototype.drawWhiteCell = function(cell) {
   ctx.fillStyle = "white";
   ctx.fillRect(cell.x + drawWhiteCellValue, cell.y + drawWhiteCellValue, cell.w -
(drawWhiteCellValue * 2), cell.h - (drawWhiteCellValue * 2));
};


//Ζωγραφίζει ένα X μέσα στο κελί
let drawXCellValue = 3;
```

```
Nonogram.prototype.drawXCell = function(cell) {
    ctx.strokeStyle = "black";

    ctx.lineWidth = 4;
    ctx.beginPath();
    ctx.moveTo(cell.x + drawXCellValue, cell.y + drawXCellValue);
    ctx.lineTo(cell.x + this.blockSize - drawXCellValue, cell.y + this.blockSize - drawXCellValue);
    ctx.moveTo(cell.x + this.blockSize - drawXCellValue, cell.y + drawXCellValue);
    ctx.lineTo(cell.x + drawXCellValue, cell.y + this.blockSize - drawXCellValue);
    ctx.stroke();
    ctx.closePath();
};
```

5. Μέσα στην if τοποθετούμε των κώδικα μας. Χρησιμοποιούμε τις συναρτήσεις που φτιάξαμε για να ζωγραφίσουμε στην πίστα μας τα κελιά αναλόγως με το τι θέλει να ζωγραφίσει ο χρήστης. Σε αυτόν των κώδικα θα προσθέσουμε στην συνέχεια κι άλλες λειτουργίες όπως τα εργαλεία επιλογής για το ζωγράφισμα των κελιών κ.α γι αυτό θα αναφερθούμε σε αυτόν πολλές φορές.

```
if(this.emptyGrid[i].value == 0) {
    this.emptyGrid[i].value = 1;
    //Καθαρίζουμε το κελί κάνοντας το άσπρο
    this.drawWhiteCell(this.emptyGrid[i]);
    //και το ζωγραφίζουμε μαύρο
    this.drawBlackCell(this.emptyGrid[i]);
}else if(this.emptyGrid[i].value == 1) {
    this.emptyGrid[i].value = 2;
    //Καθαρίζουμε το κελί κάνοντας το άσπρο
    this.drawWhiteCell(this.emptyGrid[i]);
    this.drawXCell(this.emptyGrid[i]);
}else {
    this.emptyGrid[i].value = 0;
    this.drawWhiteCell(this.emptyGrid[i]);
}
```

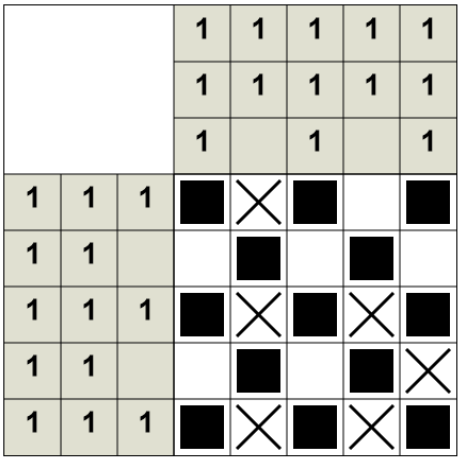
Τοποθετούμε την συνάρτηση στα controls που φτιάξαμε mousedown

```
nonogram.fillCels(startPointMouseX, startPointMouseY);
```

και touchstart

```
nonogram.fillCels(startPointTouchX, startPointTouchY);
```

Ως τώρα έχουμε φτιάξει την πιο απλή αλληλεπίδραση του χρήστη με το παιχνίδι, μπορεί να κάνει τα κελιά μαύρα, άσπρα και να τα ακυρώνει με ένα X.



5.1 Μαρκάρισμα κελιών πλέγματος

Για να διευκολύνουμε των χρήστη να ξέρει ποιό κελί έχει επιλέξει τελευταία φορά θα ζωγραφίσουμε ένα κόκκινο τετράγωνο μέσα στο κελί που έχει επιλέξει.

1. Για να το πετύχουμε αυτό φτιάχνουμε στο αρχείο **nonogram.js** στο object nonogram, δύο καινούργιες μεταβλητές που θα είναι object τις previousChoice και currentChoice.

```
this.currentChoice = {
  cell: []
};
this.previousChoice = {
  active: false,
  cell: []
};
```

2. Στην συνέχεια δημιουργούμε στο αρχείο **nonogram.drawing.js** την συνάρτηση **Nonogram.prototype.strokeCurrentChoice** η οποία παίρνει σαν παράμετρο την cell που

αντιπροσωπεύει το τελευταίο κελί που πατήθηκε. Στην συνάρτηση αυτή ελέγχουμε αν ο χρήστης είχε επιλέξει κάποιο κελί προηγουμένως έτσι ώστε να σβήσουμε το κόκκινο πλαίσιο που του είχαμε ζωγραφίσει και να ζωγραφίσουμε το κόκκινο πλαίσιο στο τωρινό κελί που έχει επιλέξει ο χρήστης.

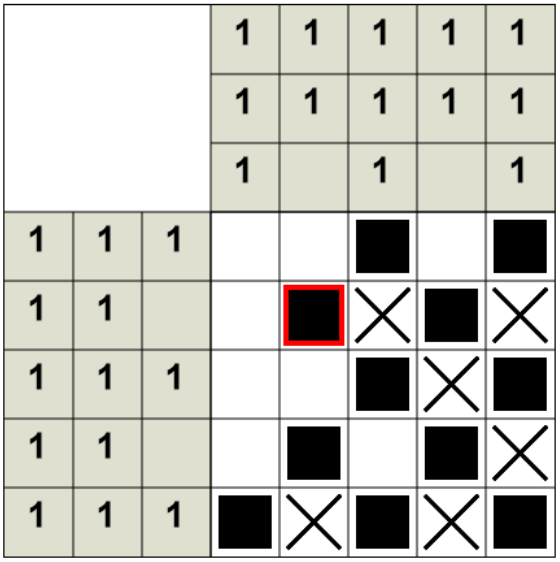
```
Nonogram.prototype.strokeCurrentChoice = function(cell) {
    if(this.previousChoice.active) {
            ctx.beginPath();
            for(let i=0; i<this.previousChoice.cell.length; i++) {
                    if(this.previousChoice.cell[i].value === 1) {
                            this.drawWhiteCell(this.previousChoice.cell[i]);
                            this.drawBlackCell(this.previousChoice.cell[i]);
                    }else if(this.previousChoice.cell[i].value === 2) {
                            this.drawWhiteCell(this.previousChoice.cell[i]);
                            this.drawXCell(this.previousChoice.cell[i]);
                    }else{
                            this.drawWhiteCell(this.previousChoice.cell[i]);
                    }
            }
            ctx.stroke();
            ctx.closePath();
            this.previousChoice.cell = []; //Αδειάζουμε τον πίνακα
    }
    this.currentChoice.cell = cell;
    this.previousChoice.cell.push(cell);
    this.previousChoice.active = true;
    ctx.strokeStyle = "red";
    ctx.lineWidth   = 4;
    ctx.strokeRect(cell.x+5, cell.y+5, this.blockSize-10, this.blockSize-10);
};
```

3. Τοποθετούμε την συνάρτηση στα σημεία που ζωγραφίζει τα κελιά η συνάρτηση **fillCels**.

```
if(this.emptyGrid[i].value == 0) {
    //…

    this.strokeCurrentChoice(this.emptyGrid[i]);
```

```
}else if(this.emptyGrid[i].value == 1) {
    //...
    this.strokeCurrentChoice(this.emptyGrid[i]);
}else {
    //...
    this.strokeCurrentChoice(this.emptyGrid[i]);
}
```

Βρισκόμαστε σε αυτό το αποτέλεσμα ως τώρα με τον χρήστη να μπορεί να βλέπει ποιο κελί έχει επιλέξει τελευταία φορά.



5.2 Μαρκάρισμα αριθμών κελιών πλέγματος

Η επόμενη λειτουργία που θα φτιάξουμε είναι να μπορεί ο χρήστης να επιλέγει τα κελιά που αντιπροσωπεύουν τους αριθμούς πλέγματος που αντιστοιχούν για τις γραμμές και τις στήλες και να τα μαρκάρει για να μπορεί να σημειώνει ποια κελιά έχει βρεί.

Στο object του NumberCell δημιουργούμε μία καινούρια ιδιότητα την value που μας δείχνει την κατάσταση του κελιού δηλαδή άμα ο χρήστης το έχει ακυρώσει η οχι.

```
function NumberCell(w, h, x, y, value, number) {
    this.w = w;
    this.h = h;
    this.x = x;
    this.y = y;
```

```
    this.value = value;
    this.number = number;
}
```

Στο αρχείο **nonogram.js** στο σημείο που δημιουργήσαμε προηγουμένως τα NumberCell προσθέτουμε και ορίζουμε μηδενική τιμή στη ιδιότητα value των κελιών για τους αριθμούς.

```
for (var i = 0; i < this.rowNumbers.length; i ++) {
   for ( var y = 0; y < this.rowNumbers[i].length; y ++) {
      this.rowNumbersGrid.push(new NumberCell(this.blockSize,
            this.blockSize,
            (y * this.blockSize),
            ((this.maxColumnNumberSize) * this.blockSize) + (i * this.blockSize),
            0, //Ορίζουμε την ιδιότητα value με μηδέν
            this.rowNumbers[i][y]));
   }
}
```

```
for (var i = 0; i < this.columnNumbers.length; i ++) {
   for ( var y = 0; y < this.columnNumbers[i].length; y ++) {
      this.columnNumbersGrid.push(new NumberCell(
            this.blockSize,
            this.blockSize,
            ((this.maxRowNumberSize) * this.blockSize) + (i * this.blockSize),
            (y * this.blockSize),
            0,
            this.columnNumbers[i][y]));
   }
}
```

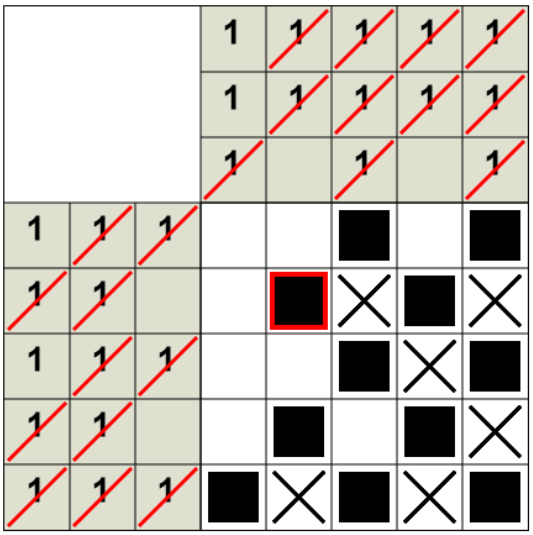
Στην συνάρτηση **fillCells** τοποθετούμε στη αρχή της συνάρτησης τον κώδικα που θα μας επιτρέψει να ακυρώνουμε τα κελιά των αριθμών. Ελέγχουμε πρώτα τα κελιά ανά γραμμή και στην συνέχεια τα κελιά ανά στήλη. Έχουμε 2 for στις οποίες ελέγχουμε αν το κελί βρισκόταν στο rowNumbersGrid ή στο columnNumbersGrid.

```
//Ορίζουμε το μέγεθος της γραμμής που θέλουμε
ctx.lineWidth = 3;
//Ξεκινάμε το "μονοπάτι" που θέλουμε να ζωγραφίσουμε
ctx.beginPath();

// κελιά ανά γραμμή
for(var i=0; i<this.rowNumbersGrid.length; i++) {
  if(mouseX >= this.rowNumbersGrid[i].x && mouseY >= this.rowNumbersGrid[i].y && mouseX <=
(this.rowNumbersGrid[i].x + this.blockSize) && mouseY <= (this.rowNumbersGrid[i].y +
this.blockSize)) {
//Αν το κελί έχει value 0 δηλαδή άμα δεν το είχε επιλέξει ο χρήστης τότε να το μαρκάρει με κόκκινη
γραμμή
      if(this.rowNumbersGrid[i].value === 0) {

         ctx.strokeStyle = "red";
         ctx.moveTo(this.rowNumbersGrid[i].x+3, (this.rowNumbersGrid[i].y + this.blockSize)-3);
         ctx.lineTo((this.rowNumbersGrid[i].x + this.blockSize)-3, this.rowNumbersGrid[i].y+3);
         this.rowNumbersGrid[i].value = 1;

      }else{
//Αν το κελί έχει value διαφορετικό από 0 δηλαδή άμα ο χρήστης το είχε επιλέξει ήδη μια φορά τότε
να το ξανά ζωγραφίσει όπως ήτανε

         ctx.fillStyle = "#e0e0d1";
         ctx.fillRect(this.rowNumbersGrid[i].x+2, this.rowNumbersGrid[i].y+2,

            this.rowNumbersGrid[i].w-3, this.rowNumbersGrid[i].h-3);

         ctx.fillStyle = "black";
         ctx.font = "bold " + (this.blockSize / 2) + "px Arial";
         ctx.fillText( this.rowNumbersGrid[i].number, (this.rowNumbersGrid[i].x) + (this.blockSize/3),

            (this.rowNumbersGrid[i].y) + ((this.blockSize+8)/2));
         this.rowNumbersGrid[i].value = 0;
      }
      break;
   }
```

```
}
```

```
// κελιά ανά στήλη

for(var i=0; i<this.columnNumbersGrid.length; i++) {
   if(mouseX >= this.columnNumbersGrid[i].x && mouseY >= this.columnNumbersGrid[i].y && mouseX

   <= (this.columnNumbersGrid[i].x + this.blockSize) && mouseY <= (this.columnNumbersGrid[i].y +

   this.blockSize)) {

      if(this.columnNumbersGrid[i].value === 0) {
         ctx.strokeStyle = "red";
         ctx.moveTo(this.columnNumbersGrid[i].x+3, (this.columnNumbersGrid[i].y + this.blockSize)-3);
         ctx.lineTo((this.columnNumbersGrid[i].x + this.blockSize)-3, this.columnNumbersGrid[i].y+3);
         this.columnNumbersGrid[i].value = 1;
      }else{
         ctx.fillStyle = "#e0e0d1";
         ctx.fillRect(this.columnNumbersGrid[i].x+2, this.columnNumbersGrid[i].y+2,

           this.columnNumbersGrid[i].w-3, this.columnNumbersGrid[i].h-3);
         ctx.fillStyle = "black";
         ctx.font = "bold " + (this.blockSize / 2) + "px Arial";
         ctx.fillText(this.columnNumbersGrid[i].number, (this.columnNumbersGrid[i].x) +

           (this.blockSize/3), (this.columnNumbersGrid[i].y) + ((this.blockSize+8)/2));
         this.columnNumbersGrid[i].value = 0;

      }
      break;

   }

}
ctx.stroke();
ctx.closePath();
```

Είμαστε πλέον σε αυτήν την κατάσταση του παιχνιδιού που μπορούμε να επιλέξουμε τα κελιά και να μαρκάρουμε τα κελιά των αριθμών που έχει βρεί ο χρήστης.



5.3 Ζωγραφίζοντας το preview της πίστας

Η επόμενη υλοποίηση που θα κάνουμε είναι να μπορεί ο χρήστης να δει την πίστα που έχει ζωγραφίσει σε preview στη πάνω αριστερή γωνία για να μπορεί να δει πιο καθαρά τι είναι αυτό που ζωγραφίζει.

Προσθέτουμε την συνάρτηση drawPreview στο αρχείο **nonogram.drawing.js** η οποία παίρνει σαν παράμετρο την cell που αντιπροσωπεύει το τελευταίο κελί που πατήθηκε.

```
Nonogram.prototype.drawPreview = function(cell) {
    let x = 0; //Οι μεταβλητές x και y αντιπροσωπεύουν τις συντεταγμένες
    let y = 0; //x και y του κελιού που πατήθηκε

    let widthPreview = this.maxRowNumberSize * this.blockSize;
    let heightPreview = this.maxColumnNumberSize * this.blockSize;
    let size; //To size θα αντιπροσωπεύει το μέγεθος του drawPreview
    if(widthPreview == heightPreview) {
            size = widthPreview-2;
            x = (Math.floor(((cell.x) - size) / this.blockSize) * Math.floor(size / this.levelGrid[0].length));
            y = (Math.floor(((cell.y) - size) / this.blockSize) * Math.floor(size / this.levelGrid.length));
    }else if(widthPreview > heightPreview){
            size = heightPreview;
            x = Math.floor(((cell.x) - widthPreview) / this.blockSize) * Math.floor(size / this.levelGrid[0].length) +
```

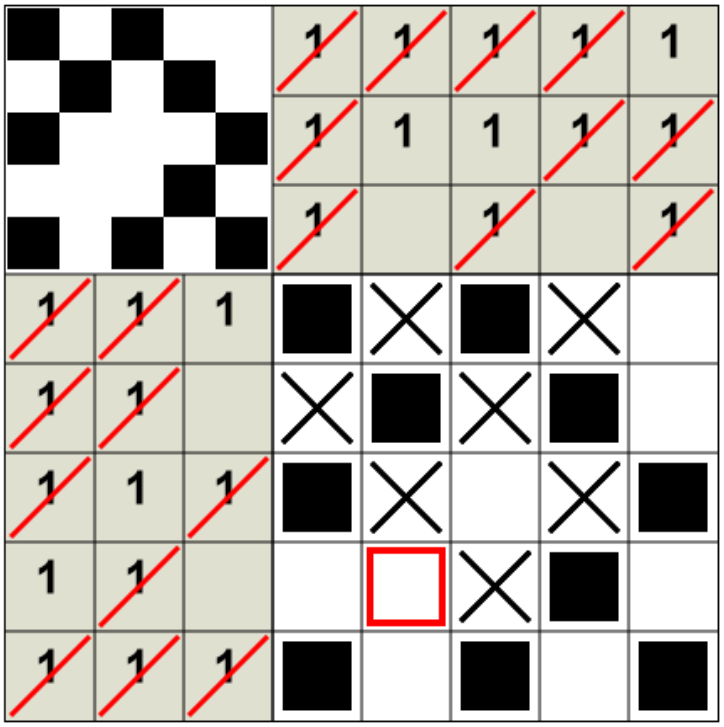
```
((widthPreview/2)-(size/2));
            y = Math.floor(((cell.y) - size) / this.blockSize) * Math.floor(size / this.levelGrid.length) +
((heightPreview/2)-(size/2));
    }else{
            size = widthPreview-8;
            x = Math.floor(((cell.x) - size) / this.blockSize) * Math.floor(size / this.levelGrid[0].length) +
((widthPreview/2)-(size/2));
            y = Math.floor(((cell.y) - heightPreview) / this.blockSize) * Math.floor(size / this.levelGrid.length) +
((heightPreview/2)-(size/2));
    }
    let widthCell = Math.floor(size / this.levelGrid[0].length); //328/5
    let heightCell = Math.floor(size / this.levelGrid.length); //328/5
    if(cell.value === 1) {
            ctx.fillStyle = "black";
            ctx.fillRect(x + Math.floor((size-(widthCell*this.levelGrid[0].length))/2), y +
Math.floor((size-(heightCell*this.levelGrid.length))/2), widthCell, heightCell);
    }else{
            ctx.fillStyle = "white";
            ctx.fillRect(x + Math.floor((size-(widthCell*this.levelGrid[0].length))/2), y +
Math.floor((size-(heightCell*this.levelGrid.length))/2), widthCell, heightCell);
    }
};
```

Προσθέτουμε την συνάρτηση drawPreview στο αρχείο **nonogram.drawing.js** στην συνάρτηση fillCels μέσα στην υλοποίηση της if και σε κάθε else, μετά την συνάρτηση strokeCurrentChoice .

```
//...
if(this.emptyGrid[i].value == 0) { //fill the cell black
    //...
    this.strokeCurrentChoice(this.emptyGrid[i]);
    this.drawPreview(this.emptyGrid[i]);
}else if(this.emptyGrid[i].value == 1) { //fill the cell with a X
    //...
    this.strokeCurrentChoice(this.emptyGrid[i]);
    this.drawPreview(this.emptyGrid[i]);
```

```
}else { //Clear the cell
  //...
  this.strokeCurrentChoice(this.emptyGrid[i]);
  this.drawPreview(this.emptyGrid[i]);
}
//...
```

Έχουμε φτάσει ως τώρα σε αυτό το σημείο και έχουμε το εξής αποτέλεσμα.



Στη συνέχεια θα προσθέσουμε την δυνατότητα με την οποία ο χρήστης να μπορεί να επιλέξει πολλαπλά κελιά.

5.4 Μαρκάρισμα πολλαπλών κελιών της πίστας

Θέλουμε ο χρήστης να μπορεί να επιλέξει πολλαπλά κελιά είτε αυτά βρίσκονται σε μία γραμμή ή σε μια στήλη. Για να το κάνουμε αυτό δημιουργούμε στο αρχείο **nonogram.drawing.js** την συνάρτηση fillMultiCells η οποία θα έχει 4 παραμέτρους τις: startPointMouseX και startPointMouseY οι οποίες θα είναι οι συντεταγμένες που πάτησε ο χρήστης το αριστερό click του ποντικιού και τις mouseX και mouseY οι οποίες είναι οι συντεταγμένες στις οποίες βρίσκεται το ποντίκι όσο ο χρήστης κρατάει πατημένο το αριστερό click. Τα events που θα χρειαστούμε για αυτήν την υλοποίηση είναι το mousedown και mousemove για το control του ποντικιού και για το touch είναι το touchstart και touchmove.

```
//Επιλογή πολλαπλών κελιών

Nonogram.prototype.fillMultiCells = function(mouseX, mouseY, startPointMouseX, startPointMouseY) {
  //...
};
```

Το πρώτο πράγμα που πρέπει να κάνουμε είναι να βρούμε το κελί στο οποίο ο χρήστης κράτησε το click πατημένο. Για να το κάνουμε αυτό πρέπει να φτιάξουμε 3 μεταβλητές: την startCellValue που θα είναι η κατάσταση(value) του πρώτου κελιού που πατήθηκε και τις x , y που θα είναι οι συντεταγμένες του. Με μία for βρίσκουμε το κελί το οποίο πατήθηκε συγκρίνοντας τις συντεταγμένες των κελιών με τις συντεταγμένες startPointMouse.

```
var startCellValue = 0;
var x = 0;
var y = 0;
for(var i=0;i<this.emptyGrid.length;i++) {
  if(startPointMouseX >= this.emptyGrid[i].x && startPointMouseY >=
    this.emptyGrid[i].y && startPointMouseX <= (this.emptyGrid[i].x +
    this.blockSize) && startPointMouseY <= (this.emptyGrid[i].y +

    this.blockSize)) {
      startCellValue = this.emptyGrid[i].value;
      x = this.emptyGrid[i].x;
      y = this.emptyGrid[i].y;
  }
}
```

Στη συνέχεια ελέγχουμε αν ο χρήστης θέλει να συμπληρώσει κελιά στη γραμμή ή στη στήλη που βρίσκεται. Για να το κάνουμε αυτό έχουμε μία if στην οποία ελέγχουμε αν έχει περάσει τις συντεταγμένες του κελιού που πάτησε και κράτησε πατημένο το αριστερό click στον άξονα x ή y.

```
if((mouseX > x && (mouseX < x + this.blockSize)) || (mouseY > y && (mouseY < y + this.blockSize)) ) {
    //...
}
```

Στη συνέχεια σε κάθε περίπτωση της if τοποθετούμε των ίδιο κώδικα και κάνουμε τα εξής βήματα:

Περνάμε από όλα τα κελιά με μία for και κάνουμε έλεγχο με μία if για να βρούμε σε ποιό κελί βρισκόμαστε. Ελέγχουμε στην συνέχεια με μία if αν βρισκόμαστε ακόμα στο ίδιο κελί και αν βρισκόμαστε τότε βγαίνουμε από την συνάρτηση με την εντολή return. Εφόσον βρισκόμαστε σε καινούργιο κελί τότε αλλάζουμε την κατάσταση του καινούργιου κελιού με την κατάσταση που είχε το αρχικό κελί. Στη συνέχεια με βάση την κατάσταση του κελιού που έχουμε ζωγραφίζουμε αντίστοιχα το κελί όπως κάναμε και στη συνάρτηση fillCells.

```
for(var i=0;i<this.emptyGrid.length;i++) {
 if(mouseX >= this.emptyGrid[i].x && mouseY >= this.emptyGrid[i].y && mouseX

 <= (this.emptyGrid[i].x + this.blockSize) && mouseY <= (this.emptyGrid[i].y +
 this.blockSize)) {
  if(this.emptyGrid[i].x == x && this.emptyGrid[i].y == y) {
   return;
  }else if(this.emptyGrid[i].x == this.currentChoice.cell.x &&
  this.emptyGrid[i].y == this.currentChoice.cell.y) {
   return;
  }
  this.emptyGrid[i].value = startCellValue;
  if(startCellValue == 1) {
   this.drawWhiteCell(this.emptyGrid[i]);
        this.drawBlackCell(this.emptyGrid[i]);
        this.drawPreview(this.emptyGrid[i]);
        this.currentChoice.cell = this.emptyGrid[i];
        this.previousChoice.cell.push(this.emptyGrid[i]);
        ctx.strokeStyle = "red";
        ctx.lineWidth   = 4;
        ctx.strokeRect(this.currentChoice.cell.x+5, this.currentChoice.cell.y+5,
   this.blockSize-10, this.blockSize-10);
  }else if(startCellValue == 2) {
   this.drawWhiteCell(this.emptyGrid[i]);
        this.drawPreview(this.emptyGrid[i]);
        this.drawXCell(this.emptyGrid[i]);
        this.currentChoice.cell = this.emptyGrid[i];
        this.previousChoice.cell.push(this.emptyGrid[i]);
```

```
            ctx.strokeStyle = "red";
            ctx.lineWidth   = 4;
            ctx.strokeRect(this.currentChoice.cell.x+5, this.currentChoice.cell.y+5,
      this.blockSize-10, this.blockSize-10);
    }else{
      this.drawWhiteCell(this.emptyGrid[i]);
            this.drawPreview(this.emptyGrid[i]);
            this.currentChoice.cell = this.emptyGrid[i];
            this.previousChoice.cell.push(this.emptyGrid[i]);
            ctx.strokeStyle = "red";
            ctx.lineWidth   = 4;
            ctx.strokeRect(this.currentChoice.cell.x+5, this.currentChoice.cell.y+5,
      this.blockSize-10, this.blockSize-10);
    }
  }
}
```

Φτιάξαμε την συνάρτηση που μας επιτρέπει να ζωγραφίζουμε πολλαπλά κελιά και τώρα θα προσθέσουμε των κώδικα για την αλληλεπίδραση του χρήστη.

Στο αρχείο **mouse.js** προσθέτουμε την μεταβλητή isDown η οποία θα μας δείχνει αν το αριστερό click του ποντικιού είναι πατημένο ή όχι.

```
let isDown = false;
```

Τοποθετούμε στο event mousedown που έχουμε ήδη την μεταβλητή isDown με τιμή true για να γνωρίζουμε ότι ο χρήστης έχει πατημένο το αριστερό click του ποντικιού.

```
$(canvas).mousedown(function(event) {
    //...
    isDown = true;
    nonogram.fillCels(startPointMouseX, startPointMouseY);
});
```

Όταν ο χρήστης αφήσει το αριστερό click θέλουμε η μεταβλητή isDown να έχει τιμή false για να ξέρουμε ότι χρήστης έχει αφήσει το αριστερό click. Για να το κάνουμε αυτό φτιάχνουμε το event mouseup.

```
$(canvas).mouseup(function(){
    isDown = false;
});
```

Επίσης θέλουμε όταν το ποντίκι βγει εκτός πίστας η μεταβλητή isDown να παίρνει τιμή false γι αυτό και φτιάχνουμε το event mouseout..

```
$(canvas).mouseout(function(){
    isDown = false;
});
```

Όταν το ποντίκι κουνιέται πάνω στον canva θέλουμε να παίρνουμε τις συντεταγμένες του ποντικιού και να ελέγχουμε αν ο χρήστης έχει ακόμα πατημένο το αριστερό click που αν ισχύει μόνο τότε μπορούμε να ζωγραφίσουμε σε πολλαπλά κελιά.

```
$(canvas).mousemove(function(event){
    mouseX = event.offsetX ;
    mouseY = event.offsetY ;
    if(isDown){
        nonogram.fillMultiCells(mouseX,

          mouseY, startPointMouseX,

          startPointMouseY);
    }
});
```

Για την επιλογή πολλαπλών κελιών τοποθετούμε στο αρχείο touch.js τα events touchmove και touchend.

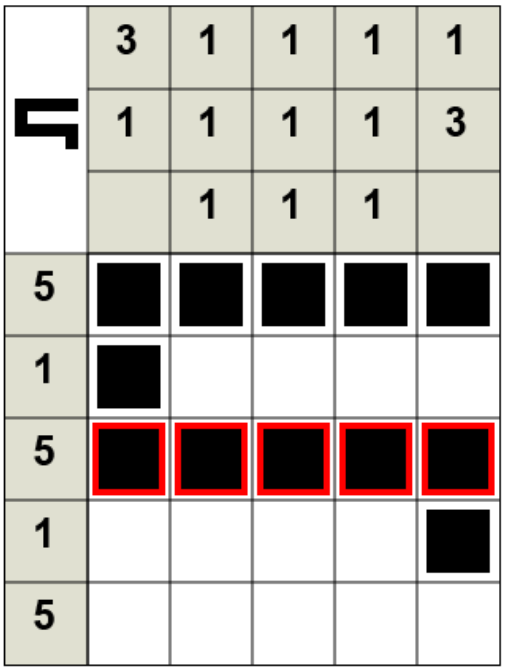
```
$(canvas).on('touchend', function(event) {
    isDown = false;
});
```

Και στο event touchmove προσθέτουμε

```
$(canvas).on('touchmove', function(event) {
    var touchX = Math.floor(event.touches[0].clientX - ((window.innerWidth - canvas.width) / 2));
```

```
    var touchY = Math.floor(event.touches[0].clientY - ((window.innerHeight - canvas.height) / 2));

    nonogram.fillMultiCells(touchX-originX, touchY, startPointTouchX, startPointTouchY);
});
```

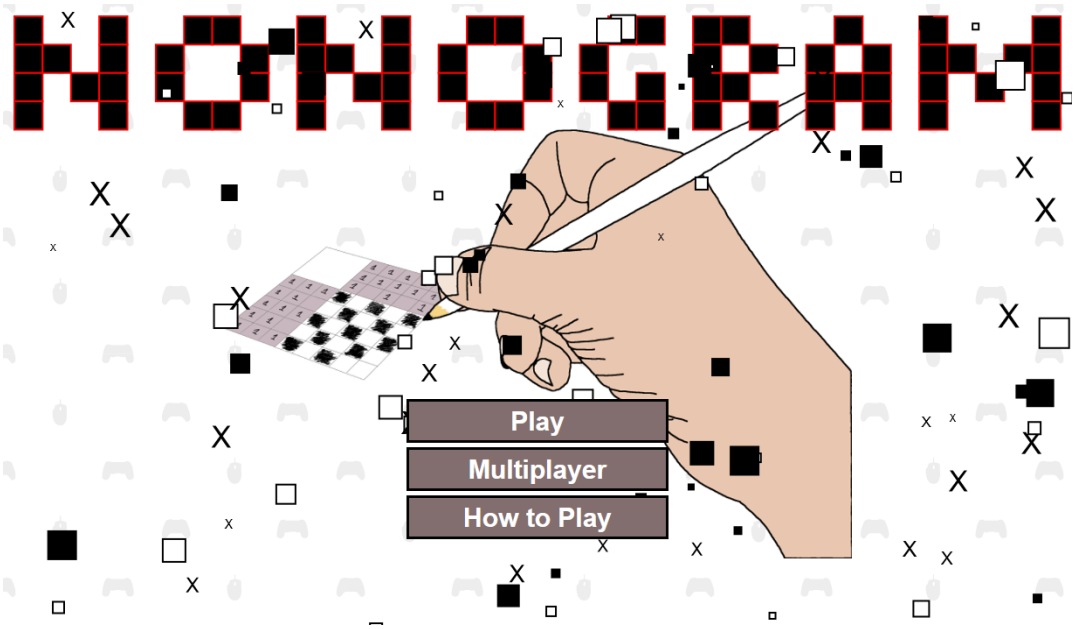
Είμαστε σε αυτό το σημείο



Η επόμενη υλοποίηση που θα κάνουμε είναι να φτιάξουμε την αρχική σελίδα και το μενού του παιχνιδιού από το οποίο ο χρήστης θα μπορεί να επιλέξει τι να παίξει singleplayer ή multiplayer ή να δει τις οδηγίες του παιχνιδιού.

6. Αρχική σελίδα και μενού πλοήγησης του παιχνιδιού

Για την αρχική σελίδα και το μενού πλοήγησης του παιχνιδιού θέλουμε να πετύχουμε το παρακάτω αποτέλεσμα.



Για να το κάνουμε αυτό θα φτιάξουμε στο αρχείο **index.html** ορισμένα καινούρια div που θα μας βοηθήσουν να τοποθετήσουμε τις επιλογές του μενού με βάση την οθόνη του χρήστη. Το πρώτο div που θα φτιάξουμε είναι το div με id **screen** και θα αφορά την οθόνη ή το παράθυρο που θα έχει ανοιχτό ο χρήστης. Μέσα σε αυτό το div θα φτιάξουμε άλλο ένα div με id **container** το οποίο θα περιέχει όλο το παιχνίδι μας και θα μας βοηθήσει να το τοποθετήσουμε στο κέντρο με βάση την οθόνη του χρήστη.

Προσθέτουμε των κώδικα css που χρειαζόμαστε για να μορφοποιήσουμε την αρχική μας σελίδα στο αρχείο **css/home.css**. Διαγράφουμε την css που είχαμε για τον canva.

```
* {
  margin: 0;
  padding: 0;
}

html, body {
  width:  100%;
  height: 100%;
  display: block;  /* No floating content on sides */
}

body {
        background: url(../img/gaming-pattern.png);
```

```
        background-repeat: repeat;
}

#screen {
        display: block;
        margin: 0 auto;
        padding: 0px;
        position: relative;
        height: 100%;
}

#container {
        position: absolute;
        top: 0%;
        left: 0%;
        display: block;
}

#game {
  display: block;
  position: relative;
}

#canvas {
  display: block;
  margin: 0 auto;
  padding: 0;
}
```

Τα tags που θα περιέχει το παιχνίδι μας για το singleplayer κομμάτι του είναι τα παρακάτω, στην συνέχεια στην ενότητα του multiplayer θα προσθέσουμε των κώδικα που χρειαζόμαστε για το μενού του multiplayer. Στο αρχείο **index.html** αφαιρούμε το tag του canvas που είχαμε και τοποθετούμε τα εξής

```
<div id="screen">
   <div id = "container">
```

```
        <!-- Game -->
        <div id="game">
      <canvas id="canvas"></canvas>
    </div>
        <!-- Menu -->
        <div id="menu"></div>
        <!-- Levels -->
        <div id="levels"></div>
        <!-- Stages 5x5 -->
        <div id="levels5x5" class="stages"></div>
        <!-- Stages 10x10 -->
        <div id="levels10x10" class="stages"></div>
        <!-- Stages 15x15 -->
        <div id="levels15x15" class="stages"></div>
        <!-- Correct for singleplayer-->
        <div id="correct-singleplayer" class="black-screen"></div>
        <!-- How to play -->
        <div id="instructions"></div>
  </div>
</div>
```

Στην συνέχεια θα αναφέρω τι περιέχει το κάθε ένα ξεχωριστά αλλά πρώτα πρέπει να φτιάξουμε την αρχική μας σελίδα.

Στο αρχείο **home.js** σβήνουμε όλο των κώδικα που είχαμε γράψει και γράφουμε τις εξής μεταβλητές: την canvas που θα περιέχει το tag element του canva μας, την ctx που θα περιέχει το context του canva μας, την screen που θα έχει το div tag element με id screen, την container που θα έχει το div tag element με id container και την state που θα περιέχει την κατάσταση που βρίσκεται το παιχνίδι δηλαδή άμα βρίσκεται στο μενού, άμα ο χρήστης παίζει singleplayer ή multiplayer.

```
"use strict";
let canvas    = document.getElementById("canvas");
let ctx       = canvas.getContext("2d");
let screen    = document.getElementById("screen");
let container = document.getElementById("container");
let state     = "menu";
```

Θέλουμε ο canvas μας να πιάσει όλοι την οθόνη του χρήστη, για να το κάνουμε αυτό προσθέτουμε των κώδικα που χρειαζόμαστε για να δώσουμε στον canva μας το μήκος και το ύψος του παραθύρου.

```
canvas.width = window.innerWidth;
canvas.height = window.innerHeight;
```

Για να μπορέσουμε να ζωγραφίσουμε την λέξη nonogram, δημιουργούμε την συνάρτηση introScreenLogo στην οποία έχουμε των κώδικα που χρειαζόμαστε. Η λέξη nonogram βρίσκεται στην μεταβλητή textLogo η οποία είναι ένας δισδιάστατος πίνακας. Όπου υπάρχει 1 σημαίνει ότι πρέπει να ζωγραφίσουμε ένα μαύρο κουτάκι στο οποίο ζωγραφίζουμε και ένα κόκκινο περίγραμμα για να το τονίσουμε. Για να μπορέσουμε να ζωγραφίσουμε το logo μας στο κέντρο υπολογίζουμε το blockSize ως εξής: παίρνουμε το μήκος του canva μας και αφαιρούμε την απόσταση που θέλουμε να δώσουμε στο logo μας (-30) από την αρχή του canva και το διαιρούμε με τον αριθμό των κελιών που χρειαζόμαστε να ζωγραφίσουμε μία σειρά από το logo μας (37). Για να μπορέσουμε να κεντράρουμε το logo τροποποιούμε τις συντεταγμένες των κελιών προσθέτοντας την μισή τιμή της απόστασης που δώσαμε προηγουμένως.

```
let introScreenLogo = {
    textLogo: [

[1,0,0,1,0,0,1,1,0,0,1,0,0,1,0,0,1,1,0,0,0,1,1,0,1,1,0,0,0,1,0,0,1,0,0,0,1],

[1,1,0,1,0,1,0,0,1,0,1,1,0,1,0,1,0,0,1,0,1,0,0,0,1,0,1,0,1,0,1,0,1,1,0,1,1],

[1,0,1,1,0,1,0,0,1,0,1,0,1,1,0,1,0,0,1,0,1,0,1,0,1,1,0,0,1,1,1,0,1,0,1,0,1],

[1,0,0,1,0,0,1,1,0,0,1,0,0,1,0,0,1,1,0,0,0,1,1,0,1,0,1,0,1,0,1,0,1,0,0,0,1]
    ],
    blockSize: 0,
    draw: function() {
      ctx.fillStyle = "black";
            this.blockSize = (canvas.width-30) / 37;
            for(let i=0; i<this.textLogo.length; i++){
                        for(let y=0; y<this.textLogo[i].length; y++){
                                    if(this.textLogo[i][y] === 1){
                                                ctx.beginPath();
```

```
                    ctx.fillRect(y*this.blockSize + 15,
                i*this.blockSize + 15,
                this.blockSize, this.blockSize);
                    ctx.strokeStyle = "red";
                    ctx.lineWidth   = 2;
                    ctx.strokeRect(y*this.blockSize + 15,
                 i*this.blockSize + 15,
                 this.blockSize, this.blockSize);
                 }
            }
        }
    }
};
```

Φτιάξαμε το logo μας, στην συνέχεια θα φτιάξουμε το animation που γίνεται με τα κουτάκια τα οποία αντιπροσωπεύουν τις επιλογές που κάνει ο χρήστης στις πίστες. Δημιουργούμε 2 objects το Rect που θα αντιπροσωπεύει τα μαύρα και άσπρα κουτάκια και το xRect που θα αντιπροσωπεύει τα κουτάκια με το X. Τα 2 object θα έχουν παρόμοιες μεταβλητές και μεθόδους, έχουν τις **x** και **y** που είναι οι συντεταγμένες που θα βρίσκονται πάνω στον canva, τις **w** και **h** που είναι το μήκος και το ύψος που θα έχει το κουτάκι, την **dx** που θα είναι η ταχύτητα με την οποία θα πηγαίνει προς τα πάνω και κάτω το κουτάκι και από τις μεθόδους την **draw** οι οποία θα ζωγραφίζει στον canva, την **update** η οποία θα ανανεώσει την θέση του αντικειμένου με βάση το που βρίσκεται στον canva εκείνη την στιγμή δηλαδή αν η συντεταγμένη y έχει ξεπεράσει τα κάτω όρια (y + w > innerHeight) ή τα πάνω όρια (y < 0) να αλλάζουμε την κατεύθυνση του αντικειμένου (dx = -dx) και την **relocate** η οποία μας δίνει καινούριες συντεταγμένες x και y, στην περίπτωση που αλλάξει το μέγεθος του παραθύρου του χρήστη ή αν αλλάξει ο προσανατολισμός της συσκευής του χρήστη, οριζόντια ή κάθετα για να μπορέσουμε να ξανά ζωγραφίσουμε τα αντικείμενα στον canva. Οι διαφορές τους είναι ότι η class Rect έχει 2 επιπλέον ιδιότητες την **color** που είναι για το χρώμα με το οποίο θα ζωγραφίσουμε το κουτάκι και την **stroke** που θα είναι το χρώμα του περιγράμματος του κουτιού και οι xRect που έχει την ιδιότητα **textSize** που θα είναι το μέγεθος που θα γράψουμε το X.

```
//Rectangle class
function Rect(x, y, w, h, dx, color, stroke) {
    this.x = x;
    this.y = y;
    this.w = w;
```

```
    this.h = h;
    this.dx = dx;
    this.color = color;
    this.stroke = stroke;

    this.draw = function() {
            ctx.beginPath();
            ctx.fillStyle = color;
            ctx.fillRect(this.x, this.y, this.w, this.h);
            ctx.stroke();
            ctx.fill();
            ctx.strokeStyle = "black";
            ctx.lineWidth   = 2;
           ctx.strokeRect(this.x, this.y, this.w, this.h);
    }

    this.update = function() {
            if(this.y + this.w > innerHeight || this.y < 0){
                    this.dx = -this.dx;
            }
            this.y += this.dx;
            this.draw();
    }

    this.relocate = function(x, y) {
            this.x = x;
            this.y = y;
    }
};
```

```
//X shape class
function xRect(x, y, w, h, dx, textSize) {
    this.x = x;
```

```
    this.y = y;
    this.w = w;
    this.h = h;
    this.dx = dx;
    this.textSize = textSize;
    this.draw = function() {
            ctx.beginPath();
            ctx.font = textSize +"px Arial";
            ctx.fillStyle = "black";
            ctx.fillText("X", this.x, this.y);
            ctx.stroke();
            ctx.fill();
    }
    this.update = function() {
            if(this.y + this.w > innerHeight || this.y < 0){
                    this.dx = -this.dx;
            }
            this.y += this.dx;
            this.draw();
    }
    this.relocate = function(x, y) {
            this.x = x;
            this.y = y;
    }
};
```

Στην συνέχεια δημιουργούμε τα κουτάκια μας, ξεκινάμε φτιάχνοντας 3 μεταβλητές, που θα είναι πίνακες που θα περιέχει τα κουτιά

```
let blackRectArray = [];
let whiteRectArray = [];
let xRectArray    = [];
```

και δημιουργούμε 30 κουτάκια από κάθε είδος

```
//Create 30 black rects
```

```
for(let i=0; i<30; i++){
    let size = Math.floor(Math.random() * 30) + 5;
    let x = Math.random() * (innerWidth - size * 2) + size;
    let y = Math.random() * (innerHeight - size * 2) + size;
    let dx = Math.random() * 4;
    let color = "black";
    blackRectArray.push(new Rect(x, y, size, size, dx, color));
}
//Create 30 white rects
for(let i=0; i<30; i++){
    let size = Math.floor(Math.random() * 30) + 5;
    let x = Math.random() * (innerWidth - size * 2) + size;
    let y = Math.random() * (innerHeight - size * 2) + size;
    let dx = Math.random() * 4;
    let color = "white";
    whiteRectArray.push(new Rect(x, y, size, size, dx, color));
}
//Create 30 x shape rects
for(let i=0; i<30; i++){
    let size = Math.floor(Math.random() * 30) + 5;
    let x = Math.random() * (innerWidth - size * 2) + size;
    let y = Math.random() * (innerHeight - size * 2) + size;
    let dx = Math.random() * 4;
    let textSize = Math.floor(Math.random() * 30) + 10;
    xRectArray.push(new xRect(x, y, size, size, dx, textSize));
}
```

Στην συνέχεια τοποθετούμε την εικόνα που έχουμε φτιάξει η οποία απεικονίζει έναν χρήστη να προσπαθεί να λύσει μία πίστα. Έχω φτιάξει διάφορες διαστάσεις της εικόνας για να μπορέσω να καλύψω τις περισσότερες διαστάσεις οθονών των συσκευών. Αναλόγως με τις διαστάσεις του παραθύρου χρησιμοποιούμε την κατάλληλη εικόνα. Δημιουργούμε ένα object image που θα αντιπροσωπεύει την εκάστοτε εικόνα μας.

```
let img = new Image();

if(window.innerHeight >= 753) {
```

```
    if(window.innerWidth >= 999) {
            img.src = "img/nono_1000X753.png";
    }else if(window.innerWidth < 999 && window.innerWidth >= 719) {
            img.src = "img/nono_720X542.png";
    }else if(window.innerWidth < 719 && window.innerWidth >= 480) {
            img.src = "img/nono_480X361.png";
    }else if(window.innerWidth < 480 && window.innerWidth >= 360) {
            img.src = "img/nono_360X271.png";
    }else if(window.innerWidth < 360 && window.innerWidth >= 304) {
            img.src = "img/nono_304X229.png";
    }
}else if(window.innerHeight < 753 && window.innerHeight >= 543) {
    if(window.innerWidth >= 719) {
            img.src = "img/nono_720X542.png";
    }else if(window.innerWidth < 719 && window.innerWidth >= 480) {
            img.src = "img/nono_480X361.png";
    }else if(window.innerWidth < 480 && window.innerWidth >= 360) {
            img.src = "img/nono_360X271.png";
    }else if(window.innerWidth < 360 && window.innerWidth >= 304) {
            img.src = "img/nono_304X229.png";
    }
}else if(window.innerHeight < 543 && window.innerHeight >= 360) {
    if(window.innerWidth >= 480) {
            img.src = "img/nono_480X361.png";
    }else if(window.innerWidth < 480 && window.innerWidth >= 360) {
            img.src = "img/nono_360X271.png";
    }else if(window.innerWidth < 360 && window.innerWidth >= 304) {
            img.src = "img/nono_304X229.png";
    }
}else if(window.innerHeight < 360) {
    if(window.innerWidth >= 360) {
            img.src = "img/nono_360X271.png";
    }else if(window.innerWidth < 360 && window.innerWidth >= 304) {
            img.src = "img/nono_304X229.png";
    }
```

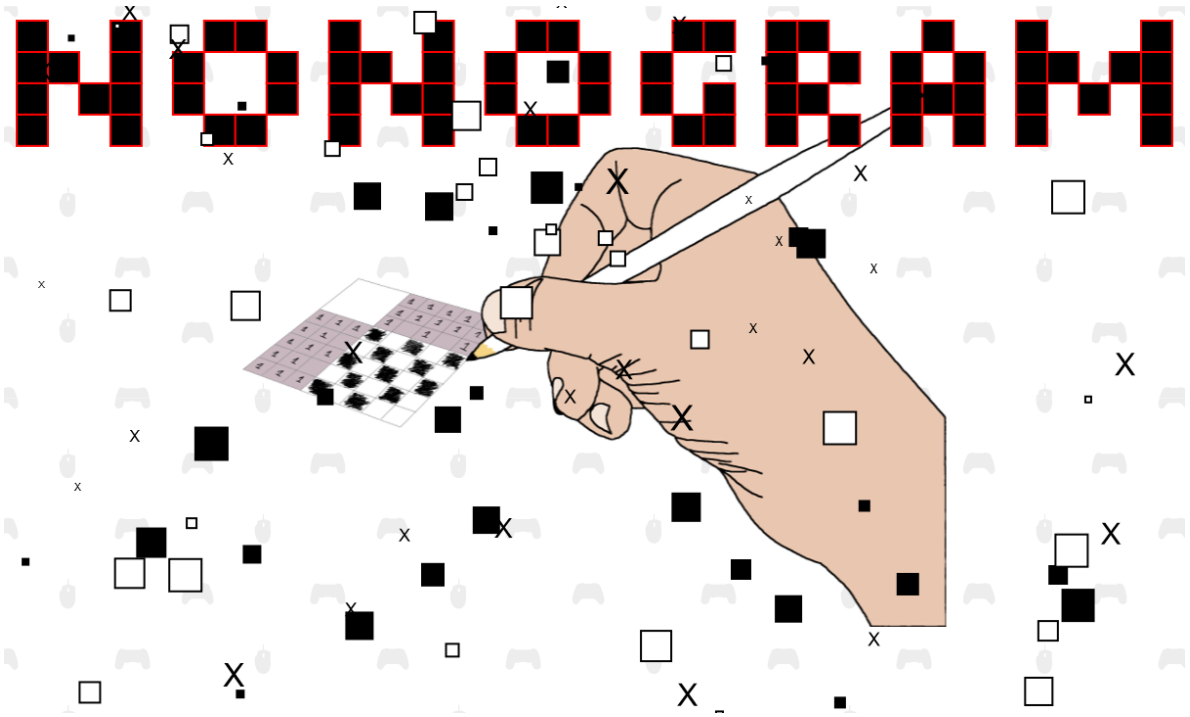
```
}
```

Για να γράφουμε λιγότερο κώδικα και για δική μας διευκόλυνση φτιάχνουμε την συνάρτηση **clearCanvas** η οποία θα μας καθαρίζει των canva. Δημιουργούμε την συνάρτηση **animate** στην οποία πρώτα θα καθαρίζουμε τον canva και στην συνέχεια θα ζωγραφίζουμε την εικόνα μας, τα αντικείμενα μας και στην οποία θα ανανεώνονται οι συντεταγμένες των αντικειμένων. Ελέγχουμε αν το παιχνίδι μας βρίσκεται στο μενού για να μπορέσουμε να ζωγραφίσουμε την αρχική οθόνη.

```
function clearCanvas() {
    ctx.clearRect(0,0,canvas.width,canvas.height);
}

function animate() {
    if(state === "menu") {
            clearCanvas();
            ctx.drawImage(img, (innerWidth/2)-(img.width/2),

                (innerHeight/2)-(img.height/2));
            introScreenLogo.draw();
            for(var i = 0; i < blackRectArray.length; i++) {
                    blackRectArray[i].update();
                    whiteRectArray[i].update();
                    xRectArray[i].update();
            }
    }
}
```

Για το animation θα χρησιμοποιήσουμε την συνάρτηση της javascript **setInternal()** η οποία μας επιτρέπει να εκτελέσουμε έναν κώδικα επαναλαμβανόμενα σε ένα χρονικό διάστημα που έχουμε ορίσει. Καλούμε την συνάρτηση στο τέλος του αρχείου **home.js**.

```
setInterval(animate, 1000/50);
```



Έχουμε φτάσει σε αυτό το αποτέλεσμα της αρχικής οθόνης, στην συνέχεια θα φτιάξουμε τις επιλογές του μενού μας.

6.1 Μενού πλοήγησης του χρήστη

Όπως αναφέραμε παραπάνω, θα αναλύσουμε το περιεχόμενο του div με id container. Για αρχή το div tag game περιέχει τον canvas μας που θα περιέχει την αρχική μας οθόνη ή την εκάστοτε πίστα που θα παίζει ο χρήστης, είτε στο singleplayer είτε στο multiplayer. Στη συνέχεια θα προστεθούν κι άλλες λειτουργίες όπως είναι τα εργαλεία της πίστας κ.α θα μιλήσουμε γι αυτά στο κεφάλαιο των εργαλείων.

Το div tag menu θα είναι το menu του παιχνιδιού το οποίο θα περιέχει 3 επιλογές την **Play** που αναφέρεται κυρίως στο singleplayer κομμάτι, το **Multiplayer** για να μπορεί ο χρήστης να παίζει πίστες με άλλους χρήστες και το **How to Play** που θα περιέχει τις οδηγίες του παιχνιδιού.

```
<!-- Menu -->
<div id="menu">
   <ul>
         <li id="play">Play</li>
         <li id="multiplayer">Multiplayer</li>
         <li id="how-to-play">How to Play</li>
```

```
    </ul>
</div>
```

Προσθέτουμε στο αρχείο **home.css** την css που χρειαζόμαστε για το menu μας.

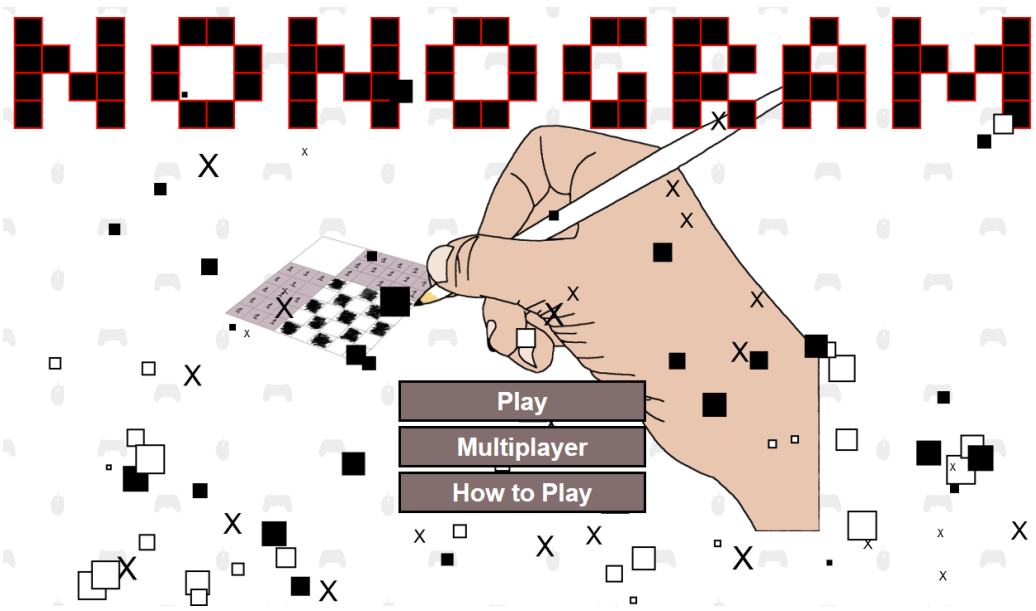
```
#menu {
  display: inline-block;
  position: absolute;
  text-align: center;
  font-weight: bold;
  font-size: 30px;
  font-family: Arial;
  bottom: 15%;
  left: 50%;
  transform: translateX(-50%);
  width: 300px;
}

#menu ul {
  display: block;
  list-style-type: none;
  padding: 0;
}

#menu li {
  background: #826e6e;
  border: 3px solid black;
  margin-bottom: 5px;
  padding-left: 20px;
  padding-right: 20px;
  padding-top: 5px;
  padding-bottom: 5px;
  cursor: pointer;
  color: white;
}
```

Δημιουργούμε στον φάκελο **js** ένα καινούργιο αρχείο με όνομα **main-menu.js**, στο οποίο θα βάλουμε την javascript που θα χρειαστούμε για την αλληλεπίδραση του χρήστη με τις επιλογές. Προσθέτουμε την javascript που χρειαζόμαστε για την αλληλεπίδραση των επιλογών εκτός του multiplayer που θα μιλήσουμε γι αυτήν στο κεφάλαιο του multiplayer.

```
$("#play").click(function(){
    $("#menu").hide();
    $("#levels").show();
});


$('#how-to-play').click(function() {
    $('#menu').hide();
    $('#instructions').show();
});
```

Προσθέτουμε το αρχείο **main-menu.js** στο αρχείο **index.js.**

```
<script src="js/main-menu.js"></script>
```



Το div tag levels θα περιέχει τις κατηγορίες των πιστών από τις οποίες θα μπορεί ο χρήστης να επιλέξει 3 κατηγορίες: την **5x5**, **10x10**, **15x15** και ένα κουμπί που θα κλείνει το μενού των κατηγοριών.

```
<!-- Levels -->
<div id="levels">
    <h3>Levels</h3>
    <div class="level level5x5">5x5</div>
    <div class="level level10x10">10x10</div>
    <div class="level level15x15">15x15</div>
    <div id="close-levels">Close</div>
</div>
```

Δημιουργούμε στον φάκελο css ένα καινούργιο αρχείο το **level.css** στο οποίο θα έχουμε την css που θα χρειαστούμε για ότι αφορά τις πίστες.

```
#levels {
  display: none;
  border: 3px solid black;
  border-radius: 25px;
  position: absolute;
  text-align: center;
  margin: 0 auto;
  top: 30%;
  transform: translateX(-50%);
  left: 50%;
  background: #826e6e;
  font-weight: bold;
  font-size: 25px;
  font-family: Arial;
  box-shadow: 3px 3px 7px grey;
  width: 250px;
  color: white;
}

#levels h3 {
  padding-top: 15px;
  padding-bottom: 15px;
  margin: 0;
```

```
  cursor: default;
}

.level {
  background: #826e6e;
  border-top: 1px solid black;
  padding: 10px;
  cursor: pointer;
}

.level:hover {
  background: linear-gradient(to bottom right, #e0e0d1, grey, #cccccc);
  color: white;
}

#close-levels {
  cursor: pointer;
  padding-top: 15px;
  padding-bottom: 15px;
  border-top: 1px solid black;
}
```

Προσθέτουμε την css στο **index.html.**

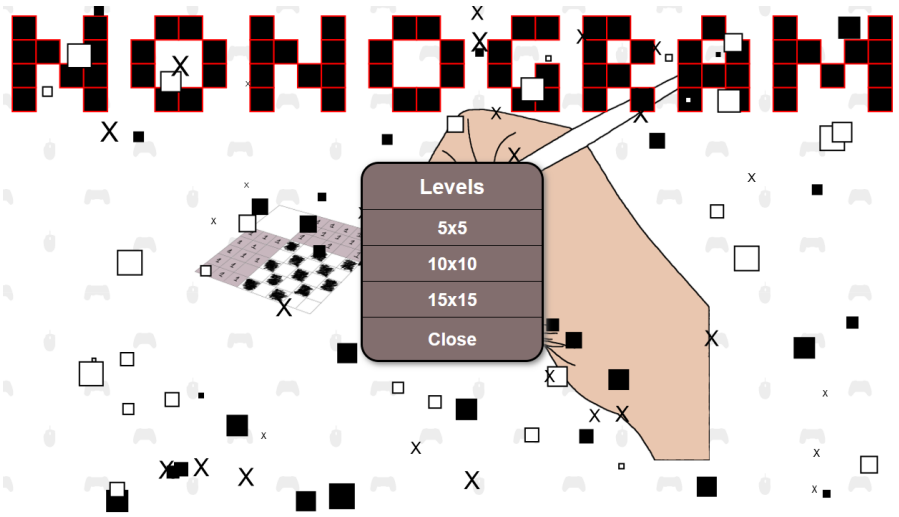
```
<link rel="stylesheet" type="text/css" href="css/level.css">
```

Προσθέτουμε στο αρχείο **main-menu.js** και την αντίστοιχη javascript που χρειαζόμαστε για την αλληλεπίδραση των κατηγοριών.

```
$("#close-levels").click(function(){
    $("#menu").show();
    $("#levels").hide();
});

$(".level5x5").click(function(){
    $("#levels5x5").show();
```

```
    $("#levels").hide();
});

$(".level10x10").click(function(){
    $("#levels10x10").show();
    $("#levels").hide();
});

$(".level15x15").click(function(){
    $("#levels15x15").show();
    $("#levels").hide();
});
```

Έχουμε φτιάξει το εξής αποτέλεσμα ως τώρα.



Τα div tag με id **levels5x5, levels10x10** και **levels15x15** θα περιέχουν το σύνολο των πιστών κάθε κατηγορίας πιστών, η υλοποίηση θα είναι ίδια για κάθε κατηγορία πιστών. Στο επόμενο κεφάλαιο θα εξηγήσουμε την υλοποίηση για την δημιουργία των πιστών, το πως λειτουργούν οι πίστες μας και θα προσθέσουμε την css και την javascript που θα χρειαστούμε.

```
<!-- Stages 5x5 -->
<div id="levels5x5" class="stages"></div>
<!-- Stages 10x10 -->
<div id="levels10x10" class="stages"></div>
<!-- Stages 15x15 -->
```

```
<div id="levels15x15" class="stages"></div>
```

Το div tag element με id **correct-singleplayer** θα εμφανίζεται όταν ο χρήστης έχει ολοκληρώσει σωστά μία πίστα ενημερώνοντας των ότι έχει λύσει την πίστα και ρωτώντας των άμα θέλει να γυρίσει πίσω στο μενού ή να ξανά παίξει την πίστα από την αρχή. Θα μιλήσουμε γι αυτό στο κεφάλαιο με τις πίστες για να υπάρχει μία πιο ολοκληρωμένη εικόνα του κώδικα.

```
<div id="correct-singleplayer" class="black-screen"></div>
```

Το div tag element με id **instructions** θα περιέχει τις οδηγίες του παιχνιδιού μας, τι είναι το nonogram και το τι κάνουν τα εργαλεία των πιστών. Έχουμε φτιάξει και ένα αυτοσχέδιο exit button με το οποίο θα κλείνουμε το παράθυρο των οδηγιών.

```
<!-- How to play -->
<div id="instructions">
   <div id="wrapper">
         <h3>How to Play</h3>

         <p>Nonograms, also known as Picross or Griddlers, are picture logic puzzles in which cells in a grid must be colored or left blank according to numbers at the side of the grid to reveal a hidden picture. In this puzzle type, the numbers are a form of discrete tomography that measures how many unbroken lines of filled-in squares there are in any given row or column. For example, a clue of "4 8 3" would mean there are sets of four, eight, and three filled squares, in that order, with at least one blank square between successive groups.</p>

         <p>These puzzles are often black and white--describing a binary image--but they can also be colored. If colored, the number clues are also colored to indicate the color of the squares. Two differently colored numbers may or may not have a space in between them. For example, a black four followed by a red two could mean four black boxes, some empty spaces, and two red boxes, or it could simply mean four black boxes followed immediately by two red ones.</p>

         <h3>Tools</h3>

         <p><b>Default</b> <img src="img/default.png">: With this tool you can use a black cell, a x mark and a white cell. The x mark can help you to mark the cells that you know that they can't be black</p>
```

```
            <p><b>Black cell</b> <img src="img/black.png">: With this tool you can use only a black cell.</p>

            <p><b>X mark</b> <img src="img/x.png">: With this tool you can use only a x mark.</p>

            <p><b>White cell</b> <img src="img/white.png">: With this tool you can use only a white cell.</p>

            <p><b>Undo</b> <img src="img/undo.png">: With this tool you can undo your choice. it can store the last 10 choices you make.</p>

        <p><b>Redo</b> <img src="img/redo.png">: With this tool you can redo your choice.</p>

            <p><b>Clear</b> <img src="img/clear.png">: With this tool you can clear your procress.</p>

            <p><b>Help</b> <img src="img/help.png">: With this tool you can find out a black cell that is correct. You can use 3 times in a stage</p>

            <p><b>Home</b> <img src="img/home.png">: With this tool you can go back to the menu.</p>
    </div>

    <div id="close-instructions">X</div>
</div>
```

Προσθέτουμε την αντίστοιχη css στο αρχείο **home.css**

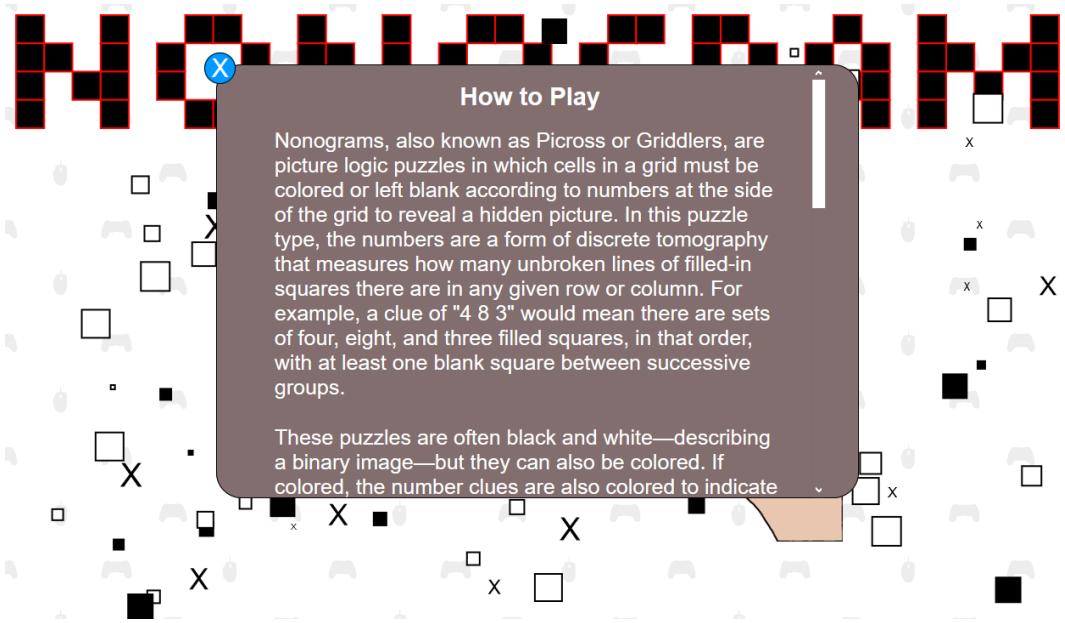
```
#instructions {
  display: none;
  position: absolute;
  top: 40%;
  left: 50%;
  transform: translate(-50%, -40%);
  height: 70%;
  background: #826e6e;
  font-size: 25px;
```

```
  border: 1px solid black;
  border-radius: 30px;
  color: white;
  font-family: Arial;
  scrollbar-color: white #826e6e;

  max-width: 900px;

  min-width: 700px;
}

#instructions h3 {
  margin: 20px;
  text-align: center;
}

#instructions p {
  margin: 0px 30px 30px 30px;
  display: block;
}

#wrapper {
  overflow-y: auto;
  width: 90%;
  height: 100%;
  margin: 0 auto;
}

#close-instructions {
  display: inline-block;
  position: absolute;
  background: #0099ff;
  border-radius: 40px;
  border: 1px solid black;
  top: -15px;
  left: -15px;
```

```
  font-size: 30px;
  font-family: "Arial";
  width: 35px;
  height: 35px;
  text-align: center;
  cursor: pointer;
  color: white;
}
```

και προσθέτουμε στο αρχείο **main-menu.js** την javascript που χρειαζόμαστε για το κουμπί που κλείνει τις οδηγίες.

```
$("#close-instructions").click(function() {
    $(this).parent().hide();
    $('#menu').show();
});
```



Αυτή είναι η υλοποίηση μας για το μενού του παιχνιδιού και για το κομμάτι του singleplayer, στο κεφάλαιο του multiplayer θα προσθέσουμε των κώδικα για τις επιλογές του multiplayer.

Η επόμενη υλοποίηση που θα κάνουμε θα αφορά τις πίστες μας, πως δημιουργούνται και πως τις επιλέγουμε.

7. Επιλογή και δημιουργία πιστών

Θέλουμε όταν ο χρήστης επιλέξει μία πίστα τότε αυτή να δημιουργείτε στον canva. Στο φάκελο js δημιουργούμε το αρχείο **stages.js** που θα έχουμε τις πίστες μας για κάθε κατηγορία 5x5, 10x10 και 15x15.

Τοποθετούμε το script **stages.js** στο αρχείο **index.html** πριν από το script του main-menu.

```
<script src="js/stages.js"></script>
```

Φτιάχνουμε 3 μεταβλητές για κάθε κατηγορία πιστών που θα είναι object οι οποίες θα περιέχουν ιδιότητες με τα ονόματα των πιστών.

```
let levels5x5 = {
      'chess':    [[1,0,1,0,1],[0,1,0,1,0],[1,0,1,0,1],[0,1,0,1,0],[1,0,1,0,1]],
      'snake':    [[1,1,1,1,1],[1,0,0,0,0],[1,1,1,1,1],[0,0,0,0,1],[1,1,1,1,1]],
      'airplane': [[0,0,1,0,0],[0,1,1,1,0],[1,1,1,1,1],[0,0,1,0,0],[0,1,1,1,0]],
      'smile':    [[1,1,0,1,1],[1,1,0,1,1],[0,0,0,0,0],[1,0,0,0,1],[0,1,1,1,0]]
};

let levels10x10 = {
      'questionmark': [[0,0,1,1,1,1,1,1,0,0],
                  [0,1,1,0,0,0,0,1,1,0],
                  [0,1,1,0,0,0,0,1,1,0],
                  [0,0,0,0,0,0,0,1,1,0],
                  [0,0,0,0,0,0,1,1,1,0],
                  [0,0,0,0,1,1,1,1,0,0],
                  [0,0,0,0,1,1,0,0,0,0],
                  [0,0,0,0,0,0,0,0,0,0],
                  [0,0,0,0,1,1,0,0,0,0],
                  [0,0,0,0,1,1,0,0,0,0]],

      'leaf':     [[0,0,0,0,1,1,1,1,1,1],
                  [0,0,0,1,0,1,0,1,0,1],
                  [0,0,1,1,0,1,0,1,1,0],
                  [0,1,0,1,0,1,1,1,1,0],
                  [0,1,0,1,0,1,1,1,1,0],
```

```
                [0,1,0,1,1,0,0,0,1,0],
                [0,1,1,1,1,1,1,1,1,0],
                [0,0,1,0,0,0,0,1,0,0],
                [0,1,0,1,1,1,1,0,0,0],
                [1,1,0,0,0,0,0,0,0,0]],

      'music':    [[0,0,0,0,0,0,1,1,1,1],
                [0,0,0,1,1,1,0,0,0,1],
                [0,0,0,1,0,0,0,1,1,1],
                [0,0,0,1,1,1,1,0,0,1],
                [0,0,0,1,0,0,0,0,0,1],
                [0,0,0,1,0,0,0,1,1,1],
                [0,1,1,1,0,0,1,1,1,1],
                [1,1,1,1,0,0,1,1,1,1],
                [1,1,1,1,0,0,0,1,1,0],
                [0,1,1,0,0,0,0,0,0,0]],

      'tree':     [[0,0,1,1,1,1,1,1,0,0],
                [0,1,1,1,1,0,1,1,1,0],
                [1,1,1,1,1,0,1,1,1,1],
                [1,1,0,1,1,1,0,0,1,1],
                [1,1,1,0,1,1,1,1,1,0],
                [0,1,1,1,1,1,1,0,0,0],
                [0,0,0,0,1,1,1,0,0,0],
                [0,0,0,0,1,1,0,0,0,0],
                [0,1,0,0,1,1,0,0,0,1],
                [1,1,1,1,1,1,1,1,1,1]]
};

let levels15x15 = {
      'dolphin' : [[0,0,0,0,0,0,1,1,0,0,0,0,0,0,0],
                [0,0,0,0,0,0,0,1,1,0,0,0,0,0,0],
                [0,0,0,0,1,1,1,1,1,1,1,1,1,0,0],
                [0,0,1,1,1,1,1,1,1,1,1,1,1,0,1,0],
                [1,1,1,1,1,1,1,1,1,1,1,1,1,0,1],
```

```
            [1,1,1,1,1,1,1,1,1,1,1,1,1,1,1],
            [1,1,1,1,1,1,1,0,1,1,0,1,1,1,1],
            [1,1,1,0,0,0,0,1,1,0,0,0,1,1,0],
            [1,1,0,0,0,0,0,0,0,0,0,0,0,1,0],
            [1,0,0,0,0,0,0,0,0,0,0,0,0,0,0],
            [1,1,0,0,0,0,0,0,0,0,0,0,0,1,1],
            [0,1,0,0,0,0,0,0,0,0,0,0,0,1,1],
            [0,0,0,1,1,0,1,1,0,1,0,1,1,0,0],
            [0,0,1,0,0,1,1,0,1,1,0,1,1,1,0],
            [1,1,1,0,1,1,1,0,1,1,1,0,1,1,1]],

    'alarm' : [[0,0,1,1,1,0,0,0,0,0,1,1,1,0,0],
          [0,1,1,1,0,0,1,1,1,0,0,1,1,1,0],
          [1,1,1,0,1,1,1,0,1,1,1,0,1,1,1],
          [1,1,0,1,1,1,1,0,1,1,1,1,0,1,1],
          [1,0,1,1,1,1,1,1,1,1,1,0,1,1,0,1],
          [0,0,1,1,1,1,1,1,1,1,0,1,1,1,0,0],
          [0,1,1,1,1,1,1,1,1,1,0,1,1,1,1,0],
          [0,1,0,0,1,0,0,0,1,1,1,0,0,1,0],
          [0,1,1,1,1,1,1,1,1,1,1,1,1,1,0],
          [0,0,1,1,1,1,1,1,1,1,1,1,1,0,0],
          [0,0,1,1,1,1,1,1,0,1,1,1,1,1,0,0],
          [0,0,0,1,1,1,1,1,0,1,1,1,1,0,0,0],
          [0,0,1,0,0,1,1,1,1,1,0,0,1,0,0],
          [0,1,0,0,1,0,1,1,1,0,1,0,0,1,0],
          [0,0,1,1,0,0,0,0,0,0,0,1,1,0,0]],

    'more-music' :  [[1,1,1,1,1,1,1,1,1,1,1,1,1,1,1],
            [1,1,1,1,0,1,1,1,1,1,1,1,1,1,1],
            [1,1,1,0,1,0,1,1,1,1,1,1,1,1,1],
            [0,0,1,0,1,0,1,0,0,0,0,0,0,0,0],
            [1,1,1,0,1,0,1,1,1,1,1,1,1,0,1],
            [0,1,1,0,0,1,1,0,0,0,0,0,0,0,0],
            [1,1,0,0,1,1,1,1,1,1,0,1,1,0,1],
            [1,0,1,0,0,1,1,0,0,0,0,0,0,0,0],
```

```
                [1,0,0,0,1,0,1,1,1,1,0,1,0,1,1],
                [1,0,1,0,1,0,1,0,0,0,0,0,0,0,0],
                [1,1,0,0,0,1,1,1,1,0,1,1,1,1,1],
                [1,1,1,0,1,1,0,0,0,0,0,0,0,0,0],
                [1,0,1,0,1,1,1,1,1,1,1,1,1,1,1],
                [1,1,0,1,1,1,1,1,1,1,1,1,1,1,1],
                [1,1,1,1,1,1,1,1,1,1,1,1,1,1,1]]
};
```

Θα χρειαστούμε 2 μεταβλητές την **currentStage** στην οποία θα αποθηκεύουμε το όνομα της πίστας και την **nonogram** η οποία θα περιέχει το παιχνίδι μας. Όταν ο χρήστης επιλέξει μία πίστα τότε θα καλείται η συνάρτηση **createLevel** η οποία θα έχει 2 παραμέτρους την **level** που αντιπροσωπεύει το grid της πίστας και την **stage** που αντιπροσωπεύει το όνομα της πίστας. Κατά την δημιουργία της πίστας μας αλλάζουμε την css του container και του canva μας για να έχουμε το επιθυμητό αποτέλεσμα. Η συνάρτηση θα δουλεύει ως εξής: Αποθηκεύουμε το όνομα της πίστας στην μεταβλητή currentStage, αυτή την μεταβλητή θα την χρειαστούμε για την αποθήκευση της προόδου της πίστας που έχει κάνει ο χρήστης, στην μεταβλητή nonogram δημιουργούμε την πίστα και στην συνέχεια τοποθετούμε την css που θέλουμε για να κεντράρουμε το container για να είναι ο canvas στο κέντρο της οθόνης, δίνουμε στον canva το πλάτος και το ύψος της πίστας μας και του ορίζουμε ένα μαύρο περίγραμμα και στην συνέχεια καθαρίζουμε τον canva και ζωγραφίζουμε την πίστα.

```
let currentStage;
let nonogram;

function createLevel(level, stage) {
    state = "level";
    currentStage = stage;
    nonogram = new Nonogram(level);
    container.style.left = "50%";
    container.style.top = "50%";
    container.style.transform = "translate(-50%, -50%)";
    canvas.width = nonogram.width;
    canvas.height = nonogram.height;
    canvas.style.border = "1px solid black";
    clearCanvas();
```

```
    nonogram.drawGrid();
    nonogram.drawRowNumbers();
    nonogram.drawColumnNumbers();
}
```

Στην συνέχεια δημιουργούμε τις πίστες για κάθε κατηγορία και τις τοποθετούμε στα αντίστοιχα div tag elements που είχαμε φτιάξει προηγουμένως στο μενού του παιχνιδιού.

```
let stages5x5 = document.getElementById('levels5x5');
let stages5x5Header = document.createElement('h3');
let stages10x10 = document.getElementById('levels10x10');
let stages10x10Header = document.createElement('h3');
let stages15x15 = document.getElementById('levels15x15');
let stages15x15Header = document.createElement('h3');
stages5x5Header.innerHTML = "5x5";
stages5x5.append(stages5x5Header);
stages10x10Header.innerHTML = "10x10";
stages10x10.append(stages10x10Header);
stages15x15Header.innerHTML = "15x15";
stages15x15.append(stages15x15Header);

Object.keys(levels5x5).forEach(key => {
    let stage = document.createElement('div');
    stage.classList.add('stage');
    stage.addEventListener('click', () => {
            createLevel(levels5x5[key], key);
    });
    let image = new Image();
    image.classList.add('correct-'+key);
    image.src = 'img/green-button.png';
    stage.innerHTML = key.toUpperCase();

    stage.append(image);
    stages5x5.append(stage);
});
let closeLevels5x5 = document.createElement('div');
```

```
closeLevels5x5.classList.add('close-levels5x5');
closeLevels5x5.innerHTML = 'Close';
stages5x5.append(closeLevels5x5);

Object.keys(levels10x10).forEach(key => {
    let stage = document.createElement('div');
    stage.classList.add('stage');
    stage.addEventListener('click', () => {
            createLevel(levels10x10[key], key);
    });
    let image = new Image();
    image.classList.add('correct-'+key);
    image.src = 'img/green-button.png';
    stage.innerHTML = key.toUpperCase();

    stage.append(image);
    stages10x10.append(stage);
});
let closeLevels10x10 = document.createElement('div');
closeLevels10x10.classList.add('close-levels10x10');
closeLevels10x10.innerHTML = 'Close';
stages10x10.append(closeLevels10x10);

Object.keys(levels15x15).forEach(key => {
    let stage = document.createElement('div');
    stage.classList.add('stage');
    stage.addEventListener('click', () => {
            createLevel(levels15x15[key], key);
    });
    let image = new Image();
    image.classList.add('correct-'+key);
    image.src = 'img/green-button.png';
    stage.innerHTML = key.toUpperCase();
    stage.append(image);

    stages15x15.append(stage);
```

```
});
let closeLevels15x15 = document.createElement('div');
closeLevels15x15.classList.add('close-levels15x15');
closeLevels15x15.innerHTML = 'Close';
stages15x15.append(closeLevels15x15);
```

Προσθέτουμε την αντίστοιχη css για την μορφοποίηση που χρειαζόμαστε στο αρχείο **level.css**.

```
#levels {
  display: none;
  border: 3px solid black;
  position: absolute;
  text-align: center;
  margin: 0 auto;
  top: 50%;
  left: 50%;
  transform: translate(-50%, -50%);
  background: #826e6e;
  font-weight: bold;
  font-size: 25px;
  font-family: Arial;
  box-shadow: 3px 3px 7px grey;
  width: 250px;
  color: white;
}

#levels h3 {
  padding-top: 15px;
  padding-bottom: 15px;
  margin: 0;
  cursor: default;
}

.level {
  background: #826e6e;
```

```
  border-top: 1px solid black;
  padding: 10px;
  cursor: pointer;
}

.level:hover {
  background: linear-gradient(to bottom right, #e0e0d1, grey, #cccccc);
  color: white;
}

#close-levels {
  cursor: pointer;
  padding-top: 15px;
  padding-bottom: 15px;
  border-top: 1px solid black;
}

.stages {
  display: none;
  border: 3px solid black;
  position: absolute;
  text-align: center;
  background: #826e6e;
  font-weight: bold;
  font-size: 25px;
  font-family: Arial;
  box-shadow: 3px 3px 7px grey;
  width: 250px;
  color: white;
}

.stages h3 {
  padding-top: 10px;
  padding-bottom: 10px;
  margin: 0;
```

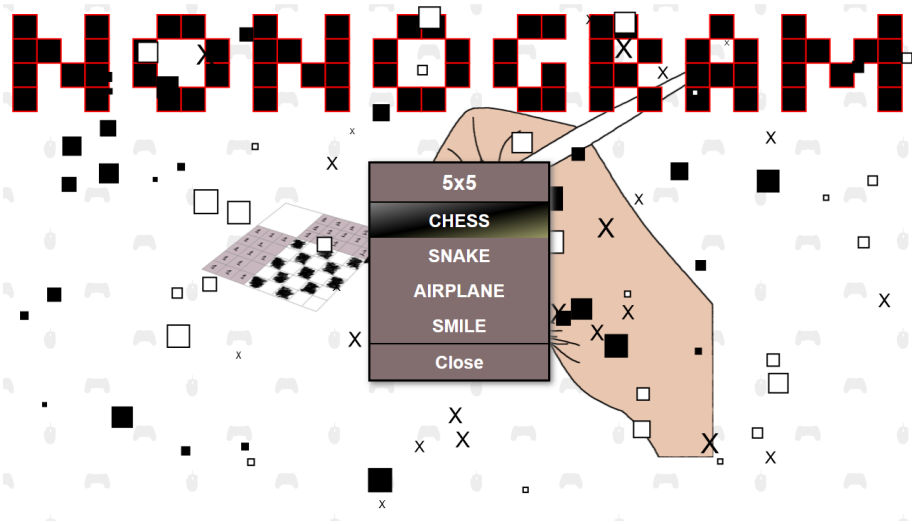
```
  cursor: default;
  border-bottom: 2px solid black;
}

.stage {
  background: #826e6e;
  cursor: pointer;
  padding: 10px;
}

.stage:hover {
  color: white;
  background: linear-gradient(to bottom right, grey, black, #999966);
}

.stage img{
  display: none;
  width: 15px;
  height: 15px;
  padding: 0px;
  padding-left: 10px;
  margin: 0px;
  position:absolute;
  left: 0;
  padding-top: 6px;
}

#levels5x5, #levels10x10, #levels15x15 {
  display: none;
  top: 50%;
  left: 50%;
  transform: translate(-50%, -50%);
}

#close-levels5x5, #close-levels10x10, #close-levels15x15 {
```

```
  cursor: pointer;
  border-top: 2px solid black;
  padding-top: 10px;
  padding-bottom: 10px;
}
```

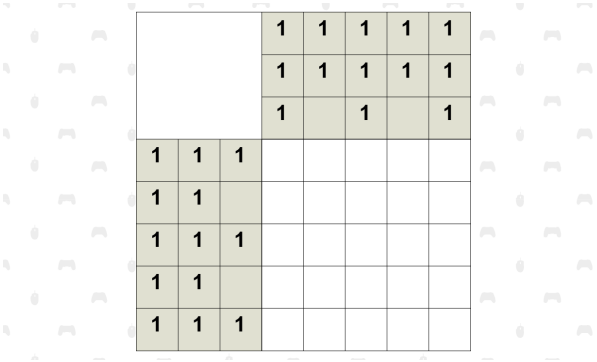
και προσθέτουμε στο αρχείο **main-menu.js** την javascript για την αλληλεπίδραση του χρήστη με το μενού των πιστών.

```
$(".close-levels5x5").click(function(){
    $("#levels5x5").hide();
    $("#levels").show();
});

$(".close-levels10x10").click(function(){
    $("#levels10x10").hide();
    $("#levels").show();
});

$(".close-levels15x15").click(function(){
    $("#levels15x15").hide();
    $("#levels").show();
});

$(".stage").click(function(){
    $("#levels5x5").hide();
    $("#levels10x10").hide();
    $("#levels15x15").hide();
    $("#container-tools").show();
});
```

Έχουμε φτιάξει το εξής αποτέλεσμα



και όταν ο χρήστης επιλέξει μία πίστα τότε το state του παιχνιδιού αλλάζει σε level, σταματάει το animation που είχαμε στο μενού, η αρχική μας οθόνη καθαρίζεται και δημιουργείται η πίστα μας.



Η επόμενη υλοποίηση που θα κάνουμε είναι να φτιάξουμε τα εργαλεία που θα μας βοηθήσουν να λύσουμε την πίστα π.χ αναίρεση, καθαρισμό της πίστας, βοήθεια κα.

8. Εργαλεία παιχνιδιού

Τοποθετούμε στο αρχείο **index.html** μέσα στο **div** με id **game** που φτιάξαμε, κάτω από τον canvas ένα καινούριο div με id=”container-tools” στο οποίο θα βάλουμε τα εργαλεία και άλλες λειτουργίες στην συνέχεια. Δημιουργούμε μία unordered list (ul) με id=”tools” που θα περιέχει τα εργαλεία.

```
<div id="container-tools">
  <ul id="tools"></ul>
</div>
```

Δημιουργούμε στον φάκελο **js** ένα καινούργιο αρχείο το **tools.js** στο οποίο θα έχουμε των κώδικα που χρειαζόμαστε για την δημιουργία, την λειτουργικότητα και την αλληλεπίδραση του χρήστη με αυτά**.**

Τοποθετούμε το νέο μας script στο **index.html.**

```
<script src="js/tools.js"></script>
```

Στο αρχείο **tools.js** φτιάχνουμε μία συνάρτηση με όνομα **createSinglePlayerTools.**

```
function createSinglePlayerTools() { ... }
```

Δημιουργούμε 4 σταθερές μεταβλητές την **1)** singlePlayerTools που θα έχει τα εργαλεία που χρειάζεται ο χρήστης για να επιλέγει τα κελιά, την **2)** singlePlayerExtraTools που θα έχει τα βοηθητικά εργαλεία (όπως επικόλληση αναίρεση, καθαρισμός της πίστας, βοήθεια και επιστροφή στο μενού), την **3)** tools που θα είναι το dom object του unordered list tag που φτιάξαμε πριν και την **4)** singleplayer στην οποία δημιουργούμε ένα καινούριο div element το οποίο θα έχει όλα τα tools του singleplayer.

```
const singlePlayerTools = ['default', 'black', 'x', 'white'];
const singlePlayerExtraTools = ['redo_undo', 'clear', 'help', 'home'];
const tools = document.getElementById("tools");
```

Δημιουργούμε ένα div tag element και του ορίζουμε το id singleplayer-tools και το εισάγουμε στο tools

```
const singleplayer = document.createElement('div');
singleplayer.id = "singleplayer-tools";
tools.appendChild(singleplayer);
```

Για κάθε ένα από τα singlePlayerTools που δημιουργήσαμε κάνουμε τα εξής:

```
for(let i=0; i<singlePlayerTools.length; i++) {
//Δημιουργούμε ένα li tag element και του ορίζουμε την class tool.
  var li = document.createElement('li');
  li.classList.add("tool");
//Δημιουργούμε ένα div tag element και του ορίζουμε σαν class το όνομα του εργαλείου που θα
αντιπροσωπεύει π.χ default.
  var div = document.createElement('div');
  div.className = singlePlayerTools[i];
```

```
//Δημιουργούμε ένα img tag element και του ορίζουμε το source μονοπάτι για την εικόνα του εργαλείου που θα αντιπροσωπεύει.
  var img = document.createElement('img');
  img.src = "img/" + singlePlayerTools[i] + ".png";
//Τέλος κάνουμε append το img tag στο li tag και το li tag στο singleplayer tag.
  div.appendChild(img);
  li.appendChild(div);
  singleplayer.appendChild(li);
}
```

Την ίδια υλοποίηση κάνουμε και για τα extra tools.

```
for(let i=0; i<singlePlayerExtraTools.length; i++) {
   var li = document.createElement('li');
   li.classList.add("extra-tool");
   var div = document.createElement('div');
   div.className = singlePlayerExtraTools[i];
   var img = document.createElement('img');
   img.src = "img/" + singlePlayerExtraTools[i] + ".png";
   div.appendChild(img);
   li.appendChild(div);
   singleplayer.appendChild(li);
}
```

Τελειώσαμε την συνάρτηση μας και την καλούμε στο ίδιο αρχείο για να φτιάξουμε τα εργαλεία.

```
createSinglePlayerTools();
```

Προσθέτουμε την css που χρειαζόμαστε για να τοποθετήσουμε τα εργαλεία στο σημείο που θέλουμε. Στον φάκελο css φτιάχνουμε ένα καινούργιο αρχείο με όνομα **tool.css** και βάζουμε το link tag στο **index.html**.

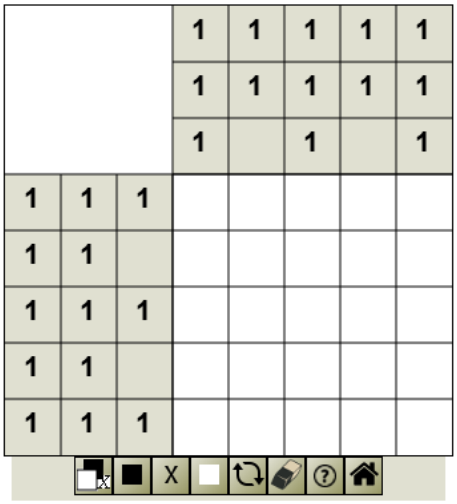
```
<link rel="stylesheet" type="text/css" href="css/tools.css">
```

```
#container-tools {
```

```
  display: none;
  position: relative;
  max-width: 326px;
  min-width: 326px;
  text-align: center;
  margin: 0 auto;
  background-color: #e0e0d1;
  min-height: 32.5px;
}

#tools {
  display: inline-block;
  list-style-type: none;
  padding: 0;
  margin: 0 auto;
}

#singleplayer-tools {
  display: none;
}

.tool, .extra-tool{
  display: list-item;
  float: left;
  border: 1px solid black;
  text-align: center;
  width: 27px;
  height: 27px;
  cursor: pointer;
  background: linear-gradient(to bottom right, #e0e0d1, #999966);
}

.black, .white, .x {
  padding-top: 5px;
  width: 100%;
```

```
  height: 100%;
}


.help {
  padding-top: 5px;
  height: 22px;
}
```

Με την html και την css που χρησιμοποιήσαμε ως τώρα έχουμε το εξής αποτέλεσμα:



Για να ξέρει ο χρήστης ποιό εργαλείο έχει ενεργό, θα αλλάξουμε το χρώμα του εργαλείου που έχει επιλέξει σε κόκκινο. Το default εργαλείο θα έχει πάντα στην αρχή κόκκινο χρώμα μέχρι να επιλέξει ο χρήστης κάποιο άλλο εργαλείο. Για να το πετύχουμε αυτό προσθέτουμε στο τέλος της συνάρτησης **createSinglePlayerTools** των κώδικα που προσθέτει την class active στο πρώτο εργαλείο που είναι το default

```
singleplayer.firstElementChild.classList.add("active");
```

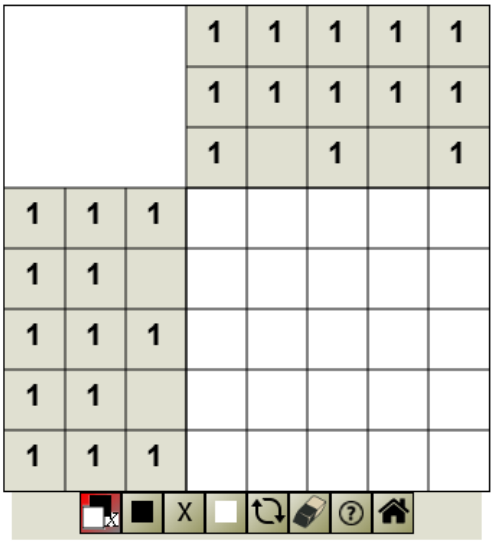
και στο αρχείο **tools.css** βάζουμε την css που χρειάζεται.

```
.active {
        background: linear-gradient(to bottom right, red, grey);
}
```

Κάθε φορά που θα επιλέγει ένα εργαλείο του single player τότε αυτό θα γίνεται active. Για να το κάνουμε αυτό τοποθετούμε των κώδικα στο αρχείο **tool.js** που θα επιτρέπει σε κάθε εργαλείο να εργαλείο του single player να γίνεται active όταν πατηθεί**.**

```
let singleplayer = document.getElementById("singleplayer-tools");
let singleplayerTools = singleplayer.getElementsByClassName("tool");

for (let i = 0; i < singleplayerTools.length; i++) {
  singleplayerTools[i].addEventListener("click", function() {
    let current = singleplayer.getElementsByClassName("active");
    if(typeof current[0] !== 'undefined') {
      current[0].className = current[0].className.replace(" active", "");
    }
    this.className += " active";
  });
}
```

Έχουμε φτιάξει αυτό το αποτέλεσμα ως τώρα.



Θέλουμε όταν ο χρήστης επιλέγει ένα εργαλείο πχ το black και στην συνέχεια επιλέξει να βγει από την πίστα και να παίξει κάποια άλλη να του έχει ενεργοποιημένο και επιλεγμένο το default tool.

Για να το κάνουμε αυτό προσθέτουμε στο αρχείο **tools.js** στο τέλος την συνάρτηση **resetTools**. Σαν παράμετρο του δίνουμε το container που περιέχει τα singleplayer tools, την ίδια διαδικασία θα κάνουμε και στο multiplayer.

```
function resetTools(toolContainer) {
  let singleplayer = document.getElementById("singleplayer-tools");
  let currentTool;
  let tools;

  if(toolContainer === "singleplayer") {
    currentTool = singleplayer.getElementsByClassName("active");
    tools = singleplayer.getElementsByClassName("tool");
    currentTool[0].className = currentTool[0].className.replace(" active", "");
    tools[0].className += " active";
  }
}
```

και την καλούμε στην συνάρτηση createLevel που βρίσκεται στο αρχείο **stages.js**, ακριβώς πριν των κώδικα που εμφανίζει τα single player tools..

```
resetTools("singleplayer");
$("#singleplayer-tools").show();
```

Στην συνέχεια θα προσθέσουμε την javascript που χρειάζεται για την αλληλεπίδραση του χρήστη με τα εργαλεία και το πως λειτουργούν.

8.1 Αλληλεπίδραση εργαλείων

Στο αρχείο **nonogram.js** προσθέτουμε μία καινούρια ιδιότητα στο object nonogram την **fillCellChoice** για να ξέρουμε ποιο εργαλείο έχει επιλέξει ο χρήστης. Θα έχουμε μόνιμα επιλεγμένο το default μέχρι να επιλέξει κάτι άλλο ο χρήστης.

```
this.fillCellChoice = "default";
```

Στο αρχείο **tools.js** τοποθετούμε των κώδικα που θα μας επιτρέψει να επιλέξουμε τα singlePlayerTools που είναι τα εργαλεία επιλογής των κελιών. Για αυτά τα εργαλεία θα υπάρχει ένας παρόμοιος κώδικας ο οποίος

δουλεύει ως εξής, αν ο χρήστης δεν έχει επιλέξει ήδη το εργαλείο τότε κάντο ενεργό και όρισε το στην ιδιότητα του **nonogram.fillCellChoice**.

```
$(".default").click(function(){
    if(nonogram.fillCellChoice !== "default") {
            nonogram.fillCellChoice = "default";
    }
});


$(".black").click(function(){
    if(nonogram.fillCellChoice !== "black") {
            nonogram.fillCellChoice = "black";
    }
});


$(".x").click(function(){
    if(nonogram.fillCellChoice !== "x") {
            nonogram.fillCellChoice = "x";
    }

});

$(".white").click(function(){
    if(nonogram.fillCellChoice !== "white") {
            nonogram.fillCellChoice = "white";
    }
});
```

Ανάλογα με το εργαλείο που έχει επιλέξει ο χρήστης εκτελείται και ο αντίστοιχος κώδικας.

8.2 Λειτουργικότητα εργαλείων

Για την λειτουργικότητα των εργαλείων ακολουθούμε την εξής λογική. Ελέγχουμε ποιό εργαλείο έχουμε ενεργό και στην συνέχεια ψάχνουμε όλα τα κελιά για να βρούμε το κελί που πατήθηκε με βάση τις

συντεταγμένες του ποντικιού που έγινε το click. Στην συνέχεια ελέγχουμε τι τιμή έχει η ιδιότητα **nonogram.emptyGrid[κελί].value** για να μάθουμε σε τι κατάσταση βρίσκεται το κελί και το ζωγραφίζουμε αναλόγως. Προσθέτουμε των κώδικα που χρειαζόμαστε στο αρχείο **nonogram.drawing.js** στην συνάρτηση fillCels. Αφαιρούμε την for που είχαμε γράψει για το μαρκάρισμα των κελιών την if που ακολουθεί για κάθε εργαλείο.

8.2.1 Εργαλείο default

Για το **default** εργαλείο έχουμε την εξής υλοποίηση.

```
if(this.fillCellChoice == "default") {
   for(var i=0;i<this.emptyGrid.length;i++) {
      if(mouseX >= this.emptyGrid[i].x && mouseY >= this.emptyGrid[i].y &&  mouseX <=
(this.emptyGrid[i].x + this.blockSize) && mouseY <= (this.emptyGrid[i].y + this.blockSize)) {
         if(this.emptyGrid[i].value == 0) { //paint the cell black
            this.emptyGrid[i].value = 1;
            this.drawWhiteCell(this.emptyGrid[i]);
            this.drawBlackCell(this.emptyGrid[i]);
            this.strokeCurrentChoice(this.emptyGrid[i]);
            this.drawPreview(this.emptyGrid[i]);
         }else if(this.emptyGrid[i].value == 1) { //fill the cell with an x
            this.emptyGrid[i].value = 2;
            this.drawWhiteCell(this.emptyGrid[i]);
            this.drawXCell(this.emptyGrid[i]);
            this.strokeCurrentChoice(this.emptyGrid[i]);
            this.drawPreview(this.emptyGrid[i]);
         }else { //Clear the cell
            this.emptyGrid[i].value = 0;
            this.drawWhiteCell(this.emptyGrid[i]);
            this.strokeCurrentChoice(this.emptyGrid[i]);
            this.drawPreview(this.emptyGrid[i]);
         }
         break; //Για να βγούμε από την for αφού έχουμε  βρεί το κελί
      }
```

```
  }
```

8.2.2 Εργαλείο black

Για την περίπτωση του εργαλείου **black:**

```
}else if(this.fillCellChoice == "black"){
  for(var i=0;i<this.emptyGrid.length;i++) {
    if(mouseX >= this.emptyGrid[i].x && mouseY >= this.emptyGrid[i].y && mouseX <=
(this.emptyGrid[i].x + this.blockSize) && mouseY <= (this.emptyGrid[i].y + this.blockSize)) {
      if(this.emptyGrid[i].value !== 1) {
        this.emptyGrid[i].value = 1;//fil the cell black
        this.drawWhiteCell(this.emptyGrid[i]);
        this.drawBlackCell(this.emptyGrid[i]);
        this.strokeCurrentChoice(this.emptyGrid[i]);
        this.drawPreview(this.emptyGrid[i]);
      }else{
        this.emptyGrid[i].value = 0;
        this.drawWhiteCell(this.emptyGrid[i]);
        this.strokeCurrentChoice(this.emptyGrid[i]);
        this.drawPreview(this.emptyGrid[i]);
      }
      break; //Για να βγούμε από την for αφού έχουμε  βρεί το κελί
    }
  }
```

8.2.3 Εργαλείο x

Για το εργαλείο **x :**

```
}else if(this.fillCellChoice == "x") {
  for(var i=0;i<this.emptyGrid.length;i++) {
    if(mouseX >= this.emptyGrid[i].x && mouseY >= this.emptyGrid[i].y && mouseX <=
```

```
(this.emptyGrid[i].x + this.blockSize) && mouseY <= (this.emptyGrid[i].y + this.blockSize)) {
        if(this.emptyGrid[i].value !== 2) {
            this.emptyGrid[i].value = 2;
            this.drawWhiteCell(this.emptyGrid[i]);
            this.drawXCell(this.emptyGrid[i]);
            this.strokeCurrentChoice(this.emptyGrid[i]);
            this.drawPreview(this.emptyGrid[i]);
        }else{
            this.emptyGrid[i].value = 0;
            this.drawWhiteCell(this.emptyGrid[i]);
            this.strokeCurrentChoice(this.emptyGrid[i]);
            this.drawPreview(this.emptyGrid[i]);
        }
        break; //Για να βγούμε από την for αφού έχουμε  βρεί το κελί
    }
```

8.2.4 Εργαλείο white

Για το εργαλείο **white :**

```
}else if(this.fillCellChoice == "white") {
   for(var i=0;i<this.emptyGrid.length;i++) {
      if(mouseX >= this.emptyGrid[i].x && mouseY >= this.emptyGrid[i].y && mouseX <=
(this.emptyGrid[i].x + this.blockSize) && mouseY <= (this.emptyGrid[i].y + this.blockSize)) {
        if(this.emptyGrid[i].value !== 0) {
            this.emptyGrid[i].value = 0;
            this.drawWhiteCell(this.emptyGrid[i]);
            this.strokeCurrentChoice(this.emptyGrid[i]);
            this.drawPreview(this.emptyGrid[i]);
        }
        break; //Για να βγούμε από την for αφού έχουμε  βρεί το κελί
     }
   }
};
```

Αυτοί είναι η υλοποίηση για τα singlePlayerTools.

Τώρα για τα singlePlayerExtraTools, έχουμε τις εξής υλοποιήσεις.

8.2.5 Εργαλείο redo undo tool

Για το **redo undo tool** που που μας επιτρέπει να αναιρούμε τις επιλογές μας, θέλουμε όταν ο χρήστης πατάει το εργαλείο να του εμφανίζει αλλά 2 εργαλεία το redo και το undo. Προσθέτουμε των κώδικα που θας μας δημιουργήσει το εργαλείο, στο κάτω μέρος της συνάρτησης **createSinglePlayerTools** ακριβώς μετά τα προηγούμενα εργαλεία που φτιάξαμε.

```
let expandRedoUndoTool = ["undo", "redo"];
let redo_undo_tool = document.getElementsByClassName("redo_undo")[0];
//Δημιουργούμε ένα καινούριο div element για την προέκταση του εργαλειου
let expand_redo_undo = document.createElement('div');
expand_redo_undo.className = 'expand';
for(let i=0; i<expandRedoUndoTool.length; i++) {
  let div = document.createElement('div');
  div.className = expandRedoUndoTool[i];
  let img = document.createElement('img');
  img.src = "img/" + expandRedoUndoTool[i] + ".png";
  div.appendChild(img);
  expand_redo_undo.appendChild(div)
}
redo_undo_tool.appendChild(expand_redo_undo);
```

Θέλουμε όταν ο χρήστης πατήσει το εργαλείο redo_undo να κάνει το χρώμα του πιο σκούρο και να μας εμφανίζει τα άλλα δύο εργαλεία. Για να το κάνουμε αυτό τοποθετούμε των κώδικα που χρειαζόμαστε στο αρχείο **tools.js**.

```
$(".redo_undo").click(function() {
  if($(".expand").is(":hidden")) {
        $(".redo_undo").css({"background": "linear-gradient(to bottom right, grey, #999966)"});
        $(".expand").show();
  }else{
```

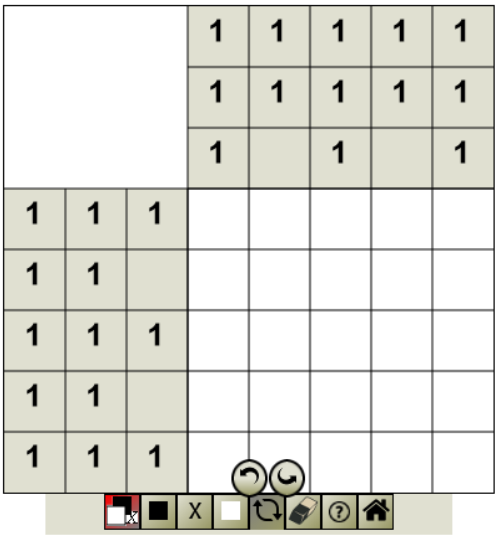
```
            $(".expand").hide();
            $(".redo_undo").css({"background": "linear-gradient(to bottom right, #e0e0d1, #999966)"});
    }
});
```

Προσθέτουμε στο αρχείο **tools.css** την css που χρειαζόμαστε για να είναι τα εργαλεία του expand στο σημείο που θέλουμε.

```
.redo_undo {
  display: block;
  position: relative;
}

.expand {
  display: none;
}

.redo {
  display: inline;
  width: 15px;
  height: 15px;
  background: linear-gradient(to bottom, #e0e0d1 50%, #999966 100%);
  margin-right: 1px;
  padding: 5px;
  border: 2px solid black;
  border-radius: 25px;
  position: absolute;
  top: -95%;
}

.undo {
  display: inline;
  width: 15px;
  height: 15px;
  background: linear-gradient(to bottom, #e0e0d1 50%, #999966 100%);
```

```
  margin-right: 1px;
  padding: 5px;
  border: 2px solid black;
  border-radius: 25px;
  left: -58%;
  top: -95%;
  position: absolute;
}

.redo:active, .undo:active {
  background: linear-gradient(to bottom right, red, grey);
}
```

Έχουμε αυτό το αποτέλεσμα ως τώρα, στην συνέχεια θα προσθέσουμε των κώδικα για την υλοποίηση των redo και undo tool.



Για αυτήν την υλοποίηση θα δημιουργήσουμε στο object Nonogram μία καινούρια μεταβλητή την cellChoices που θα είναι object , με την οποία θα μπορούμε να κρατήσουμε το ιστορικό των κινήσεων του χρήστη. Προσθέτουμε στο object **nonogram** την μεταβλητή cellChoices που έχει τις εξής ιδιότητες και μεθόδους.

```
this.cellChoices = {
   pastCells: [], newCells : [], index: 0,
   update   : function() {
      if(this.index < this.pastCells.length) {
```

```
            let limit = this.pastCells.length;
                for(let i=this.index; i<limit; i++) {
                    this.pastCells.pop();
                    this.newCells.pop();
                }
                this.index = this.pastCells.length;
        }
    }
};
```

Κάθε φορά που ο χρήστης θα πατάει ένα κελί τότε θα αποθηκεύουμε στον πίνακα pastCells που είναι ιδιότητα του object cellChoices, το παλιό value του κελιού και στην συνέχεια αποθηκεύουμε στον πίνακα newCells το νέο value του κελιού. Προσθέτουμε των κώδικα στην συνάρτηση fillCels σε κάθε if που ελέγχει το value του κελιού και αυξάνουμε την ιδιότητα index η οποία μας δείχνει ουσιαστικά το κελί που έχει επιλεχθεί τελευταίο.

Για την if της επιλογής **default**:

```
...

if(this.fillCellChoice == "default") {
  for(var i=0;i<this.emptyGrid.length;i++) {
    if(mouseX >= this.emptyGrid[i].x && mouseY >= this.emptyGrid[i].y &&
    mouseX <= (this.emptyGrid[i].x + this.blockSize) && mouseY <=
    (this.emptyGrid[i].y + this.blockSize)) {
        if(this.emptyGrid[i].value == 0) { //paint the cell black
            this.cellChoices.update();
            this.cellChoices.pastCells.push({cell: i, value: 0});
            this.emptyGrid[i].value = 1;
            this.drawWhiteCell(this.emptyGrid[i]);
            this.drawBlackCell(this.emptyGrid[i]);
            this.cellChoices.newCells.push({cell: i, value: 1});
            this.strokeCurrentChoice(this.emptyGrid[i]);
            this.drawPreview(this.emptyGrid[i]);
            this.cellChoices.index ++;
        }else if(this.emptyGrid[i].value == 1) { //fill the cell with an x
            this.cellChoices.update();
```

```
            this.cellChoices.pastCells.push({cell: i, value: 1});
            this.emptyGrid[i].value = 2;
            this.drawWhiteCell(this.emptyGrid[i]);
            this.drawXCell(this.emptyGrid[i]);
            this.cellChoices.newCells.push({cell: i, value: 2});
            this.strokeCurrentChoice(this.emptyGrid[i]);
            this.drawPreview(this.emptyGrid[i]);
            this.cellChoices.index ++;
        }else { //Clear the cell
             this.cellChoices.update();
            this.cellChoices.pastCells.push({cell: i, value: 2});
            this.emptyGrid[i].value = 0;
            this.drawWhiteCell(this.emptyGrid[i]);
            this.cellChoices.newCells.push({cell: i, value: 0});
            this.strokeCurrentChoice(this.emptyGrid[i]);
            this.drawPreview(this.emptyGrid[i]);
            this.cellChoices.index ++;
        }
        break;
    }
  }
}
```

Για την επιλογή του εργαλείου **black**:

```
else if(this.fillCellChoice == "black"){
   for(var i=0;i<this.emptyGrid.length;i++) {
      if(mouseX >= this.emptyGrid[i].x && mouseY >= this.emptyGrid[i].y &&

      mouseX <= (this.emptyGrid[i].x + this.blockSize) && mouseY <=

      (this.emptyGrid[i].y + this.blockSize)) {
               if(this.emptyGrid[i].value !== 1) {
                        this.cellChoices.update();
                        if(this.emptyGrid[i].value == 0) {
                           this.cellChoices.pastCells.push({cell: i, value: 0});
```

```
                    }else{
                        this.cellChoices.pastCells.push({cell: i, value: 2});
                    }
                    this.emptyGrid[i].value = 1;//fil the cell black
                    this.drawWhiteCell(this.emptyGrid[i]);
                    this.drawBlackCell(this.emptyGrid[i]);
                    this.cellChoices.newCells.push({cell: i, value: 1});
                    this.strokeCurrentChoice(this.emptyGrid[i]);
                    this.drawPreview(this.emptyGrid[i]);
                    this.cellChoices.index ++;
                }else{
                    this.cellChoices.update();
                    this.cellChoices.pastCells.push({cell: i, value: 1});
                    this.emptyGrid[i].value = 0;
                    this.drawWhiteCell(this.emptyGrid[i]);
                    this.cellChoices.newCells.push({cell: i, value: 0});
                    this.strokeCurrentChoice(this.emptyGrid[i]);
                    this.drawPreview(this.emptyGrid[i]);
                    this.cellChoices.index ++;
                }
                break; //Για να βγούμε από την for αφού έχουμε  βρεί το κελί
        }
    }
}
```

Για την επιλογή του εργαλείου X:

```
else if(this.fillCellChoice == "x") {
    for(var i=0;i<this.emptyGrid.length;i++) {
            if(mouseX >= this.emptyGrid[i].x && mouseY >= this.emptyGrid[i].y &&
        mouseX <= (this.emptyGrid[i].x + this.blockSize) && mouseY <=
        (this.emptyGrid[i].y + this.blockSize)) {
                    if(this.emptyGrid[i].value !== 2) {
                            this.cellChoices.update();
                            if(this.emptyGrid[i].value == 0) {
```

```
                        this.cellChoices.pastCells.push({cell: i, value: 0});
                    }else{
                        this.cellChoices.pastCells.push({cell: i, value: 1});
                    }
                    this.emptyGrid[i].value = 2;
                    this.drawWhiteCell(this.emptyGrid[i]);
                    this.drawXCell(this.emptyGrid[i]);
                    this.cellChoices.newCells.push({cell: i, value: 2});
                    this.strokeCurrentChoice(this.emptyGrid[i]);
                    this.drawPreview(this.emptyGrid[i]);
                    this.cellChoices.index ++;
                }else{
                    this.cellChoices.update();
                    this.cellChoices.pastCells.push({cell: i, value: 2});
                    this.emptyGrid[i].value = 0;
                    this.drawWhiteCell(this.emptyGrid[i]);
                    this.cellChoices.newCells.push({cell: i, value: 0});
                    this.strokeCurrentChoice(this.emptyGrid[i]);
                    this.drawPreview(this.emptyGrid[i]);
                    this.cellChoices.index ++;
                }
                break; //Για να βγούμε από την for αφού έχουμε  βρεί το κελί
            }
        }
    }
```

Για την επιλογή του εργαλείου white:

```
else if(this.fillCellChoice == "white") {
    for(var i=0;i<this.emptyGrid.length;i++) {
            if(mouseX >= this.emptyGrid[i].x && mouseY >= this.emptyGrid[i].y &&
        mouseX <= (this.emptyGrid[i].x + this.blockSize) && mouseY <=
        (this.emptyGrid[i].y + this.blockSize)) {
                    if(this.emptyGrid[i].value !== 0) {
```

```
                this.cellChoices.update();
                if(this.emptyGrid[i].value == 1) {
                        this.cellChoices.pastCells.push({cell: i, value: 1});
                }else{
                        this.cellChoices.pastCells.push({cell: i, value: 2});
                }
                this.emptyGrid[i].value = 0;
                this.drawWhiteCell(this.emptyGrid[i]);
                this.cellChoices.newCells.push({cell: i, value: 0});
                this.strokeCurrentChoice(this.emptyGrid[i]);
                this.drawPreview(this.emptyGrid[i]);
                this.cellChoices.index ++;
            }
            break; //Για να βγούμε από την for αφού έχουμε  βρεί το κελί
        }
    }
}
```

Με την ίδια λογική προσθέτουμε των ίδιο κώδικα και στην συνάρτηση fillMultiCells.

```
if((mouseX > x && (mouseX < x + this.blockSize)) || (mouseY > y && (mouseY < y + this.blockSize))) {
        for(var i=0;i<this.emptyGrid.length;i++) {
            if(mouseX >= this.emptyGrid[i].x && mouseY >= this.emptyGrid[i].y && mouseX
            <= (this.emptyGrid[i].x + this.blockSize) && mouseY <= (this.emptyGrid[i].y +
            this.blockSize)) {
                if(this.emptyGrid[i].x == x && this.emptyGrid[i].y == y) {
                    return;
                }else if(this.emptyGrid[i].x == this.currentChoice.cell.x &&
                    this.emptyGrid[i].y == this.currentChoice.cell.y) {
                    return;
                }
                this.cellChoices.pastCells.push({cell: i, value: this.emptyGrid[i].value});
                this.emptyGrid[i].value = startCellValue;
                if(startCellValue == 1) {
                    this.drawWhiteCell(this.emptyGrid[i]);
```

```
                this.drawBlackCell(this.emptyGrid[i]);
                this.drawPreview(this.emptyGrid[i]);
                this.currentChoice.cell = this.emptyGrid[i];
                this.previousChoice.cell.push(this.emptyGrid[i]);
                ctx.strokeStyle = "red";
                ctx.lineWidth   = 4;
                ctx.strokeRect(this.currentChoice.cell.x+5, this.currentChoice.cell.y+5,
                this.blockSize-10, this.blockSize-10);
                this.cellChoices.newCells.push({cell: i, value: 1});
                this.cellChoices.index ++;
            }else if(startCellValue == 2) {
                this.drawWhiteCell(this.emptyGrid[i]);
                this.drawPreview(this.emptyGrid[i]);
                this.drawXCell(this.emptyGrid[i]);
                this.currentChoice.cell = this.emptyGrid[i];
                this.previousChoice.cell.push(this.emptyGrid[i]);
                ctx.strokeStyle = "red";
                ctx.lineWidth   = 4;
                ctx.strokeRect(this.currentChoice.cell.x+5, this.currentChoice.cell.y+5,
                this.blockSize-10, this.blockSize-10);
                this.cellChoices.newCells.push({cell: i, value: 1});
                this.cellChoices.index ++;
            }else{
                this.drawWhiteCell(this.emptyGrid[i]);
                this.drawPreview(this.emptyGrid[i]);
                this.currentChoice.cell = this.emptyGrid[i];
                this.previousChoice.cell.push(this.emptyGrid[i]);
                ctx.strokeStyle = "red";
                ctx.lineWidth   = 4;
                ctx.strokeRect(this.currentChoice.cell.x+5, this.currentChoice.cell.y+5,
                this.blockSize-10, this.blockSize-10);
                this.cellChoices.newCells.push({cell: i, value: 1});
                this.cellChoices.index ++;
            }
        }
```

```
        }
    }
```

Προσθέτουμε στο αρχείο **tools.js** των κώδικα για την λειτουργία των **tools redo** και **undo**.

Στην περίπτωση που ο χρήστης πατήσει το εργαλείο undo και δεν υπάρχει κάποια επιλογή τότε κάνουμε return από την συνάρτηση. Παίρνουμε το προηγούμενο index και cell για να μπορέσουμε να ζωγραφίσουμε το προηγούμενο κελί αναλόγως με το value του κελιού. Χρησιμοποιούμε την μέθοδο stopPropagation του event για να εμποδίσουμε το το click να εκτελεστεί και για το parent element, γιατί άμα γίνει click στο undo ή redo τότε θα γίνει και στο parent element redo_undo με το οποίο θα κλείσει το expand.

```
$(".undo").click(function(event){
    event.stopPropagation();
    if(nonogram.cellChoices.index == 0) {
        return;
    }
    let index = nonogram.cellChoices.index-1;
    let cell = nonogram.cellChoices.pastCells[index].cell;
    if(nonogram.cellChoices.pastCells[index].value == 0) {
        //white cell
        nonogram.emptyGrid[cell].value = 0;
        nonogram.drawWhiteCell(nonogram.emptyGrid[cell]);
        nonogram.drawPreview(nonogram.emptyGrid[cell]);

        nonogram.strokeCurrentChoice(nonogram.emptyGrid[cell]);
        nonogram.cellChoices.index --;
    }else if(nonogram.cellChoices.pastCells[index].value == 1) {
        //black cell
        nonogram.emptyGrid[cell].value = 1;
       nonogram.drawWhiteCell(nonogram.emptyGrid[cell]);
       nonogram.drawBlackCell(nonogram.emptyGrid[cell]);
       nonogram.drawPreview(nonogram.emptyGrid[cell]);
       nonogram.strokeCurrentChoice(nonogram.emptyGrid[cell]);
       nonogram.cellChoices.index --;
    }else if(nonogram.cellChoices.pastCells[index].value == 2) {
        //x cell
```

```
        nonogram.emptyGrid[cell].value = 2;
        nonogram.drawWhiteCell(nonogram.emptyGrid[cell]);
        nonogram.drawXCell(nonogram.emptyGrid[cell]);
        nonogram.drawPreview(nonogram.emptyGrid[cell]);
        nonogram.strokeCurrentChoice(nonogram.emptyGrid[cell]);
        nonogram.cellChoices.index --;
    }
});
```

Παρόμοια υλοποίηση κάνουμε και στο **redo tool** με μόνη διαφορά την if που ελέγχουμε αν έχει γίνει τουλάχιστον μία φορά **undo** αλλιώς κάνουμε return και βγαίνουμε από την συνάρτηση.

```
$(".redo").click(function(event){
    event.stopPropagation();
    if(nonogram.cellChoices.index == nonogram.cellChoices.newCells.length) {
            return;
    }
    let index;
    index = nonogram.cellChoices.index;
    let cell = nonogram.cellChoices.newCells[index].cell;
    if(nonogram.cellChoices.newCells[index].value == 0) {
            //white cell
            nonogram.emptyGrid[cell].value = 0;
            nonogram.drawWhiteCell(nonogram.emptyGrid[cell]);
            nonogram.drawPreview(nonogram.emptyGrid[cell]);
            nonogram.strokeCurrentChoice(nonogram.emptyGrid[cell]);
            nonogram.cellChoices.index ++;
    }else if(nonogram.cellChoices.newCells[index].value == 1) {
            //black cell
            nonogram.emptyGrid[cell].value = 1;
            nonogram.drawWhiteCell(nonogram.emptyGrid[cell]);
            nonogram.drawBlackCell(nonogram.emptyGrid[cell]);
            nonogram.drawPreview(nonogram.emptyGrid[cell]);
            nonogram.strokeCurrentChoice(nonogram.emptyGrid[cell]);
            nonogram.cellChoices.index ++;
```

```
    }else if(nonogram.cellChoices.newCells[index].value == 2) {
            //x cell
            nonogram.emptyGrid[cell].value = 2;
            nonogram.drawWhiteCell(nonogram.emptyGrid[cell]);
            nonogram.drawXCell(nonogram.emptyGrid[cell]);
            nonogram.drawPreview(nonogram.emptyGrid[cell]);
            nonogram.strokeCurrentChoice(nonogram.emptyGrid[cell]);
            nonogram.cellChoices.index ++;
    }
});
```

8.2.6 Εργαλείο clear

Για το εργαλείο **clear:**

Μηδενίζουμε όλες τις επιλογές του χρήστη, τα κελιά και τα κελιά των αριθμών και στην συνέχεια καθαρίζουμε των canva και των ξανά ζωγραφίζουμε.

```
$(".clear").click(function() {
    for(let i=0; i<nonogram.emptyGrid.length; i++) {
            nonogram.emptyGrid[i].value = 0;
    }
    for(let i=0; i<nonogram.rowNumbersGrid.length; i++) {
            nonogram.rowNumbersGrid[i].value = 0;
    }
    for(let i=0; i<nonogram.columnNumbersGrid.length; i++) {
            nonogram.columnNumbersGrid[i].value = 0;
    }
    clearCanvas();
    nonogram.drawGrid();
    nonogram.drawRowNumbers();
    nonogram.drawColumnNumbers();
    nonogram.cellChoices.index = 0;
    nonogram.cellChoices.update();
});
```

8.2.7 Εργαλείο help

Για το εργαλείο **help:**

Φτιάχνουμε το object helpChoices το οποίο περιέχει 3 ιδιότητες που θα είναι πίνακες την wrong, την correct και την index. Η wrong περιέχει την λάθος επιλογή του χρήστη, η correct περιέχει την σωστή επιλογή του κελιού και η index περιέχει τον αριθμό του κελιού. Στην συνέχεια έχουμε 2 for στις οποίες βρίσκουμε τις λάθος επιλογές των κελιών και αποθηκεύουμε πάντα τις λάθος και σωστές επιλογές και των αριθμό που αντιπροσωπεύει το κελί.

Δημιουργούμε μία μεταβλητή την randomChoices την οποία θα χρησιμοποιήσουμε για να πάρουμε ένα τυχαίο λάθος από τα λάθη του χρήστη και στην συνέχεια ελέγχουμε το λάθος του χρήστη, αν είχε βάλει μαύρο κελί σε κελί που έπρεπε να είναι κενό ή αν είχε βάλει χ σε κελί που έπρεπε να είναι μαύρο. Και στις 2 if έχουμε παρόμοιες περιπτώσεις: αλλάζουμε την επιλογή που έκανε ο παίχτης στο κελί με την σωστή επιλογή, στην συνέχεια για να δείξουμε ότι είναι μία βοήθεια ζωγραφίζουμε ένα πράσινο περίγραμμα και χρησιμοποιώντας την συνάρτηση setTimeout δημιουργούμε ένα εφέ για να δείξουμε στον χρήστη ποιό κελί διορθώθηκε.

```
$(".help").click(function() {
    let helpChoices = { wrong: [], correct: [], index: [] };
    for(let i=0; i<nonogram.levelGrid.length; i++) {

        for(let y=0; y<nonogram.levelGrid[i].length; y++) {
            if(nonogram.levelGrid[i][y] === 1 &&

            nonogram.emptyGrid[(i*nonogram.levelGrid[0].length)+y].value === 2) {

                helpChoices.wrong.push(nonogram.emptyGrid[(i*nonogram.levelGrid[0].length)+y]);
                helpChoices.correct.push(nonogram.levelGrid[i][y]);
                helpChoices.index.push(i*nonogram.levelGrid[0].length+y);
            }else if(nonogram.levelGrid[i][y] === 0 &&

             nonogram.emptyGrid[(i*nonogram.levelGrid[0].length)+y].value === 1) {

                helpChoices.wrong.push(nonogram.emptyGrid[(i*nonogram.levelGrid[0].length)+y]);
                helpChoices.correct.push(nonogram.levelGrid[i][y]);
                helpChoices.index.push(i*nonogram.levelGrid[0].length+y);
             }
```

```
    }
}
let randomChoice = Math.floor(Math.random() * helpChoices.index.length);
if(helpChoices.correct[randomChoice] === 0 &&

helpChoices.wrong[randomChoice].value === 1) {

   nonogram.emptyGrid[helpChoices.index[randomChoice]].value = 2;

   ctx.strokeStyle = "green";
   ctx.lineWidth   = 4;
   ctx.strokeRect(nonogram.emptyGrid[helpChoices.index[randomChoice]].x+5,

    nonogram.emptyGrid[helpChoices.index[randomChoice]].y+5,

    nonogram.blockSize-10, nonogram.blockSize-10);
   setTimeout( function() {
          nonogram.drawWhiteCell(nonogram.emptyGrid[helpChoices.index[randomChoice]]);
          ctx.strokeStyle = "green";
          ctx.strokeRect(nonogram.emptyGrid[helpChoices.index[randomChoice]].x+5,

          nonogram.emptyGrid[helpChoices.index[randomChoice]].y+5, nonogram.blockSize-10,

          nonogram.blockSize-10);
   }, 500);
   setTimeout( function() {
      nonogram.drawXCell(nonogram.emptyGrid[helpChoices.index[randomChoice]]);
      ctx.strokeStyle = "green";
      ctx.strokeRect(nonogram.emptyGrid[helpChoices.index[randomChoice]].x+5,

       nonogram.emptyGrid[helpChoices.index[randomChoice]].y+5,

       nonogram.blockSize-10, nonogram.blockSize-10);
       }, 1000 );
       nonogram.drawPreview(nonogram.emptyGrid[helpChoices.index[randomChoice]]);
       ctx.strokeStyle = "black";
}else if(helpChoices.correct[randomChoice] === 1 &&

helpChoices.wrong[randomChoice].value === 2) {
```

```
        nonogram.emptyGrid[helpChoices.index[randomChoice]].value = 1;
        ctx.strokeStyle = "green";
        ctx.strokeRect(nonogram.emptyGrid[helpChoices.index[randomChoice]].x+5,

        nonogram.emptyGrid[helpChoices.index[randomChoice]].y+5,

        nonogram.blockSize-10, nonogram.blockSize-10);

        setTimeout( function() {
          nonogram.drawWhiteCell(nonogram.emptyGrid[helpChoices.index[randomChoice]]);
                ctx.strokeStyle = "green";
          ctx.strokeRect(nonogram.emptyGrid[helpChoices.index[randomChoice]].x+5,

          nonogram.emptyGrid[helpChoices.index[randomChoice]].y+5,

          nonogram.blockSize-10, nonogram.blockSize-10);
              }, 500);
              setTimeout( function() {
                nonogram.drawBlackCell(nonogram.emptyGrid[helpChoices.index[randomChoice]]);
                ctx.strokeStyle = "green";
                ctx.strokeRect(nonogram.emptyGrid[helpChoices.index[randomChoice]].x+5,

          nonogram.emptyGrid[helpChoices.index[randomChoice]].y+5,

          nonogram.blockSize-10, nonogram.blockSize-10);
              }, 1000 );
              nonogram.drawPreview(nonogram.emptyGrid[helpChoices.index[randomChoice]]);
              ctx.strokeStyle = "black";
    }
});
```

8.2.8 Εργαλείο home

Για το εργαλείο **home:**

```
$(".home").click(function(){
    $("#container-tools").hide();
```

```
    ctx.clearRect(0, 0, canvas.width, canvas.height);
    container.style.transform = "none";
    container.style.left = "0%";
    container.style.top = "0%";
    canvas.width = innerWidth;
    canvas.height = innerHeight;
    canvas.style.border = "none";
    state = "menu";
    $("#menu").show();
});
```

8.2.9 Εργαλείο progress

Το εργαλείο progress είναι ένα βοηθητικό εργαλείο το οποίο βοηθάει τον χρήστη να καταλάβει πόσα κελιά έχει επιλέξει. Υπολογίζουμε μόνο τα κελιά με μαύρο και X, τα λευκά κελιά δεν υπολογίζονται.

Στο αρχείο **index.html** προσθέτουμε στο div tag container tools μετά το ul tools των εξής κώδικα

```
<div id="info-right">
    <div id="info-progress">
        <span id="info-current-progress">0%</span>
    </div>
</div>
```

Προσθέτουμε την css στο αρχείο **tools.css.**

```
#info-right {
  display: inline-block;
  float: right;
}

#info-progress {
  width: 60px;
  height: 25px;
  font-family: 'Press Start 2P';
```

```
  padding: 2px;
  background-color: black;
  color: #ffcc00;
  text-align: right;
}
```

Χρησιμοποιούμε στο info-progress το font-family ‘Press Start 2P’ από το google fonts. Προσθέτουμε το link της css για το font στο **index.html**

```
<link href="https://fonts.googleapis.com/css?family=Press+Start+2P" rel="stylesheet">
```

Για να βρίσκουμε το progress της πίστας δημιουργούμε στο object nonogram την μέθοδο **findProgress**. Με αυτή την μέθοδο βρίσκουμε το progress ελέγχοντας το value όλων των κελιών και άμα είναι διάφορο του μηδενός τότε αυξάνουμε το progress κατά 1 και στο τέλος διαιρούμε το αποτέλεσμα με το πλήθος των κελιών της πίστας.

```
this.findProgress = function() {
    let progress = 0;
    for(let i=0; i<this.emptyGrid.length; i++) {
        if(this.emptyGrid[i].value != 0) {
            progress++;
        }
    }
    progress = (progress * 100) / this.emptyGrid.length;
    return Math.floor(progress);

};
```

Προσθέτουμε στα αρχεία **mouse.js** και **touch.js** στα events mousedown, mousemove, mouseup, touchstart και touchmove των κώδικα που χρειαζόμαστε για να υπολογίζεται και να ανανεώνεται το progress όταν ο χρήστης κάνει μία επιλογή.

```
$(canvas).mousedown(function(event) {
    ...
    nonogram.findProgress();
});
```

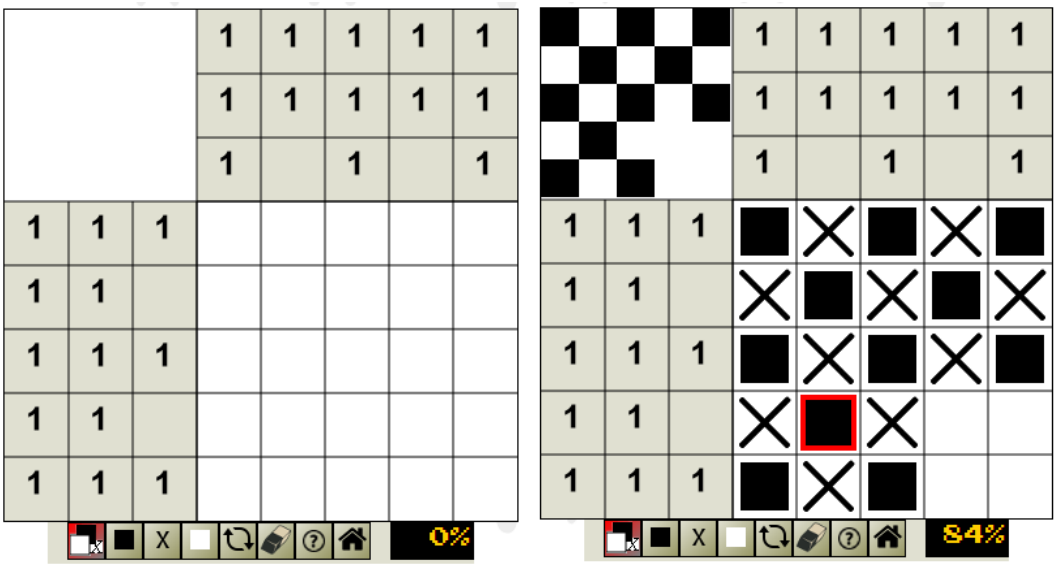
```
$(canvas).mouseup(function(){
    ...
    $("#info-current-progress").text("");
    $("#info-current-progress").text(nonogram.findProgress() + "%");
});

$(canvas).mousemove(function(event){
            if(isDown){
                ...
                $("#info-current-progress").text("");
                $("#info-current-progress").text(nonogram.findProgress() + "%");
            }
});

$(canvas).on('touchstart', function(event) {
    ...
    nonogram.findProgress();
    $("#info-current-progress").text("");
    $("#info-current-progress").text(nonogram.findProgress() + "%");
});

$(canvas).on('touchmove', function(event) {
    ...
    nonogram.findProgress();
    $("#info-current-progress").text("");
    $("#info-current-progress").text(nonogram.findProgress() + "%");
});
```

Προσθέτουμε και στα εργαλεία **redo**, **undo, clear** και **help** των κώδικα για να ανανεώνεται το progress όταν ο χρήστης χρησιμοποιήσει αυτά τα εργαλεία. Στο αρχείο **tools.js** που έχουμε των κώδικα για την αλληλεπίδραση με τα εργαλεία τοποθετούμε στο event click για τα εργαλεία που αναφέραμε, στο τέλος του κάθε event των κώδικα.

```
nonogram.findProgress();
```

```
$("#info-current-progress").text("");
$("#info-current-progress").text(nonogram.findProgress() + "%");
```

Έχουμε φτιάξει αυτό το αποτέλεσμα.



Στην συνέχεια θα μιλήσουμε για την αποθήκευση της προόδου του χρήστη στις πίστες.

9. Αποθήκευση της προόδου του παιχνιδιού

Για να αποθηκεύσουμε την πρόοδο που έχει κάνει ο χρήστης στο παιχνίδι θα χρησιμοποιήσουμε την τεχνολογία storage που έχει ο κάθε browser η οποία μας επιτρέπει να αποθηκεύουμε δεδομένα στον browser.

Ξεκινάμε δημιουργώντας στον φάκελο js ένα καινούργιο αρχείο το **store.js** και τοποθετούμε το script στο αρχείο **index.html** πριν το script του main-menu**.**

```
<script src="js/store.js"></script>
```

Στο αρχείο **nonogram.js** προσθέτουμε στο object nonogram μία ιδιότητα που θα είναι object την userChoices στην οποία θα αποθηκεύσουμε όλες τις επιλογές που έχει κάνει ο χρήστης στην πίστα και προσθέτουμε την συνάρτηση findUserChoices με την οποία θα βρίσκουμε και θα αποθηκεύουμε όλες τις επιλογές που έχει κάνει ο χρήστης στην πίστα, στις ιδιότητες του object userChoices.

```
this.userChoices = {};
this.userChoices.levelGrid = [];
```

```
this.userChoices.rowNumbersGrid = [];
this.userChoices.columnNumbersGrid = [];

this.findUserChoices = function() {
    for(let i = 0; i < this.emptyGrid.length; i++) {
        this.userChoices.levelGrid[i] = this.emptyGrid[i].value;
    }

    for(let i=0; i<this.rowNumbersGrid.length; i++) {
        this.userChoices.rowNumbersGrid[i] = this.rowNumbersGrid[i].value;
    }

    for(let i=0; i<this.columnNumbersGrid.length; i++) {
       this.userChoices.columnNumbersGrid[i] = this.columnNumbersGrid[i].value;
    }
}
```

Τοποθετούμε στο αρχείο **store.js** των κώδικα που χρειαζόμαστε για να κάνουμε την αποθήκευση.

Έχουμε 3 συναρτήσεις σε αυτό το αρχείο την **store** οι οποία ελέγχει αν ο browser υποστηρίζει το storage, αν το υποστηρίζει τότε ελέγχει αν δεν υπάρχει ήδη αποθηκευμένη πρόοδος της πίστας και τότε την δημιουργεί αλλιώς ανανεώνει την υπάρχων. Την συνάρτηση **retrieve** η οποία βρίσκει και παίρνει την πρόοδο της πίστας και την συνάρτηση **isCorrect** η οποία ελέγχει αν ο παίκτης έχει ολοκληρώσει αυτή την πίστα.

```
function store(level, progress) {
    // Check browser support
    if (typeof(Storage) !== "undefined") {
            if(!localStorage.getItem(level)){
                    localStorage.setItem(level, progress);
            }else{
                    localStorage[level] = progress;
            }
    } else {
            alert("Sorry, your browser does not support Web Storage...");
    }
};
```

```
function retrieve(level) {
    return localStorage.getItem(level).split(',').map(function(item){
            return parseInt(item, 10);
    });
};

function isCorrect(level) {
    if(localStorage.getItem(level) === "1") {
            return true;
    }
};
```

Δημιουργούμε στο object nonogram την συνάρτηση retrieveProgress με την οποία θα παίρνουμε τα δεδομένα της πίστας που έχουν αποθηκευτεί στο localstorage και θα τα προσθέτουμε στα δεδομένα της πίστας που έχουμε δημιουργήσει

```
this.retrieveProgress = function(levelGrid, rowNumbersGrid, columnNumbersGrid) {
    this.userChoices.levelGrid = levelGrid;
    this.userChoices.rowNumbersGrid = rowNumbersGrid;
    this.userChoices.columnNumbersGrid = columnNumbersGrid;
    for(let i=0; i < this.emptyGrid.length; i++) {
            this.emptyGrid[i].value = this.userChoices.levelGrid[i];
    }

    for(let i=0; i<this.rowNumbersGrid.length; i++) {
            this.rowNumbersGrid[i].value = this.userChoices.rowNumbersGrid[i];
    }

    for(let i=0; i<this.columnNumbersGrid.length; i++) {
            this.columnNumbersGrid[i].value = this.userChoices.columnNumbersGrid[i];
    }
};
```

και τοποθετούμε την συνάρτηση redrawProgress στο αρχείο **nonogram.drawing.js** με την οποία ξανά ζωγραφίζουμε την πίστα βασισμένοι στο progress που έχουμε ανακτήσει.

```
Nonogram.prototype.redrawProgress = function() {
  for(let i=0; i<this.emptyGrid.length; i++) {
    if(this.emptyGrid[i].value == 1){
            this.drawBlackCell(this.emptyGrid[i]);
            this.drawPreview(this.emptyGrid[i]);
          }else if(this.emptyGrid[i].value == 2) {
                this.drawWhiteCell(this.emptyGrid[i]);
                this.drawXCell(this.emptyGrid[i]);
                this.drawPreview(this.emptyGrid[i]);
          }
  }
  ctx.beginPath();
  ctx.strokeStyle = "red";
  ctx.lineWidth = 3;
  for(let i=0; i<this.rowNumbersGrid.length; i++) {
    if(this.rowNumbersGrid[i].value == 1) {
            ctx.moveTo(this.rowNumbersGrid[i].x+3,

        (this.rowNumbersGrid[i].y + this.blockSize)-3);
                ctx.lineTo((this.rowNumbersGrid[i].x + this.blockSize)-3,

        this.rowNumbersGrid[i].y+3);
          }
  }

  for(let i=0; i<this.columnNumbersGrid.length; i++) {
    if(this.columnNumbersGrid[i].value == 1) {
            ctx.moveTo(this.columnNumbersGrid[i].x+3,

        (this.columnNumbersGrid[i].y + this.blockSize)-3);
                ctx.lineTo((this.columnNumbersGrid[i].x + this.blockSize)-3,

        this.columnNumbersGrid[i].y+3);
          }
  }
  ctx.closePath();
```

```
    ctx.stroke();
};
```

Όταν δημιουργείται η πίστα ελέγχουμε αν ο χρήστης έχει ήδη αποθηκευμένη πρόοδο της πίστας, στην περίπτωση που δεν έχει απλά φτιάχνουμε την πίστα και στην περίπτωση που έχει καλούμε την συνάρτηση retrieveProgress για να πάρουμε την πρόοδο του χρήστη και καλούμε την συνάρτηση redrawProgress για να ζωγραφίσουμε την πρόοδο του χρήστη.

```
function createLevel(level, stage) {
            ...
            clearCanvas();
            if(!localStorage.getItem(currentStage)) {
                nonogram.drawGrid();
                nonogram.drawRowNumbers();
                nonogram.drawColumnNumbers();
            }else{
                nonogram.drawGrid();
                nonogram.drawRowNumbers();
                nonogram.drawColumnNumbers();
                nonogram.retrieveProgress(retrieve(currentStage),

            retrieve('rowNumbersGrid-'+currentStage),

            retrieve('columnNumbersGrid-'+currentStage));
                nonogram.redrawProgress();
            }
            ...

}
```

Την συνάρτηση **isCorrect** την εκτελούμε όταν δημιουργείται το menu του παιχνιδιου με την οποία βλέπουμε τις πίστες που έχει ολοκληρώσει ο χρήστης. Θα μιλήσουμε γι αυτή την υλοποίηση στο επόμενο κεφάλαιο.

Την συνάρτηση **store** την χρησιμοποιούμε όταν ο χρήστης κάνει επιλογές πάνω στην πίστα ή αν χρησιμοποιήσει κάποιο από τα extra tools.

Για τις επιλογές που κάνει ο χρήστης πάνω στην πίστα τοποθετούμε των κώδικα στα events mousedown, mouseup και mouseout,

```
$(canvas).mousedown(function(event) {
    ...
    nonogram.findUserChoices();
    store(currentStage, nonogram.userChoices.levelGrid);
    store('rowNumbersGrid-'+currentStage, nonogram.userChoices.rowNumbersGrid);
    store('columnNumbersGrid-'+currentStage, nonogram.userChoices.columnNumbersGrid);
    nonogram.findProgress();
});

$(canvas).mouseup(function(){
    isDown = false;
    nonogram.findUserChoices();
    store(currentStage, nonogram.userChoices.levelGrid);
    store('rowNumbersGrid-'+currentStage, nonogram.userChoices.rowNumbersGrid);
    store('columnNumbersGrid-'+currentStage, nonogram.userChoices.columnNumbersGrid);
    $("#info-current-progress").text("");
    $("#info-current-progress").text(nonogram.findProgress() + "%");
});

$(canvas).mouseout(function() {
    isDown = false;
    if(isDown){
            nonogram.findUserChoices();
            store(currentStage, nonogram.userChoices.levelGrid);
            store('rowNumbersGrid-'+currentStage, nonogram.userChoices.rowNumbersGrid);
            store('columnNumbersGrid-'+currentStage, nonogram.userChoices.columnNumbersGrid);
    }
});
```

Για τα extra tools τοποθετούμε των κώδικα στα click events των tools **undo, redo, clear** και **help**.

Προσθέτουμε των κώδικα στο τέλος κάθε event πριν από το κάλεσμα της συνάρτησης **findProgress.**

```
nonogram.findUserChoices();
store(currentStage, nonogram.userChoices.levelGrid);
store('rowNumbersGrid-'+currentStage, nonogram.userChoices.rowNumbersGrid);
```

```
store('columnNumbersGrid-'+currentStage, nonogram.userChoices.columnNumbersGrid);
```

Στην συνέχεια θα μιλήσουμε για το πως γίνεται ο έλεγχος της προόδου της πίστας που παίζει ο χρήστης και τι γίνεται όταν ο χρήστης λύσει σωστά την πίστα.

9.1 Έλεγχος της προόδου της πίστας

Για να κάνουμε τον έλεγχο της προόδου της πίστας προσθέτουμε στο object nonogram την μέθοδο checkProgress η οποία ελέγχει όλα τα κελιά της πίστας αν είσαι σωστά σε σχέση με την πίστα που έχουμε ορίσει.

```
this.checkProgress = function() {
  var index = 0;
    for(var i=0;i<this.levelGrid.length;i++) {
      for(var y=0;y<this.levelGrid[i].length;y++) {
            if(this.levelGrid[i][y] == 1 && this.emptyGrid[index].value == 1) {
              this.correct = true;
            }
            else if(this.levelGrid[i][y] == 0 && (this.emptyGrid[index].value == 0

             || this.emptyGrid[index].value == 2)){
              this.correct = true;
            }
            else{
              this.correct = false;
              return false;
            }
             index ++;
          }
    }
    if(this.correct == true) {
          return true;
    }
}
```

Την μέθοδο αυτή την καλούμε στο event mouseup του canvas δηλαδή όταν ο χρήστης έχει κάνει επιλογή κελιού. Προσθέτουμε των κώδικα που χρειαζόμαστε στο event click του mouseup και του touchend.

```
if(nonogram.checkProgress()) {
    $("#correct-singleplayer").show();
    store("correct-" + currentStage, 1);
    $(".correct-" + currentStage).show();
}else{
    $("#correct-singleplayer").hide();
    store("correct-" + currentStage, 0);
    $(".correct-" + currentStage).hide();
}
```

Εφόσον ο χρήστης έχει λύσει την πίστα θα του εμφανίζεται το εξής μενού.

Για να κάνουμε αυτό το αποτέλεσμα προσθέτουμε στο αρχείο **index.html** στο div tag correct-singleplayer τα tags μας. Έχουμε 2 κουμπιά το ένα είναι το restart με το οποίο ο χρήστης παίζει από την αρχή την πίστα και το continueGame με το οποίο ο χρήστης επιστρέφη στο μενού επιλογής πιστών.

```
<div id="correct-singleplayer" class="black-screen">
        <div id="correct-multiplayer-popup">
```

```
        <h2>Correct !</h2>
        <img src="img/green-button.png" style="width:60px; height:60px; margin: 0 auto;">
        <p id="restart">Restart</p>
        <p id="continueGame">Exit</p>
        </div>
</div>
```

Προσθέτουμε και την αντίστοιχη css στο αρχείο **home.css** για να έχουμε το οπτικό αποτέλεσμα που θέλουμε.

```
.black-screen {
  position: absolute;
  background-color: rgba(0, 0, 0, 0.6);
  top:0;
  width: 100%;
  height: 100%;
}

#correct-singleplayer {
  display: none;
}

#correct-popup {
  position: absolute;
  top: 50%;
  left: 50%;
  transform: translate(-50%, -50%);
  text-align: center;
  background: linear-gradient(to bottom right, #e0e0d1, white,  #999966);
  border: 2px solid black;
  border-radius: 30px;
  padding: 10px;
  box-shadow: 1px 3px 5px grey;
  width: 200px;
}
```

```css
#correct-popup p {
  padding: 10px;
  margin: 10px;
  cursor: pointer;
  font-weight: bold;
  background: linear-gradient(to bottom right, #e0e0d1, grey);
  border-radius: 30px;
  border: 1px solid black;
}

#correct-popup p:hover {
  background: linear-gradient(to bottom right, #e0e0d1, white);
}

#correct-popup h2 {
  position: relative;
  margin: 0 auto;
  padding: 0 auto;
  text-align: center;
}
```

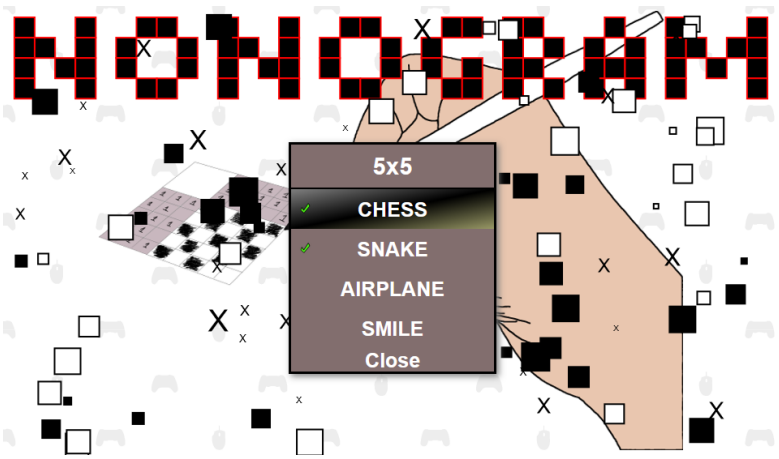
Για την υλοποίηση του restart και του continueGame προσθέτουμε των κώδικα στο αρχείο **tools.js**

```
$('#restart').click(function() {
   for(let i=0; i<nonogram.emptyGrid.length; i++) {
          nonogram.emptyGrid[i].value = 0;
   }

   for(let i=0; i<nonogram.rowNumbersGrid.length; i++) {
          nonogram.rowNumbersGrid[i].value = 0;
   }

   for(let i=0; i<nonogram.columnNumbersGrid.length; i++) {
          nonogram.columnNumbersGrid[i].value = 0;
```

```
    }

    ctx.clearRect(0,0,canvas.width, canvas.height);
    nonogram.drawGrid();
    nonogram.drawRowNumbers();
    nonogram.drawColumnNumbers();
    nonogram.findUserChoices();
    store(currentStage, nonogram.userChoices.levelGrid);
    store('rowNumbersGrid-'+currentStage, nonogram.userChoices.rowNumbersGrid);
    store('columnNumbersGrid-'+currentStage, nonogram.userChoices.columnNumbersGrid);
    store("correct-" + currentStage, 0);
    $("#correct-level-tools").hide();
    $("#singleplayer-tools").show();
    $("#correct-singleplayer").hide();
    $(".correct-" + currentStage).hide();
    $("#info-current-progress").text("");
    $("#info-current-progress").text(nonogram.findProgress() + "%");
});
```

```
$("#continueGame").click(function(){
    $("#container-tools").hide();
    ctx.clearRect(0, 0, canvas.width, canvas.height);
    container.style.transform = "none";
    container.style.left = "0%";
    container.style.top = "0%";
    canvas.width = innerWidth;
    canvas.height = innerHeight;
    canvas.style.border = "none";
    state = "menu";
    $("#correct-singleplayer").hide();
    $("#levels").show();
});
```

Εφόσον ο χρήστης έχει τελειώσει σωστά μία πίστα και για να μπορεί να ξέρει ποιές πίστες έχει τελειώσει θα προσθέσουμε μία εικόνα δίπλα από τις πίστες που έχει ολοκληρώσει.



Για να το κάνουμε αυτό προσθέτουμε των κώδικα που χρειαζόμαστε στα εξής σημεία:

1. Στο αρχείο **stage.js** προσθέτουμε την μεταβλητή **allStages** η οποία περιέχει τις ονομασίες όλων των πιστών.

```
let allStages = ['chess', 'snake', 'airplane', 'smile', 'questionmark', 'leaf', 'music', 'tree', 'dolphin', 'alarm', 'more-music'];
```

και τοποθετούμε στην συνάρτηση createLevel των κώδικα που θα ελέγχει αν ο χρήστης έχει ολοκληρωμένη την πίστα σωστά όταν την δημιουργήσει. Τοποθετούμε τον κώδικα πριν το κάλεσμα της συνάρτησης resetTools.

```
if(nonogram.checkProgress()) {
        $("#correct-singleplayer").show();
        $("#correct-level-tools").show();
}
resetTools("singleplayer");
```

2. Στο αρχείο **main-menu.js** τοποθετούμε των κώδικα στο τέλος του αρχείου, ο οποίος ελέγχει όλες τις πίστες και σε όσες έχουν ολοκληρωθεί εμφανίζει το εικονίδιο δίπλα από το όνομα ότι την έχει ολοκληρώσει.

```
for(let i=0; i<allStages.length; i++) {
        if(isCorrect("correct-" + allStages[i])) {
            $(".correct-" + allStages[i]).show();
        }
}
```

Αυτά είχαμε να πούμε για τον έλεγχο της προόδου του χρήστη, στο επόμενο κεφάλαιο θα μιλήσουμε για δύο έξτρα λειτουργίες οι οποίες θα μας βοηθήσουν στην επίλυση των πιστών.

10. Έξτρα λειτουργίες

Σε αυτό το κεφάλαιο θα μιλήσουμε για τις έξτρα λειτουργίες οι οποίες είναι η **zoom** και **drag**. Η zoom μας επιτρέπει να κάνουμε zoom στις πίστες για να μπορέσουμε να έχουμε μία καλύτερη οπτική της πίστας και η drag με την οποία θα μπορούμε να μετακινούμε την πίστα ανάλογα με το zoom που έχει γίνει. Αυτές οι λειτουργίες μας βοηθάνε κυρίως στην επίλυση μεγάλων πιστών και κυρίως στους χρήστες οι οποίοι παίζουν από smartphone και tablets. Το zoom και το drag για να δουλέψουν χρειάζονται το ένα το άλλο. Για να μπορέσεις να κάνεις drag πρέπει να έχεις κάνει zoom και για να μπορέσεις να δεις την πίστα όταν έχεις κάνει zoom χρειάζεται το drag. Θα ξεκινησουμε με τον κώδικα που χρειάζεται για την υλοποίηση του zoom και στην συνέχεια θα προσθέσουμε των κώδικα του drag.

10.1 Υλοποίηση λειτουργίας zoom

Ο μηχανισμός που θα ελέγχει την κλίμακα του zoom της πίστας είναι το scroll του ποντικιού, γι αυτό θα φτιάξουμε μια συνάρτηση η οποία θα καταγράφει την κίνηση του scroll. Αν ο χρήστης κάνει scroll προς τα πάνω τότε θα το υπολογίζουμε σαν zoom in και άμα κάνει scroll προς τα κάτω τότε θα κάνει zoom out.

Στον φάκελο controls φτιάχνουμε το αρχείο **zoom_drag.js** και προσθέτουμε το script στο index.html.

```
<script src="js/controls/zoom_drag.js"></script>
```

Για να το πετύχουμε αυτό φτιάχνουμε στο αρχείο **zoom_drag.j**s την συνάρτηση **handleScroll** η οποία θα παίρνει σαν παράμετρο την value η οποία θα αντιπροσωπεύει το scroll που γίνεται, δηλαδή άμα το value είναι -3 ή -100 θα θεωρείται σαν zoom in και αν θα είναι 3 ή 100 θα θεωρείται σαν zoom out. Δημιουργούμε τις μεταβλητές που θα χρειαστούμε. Η originX και originY αντιπροσωπεύουν τις origin συντεταγμένες του canva οι οποίες είναι οι συντεταγμένες που ορίζουν την αρχή των συντεταγμένων της πίστας μας, η originWidth και

originHeight αντιπροσωπεύουν το μέγεθος της πίστας μας, η scaleFactor αντιπροσωπεύει την κλίμακα του zoom που υφίσταται στην πίστα και η translatePos παίρνεις τις συντεταγμένες του ποντικιού που έγινε το zoom πάνω στην πίστα.

```
let originX = 0;
let originY = 0;
let originWidth = 0;
let originHeight = 0;
let scaleFactor = 1;
let translatePos = {x: 0,y: 0};
```

Έχουμε ορίσει ένα όριο για το zoom που γίνεται στην πίστα, δεν μπορεί να υπερβεί το 2.5 δηλαδή δεν μπορεί να γίνει υπερβολικό zoom in και δεν μπορεί να κατέβει το 1 δηλαδή δεν μπορεί να ξεπεράσει το αρχικό zoom out της πίστας. Στο τέλος κάθε περίπτωσης εκτελούμε την συνάρτηση trackTransforms η οποία κρατάει το ιστορικό των origin συντεταγμένων του canva. Αυτή η συνάρτηση μας βοηθάει να ζωγραφίσουμε στον canva βασιζόμενοι στο zoom που έχει γίνει. Προσθέτουμε την συνάρτηση handleScroll και trackTransform στο αρχείο **zoom_drag.js**.

```
function trackTransforms(x, y, w, h) {
    originX = x;
    originY = y;
    originWidth = w;
    originHeight = h;
};
```

```
function handleScroll(value) {
  if(value == -3 || value == -100) { //zoom in
    if(scaleFactor < 2.5) {
      scaleFactor += 0.1;
            translatePos.x = (mouseX-originX)/scaleFactor;
      translatePos.y = (mouseY-originY)/scaleFactor;
            zoom(scaleFactor, translatePos);
            translatePos.x = -((scaleFactor*translatePos.x)-translatePos.x);
            translatePos.y = -((scaleFactor*translatePos.y)-translatePos.y);
```

```
            originX = translatePos.x;
            originY = translatePos.y;
            trackTransforms(translatePos.x, translatePos.y,

                    translatePos.x+(scaleFactor*canvas.width),

                     translatePos.y+(scaleFactor*canvas.height));
    }
  }else if(value == 3 || value == 100) { //zoom out
    if(scaleFactor > 1) {
      scaleFactor -= 0.1;
            translatePos.x = (mouseX-originX)/scaleFactor;
            translatePos.y = (mouseY-originY)/scaleFactor;
            zoom(scaleFactor, translatePos);
            translatePos.x = -((scaleFactor*translatePos.x)-translatePos.x);
            translatePos.y = -((scaleFactor*translatePos.y)-translatePos.y);
            originX = translatePos.x;
            originY = translatePos.y;
            trackTransforms(translatePos.x, translatePos.y,

                    translatePos.x+(scaleFactor*canvas.width),

                    translatePos.y+(scaleFactor*canvas.height));
    }
  }
};
```

Καλούμε την συνάρτηση trackTransforms για να ορίσουμε τις μεταβλητές.

```
trackTransforms(0,0,canvas.width, canvas.height);
```

Προσθέτουμε την συνάρτηση **zoom** στο αρχείο **zoom_drag.js** η οποία εφαρμόζει το zoom που γίνεται στην πίστα. Χρησιμοποιούμε δύο μεθόδους από το context του canvas που μας επιτρέπουν να κάνουμε το zoom, είναι η μέθοδος translate και scale. Η translate μετατοπίζει τις αρχικές συντεταγμένες του canva και η scale μετατοπίζει το μέγεθος της κλίμακας του canva δηλαδή άμα το scale είναι 2 τότε τα pixel θα έχουν διπλάσιο μέγεθος. Εφαρμόζουμε το zoom και ξανά ζωγραφίζουμε την πίστα.

```
function zoom(scaleFactor, translatePos) {
    clearCanvas();
    ctx.save();
    ctx.translate(translatePos.x, translatePos.y);
    ctx.scale(scaleFactor,scaleFactor);
    ctx.translate(-translatePos.x, -translatePos.y);
    nonogram.drawGrid();
    nonogram.drawRowNumbers();
    nonogram.drawColumnNumbers();
    for(let i=0; i<nonogram.emptyGrid.length; i++) {
            if(nonogram.emptyGrid[i].value === 1){
              nonogram.drawBlackCell(nonogram.emptyGrid[i]);
              nonogram.drawPreview(nonogram.emptyGrid[i]);
            }else if(nonogram.emptyGrid[i].value === 2) {
              nonogram.drawXCell(nonogram.emptyGrid[i]);
              nonogram.drawPreview(nonogram.emptyGrid[i]);
            }
    }
    ctx.beginPath();
    ctx.strokeStyle = "red";
    ctx.lineWidth = 3;
    for(let i=0; i<nonogram.rowNumbersGrid.length; i++) {
            if(nonogram.rowNumbersGrid[i].value === 1) {
              ctx.moveTo(nonogram.rowNumbersGrid[i].x+3,

                 (nonogram.rowNumbersGrid[i].y + nonogram.blockSize)-3);
              ctx.lineTo((nonogram.rowNumbersGrid[i].x + nonogram.blockSize)-3,

                 nonogram.rowNumbersGrid[i].y+3);
            }
    }
    for(let i=0; i<nonogram.columnNumbersGrid.length; i++) {
            if(nonogram.columnNumbersGrid[i].value === 1) {
              ctx.moveTo(nonogram.columnNumbersGrid[i].x+3,

                 (nonogram.columnNumbersGrid[i].y + nonogram.blockSize)-3);
```

```
            ctx.lineTo((nonogram.columnNumbersGrid[i].x + nonogram.blockSize)-3,

                nonogram.columnNumbersGrid[i].y+3);
            }
    }
    ctx.closePath();
    ctx.stroke();
    ctx.restore();
};
```

Προσθέτουμε στο αρχείο **zoom_drag.js** των κώδικα που χρειαζόμαστε για την αλληλεπίδραση του χρήστη με το scroll του ποντικιού.

```
$(canvas).bind('mousewheel', function(event) {
    if(state === "level" || state === "multiplayer") {
            handleScroll(event.originalEvent.deltaY);
    }
});

$(canvas).bind('DOMMouseScroll', function(event) {
    if(state === "level" || state === "multiplayer") {
            handleScroll(event.detail);
    }
});
```

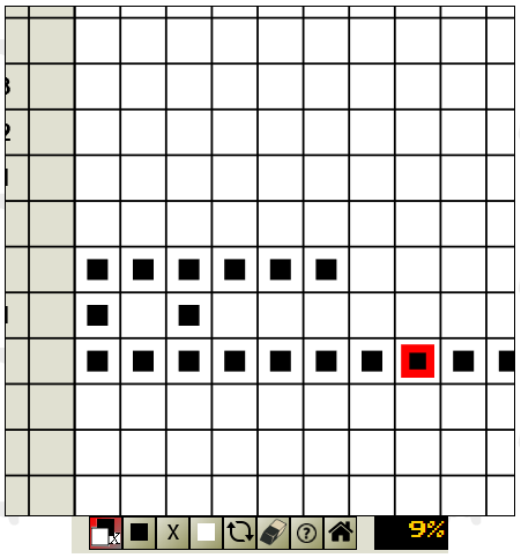
Όταν ο χρήστης έχει κάνει zoom και θέλει να επιλέξει κελιά για να μπορέσει να ζωγραφίσει τα κελιά με βάση το zoom που έχει γίνει πρέπει να τροποποιήσουμε των κώδικα που ζωγραφίζει τα κελιά.Στο event **mousedown** του canva προσθέτουμε των κώδικα,

```
...

isDown = true;
ctx.save();
ctx.translate(originX,originY);
ctx.scale(scaleFactor,scaleFactor);
nonogram.fillCels((startPointMouseX-originX)/scaleFactor,
```

```
    (startPointMouseY-originY)/scaleFactor);
ctx.restore();
nonogram.findUserChoices();

...
```

στο event **mousemove** προσθέτουμε των κώδικα

```
...
ctx.save();
ctx.translate(originX,originY);
ctx.scale(scaleFactor,scaleFactor);
nonogram.fillMultiCells((mouseX-originX)/scaleFactor,
    (mouseY-originY)/scaleFactor, (startPointMouseX-originX)/scaleFactor,
    (startPointMouseY-originY)/scaleFactor);
ctx.restore();
$("#info-current-progress").text("");
...
```



Αυτή είναι υλοποίηση του zoom.

10.2 Υλοποίηση λειτουργίας drag

Η υλοποίηση του drag θέλουμε να κάνει το εξής, όταν ο χρήστης κάνει zoom θα του εμφανίζει τέσσερα κουμπιά πάνω, κάτω, δεξιά και αριστερά με τα οποία θα μπορεί να κουνάει την πίστα και να την μετακινεί στα σημεία που θέλει για να μπορεί να επιλέξει τα κελιά που θέλει χωρίς να τον ενοχλεί το zoom.

Προσθέτουμε στο αρχείο **zoom_drag.js** τις μεταβλητές που θα χρειαστούμε για το drag,

```
let myLimit = 300;
let limitTop = myLimit;
let limitLeft = myLimit;
let limitBottom;
let limitRight;
let dragged = 0;
let dragStart = {x:0,y:0};
let activeDragControl;
let topControl = document.getElementById('top');
let leftControl = document.getElementById('left');
let rightControl = document.getElementById('right');
let bottomControl = document.getElementById('bottom');
```

Οριοθετούμε τα όρια limitBottom και limitRight όταν δημιουργείται η πίστα μας. Στο αρχείο **stages** μέσα στην συνάρτηση createLevel τοποθετούμε των κώδικα μας στο τέλος της συνάρτησης.

```
limitBottom = nonogram.height-myLimit;
limitRight = nonogram.width-myLimit;
```

Στην συνέχεια φτιάχνουμε στο αρχείο **zoom_drag.js** την συνάρτηση **drag** και **dragControl.**

Η drag δουλεύει ως εξής, δέχεται σαν παράμετρο την translatePos, καθαρίζει τον canva, επαναπροσδιορίζει τις origin συντεταγμένες του canva, εφαρμόζει το scale που έχει γίνει και στην συνέχεια ξανά ζωγραφίζει την πίστα χρησιμοποιώντας τις συναρτήσεις retrieveProgress και redrawProgress.

```
function drag(translatePos) {
    clearCanvas();
    ctx.save();
    ctx.translate(translatePos.x,translatePos.y);
```

```
    ctx.scale(scaleFactor,scaleFactor);
    nonogram.drawGrid();
    nonogram.drawRowNumbers();
    nonogram.drawColumnNumbers();
    nonogram.retrieveProgress(retrieve(currentStage),
        retrieve('rowNumbersGrid-'+currentStage),
        retrieve('columnNumbersGrid-'+currentStage));
    nonogram.redrawProgress();
    ctx.restore();
};
```

Η dragControl δέχεται σαν παράμετρο τις x και y μεταβλητές οι οποίες είναι οι συντεταγμένες του ποντικιού αφαιρώντας τις συντεταγμένες από τις οποίες ξεκίνησε το drag και στην συνέχεια αυτές οι συντεταγμένες δηλώνονται στην translatePos οι οποία ορίζεται σαν παράμετρο στην συνάρτηση drag. Σε κάθε περίπτωση της if ελέγχουμε να βρούμε σε ποια όρια βρισκόμαστε και πράττουμε αναλόγως.

```
function dragControl(x,y) {
    translatePos.x = x;
    translatePos.y = y;
    if((limitTop>translatePos.y) && (limitLeft>translatePos.x) &&
      (limitRight<(translatePos.x+(scaleFactor*canvas.width))) &&
      (limitBottom<(translatePos.y+(scaleFactor*canvas.height)))) {
            drag(translatePos);
            trackTransforms(translatePos.x, translatePos.y, translatePos.x+(scaleFactor*canvas.width),
              translatePos.y+(scaleFactor*canvas.height));
    }else if(limitTop<=translatePos.y && limitLeft<=translatePos.x) {
            translatePos.x = originX;
            translatePos.y = originY;
            drag(translatePos);
            trackTransforms(translatePos.x, translatePos.y, translatePos.x+(scaleFactor*canvas.width),
              translatePos.y+(scaleFactor*canvas.height));
    }else if(limitTop<=translatePos.y && limitRight>=(translatePos.x+(scaleFactor*limitRight))) {
```

```
        translatePos.x = originX;
        translatePos.y = originY;
        drag(translatePos);
        trackTransforms(translatePos.x, translatePos.y, translatePos.x+(scaleFactor*canvas.width),
          translatePos.y+(scaleFactor*canvas.height));
}else if(limitRight>=(translatePos.x+(scaleFactor*limitRight)) &&
  limitBottom>=(translatePos.y+(scaleFactor*limitBottom))) {
        translatePos.x = originX;
        translatePos.y = originY;
        drag(translatePos);
        trackTransforms(translatePos.x, translatePos.y, translatePos.x+(scaleFactor*canvas.width),
          translatePos.y+(scaleFactor*canvas.height));
}else if(limitBottom>=(translatePos.y+(scaleFactor*limitBottom)) && limitLeft<=translatePos.x) {
        translatePos.x = originX;
        translatePos.y = originY;
        drag(translatePos);
        trackTransforms(translatePos.x, translatePos.y, translatePos.x+(scaleFactor*canvas.width),
          translatePos.y+(scaleFactor*canvas.height));
}else if(limitTop<=translatePos.y) {
        translatePos.y = originY;
        drag(translatePos);
        trackTransforms(translatePos.x, translatePos.y, translatePos.x+(scaleFactor*canvas.width),
          translatePos.y+(scaleFactor*canvas.height));
}else if(limitLeft<=translatePos.x) {
        translatePos.x = originX;
        drag(translatePos);
        trackTransforms(translatePos.x, translatePos.y, translatePos.x+(scaleFactor*canvas.width),
          translatePos.y+(scaleFactor*canvas.height));
}else if(limitRight>=(translatePos.x+(scaleFactor*canvas.width))) {
        translatePos.x = originX;
        drag(translatePos);
```

```
            trackTransforms(translatePos.x, translatePos.y, translatePos.x+(scaleFactor*canvas.width),
              translatePos.y+(scaleFactor*canvas.height));
    }else if(limitBottom>=(translatePos.y+(scaleFactor*limitBottom))) {
            translatePos.y = originY;
            drag(translatePos);
            trackTransforms(translatePos.x, translatePos.y, translatePos.x+(scaleFactor*canvas.width),
              translatePos.y+(scaleFactor*canvas.height));
    }
    else{
            translatePos.x = originX;
            translatePos.y = originY;
    }
};
```

Προσθέτουμε στο τέλος της συνάρτησης **zoom** των κώδικα που χρειαζόμαστε για να εμφανίζονται τα κουμπιά του drag όταν έχει γίνει zoom.

```
if(scaleFactor !== 1) {
            $(topControl).show();
            $(leftControl).show();
            $(rightControl).show();
            $(bottomControl).show();
}else{
            $(topControl).hide();
            $(leftControl).hide();
            $(rightControl).hide();
            $(bottomControl).hide();
}
```

Στην συνέχεια προσθέτουμε στο αρχείο **index.html** τα κουμπιά με τα οποία θα μπορούμε να εφαρμόσουμε το drag. Στο div tag με id game μετά το tag του canva προσθέτουμε των κώδικα.

```
<div id="top"><img src="img/left_right.png"></div>
<div id="left"><img src="img/top_down.png"></div>
```

```
<div id="right"><img src="img/top_down.png"></div>
<div id="bottom"><img src="img/left_right.png"></div>
```

Στο αρχείο **tools.css** προσθέτουμε την css για την μορφοποίηση των κουμπιών.

```
#top, #left, #right, #bottom {
  display: none;
  position: absolute;
  border: 1px solid black;
  background: rgba(0,250,250,1);
  opacity: 0.5;
}

#top {
  top: 10px;
  right: 45%;
  width: 26px;
  height: 25px;
}

#left {
  left: 10px;
  top: 45%;
  width: 25px;
  height: 26px;
}

#right {
  right: 10px;
  top: 45%;
  width: 25px;
  height: 26px;
}

#bottom {
```

```
  bottom: 40px;
  right: 45%;
  width: 26px;
  height: 25px;
}
```

Για να πετύχουμε την λειτουργικότητα των κουμπιών του drag προσθέτουμε στο αρχείο **zoom_drag.js** των κώδικα

```
// --- Drag Controls for mouse
topControl.addEventListener('mousemove', function(event) {
    mouseX = event.offsetX || (event.pageX - topControl.offsetLeft);
    mouseY = event.offsetY || (event.pageY - topControl.offsetTop);
});

topControl.addEventListener('mousedown', function(event) {
    event.preventDefault();
    mouseX = event.offsetX || (event.pageX - topControl.offsetLeft);
    mouseY = event.offsetY || (event.pageY - topControl.offsetTop);
    $(this).hide();
    activeDragControl = "top";
});

leftControl.addEventListener('mousemove', function(event) {
    mouseX = event.offsetX || (event.pageX - leftControl.offsetLeft);
    mouseY = event.offsetY || (event.pageY - leftControl.offsetTop);
});

leftControl.addEventListener('mousedown', function(event) {
    event.preventDefault();
    mouseX = event.offsetX || (event.pageX - leftControl.offsetLeft);
    mouseY = event.offsetY || (event.pageY - leftControl.offsetTop);
    $(this).hide();
    activeDragControl = " left";
});
```

```
rightControl.addEventListener('mousemove', function(event) {
    mouseX = event.offsetX || (event.pageX - rightControl.offsetLeft);
    mouseY = event.offsetY || (event.pageY - rightControl.offsetTop);
});

rightControl.addEventListener('mousedown', function(event) {
    event.preventDefault();
    mouseX = event.offsetX || (event.pageX - rightControl.offsetLeft);
    mouseY = event.offsetY || (event.pageY - rightControl.offsetTop);
    $(this).hide();
    activeDragControl = "right";
});

bottomControl.addEventListener('mousemove', function(event) {
    mouseX = event.offsetX || (event.pageX - bottomControl.offsetLeft);
    mouseY = event.offsetY || (event.pageY - bottomControl.offsetTop);
});

bottomControl.addEventListener('mousedown', function(event) {
    event.preventDefault();
    mouseX = event.offsetX || (event.pageX - bottomControl.offsetLeft);
    mouseY = event.offsetY || (event.pageY - bottomControl.offsetTop);
    $(this).hide();
    activeDragControl = "bottom";
});
```

Τροποποιούμαι όλο των κώδικα στο event **mousedown** του canva.

```
startPointMouseX = event.offsetX || (event.pageX - canvas.offsetLeft);

startPointMouseY = event.offsetY || (event.pageY - canvas.offsetTop);
if(state === "level") {
    if(startPointMouseX<originX) {
        dragStart.x = startPointMouseX - translatePos.x;
                dragStart.y = startPointMouseY - translatePos.y;
```

```
            dragged = true;

    }else if(startPointMouseY<originY) {
        dragStart.x = startPointMouseX - translatePos.x;
            dragStart.y = startPointMouseY - translatePos.y;
            dragged = true;
    }else if(startPointMouseX>originWidth) {
        dragStart.x = startPointMouseX - translatePos.x;
            dragStart.y = startPointMouseY - translatePos.y;
        dragged = true;
    }else if(startPointMouseY>originHeight) {
        dragStart.x = startPointMouseX - translatePos.x;
            dragStart.y = startPointMouseY - translatePos.y;
            dragged = true;
    }else{
        isDown = true;
            ctx.save();
            ctx.translate(originX,originY);
            ctx.scale(scaleFactor,scaleFactor);
            nonogram.fillCels((startPointMouseX-originX)/scaleFactor,

                    (startPointMouseY-originY)/scaleFactor);
            ctx.restore();
            nonogram.findUserChoices();
            store(currentStage, nonogram.userChoices.levelGrid);
            store('rowNumbersGrid-'+currentStage,

            nonogram.userChoices.rowNumbersGrid);
            store('columnNumbersGrid-'+currentStage,

            nonogram.userChoices.columnNumbersGrid);
            nonogram.findProgress();
    }
}else if(state === "multiplayer") {
    //coming soon

}
```

Προσθέτουμε στο event **mousemove** του canva δύο if με τις οποίες ελέγχουμε αν γίνεται drag μέσο των tools ή από το κενό που δημιουργείται στον canva

```
if(dragged){
    dragControl(mouseX-dragStart.x, mouseY-dragStart.y);
    $(topControl).hide();
    $(leftControl).hide();
    $(rightControl).hide();
    $(bottomControl).hide();
}
if(activeDragControl) {
    $(topControl).hide();
    $(leftControl).hide();
    $(rightControl).hide();
    $(bottomControl).hide();
    dragControl(mouseX-dragStart.x, mouseY-dragStart.y);
}
```

Τροποποιούμαι των κώδικα του event **mouseup** του canva

```
$(canvas).mouseup(function(){
    if(state === "level") {
            isDown = false;
            if(dragged){
                    $(topControl).show();
                    $(leftControl).show();
                    $(rightControl).show();
                    $(bottomControl).show();
                    dragged = false;
            }
            if(activeDragControl) {
                    activeDragControl = null;
            }
            if(nonogram.checkProgress()) {
                    $("#correct").show();
```

```
                store("correct-" + currentStage, 1);
                $(".correct-" + currentStage).show();
            }else{
                $("#correct").hide();
                store("correct-" + currentStage, 0);
                $(".correct-" + currentStage).hide();
            }
            nonogram.findUserChoices();
            store(currentStage, nonogram.userChoices.levelGrid);
            store('rowNumbersGrid-'+currentStage, nonogram.userChoices.rowNumbersGrid);
            store('columnNumbersGrid-'+currentStage, nonogram.userChoices.columnNumbersGrid);
            $("#info-current-progress").text("");
            $("#info-current-progress").text(nonogram.findProgress() + "%");
    }
});
```

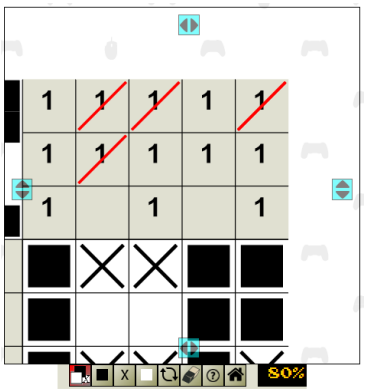
και το event **mouseout** του canva.

```
$(canvas).mouseout(function(){
            isDown = false;
            if(dragged) {
                activeDragControl = false;
                dragged = false;
                $(topControl).show();
                $(leftControl).show();
                $(rightControl).show();
                $(bottomControl).show();
            }
            if(isDown){
                nonogram.findUserChoices();
                store(currentStage, nonogram.userChoices.levelGrid);
                store('rowNumbersGrid-'+currentStage, nonogram.userChoices.rowNumbersGrid);
                store('columnNumbersGrid-'+currentStage, nonogram.userChoices.columnNumbersGrid);
            }
});
```

Προσθέτουμε στον canva το event **mouseover** με το οποίο μπορούμε να πάρουμε τις συντεταγμένες του ποντικιού όταν ο χρήστης πατήσει ένα από τα drag control.

```
canvas.addEventListener("mouseover", function(evt) {
    if(activeDragControl) {
            lastX = evt.offsetX || (evt.pageX - canvas.offsetLeft);
            lastY = evt.offsetY || (evt.pageY - canvas.offsetTop);
            if(isNaN(translatePos.x)) {
                    translatePos.x = 0;
                    translatePos.y = 0;
            }
            dragStart.x = lastX - translatePos.x;
            dragStart.y = lastY - translatePos.y;
            dragged = true;
    }

},false);
```

Έχουμε φτιάξει το εξής αποτέλεσμα.



Στο επόμενο κεφάλαιο θα αναλύσουμε το multiplayer κομμάτι του παιχνιδιού.

11. Multiplayer

Κάθε φορά που ένας χρήστης συνδέεται στην σελίδα μας, χρησιμοποιώντας το socket.io, αντιστοιχείται σε ένα socket με το οποίο μπορεί να επικοινωνεί ασύγχρονα με τον server και έχει την δυνατότητα να στέλνει και να δέχεται events και δεδομένα.

Για να μπορέσουμε να χρησιμοποιήσουμε το socket από πλευράς client πρέπει να προσθέσουμε στο αρχείο **index.html** το script που θα μας επιτρέψει να χρησιμοποιούμε το socket. Προσθέτουμε το script στην αρχή των scripts.

```
<script src="/socket.io/socket.io.js"></script>
```

Θα προσθέσουμε έναν counter στην σελίδα μας, ο οποίος θα μας δείχνει το σύνολο των χρηστών που βρίσκονται στην σελίδα. Για να το κάνουμε αυτό προσθέτουμε στο αρχείο **index.html** μέσα στο div tag container στο τέλος, το div tag με id clients-count

```
<div id="clients-count"></div>
```

και στο αρχείο **home.css** προσθέτουμε την css για την μορφοποίηση.

```
#clients-count {
  display: block;
  position: absolute;
  bottom: 15px;
  left: 15px;
  font-weight: bold;
  font-size: 30px;
  font-family: Arial;
  background: #826e6e;
  color: white;
  border: 3px solid black;
  padding: 5px;
}
```

Για να μπορέσουμε να ανανεώνουμε τον counter στο client του κάθε χρήστη, δημιουργούμε στο αρχείο **server.js** την global μεταβλητή **clients**

```
let clients = 0;
```

η οποία θα αυξάνεται κάθε φορά όταν ένας χρήστης συνδέεται στην σελίδα μας. Προσθέτουμε στο event connection του server μας την αύξηση της μεταβλητής και στο event disconnect του sock κάθε χρήστη μειώνουμε την μεταβλητή και κάνουμε την ανανέωση του counter στους χρήστες με το event refresh counter του server.

```
io.on('connection', (sock) => {
    //when a user connects to the server
    clients ++;
    ...

    sock.on('disconnect', () => {
            console.log('user disconnect');
            clients --;
            io.sockets.emit('refresh counter', { description: 'Players online: ' + clients });
    });

    //Message to all connected clients
    io.sockets.emit('refresh counter', { description: 'Players online: ' + clients});
})
```

Δημιουργούμε στον φάκελο js το αρχείο **nonogram.multiplayer.js** στο οποίο θα έχουμε των κώδικα που θα χρειαστούμε για το multiplayer από πλευράς client. Προσθέτουμε το script στο αρχείο **index.html.**

```
<script src="js/nonogram.multiplayer.js"></script>
```

Δημιουργούμε στο αρχείο **nonogram.multiplayer.js** το object sock το οποίο μας επιτρέπει να επικοινωνούμε με τον server

```
const sock = io();
```

και προσθέτουμε των κώδικα ο οποίος όταν λάβει το event refresh counter θα αλλάζει τον αριθμό των χρηστών.

```
//Client Counter
```

```
sock.on('refresh counter', (data) => {
    $('#clients-count').text(data.description);
});
```

Η υλοποίηση που θα κάνουμε για το multiplayer έχει ως εξής: όταν ο χρήστης θα πατήσει το κουμπί του multiplayer από το κεντρικό μενού τότε θα στέλνει το event multiplayer στον server ότι ψάχνει ο χρήστης να βρει συμπαίκτη, στην συνέχεια θα του εμφανίζεται ένα μήνυμα ότι ψάχνετε συμπαίκτης και όταν βρεθεί τότε θα δημιουργείται το multiplayer game. Θα ξεκινήσουμε φτιάχνοντας την λειτουργικότητα της επιλογής multiplayer. Στο αρχείο **main-menu.js** προσθέτουμε των κώδικα της λειτουργικότητας της επιλογής multiplayer. Για το event multiplayer θα μιλήσουμε στο επόμενο κεφάλαιο που αφορά τα events.

```
$('#multiplayer').click(function() {
    $('#menu').hide();
    $('#game-lobbie').show();
    $('#exit-multiplayer-waiting-lobby').show();
    sock.emit('multiplayer', "User looking for a team mate");
});
```

Στην συνέχεια του εμφανίζεται το μήνυμα της αναμονής. Προσθέτουμε των κώδικα στο αρχείο **index.html** μέσα στο div tag container.

```
<!-- Game lobbie for multiplayer -->
<div id="game-lobbie">
        <h3>Multiplayer</h3>
        <div id="msg">Searching for player...</div>
        <div class="lds-ellipsis">
        <div></div><div></div><div></div><div></div>
        </div>
        <div id="exit-multiplayer-waiting-lobby">Exit</div>
</div>
```

και δημιουργούμε ένα καινούργιο αρχείο css το **multiplayer.css** στην οποία θα έχουμε την css που χρειαζόμαστε για το multiplayer. Προσθέτουμε το link του css αρχείου στο index.html,

```
<link rel="stylesheet" type="text/css" href="css/multiplayer.css">
```

των κώδικα που χρειαζόμαστε στο css αρχείο

```
#game-lobbie {
    display: none;
    border: 3px solid black;
    border-radius: 25px;
    position: absolute;
    text-align: center;
    margin: 0 auto;
    top: 40%;
    transform: translate(-50%, -40%);
    left: 50%;
    background: #826e6e;
    font-weight: bold;
    font-size: 25px ;
    font-family: Arial;
    font-size: 25px;
    box-shadow: 3px 3px 7px grey;
    width: 300px;
    color: white;
}

#game-lobbie h3 {
    padding: 10px;
    border-bottom: 3px solid black;
}

#msg {
    padding-top: 10px;
}

#exit-multiplayer-waiting-lobby {
    cursor: pointer;
    margin-top: 10px;
    border-top: 3px solid black;
```

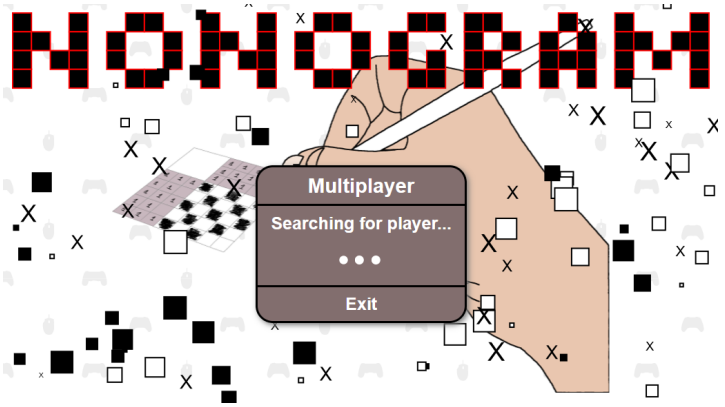
```
    padding: 10px;
}

/*Loading dots*/
.lds-ellipsis {
    display: inline-block;
    position: relative;
    width: 80px;
    height: 60px;
}
.lds-ellipsis div {
    position: absolute;
    top: 33px;
    width: 13px;
    height: 13px;
    border-radius: 50%;
    background: #fff;
    animation-timing-function: cubic-bezier(0, 1, 1, 0);
}
.lds-ellipsis div:nth-child(1) {
    left: 8px;
    animation: lds-ellipsis1 0.6s infinite;
}
.lds-ellipsis div:nth-child(2) {
    left: 8px;
    animation: lds-ellipsis2 0.6s infinite;
}
.lds-ellipsis div:nth-child(3) {
    left: 32px;
    animation: lds-ellipsis2 0.6s infinite;
}
.lds-ellipsis div:nth-child(4) {
    left: 56px;
    animation: lds-ellipsis3 0.6s infinite;
}
```

```
@keyframes lds-ellipsis1 {
  0% {
    transform: scale(0);
  }
  100% {
    transform: scale(1);
  }
}
@keyframes lds-ellipsis3 {
  0% {
    transform: scale(1);
  }
  100% {
    transform: scale(0);
  }
}
@keyframes lds-ellipsis2 {
  0% {
    transform: translate(0, 0);
  }
  100% {
    transform: translate(24px, 0);
  }
}
```

και προσθέτουμε των κώδικα της αλληλεπίδρασης του χρήστη με την επιλογή exit του div tag με id exit multiplayer waiting lobby στο αρχείο **main-menu.js**. Για το event exit multiplayer waiting lobby θα μιλήσουμε στο επόμενο κεφάλαιο.

```
$('#exit-multiplayer-waiting-lobby').click(function() {
  $('#menu').show();
  $('#game-lobbie').hide();
  sock.emit('exit multiplayer waiting lobby', 'Player left the lobby');
});
```

Η κατάσταση στην οποία βρίσκεται ο χρήστης είναι η εξής.



Επίσης κρύβουμε των clients counter όταν δημιουργείται μία πίστα στο singleplayer. Προσθέτουμε στο αρχείο stages.js, μέσα στην συνάρτηση createLevel στο τέλος, των κώδικα που χρειαζόμαστε

```
$("#clients-count").hide();
```

Για το multiplayer θα φτιάξουμε ένα module στον server με όνομα NonogramMultiplayerGame το οποίο θα περιέχει μια class που θα αντιπροσωπεύει το παιχνίδι μας και θα περιέχει ιδιότητες και μεθοδους που θα χρειαστούμε για το multiplayer παιχνίδι μεταξύ των δύο παικτών.Στον φάκελο server δημιουργούμε το αρχείο **nonogramMultiplayerGame.js** στο οποίο θα φτιάξουμε την class NonogramMultiplayerGame,

```
class NonogramMultiplayerGame {...};
```

η οποία στο constructor θα περιέχει τις μεταβλητές, **players** η οποία θα είναι πίνακας με τα socket id των παικτών, την **turn** η οποία μας δείχνει των παίκτη που είναι η σειρά του για να παίξει, την **nonogram** η οποία θα περιέχει τα δεδομένα της πίστας μας, την **choice** η οποία θα περιέχει την επιλογή που έχει κάνει ο εκάστοτε παίκτης και την **roomId** η οποία θα περιέχει το id του room που βρίσκονται οι παίκτες και σε κάθε παίκτη δημιουργούμε ορισμένα events στα οποία όταν θα δέχεται από τον server τα συγκεκριμένα events τότε θα γίνεται το εξής:

Με το event **turn** θα ορίζουμε ποιός παίκτης θα παίξει πρώτος όταν ξεκινήσει το παιχνίδι εκτελώντας την μέθοδο _playerTurn.

Με το event **empty grid** ορίζουμε και ανανεώνουμε τα δεδομένα της πίστας μας.

Με το event **nonogram** ορίζουμε για πρώτη φορά την πίστα μας.

Με το event **correct** ελέγχουμε την πρόοδο που έχουν κάνει οι χρήστες για να βρούμε άμα έχουν βρει την λύση της πίστας και στο τέλος του constructor καλούμε την μέθοδο **_playerTurn** για να ορίσουμε ποιος παίκτης θα παίξει πρώτος.

```
constructor(p1, p2,room) {
        this._players = [p1, p2];
        this._turn = null;
        this._nonogram = null;
        this._choice = null;
        this._roomId = room;

        this._players.forEach( (player, idx) => {
                player.on('turn', () => {
              this._playerTurn();
                });

                player.on('empty grid', (value) => {
          this._updateNonogram(value);
                });

                player.on('nonogram', (nonogram) => {
                   this._nonogram = nonogram;
                });

                player.on('correct', ()=> {
                     this._checkProgress();
                });
        });

        this._playerTurn();
    }
```

Οι μέθοδοι που θα περιέχει η class είναι οι εξής:

```
    _sendToPlayers(msg) {
        this._players.forEach( (player) => {
                player.emit('multiplayer', msg);
        });
    }
```

```
_playerTurn() {
        if(this._turn === this._players[0]) {
                //player 2 turn
                this._turn.emit('wait');
                this._turn = this._players[1];
                this._turn.emit('turn', 'Turn of player 2');
                this._turn.emit('your turn to play');
                console.log("Turn of player 2 player 1 must wait");
        }else{
                //player 1 turn
                this._players[1].emit('wait');
                this._turn = this._players[0];
                this._turn.emit('turn', 'Turn of player 1');
                this._turn.emit('your turn to play');
                console.log("Turn of player 1 player 2 must wait");
        }
}

_sendNonogramToPlayer() {
        if(this._turn === this._players[0]) {
                this._players[1].emit('nonogram', this._nonogram);
        }else{
                this._players[0].emit('nonogram', this._nonogram);
        }
}

_updateNonogram(data) {
        if(this._turn === this._players[0]) {
                this._players[1].emit('update', data);
        }else{
                this._players[0].emit('update', data);
        }
}
```

```
_checkProgress() {
        this._players.forEach( (player) => {
                player.emit('correct');
        });
}
```

Στο τέλος του αρχείου κάνουμε export το module μας.

```
module.exports = NonogramMultiplayerGame;
```

Τοποθετούμε το module στο αρχείο **server.js** μαζί με τα άλλα module.

```
const NonogramMultiplayerGame = require('./nonogramMultiplayerGame');
```

Στο αρχείο **server.js** τοποθετούμε τις εξής μεταβλητές οι οποίες είναι global: την **waitingPlayer** η οποία θα αντιπροσωπεύει των εκάστοτε παίκτη που έχει μπει στο multiplayer και περιμένει να παίξει, και την **roomno** η οποία αντιπροσωπεύει τον αριθμό των δωματίων που έχουν δημιουργηθεί.

```
let waitingPlayer = null;
let roomno = 1;
```

Θα ξεκινήσουμε με την δομή του server για το multiplayer και μετά θα αναλύσουμε των κώδικα που χρειαζόμαστε στον client. Όταν ο χρήστης συνδέεται στην σελίδα μας τότε στην σύνδεση που γίνεται στον server

```
io.on('connection', (sock) => {...}
```

δημιουργούνται οι μεταβλητές: **inRoom** η οποία αντιπροσωπεύει τον αριθμό του multiplayer lobby που βρίσκεται ο χρήστης, την **multiplayerGame** η οποία είναι object και θα περιέχει τις πληροφορίες των παικτών, το room που παίζουν και το multiplayer nonogram που θα παίζουν και την **playersInformation** η οποία θα είναι και αυτοί object η οποία θα αποθηκεύσει τις πληροφορίες των παικτών και το room που στην συνέχεια θα δημιουργήσουν το multiplayerGame.

```
let inRoom = 0;
let multiplayerGame;
let playersInformation;
```

Καθώς κάνει την σύνδεση στην σελίδα τότε ο χρήστης τοποθετείται στο namespace all στο οποίο βρίσκονται όλοι οι χρήστες που βρίσκονται στην σελίδα εκτός από αυτούς που παίζουν ήδη στο multiplayer.

```
sock.join('all');
```

Αυτός είναι ο κώδικας που χρειαζόμαστε στον server κατά την σύνδεση του χρήστη στο site για το multiplayer. Στην συνέχεια θα αναλύσουμε τα events που χρειαζόμαστε στον server για την επικοινωνία του server με τον χρήστη και για τα events που χρειαζόμαστε στον client για την επικοινωνία του χρήστη με τον server.

11.1 Τα Multiplayer events του server

Τα events που θέλουμε στον server για την επικοινωνία του χρήστη με τον server τοποθετούνται στο io.on('connection') και είναι τα εξής:

Το βασικό event είναι το **multiplayer:**

```
sock.on('multiplayer', (data) => { ... });
```

Αυτό το event ενεργοποιείται όταν ο χρήστης πατήσει το κουμπί multiplayer από το μενού. Κάνει το εξής: ελέγχει αν το πιο πρόσφατο room που υπάρχει έχει πάνω από 2 χρήστες, αν έχει τότε αυξάνουμε τον αριθμό της μεταβλητής room για να έχουμε ένα καινούργιο room που να μπορεί να χρησιμοποιηθεί.

```
if(io.nsps['/'].adapter.rooms["room-"+roomno] && io.nsps['/'].adapter.rooms["room-"+roomno].length > 1)
{
   roomno++;
}
```

Στην συνέχεια ελέγχουμε αν υπάρχει ήδη χρήστης που περιμένει να παίξει multiplayer, αν υπάρχει τότε αποθηκεύουμε το room στο οποίο θα παίξει ο χρήστης στην μεταβλητή inRoom, αφαιρούμε τον χρήστη από το namespace all και τον προσθέτουμε στο namespace room που θα παίξει multiplayer. Έπειτα δημιουργούμε το object playersInformation στο οποίο αποθηκεύουμε από τους δύο παίκτες το socket id, ο παίκτης που περίμενε ήδη θα είναι πάντα ο πρώτος παίκτης και ο άλλος ο δεύτερος και αποθηκεύουμε το room στο οποίο θα παίξουν. Στέλνουμε στον κάθε χρήστη ποιός παίκτης είναι, τους στέλνουμε τα δεδομένα του playersInformation, στέλνουμε στον δεύτερο παίκτη το event wait που σημαίνει ότι δεν είναι ο γύρος του και ότι πρέπει να περιμένει, στέλνουμε στον πρώτο παίκτη το event your turn to play που σημαίνει ότι είναι ο

γύρος του και μπορεί να παίξει και στο τέλος κάνουμε την μεταβλητή waitingPlayer null γιατί δεν έχουμε πλέον κάποιον παίκτη να περιμένει.

Αλλιώς στην περίπτωση που δεν υπάρχει παίκτης να περιμένει τότε δηλώνουμε το πιο πρόσφατο room στην μεταβλητή inRoom = roomno, αφαιρούμε τον χρήστη από το namespace all και των βάζουμε στο room που θα παίξει. Στην συνέχεια τον δηλώνουμε ότι είναι ο παίκτης που περιμένει να παίξει στην μεταβλητή waitingPlayer, στέλνουμε στον χρήστη με το event room το δωμάτιο στο οποίο θα παίξει και τέλος του στέλνουμε με το event message μήνυμα ότι περιμένει για συμπαίκτη.

```
if(waitingPlayer) {
    inRoom = roomno;
    sock.leave('all');
    sock.join('room-'+inRoom);
    let playersInformation = {};
    playersInformation.player1 = waitingPlayer.id;
    playersInformation.player2 = sock.id
    playersInformation.room    = inRoom;
    waitingPlayer.emit('player', 'player1');
    sock.emit('player', 'player2');
    io.to("room-"+inRoom).emit('multiplayer', playersInformation);
    sock.emit('wait');
    sock.to("room-"+inRoom).broadcast.emit('your turn to play');
    waitingPlayer = null;
}else {
    inRoom = roomno;
    sock.leave('all');
    sock.join("room-"+inRoom);
    waitingPlayer = sock;
    waitingPlayer.emit('room', inRoom);
    waitingPlayer.emit('message', 'Waiting for an opponent');
}
```

To event **multiplayer game:**

```
sock.on('multiplayer game', (data)=> { ... });
```

Αυτό το event αποθηκεύει το multiplayer παιχνίδι μας δηλαδή τους παίκτες, το room, ποιανού παίκτη είναι ο γύρος τώρα, το grid του nonogram και την πιό πρόσφατη επιλογή κελιού που έγινε στο παιχνίδι. Το data θα αναλύσουμε τι περιέχει στο κεφάλαιο με τα events του client.

```
 multiplayerGame = data;
```

To event **correct:**

```
sock.on('correct', (data) => { ... });
```

Αυτό το event ενημερώνει τους παίκτες ότι έχουν λύσει την πίστα. Ο έλεγχος για την ολοκλήρωσή της πίστας γίνεται στον client και στέλνεται το event correct στον server.

```
io.to("room-"+inRoom).emit('correct');
```

To event **update progress:**

```
sock.on('update progress', (data) => { ... });
```

Αυτό το event χρησιμοποιείται για την ανανέωση του πλέγματος της πίστας κάθε φορά που γίνεται μία επιλογή κελιού από τον παίκτη που παίζει.

```
sock.to("room-"+inRoom).broadcast.emit('update', data);
```

To event **play next level:**

```
sock.on('play next level', (data) => { ... });
```

Αυτό το event χρησιμοποιείται για το αν ο κάθε χρήστης επιθυμεί να παίξει κι άλλη πίστα με τον ίδιο συμπαίκτη.

```
io.to("room-"+inRoom).emit('play next level', data);
```

To event **multiplayer finished:**

```
sock.on('multiplayer finished', () => { ... });
```

Αυτό το event ενεργοποιείται όταν ο παίκτης έχει φτάσει και έχει τελειώσει την τελευταία πίστα του multiplayer.

```
sock.to("room-"+inRoom).broadcast.emit('multiplayer finished');
```

To event **exit-multiplayer:**

```
sock.on('exit-multiplayer', (data) => { ... });
```

Αυτό το event εκτελείται όταν ο χρήστης καθώς παίζει multiplayer πατήσει το κουμπί home και βγει απο το multiplayer. Τότε αφαιρούμε τον χρήστη απο το room που έπαιζε και τον ξανά βάζουμε στο room all. Στην συνέχεια αναλόγως με το ποιός χρήστης έχει βγεί στέλνουμε μήνυμα στον άλλον παίκτη ότι ο άλλος παίκτης βγήκε και των αφαιρούμε από το room και των προσθέτουμε στο room all. Τέλος, μηδενίζουμε την μεταβλητή inRoom = 0.

```
if(sock.id === data.player1) {
   io.to(data.player2).emit('exit-multiplayer', 'Player 1 left the lobby...');
   io.nsps['/'].sockets[data.player2].leave('room-'+inRoom);
   io.nsps['/'].sockets[data.player2].join('all');
   io.nsps['/'].sockets[sock.id].leave('room-'+inRoom);
   io.nsps['/'].sockets[sock.id].join('all');
}else if(sock.id === data.player2) {
   io.to(data.player1).emit('exit-multiplayer','Player 2 left the lobby...');
   io.nsps['/'].sockets[data.player1].leave('room-'+inRoom);
   io.nsps['/'].sockets[data.player1].join('all');
   io.nsps['/'].sockets[sock.id].leave('room-'+inRoom);
   io.nsps['/'].sockets[sock.id].join('all');
}
inRoom = 0;
```

To event **exit multiplayer waiting lobby:**

```
sock.on('exit multiplayer waiting lobby', () => { ... });
```

Αυτό το event ενεργοποιείται όταν ο χρήστης πατήσει το exit button από το μενού αναζήτησης συμπαίκτη. Κάνουμε το waitingPlayer = null γιατί δεν υπάρχει παίκτης πλέον που να περιμένει να παίξει, αφαιρούμε τον χρήστη από το room που προοριζόταν να παίξει και μηδενίζουμε την μεταβλητή inRoom = 0.

```
waitingPlayer = null;
sock.leave("room-"+inRoom);
inRoom = 0;
```

Το event **disconnect:**

```
sock.on('disconnect', () => { ... });
```

Αυτό το event ενεργοποιείται όταν ένας χρήστης αποσυνδεθεί από την σελίδα.

Μειώνουμε των αριθμό των χρηστών κατά 1 μιας και ο χρήστης έφυγε από την σελίδα και ενημερώνουμε τους υπόλοιπους χρήστες με των νέο αριθμό χρηστών. Στην συνέχεια ελέγχουμε αν ήταν σε κάποιο room και αν έπαιζε εκείνη την στιγμή multiplayer, αν έπαιζε ελέγχουμε για να βρούμε ποιός παίκτης ήταν για να μπορέσουμε να ενημερώσουμε τον άλλο χρήστη ότι ο συμπαίκτης του έφυγε. Την ενημέρωση την κάνουμε με το event player-left που στέλνουμε στον client.

```
clients --;
io.sockets.emit('refresh counter', { description: 'Players online: ' + clients });
if(inRoom != 0) {
  if(multiplayerGame) {
    if(sock.id === multiplayerGame.player1) {
      io.to(multiplayerGame.player2).emit('player-left', 'Player 1 left the lobby...');
    }else if(sock.id === multiplayerGame.player2) {
            io.to(multiplayerGame.player1).emit('player-left','Player 2 left the lobby...');
    }
  }
}
```

Το event **player left:**

```
sock.on('player-left', () => { ... });
```

Αυτό το event ενεργοποιείται όταν ένας συμπαίκτης αποσυνδεθεί από την σελίδα. Με αυτό το event αφαιρούμε των χρήστη που παραμένει στο παιχνίδι από το room που έπαιζε multiplayer και των ξανά βάζουμε στο δωμάτιο all.

```
iRoom =0;
io.nsps['/'].sockets[sock.id].leave('room-'+inRoom);
io.nsps['/'].sockets[sock.id].join('all');
```

Το event **end turn:**

```
sock.on('end-turn', () => { ... });
```

Αυτό το event ενεργοποιείται όταν ένας παίκτης έχει κάνει επιλογή κελιού, το οποίο σημαίνει ότι έχει τελειώσει ο γύρος του και είναι η σειρά του συμπαίκτη του να παίξει.

```
sock.on('end-turn', () => {
    sock.emit('end-turn');
    sock.to("room-"+inRoom).broadcast.emit('your turn to play');
});
```

Αυτά είναι τα events και ο κώδικας που έχουμε στον server για την επικοινωνία του χρήστη με τον server. Στην συνέχεια θα μιλήσουμε για τον κώδικα και τα events που χρειαζόμαστε στον client για την επικοινωνία του server με τον χρήστη.

11.2 Δημιουργία των multiplayer levels και της λειτουργικότητας επιλογής επόμενης πίστας

Δημιουργούμε στον φάκελο js το αρχείο multiplayerStages.js, στο οποίο θα έχουμε των κώδικα που χρειαζόμαστε για τις πίστες μας. Τοποθετούμε το script στο index.html μετά το script του stages.

```
<script src="js/multiplayerStages.js"></script>
```

Δημιουργούμε 2 συναρτήσεις οι οποίες θα μας βοηθήσουν να δημιουργούμε τις πίστες του multiplayer. Οι συναρτήσεις είναι η **createMultiplayerStage** και η **createNextMultiplayerStage**. Για αρχή δημιουργούμε

δύο μεταβλητές, την **turn** που θα αντιπροσωπεύει τον γύρο του χρήστη και την **wait** με την οποία λέμε στον χρήστη να περιμένει τον γύρο του.

```
let turn = false;
let wait = false;
```

Στην **createMultiplayerStage** κάνουμε τις εξής ενέργειες, κρύβουμε τα εργαλεία του singleplayer για να μην φαίνονται στο multiplayer, δηλώνουμε στην μεταβλητή multiplayerStageIndex ότι θα παίξει την πρώτη πίστα, χρησιμοποιώντας την συνάρτηση setTimeout για να δώσουμε ένα εφέ φορτώματος κάνουμε τα εξής, δηλώνουμε την κατάσταση(state) που βρίσκεται το παιχνίδι σε multiplayer, κρύβουμε το παράθυρο game-lobbie που ενημέρωσε τον χρήστη ότι βρήκε συμπαίκτη και επαναφέρουμε το μήνυμα Searching for player ... , εμφανίζουμε το container-tools και τα multiplayer-tools, καθαρίζουμε τον canvas, με την κατάλληλη css μορφοποιούμε το container μας για να μπορέσουμε να εφαρμόσουμε στην συνέχεια τον canvas μας στο κέντρο, δημιουργούμε στην μεταβλητή nonogram την πίστα που θα παίξουν οι παίκτες, ζωγραφίζουμε την πίστα στον canvas, ελέγχουμε αν δεν είναι η σειρά του εκάστοτε χρήστη αν ισχύει η συνθήκη τότε του εμφανίζουμε μήνυμα να περιμένει την σειρά του, με την συνάρτηση resetTools("multiplayer") επαναφέρουμε την επιλογή του active tool στην αρχική του θέση, μηδενίζουμε το info current progress και τέλος κλείνουμε το clients count.

```
function createMultiplayerStage() {
    $("#singleplayer-tools").hide();
    multiplayerStageIndex = 0;
    $("#exit-multiplayer-waiting-lobby").hide();
    setTimeout(function(){
            state = "multiplayer";
            $('#game-lobbie').hide();
            $('#msg').text("Searching for player...");
            $("#container-tools").show();
            $("#multiplayer-tools").show();
            clearCanvas();
            container.style.left = "50%";
            container.style.top = "50%";
            container.style.transform = "translate(-50%, -50%)";
            nonogram = new

Nonogram(multiplayerStages[multiplayerStagesNames[multiplayerStageIndex]]);
```

```
            canvas.width = nonogram.width;
            canvas.height = nonogram.height;
            canvas.style.border = "1px solid black";
            ctx.clearRect(0, 0, innerWidth, innerHeight);
            nonogram.drawGrid();
            nonogram.drawRowNumbers();
            nonogram.drawColumnNumbers();
            if(turn === false) {
          $("#waiting-screen").show();
            }
            resetTools("multiplayer");
            $("#info-current-progress").text("");
            $("#info-current-progress").text(nonogram.findProgress() + "%");
            limitBottom = nonogram.height-myLimit;
            limitRight = nonogram.width-myLimit;
            $("#clients-count").hide();
    }, 3000);
};
```

Με την συνάρτηση **createNextMultiplayerStage** δημιουργούμε την επόμενη πίστα του multiplayer. Ο κώδικας είναι παρόμοιος με των κώδικα της συνάρτησης createMultiplayerLevel με την μόνη διαφορά ότι εδώ εμφανίζεται μήνυμα στον χρήστη για την επόμενη πίστα και ελέγχουμε να δούμε αν ο χρήστης είναι ο πρώτος παίκτης, στην περίπτωση που είναι ο πρώτος του δίνουμε τον πρώτο γύρο.

```
function createNextMultiplayerStage() {
    $('#multiplayer-next-stage-popup').show();
    setTimeout(function(){
            $('#multiplayer-next-stage-popup').hide();
            nonogram = new Nonogram(multiplayerStages[multiplayerStagesNames[multiplayerStageIndex]]);
            canvas.width = nonogram.width;
            canvas.height = nonogram.height;
```

```
            ctx.clearRect(0, 0, innerWidth, innerHeight);
            nonogram.drawGrid();
            nonogram.drawRowNumbers();
            nonogram.drawColumnNumbers();

            resetTools("multiplayer");
            $("#info-current-progress").text("");
            $("#info-current-progress").text(nonogram.findProgress() + "%");
            if(player == 'player1') {
                    wait = false;
                    turn = true;
            }else {
                    wait = false;
                    turn = false;
                    $('#waiting-screen').show();
            }

    }, 3000);
};
```

Προσθέτουμε στο index.html τα tags που χρειαζόμαστε για την αλληλεπίδραση του χρήστη για το αν θέλει να παίξει την επόμενη πίστα του multiplayer με τον ίδιο συμπαίκτη. Όταν οι παίκτες θα έχουν ολοκληρώσει την πίστα σωστά, τότε θα τους εμφανίζεται το παρακάτω μήνυμα.

```
<!-- Correct for multiplayer-->
<div id="correct-multiplayer" class="black-screen">
    <div id="correct-popup">
            <h2>Correct !</h2>
            <img src="img/green-button.png" style="width:60px; height:60px; margin: 0 auto;">
            <div id="next-stage">
                    <h3>Next stage?</h3>
                    <div id="choices">
                            <div class="player-choice">
                                    Player 1
                                    <div id="player1-choice" class="choice"></div>
```

```
                        </div>
                        <div id="player-choice" class="player-choice">
                                Player 2
                                <div id="player2-choice" class="choice"></div>
                        </div>
                </div>
                <div id="votes">
                        <div id="yes" class="vote">YES</div>
                        <div id="no" class="vote">NO</div>
                </div>
        </div>

        <p id="exit-multiplayer">Exit</p>
    </div>
</div>
```

Προσθέτουμε στο index.html και το div tag με id multiplayer-next-stage-popup στο οποίο ενημερώνουμε τον χρήστη ότι δημιουργείται η επόμενη πίστα.

```
<!-- Next stage for multiplayer -->
<div id="multiplayer-next-stage-popup" class="black-screen">
    <div id="multiplayer-next-stage">
        <h2>Creating next stage...</h2>
        <p>Please wait</p>
        <p id="next-stage"></p>
    </div>
</div>
```

Προσθέτουμε και την αντίστοιχη css που χρειαζόμαστε στο multiplayer.css.

```
#correct-multiplayer {
  display: none;
}

#correct-multiplayer-popup {
  position: absolute;
```

```
  top: 50%;
  left: 50%;
  transform: translate(-50%, -50%);
  text-align: center;
  background: linear-gradient(to bottom right, #e0e0d1, white,  #999966);
  border: 2px solid black;
  border-radius: 30px;
  padding: 10px;
  box-shadow: 1px 3px 5px grey;
}

#correct-multiplayer-popup p {
  padding: 0 auto;
  margin: 0 auto;
  cursor: pointer;
  font-weight: bold;
  background: linear-gradient(to bottom right, #e0e0d1, grey);
  border-radius: 30px;
  border: 1px solid black;
}

#correct-multiplayer-popup p:hover {
  background: linear-gradient(to bottom right, #e0e0d1, white);
}

#correct-multiplayer-popup h2 {
    position: relative;
    margin: 0 auto;
    padding: 0 auto;
    text-align: center;
}

#multiplayer-next-stage-popup {
    display: none;
}
```

```
#multiplayer-next-stage {
    position: absolute;
    padding: 10px;
    top: 50%;
    left: 50%;
    transform: translate(-50%, -50%);
    min-width: 300px;
    text-align: center;
    background: linear-gradient(to bottom right, #e0e0d1, white,  #999966);
    border: 2px solid black;
    border-radius: 30px;
    box-shadow: 1px 3px 5px grey;
    font-size: 13px;
}

#multiplayer-next-stage-popup h2 {
    border-bottom: 3px solid black;
}

#votes {
    display: inline-block;
}

.vote {
    float: left;
    margin: 5px;
    border: 1px solid black;
    border-radius: 30px;
    padding: 5px;
            background: linear-gradient(to bottom right, #99ccff, #3399ff, blue);
            color: white;
            width: 32px;
}
```

```
.vote:hover {
    cursor: pointer;
            background: linear-gradient(to bottom right, #3399ff, #3399ff, blue);

}

.vote:active {
            background: linear-gradient(to bottom right, black, #3399ff, blue);
            color: grey;
}

.player-choice {
    display: inline-block;
    margin: 5px;
}

.choice {
    border: 1px solid black;
    background-color: white;
    width: 10px;
    height: 10px;
    margin: 0 auto;
}

.choice-yes {
    background-color: green;
}

.choice-no {
    background-color: red;
}
```

Και προσθέτουμε στο αρχείο **nonogram.multiplayer.js** των κώδικα για την αλληλεπίδραση του χρήστη με τις επιλογές yes, no και exit.

```
$('#yes').click(function() {
    $('#'+ player +'-choice').addClass('choice-yes');
    let data = {
            player: player,
            choice: 'yes'
    };
    if(player == 'player1') {
            player1Choice = 'yes';
    }else if(player == 'player2Choice') {
            player2Choice = 'yes';
    }
    sock.emit('choice', data);
});

$('#no').click(function() {
    $('#'+ player +'-choice').addClass('choice-no');
    let data = {
            player: player,
            choice: 'no'
    };
    if(player == 'player1') {
            player1Choice = 'no';
    }else if(player == 'player2Choice') {
            player2Choice = 'no';
    }
    sock.emit('choice', data);
});

$("#exit-multiplayer").click(function(){
    $("#container-tools").hide();
    ctx.clearRect(0, 0, canvas.width, canvas.height);
    container.style.transform = "none";
    container.style.left = "0%";
    container.style.top = "0%";
```

```
    canvas.width = innerWidth;
    canvas.height = innerHeight;
    canvas.style.border = "none";
    state = "menu";
    $("#correct-multiplayer").hide();
    $("#levels").show();
    $("#clients-count").show();
});
```

Τέλος, προσθέτουμε τις πίστες του multiplayer στο αρχείο multiplayerStages.js οι οποίες θα είναι πέντε.

```
let multiplayerStagesNames = ['android', 'cuppa', 'skull', 'clown', 'candle'];
let multiplayerStages = {
    'android': [[0,1,1,1,0],
                [1,0,1,0,1],
                [1,1,1,1,1],
                [0,0,0,0,0],
                [1,1,1,1,1],],

    'cuppa'    : [[0,0,1,0,1,0,1,0,0,0],
                  [0,0,1,0,1,0,1,0,0,0],
                  [0,0,0,0,0,0,0,0,0,0],
                  [0,1,1,1,1,1,1,1,0,0],
                  [0,1,1,0,1,1,1,1,1,1],
                  [0,1,1,0,1,1,1,1,0,1],
                  [0,1,1,1,1,1,1,1,1,0],
                  [0,1,1,1,1,1,1,1,0,0],
                  [1,0,1,1,1,1,1,0,0,1],
                  [0,1,1,1,1,1,1,1,1,0],],

    'skull':   [[0,1,1,1,0],
                 [1,1,1,1,1],
                 [1,0,1,0,1],
                 [1,1,1,1,1],
                [0,1,0,1,0],],
```

```
  'clown': [[0,0,0,1,1,1,1,0,0,0],
                    [0,0,1,0,0,1,1,1,0,0],
                    [1,1,1,1,1,1,1,1,1,1],
                    [0,1,0,1,0,0,1,0,1,0],
                    [1,0,0,0,1,1,0,0,0,1],
                    [1,0,0,0,1,1,0,0,0,1],
                    [0,1,0,0,0,0,1,0,1,0],
                    [0,1,0,0,1,1,0,0,1,0],
                    [0,0,1,0,0,0,0,1,0,0],
                    [0,1,1,1,1,1,1,1,1,0],],

  'candle': [ [1,0,1,0,1,1,1,1,1,1,0,0,1,1,1],
                          [1,1,1,0,0,1,1,0,1,1,1,0,0,1,1],
                          [1,1,1,0,1,0,1,1,1,1,1,0,0,0,1],
                          [1,1,1,1,0,0,1,1,1,1,0,0,0,0,1],
                          [1,1,1,1,1,0,1,1,1,1,1,0,0,0,1],
                          [1,1,1,1,1,0,1,1,1,1,1,0,0,0,1],
                          [1,1,1,0,0,0,0,0,1,1,1,0,0,1,1],
                          [1,1,1,0,0,0,0,0,1,1,1,0,0,1,1],
                          [1,1,1,0,1,0,0,0,1,1,1,1,1,1,1],
                          [1,1,1,0,0,0,0,0,1,1,1,1,1,1,1],
                          [1,1,1,0,0,0,0,0,1,1,1,0,1,1,1],
                          [1,1,1,0,0,0,0,0,1,1,0,1,0,1,1],
                          [0,1,1,0,0,0,0,0,1,1,0,0,1,1,1],
                          [1,0,0,0,0,0,0,0,0,0,1,1,1,1,1],
                          [1,1,1,1,1,1,1,1,1,1,1,1,1,1,1],]
};
```

Αυτά για όσον αφορά τις πίστες του multiplayer, στο επόμενο κεφάλαιο θα μιλήσουμε για τα events που θα χρειαστούμε στον client.

11.3 Τα Multiplayer events του client

Ξεκινάμε δημιουργώντας στο αρχείο **nonogram.multiplayer.js** την class για το object NonogramMultiplayer για να έχουμε αποθηκευμένα στην πλευρά του client τα δεδομένα του multiplayer παιχνιδιού που παίζει ο χρήστης.

```
class NonogramMultiplayer {
    constructor(p1, p2, room) {
            this.player1 = p1;
            this.player2 = p2;
            this.roomId = room;

            this.turn = null;
            this.nonogram = null;
            this.choice = null;
    }
};
```

Φτιάχνουμε τις μεταβλητές gameRoom, multiplayerGame, player και multiplayerStageIndex και στην συνέχεια αναλύουμε τα events που θα χρειαστούμε στον client.

```
let gameRoom;
let multiplayerGame;
let player;
let multiplayerStageIndex;
```

Το event **multiplayer:**

```
sock.on('multiplayer', (game) => { ... });
```

Σε αυτό το event ενημερωνόμαστε από τον server ότι ο χρήστης βρήκε συμπαίκτη, δημιουργούμε στην μεταβλητή multiplayerGame το object NonogramMultiplayerGame μετα δεδομένα του παιχνιδιού μας που μας έστειλε ο server, με την συνάρτηση createMultplayerLevel() δημιουργούμε την πίστα μας και στο τέλος στέλνουμε τα δεδομένα του παιχνιδιού μας στον server.

```
$('#msg').text("Player found!");
multiplayerGame = new NonogramMultiplayer(game.player1, game.player2, game.room);
createMultiplayerStage();
```

```
sock.emit('multiplayer game', multiplayerGame);
```

To event **your turn to play:**

```
sock.on('your turn to play', () => { ... });
```

Σε αυτό το event ενημερώνουμε των χρήστη ότι είναι ο γύρος του και μπορεί να παίξει.

```
wait = false;
turn = true;
$("#waiting-screen").hide();
```

Στο αρχείο **index.html** προσθέτουμε μέσα στο div tag με id game, πρίν το div tag με id containers tool, το div tag με id waiting screen με το οποίο θα δείχνουμε στον χρήστη ότι δεν μπορεί να παίξει.

```
<div id="waiting-screen" class="black-screen">
    <h1>Wait your turn..</h1>
</div>
```

Προσθέτουμε στο αρχείο **multiplayer.css** την css που χρειαζόμαστε για την μορφοποίηση που θέλουμε.

```
#waiting-screen {
    display: none;
    color: white;
}

#waiting-screen h1 {
    text-align: center;
    margin-top: 58%;
}
```

To event **correct:**

```
sock.on('correct', () => { ... });
```

Σε αυτό το event ενημερώνουμε τον χρήστη ότι έχει λυθεί η πίστα σωστά και του εμφανίζεται το μενού για να επιλέξει άμα θέλει να παίξει και την επόμενη πίστα με τον ίδιο συμπαίκτη ή να επιστρέψει στο κεντρικό μενού.

```
$('#waiting-screen').hide();
$("#correct-multiplayer").show();
```

To event **end-turn:**

```
sock.on('end-turn', () => { ... });
```

Σε αυτό το event ενημερώνουμε τον χρήστη ότι τελείωσε ο γύρος του και πρέπει να περιμένει τον γύρο του.

```
$('#waiting-screen').show();
```

To event **room:**

```
sock.on('room', (room) => { ... });
```

Σε αυτό το event αποθηκεύουμε το room στο οποίο θα παίξει ο χρήστης στην μεταβλητή gameRoom.

```
gameRoom = 'room-'+room;
```

To event **player:**

```
sock.on('player', (number) => { ... });
```

Σε αυτό το event αποθηκεύουμε των αριθμό του παίκτη, άμα είναι ο πρώτος η ο δεύτερος.

```
player = number;
```

To event **multiplayer finished:**

```
sock.on('multiplayer finished', () => { ... });
```

Σε αυτό το event ενημερώνουμε τον χρήστη ότι έχει τελειώσει το multiplayer και του εμφανίζεται ένα pop up το οποίο τον ενημερώνει ότι τελείωσε το παιχνίδι και επιστρέφει τον χρήστη στο κεντρικό μενού.

```
$('#waiting-screen').hide();

$('#multiplayer-finished-popup').show();
```

Προσθέτουμε στο index.html το div tag me id multiplayer-finished-popup,

```
<!-- Multiplayer finished -->
<div id="multiplayer-finished-popup" class="black-screen">
    <div id="multiplayer-finished">
            <h1>Congratulation!!</h1>
            <p>You finished the multiplayer!</p>
            <p>Thanks for playing</p>
            <div id="close-multiplayer">Exit</div>
    </div>
</div>
```

στο αρχείο multiplayer.css προσθέτουμε την css που χρειαζόμαστε για την μορφοποίηση που θέλουμε

```
#multiplayer-finished-popup {
        display: none;
}

#multiplayer-finished {
        position: absolute;
        padding: 10px;
        margin-top: -100px;
        top: 50%;
        left: 50%;
        transform: translateX(-50%);
        width: 300px;
        height: auto;
        text-align: center;
        background: linear-gradient(to bottom right, #e0e0d1, white,  #999966);
        border: 2px solid black;
        border-radius: 30px;
```

```
        box-shadow: 1px 3px 5px grey;
}

#close-multiplayer {
        background: linear-gradient(to bottom right, red, grey);
        cursor: pointer;
        font-weight: bold;
        border-radius: 20px;
        border: 1px solid black;
        color: white;
        display: inline-block;
        padding: 10px;
}

#close-multiplayer:hover {
        background: linear-gradient(to top right, red, red, white);
}

#close-multiplayer:active {
        background: linear-gradient(to top right, #800000, #800000, grey);
}
```

και στο αρχείο main-menu.js προσθέτουμε των κώδικα που χρειαζόμαστε για την αλληλεπίδραση του χρήστη με το exit button.

```
$('#close-multiplayer').click(function() {
    $("#container-tools").hide();
    ctx.clearRect(0, 0, canvas.width, canvas.height);
    container.style.transform = "none";
    container.style.left = "0%";
    container.style.top = "0%";
    canvas.width = innerWidth;
    canvas.height = innerHeight;
    canvas.style.border = "none";
    state = "menu";
```

```
    $("#multiplayer-finished-popup").hide();
    $("#menu").show();
    $("#clients-count").show();
});
```

To event **exit-multiplayer:**

```
sock.on('exit-multiplayer', (data) => { ... });
```

Σε αυτό το event ενημερώνουμε τον χρήστη ότι ο συμπαίκτης του έχει αποχωρήσει από το παιχνίδι.

```
if($("#waiting-screen").show()) {
    $("#waiting-screen").hide();
}
$('#container-tools').hide();
$('#player-left-info').text(data);
$('#player-left').show();
```

Προσθέτουμε στο αρχείο **index.html** μέσα στο div tag game, πρίν το div tag container-tools τα tags που χρειαζόμαστε για να μπορέσουμε να ενημερώσουμε τον χρήστη,

```
<!-- Player left the lobby -->
<div id="player-left" class="black-screen">
    <div id="player-left-popup">
        <h3 id="player-left-info"></h3>
        <img src="img/exit.png">
        <p id="player-left-exit-to-menu">Exit to menu</p>
    </div>
</div>
```

στο αρχείο **multiplayer.css** προσθέτουμε την css που θέλουμε για την μορφοποίηση,

```
#player-left {
    display: none;
    color: white;
}
```

```
#player-left-popup {
    position: relative;
    text-align: center;
    background: linear-gradient(to bottom right, #e0e0d1, white,  #999966);
    border: 2px solid black;
    border-radius: 30px;
    padding: 10px;
    box-shadow: 1px 3px 5px grey;
    width: 230px;
    top: 50%;
    left: 50%;
    transform: translate(-50%, 50%);

    font-size: 22px;
}

#player-left-info {
    color: black;
}

#player-left-exit-to-menu {
  cursor: pointer;
  font-weight: bold;
  background: rgba(100,100,100,0.7);
  border-radius: 20px;
  border: 1px solid black;
  color: white;
  display: inline-block;
  padding: 10px;
}

#player-left-popup img {
    display: block;
    width:60px;
```

```
    height:60px;
    margin: 0 auto;
    padding: 10px;
}
```

προσθέτουμε στον φάκελο img την εικόνα exit.png που θα χρησιμοποιήσουμε

και τέλος προσθέτουμε την javascript που χρειαζόμαστε για την αλληλεπίδραση του χρήστη με την επιλογή exit to menu.

```
$('#player-left-exit-to-menu').click( () => {
    $("#player-left").hide();
    $("#container-tools").hide();
    ctx.clearRect(0, 0, canvas.width, canvas.height);
    container.style.transform = "none";
    container.style.left = "0%";
    canvas.width = innerWidth;
    canvas.height = innerHeight;
    canvas.style.border = "none";
    state = "menu";
    $("#menu").show();
    $("#clients-count").show();
});
```

Έχουμε φτιάξει αυτό το αποτέλεσμα.

To event **player-left:**

```
sock.on('player-left', (data) => { ... });
```

Σε αυτό το event στέλνουμε μήνυμα στον χρήστη που παραμένει στο παιχνίδι ότι ο συμπαίκτης του έφυγε.

```
if($("#waiting-screen").show()) {
    $("#waiting-screen").hide();
}
$('#container-tools').hide();
$('#player-left-info').text(data);
$('#player-left').show();
sock.emit('player-left');
```

To event **update:**

```
sock.on('update', (data) => { ... });
```

Σε αυτό το event ενημερώνουμε το πλέγμα του παιχνιδιού μας με την επιλογή που έχει κάνει ο εκάστοτε παίκτης στον άλλον παίκτη. Η συνάρτηση είναι παρόμοια με αυτή που έχουμε και στο singleplayer με την διαφορά ότι έχουμε και την συνάρτηση **strokeTeamMateChoice** η οποία μας δείχνει τι έχει επιλέξει ο συμπαίκτης του χρήστη. Για την συνάρτηση drawRedCell θα μιλήσουμε σε επόμενο κεφάλαιο.

```
sock.on('update', (data) => {
        ctx.save();
        ctx.translate(originX,originY);
        ctx.scale(scaleFactor,scaleFactor);
        if(data.dataType === "fill cell") {
        if(data.fillCellChoice === "default") {
        nonogram.emptyGrid[data.cell].value = data.value;
        nonogram.emptyGrid[data.cell].playerChoice = "team mate";
        if(nonogram.emptyGrid[data.cell].value === 1) {
        nonogram.drawRedCell(nonogram.emptyGrid[data.cell]);
        nonogram.strokeTeamMateChoice(nonogram.emptyGrid[data.cell]);
        nonogram.drawPreview(nonogram.emptyGrid[data.cell]);
        }else if(nonogram.emptyGrid[data.cell].value === 2) {
        nonogram.drawWhiteCell(nonogram.emptyGrid[data.cell]);
        nonogram.drawXCell(nonogram.emptyGrid[data.cell]);
        nonogram.strokeTeamMateChoice(nonogram.emptyGrid[data.cell]);
        nonogram.drawPreview(nonogram.emptyGrid[data.cell]);
        }else{
        nonogram.drawWhiteCell(nonogram.emptyGrid[data.cell]);
        nonogram.strokeTeamMateChoice(nonogram.emptyGrid[data.cell]);
        nonogram.drawPreview(nonogram.emptyGrid[data.cell]);
        }
        }else if(data.fillCellChoice === "black") {
        nonogram.emptyGrid[data.cell].value = data.value;
        if(nonogram.emptyGrid[data.cell].value === 1) {
        nonogram.drawRedCell(nonogram.emptyGrid[data.cell]);;
        nonogram.strokeTeamMateChoice(nonogram.emptyGrid[data.cell]);
        nonogram.drawPreview(nonogram.emptyGrid[data.cell]);
        }else{
        nonogram.drawWhiteCell(nonogram.emptyGrid[data.cell]);
```

```
            nonogram.strokeTeamMateChoice(nonogram.emptyGrid[data.cell]);
            nonogram.drawPreview(nonogram.emptyGrid[data.cell]);
            }
            }else if(data.fillCellChoice === "x") {
            nonogram.emptyGrid[data.cell].value = data.value;
            if(nonogram.emptyGrid[data.cell].value === 2) {
            nonogram.drawWhiteCell(nonogram.emptyGrid[data.cell]);
            nonogram.drawXCell(nonogram.emptyGrid[data.cell]);
            nonogram.strokeTeamMateChoice(nonogram.emptyGrid[data.cell]);
            nonogram.drawPreview(nonogram.emptyGrid[data.cell]);
            }else{
            nonogram.drawWhiteCell(nonogram.emptyGrid[data.cell]);
            nonogram.strokeTeamMateChoice(nonogram.emptyGrid[data.cell]);
            nonogram.drawPreview(nonogram.emptyGrid[data.cell]);
            }
            }else if(data.fillCellChoice === "white") {
            nonogram.emptyGrid[data.cell].value = data.value;
            if(nonogram.emptyGrid[data.cell].value === 0) {
            nonogram.drawWhiteCell(nonogram.emptyGrid[data.cell]);
            nonogram.strokeTeamMateChoice(nonogram.emptyGrid[data.cell]);
            nonogram.drawPreview(nonogram.emptyGrid[data.cell]);
            }
            }
            }else if(data.dataType === "fill cell row numbers grid") {
            nonogram.rowNumbersGrid[data.cell].value = data.value;

            ctx.lineWidth = 3;
            ctx.beginPath();
            if(nonogram.rowNumbersGrid[data.cell].value == 1) {
            ctx.strokeStyle = "red";
            ctx.moveTo(nonogram.rowNumbersGrid[data.cell].x+3, (nonogram.rowNumbersGrid[data.cell].y +
nonogram.blockSize)-3);
            ctx.lineTo((nonogram.rowNumbersGrid[data.cell].x + nonogram.blockSize)-3,
nonogram.rowNumbersGrid[data.cell].y+3);
            }else{
```

```
		ctx.fillStyle = "#e0e0d1";
		ctx.fillRect(nonogram.rowNumbersGrid[data.cell].x+2, nonogram.rowNumbersGrid[data.cell].y+2,
nonogram.rowNumbersGrid[data.cell].w-3, nonogram.rowNumbersGrid[data.cell].h-3);
		ctx.fillStyle = "black";
		if(nonogram.rowNumbersGrid[data.cell].number < 10) {
			ctx.font = (nonogram.blockSize)+"px Arial";
			ctx.fillText(nonogram.rowNumbersGrid[data.cell].number,
			(nonogram.rowNumbersGrid[data.cell].x+(Math.floor(nonogram.blockSize/4))),

(nonogram.rowNumbersGrid[data.cell].y+(nonogram.blockSize-Math.floor(nonogram.blockSize/6))));
		}else{
			ctx.font = (nonogram.blockSize-3)+"px Arial";
			ctx.fillText(nonogram.rowNumbersGrid[data.cell].number,
			(nonogram.rowNumbersGrid[data.cell].x),

(nonogram.rowNumbersGrid[data.cell].y+(nonogram.blockSize-Math.floor(nonogram.blockSize/5))));
		}
		}
		ctx.stroke();
		ctx.closePath();
		}else if(data.dataType === "fill cell column numbers grid") {
		nonogram.columnNumbersGrid[data.cell].value = data.value;
		ctx.lineWidth = 3;
		ctx.beginPath();
		if(nonogram.columnNumbersGrid[data.cell].value === 1) {
		ctx.strokeStyle = "red";
		ctx.moveTo(nonogram.columnNumbersGrid[data.cell].x+3,
(nonogram.columnNumbersGrid[data.cell].y + nonogram.blockSize)-3);
		ctx.lineTo((nonogram.columnNumbersGrid[data.cell].x + nonogram.blockSize)-3,
nonogram.columnNumbersGrid[data.cell].y+3);
		}else{
		ctx.fillStyle = "#e0e0d1";
		ctx.fillRect(nonogram.columnNumbersGrid[data.cell].x+2,
nonogram.columnNumbersGrid[data.cell].y+2, nonogram.columnNumbersGrid[data.cell].w-3,
nonogram.columnNumbersGrid[data.cell].h-3);
```

```
        ctx.fillStyle = "black";
        if(nonogram.columnNumbersGrid[data.cell].number < 10) {
                ctx.font = (nonogram.blockSize)+"px Arial";
                ctx.fillText(nonogram.columnNumbersGrid[data.cell].number,
                (nonogram.columnNumbersGrid[data.cell].x+(Math.floor(nonogram.blockSize/4))),

(nonogram.columnNumbersGrid[data.cell].y+(nonogram.blockSize-Math.floor(nonogram.blockSize/6))));
        }else{
                ctx.font = (nonogram.blockSize-3)+"px Arial";
                ctx.fillText(nonogram.columnNumbersGrid[data.cell].number,
                (nonogram.columnNumbersGrid[data.cell].x),

(nonogram.columnNumbersGrid[data.cell].y+(nonogram.blockSize-Math.floor(nonogram.blockSize/5))));
        }
        }
        ctx.stroke();
        ctx.closePath();
        }
        ctx.restore();
        $("#info-current-progress").text("");
        $("#info-current-progress").text(nonogram.findProgress() + "%");
});
```

Προσθέτουμε στο αρχείο **nonogram.drawing.js** την συνάρτηση strokeTeamMateChoice

```
Nonogram.prototype.strokeTeamMateChoice = function(cell) {
        if(this.previousTeamMateChoice.active) {
        ctx.beginPath();
        for(let i=0; i<this.previousTeamMateChoice.cell.length; i++) {
        if(this.previousTeamMateChoice.cell[i].value === 1) {
        this.drawWhiteCell(this.previousTeamMateChoice.cell[i]);
        this.drawRedCell(this.previousTeamMateChoice.cell[i]);
        }else if(this.previousTeamMateChoice.cell[i].value === 2) {
        this.drawWhiteCell(this.previousTeamMateChoice.cell[i]);
```

```
        this.drawXCell(this.previousTeamMateChoice.cell[i]);
        }else{
        ctx.fillStyle = "white";
        ctx.fillRect(this.previousTeamMateChoice.cell[i].x + 2,
                        this.previousTeamMateChoice.cell[i].y + 2,
                        this.previousTeamMateChoice.cell[i].w - 4,
                        this.previousTeamMateChoice.cell[i].h - 4);
        }
        }
        ctx.stroke();
        ctx.closePath();
        this.previousTeamMateChoice.cell = [];
        }
        this.previousTeamMateChoice.cell.push(cell);
        this.previousTeamMateChoice.active = true;
        ctx.strokeStyle = "blue";
        ctx.lineWidth   = 4;
        ctx.strokeRect(cell.x+5, cell.y+5, this.blockSize-10, this.blockSize-10);
};
```

και προσθέτουμε στο object nonogram την ιδιότητα previousTeamMateChoice

```
this.previousTeamMateChoice = {
    active: false
};
this.previousTeamMateChoice.cell = [];
```

To event **choice:**

```
sock.on('choice', () => { ... });
```

Σε αυτό το event ελέγχουμε αν οι παίκτες επιθυμούν να παίξουν την επόμενη πίστα του multiplayer μαζί. Αν όχι τότε επιστρέφουν στο αρχικό μενού.

```
sock.on('choice', (data) => {
	if(data.player == "player1") {
		player1Choice = data.choice;
		if(player1Choice == 'yes') {
			$('#player1-choice').addClass('choice-yes');
		}else if(player1Choice == 'no') {
			$('#player1-choice').addClass('choice-no');
		}
	}else if(data.player == "player2") {
		player2Choice = data.choice;
		if(player2Choice == 'yes') {
			$('#player2-choice').addClass('choice-yes');
		}else if(player2Choice == 'no') {
			$('#player2-choice').addClass('choice-no');
		}
	}

	if(player1Choice == 'yes' && player2Choice == 'yes') {
		$('#correct-multiplayer').hide();
		multiplayerStageIndex++;
		createNextMultiplayerStage();
		$('#player1-choice').removeClass('choice-yes');
		$('#player2-choice').removeClass('choice-yes');
		player1Choice = "";
		player2Choice = "";
	}
});
```

11.4 Δημιουργία multiplayer tools

Όσον αφορά τα tools του multiplayer θα έχουμε τα κλασικά tools default, black, x, white και το extra tool το home. Προσθέτουμε στο αρχείο **tools.js** την συνάρτηση createMultiPlayerTools στην αρχή του αρχείου μετά την δημιουργία των εργαλείων του singleplayer, η οποία θα δημιουργεί τα εργαλεία μας και θα τα προσθέτει στο div tag tools σε ένα καινούργιο div tag με id multiplayer-tools,

```
function createMultiPlayerTools() {
    const multiPlayerTools = ['default', 'black', 'x', 'white'];
    const multiPlayerExtraTools = ['home'];
    const tools = document.getElementById("tools");
    const multiplayer = document.createElement('div');
    multiplayer.id = "multiplayer-tools";
    tools.appendChild(multiplayer);

    for(let i=0; i<multiPlayerTools.length; i++) {
            var li = document.createElement('li');
            li.classList.add("tool");
            var div = document.createElement('div');
            div.className = multiPlayerTools[i];
            var img = document.createElement('img');
            img.src = "img/" + multiPlayerTools[i] + ".png";
            div.appendChild(img);
            li.appendChild(div);
            multiplayer.appendChild(li);
    }

    for(let i=0; i<multiPlayerExtraTools.length; i++) {
            var li = document.createElement('li');
            li.classList.add("tool");
            var div = document.createElement('div');
            div.className = multiPlayerExtraTools[i];
            var img = document.createElement('img');
            img.src = "img/" + multiPlayerExtraTools[i] + ".png";
            div.appendChild(img);
            li.appendChild(div);
            multiplayer.appendChild(li);
    }
    multiplayer.firstElementChild.classList.add("active");
};
```

καλούμε την συνάρτηση για να δημιουργήσουμε τα εργαλεία

```
createMultiPlayerTools();
```

και στην συνέχεια προσθέτουμε των κώδικα ο οποίος μας επιτρέπει να μετατρέπουμε το εργαλείο που έχουμε επιλέξει σε active για να ξέρουμε ποιο έχουμε επιλέξει.

```
let multiplayer = document.getElementById("multiplayer-tools");
let multiplayerTools = multiplayer.getElementsByClassName("tool");

for (let i = 0; i < multiplayerTools.length; i++) {
  multiplayerTools[i].addEventListener("click", function() {
            let current = multiplayer.getElementsByClassName("active");

            if(typeof current[0] !== 'undefined') {
            current[0].className = current[0].className.replace(" active", "");
            }

            this.className += " active";
  });
}
```

Στην συνάρτηση resetTools τοποθετούμε των κώδικα που χρειαζόμαστε για να γίνεται reset το active tool του multiplayer όταν ο χρήστης βγει από την πίστα.

```
function resetTools(toolContainer) {
    let singleplayer = document.getElementById("singleplayer-tools");
    let multiplayer = document.getElementById("multiplayer-tools");
    let currentTool;
    let tools;


    if(toolContainer === "singleplayer") {
        currentTool = singleplayer.getElementsByClassName("active");
                tools = singleplayer.getElementsByClassName("tool");
                currentTool[0].className = currentTool[0].className.replace(" active",

           "");
```

```
            tools[0].className += " active";
    }else if(toolContainer === "multiplayer") {
            currentTool = multiplayer.getElementsByClassName("active");
            tools = multiplayer.getElementsByClassName("tool");
            currentTool[0].className = currentTool[0].className.replace(" active",

        "");
            tools[0].className += " active";
    }
};
```

Προσθέτουμε την css που χρειαζόμαστε στον φάκελο tools.css

```
#multiplayer-tools {
  display: none;
}
```

Όταν θα πατάει ο χρήστης το εργαλείο home όταν παίζει multiplayer θέλουμε να στέλνει μήνυμα στον server ότι ο παίκτης βγήκε από το lobby για να μπορέσουμε να ενημερώσουμε και τον συμπαίκτη του ότι το παιχνίδι τελείωσε. Στο αρχείο **tools.js** τροποποιούμε των κώδικα του εργαλείου home. Επιπροσθέτως όταν ο χρήστης πατήσει το εργαλείο home θα εμφανίζεται ξανά το clients count και απενεργοποιούμε το εργαλείο drag άμα ήταν ενεργό.

```
$(".home").click(function(){
    if(state == "multiplayer") {
            if(turn === false) {
                    $("#waiting-screen").hide();
            }
            sock.emit('exit-multiplayer', multiplayerGame);
            currentLevel = "none";
            turn = false;
            wait = false;
    }

    $("#container-tools").hide();
    ctx.clearRect(0, 0, canvas.width, canvas.height);
```

```
    container.style.transform = "none";
    container.style.left = "0%";
    container.style.top = "0%";
    canvas.width = innerWidth;
    canvas.height = innerHeight;
    canvas.style.border = "none";
    state = "menu";
    $("#menu").show();
    $("#clients-count").show();
    if($('#top').show()) {
            $('#top').hide();
            $('#bottom').hide();
            $('#left').hide();
            $('#right').hide();
    }
});
```

Στο επόμενο κεφάλαιο θα μιλήσουμε για την επιλογή και το μαρκάρισμα των κελιών στο multiplayer.

11.5 Επιλογή και μαρκάρισμα των κελιών στο multiplayer

Ο τρόπος με τον οποίο θέλουμε να επιλέγουν οι παίκτες τα κελιά στο multiplayer είναι ο εξής. Ο πρώτος παίκτης όταν κάνει επιλογή κελιού τότε θα του εμφανίζεται μήνυμα ότι θα πρέπει να περιμένει τον γύρο του για να ξανά παίξει. Η επιλογή που έκανε στέλνεται στον server με το event update progress, στο οποίο στέλνεται η μεταβλητή gameData που περιέχει την επιλογή του χρήστη. Η επιλογή gameData παίρνει τα δεδομένα της από την συνάρτηση multiplayerFillCels.

Θέλουμε να διαφοροποιήσουμε με χρώμα τις επιλογές των κελιών που κάνουν οι παίκτες. Πάντα ο χρήστης που παίζει θα κάνει επιλογές με μπλε χρώμα και ο συμπαίκτης του θα έχει κόκκινο χρώμα. Για να το πετύχουμε αυτό θα προσθέσουμε μία καινούργια ιδιότητα στο object Cell την playerChoice. Όταν

δημιουργείται το grid της πίστας οριοθετούμε την ιδιότητα. Στο αρχείο **nonogram.js** προσθέτουμε των κώδικα που χρειαζόμαστε

```
function Cell(w, h, x, y, value, playerChoice) {
    this.w = w;
    this.h = h;
    this.x = x;
    this.y = y;
    this.value = value;
    this.playerChoice = playerChoice;
};
```

```
this.emptyGrid = [];
for(var i=this.maxColumnNumberSize*this.blockSize; i<this.height;

i+=this.blockSize) {
    for(var y=this.maxRowNumberSize*this.blockSize; y<this.width;

    y+=this.blockSize) {
        this.emptyGrid.push(new Cell(this.blockSize, this.blockSize, y, i, 0,

        ""));
    }
}
```

Ξεκινάμε δημιουργώντας την συνάρτηση multiplayerFillCels με την οποία θα ζωγραφίζουμε τα κελιά της πίστας, αναλόγως με το πιο κελί και πιο εργαλείο έχει επιλέξει ο χρήστης. Προσθέτουμε στο αρχείο **nonogram.drawing.js** τις συναρτήσεις multiplayerFillCels, multiplayerStrokeRowNumbers, multiplayerStrokeColumnNumbers, drawRedCell, drawBlueCell και strokeMultiplayerCurrentChoice.

```
Nonogram.prototype.multiplayerFillCels = function(mouseX, mouseY) {
            var gameData = {
            dataType:          "",
            fillCellChoice: "",
```

```
cell:           0,
value:          0,
room:           gameRoom,
originX:        originX,
originY:        originY,
scaleFactor:    scaleFactor
};
ctx.lineWidth = 3;
for(var i=0; i<this.rowNumbersGrid.length; i++) {
if(mouseX >= this.rowNumbersGrid[i].x && mouseY >= this.rowNumbersGrid[i].y &&
mouseX <= (this.rowNumbersGrid[i].x + this.blockSize) &&
mouseY <= (this.rowNumbersGrid[i].y + this.blockSize)) {
gameData = nonogram.multiplayerStrokeRowNumbersChoice(i, gameData);
return gameData;
}
}
for(var i=0; i<this.columnNumbersGrid.length; i++) {
if(mouseX >= this.columnNumbersGrid[i].x && mouseY >= this.columnNumbersGrid[i].y &&
mouseX <= (this.columnNumbersGrid[i].x + this.blockSize) && mouseY <=
(this.columnNumbersGrid[i].y + this.blockSize)) {
gameData = nonogram.multiplayerStrokeColumnNumbersChoice(i, gameData);
return gameData;
}
}
var block = this.blockSize;
var columnSize = this.maxColumnNumberSize;
var columnLength = this.levelGrid.length;
var rowLength = this.levelGrid[0].length;
var rowSize = this.maxRowNumberSize;
if(this.fillCellChoice == "default") {
for(var i=0;i<this.emptyGrid.length;i++) {
var x = this.emptyGrid[i].x, y = this.emptyGrid[i].y;
var value = this.emptyGrid[i].value;
var width = this.emptyGrid[i].w, height = this.emptyGrid[i].h;
var xPos = ((x - (rowSize * block)) / block) * Math.floor(((rowSize * block) / rowLength)) - 2;
```

```
        var yPos = ((y - (columnSize * block)) / block) * Math.floor(((columnSize * block) /
columnLength)) - 2;
        if(mouseX >= x && mouseY >= y && mouseX <= (x + block) && mouseY <= (y + block)) {
        if(value == 0) {
                this.emptyGrid[i].value = 1;
                this.drawBlueCell(this.emptyGrid[i]);
                this.drawPreview(this.emptyGrid[i]);
                this.strokeMultiplayerCurrentChoice(this.emptyGrid[i]);
                gameData.value  = 1;
        }else if(value == 1) {
                this.emptyGrid[i].value = 2;
                this.drawWhiteCell(this.emptyGrid[i]);
                this.drawXCell(this.emptyGrid[i]);
                this.drawPreview(this.emptyGrid[i]);
                this.strokeMultiplayerCurrentChoice(this.emptyGrid[i]);
                gameData.value  = 2;
        }else {
                this.emptyGrid[i].value = 0;
                ctx.fillStyle = "white";
                ctx.fillRect(x + 2, y + 2, width - 3, height - 3);
                this.drawPreview(this.emptyGrid[i]);
                this.strokeMultiplayerCurrentChoice(this.emptyGrid[i]);
                gameData.value  = 0;
        }
        this.emptyGrid[i].playerChoice = "yours";
        gameData.dataType = "fill cell";
        gameData.fillCellChoice = "default";
        gameData.cell  =  i;
        return gameData;
        }
        }
        }else if(this.fillCellChoice == "black") {
        for(var i=0;i<this.emptyGrid.length;i++) {
        var x = this.emptyGrid[i].x, y = this.emptyGrid[i].y;
        var value = this.emptyGrid[i].value;
```

```
var width = this.emptyGrid[i].w, height = this.emptyGrid[i].h;
var xPos = ((x - (rowSize * block)) / block) *
        Math.floor(((rowSize * block) / rowLength)) - 2;
var yPos = ((y - (columnSize * block)) / block) *
        Math.floor(((columnSize * block) / columnLength)) - 2;
if(mouseX >= x && mouseY >= y && mouseX <= (x + block) && mouseY <= (y + block)) {
if(this.emptyGrid[i].value !== 1) {
        this.emptyGrid[i].value = 1;
        this.drawBlueCell(this.emptyGrid[i]);
        this.drawPreview(this.emptyGrid[i]);
        this.strokeMultiplayerCurrentChoice(this.emptyGrid[i]);
        gameData.value  = 1;
}else{
        this.emptyGrid[i].value = 0;
        this.drawWhiteCell(this.emptyGrid[i]);
        this.drawPreview(this.emptyGrid[i]);
        this.strokeMultiplayerCurrentChoice(this.emptyGrid[i]);
        gameData.value  = 0;
}
this.emptyGrid[i].playerChoice = "yours";
gameData.dataType = "fill cell";
gameData.fillCellChoice = "black";
gameData.cell  =  i;
return gameData;
}
}
}else if(this.fillCellChoice == "x") {
for(var i=0;i<this.emptyGrid.length;i++) {
var value = this.emptyGrid[i].value;
var x = this.emptyGrid[i].x, y = this.emptyGrid[i].y;
var width = this.emptyGrid[i].w, height = this.emptyGrid[i].h;
var xPos = ((x - (rowSize * block)) / block) *
        Math.floor(((rowSize * block) / rowLength)) - 2;
var yPos = ((y - (columnSize * block)) / block) *
        Math.floor(((columnSize * block) / columnLength)) - 2;
```

```
        if(mouseX >= x && mouseY >= y && mouseX <= (x + block) && mouseY <= (y + block)) {
        if(this.emptyGrid[i].value !== 2) {
                this.emptyGrid[i].value = 2;
                                this.drawWhiteCell(this.emptyGrid[i]);
                                this.drawXCell(this.emptyGrid[i]);
                                this.drawPreview(this.emptyGrid[i]);
                                this.strokeMultiplayerCurrentChoice(this.emptyGrid[i]);
                                gameData.value  = 2;
        }else{
                this.emptyGrid[i].value = 0;
                                this.drawWhiteCell(this.emptyGrid[i]);
                                this.drawPreview(this.emptyGrid[i]);
                                this.strokeMultiplayerCurrentChoice(this.emptyGrid[i]);
                                gameData.value  = 0;
        }
        this.emptyGrid[i].playerChoice = "yours";
        gameData.dataType = "fill cell";
        gameData.fillCellChoice = "x";
        gameData.cell  =  i;
        return gameData;
        }
        }
        }else if(this.fillCellChoice == "white") {
         for(var i=0;i<this.emptyGrid.length;i++) {
        var value = this.emptyGrid[i].value;
        var x = this.emptyGrid[i].x, y = this.emptyGrid[i].y;
        var width = this.emptyGrid[i].w, height = this.emptyGrid[i].h;
        var xPos = ((x - (rowSize * block)) / block) * Math.floor(((rowSize * block) / rowLength)) - 2;
        var yPos = ((y - (columnSize * block)) / block) * Math.floor(((columnSize * block) /
columnLength)) - 2;
        if(mouseX >= x && mouseY >= y && mouseX <= (x + block) && mouseY <= (y + block)) {
        if(this.emptyGrid[i].value !== 0) {
                this.emptyGrid[i].value = 0;
                this.drawWhiteCell(this.emptyGrid[i]);
                this.drawPreview(this.emptyGrid[i]);
```

```
                this.strokeMultiplayerCurrentChoice(this.emptyGrid[i]);
                gameData.value  = 0;
            }
            this.emptyGrid[i].playerChoice = "yours";
            gameData.dataType = "fill cell";
            gameData.fillCellChoice = "white";
            gameData.cell  =  i;
            return gameData;
            }
            }
    }
};
```

```
Nonogram.prototype.multiplayerStrokeRowNumbersChoice = function(i, gameData) {
            if(this.rowNumbersGrid[i].value == 0) {
             ctx.beginPath();
             ctx.strokeStyle = "red";
             ctx.moveTo(this.rowNumbersGrid[i].x+3,
                    (this.rowNumbersGrid[i].y + this.blockSize)-3);
             ctx.lineTo((this.rowNumbersGrid[i].x + this.blockSize)-3,
                    this.rowNumbersGrid[i].y+3);
             ctx.stroke();
             ctx.closePath();
             ctx.strokeStyle = "black";
             this.rowNumbersGrid[i].value = 1;
             gameData.dataType = "fill cell row numbers grid";
             gameData.cell  =  i;
             gameData.value  = 1;
             return gameData;
            }else{
             ctx.fillStyle = "#e0e0d1";
             ctx.fillRect(this.rowNumbersGrid[i].x+2, this.rowNumbersGrid[i].y+2,
this.rowNumbersGrid[i].w-3, this.rowNumbersGrid[i].h-3);
```

```
            ctx.fillStyle = "black";
            if(this.rowNumbersGrid[i].number < 10) {
                        ctx.font = (this.blockSize)+"px Arial";
                        ctx.fillText(this.rowNumbersGrid[i].number,
                                    (this.rowNumbersGrid[i].x+(Math.floor(this.blockSize/4))),
                                    (this.rowNumbersGrid[i].y+(this.blockSize-Math.floor(this.blockSize/6))));
            }else{
                        ctx.font = (this.blockSize-3)+"px Arial";
                        ctx.fillText(this.rowNumbersGrid[i].number,
                                    (this.rowNumbersGrid[i].x),
                                    (this.rowNumbersGrid[i].y+(this.blockSize-Math.floor(this.blockSize/5))));
            }
            this.rowNumbersGrid[i].value = 0;
            gameData.dataType = "fill cell row numbers grid";
            gameData.cell  =  i;
            gameData.value  = 0;
            return gameData;
            }
};
```

```
Nonogram.prototype.multiplayerStrokeColumnNumbersChoice = function(i, gameData) {
            if(this.columnNumbersGrid[i].value == 0) {
            ctx.beginPath();
            ctx.strokeStyle = "red";
            ctx.moveTo(this.columnNumbersGrid[i].x+3, (this.columnNumbersGrid[i].y +
                        this.blockSize)-3);
            ctx.lineTo((this.columnNumbersGrid[i].x + this.blockSize)-3,
                        this.columnNumbersGrid[i].y+3);
            ctx.stroke();
            ctx.closePath();
            ctx.strokeStyle = "black";
            this.columnNumbersGrid[i].value = 1;
            gameData.dataType = "fill cell column numbers grid";
```

```
            gameData.cell  =  i;
            gameData.value  = 1;
            return gameData;
            }else{
            ctx.fillStyle = "#e0e0d1";
            ctx.fillRect(this.columnNumbersGrid[i].x+2, this.columnNumbersGrid[i].y+2,
this.columnNumbersGrid[i].w-3, this.columnNumbersGrid[i].h-3);
            ctx.fillStyle = "black";
             if(this.columnNumbersGrid[i].number < 10) {
                        ctx.font = (this.blockSize)+"px Arial";
                        ctx.fillText(this.columnNumbersGrid[i].number,
                                    (this.columnNumbersGrid[i].x+(Math.floor(this.blockSize/4))),
                                    (this.columnNumbersGrid[i].y+(this.blockSize-Math.floor(this.blockSize/6))));
             }else{
                        ctx.font = (this.blockSize-3)+"px Arial";
                        ctx.fillText(this.columnNumbersGrid[i].number,
                                    (this.columnNumbersGrid[i].x),
                                    (this.columnNumbersGrid[i].y+(this.blockSize-Math.floor(this.blockSize/5))));
             }
            this.columnNumbersGrid[i].value = 0;
             gameData.dataType = "fill cell column numbers grid";
             gameData.cell  =  i;
             gameData.value  = 0;
             return gameData;
            }
};
```

```
//Ζωγραφίζει το κελί κόκκινο
let drawRedCellValue = 3;
Nonogram.prototype.drawRedCell = function(cell) {
            ctx.fillStyle = 'red';
            ctx.fillRect(cell.x + (drawBlackCellValue/scaleFactor),
             cell.y + (drawBlackCellValue/scaleFactor),
```

```
            cell.w - ((drawBlackCellValue/scaleFactor) * 2),
            cell.h - ((drawBlackCellValue/scaleFactor) * 2));
};

//Ζωγραφίζει το κελί μπλε
let drawBlueCellValue = 3;
Nonogram.prototype.drawBlueCell = function(cell) {
            ctx.fillStyle = 'blue';
            ctx.fillRect(cell.x + (drawBlackCellValue/scaleFactor),
            cell.y + (drawBlackCellValue/scaleFactor),
            cell.w - ((drawBlackCellValue/scaleFactor) * 2),
            cell.h - ((drawBlackCellValue/scaleFactor) * 2));
};
```

```
Nonogram.prototype.strokeMultiplayerCurrentChoice = function(cell) {
            if(this.previousChoice.active) {
            ctx.beginPath();
            for(let i=0; i<this.previousChoice.cell.length; i++) {
            if(this.previousChoice.cell[i].value === 1) {
            this.drawWhiteCell(this.previousChoice.cell[i]);
            this.drawBlueCell(this.previousChoice.cell[i]);
            }else if(this.previousChoice.cell[i].value === 2) {
            this.drawWhiteCell(this.previousChoice.cell[i]);
            this.drawXCell(this.previousChoice.cell[i]);
            }else{
            this.drawWhiteCell(this.previousChoice.cell[i]);
            }
            }
            ctx.stroke();
            ctx.closePath();
            this.previousChoice.cell = []; //Αδειάζουμε των πίνακα
            }
            this.currentChoice.cell = cell;
```

```
        this.previousChoice.cell.push(cell);
        this.previousChoice.active = true;
        ctx.strokeStyle = "red";
        ctx.lineWidth   = 4;
        ctx.strokeRect(cell.x+5, cell.y+5, this.blockSize-10, this.blockSize-10);
};
```

Τοποθετούμε των κώδικα που χρειαζόμαστε στο event mousedown του canva στην else if που ελέγχει αν το state που βρισκόμαστε είναι το multiplayer. Ελέγχουμε αν είναι ο γύρος του χρήστη, εάν είναι τότε ζωγραφίζουμε το κελί που επέλεξε ο χρήστης, αποθηκεύουμε στην μεταβλητή gameData την επιλογή που έκανε και την στέλνουμε στον server με το event empty grid, ορίζουμε την μεταβλητή turn = false, ανανεώνουμε το info current progress κσι τέλος ελέγχουμε με την επιλογή που έκανε ο χρήστης αν έλυσε την πίστα

```
else if(state === "multiplayer") {
    if(startPointMouseX<originX) {
        dragStart.x = startPointMouseX - translatePos.x;
        dragStart.y = startPointMouseY - translatePos.y;
        dragged = true;
    }else if(startPointMouseY<originY) {
        dragStart.x = startPointMouseX - translatePos.x;
        dragStart.y = startPointMouseY - translatePos.y;
        dragged = true;
    }else if(startPointMouseX>originWidth) {
        dragStart.x = startPointMouseX - translatePos.x;
        dragStart.y = startPointMouseY - translatePos.y;
        dragged = true;
    }else if(startPointMouseY>originHeight) {
        dragStart.x = startPointMouseX - translatePos.x;
        dragStart.y = startPointMouseY - translatePos.y;
        dragged = true;
    }else{
        if(turn === true) {
            ctx.save();
            ctx.translate(originX,originY);
            ctx.scale(scaleFactor,scaleFactor);
            var gameData =
nonogram.multiplayerFillCels((startPointMouseX-originX)/scaleFactor,
(startPointMouseY-originY)/scaleFactor);
            ctx.restore();
```

```
            if(gameData) {
                sock.emit('update progress', gameData);
                turn = false;
                $("#info-current-progress").text("");
                $("#info-current-progress").text(nonogram.findProgress() +
"%");
                if(nonogram.checkProgress()) {
                    if(multiplayerStageIndex ==
                    (multiplayerStagesNames.length-1)) {
                        $('#multiplayer-finished-popup').show();
                        sock.emit('multiplayer finished');
                    }else{
                        sock.emit('correct' , multiplayerGame);
                    }
                }else{
                    $("#correct-multiplayer").hide();
                    sock.emit('end-turn');
                }
                    gameData = null;
            }
        }
    }
}
```

11.6 Λειτουργικότητα zoom και drag στο multiplayer

Για την λειτουργικότητα τροποποιούμε και προσθέτουμε κώδικα στα εξής σημεία.

Στο event mouseup του canvas

```
$(canvas).mouseup(function(){
        if(state === "level") {
        isDown = false;
        if(dragged){
        $(topControl).show();
        $(leftControl).show();
        $(rightControl).show();
        $(bottomControl).show();
        dragged = false;
        }
        if(activeDragControl) {
```

```
        activeDragControl = null;
        }
        if(nonogram.checkProgress()) {
        $("#correct-singleplayer").show();
        store("correct-" + currentStage, 1);
        $(".correct-" + currentStage).show();
        }else{
        $("#correct-singleplayer").hide();
        store("correct-" + currentStage, 0);
        $(".correct-" + currentStage).hide();
        }
        nonogram.findUserChoices();
        store(currentStage, nonogram.userChoices.levelGrid);
        store('rowNumbersGrid-'+currentStage, nonogram.userChoices.rowNumbersGrid);
        store('columnNumbersGrid-'+currentStage, nonogram.userChoices.columnNumbersGrid);
        $("#info-current-progress").text("");
        $("#info-current-progress").text(nonogram.findProgress() + "%");
        }else if(state === "multiplayer") {
        isDown = false;
        if(dragged){
        $(topControl).show();
        $(leftControl).show();
        $(rightControl).show();
        $(bottomControl).show();
        dragged = false;

        }
        if(activeDragControl) {
        activeDragControl = null;
        }
        }
});
```

Στην συνάρτηση zoom

```
function zoom(scaleFactor, translatePos) {
    clearCanvas();
    ctx.save();
    ctx.translate(translatePos.x, translatePos.y);
    ctx.scale(scaleFactor,scaleFactor);
    ctx.translate(-translatePos.x, -translatePos.y);
    nonogram.drawGrid();
    nonogram.drawRowNumbers();
    nonogram.drawColumnNumbers();
    if(state === "level") {
        for(let i=0; i<nonogram.emptyGrid.length; i++) {
            if(nonogram.emptyGrid[i].value === 1){
                nonogram.drawBlackCell(nonogram.emptyGrid[i]);
                nonogram.drawPreview(nonogram.emptyGrid[i]);
            }else if(nonogram.emptyGrid[i].value === 2) {
                nonogram.drawXCell(nonogram.emptyGrid[i]);
                nonogram.drawPreview(nonogram.emptyGrid[i]);
            }
        }
            ctx.beginPath();
            ctx.strokeStyle = "red";
            ctx.lineWidth = 3;
            for(let i=0; i<nonogram.rowNumbersGrid.length; i++) {
            if(nonogram.rowNumbersGrid[i].value === 1) {
                ctx.moveTo(nonogram.rowNumbersGrid[i].x+3,

                    (nonogram.rowNumbersGrid[i].y + nonogram.blockSize)-3);
                ctx.lineTo((nonogram.rowNumbersGrid[i].x +

                    nonogram.blockSize)-3, nonogram.rowNumbersGrid[i].y+3);
            }
            }
            for(let i=0; i<nonogram.columnNumbersGrid.length; i++) {
            if(nonogram.columnNumbersGrid[i].value === 1) {
                ctx.moveTo(nonogram.columnNumbersGrid[i].x+3,
```

```
            (nonogram.columnNumbersGrid[i].y + nonogram.blockSize)-3);
            ctx.lineTo((nonogram.columnNumbersGrid[i].x +
            nonogram.blockSize)-3, nonogram.columnNumbersGrid[i].y+3);
        }
        }
}else if(state === "multiplayer") {
    for(let i=0; i<nonogram.emptyGrid.length; i++) {
        if(nonogram.emptyGrid[i].value === 1){
            if(nonogram.emptyGrid[i].playerChoice === "yours") {
                nonogram.drawBlueCell(nonogram.emptyGrid[i]);
                    nonogram.drawPreview(nonogram.emptyGrid[i]);
            }else if(nonogram.emptyGrid[i].playerChoice === "team mate") {
                    nonogram.drawRedCell(nonogram.emptyGrid[i]);
                    nonogram.drawPreview(nonogram.emptyGrid[i]);
            }
        }else if(nonogram.emptyGrid[i].value === 2) {
            nonogram.drawXCell(nonogram.emptyGrid[i]);
            nonogram.drawPreview(nonogram.emptyGrid[i]);
        }
    }
        ctx.beginPath();
        ctx.strokeStyle = "red";
        ctx.lineWidth = 3;
        for(let i=0; i<nonogram.rowNumbersGrid.length; i++) {
        if(nonogram.rowNumbersGrid[i].value === 1) {
            ctx.moveTo(nonogram.rowNumbersGrid[i].x+3,
            (nonogram.rowNumbersGrid[i].y + nonogram.blockSize)-3);
            ctx.lineTo((nonogram.rowNumbersGrid[i].x +
            nonogram.blockSize)-3, nonogram.rowNumbersGrid[i].y+3);
        }
        }
        for(let i=0; i<nonogram.columnNumbersGrid.length; i++) {
        if(nonogram.columnNumbersGrid[i].value === 1) {
```

```
            ctx.moveTo(nonogram.columnNumbersGrid[i].x+3,
             (nonogram.columnNumbersGrid[i].y + nonogram.blockSize)-3);
            ctx.lineTo((nonogram.columnNumbersGrid[i].x +
             nonogram.blockSize)-3, nonogram.columnNumbersGrid[i].y+3);
          }
           }
    }
    ctx.closePath();
    ctx.stroke();
    ctx.restore();
    if(scaleFactor !== 1) {
            $(topControl).show();
            $(leftControl).show();
            $(rightControl).show();
            $(bottomControl).show();
    }else{
            $(topControl).hide();
            $(leftControl).hide();
            $(rightControl).hide();
            $(bottomControl).hide();
    }
};
```

Στην συνάρτηση drag

```
function drag(translatePos) {
    clearCanvas();
    ctx.save();
    ctx.translate(translatePos.x,translatePos.y);
    ctx.scale(scaleFactor,scaleFactor);
    nonogram.drawGrid();
    nonogram.drawRowNumbers();
    nonogram.drawColumnNumbers();
    if(state === "level") {
```

```
    nonogram.redrawProgress();
}else if(state === "multiplayer") {
          for(let i=0; i<nonogram.emptyGrid.length; i++) {
         if(nonogram.emptyGrid[i].value === 1){
           if(nonogram.emptyGrid[i].playerChoice === "yours") {
                    nonogram.drawBlueCell(nonogram.emptyGrid[i]);
                    nonogram.drawPreview(nonogram.emptyGrid[i]);
           }else if(nonogram.emptyGrid[i].playerChoice === "team mate") {
                    nonogram.drawRedCell(nonogram.emptyGrid[i]);
                    nonogram.drawPreview(nonogram.emptyGrid[i]);
           }
         }else if(nonogram.emptyGrid[i].value === 2) {
           nonogram.drawXCell(nonogram.emptyGrid[i]);
           nonogram.drawPreview(nonogram.emptyGrid[i]);
         }
          }
          ctx.beginPath();
          ctx.strokeStyle = "red";
          ctx.lineWidth = 3;
          for(let i=0; i<nonogram.rowNumbersGrid.length; i++) {
         if(nonogram.rowNumbersGrid[i].value === 1) {
           ctx.moveTo(nonogram.rowNumbersGrid[i].x+3,
             (nonogram.rowNumbersGrid[i].y + nonogram.blockSize)-3);
           ctx.lineTo((nonogram.rowNumbersGrid[i].x +
             nonogram.blockSize)-3, nonogram.rowNumbersGrid[i].y+3);
         }
          }
          for(let i=0; i<nonogram.columnNumbersGrid.length; i++) {
        if(nonogram.columnNumbersGrid[i].value === 1) {
          ctx.moveTo(nonogram.columnNumbersGrid[i].x+3,
           (nonogram.columnNumbersGrid[i].y + nonogram.blockSize)-3);
          ctx.lineTo((nonogram.columnNumbersGrid[i].x +
           nonogram.blockSize)-3, nonogram.columnNumbersGrid[i].y+3);
```

```
            }
            }
    }
    ctx.closePath();
    ctx.stroke();

    ctx.restore();
};
```

12. Responsive design του παιχνιδιού

Με τον όρο responsive εννοούμε ότι θέλουμε η σελίδα και το παιχνίδι μας να προσαρμόζεται δυναμίκα ανάλογα με το μέγεθος του παραθύρου του browser. Για να κάνουμε την σελίδα μας αλλά και το παιχνίδι μας responsive δημιουργούμε το αρχείο responsive.css στο οποίο θα έχουμε την css που χρειαζόμαστε για την μορφοποίηση που θέλουμε να δώσουμε.

```
@media only screen and (max-height: 462px) {
    #container-tools{
            text-align: right;
    }

    .expand {
            left: 100px;
    }

    #waiting-screen h1 {
            margin-top: 54%;
    }

    #player-left-popup {
            text-align:  center;
```

```
    }

    #multiplayer-next-stage{
        min-width: 176px;
    }
}

@media only screen and (max-height: 402px) {
    #levels {
        font-size: 20px;
    }

    #levels5x5, #levels10x10, #levels15x15 {
        font-size: 20px;
    }
}

@media only screen and (max-height: 381px) {
    #player-left-popup {
        font-size: 15px;
    }
}

@media only screen and (max-height: 359px) {
    #container-tools {
        position: absolute;
        top: 50%;
        left: 100%;
        width: 40px;
        min-width: 0px;
        transform: translateY(-50%);
    }

    #tools {
        float: none;
```

```
        position: relative;
        left: 4px;
    }

    #info-progress {
        font-size: 9px;
        width: 36px;
        height: auto;
    }

    .expand {
        left: -26px;
        top: 102px;
    }

    #waiting-screen h1 {
        margin-top: 47%;
    }

    #player-left-popup {
        font-size: 15px;
        width: 135px;
    }
}

@media only screen and (max-height: 300px) {
    #menu {
        left:  75%;
        bottom: 5%;
        font-size: 25px;
    }

    #clients-count {
        font-size: 20px;
    }
```

```
    #instructions {
        top: 35%;
        transform: translate(-32%, -40%);
        font-size: 16px;
    }
}

@media only screen and (max-width: 372px) and (max-height: 463px){
    div#container-tools{
        text-align: left;
    }

    .expand {
        left: 100px;
    }
}

@media only screen and (max-width: 372px) {
    div#container-tools{
        text-align: left;
    }

    .expand {
        left: 100px;
    }
}

@media only screen and (max-width: 1000px) {
    #instructions {
        min-width: 500px;
    }
}

@media only screen and (max-width: 730px) {
```

```
    #instructions {
            min-width: 400px;
    }
}

@media only screen and (max-width: 550px) {
    #instructions {
            min-width: 300px;
            font-size: 17px;
    }
}
```

Προσθέτουμε την css στο αρχείο index.html

```
<link rel="stylesheet" type="text/css" href="css/responsive.css">
```

Στο αρχείο home.js προσθέτουμε την μέθοδο του window την resize η οποία εκτελείται όποτε αλλάζει το μέγεθος του παραθύρου. Σε αυτήν την μέθοδο επαναποθετούμε τα στοιχεία της πίστας βασιζόμενοι στην αλλαγή του μεγέθους του παραθύρου μας.

```
//Window resize
$(window).resize( () => {
            canvas.width = window.innerWidth;
            canvas.height = window.innerHeight;

            if(state === 'menu') {
            ctx.drawImage(img, (innerWidth/2)-(img.width/2), (innerHeight/2)-(img.height/2));
            introScreenLogo.draw();
            for(let i=0; i<30; i++) {
            blackRectArray[i].relocate(Math.random() * (innerWidth - blackRectArray[i].w * 2) +
blackRectArray[i].w, Math.random() * (innerHeight - blackRectArray[i].w * 2) + blackRectArray[i].w);
            whiteRectArray[i].relocate(Math.random() * (innerWidth - whiteRectArray[i].w * 2) +
whiteRectArray[i].w, Math.random() * (innerHeight - whiteRectArray[i].w * 2) + whiteRectArray[i].w);
            xRectArray[i].relocate(Math.random() * (innerWidth - xRectArray[i].w * 2) + xRectArray[i].w,
Math.random() * (innerHeight - xRectArray[i].w * 2) + xRectArray[i].w);
            blackRectArray[i].update();
```

```
            whiteRectArray[i].update();
            xRectArray[i].update();
            }
            }else if(state === 'level') {
            if(window.innerHeight > 0 && window.innerWidth > 0) {
            nonogram.relocate();
            nonogram.findUserChoices();
            ctx.save();
            ctx.translate(originX,originY);
            ctx.scale(scaleFactor,scaleFactor);
            nonogram.retrieveProgress(retrieve(currentStage),
retrieve('rowNumbersGrid-'+currentStage),retrieve('columnNumbersGrid-'+currentStage));
            nonogram.redrawProgress();
            ctx.restore();
            limitBottom = nonogram.height-myLimit;
            limitRight = nonogram.width-myLimit;
            }
            }else if(state === "multiplayer") {
            if(window.innerHeight > 0 && window.innerWidth > 0) {
            nonogram.relocate();
            nonogram.findUserChoices();
            ctx.save();
            ctx.translate(originX,originY);
            ctx.scale(scaleFactor,scaleFactor);
            nonogram.redrawProgress();
            nonogram.strokeTeamMateChoice();
            ctx.restore();
            limitBottom = nonogram.height-myLimit;
            limitRight = nonogram.width-myLimit;
            }
            }
            //Change the size of the image
            if(window.innerHeight >= 753) {
            if(window.innerWidth >= 999) {
            img.src = "img/nono_1000X753.png";
```

```
        }else if(window.innerWidth < 999 && window.innerWidth >= 719) {
        img.src = "img/nono_720X542.png";
        }else if(window.innerWidth < 719 && window.innerWidth >= 480) {
        img.src = "img/nono_480X361.png";
        }else if(window.innerWidth < 480 && window.innerWidth >= 360) {
        img.src = "img/nono_360X271.png";
        }else if(window.innerWidth < 360 && window.innerWidth >= 304) {
        img.src = "img/nono_304X229.png";
        }
        }else if(window.innerHeight < 753 && window.innerHeight >= 543) {
        if(window.innerWidth >= 719) {
        img.src = "img/nono_720X542.png";
        }else if(window.innerWidth < 719 && window.innerWidth >= 480) {
        img.src = "img/nono_480X361.png";
        }else if(window.innerWidth < 480 && window.innerWidth >= 360) {
        img.src = "img/nono_360X271.png";
        }else if(window.innerWidth < 360 && window.innerWidth >= 304) {
        img.src = "img/nono_304X229.png";
        }
        }else if(window.innerHeight < 543 && window.innerHeight >= 360) {
        if(window.innerWidth >= 480) {
        img.src = "img/nono_480X361.png";
        }else if(window.innerWidth < 480 && window.innerWidth >= 360) {
        img.src = "img/nono_360X271.png";
        }else if(window.innerWidth < 360 && window.innerWidth >= 304) {
        img.src = "img/nono_304X229.png";
        }
        }else if(window.innerHeight < 360) {
        if(window.innerWidth >= 360) {
        img.src = "img/nono_360X271.png";
        }else if(window.innerWidth < 360 && window.innerWidth >= 304) {
        img.src = "img/nono_304X229.png";
        }
        }
});
```

Τοποθετούμε στο object nonogram την μέθοδο relocate η οποία χρησιμοποιείται για την αναδιάταξη της πίστας μας και των κελιών της πίστας μας βασιζόμενοι στην αλλαγή του μεγέθους της σελίδας μας.

```
this.relocate = function() {
    windowWidth = window.innerWidth;
    windowHeight = window.innerHeight;
    if(windowWidth > windowHeight) {
        if(windowHeight > 359) {
                size = windowHeight - 50;
            }else{
                    size = windowHeight;
            }
    }else{
        size = windowWidth;
    }
    this.blockSize = Math.floor((size / maxSize) - 1);
    this.width = (this.levelGrid[0].length + this.maxRowNumberSize) *

      this.blockSize;
    this.height = (this.levelGrid.length + this.maxColumnNumberSize) *

     this.blockSize;
    canvas.width = this.width;
    canvas.height = this.height;
    ctx.save();
    ctx.translate(originX,originY);
    ctx.scale(scaleFactor,scaleFactor);
    this.drawGrid();
    ctx.restore();
    //---recalibrate the coordinates of every cell
    var indexCells = 0;
    for (var i = (this.maxColumnNumberSize ) * this.blockSize; i <

    this.height; i += this.blockSize ) {
        for ( var y = (this.maxRowNumberSize ) * this.blockSize; y <

        this.width; y += this.blockSize ) {
```

```
            this.emptyGrid[indexCells].w = this.blockSize;
                this.emptyGrid[indexCells].h = this.blockSize;
                this.emptyGrid[indexCells].x = y;
                this.emptyGrid[indexCells].y = i;
                indexCells++;
    }
}
//Cell numbers of every row
var indexRow = 0;
for (var i = 0; i < this.rowNumbers.length; i ++) {
    for ( var y = 0; y < this.rowNumbers[i].length; y ++) {
            this.rowNumbersGrid[indexRow].w = this.blockSize;
                this.rowNumbersGrid[indexRow].h = this.blockSize;
                this.rowNumbersGrid[indexRow].x = (y * this.blockSize);
                this.rowNumbersGrid[indexRow].y = ( (this.maxColumnNumberSize) *
        this.blockSize) + (i * this.blockSize);
                indexRow++;
    }
}
//Cell numbers of every column
var indexColumn = 0;
for (var i = 0; i < this.columnNumbers.length; i ++) {
    for ( var y = 0; y < this.columnNumbers[i].length; y ++) {
            this.columnNumbersGrid[indexColumn].w = this.blockSize;
                this.columnNumbersGrid[indexColumn].h = this.blockSize;
                this.columnNumbersGrid[indexColumn].x = ((this.maxRowNumberSize) *
        this.blockSize) + (i * this.blockSize);
                this.columnNumbersGrid[indexColumn].y = (y * this.blockSize);
                indexColumn++;
    }
}
ctx.save();
ctx.translate(originX,originY);
ctx.scale(scaleFactor,scaleFactor);
```

```
    this.drawRowNumbers();
    this.drawColumnNumbers();
    ctx.restore();
};
```

Το responsive design μας είναι έτοιμο. Στο επόμενο και τελευταίο κεφάλαιο θα μιλήσουμε για τα touch controls.

13. Λειτουργικότητα touch controls του παιχνιδιού

Αναφέραμε στο κεφάλαιο 4.2 την απλή αλληλεπίδραση του χρήστη με touch, σε αυτό το κεφάλαιο θα μιλήσουμε πιο αναλυτικά για το πως λειτουργεί το touch. Στο αρχείο **touch.js** θα προσθέσουμε τα 3 βασικά events που θα χρειαστούμε τα οποία είναι τα **touchstart**, **touchend** και **touchmove**. Ξεκινάμε δημιουργώντας ορισμένες μεταβλητές οι οποίες θα μας βοηθήσουν στην αλληλεπίδραση με την οθόνη αφής και θα φτιάξουμε 3 συναρτήσεις τις οποίες θα τις χρειαστούμε για την αλληλεπίδραση των λειτουργιών zoom, drag και touch που γίνονται στην οθόνη αφής.

Οι global μεταβλητές που θα χρειαστούμε είναι οι: evCache η οποία είναι πίνακας και θα αποθηκεύει τα ενεργά touch events, η prevDif η οποία θα χρησιμοποιείται για την υλοποίηση του zoom για το touch, το zoom θα υλοποιείται με την χρήση δυο δακτύλων και η prevDif θα αντιπρωσωπεύει την προηγούμενη απόσταση των δυο touch, η touches η οποία θα μετράει το πλήθος των touch στην οθόνη για να ξέρουμε πότε γίνεται zoom, την touchZoom η οποία θα ελέγχει αν γίνεται zoom και την gameData η οποία θα αντιπροσωπεύει το κελί που έχει επιλέξει ο χρήστης στο multiplayer.

```
$(canvas).on('touchstart', function(event) {
    event.preventDefault();
    startPointTouchX = Math.floor(event.touches[0].clientX -
      ((window.innerWidth - canvas.width)/2));
    startPointTouchY = Math.floor(event.touches[0].clientY -
      ((window.innerHeight - canvas.height)/2));
    if(state === 'level') {
        evCache.push(event.touches[touches]);
        touches++;
        if(evCache.length == 2) touchZoom = true;
        if(startPointTouchX<originX) {
```

```
            dragStart.x = startPointTouchX - translatePos.x;
            dragStart.y = startPointTouchY - translatePos.y;
            dragged = true;
        }else if(startPointTouchY<originY) {
            dragStart.x = startPointTouchX - translatePos.x;
            dragStart.y = startPointTouchY - translatePos.y;
            dragged = true;
        }else if(startPointTouchX>originWidth) {
            dragStart.x = startPointTouchX - translatePos.x;
            dragStart.y = startPointTouchY - translatePos.y;
            dragged = true;
        }else if(startPointTouchY>originHeight) {
            dragStart.x = startPointTouchX - translatePos.x;
            dragStart.y = startPointTouchY - translatePos.y;
            dragged = true;
        }else{
            if(evCache.length == 1) {
                isDown = true;
                ctx.save();
                ctx.translate(originX,originY);
                ctx.scale(scaleFactor,scaleFactor);
                nonogram.fillCels((startPointTouchX-originX)/scaleFactor,
                  (startPointTouchY-originY)/scaleFactor);
                ctx.restore();
                $("#info-current-progress").text("");
                $("#info-current-progress").text(nonogram.findProgress() +"%");
                nonogram.findUserChoices();
                store(currentStage, nonogram.userChoices.levelGrid);
                store('rowNumbersGrid-'+currentStage,
                  nonogram.userChoices.rowNumbersGrid);
                store('columnNumbersGrid-'+currentStage,
                nonogram.userChoices.columnNumbersGrid);
            }else if(evCache.length == 2) {
                  $(".undo").click();
            }
        }
    }else if(state === 'multiplayer') {
        evCache.push(event.touches[touches]);
        touches++;
        if(evCache.length == 2) {
            touchZoom = true;
        }
        if(startPointTouchX<originX) {
            dragStart.x = startPointTouchX - translatePos.x;
            dragStart.y = startPointTouchY - translatePos.y;
```

```
            dragged = true;
        }else if(startPointTouchY<originY) {
            dragStart.x = startPointTouchX - translatePos.x;
            dragStart.y = startPointTouchY - translatePos.y;
            dragged = true;
        }else if(startPointTouchX>originWidth) {
            dragStart.x = startPointTouchX - translatePos.x;
            dragStart.y = startPointTouchY - translatePos.y;
            dragged = true;
        }else if(startPointTouchY>originHeight) {
            dragStart.x = startPointTouchX - translatePos.x;
            dragStart.y = startPointTouchY - translatePos.y;
            dragged = true;
        }
    }
});

$(canvas).on('touchend', function(event) {
    if(state === "level") {
        touchup_handler(event);
        touches--;
        isDown = false;
        if(evCache.length == 0 || evCache.length == 1) touchZoom = false;
        if(dragged){
            $(topControl).show();
            $(leftControl).show();
            $(rightControl).show();
            $(bottomControl).show();
            dragged = false;
        }
        if(nonogram.checkProgress()) {
            $("#correct-singleplayer").show();
            store("correct-" + currentStage, 1);
            $(".correct-" + currentStage).show();
        }else{
            $("#correct-singleplayer").hide();
            store("correct-" + currentStage, 0);
            $(".correct-" + currentStage).hide();
        }
        nonogram.findUserChoices();
        store(currentStage, nonogram.userChoices.levelGrid);
        store('rowNumbersGrid-'+currentStage,
          nonogram.userChoices.rowNumbersGrid);
        store('columnNumbersGrid-'+currentStage,
        nonogram.userChoices.columnNumbersGrid);
```

```
        $("#info-current-progress").text("");
        $("#info-current-progress").text(nonogram.findProgress() + "%");
    }else if(state === "multiplayer") {
        touchup_handler(event);
        touches--;
        if(dragged){
            $(topControl).show();
            $(leftControl).show();
            $(rightControl).show();
            $(bottomControl).show();
            dragged = false;
        }

        if(touchZoom) {
            if(evCache.length == 1) {
                return;
            }else if(evCache.length == 0) {
                touchZoom = false;
                return;
            }
        }else {
            if(turn === true) {
                ctx.save();
                ctx.translate(originX,originY);
                ctx.scale(scaleFactor,scaleFactor);
                gameData =
nonogram.multiplayerFillCels((startPointTouchX-originX)/scaleFactor,
(startPointTouchY-originY)/scaleFactor);
                ctx.restore();
                if(gameData) {
                    turn = false;
                    $("#info-current-progress").text("");
                    $("#info-current-progress").text(nonogram.findProgress() +
                      "%");
                    sock.emit('update progress', gameData);
                    if(nonogram.checkProgress()) {
                        if(multiplayerStageIndex ==
                        (multiplayerStagesNames.length-1)) {
                            $('#multiplayer-finished-popup').show();
                            sock.emit('multiplayer finished');
                        }else{
                            sock.emit('correct' , multiplayerGame);
                        }
                    }else{
                        $("#correct-multiplayer").hide();
```

```
                        sock.emit('end-turn');
                    }
                    gameData = null;
                }
            }
        }
    }
});

$(canvas).on('touchmove', function(event) {
    event.preventDefault();
    var touchX = Math.floor(event.touches[0].clientX - ((window.innerWidth -
      canvas.width) / 2));
    var touchY = Math.floor(event.touches[0].clientY - ((window.innerHeight -
      canvas.height) / 2));
    if(state ==="level") {
        if(evCache.length == 2) touch_zoom_handler(event);
        if(activeDragControl) {
            $(topControl).hide();
            $(leftControl).hide();
            $(rightControl).hide();
            $(bottomControl).hide();
            dragControl(touchX-dragStart.x, touchY-dragStart.y);
        }
        if(dragged) {
            dragControl(touchX-dragStart.x, touchY-dragStart.y);
            $(topControl).hide();
            $(leftControl).hide();
            $(rightControl).hide();
            $(bottomControl).hide();
        }
        if(isDown && evCache.length == 1){
            ctx.save();
            ctx.translate(originX,originY);
            ctx.scale(scaleFactor,scaleFactor);
            nonogram.fillMultiCells((touchX-originX)/scaleFactor,
              (touchY-originY)/scaleFactor,
              (startPointTouchX-originX)/scaleFactor,
              (startPointTouchY-originY)/scaleFactor);
            ctx.restore();
        }
    }else if(state === "multiplayer") {
        if(evCache.length == 2) {
            touch_zoom_handler(event);
        }
```

```
        if(activeDragControl) {
            $(topControl).hide();
            $(leftControl).hide();
            $(rightControl).hide();
            $(bottomControl).hide();
            dragControl(touchX-dragStart.x, touchY-dragStart.y);
        }
        if(dragged) {
            dragControl(touchX-dragStart.x, touchY-dragStart.y);
            $(topControl).hide();
            $(leftControl).hide();
            $(rightControl).hide();
            $(bottomControl).hide();
        }
    }
});
```

Προσθέτουμε τις συναρτήσεις touch_zoom_handler, touchup_handler και remove_event.

```
function touch_zoom_handler(event) {
        // Find this event in the cache and update its record with this event
        for(var i=0; i<evCache.length; i++) {
        if(event.changedTouches[0].identifier == evCache[i].identifier) {
        evCache[i] = event.changedTouches[0];
        break;
        }
        }
        // If two pointers are down, check for pinch gestures
        if (evCache.length == 2) {
        // Calculate the distance between the two pointers
        var curDiffX = Math.abs(evCache[0].clientX - evCache[1].clientX);
        var curDiffY = Math.abs(evCache[0].clientY - evCache[1].clientY);
        if (prevDiff > 0) {
        if (curDiffX > prevDiff) {
        // The distance between the two pointers has increased
        if(scaleFactor < 2.5) {
                scaleFactor += 0.1;
                translatePos.x = ((Math.abs(evCache[0].clientX + evCache[1].clientX)/2) -
originX)/scaleFactor;
```

```
                translatePos.y = ((Math.abs(evCache[0].clientY + evCache[1].clientY)/2) -
originY)/scaleFactor;
                zoom(scaleFactor, translatePos);
                translatePos.x = -((scaleFactor*translatePos.x)-translatePos.x);
                translatePos.y = -((scaleFactor*translatePos.y)-translatePos.y);
                originX = translatePos.x;
                originY = translatePos.y;
                trackTransforms(translatePos.x, translatePos.y, translatePos.x+(scaleFactor*canvas.width),
translatePos.y+(scaleFactor*canvas.height));
        }
        }
        if (curDiffX < prevDiff) {
        // The distance between the two pointers has decreased
        if(scaleFactor > 1) {
                scaleFactor -= 0.1;
                translatePos.x = ((Math.abs(evCache[0].clientX + evCache[1].clientX)/2) -
originX)/scaleFactor;
                translatePos.y = ((Math.abs(evCache[0].clientY + evCache[1].clientY)/2) -
originY)/scaleFactor;
                zoom(scaleFactor, translatePos);
                translatePos.x = -((scaleFactor*translatePos.x)-translatePos.x);
                translatePos.y = -((scaleFactor*translatePos.y)-translatePos.y);
                originX = translatePos.x;
                originY = translatePos.y;
                trackTransforms(translatePos.x, translatePos.y, translatePos.x+(scaleFactor*canvas.width),
translatePos.y+(scaleFactor*canvas.height));
        }
        }
        }
        // Cache the distance for the next move event
        prevDiff = curDiffX;
        }
};

function touchup_handler(event) {
```

```
        // Remove this pointer from the cache
        // If the number of pointers down is less than two then reset diff tracker
        if (evCache.length < 2) {
        prevDiff = -1;
        }

        for(var i=0;i<event.changedTouches.length; i++) {
        remove_event(event.changedTouches[i]);
        }
};

// Cache management
function remove_event(event) {
        // Remove this event from the target's cache
        for (var i = 0; i<evCache.length; i++) {
        if(evCache[i].identifier == event.identifier) {
        evCache.splice(i, 1);
        break;
        }
        }
};
```

Προσθέτουμε στο αρχείο **zoom_drag.js** των κώδικα που χρειαζόμαστε για να λειτουργούν τα drag controls σε touch χειρισμό.

```
// --- Drag Controls for touch
let touchDragStartX = 0;
let touchDragStartY = 0;

leftControl.addEventListener("touchmove", function(event) {
        if(activeDragControl) {
        touchX = Math.floor(event.touches[0].clientX - ((window.innerWidth - canvas.width) / 2)) -
Math.floor(touchDragStartX);
        touchY = Math.floor(event.touches[0].clientY) - touchDragStartY;
        if(isNaN(translatePos.x)) {
```

```
            translatePos.x = 0;
            translatePos.y = 0;
        }
        $(topControl).hide();
        $(leftControl).hide();
        $(rightControl).hide();
        $(bottomControl).hide();
        dragControl(touchX, touchY);
        }
});

leftControl.addEventListener('touchstart', function(event) {
        event.preventDefault();
        touchDragStartX = Math.floor(event.touches[0].clientX - ((window.innerWidth - canvas.width) /
2)) - translatePos.x;
        touchDragStartY = Math.floor(event.touches[0].clientY) - translatePos.y;
        $(this).hide();
        activeDragControl = "left";
});

leftControl.addEventListener("touchend", function(event) {
        if(activeDragControl) {
        $(topControl).show();
        $(leftControl).show();
        $(rightControl).show();
        $(bottomControl).show();
        activeDragControl = null;
        }
});

topControl.addEventListener("touchmove", function(event) {
        if(activeDragControl) {
        touchX = Math.floor(event.touches[0].clientX - ((window.innerWidth - canvas.width) / 2)) -
Math.floor(touchDragStartX);
        touchY = Math.floor(event.touches[0].clientY) - touchDragStartY;
```

```
        if(isNaN(translatePos.x)) {
        translatePos.x = 0;
        translatePos.y = 0;
        }
        $(topControl).hide();
        $(leftControl).hide();
        $(rightControl).hide();
        $(bottomControl).hide();
        dragControl(touchX, touchY);
        }
});

topControl.addEventListener('touchstart', function(event) {
        event.preventDefault();
        touchDragStartX = Math.floor(event.touches[0].clientX - ((window.innerWidth - canvas.width) /
2)) - translatePos.x;
        touchDragStartY = Math.floor(event.touches[0].clientY) - translatePos.y;
        $(this).hide();
        activeDragControl = "top";
});

topControl.addEventListener("touchend", function(event) {
        if(activeDragControl) {
        $(topControl).show();
        $(leftControl).show();
        $(rightControl).show();
        $(bottomControl).show();
        activeDragControl = null;
        }
});

rightControl.addEventListener("touchmove", function(event) {
        if(activeDragControl) {
        touchX = Math.floor(event.touches[0].clientX - ((window.innerWidth - canvas.width) / 2)) -
Math.floor(touchDragStartX);
```

```
        touchY = Math.floor(event.touches[0].clientY) - touchDragStartY;
        if(isNaN(translatePos.x)) {
        translatePos.x = 0;
        translatePos.y = 0;
        }
        $(topControl).hide();
        $(leftControl).hide();
        $(rightControl).hide();
        $(bottomControl).hide();
        dragControl(touchX, touchY);
        }
});

rightControl.addEventListener('touchstart', function(event) {
        event.preventDefault();
        touchDragStartX = Math.floor(event.touches[0].clientX - ((window.innerWidth - canvas.width) /
2)) - translatePos.x;
        touchDragStartY = Math.floor(event.touches[0].clientY) - translatePos.y;
        $(this).hide();
        activeDragControl = "right";
});

rightControl.addEventListener("touchend", function(event) {
        if(activeDragControl) {
        $(topControl).show();
        $(leftControl).show();
        $(rightControl).show();
        $(bottomControl).show();
        activeDragControl = null;
        }
});

bottomControl.addEventListener("touchmove", function(event) {
        if(activeDragControl) {
        touchX = Math.floor(event.touches[0].clientX - ((window.innerWidth - canvas.width) / 2)) -
```

```
Math.floor(touchDragStartX);
            touchY = Math.floor(event.touches[0].clientY) - touchDragStartY;
            if(isNaN(translatePos.x)) {
            translatePos.x = 0;
            translatePos.y = 0;
            }
            $(topControl).hide();
            $(leftControl).hide();
            $(rightControl).hide();
            $(bottomControl).hide();
            dragControl(touchX, touchY);
            }
});

bottomControl.addEventListener('touchstart', function(event) {
            event.preventDefault();
            touchDragStartX = Math.floor(event.touches[0].clientX - ((window.innerWidth - canvas.width) /
2)) - translatePos.x;
            touchDragStartY = Math.floor(event.touches[0].clientY) - translatePos.y;
            $(this).hide();
            activeDragControl = "bottom";
});

bottomControl.addEventListener("touchend", function(event) {
    if(activeDragControl) {
        $(topControl).show();
              $(leftControl).show();
              $(rightControl).show();
              $(bottomControl).show();
              activeDragControl = null;
            }
});
```

Τελειώσαμε την υλοποίηση των touch controls. Στο επόμενο κεφάλαιο θα μιλήσουμε για τα συμπεράσματα της πτυχιακής εργασίας.

14. Συμπεράσματα της πτυχιακής εργασίας

Από την μελέτη που έκανα για την πτυχιακή εργασία έμαθα να χρησιμοποιώ τις τεχνολογίες node.js, socket.io, express.js και ορισμένες λειτουργίες του browser όπως το storage. Θα χρησιμοποιήσω αρκετά πράγματα από αυτά που έμαθα σε μελλοντικά μου project, ένα από αυτά είναι η ασύγχρονη επικοινωνία του server με τον client χρησιμοποιώντας το socket.io.

Πράγματα τα οποία με δυσκόλεψαν ήταν η υλοποίηση των έξτρα λειτουργιών zoom και drag. Ήταν δύσκολη η κατανόηση των scale και translate μεθόδων, για να μπορέψω να μετατρέψω τις συντεταγμένες του ποντικιού στον canva υπολογίζοντας τις συντεταγμένες του "κόσμου" της πίστας με τις αλλαγές που έχουν γίνει με το scale και το translate. Επίσης με δυσκόλεψε και η υλοποίηση του multiplayer, συγκεκριμένα η δημιουργία του room που θα παίζανε και θα επικοινωνούσαν οι δυο παίκτες ανεξάρτητα από τον server.

Σε γενικές γραμμές θεωρώ ότι η συγκεκριμένη εργασία με βοήθησε στο να γίνω καλύτερος προγραμματιστής απ ότι ήμουν, έμαθα να γράφω πιο καθαρό κώδικα και να διαχωρίζω των κώδικα μου σε διαφορετικά αρχεία το οποίο με βοήθησε στο να μην γράφω επαναλαμβανόμενο κώδικα.

Των κώδικα της εργασίας θα των βρείτε στην σελίδα μου στο github

https://github.com/DinosMpo/Nonogram-Complete-Steps

15. Βιβλιογραφία

Ορισμένα links τα οποία με βοήθησαν στο να μάθω και να κατανοήσω ή να εμπνευστώ ορισμένα πράγματα, είτε από tutorials είτε από videos στο youtube είναι τα εξής:

https://www.youtube.com/watch?v=EO6OkltgudE

https://www.youtube.com/watch?v=83L6B13ixQ0

https://www.youtube.com/watch?v=yq2au9EfeRQ

https://www.youtube.com/watch?v=vxljFhP2krI

https://developer.mozilla.org/en-US/docs/Web/API/Pointer_events/Pinch_zoom_gestures

https://www.youtube.com/watch?v=xVcVbCLmKew