Πτυχιακή Εργασία

Ανάλυση και εφαρμογή κανόνων και θεωρίας για τη δημιουργία ενός διαδραστικού παιχνιδιού.

Ονοματεπώνυμο φοιτητή: Παναγιώτης Ζουλούμης 1383 Α.Μ

Χρήστος Μπερερής 1048 Α.Μ

Επιβλέπων καθηγητής: Σωτήρης Χριστδούλου

Αντίρριο 2017

Πίνακας περιεχομένων

Περίληψη

Σκοπός της παρούσας εργασίας είναι η ανάλυση και εφαρμογή κανόνων και θεωρίας για τη δημιουργία ενός διαδραστικού παιχνιδιού.

Το παιχνίδι το οποίο αφορά η εργασία αυτή, είναι ο γνωστός σε όλους μας Ναρκαλιευτής.

Η υλοποίηση θα γίνει με χρήση HTML5, CSS και JavaScript.

Όσον αφορά τους κανόνες του παιχνιδιού, θα περιλαμβάνεται ένα ταμπλό με κουμπάκια, τα οποία θα κρύβουν κενό τετράγωνο, αριθμό ή βόμβα.

Όταν ο χρήστης πατήσει σε κουμπί που κρύβει κενό τετράγωνο, αυτό θα ανοίγει και θα προκαλεί και τα γειτονικά κενά τετράγωνα να ανοίξουν. Αυτό θα επαναλαμβάνεται μέχρι να βρεθεί τετράγωνο με βόμβα ή τετράγωνο με αριθμό. Το τετράγωνο με αριθμό θα δείχνει (μέσω του αριθμού του) με πόσα τετράγωνα με βόμβα γειτονεύει.

Εάν ο χρήστης πατήσει πάνω σε κουμπί που κρύβει τετράγωνο με βόμβα το παιχνίδι τερματίζεται και ο χρήστης “χάνει” το παιχνίδι.

Αν ο χρήστης μαρκάρει σωστά, μαντεύοντας και υπολογίζοντας τους αριθμούς, όλα τα τετράγωνα με βόμβα, το παιχνίδι τερματίζεται και ο χρήστης “νικάει” το παιχνίδι.

Όσον αφορά τη λειτουργικότητα του παιχνιδιού, θα υπάρχει η δυνατότητα, πριν την έναρξη του παιχνιδιού, ο χρήστης να μπορεί να επιλέξει ανάμεσα σε επίπεδα δυσκολίας, που θα περιλαμβάνουν προεπιλεγμένων διαστάσεων ταμπλό και αριθμό βομβών ή οριζόμενα από τον χρήστη, τόσο της διάστασης του ταμπλό όσο και του αριθμού των βομβών.

Επίσης θα δίνεται η δυνατότητα στο χρήστη να “μαρκάρει” τετράγωνα του ταμπλό με το σημάδι της βόμβας ή το ερωτηματικό. Τα σημάδια - μαρκαρίσματα αυτά, θα απαγορεύουν ή θα ζητούν την επιβεβαίωση του χρήστη στη περίπτωση που πατήσει επάνω τους, ώστε να τον “προλάβουν” από λανθασμένο πάτημα σε αυτά.

Η οθόνη του παιχνιδιού θα περιέχει χρονόμετρο που θα δείχνει το χρόνο του παιχνιδιού, ενώ θα περιέχει και πεδίο το οποίο θα δείχνει τον αριθμό των βομβών που καλείται ο χρήστης να βρει.

Τέλος, θα υπάρχει ιστορικό των νικηφόρων παιχνιδιών, το οποίο θα αποθηκεύεται μέσω cookies στον περιηγητή του χρήστη.

Τα κουμπιά, τα τετράγωνα, οι βόμβες, οι σημαίες, το ταμπλό και τα υπόλοιπα γραφικά στοιχεία του παιχνιδιού θα υλοποιηθούν σε πρόγραμμα επεξεργασίας φωτογραφίας (Photoshop).

Μέσω του κώδικα html, της χρήσης δηλώσεων CSS και της γλώσσας προγραμματισμού JavaScript, θα δοθεί ιδιαίτερη μνεία, ώστε το παιχνίδι να μπορεί να παίζεται σε σχεδόν όλα τα μεγέθη οθονών, από υπολογιστή έως κινητό τηλέφωνο, προσαρμόζοντας αυτόματα το μέγεθος των στοιχείων του και δίνοντας στο χρήστη δευτερεύουσες επιλογές για το “μαρκάρισμα” των κουμπιών – τετραγώνων.

Summary

The purpose of this project is the analysis and application of rules and theory for the creation of an interactive game, known to us all as Minesweeper.

The implementation of the application made using HTML5, CSS and pure JavaScript, without using other additional libraries.

The game includes a dashboard with buttons that will hide an empty square, number, or bomb. When the user presses a hidden empty square button, it opens and causes the neighboring empty squares to open. This is repeated until a block with a bomb or block number is found. Each square number indicates how many bomb squares adjoin to it. If the user clicks on a button that hides square bomb the game ends and the user "loses", while if the user marks correctly guessing and figuring out the numbers, all the blocks with a bomb, the game ends and the user "wins" the game.

Regarding the functionality of the game, there is a choice of default and user-defined levels of difficulty. In the game screen, the user is informed about the time that has elapsed and the bombs still to be found. Also implemented history of successful games.

JavaScript

Η javascript είναι μια γλώσσα η οποία έχει σχεδιαστεί και χρησιμοποιείται για να υλοποιήσει την έννοια της διαδραστικότητας σε html σελίδες.

Είναι μια scripting γλώσσα, δηλαδή ο κώδικας που αναπτύσσεται από τον προγραμματιστή, εκτελείται χωρίς να πρέπει πρώτα να περάσει από την διαδικασία της μετατροπής σε γλώσσα μηχανής (compiling).

Η JavaScript εκτελείται μέσα από το περιβάλλον ενός περιηγητή όπως ο Firefox, ο Chrome, ο Internet Explorer κλπ.

Αν και το όνομά της παραπέμπει στη γλώσσα Java, δεν είναι σε καμία περίπτωση η ίδια γλώσσα και δε θα πρέπει να συγχέεται με αυτή.

Και οι δύο γλώσσες αναπτύχθηκαν από την εταιρεία Sun, αν και αρχικά η JavaScript ξεκίνησε να αναπτύσσεται αυτόνομα από την Netscape πολύ σύντομα συμφώνησε να συνεργαστεί με τη Sun.

Οι δύο λοιπόν γλώσσες μοιάζουν απλά μεταξύ τους αλλά είναι πολύ διαφορετικές στον τρόπο υλοποίησης και εκτέλεσής τους.

Η Java χρειάζεται ένα περιβάλλον ανάπτυξης και μεταγλώττισης, το JDK ή Java Developers Kit που προσφέρεται δωρεάν από την εταιρεία Sun και για την εκτέλεση των προγραμμάτων της, ο χρήστης θα πρέπει να έχει εγκαταστήσει στον υπολογιστή του το περιβάλλον εκτέλεσης Java, το οποίο επίσης διατίθεται δωρεάν.

Αντίθετα, η JavaScript δε χρειάζεται μεταγλώττιση. Γράφοντας μία σειρά από εντολές της σε ένα απλό αρχείο κειμένου, όπως είναι και ο κώδικας HTML, ο οποιοσδήποτε περιηγητής θα τις εκτελέσει χωρίς επιπρόσθετες ενέργειες.

Σαφώς δεν έχουν οι δύο γλώσσες τις ίδιες δυνατότητες. Η Java είναι μία πλήρη γλώσσα προγραμματισμού ενώ η JavaScript ένα απλό υποσύνολό της που επικεντρώνεται στη χρήση του Internet και την αντίδραση κυρίως του περιηγητή σε γεγονότα και ενέργειες του χρήστη.

Έτσι, με τη JavaScript μπορούμε να εκτελέσουμε κάποιες ενέργειες – εργασίες, όταν για παράδειγμα ο χρήστης κάνει κλικ, πατήσει δηλαδή με το αριστερό κουμπί του ποντικιού του ή πατήσει στιγμιαία με το δάκτυλό του στην οθόνη του κινητού τηλεφώνου, πάνω σε ένα html στοιχείο και να λαμβάνουμε τα αντίστοιχα αποτελέσματα.

Επίσης μία ιδιαίτερα χρήσιμη δυνατότητα της JavaScript είναι ότι πολύ απλά και εύκολα μπορεί να διαβάσει, να αναζητήσει και να αλλάξει τα περιεχόμενα και τις ιδιότητες ενός html στοιχείου.

Μπορεί επίσης να χρησιμοποιηθεί για να επικυρώσει δεδομένα μίας φόρμας δεδομένων (validate) πριν αυτά υποβληθούν στον server, γλυτώνοντάς τον από επιπλέον ελέγχους.

Ένα επίσης σημαντικό στοιχείο είναι ότι με την βοήθεια της JavaScript μπορούμε να εντοπίσουμε τον περιηγητή του επισκέπτη και ανάλογα με τον τύπο του, να στείλουμε και να φορτώσουμε τελικά σε αυτόν, την αντίστοιχη σελίδα που είναι κατάλληλα φτιαγμένη για αυτόν τον περιηγητή αλλά και το είδος της συσκευής που ο περιηγητής εκτελείται, όπως για παράδειγμα ένας υπολογιστής ή ένα κινητό τηλέφωνο ή ένα tablet.

Επιπρόσθετα, μία δυνατότητα που χρησιμοποιείται και στην υλοποίηση του τρέχοντος παιχνιδιού, μπορούμε να δημιουργούμε cookies, τα οποία είναι μικρά αρχεία πληροφοριών και μέσω αυτών, να αποθηκεύουμε και να λαμβάνουμε πληροφορίες στον υπολογιστή του επισκέπτη.

Η σύνταξή της είναι σχετικά απλή όπως αναφέρθηκε ήδη.

Για να εισάγουμε ένα κώδικα JavaScript σε ένα html αρχείο, χρησιμοποιούμε σε αυτό την ετικέτα <script> και μέσα σε αυτή την ετικέτα χρησιμοποιούμε το όρισμα “type” για να ορίσουμε την scripting γλώσσα που θα χρησιμοποιήσουμε.

Οι ετικέτες <script>...</script> μας δηλώνουν που αρχίζει και που τελειώνει ο κώδικας JavaScript.

Εάν ο κώδικας δεν περικλείεται από τις ετικέτες <script>...</script>, τότε ο περιηγητής θα τις θεωρήσει απλά κείμενο και δε θα τις εκτελέσει!

Υπάρχει και δεύτερος τρόπο εισαγωγής κώδικα JavaScript σε ένα αρχείο html και αυτό είναι με την κλήση και ενσωμάτωση εξωτερικού αρχείου, συνήθως με επέκταση .js και πάλι μέσω χρήσης των ετικετών <script>...</script>, μόνο που σε αυτή την περίπτωση γίνεται χρήση του ορίσματος "src" στο οποίο αναφέρεται το όνομα του αρχείου με τον κώδικα JavaScript.

Σημαντικό σημείο είναι το ότι στα αρχεία .js δεν πρέπει να υπάρχουν οι ετικέτες <script>...</script>.

Ο κώδικας προς εκτέλεση της JavaScript μπορεί να εμφανίζεται σε οποιοδήποτε σημείο ενός html αρχείου, είτε στην ενότητα <body>, είτε στην ενότητα <head>. Συνήθως συναντάται στην ενότητα <head>. Προγραμματιστικά όμως είναι καλύτερο να ενσωματώνεται στο τέλος του αρχείου html. Έτσι η στατική δομή της σελίδας θα έχει ήδη φορτωθεί από τον περιηγητή και τα δομικά της στοιχεία θα είναι πλέον διαθέσιμα στην JavaScript.

Σήμερα πλέον, όλοι οι περιηγητές είναι συμβατοί με τη JavaScript. Το γεγονός ότι η γλώσσα υποστηρίζει τόσο τον functional προγραμματισμό όσο και τον αντικειμενοστραφή (object oriented) την κάνει ακόμα πιο δημοφιλή.

Η γλώσσα είναι πολύ γρήγορη και χρήσιμη και επιτρέπει γρήγορη επικοινωνία ανάμεσα στον περιηγητή και τον server.

Χρησιμοποιείται με ενιαίο τρόπο στο διαδίκτυο, ανεξαρτήτως από την γλώσσα προγραμματισμού στον server (PHP, Python κλπ).

Πολύ βασικό στοιχείο είναι το ότι, ενώ μπορεί η σύνδεση στο διαδίκτυο να μην είναι πάντα εφικτή, η JavaScript όμως δουλεύει ακόμα και όταν ο περιηγητής είναι εκτός σύνδεσης.

Η γλώσσα είναι συμβατή με όλα τους σύγχρονους υπολογιστές, Tablets, κινητά τηλέφωνα και γενικότερα κάθε συσκευή που περιέχει κάποιον περιηγητή, ο οποίος χρησιμοποιείται για ανάγνωση και προβολή σελίδων html.

Αν και αρχικά όπως αναφέραμε η γλώσσα αναπτύχθηκε από μία και μόνη εταιρεία για χρήση από τον περιηγητή της, σήμερα πλέον, η

γλώσσα χρησιμοποιείται από τις μεγαλύτερες εταιρίες στον χώρο της τεχνολογίας, όπως οι Apple, Google και η Microsoft.

Τα τελευταία χρόνια η JavaScript έχει τραβήξει ακόμη περισσότερο τα βλέμματα καθώς υλοποιούνται πολλά ενδιαφέροντα λογισμικά που την υποστηρίζουν. Η γλώσσα έχει εδραιωθεί στο διαδίκτυο, ενώ παράλληλα έχει αρχίσει να επεκτείνεται και σε άλλους τομείς καθώς οι βιβλιοθήκες που έχουν υλοποιηθεί σε αυτήν την κάνουν ακόμα πιο ευέλικτη και πιο αναγνωρίσιμη.

Κάποιες πολύ ενδιαφέρουσες βιβλιοθήκες που αναδεικνύουν την χρησιμότητα της γλώσσας αλλά και την συνεισφορά της στον προγραμματισμό για το διαδίκτυο είναι οι:

JQuery

Είναι η βιβλιοθήκη της JavaScript που έχει χρησιμοποιηθεί όσο καμία άλλη όλα αυτά τα χρόνια.

Η JQuery απλοποιεί τον προγραμματισμό καθώς με την βοήθεια της μπορούν να υλοποιηθούν εργασίες πολύ απλά, οι οποίες με την συμβατική JavaScript θα ήταν ιδιαίτερα πολύπλοκες.

Στα πλεονεκτήματά της συγκαταλέγεται και ο μεγάλος αριθμός δυνατοτήτων και λειτουργιών που έχει, καθώς και η μεγάλη κοινότητα ανοικτού λογισμικού (open-source community) που την υποστηρίζει.

Θεωρείται από τις απαραίτητες βιβλιοθήκες για την υλοποίηση μιας διαδικτυακής εφαρμογής, είτε αυτή είναι απλή, είτε όχι.

Node.JS

Οι τεχνολογίες στους servers αλλάζουν συνεχώς και οι προγραμματιστές έχουν περάσει από διάφορες γλώσσες όπως PHP, JSP, .NET, Ruby, Python κ.α. Πάντοτε, όμως χρησιμοποιούσαν την JavaScript για τον προγραμματισμό στην μεριά του πελάτη - επισκέπτη (περιηγητή).

Με τον καιρό, οι προγραμματιστές αντιλήφθηκαν πως το να χρησιμοποιούν δύο διαφορετικές γλώσσες έκανε δυσκολότερη την κατάσταση. Έτσι η απάντηση ήταν μία: JavaScript και στον server!

Η Node.js χρησιμοποιείται στις εφαρμογές πραγματικού χρόνου (real-time), στέλνοντας δεδομένα μέσω web sockets.

Αυτό που την κάνει τόσο επαναστατική είναι ότι μετά από 20 χρόνια με χρήση μόνο απλών ερωτο-απαντήσεων τύπου αίτημα – απάντηση (request – response), πλέον μπορούν να υπάρξουν εφαρμογές πραγματικού χρόνου με επικοινωνία που ξεκινά και από τις δυο μεριές, του πελάτη αλλά και του server, οι οποίοι μπορούν να ανταλλάξουν δεδομένα ελεύθερα, χωρίς περιορισμούς.

Αυτό έρχεται σε αντίθεση με τα όσα ίσχυαν μέχρι πριν λίγα χρόνια, όταν μόνο ο πελάτης (περιηγητής) μπορούσε να ξεκινήσει την επικοινωνία (Ajax).

Επιπροσθέτως, η Node.js, βασίζεται στην ισχύουσα δομή του web (HTML, CSS και JS) και χρησιμοποιεί την διαδεδομένη θύρα 80.

Angular JS

Το ενδιαφέρον framework ανοιχτού λογισμικού της Google, αλλάζει τα δεδομένα στον τομέα στον διαδικτυακών εφαρμογών, καθώς έχει σχεδιαστεί με τους εξής στόχους:

Αποδέσμευση του DOM από την λογική της εφαρμογής. Αυτό βελτιώνει την δυνατότητα ελέγχου του κώδικα.

Ο έλεγχος της εφαρμογής έχει ίση σημασία με την διαδικασία υλοποίησης. Η δυσκολία των δοκιμών επηρεάζεται δραματικά από τον τρόπο που είναι δομημένος ο κώδικας.

Αποσύνδεση της πλευράς του πελάτη (περιηγητής) από την πλευρά του διακομιστή (server). Αυτό επιτρέπει το έργο να προχωρήσει παράλληλα και επιτρέπει την επαναχρησιμοποίηση κώδικα και στις δύο πλευρές.

Οδηγός για τους προγραμματιστές σε όλη τη διαδικασία υλοποίησης μιας εφαρμογής: από το σχεδιασμό του UI, στη δοκιμή.

Ακολουθεί το μοτίβο MVC της μηχανικής λογισμικού και ενθαρρύνει τη χαλαρή σύνδεση μεταξύ παρουσίασης, δεδομένων και λογικής.

Ένα μεγάλο μέρος της επιβάρυνσης για το backend μειώνεται, οδηγώντας σε πολύ ελαφρύτερες web εφαρμογές.

Γενικά, το AngularJS (http://angularjs.org/) δένει την HTML (views) σε αντικείμενα JavaScript (models). Όταν συμβαίνει μια αλλαγή στα models, η σελίδα ανανεώνεται αυτόματα. Συμβαίνει επίσης και το αντίθετο, όταν δηλαδή ένα text field ενημερώνεται στο view, το AngularJS χειρίζεται την κατάσταση αυτόματα χωρίς να χρειάζεται να

γίνει κάποια επιπλέον ενέργεια στην HMTL, ή στην JQuery με την σύνδεση κάποιου event.

PhoneGap

Τα τελευταία χρόνια, όμως, παρατηρείται μια κατακόρυφη αύξηση της χρήσης κινητών συσκευών και των εφαρμογών τους. Η JavaScript έχει αρχίσει να διεισδύει και σε αυτόν τον τομέα και μάλιστα με πολύ επιτυχημένα, πρώτα βήματα, όπως το PhoneGap (ή αλλιώς Apache Cordova).

Το PhoneGap είναι ένα framework για ανάπτυξη λογισμικού σε κινητές συσκευές που έχει πλέον αγοραστεί από την Adobe Systems και το οποίο δίνει την δυνατότητα στους προγραμματιστές, να δημιουργήσουν εφαρμογές για κινητές συσκευές με χρήση HTML5, CSS3 και JavaScript και όχι στις γλώσσες που υποστηρίζονται ανά συσκευή.

Οι εφαρμογές που προκύπτουν είναι υβριδικές. Δηλαδή, δεν κάνουν χρήση των frameworks που ορίζει το λειτουργικό της κινητής συσκευής, αλλά ταυτόχρονα δεν αποτελούν καθαρά web-based εφαρμογές.

Οι πλατφόρμες κινητών συσκευών, όπως το iOS, το Android και το Windows έχουν διαφορετικές απαιτήσεις, κανόνες και γλώσσες προγραμματισμού μεταξύ τους. Με την χρήση όμως του PhoneGap, η υλοποίηση πλέον μιας εφαρμογής γίνεται μια φορά για όλες τις πλατφόρμες.

Παρόλα αυτά, το PhoneGap έχει κάποια μειονεκτήματα, όπως ότι δεν μπορεί να χρησιμοποιηθεί για την υλοποίηση εφαρμογών με πολλά γραφικά (πχ παιχνίδι) αλλά και το γεγονός πως δεν περιέχει κάποια βάση έτοιμου UI περιεχομένου, με αποτέλεσμα μερικές φορές η υλοποίηση να κρατάει περισσότερο.

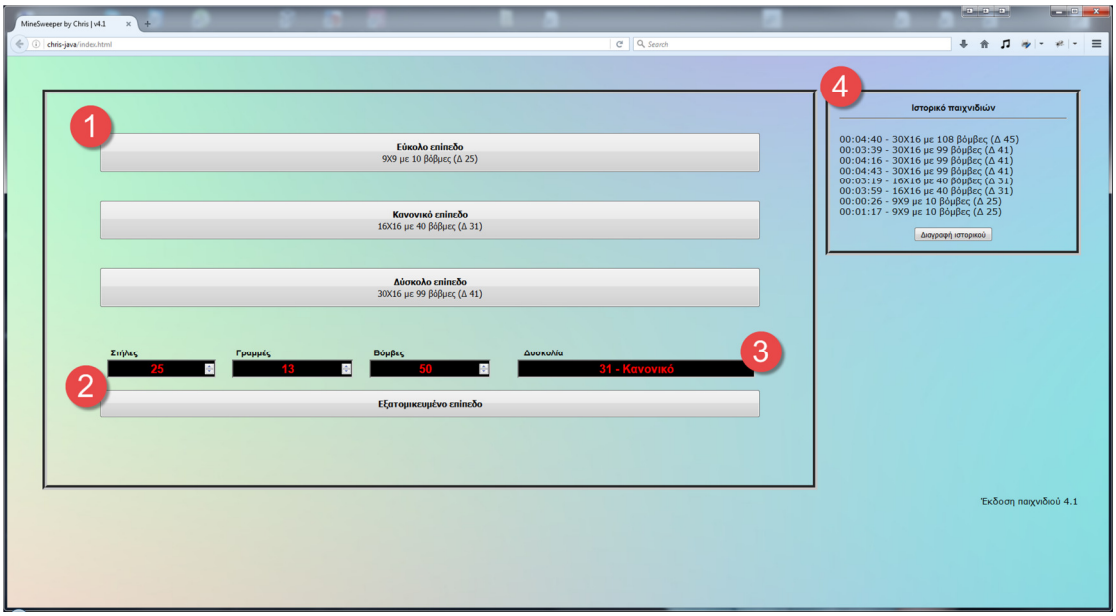
Όμως, το πρώτο βήμα για την ενοποίηση όλων των πλατφόρμων που χρησιμοποιούνται στις κινητές συσκευές έγινε. Και έγινε με την βοήθεια της JavaScript.

Το παιχνίδι

Στην ενότητα αυτή θα αναλύσουμε το παιχνίδι, όπως εμφανίζεται στην οθόνη του παίκτη – επισκέπτη της σελίδας.

Θα δούμε τις οθόνες που θα συναντήσει, καθώς και τον τρόπο που καλείται να παίξει, τα βήματα που μπορεί να ακολουθήσει, τα μηνύματα που θα δει και τον τρόπο ολοκλήρωσης του παιχνιδιού.

Αρχική οθόνη



Στην αρχική οθόνη του παιχνιδιού εμφανίζονται τα στοιχεία επιλογής επιπέδου δυσκολίας και έναρξης καθώς και το ιστορικό των παιχνιδιών.

Στην περιοχή (1) έχουν τοποθετηθεί τρία κουμπιά, τα οποία ξεκινούν ένα νέο παιχνίδι με τον ανάλογο συνδυασμό στηλών, γραμών και βομβών.

Έτσι μπορεί ο χρήστης να ξεκινήσει ένα παιχνίδι:

- εύκολου επιπέδου, το οποίο θα έχει εννέα στήλες και γραμμές με δέκα βόμβες
- κανονικού επιπέδου, με δεκαέξη στήλες και γραμμές και 40 βόμβες
- δύσκολου επιπέδου, με τριάντα στήλες, δεκαέξη γραμμές και 99 βόμβες

Η περιοχή (2) δίνει στο χρήστη τη δυνατότητα να φτιάξει το δικό του παιχνίδι, όπου μπορεί να επιλέξει τις στήλες, τις γραμμές και τις βόμβες που αυτό θα περιέχει.

Στην περιοχή (3) εμφανίζεται το επίπεδο δυσκολίας του παιχνιδιού, το οποίο έχει χωριστεί σε έξη βασικές κατηγορίες ανάλογα με το επίπεδο δυσκολίας:

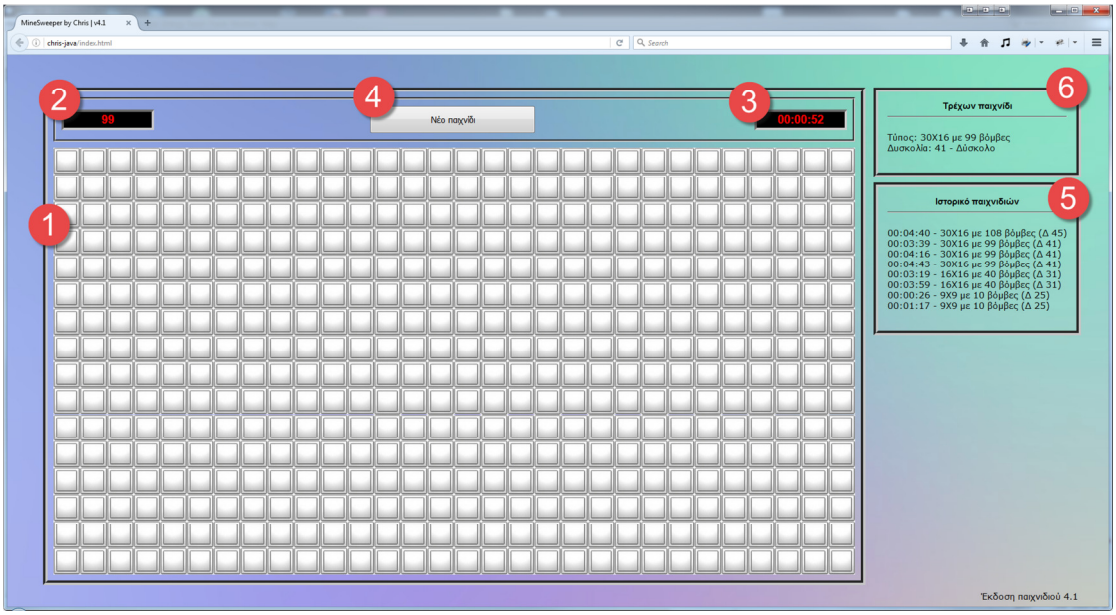
1. από 01 έως 14 – Αρχάριο
2. από 15 έως 29 – Εύκολο
3. από 30 έως 39 – Κανονικό
4. από 40 έως 49 – Δύσκολο
5. από 50 έως 69 – Πολύ δύσκολο
6. από 70 έως 99 – Extreme

Στην περιοχή (4) εμφανίζεται το ιστορικό των τελευταίων είκοσι παιχνιδιών που ο παίκτης έχει επιτυχώς ολοκληρώσει (βρήκε δηλαδή σωστά όλες τις βόμβες).

Η εμφάνιση γίνεται πρώτα κατά σειρά επιπέδου και στη συνέχεια κατά σειρά χρόνου ολοκλήρωσης. Έτσι παιχνίδια με δυσκολότερο επίπεδο δυσκολίας εμφανίζονται πάντα στην κορυφή, ενώ τα ευκολότερα στο κάτω μέρος. Ανάμεσα σε παιχνίδια ίδιου επιπέδου, όσο λιγότερο χρόνο ολοκλήρωσης έχει ένα παιχνίδι, τόσο ψηλότερα θα εμφανιστεί.

Τέλος, στο κάτω μέρος της περιοχής εμφανίζεται το κουμπί καθαρισμού και διαγραφής του ιστορικού του παίκτη.

Οθόνη παιχνιδιού



Αφού ο παίκτης επιλέξει το επίπεδο δυσκολίας που θέλει να παίξει, μέσω των ανάλογων κουμπιών της αρχικής οθόνης (ένα από τα τρία έτοιμα ή εξατομικευμένο), εμφανίζεται η κύρια οθόνη του παιχνιδιού.

Η περιοχή (1) αποτελείται από ένα πλέγμα κουμπιών με τις ανάλογες στήλες και γραμμές.

Στην περιοχή (2) εμφανίζονται οι βόμβες που έχουν απομείνει να βρει ο παίκτης, από τις οποίες αφαιρείται μία κάθε φορά που κάποια θέση του πλέγματος μαρκάρεται ως βόμβα από τον παίκτη.

Στην περιοχή (3) αναγράφεται ο χρόνος που έχει παρέλθει από την έναρξη του παιχνιδιού.

Με το κουμπί στην περιοχή (4) δίνεται στον παίκτη η δυνατότητα να εγκαταλείψει το τρέχων παιχνίδι και να επιστρέψει στην αρχική οθόνη. Με την ενέργεια αυτή, οι βόμβες που έχει βρει και ο χρόνος που έχει παρέλθει ακυρώνονται και δεν καταγράφονται στο ιστορικό.

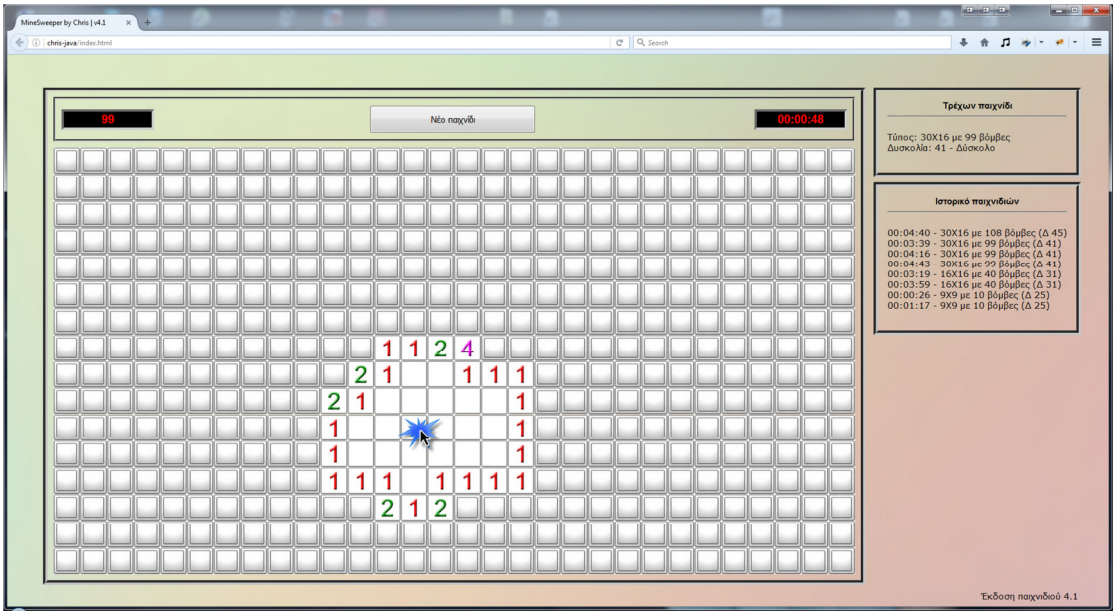
Στην περιοχή (5), όπως και στην αρχική οθόνη, εμφανίζεται το ιστορικό των τελευταίων είκοσι παιχνιδιών, από όπου όμως τώρα απουσιάζει το κουμπί διαγραφής του ιστορικού.

Τέλος, στην περιοχή (6) εμφανίζονται πληροφοριακά στοιχεία σχετικά με το τρέχων παιχνίδι, όπως ο τύπος του παιχνιδιού (στήλες, γραμμές και βόμβες) καθώς και το επίπεδο δυσκολίας.

Το πρώτο κλικ

Κάνοντας κλικ ο παίκτης σε κάποιο κελί υπάρχουν τρεις διαφορετικές περιπτώσεις.

Η πρώτη περίπτωση είναι το κελί αυτό να είναι άδειο, να μην περιέχει δηλαδή κάποιον αριθμό ή βόμβα.

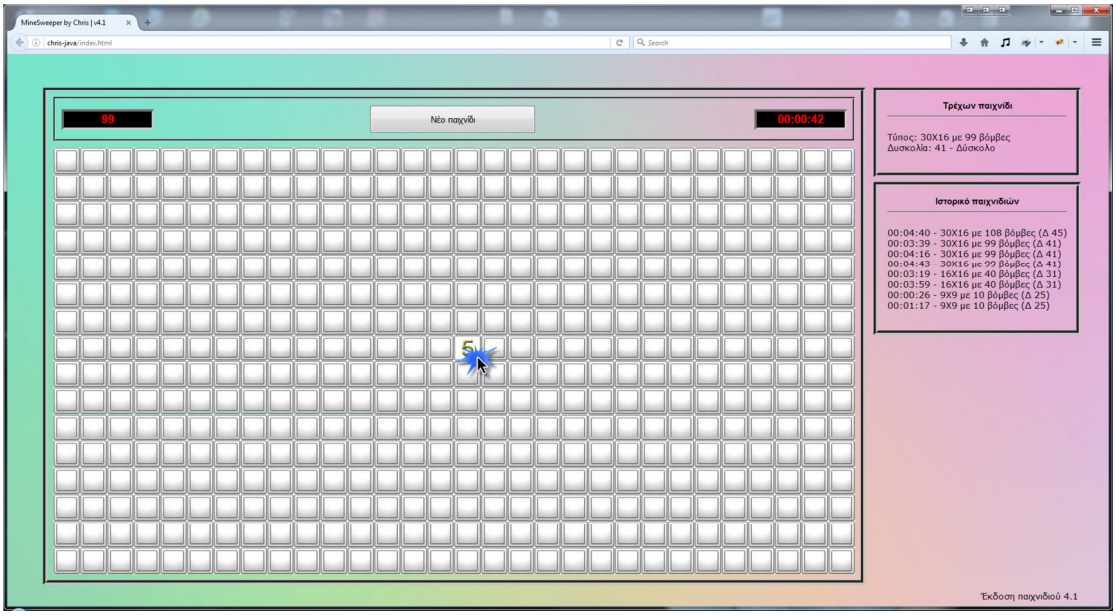


Τότε όλα τα γειτονικά άδεια κελιά θα ανοίξουν και η διαδικασία αυτή θα σταματήσει στα κελιά που περιέχουν κάποιον αριθμό. Ο αριθμός αυτός δείχνει με πόσα κελιά που περιέχουν βόμβα ακουμπάει το κελί αυτό. Όταν λέμε ακουμπάει, εννοούμε σε οποιαδήποτε από τα οκτώ γειτονικά του κελιά, οριζόντια, κάθετα και διαγώνια.

Καλείται τώρα ο παίκτης να μαντέψει, συνδυάζοντας τις πληροφορίες γειτονίας, πιο κελί είναι βόμβα, οπότε και να το μαρκάρει ή ότι είναι άδειο ή περιέχει αριθμό, οπότε και να κάνει κλικ επάνω του ώστε να ανοίξουν νέα άδεια κελιά ή να πληροφορηθεί εκ νέου με πόσα κελιά βόμβες επικοινωνεί το νέο αυτό κελί.

Με τον τρόπο αυτό, ανοίγοντας δηλαδή νέα κελιά και μαρκάροντας αυτά που πιθανώς είναι βόμβες, ο παίκτης προχωρεί έως ότου μαρκάρει όλες τις βόμβες ή από λανθασμένο υπολογισμό πέσει πάνω σε βόμβα.

Η δεύτερη περίπτωση είναι το κελί αυτό να περιέχει κάποιον αριθμό.



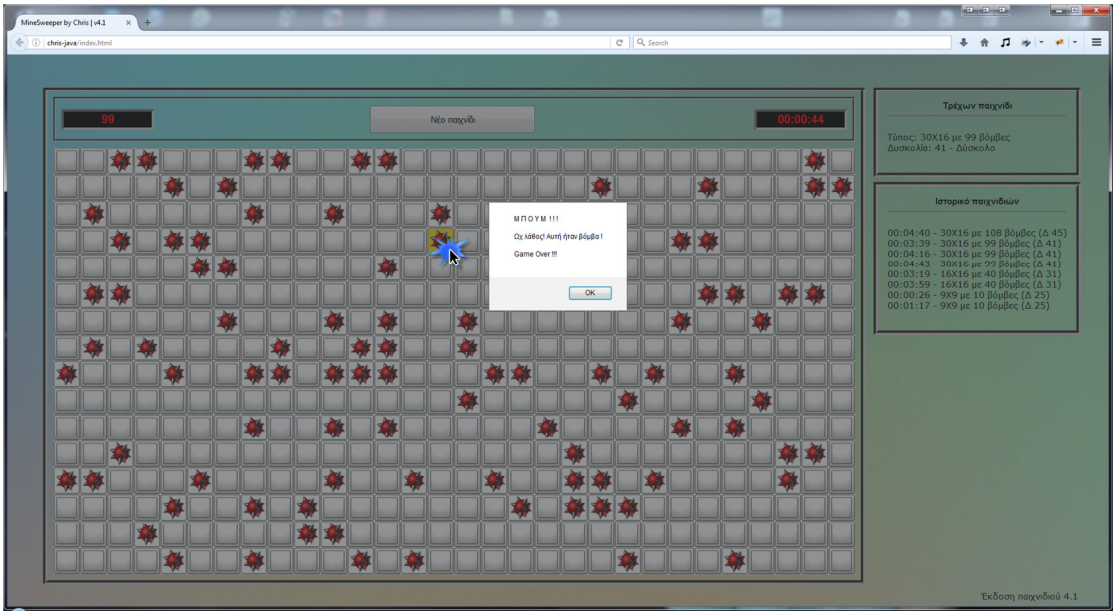
Δυστυχώς, η περίπτωση αυτή δεν είναι η πλέον επιθυμητή, καθώς θα ανοίξει μόνο το κελί αυτό, με την ελάχιστη πληροφορία που μπορεί να παρέχει στον παίκτη, με πόσα δηλαδή κελιά βόμβες ακουμπάει το κελί αυτό…

Σε τέτοιες περιπτώσεις ο παίκτης καλείται να επαναλάβει τη διαδικασία της “τυφλής” αναζήτησης άδειου κελιού. Θα πρέπει δηλαδή να κάνει τυχαία κλικ σε γειτονικά ή άλλα απομακρυσμένα κελιά, μέχρις ότου πέσει επάνω σε κάποιο άδειο (χωρίς δηλαδή αριθμό κελί).

Συνήθως η διαδικασία αυτή ακολουθείται στην έναρξη του παιχνιδιού τις περισσότερες φορές.

Μόνο αφού ανοίξουν κάποια κελιά με επαρκή πληροφόρηση γειτονίας με βόμβες, μπορεί ο παίκτης να ξεκινήσει τους υπολογισμούς του.

Στη τρίτη περίπτωση το κελί θα είναι δυστυχώς βόμβα!



Στην περίπτωση αυτή, ο χρόνος σταματά να τρέχει και ένα μήνυμα πληροφορεί τον παίκτη ότι το κελί αυτό είναι βόμβα.

Όλα τα κελιά με βόμβα αποκαλύπτονται και το επιλεγμένο φωτίζεται με κίτρινο χρώμα.

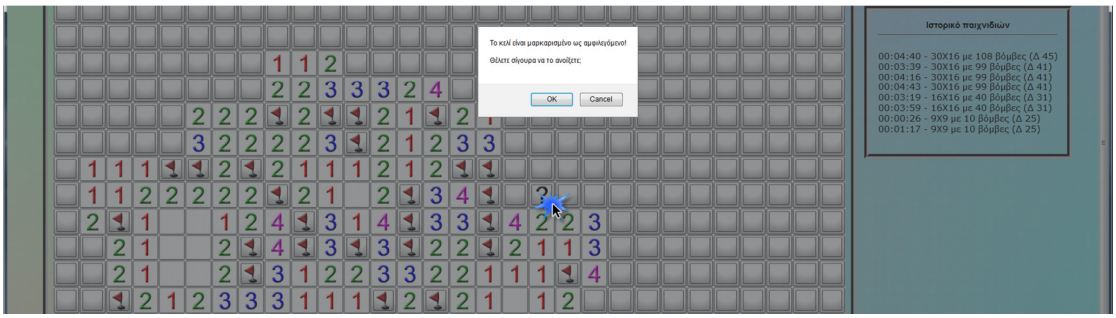
Η μόνη επιλογή του παίκτη είναι η έναρξη νέου παιχνιδιού.

Όπως και στην προηγούμενη περίπτωση, όπου ο παίκτης έπεσε πάνω σε κελί με αριθμό, η διαδικασία αυτή συναντάται αρκετά συχνά στην έναρξη του παιχνιδιού.

Μόνο αφού ο παίκτης βρει στη αρχή του παιχνιδιού κενό κελί, το οποίο θα ανοίξει άλλα και θα του δώσει πληροφορίες σχετικά με τα γειτονικά κελιά βόμβες, θα μπορέσει να ξεκινήσει το παιχνίδι.

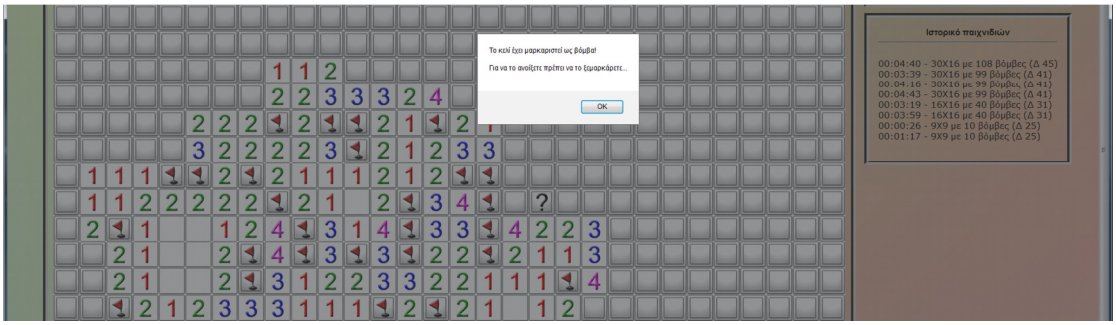
Το παιχνίδι σε εξέλιξη

Το παιχνίδι εξελίσσεται καθώς ο παίκτης κάνει κλικ στα κελιά που κρίνει ότι είναι άδεια ή περιέχουν κάποιον αριθμό, βασιζόμενος στο συνδυασμό πληροφοριών που παίρνει από τα ήδη ανοικτά κελιά με αριθμό. Σε κάθε κελί που κρίνει ότι είναι σίγουρα βόμβα κάνει δεξί κλικ και με τον τρόπο αυτό το μαρκάρει τοποθετώντας επάνω του μία κόκκινη σημαία. Στην περίπτωση που δεν είναι σίγουρος για κάποιο κελί, μπορεί να πατήσει δύο φορές το δεξί κλικ. Τότε το κελί αυτό μαρκάρεται ως αμφιλεγόμενο και εμφανίζεται επάνω του ένα ερωτηματικό.



Εάν κατά λάθος προσπαθήσει να κάνει κλικ επάνω στο κελί αυτό, τότε ένα μήνυμα τον προειδοποιεί ότι το κελί θεωρείται αμφιλεγόμενο και τον καλεί να επιβεβαιώσει ότι θέλει να το ανοίξει.

Αντίστοιχο μήνυμα εμφανίζεται όταν ο παίκτης κάνει κλικ σε κουμπί μαρκαρισμένο ως βόμβα.

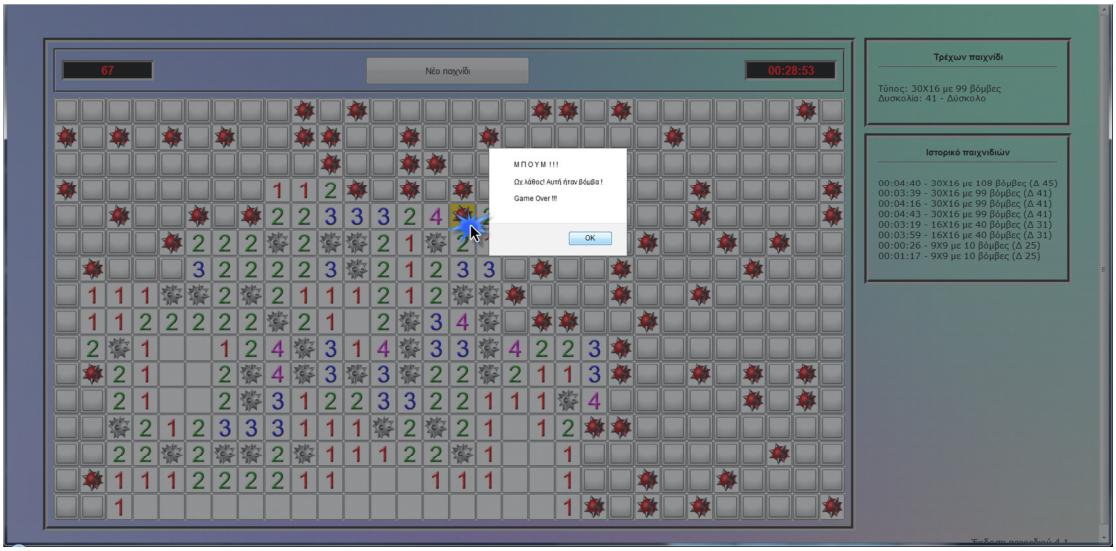


Στην περίπτωση όμως αυτή, δεν δίνεται η δυνατότητα ανοίγματος του κελιού, αλλά του προτείνει να κάνει δεύτερη και τρίτη φορά κλικ επάνω του ώστε να καθαριστεί κάθε σημάδι μαρκαρίσματος και στη συνέχεια μπορεί φυσικά να κάνει κλικ επάνω του και να το ανοίξει.

Ολοκλήρωση του παιχνιδιού

Το παιχνίδι ολοκληρώνεται – τερματίζει με δύο τρόπους.

Πρώτος τρόπος, ο παίκτης κάνει κλικ σε κελί που περιέχει βόμβα.



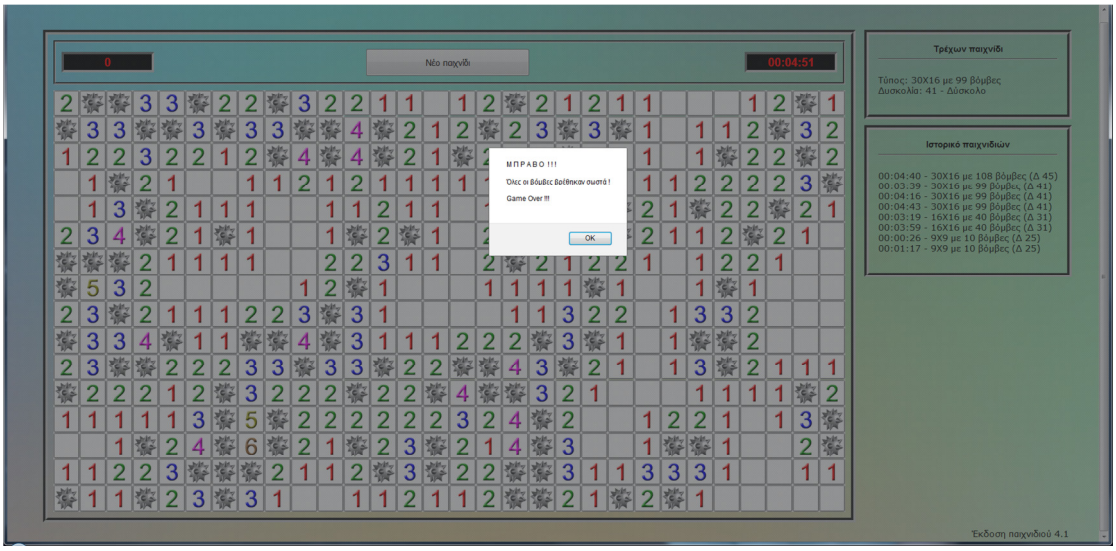
Αν από λάθος υπολογισμό του παίκτη ή επειδή τα στοιχεία είναι ανεπαρκή και μόνο με τυχαία επιλογή κάποιου κελιού μπορεί να προχωρήσει το παιχνίδι, γίνει κλικ σε κελί που τελικά περιέχει βόμβα, τότε ένα μήνυμα ενημερώνει τον παίκτη ότι δυστυχώς έχασε.

Όλα τα κελιά που περιέχουν βόμβα εμφανίζονται με μία κόκκινη βόμβα. Τα κελιά που ήδη έχει μαρκάρει ο παίκτης ως βόμβα, τοποθετώντας επάνω τους την ανάλογη σημαία, μετατρέπονται σε γκρίζα βόμβα και το κελί που λανθασμένα έκανε επάνω του κλικ ο παίκτης εμφανίζεται με κόκκινη βόμβα και κίτρινο χρώμα στο υπόβαθρο.

Ο χρόνος σταματάει να μετράει και τα στατιστικά στοιχεία του παιχνιδιού (χρόνος που έχει παρέλθει από την αρχή του παιχνιδιού καθώς και οι βόμβες που έχουν ήδη βρεθεί) χάνονται και δεν καταγράφονται στο ιστορικό.

Αφού ο παίκτης κλείσει τον μήνυμα, η οθόνη του παραμένει στο παιχνίδι και η μόνο επιλογή που έχει πλέον ο παίκτης, είναι πατώντας το κουμπί «Νέο παιχνίδι» να επιστρέψει στην αρχική οθόνη, για να ξεκινήσει και πάλι ένα νέο γύρο παιχνιδιού.

Δεύτερος τρόπος ολοκλήρωσης του παιχνιδιού είναι ο παίκτης να βρει με επιτυχία όλες τις βόμβες.



Στην περίπτωση αυτή ο χρόνος σταματάει και πάλι και το ανάλογο μήνυμα ειδοποιεί τον παίκτη για την επιτυχή ολοκλήρωση του παιχνιδιού.

Τώρα όμως τα στατιστικά στοιχεία του παιχνιδιού θα καταγραφούν από το πρόγραμμα και στην επόμενη μεταφορά στην αρχική οθόνη, θα εμφανιστούν την λίστα του ιστορικού εάν ο χρόνος ολοκλήρωσης βρίσκεται μέσα στις πρώτες είκοσι θέσης κατάταξης.

Οι χρόνοι κατάταξης εμφανίζονται καταγράφονται με βάση το επίπεδο δυσκολίας του παιχνιδιού.

Έτσι ένα παιχνίδι μεγαλύτερης δυσκολίας θα καταταχθεί σε υψηλότερη θέση, ακόμα και αν ο χρόνος ολοκλήρωσής του είναι μεγαλύτερος από το χρόνο ολοκλήρωσης ενός παιχνιδιού μικρότερου επιπέδου δυσκολίας, το οποίο και είναι φυσιολογικό να ολοκληρωθεί σε συντομότερο χρόνο!

Αυτός ο τρόπος κατάταξης και εμφάνισης του ιστορικού είναι δικαιότερος από το να καταγράφονται τα αποτελέσματα απλά και μόνο βάση του χρόνου ολοκλήρωσης, καθώς τότε κανένα παιχνίδι υψηλού επιπέδου δυσκολίας δε θα καταγραφόταν, αφού ο χρόνος ολοκλήρωσής του θα είναι πάντα μεγαλύτερος από ένα ευκολότερο παιχνίδι…

Υλοποίηση – Κώδικας

Η υλοποίηση του παιχνιδιού στηρίχθηκε στη χρήση τεσσάρων βασικών στοιχείων.

Βασικά λειτουργικά στοιχεία

Το πρώτο στοιχείο είναι το αρχείο **index.html** το οποίο περιέχει τη βασική δομή των στοιχείων που εμφανίζονται στην οθόνη του παίκτη – επισκέπτη.

Το δεύτερο στοιχείο είναι το αρχείο **common.css** το οποίο περιέχει τις δηλώσεις μορφοποίησης κάθε στοιχείου που εμφανίζεται στην οθόνη του επισκέπτη.

Το τρίτο στοιχείο είναι διάφορες **εικόνες τύπου JPG** οι οποίες εμφανίζονται στην οθόνη του επισκέπτη προσδίδοντας χρώμα και εικόνα στο σύνολο του παιχνιδιού και ορίζουν το εικαστικό κομμάτι της εφαρμογής, σε συνδυασμό πάντα με το δεύτερο στοιχείο το οποίο ορίζει τον τρόπο εμφάνισής τους μέσα στο περιβάλλον του παιχνιδιού.

Το τέταρτο και τελευταίο στοιχείο είναι το αρχείο **game.js** το οποίο περιέχει τον κώδικα σε JavaScript και υλοποιεί την διαδραστικότητα του παιχνιδιού.

Κάθε ένα από τα στοιχεία αυτά επιτελεί το δικό του έργο και συνεργάζεται με τα υπόλοιπα τρία.

Το πρώτο στοιχείο ορίζει τα βασικά στατικά στοιχεία του παιχνιδιού, το δεύτερο ορίζει τον τρόπο με τον οποίο κάθε στοιχείο θα φαίνεται ή δε θα φαίνεται, το τρίτο αποστέλλεται και εμφανίζεται στην οθόνη του επισκέπτη κατόπιν των ανάλογων αιτήσεων και το τέταρτο αναλαμβάνει να κάνει το παιχνίδι διαδραστικό και λειτουργικό, αντιδρώντας στις ενέργειες του παίκτη και θέτοντας τους όρους και τους κανόνες εμφάνισης και απόκρυψης στοιχείων στην οθόνη του.

Ανάλυση – INDEX.HTML

Το στοιχείο (αρχείο) αυτό είναι το κύριο που καλείται και φορτώνεται στον περιηγητή (browser) του παίκτη.

Περιέχει μόνο HTML κώδικα και η δομή του είναι σχετικά απλή.

Ενότητα <doctype>, <html> και <head>

```
1  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
2  <html xmlns="http://www.w3.org/1999/xhtml" xmlns:og="http://ogp.me/ns#" xmlns:fb="http://www.facebook.com/2008/fbml" lang='el'>
3
4  <head>
5
6  <meta http-equiv='Content-Type' content='text/html; charset=utf-8' />
7  <meta name="viewport" content="width=750, user scalable=yes" />
8  <link href="images/favicon.png" rel="icon" type="image/png" />
9  <title>MineSweeper by Chris | v4.1</title>
10
11 <link href='common.css' rel='stylesheet' type='text/css' />
12
13 </head>
```

Πρώτα ορίζεται το *DOCTYPE*, ο HTML τύπος και η γλώσσα της σελίδας, δηλώσεις που χρειάζεται και χρησιμοποιεί ο περιηγητής (browser) για τη σωστή εμφάνιση των ελληνικών κυρίως.

Στη συνέχεια ορίζεται η ενότητα *HEAD*.

Εδώ ορίζουμε ότι οι χαρακτήρες που θα εμφανίζονται από τον περιηγητή θα είναι τύπου UTF-8, δήλωση απαραίτητη για τη σωστή εμφάνιση των ελληνικών γραμμάτων καθώς ορίζει τη χρήση 8bit χαρακτήρων.

Στη συνέχεια ορίζουμε το *viewport* ή αλλιώς το *παράθυρο προβολής* του περιηγητή, δήλωση που αναφέρεται κυρίως στην εμφάνιση σε κινητά, ταμπλέτες κλπ, οι οποίες δε διαθέτουν το μέγεθος της οθόνης ενός σταθερού υπολογιστή. Εδώ θέτουμε το όριο της οθόνης για το οποίο σχεδιαστικά έχει στηθεί η σελίδα και ορίζουμε ότι επιτρέπουμε στον περιηγητή τη λειτουργία μεγέθυνσης (zoom) από τον χρήστη.

Μετά ορίζουμε στο περιηγητή το εικονίδιο το οποίο θα εμφανίζει στο αριστερό μέρος της καρτέλας που φιλοξενεί το παιχνίδι ενώ μέσω της δήλωσης *<title>…</title>* ορίζουμε τον τίτλο που θα φαίνεται δίπλα στο εικονίδιο αυτό, καθώς και στο κουμπί του περιηγητή που υπάρχει στη μπάρα εργασιών.

Στο τέλος της ενότητας HEAD μέσω της δήλωσης *<link … />* καλούμε το στοιχείο **common.css** να ενσωματωθεί στον περιηγητή.

Ενότητα <body>

```
<body>

<div id="backShow-1" class="backShow"></div>
<div id="backShow-2" class="backShow"></div>
<div id="backShow-3" class="backShow"></div>
<div id="backShow-4" class="backShow"></div>
<div id="backShow-5" class="backShow"></div>
<div id="backShow-6" class="backShow"></div>
<div id="backShow-7" class="backShow"></div>
<div id="backShow-8" class="backShow"></div>
<div id="backShow-9" class="backShow"></div>

<div id="mainPage"><div id="mainPageInner">
  <table id="mainPageTable" cellpadding="0px" cellspacing="0px" align="center">
    <tr>
      <td id="mainPageLeftTD" class="mainPageCols"><div id="mainDiv">
        <div id="centerStart" class="centerDiv" align="center">
          <table id="gameLevels" cellpadding='0px' cellspacing='0px'>
        </div>
        <div id="centerGame" class="centerDiv" align="center">
          <div id='gameStatsDiv'>
            <table id='gameStatsDivInner' cellpadding='0px' cellspacing='0px'><tr>
          </div>
          <table id='gamePad' cellpadding="0px" cellspacing="0px"></table>
        </div>
      </div></td>
      <td id="mainPageRightTD" class="mainPageCols">
        <div id="currentGameDiv" class="centerDiv">
        <div id="statisticsDiv" class="centerDiv">
      </td>
    </tr>
    <tr><td colspan="2">
      <div id="copyrightDiv">'Εκδοση παιχνιδιού 4.1</div>
    </td></tr>
  </table>
</div></div>

<script type="text/javascript" src="game.js"></script>
<script>
  window.onload=function(){
    startUp();
  };
</script>

</body>
```

Στην ενότητα αυτή περιγράφεται η κύρια δομή της εφαρμογής.

Στοιχεία backShow-1 μέχρι 9

Στην αρχή υπάρχουν εννέα στοιχεία * * ***backShow-1*** μέχρι ***9***. Τα στοιχεία αυτά χρησιμοποιούνται για να φιλοξενήσουν τις εικόνες για το χρώμα που βλέπει ο παίκτης να εναλλάσσεται στο υπόβαθρο (background) του παιχνιδιού.

Στοιχεία mainPage και mainPageInner

Στη συνέχεια υπάρχουν δύο εξωτερικά στοιχεία * * **mainPage** και **mainPageInner**, τα οποία αποτελούν το περιτύλιγμα θα μπορούσαμε να πούμε σχεδιαστικά της εφαρμογής.

Στοιχείο mainPageTable

Στο εσωτερικό των δύο αυτών * * υπάρχει ο πίνακας **mainPageTable**, ο οποίος στην πρώτη του γραμμή περιέχει δύο στήλες, τη **mainPageLeftTD** και τη **mainPageRightTD**, ενώ στη δεύτερη γραμμή του περιέχει το στοιχείο * * **copyrightDiv**.

Ας αναλύσουμε το εσωτερικό περιεχόμενο των δύο κύριων στηλών που αποτελούν τον κορμό της εμφάνισης της εφαρμογής.

Στήλη mainPageLeftTD

Η στήλη **mainPageLeftTD** περιέχει δύο στοιχεία * *, το **centerStart** και το **centerGame** (περιτυλιγμένα από το στοιχείο * * **mainDiv** καθαρά και μόνο για λόγους εικαστικής λειτουργικότητας μέσω *css*).

Το μεν πρώτο στοιχείο περιέχει την αρχική οθόνη, ενώ το δεύτερο την οθόνη του παιχνιδιού. Το πρόβλημα είναι τώρα, και τα δύο στοιχεία εμφανίζονται συγχρόνως στην οθόνη; Η απάντηση είναι όχι!

Το ιδιαίτερο χαρακτηριστικό αυτών των δύο στοιχείων είναι πως στη πραγματικότητα, μόνο το ένα από τα δύο είναι ορατό κάθε φορά στην οθόνη του παίκτη, όπως φαίνεται παρακάτω και από τον debugger:

Εμφανίζοντας έτσι το ένα στοιχείο και αποκρύπτοντας κάθε φορά το άλλο, μεταβαίνουμε από την αρχική οθόνη στην οθόνη του παιχνιδιού, χωρίς να χρειάζεται να ξαναφορτώνουμε το αρχείο *index.html* και να χάνουμε τα δεδομένα που υπάρχουν στην μνήμη του περιηγητή από την JavaScript, ενώ σαφώς ο κώδικας είναι ταχύτερος στην εκτέλεσή του και το παιχνίδι αντιδρά γρηγορότερα στις επιλογές του χρήστη.

Στοιχείο centerStart

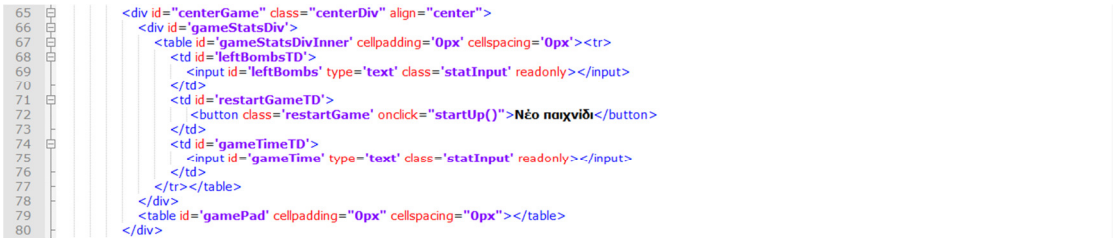
```
<div id="centerStart" class="centerDiv" align="center">
  <table id="gameLevels" cellpadding='0px' cellspacing='0px'>
    <tr><td colspan="4" class="modeTD">
      <button class='restartGame' onclick="startGame(9, 9, 10)"><b>Εύκολο επίπεδο</b><br/>9X9 με 10 βόμβες (Δ 25)</button>
    </td></tr>
    <tr><td colspan="4" class="modeTD">
      <button class='restartGame' onclick="startGame(16, 16, 40)"><b>Κανονικό επίπεδο</b><br/>16X16 με 40 βόμβες (Δ 31)</button>
    </td></tr>
    <tr><td colspan="4" class="modeTD">
      <button class='restartGame' onclick="startGame(30, 16, 99)"><b>Δύσκολο επίπεδο</b><br/>30X16 με 99 βόμβες (Δ 41)</button>
    </td></tr>
    <tr>
      <td class="customModeTD" width="20%" style="padding-left: 2%;">
        <p class="customLabel customInputLabel">Στήλες</p>
        <input id='gameCols' type="number" class="customModeInput" min="5" max="50" onchange="updateLevel()"></input>
      </td>
      <td class="customModeTD" width="20%">
        <p class="customLabel customInputLabel">Γραμμές</p>
        <input id='gameRows' type="number" class="customModeInput" min="5" max="50" onchange="updateLevel()"></input>
      </td>
      <td class="customModeTD" width="20%">
        <p class="customLabel customInputLabel">Βόμβες</p>
        <input id='gameBombs' type="number" class="customModeInput" min="2" onchange="updateLevel()"></input>
      </td>
      <td class="customModeTD">
        <p class="customLabel customInputLabel">Δυσκολία</p>
        <input id='gameLevel' class='customModeInput' readonly></input>
      </td>
    </tr>
    <tr><td colspan="4" class="modeTD">
      <button class='restartGame' onclick="startGame(0, 0, 0)"><b>Εξατομικευμένο επίπεδο</b></button>
    </td></tr>
  </table>
</div>
```

Αναλύοντας τώρα το στοιχείο **centerStart** που όπως αναφέρθηκε φιλοξενεί τα στοιχεία της αρχικής οθόνης, βλέπουμε ότι περιλαμβάνει έναν πίνακα (*table*), τον **gameLevels** ο οποίος στις τρεις πρώτες γραμμές του περιέχει από ένα κουμπί (*button*), μέσω του οποίου ξεκινάει ένα νέο παιχνίδι, με διαφορετικό προεπιλεγμένο επίπεδο δυσκολίας. Η τέταρτη γραμμή του περιλαμβάνει τέσσερα πεδία (*input*), στα οποία τρία πρώτα ο χρήστης καλείται να επιλέξει τα χαρακτηριστικά του παιχνιδιού (στήλες, γραμμές και βόμβες), ενώ στο τέταρτο θα εμφανιστεί το επίπεδο δυσκολίας. Τέλος στην πέμπτη γραμμή υπάρχει το ανάλογο κουμπί για την έναρξη του εξατομικευμένου αυτού παιχνιδιού.

Βλέπουμε εδώ ότι στα τέσσερα κουμπιά έναρξης του παιχνιδιού, υπάρχει η κλήση της ρουτίνας **startGame()** στο συμβάν **onclick**. Αυτό σημαίνει ότι μόλις ο χρήστη κάνει κλικ (πατήσει) κάποιο από τα κουμπιά αυτά, θα εκτελεστεί η ρουτίνα αυτή. Αναλυτικά στοιχεία για τις παραμέτρους και τη λειτουργία της ρουτίνας **startGame()** θα δούμε παρακάτω, στην ανάλυση του JavaScript κώδικα της εφαρμογής.

Αντίστοιχα, στα πεδία χαρακτηριστικών υπάρχει η κλήση της ρουτίνας **updateLevel()** στο συμβάν όμως **onchange**. Η διαφορά είναι τώρα ότι η εκτέλεση της ρουτίνας θα πραγματοποιείται κάθε φορά που το περιεχόμενο του πεδίου αλλάζει. Αυτό θα συμβεί όταν γράψει ή σβήσει κάτι ο χρήστης μέσα στο πεδίο και στη συνέχεια μεταβεί έξω από το πεδίο (είτε με το πλήκτρο *tab* στο πληκτρολόγιο είτε με κλικ του ποντικιού), ή πατήσει τα κουμπάκια που αυξομειώνουν το αριθμό που περιέχεται στο πεδίο.

Στοιχείο centerGame



Συνεχίζοντας τη βαθύτερη ανάλυση των στοιχείων προχωράμε στο **centerGame** που όπως αναφέρθηκε φιλοξενεί τα στοιχεία της οθόνης του παιχνιδιού.

Βλέπουμε την ύπαρξη ενός πίνακα (*table*), του **gameStatsDivInner**, ο οποίος διαθέτει ως περιτύλιγμα το στοιχείο * * **gameStatsDiv**. Ο πίνακας αυτός διαθέτει τρεις στήλες, όπου στην πρώτη και στην τρίτη φιλοξενούνται από ένα πεδίο (*input*), στα οποία ο χρήστης δεν μπορεί να γράψει κάτι μέσα (έχουν ενεργό το χαρακτηριστικό *readonly*) και στη δεύτερη στήλη φιλοξενεί ένα κουμπί (*button*), το οποίο όπως και πριν, θα εκτελέσει μία ρουτίνα, την **startUp()**, μόλις ενεργοποιηθεί (*fire*) το συμβάν **onclick** (ο χρήστης πατήσει το κουμπί με το ποντίκι του).

Στο κάτω μέρος του στοιχείου **centerGame** βλέπουμε την ύπαρξη ενός άδειου (!) πίνακα, του **gamePad**. Ο πίνακας αυτός δεν περιέχει γραμμές και στήλες (όπως θα έπρεπε), γιατί απλά θα γεμίσει αργότερα μέσω του κώδικα JavaScript και συγκεκριμένα όταν θα εκτελεστεί η ρουτίνα **fillTable()**, η οποία επίσης περιέχεται και υλοποιείται στο στοιχείο **game.js**.

Στήλη mainPageRightTD

Στη συνέχεια ας δούμε τα περιεχόμενα της στήλης **mainPageRightTD** η οποία φιλοξενεί τα στοιχεία που βρίσκονται στη δεξιά μεριά της οθόνης του παιχνιδιού.

Βλέπουμε λοιπόν ότι και αυτή η στήλη περιέχει δύο στοιχεία * *, το **currentGameDiv** και το **statisticsDiv**.

Μέσω του debugger βλέπουμε ότι και πάλι γίνεται κάτι αντίστοιχο (απόκρυψη του ενός) με τα δύο αυτά στοιχεία:

Η διαφορά είναι τώρα οτι το στοιχείο **currentGameDiv** αποκρύπτεται στην αρχική οθόνη και εμφανίζεται μόνο στην οθόνη του παιχνιδιού, καθώς φιλοξενεί τα πληροφοριακά στοιχεία του τρέχοντος παιχνιδιού.

Στοιχείο currentGameDiv

```
83      <div id="currentGameDiv" class="centerDiv">
84        <p class="customLabel" style="text-align:center;">Τρέχων παιχνίδι</p>
85        <hr><br>
86        <p id="currentGameInfo"></p><br>
87      </div>
```

Το στοιχείο αυτό είναι απλό στη δομή του. Περιέχει δύο παραγράφους *<p>* και μία οριζόντια διαχωριστική γραμμή *<hr>*.

Η δεύτερη παράγραφος, η **currentGameInfo** αν και είναι άδεια αρχικά, θα γεμίσει αργότερα μέσω κώδικα JavaScript με τα πληροφοριακά στοιχεία του τρέχοντος παιχνιδιού, συγκεκριμένα, με την εκτέλεση της ρουτίνας **showGameInfo()**.

Στοιχείο statisticsDiv

```
88      <div id="statisticsDiv" class="centerDiv">
89        <p class="customLabel" style="text-align:center;">Ιστορικό παιχνιδιών</p>
90        <hr><br>
91        <p id="gamesHistory"></p><br>
92        <div style="text-align:center;">
93          <button id='resetHistory' onclick="eraseCookie('ChrisMinesGame')">Διαγραφή ιστορικού</button>
94        </div>
95      </div>
```

Όπως και το προηγούμενο στοιχείο και αυτό περιέχει τις δύο παραγράφους *<p>* με τη διαχωριστική γραμμή *<hr>* στη μέση. Και εδώ η δεύτερη παράγραφος **gamesHistory** αν και είναι αρχικά άδεια, θα γεμίσει αργότερα μέσω κώδικα JavaScript και συγκεκριμένα με την εκτέλεση της ρουτίνας **getStatistics()**.

Το επιπρόσθετο στοιχείο εδώ είναι οτι στο κάτω μέρος φιλοξενείται ένα κουμπί (*button*), το **resetHistory**, το οποίο και αυτό εκτελεί μία ρουτίνα στο συμβάν **onclick**, την **eraseCookie()**.

Στοιχεία <script>

Συνεχίζοντας την ανάλυση βλέπουμε οτι στο τέλος της ενότητας *<body>* υπάρχουν δύο στοιχεία *<script>*.

Το μεν πρώτο ενσωματώνει στον περιηγητή το στοιχείο **game.js** το οποίο όπως αναφέρθηκε και στην ενότητα *Βασικά Λειτουργικά Στοιχεία* είναι ο κώδικας JavaScript που είναι υπεύθυνος για τη λειτουργικότητα και διαδραστικότητα του παιχνιδιού.

Το μεν δεύτερο καλεί τη ρουτίνα (*function*) **startUp()** της JavaScript (που περιέχεται και υλοποιείται στο στοιχείο **game.js** που ενσωματώθηκε από το προηγούμενο *<script>*).

Το ιδιαίτερο στοιχείο αυτής της κλήσης είναι ότι πραγματοποιείται αφού ολοκληρωθεί το φόρτωμα όλων των απαραίτητων στοιχείων *DOM* από τον περιηγητή, συμπεριλαμβανομένων και των εικόνων κλπ. Αυτό επιτυγχάνεται με την κλήση της **startUp()** μέσα στο συμβάν (*event*) **onload()** του στοιχείου **window**.

Μία διαφοροποίηση που θα μπορούσε να υπάρχει είναι τη κλήση της εν λόγω ρουτίνας να γίνει μέσα από το συμβάν **onload()** του στοιχείου **document**. Στην περίπτωση αυτή η ρουτίνα θα εκτελεστεί αμέσως μόλις τα βασικά στοιχεία *DOM* που περιγράφονται στο τρέχων *HTML* είναι διαθέσιμα, άσχετα αν τα υπόλοιπα στοιχεία (εικόνες, css κλπ) έχουν φορτωθεί. Τότε η ρουτίνα θα λειτουργήσει μεν σωστά, στην περίπτωση όμως αργής σύνδεσης στο internet ο περιηγητής μπορεί να μην έχει προλάβει να φορτώσει κάποια γραφικά δεδομένα (εικόνες) ή στοιχεία

μορφοποίησης (css) και μέχρι αυτό να ολοκληρωθεί η σελίδα θα φαίνεται ελλιπής και άσχημη.

Σαφώς, η κλήση της ρουτίνας **startUp()** απλά, έξω από κάθε συμβάν **onload()** μπορεί να αποβεί μοιραία για την εκτέλεσή της! Στην περίπτωση αυτή υπάρχει ο κίνδυνος να ξεκινήσει η εκτέλεσή της πριν ακόμα κάποια *DOM* στοιχεία είναι διαθέσιμα, οπότε και η JavaScript θα καταρρεύσει λόγω σφάλματος και η εκτέλεση του οποιουδήποτε κώδικα από εκεί και μετά θα διακοπεί.

Ανάλυση – COMMON.CSS

Η χρήση του στοιχείου (αρχείου) αυτού είναι πολύ σημαντική για την εμφάνιση των στοιχείων κάθε σελίδας HTML, αλλά κυρίως για την παραμετροποίηση και ομαδοποίηση των δηλώσεων. Επίσης καθιστά ιδιαίτερα εύκολη την επέμβαση στο μέλλον σε αλλαγές στην εμφάνιση.

Στο αρχείο αυτό ορίζουμε τις λεπτομέρειες εμφάνισης κάθε δομικού στοιχείου που περιέχεται στην HTML σελίδα που εμφανίζει ο περιηγητής του επισκέπτη, αναφερόμενοι σε αυτό είτε μέσω ενός συγκεκριμένου χαρακτηριστικού (*tag*) όπως για παράδειγμα το id ενός στοιχείου, είτε της κλάσης (*class*) που ανήκουν κάποια στοιχεία, είτε και τα στοιχεία αυτά καθαυτά όπως για παράδειγμα *table*, *td*, *div*, *html*, *body*, *a* κλπ.

Για κάθε στοιχείο του οποίου ορίζουμε εδώ τις δηλώσεις εμφάνισης, αν δεν χρησιμοποιούσαμε το αρχείο αυτό, θα έπρεπε να τις ορίσουμε στο χαρακτηριστικό *style="…"* του στοιχείου. Αυτό θεωρητικά φαίνεται απλό. Γιατί άλλωστε να γράφουμε τα ορίσματα σε ξεχωριστό αρχείο αφού μπορούμε να τα ενσωματώσουμε μέσα στον ορισμό του κάθε στοιχείου;

Η απάντηση βρίσκεται στην ομαδοποίηση που μπορούμε να έχουμε μέσω του αρχείου αυτού. Ορίζοντας λοιπόν σε κάποια στοιχεία μία συγκεκριμένη κλάση για παράδειγμα, εδώ απλά περιγράφουμε τις ιδιότητες εμφάνισης που θέλουμε να έχει αυτή η κλάση. Αμέσως, κάθε στοιχείο με την κλάση αυτή θα κληρονομήσει τις ιδιότητες που θέσαμε. Το ίδιο μπορούμε να κάνουμε και για κάθε στοιχείο γενικότερα. Έτσι μπορούμε να ορίσουμε γενικότερα πως θέλουμε να εμφανίζονται όλα τα στοιχεία * *, όλοι οι πίνακες *<table>*, αλλά και συνδυασμό των παραπάνω, για παράδειγμα όλα τα στοιχεία με συγκεκριμένη κλάση.

Έτσι, χωρίς τη χρήση του αρχείου css, σε μία εφαρμογή που εμφανίζει εκατοντάδες στοιχεία DOM στον περιηγητή του επισκέπτη, θα ήταν αδύνατον να μπούμε και να κάνουμε την παραμικρή αλλαγή, πολύ δε μάλλον να δοκιμάσουμε να δούμε πως θα φαίνεται μία αλλαγή, καθώς θα έπρεπε να ανατρέξουμε και να διορθώσουμε χιλιάδες γραμμές κώδικα HTML με το χέρι. Με τη χρήση όμως του css, χρειάζεται απλά να μπούμε και να αλλάξουμε μία και μόνο δήλωση και να επηρεάσουμε αμέσως την εμφάνιση ολόκληρης της εφαρμογής.

Αναλυτικότερα τώρα, στην αρχή του αρχείου αυτού βλέπουμε τις δηλώσεις που αφορούν γενικά κάποια στοιχεία DOM.

```css
html, body {
    height:100%;
    padding: 0px;
    margin: 0px;
    font-family: 'Verdana';
    font-size: 15px;
    line-height: 18px;
    color: #000;
    background-color: #fff;
}

tr, td, th,
p,
a, a img,
h1, h2, h3, h4,
ul, ol,
img {
    padding: 0px;
    margin: 0px;
    border: 0 none;
    font-family: 'Verdana';
    text-decoration: none;
}

ul, ol {
    padding-left: 20px;
}

a:link, a:visited, a:active {
    color: #000;
    text-decoration: none;
}

a:hover {
    color: #f00;
    text-decoration: none;
}
```

Έχουμε λοιπόν αρχικά τις δηλώσεις για τις ενότητες *html* και *body* οι οποίες επηρεάζουν καθολικά την εμφάνιση της εφαρμογής.

Στη συνέχεια υπάρχουν οι δηλώσεις για συγκεκριμένα στοιχεία DOM όπως οι γραμμές *<tr>*, τα κελιά *<td>*, οι παράγραφοι *<p>*, οι εικόνες *<img>*, οι λίστες *<ul>* και *<ol>* κλπ.

Τέλος ορίζεται ο τρόπος εμφάνισης των δεσμών (links) *<a>* γενικά (*link, active*), όταν τους έχουμε ήδη επισκεφτεί (*visited*) ή όταν το ποντίκι βρίσκεται επάνω τους (*hover*).

Μέσα από τις δηλώσεις css βλέπουμε οτι ορίζουμε το ύψος (*height*) που θέλουμε να έχει το κάθε στοιχείο, το εσωτερικό κενό (*padding*), την εξωτερική απόσταση από τα γειτονικά στοιχεία (*margin*), το ύψος της γραμμής (*line-height*), τη γραμματοσειρά (*font-family*), το χρώμα των γραμμάτων (*color*), το μέγεθός τους (*font-size*), το χρώμα του υπόβαθρου (*background-color*), την εικόνα του υπόβαθρου (*backgroung:url*) καθώς και τον τρόπο εμφάνισής της (*background-size*), την ορατότητα ή όχι (*display*), τη διαφάνεια (*opacity*), αλλά και κάθε άλλη λεπτομέρεια της εμφάνισής του.

Στο υπόλοιπο αρχείο ακολουθούν δηλώσεις για κάποια συγκεκριμένα στοιχεία (**mainDiv**, **centerStart**, **gameLevels**, **centerGame**, **gameStatsDiv** κλπ) και δηλώσεις που αφορούν συγκεκριμένες κλάσεις (**centerDiv**, **restartGame**, **customLabel**, **statInput** κλπ).

Ένα ιδιαίτερο χαρακτηριστικό που αξίζει να αναλυθεί είναι το *transition*. Το χαρακτηριστικό αυτό ορίζεται στη κλάση **backShow**:

Με τη χρήση του, ορίζουμε το είδος του χαρακτηριστικού και το χρόνο μετάβασης από μία τιμή του σε μία άλλη.

Στην προκειμένη περίπτωση, έχουμε δηλώσει ότι όταν σε ένα στοιχείο με κλάση **backShow** αλλάξει η διαφάνεια (*opacity*), αυτό θέλουμε να γίνει σταδιακά μέσα σε δέκα δευτερόλεπτα.

Στη συνέχεια ορίζουμε ότι ένα στοιχείο με τις κλάσεις **backShow** και **visible** έχει τα ίδια χαρακτηριστικά (λόγω του ότι περιέχει την κλάση **backShow**), αλλάζει όμως η διαφάνειά του (από 0 σε 1).

Όταν λοιπόν σε ένα στοιχείο του ορίσουμε την κλάση **backShow**, το στοιχείο αυτό θα κρυφτεί (καθώς η διαφάνειά του είναι 0). Μόλις όμως του προσθέσουμε και την κλάση **visible**, το στοιχείο αυτό θα αρχίσει σταδιακά, μέσα σε δέκα δευτερόλεπτα να εμφανίζεται, να αλλάζει δηλαδή η τιμή της διαφάνειάς του από 0 που ήταν, μέχρι τελικά το 1.

Το εφέ αυτό χρησιμοποιείται στα στοιχεία **backShow-1** μέχρι **9**,

όπου προσθέτοντας και αφαιρώντας την κλάση **visible**, τα προκαλούμε να σβήνουν και να εμφανίζονται με τυχαία σειρά.

Καθώς λοιπόν κάθε στοιχείο **backShow-**x έχει μία εικόνα στο υπόβαθρό του μέσω της δήλωσης *background:url*, όταν αλλάζει σταδιακά η διαφάνειά του εμφανίζεται η εικόνα και την ίδια στιγμή κάποιο άλλο κρύβεται και έτσι να πραγματοποιείται η αλλαγή του χρώματος του υπόβαθρου της εφαρμογής.

Η διαδικασία αυτή υλοποιείται στη μέσα στη JavaScript ρουτίνα **startUp()** της οποίας την ανάλυση θα δούμε στην επόμενη ενότητα.

Ανάλυση – GAME.JS

Στο αρχείο αυτό περιέχεται όλος ο κώδικας JavaScript.

Όπως αναφέρθηκε και στην ενότητα *Ανάλυση – INDEX.HTML* και συγκεκριμένα στην ανάλυση *Στοιχεία <script>*, το αρχείο αυτό ενσωματώνεται στο αρχείο index.html μέσω ενός στοιχείου *<script>* στο κάτω μέρος του.

Καθολικές μεταβλητές

Αμέσως, με την ενσωμάτωσή του, εκτελούνται οι πρώτες δηλώσεις του κώδικα:

```
var gameCols = 25;
var gameRows = 13;
var gameBombs = 50;
var gameFlags = 0;
var startTime = new Date();
var gameTime = null;
var nowBack = 0;
var historyArray = [];
var gameIsOn = false;

var cellDefaultClass = 'gameCell linkItem';

document.getElementById('gameCols').value = gameCols;
document.getElementById('gameRows').value = gameRows;
document.getElementById('gameBombs').value = gameBombs;
updateLevel();
```

Στο πάνω μέρος έχουμε τη δήλωση των καθολικών (global) μεταβλητών, οι οποίες ισχύουν παντού, σε όλον τον κώδικα και για όση ώρα η τρέχουσα σελίδα είναι φορτωμένη στον περιηγητή.

Αναλυτικά δηλώνονται οι παρακάτω μεταβλητές:

- **gameCols**: οι κάθετες στήλες του παιχνιδιού,
- **gameRows**: οι οριζόντιες γραμμές του παιχνιδιού,
- **gameBombs**: οι βόμβες που καλείται να βρει ο παίκτης,
- **gameFlags**: τα μαρκαρισμένα ως βόμβες κελιά,
- **startTime**: η ώρα έναρξης του παιχνιδιού,
- **gameTime**: ο δείκτης της επαναλαμβανόμενης ρουτίνας εμφάνισης του χρόνου του παιχνιδιού,
- **nowBack**: ο αριθμός του στοιχείου **backShow-**x το οποίο είναι αυτή τη στιγμή ορατό,
- **historyArray**: ο πίνακας με τα δεδομένα του ιστορικού,
- **gameIsOn**: δείκτης (flag) με την τρέχουσα κατάσταση του παιχνιδιού (ενεργό ή τελείωσε),
- **cellDefaultClass**: λεκτικό με τη προδηλωμένη (default) τιμή κλάσεων κάθε κελιού του παιχνιδιού.

Αρχικοποίηση στοιχείων οθόνης

Μόλις ολοκληρωθεί η φόρτωση του αρχείου *index.html* και στη συνέχεια μέσω του πρώτου στοιχείου *<script>* ενσωματωθεί και το αρχείο *game.js*, εκτελούνται τέσσερις εντολές.

```
13  document.getElementById('gameCols').value = gameCols;
14  document.getElementById('gameRows').value = gameRows;
15  document.getElementById('gameBombs').value = gameBombs;
16  updateLevel();
```

Οι τρεις πρώτες κάνουν σε πρώτο στάδιο χρήση της μεθόδου **getElementById()** του αντικειμένου **document**.

Η μέθοδος αυτή αναζητά και επιστρέφει το πρώτο DOM αντικείμενο που έχει το χαρακτηριστικό *id*, με τιμή το λεκτικό του ορίσματος της μεθόδου. Εάν δεν βρεθεί κάποιο αντικείμενο η μέθοδος επιστρέφει την τιμή *null*.

Είναι ίσως η βασικότερη σε χρήση μέθοδος αναζήτησης στοιχείων στο τρέχων HTML έγγραφο.

Η πρώτη λοιπόν εντολή, αναζητά και επιστρέφει το αντικείμενο με *ID* “gameCols”, το οποίο είναι ένα στοιχείο τύπου *<input>* και βρίσκεται στο πρώτο κελί της τέταρτης γραμμής του πίνακα **gameLevels**. Αντίστοιχα, η δεύτερη εντολή αναζητά και επιστρέφει το αντικείμενο με *id* “gameRows” και η τρίτη το αντικείμενο με *id* “gameBombs”.

Στη συνέχεια, η κάθε μία εντολή ορίζει την ιδιότητα *value* κάθε αντικειμένου που έχει επιστραφεί μέσω της **getElementById()** και του θέτει την τιμή της ανάλογης καθολικής μεταβλητής.

Ο λόγος εκτέλεσης των τριών αυτών εντολών είναι η αρχικοποίηση των πεδίων εξατομικευμένου επιπέδου με την τιμή της ανάλογης μεταβλητής. Αυτό σημαίνει ότι με τον ορισμό τιμής στην ιδιότητα *value* του κάθε αντικειμένου, το αντίστοιχο στοιχείο HTML θα εμφανίσει την τιμή αυτή.

Η τέταρτη εντολή που εκτελείται είναι η κλήση της ρουτίνας **updateLevel()** η οποία είναι υπεύθυνη να ενημερώσει την ιδιότητα *value* του στοιχείου με *id* **gameLevel** και την ανάλυση της οποίας θα δούμε λίγο παρακάτω.

Ρουτίνα startUp()

Η πρώτη ρουτίνα που εκτελείται μετά την ολοκλήρωση φόρτωσης του *index.html*, μέσω του δεύτερου στοιχείου *<script>*.

```
function startUp() {
    getStatistics();
    document.getElementById('centerGame').style.display = 'none';
    document.getElementById('centerStart').style.display = '';
    document.getElementById('resetHistory').style.display = '';
    document.getElementById('currentGameDiv').style.display = 'none';

    nowBack = Math.floor((Math.random() * 9) + 1);
    document.getElementById('backShow-' + nowBack).className = 'backShow visible';
    setInterval(function(){
        document.getElementById('backShow-' + nowBack).className = 'backShow';
        do {
            newBack = Math.floor((Math.random() * 9) + 1);
        } while (newBack == nowBack);
        nowBack = newBack;
        document.getElementById('backShow-' + nowBack).className = 'backShow visible';
    }, 12000);
}
```

Η ρουτίνα αυτή δε δέχεται ορίσματα στην κλήση της.

Στη πρώτη της σειρά καλεί τη ρουτίνα **getStatistics()** που είναι υπεύθυνη για την εμφάνιση του ιστορικού των παιχνιδιών.

Στη συνέχεια μέσω τεσσάρων κλήσεων της **getElementById()** παίρνουμε τέσσερα αντικείμενα που αναφέρονται σε αντίστοιχα DOM στοιχεία.

Σε κάθε ένα καλείται το αντικείμενο *style* το οποίο είναι υπεύθυνο για τη διαχείριση του χαρακτηριστικού *style="…"* κάθε DOM στοιχείου. Στο αντικείμενο αυτό δίνουμε τιμή στην ιδιότητα *display*, η οποία ορίζει την εμφάνιση ή όχι κάθε στοιχείου. Η ιδιότητα αυτή ορίζεται επίσης και στο αρχείο *common.css* για κάθε στοιχείο, ανάλογα με το *id* ή και την κλάση του (*class*).

Έτσι στο πρώτο στοιχείο με *id* **centerGame** θέτοντας την τιμή *"none"* στην πραγματικότητα το καθιστούμε αόρατο. Πληροφοριακά, υπάρχει και άλλη μία ιδιότητα με την οποία μπορούμε να κάνουμε αόρατο ένα στοιχείο, η *visibility:hidden*. Η διαφορά τους είναι ότι η μεν πρώτη καθιστά εξαφανίζει τελείως το αντικείμενο από το χώρο, δεν καταλαμβάνει δηλαδή καθόλου χώρο (ώστε να επηρεάζει πιθανώς τη θέση ή το μέγεθος γειτονικών στοιχείων), ενώ η δεύτερη το καθιστά απλώς αόρατο στον χρήστη, καταλαμβάνει όμως το χώρο του μέσα στη σελίδα (επηρεάζοντας πιθανώς γειτονικά στοιχεία).

Το ίδιο γίνεται (το κρύβουμε δηλαδή) και με το τέταρτο στοιχείο, με *ID* **currentGameDiv**.

Αντίθετα, στο δεύτερο και τρίτο στοιχείο τους καθαρίζουμε την ιδιότητα *display* με αποτέλεσμα τα στοιχεία αυτά να εμφανιστούν μέσα στη σελίδα.

Με τον τρόπο αυτό, στην πραγματικότητα, κρύβουμε το στοιχείο * * **centerGame** το οποίο φιλοξενεί το κυρίως παιχνίδι, καθώς και το στοιχείο * * **currentGameDiv** το οποίο φιλοξενεί τις πληροφορίες του τρέχοντος παιχνιδιού και εμφανίζουμε το στοιχείο * * **centerStart** το οποί φιλοξενεί την αρχική σελίδα καθώς και το στοιχείο *<button>* **resetHistory** μέσω του οποίου γίνεται καθαρισμός του ιστορικού.

Στις επόμενες δέκα γραμμές, υλοποιείται το εφέ αλλαγής χρωμάτων του υπόβαθρου της σελίδας.

Αρχικά ορίζουμε ένα από τα εννέα στοιχεία * * με *id* **backShow-**x ως το τρέχων ορατό.

Αυτό γίνεται αρχικά με την κλήση της **Math.random()**. Η ρουτίνα αυτή θα επιστρέψει έναν (ψευτο)τυχαίο αριθμό από το μηδέν μέχρι το ένα (αλλά πάντα μικρότερο του ένα). Πολλαπλασιάζοντας τον αριθμό αυτό με το εννέα τον μετατρέπουμε σε έναν αριθμό από το μηδέν μέχρι το εννέα (αλλά ποτέ το εννέα). Προσθέτοντας τώρα το ένα στο τέλος, τον μετατρέπουμε σε κάποιον αριθμό από το ένα μέχρι το δέκα (μικρότερο όμως πάντα του δέκα). Τέλος περικλείουμε όλο αυτό με την ρουτίνα **Math.floor()** η οποία μας επιστρέφει τον κοντινότερο πάντα προς τα κάτω ακέραιο αριθμό ενός δεκαδικού. Τελικά θα έχουμε έναν (ψευτο)τυχαίο ακέραιο αριθμό από το ένα μέχρι το εννέα.

Στη συνέχεια θέτουμε την ιδιότητα *class* του στοιχείου με *id* **backShow-** και τον τυχαίο αριθμό (από το ένα ως το εννέα από πριν), με την τιμή *“backShow visible”*. Εκμεταλλευόμενοι την *css* ιδιότητα *transition* όπως αναλύθηκε στην ενότητα *Ανάλυση – COMMON.CSS* το στοιχείο * * θα αρχίσει σταδιακά να εμφανίζεται.

Στο επόμενο βήμα γίνεται κλήση της ρουτίνας **setInterval()** μέσω της οποίας ορίζουμε την επανάληψη μίας ρουτίνας ανά συγκεκριμένο χρόνο, στην περίπτωσή μας κάθε 12000 milliseconds δηλαδή κάθε δώδεκα δευτερόλεπτα.

Η ρουτίνα που θα επαναλαμβάνεται θα κάνει τα παρακάτω βήματα…

Αρχικά αλλάζει την κλάση του τρέχοντος ενεργού στοιχείου * * **backShow-**x από *“backShow visible”* σε μόνο *“backShow”*. Τώρα, εκμεταλλευόμενοι και πάλι τη *css* ιδιότητα *transition*, το στοιχείο θα αρχίσει σταδιακά να σβήνει (να γίνεται διάφανο).

Εντωμεταξύ, στο επόμενο βήμα μέσω ενός *do..while()* βρόχου αναζητούμε τον επόμενο (ψευδο)τυχαίο αριθμό (ο οποίος δεν πρέπει

να είναι ίδιος με τον τρέχων), που θα ορίσει το στοιχείο * * **backShow**-x το οποίο μέσω της αλλαγής της κλάσης του, το αναγκάζουμε σταδιακά να εμφανίζεται (να γίνεται αδιάφανο).

Έτσι, κάθε δώδεκα δευτερόλεπτα, το ένα στοιχείο γίνεται σταδιακά διάφανο, ενώ το άλλο σταδιακά αδιάφανο, παρουσιάζοντας στον επισκέπτη το εφέ της συνεχούς μετάβασης από ένα χρώμα σε ένα άλλο!

Μετάβαση σε κατάσταση αναμονής συμβάντος

Στο σημείο αυτό η εφαρμογή μπαίνει σε κατάσταση αναμονής κάποιου συμβάντος (*event*) από το χρήστη.

Όπως αναφέρθηκε στην ενότητα *Ανάλυση – INDEX.HTML* κάθε στοιχείο τύπου *<button>* έχει συνδεθεί με κάποιο συμβάν. Συγκεκριμένα τα τέσσερα κουμπιά που ξεκινούν το παιχνίδι (προεπιλεγμένα επίπεδα και εξατομικευμένο) έχουν συνδεθεί με την εκτέλεση της ρουτίνας **startGame()**, τα πεδία επιλογής των χαρακτηριστικών του εξατομικευμένου επιπέδου με την εκτέλεση της ρουτίνας **updateLevel()** και το κουμπί διαγραφής του ιστορικού με την εκτέλεση της ρουτίνας **eraseCookie()**.

Ο κώδικας λοιπόν, θα περιμένει να πατηθεί κάποιο από τα κουμπιά ή να αλλάξουν τα δεδομένα ενός από τα τρία πεδία, ώστε να εκτελεστεί η ανάλογη ρουτίνα. Παράλληλα όμως, όπως αναλύθηκε πιο πριν, ο κώδικας εναλλαγής των εικόνων στο υπόβαθρο σαφώς συνεχίζει να εκτελείται (ανά δώδεκα δευτερόλεπτα).

Βλέπουμε λοιπόν εδώ ένα χαρακτηριστικό παράδειγμα ασύγχρονης εκτέλεσης κώδικα, ο οποίος περιλαμβάνει την εμφάνιση κάποιου στοιχείο, παράλληλα το σβήσιμο κάποιου άλλου, ενώ την ίδια στιγμή περιμένει να αντιδράσει σε συμβάν που θα προκαλέσει ο χρήστης.

Ρουτίνα getStatistics()

Η ρουτίνα καλείται όπως αναφέρθηκε, στην αρχή της **startUp()**.

```
37  function getStatistics() {
38      var historyCookie = readCookie('ChrisMinesGame');
39      if (historyCookie) {
40          historyArray = historyCookie.split('<br>');
41          document.getElementById('gamesHistory').innerHTML = historyCookie;
42      }
43  }
```

Είναι υπεύθυνη να εμφανίσει το ιστορικό των παιχνιδιών.

Το ιστορικό αυτό αποθηκεύεται από τη ρουτίνα **saveStatistics()** σε ένα *cookie* στον περιηγητή του χρήστη.

Μέσω κλήσης της **readCookie()** παίρνει τα περιεχόμενα του *cookie* με το όνομα **ChrisMinesGame** και τα αποθηκεύει σε μία τοπική μεταβλητή.

Αφού γίνει ο απαραίτητος έλεγχος για την ύπαρξη ή όχι ιστορικού, το λεκτικό αυτό χωρίζεται σε γραμμές, όπου αυτές ορίζονται από την ύπαρξη του λεκτικού *
* (*html* αλλαγή γραμμής) και κρατούνται σε μορφή πίνακα στην καθολική μεταβλητή **historyArray**. Αργότερα θα δούμε ότι η χρηστικότητα του πίνακα αυτού είναι η εύκολη προσθήκη νέων παιχνιδιών καθώς και η ανακατάταξή τους.

Τέλος, μέσω της ιδιότητας *innerHTML* μεταφέρονται τα δεδομένα που πήραμε από το *cookie* ως περιεχόμενο του στοιχείου **gamesHistory**.

Ρουτίνα readCookie()

Τα *cookies* είναι διάφορες πληροφορίες σε μορφή κειμένου, που αποθηκεύει ο κάθε περιηγητής ξεχωριστά στον υπολογιστή μας και μέσα τους μπορεί να κρατά πληροφορίες ο κάθε ιστότοπος όπως το όνομα χρήστη που χρησιμοποιήσαμε τελευταία φορά για να πραγματοποιήσουμε είσοδο, προτιμήσεις εμφάνισης του ιστότοπου, προτιμώμενη γλώσσας εφάνισης, τα προϊόντα που επισκεφτήκαμε, σελίδες που είδαμε κλπ.

Το βασικό χαρακτηριστικό των *cookies* είναι ότι κάθε περιηγητής έχει τα δικά του και κάθε ιστότοπος μπορεί να ελέγχει, διαβάζει, μεταβάλει και διαγράφει μόνο τα δικά του και όχι άλλου ιστότοπου ή περιηγητή.

Μέσω του debugger μπορούμε να δούμε τα περιεχόμενα των *cookies* που έχουν δημιουργηθεί από την εφαρμογή στον Firefox συγκεκριμένα. Βλέπουμε ότι είναι σε απλό κείμενο χωρίς καμιά προστασία. Το ότι όμως εμείς μπορούμε να δούμε τα περιεχόμενα οποιουδήποτε ιστότοπου επισκεπτόμαστε, δε σημαίνει (ευτυχώς) ότι μπορεί (έτσι ανεξέλεγκτα) να τα δει και ο κάθε ιστότοπος.

Παρατηρούμε επίσης, ότι η εφαρμογή μας διαθέτει δύο *cookies*. Όταν αυτά θα αναζητηθούνε, η JavaScript θα μας επιστρέψει ένα μεγάλο

λεκτικό, το οποίο θα περιέχει όλα τα *cookies* ενωμένα μεταξύ τους με ένα ελληνικό ερωτηματικό (;). Είναι ευθύνη της εφαρμογής στη συνέχεια ο διαχωρισμός τους και επιστροφή των περιεχομένων του σωστού *cookie*.

Η τρέχουσα ρουτίνα είναι υπεύθυνη για το διάβασμα του σωστού *cookie*.

```
function readCookie(name) {
    var cookieName = name + "=";
    var cookiesArray = document.cookie.split(';');

    for(var i=0; i < cookiesArray.length; i++) {
        var c = cookiesArray[i];
        while (c.charAt(0)==' ') {
            c = c.substring(1, c.length);
        }
        if (c.indexOf(cookieName) == 0) return c.substring(cookieName.length, c.length);
    }
    return null;
}
```

Η ρουτίνα δέχεται ένα όρισμα κατά την κλήση της και το οποίο περιλαμβάνει το όνομα του *cookie* του οποίου αναζητάμε την τιμή του.

Αφού λοιπόν μετατρέψουμε το ζητούμενο όνομα προσθέτοντας στο τέλος τον χαρακτήρα ίσον (=), παίρνουμε μέσω της **document.cookie** ένα μεγάλο λεκτικό, το οποίο στη συνέχεια το χωρίζουμε σε πίνακα, ορίζοντας ως χαρακτήρα διαχωρισμού το ελληνικό ερωτηματικό (;).

Ο πίνακας αυτός θα περιέχει τόσες γραμμές, όσα και τα *cookies* που είναι διαθέσιμα στην εφαρμογή μας από τον περιηγητή.

Στον επόμενο βρόχο παίρνουμε κάθε μία από αυτές τις γραμμές.

Πρώτα καθαρίζουμε τυχών κενά στην αρχή του λεκτικού με χρήση της **charAt(0)** με την οποία αναζητάμε τον πρώτο (κάθε φορά χαρακτήρα) και στη συνέχεια της **substring()** με την οποία επιστρέφουμε το υπόλοιπο λεκτικό.

Στη συνέχεια ελέγχουμε αν το όνομα του *cookie* μαζί με το ίσον (=) υπάρχει στην αρχή του λεκτικού μέσω της **indexOf()**. Αν βρεθεί, η ρουτίνα επιστρέφει το υπόλοιπο λεκτικό, ενώ αν τελικά δε βρεθεί σε καμία γραμμή του πίνακα, η ρουτίνα επιστρέφει την τιμή *null*.

Ρουτίνα startGame()

Η ρουτίνα αυτή είναι μία αυτές που καλούνται μέσω κάποιου συμβάντος από το χρήστη.

Η συγκεκριμένη καλείται όταν ο χρήστης κάνει *click*, πατήσει δηλαδή κάποιο από τα τέσσερα κουμπιά έναρξης παιχνιδιού.

```
59  function startGame(cols, rows, bombs) {
60      if (cols * rows * bombs == 0) {
61          gameCols = parseInt(document.getElementById('gameCols').value);
62          gameRows = parseInt(document.getElementById('gameRows').value);
63          gameBombs = parseInt(document.getElementById('gameBombs').value);
64      } else {
65          gameCols = cols;
66          gameRows = rows;
67          gameBombs = bombs;
68      }
69      if (gameCols < 5 || gameCols > 50) {
70          alert('Ας σοβαρευτούμε!\n\nΟι στήλες μπορούν να είναι από 5 εώς 50...');
71      } else if (gameRows < 5 || gameRows > 50) {
72          alert('Ας σοβαρευτούμε!\n\nΟι γραμμές μπορούν να είναι από 5 εώς 50...');
73      } else if (gameBombs < 2) {
74          alert('Ας σοβαρευτούμε!\n\nΠρέπει να έχετε τουλάχιστον 2 βόμβες...');
75      } else if (getGameLevel(gameCols, gameRows, gameBombs) > 100) {
76          alert('Ας σοβαρευτούμε!\n\nΕίναι πολλές οι βόμβες...');
77      } else {
78          gameFlags = 0;
79          gameIsOn = true;
80
81          fillTable();
82          fillBombs();
83          showGameInfo();
84          startTime = new Date();
85          clearTimeout(gameTime);
86          gameTime=setInterval('displayTheTime()', 1000);
87
88          document.getElementById('centerStart').style.display = 'none';
89          document.getElementById('centerGame').style.display = '';
90          document.getElementById('resetHistory').style.display = 'none';
91          document.getElementById('currentGameDiv').style.display = '';
92      }
93  }
```

Είναι η βασική ρουτίνα ελέγχου, αρχικοποίησης και τελικά έναρξης ενός παιχνιδιού.

Δέχεται τρία ορίσματα, που είναι οι τιμές των σειρών, των γραμμών και των βομβών με τις οποίες θα στηθεί το παιχνίδι.

Στις πρώτες γραμμές της υπάρχει ένας απλός έλεγχος για το αν έχουν σταλεί τα ορίσματα αυτά και δεν είναι κανένα μηδέν. Αν τα ορίσματα είναι σωστά μεταφέρονται στις αντίστοιχες καθολικές μεταβλητές. Αν κάποιο από τα ορίσματα είναι μηδέν, τότε η εφαρμογή διαβάζει τα δεδομένα από τα στοιχεία *<input>* τα οποία φιλοξενούν τις τιμές του εξατομικευμένου επιπέδου. Το διάβασμα κάθε στοιχείου *<input>* πραγματοποιείται σε τρία διαφορετικά βήματα. Πρώτα με κλήση της **getElementById()** παίρνουμε το αντικείμενο του στοιχείου DOM. Στην συνέχεια μέσω της ιδιότητας **value** του αντικειμένου παίρνουμε τα περιεχόμενά του. Τέλος γίνεται κλήση της ρουτίνας **parseInt()** μέσω της οποίας μετατρέπουμε το αλφαριθμητικό αποτέλεσμα της ιδιότητας **value** σε αριθμητικό.

Στη συνέχεια του κώδικα γίνεται ακόμα ένας έλεγχος σχετικά με το ανώτερο και κατώτερο όριο των τιμών αυτών. Αν η τιμή κάποιου ορίσματος είναι εκτός των αποδεκτών ορίων, εμφανίζεται το ανάλογο μήνυμα. Τα όρια βέβαια έχουν τεθεί αυθαίρετα, απλά και μόνο για τον

περιορισμό του μεγέθους του τελικού πίνακα του παιχνιδιού, σε κάτι… λογικό!

Ένας βασικός έλεγχος είναι αυτός που έχει να κάνει με το επίπεδο δυσκολίας του παιχνιδιού. Ο έλεγχος αυτός πραγματοποιείται στη ρουτίνα **getGameLevel()** και ορίζεται με έναν αριθμό από το ένα ως το εκατό. Αναλυτικά θα δούμε παρακάτω πως αυτό υλοποιείται.

Εάν τώρα όλοι οι έλεγχοι είναι επιτυχείς, ο κώδικας ορίζει την καθολική μεταβλητή **gameFlags** που κρατά τις μαρκαρισμένες από το χρήστη βόμβες και ενεργοποιεί τον δείκτη **gameIsOn** ώστε να εκτελούνται πλέον ρουτίνες οι οποίες έχουν να κάνουν με το παιχνίδι την ώρα που παίζεται και μόνο.

Στις επόμενες εντολές καλούνται οι ρουτίνες **fillTable()** η οποία θα γεμίσει στην οθόνη τον πίνακα με τα κελιά – κουμπιά του παιχνιδιού, η **fillBombs()** η οποία θα αλλάξει κάποια από τα κελιά και θα τα ορίσει ως βόμβες και η **showGameInfo()** η οποία θα εμφανίσει τα πληροφοριακά στοιχεία του τρέχοντος παιχνιδιού στο δεξιά επάνω μέρος της οθόνης.

Βλέπουμε μετά τρεις εντολές οι οποίες είναι υπεύθυνες για την εμφάνιση της ώρα που έχει παρέλθει από τη στιγμή έναρξης του παιχνιδιού.

Η πρώτη εντολή κρατά στην καθολική μεταβλητή **startTime** την τρέχουσα ημερομηνία και ώρα σε μορφή αριθμού, ώστε να τον χρησιμοποιήσουμε αργότερα, όταν θα θέλουμε να ελέγξουμε το πόση ώρα έχει παρέλθει από τη στιγμή έναρξης του παιχνιδιού.

Την τρίτη, τη **setInterval()** τη συναντήσαμε και προηγουμένως στη **startUp()**. Η ρουτίνα αυτή είναι υπεύθυνη όπως είδαμε να εκτελεί κάποιον κώδικα ανά τακτά χρονικά διαστήματα.

Στην λοιπόν τρέχουσα περίπτωση, καλείται ανά 1000 milliseconds, δηλαδή κάθε δευτερόλεπτο η ρουτίνα **displayTheTime()**, η οποία όπως θα δούμε και παρακάτω είναι υπεύθυνη να εμφανίζει το χρόνο που έχει παρέλθει στο παιχνίδι.

Το πρόβλημα όμως που παρουσιάζεται είναι πως, κάποια στιγμή, θα πρέπει να μπορούμε, αν θέλουμε, να σταματήσουμε αυτή την επαναλαμβανόμενη εκτέλεση κάποιας ρουτίνας.

Η λύση στο πρόβλημα αυτό είναι να κρατήσουμε σε μία μεταβλητή (εδώ την αποθηκεύουμε στην καθολική μεταβλητή **gameTime**) το δείκτη που επιστρέφει η κλήση της **setInterval()** και στη συνέχεια να τον χρησιμοποιήσουμε ως όρισμα σε κλήση της **clearTimeout()**.

Αυτό γίνεται λοιπόν στη δεύτερη από τις δύο αυτές εντολές.

Αφού λοιπόν καθαρίζουμε (τερματίσουμε) κάποια πιθανώς ενεργή επαναλαμβανόμενη εκτέλεση, στη συνέχεια, με κλήση της **setInterval()** δημιουργούμε τη νέα.

Τέλος, όπως κάναμε αντίστοιχα και στη **startUp()**, θέτοντας την τιμή της *css* ιδιότητας *display* σε "*none*", κρύβουμε τώρα τα στοιχεία <*div*> **centerStart** (που φιλοξενεί την αρχική σελίδα) και το <*button*> **resetHistory** (με το οποίο καθαρίζουμε το ιστορικό) και στη συνέχεια θέτοντας την τιμή της ιδιότητας σε "" (άδεια – κενή τιμή) εμφανίζουμε τα στοιχεία <*div*> **centerGame** (που φιλοξενεί το κυρίως παιχνίδι) και το **currentGameDiv** (που φιλοξενεί τα πληροφοριακά στοιχεία του τρέχοντος παιχνιδιού).

Ρουτίνα fillTable()

Η ρουτίνα αυτή καλείται μέσα από την **startGame()** και αναλαμβάνει να γεμίσει με γραμμές και κελιά τον πίνακα (*table*) **gamepad**.

```
function fillTable() {
    var theTable = document.getElementById('gamePad');
    theTable.innerHTML = '';

    var cellIndex = 0;
    for (var theRow = 1; theRow <= gameRows; theRow++) {
        var thisRow = theTable.insertRow(-1);
        for (var theCol = 1; theCol <= gameCols; theCol++) {
            var thisCell = thisRow.insertCell(-1);
            thisCell.addEventListener('click', function(ev) {
                ev.preventDefault();
                leftClickOnCell(this);
            });
            thisCell.addEventListener('contextmenu', function(ev) {
                ev.preventDefault();
                rightClickOnCell(this);
            });
            thisCell.setAttribute('id', 'cell-' + theRow + 'x' + theCol);
            thisCell.setAttribute('alt', 'cellIndex-' + ++cellIndex);
            thisCell.className = cellDefaultClass;
            thisCell.innerHTML = '<img' +
                ' src="images/normal.jpg"' +
                ' class="cellImage normalImage"' +
                ' />';
        }
    }
}
```

Στην αρχή της κρατάμε στην τοπική μεταβλητή **theTable** το αντικείμενο του στοιχείου * * **gamePad** μέσω της **getElementById()**.

Στη συνέχεια καθαρίζουμε τα περιεχόμενα του πίνακα (*table*) αυτού, σβήνοντας όλες τις ήδη υπάρχουσες γραμμές και κελιά, θέτοντας ως κενή (“”) την ιδιότητα **innerHTML** του αντικειμένου αυτού.

Στο επόμενο κομμάτι κώδικα εκτελούνται δύο βρόχοι. Ο ένας μετρώντας τις γραμμές που έχει δηλώσει ο χρήστη ότι θέλει να περιέχει το παιχνίδι του και ο δεύτερος τις στήλες.

Για κάθε γραμμή εκτελείται στο αντικείμενο **theTable** η μέθοδος **insertRow()** μέσω της οποίας ένα νέο στοιχείο *<tr>* γραμμής πίνακα προστίθεται στις ήδη υπάρχουσες και στη συνέχεια κρατάμε το αντικείμενο της νέας αυτής γραμμής στην τοπική (μέσα στο βρόχο) μεταβλητή **thisRow**.

Για κάθε κελί τώρα εκτελείται η μέθοδος **insertCell()** στο αντικείμενο της γραμμής που ήδη προστέθηκε, μέσω της οποίας σαφώς προστίθεται ένα νέο στοιχείο *<td>* κελί γραμμής (ή στήλη πίνακα απλά) στα ήδη υπάρχοντα της γραμμής και φυσικά κρατάμε και εδώ σε μία πάλι τοπική (μέσα στο βρόχο) μεταβλητή, την **thisCell** το νέο αυτό αντικείμενο.

Σε αυτό το αντικείμενο κελιού (και σε κάθε επόμενο φυσικά) πραγματοποιούνται κάποιες ρυθμίσεις – αλλαγές.

Πρώτα μέσω κλήσης της μεθόδου **addEventListener()** ορίζουμε ότι το κελί αυτό θα αντιδρά σε κάποιο συμβάν που προκαλείται από το χρήστη.

Στην πρώτη χρήση της μεθόδου αυτής, ορίζουμε ότι θα αντιδρά στο πάτημα (*click*) του κελιού αυτού και στη δεύτερη στο πάτημα με το δεξί πλήκτρο του ποντικιού ή το παρατεταμένο πάτημα στην οθόνη του κινητού τηλεφώνου (*contextmenu*).

Και στις δύο περιπτώσεις βλέπουμε ότι θα εκτελεστεί μία ρουτίνα με ένα όρισμα, το **ev**. Στο όρισμα αυτό η JavaScript θα βάλει το αντικείμενο του συμβάντος (*event*). Το πρώτο που έχουμε να κάνουμε είναι να ενημερώσουμε τη JavaScript μέσω της **preventDefault()** ότι δε θέλουμε να εκτελέσει τον οποιονδήποτε κώδικα είναι προεπιλεγμένος να εκτελεστεί (για παράδειγμα να εμφανίσει το υπομενού που βλέπουμε κάνοντας δεξί κλικ σε κάποιο σημείο μίας σελίδας).

Στη συνέχεια ορίζουμε ότι θέλουμε να εκτελεστεί κάποια συγκεκριμένη ρουτίνα. Στη μεν πρώτη περίπτωση (στο πάτημα – *click* του κελιού αυτού) η **leftClickOnCell()** και στη δεύτερη (στο πάτημα με το δεξί πλήκτρο του ποντικιού) η **rightClickOnCell()**.

Και στις δύο ρουτίνες στέλνουμε ως όρισμα το αντικείμενο **this**, στο οποίο η JavaScript θα μεταφέρει εκείνη τη στιγμή της κλήσης, το αντικείμενο του στοιχείου που προκάλεσε το συμβάν (*event*) αυτό, δηλαδή το κελί στο οποίο έγινε το *click* ή το δεξί *click* στην περίπτωσή μας.

Οι επόμενες κλήσεις μεθόδων είναι απλές και έχουν να κάνουν με τον ορισμό των χαρακτηριστικών του κελιού.

Μέσω της **setAttribute()** ορίζουμε τα χαρακτηριστικά *id* και *alt* του κάθε στοιχείου. Τα δύο αυτά χαρακτηριστικά κρατάνε το μεν *id* τις συντεταγμένες του κελιού σε μορφή *cell-*$^{ΑριθμόςΓραμμής}x^{ΑριθμόςΣτήλης}$ ενώ το *alt* τον αύξοντα αριθμό του κελιού τον οποίο παίρνει από τη μεταβλητή **cellIndex**, η οποία αυξάνεται αυτόματα σε κάθε εκτέλεση του βρόχου. Τις πληροφορίες αυτές θα τις χρησιμοποιήσουμε αργότερα στο παιχνίδι για την εύκολη αναζήτηση του κάθε κελιού.

Μέσω της **className()** ορίζουμε την κλάση (*class*) του στοιχείου. Για να αποφύγουμε την πιθανότητα λάθους κατά την εγγραφή του κώδικα, αλλά και για να μπορούμε να ελέγξουμε αργότερα ευκολότερα τα περιεχόμενα της κλάσης του κάθε στοιχείου, έχουμε κρατήσει το λεκτικό με την προδηλωμένη (*default*) τιμή της κλάσης, στην καθολική (*global*) μεταβλητή **cellDefaultClass**.

Τέλος μέσω της ιδιότητας **innerHTML** προσθέτουμε στο εσωτερικό του κελιού μία εικόνα που εμφανίζει το κρυμμένο – κλειστό κελί (αρχικά) στον παίκτη.

Η προσθήκη αυτή γίνεται με χτίσιμο κώδικα HTML που ορίζει ένα στοιχείο <*img*>. Στην περιγραφή του βλέπουμε ότι ορίζεται η ιδιότητα *src* η οποία κρατάει τον φάκελο και το όνομα του αρχείου εικόνας, ενώ συγχρόνως ορίζεται και η ιδιότητα *class* με την κλάση της εικόνας αυτής.

Η κλάση συγκεκριμένα, θα χρησιμοποιηθεί στην πορεία του παιχνιδιού για να ελέγξουμε αν το κάθε κελί είναι κλειστό, ανοικτό, βόμβα, μαρκαρισμένο ως βόμβα ή μαρκαρισμένο ως αμφιλεγόμενο.

Ρουτίνα fillBombs()

Η δεύτερη ρουτίνα που καλείται μέσα από την **startGame()** και είναι υπεύθυνη να ορίσει ποια από τα κελιά είναι βόμβες.

```
123 function fillBombs() {
124     var totalCells = gameRows * gameCols;
125
126     for (var i = 1; i <= gameBombs; i++) {
127         var theIndex = Math.floor((Math.random() * totalCells) + 1);
128         var theCell = getCellByIndex(theIndex);
129         var theClass = theCell.className;
130         if (theClass.indexOf('isBomb') < 0) {
131             theCell.className = cellDefaultClass + ' isBomb';
132         } else {
133             i--;
134         }
135     }
136     document.getElementById('leftBombs').value = gameBombs;
137 }
```

Η δομή της είναι σχετικά απλή.

Αρχικά κρατάμε στη τοπική μεταβλητή **totalCells** το σύνολο των κελιών του παιχνιδιού, μέσω του πολλαπλασιασμού των επιθυμητών γραμμών με τις αντίστοιχες στήλες.

Στη συνέχεια εκτελείται ένα βρόχος τόσες φορές όσες και οι βόμβες που θέλουμε να τοποθετήσουμε.

Στο βρόχο αυτό πρώτα βρίσκουμε έναν τυχαίο αριθμό που θα είναι το κελί το οποίο θα ορίσουμε ως βόμβα.

Στη συνέχεια καλείται η ρουτίνα **getCellByIndex()** η οποία επιστρέφει το αντικείμενο του κελιού αυτού.

Εδώ γίνεται ένας έλεγχος με βάση την κλάση του κελιού. Αναζητούμε λοιπόν με χρήση της **indexOf()** εάν το λεκτικό της κλάσης περιέχει το λεκτικό **isBomb**. Εάν το λεκτικό βρεθεί να περιέχεται (σε κάποια θέση του συνολικού λεκτικού, τότε ο τρέχων βρόχος θα πρέπει να ξαναεκτελεστεί και για το λόγω αυτό αφαιρούμε ένα από τον μετρητή του βρόχου.

Εάν το κελί δεν είναι βόμβα (το λεκτικό δηλαδή δεν βρέθηκε στο συνολικό των κλάσεων του στοιχείου), τότε απλά προσθέτουμε την κλάση **isBomb** στις υπόλοιπες του κελιού μέσω της ιδιότητας **className** του αντικειμένου.

Στο τέλος ενημερώνεται το στοιχείο *<input>* **leftBombs** μέσω της ιδιότητας *value* του αντικειμένου του, ώστε να εμφανίσει τις υπολειπόμενες βόμβες που καλείται να ανακαλύψει ο παίκτης.

Ρουτίνα getCellByIndex

Η ρουτίνα αυτή καλείται μέσα από την **fillBombs()** και μετατρέπει το όρισμα που δέχεται και είναι ο αύξων αριθμός ενός κελιού στον πίνακα των κελιών του παιχνιδιού στο αντίστοιχο αντικείμενο του κελιού αυτού, το οποίο και επιστρέφει.

```
139 function getCellByIndex(theIndex) {
140     var theRow = Math.floor(theIndex / gameCols);
141     var theCol = theIndex - (theRow * gameCols);
142     theCol = (theCol == 0) ? gameCols : theCol;
143     theRow = (theIndex % gameCols == 0) ? theRow : ++theRow;
144
145     return document.getElementById('cell-' + theRow + 'x' + theCol);
146 }
```

Όπως είδαμε στην ανάλυση της ρουτίνας **fillTable()**, κάθε κελί του πίνακα έχει δύο χαρακτηριστικά με ειδικό περιεχόμενο. Το μεν *id* περιέχει την υπογραφή του κελιού σαν γραμμή και στήλη, ενώ το *alt* περιέχει τον αύξων αριθμό του.

Θα ήταν ιδανικό να μπορούσαμε απλώς καλώντας κάποια μέθοδο της JavaScript να μας επιστρέψει το αντικείμενο του στοιχείου με μία συγκεκριμένη τιμή στο χαρακτηριστικό *alt*. Τέτοια ρουτίνα όμως δε διαθέτει η JavaScript.

Οι λύσεις λοιπόν που έχουμε είναι ή να αναζητήσουμε όλα τα *<td>* κελιά και στη συνέχεια να ελέγξουμε ένα προς ένα τα περιεχόμενα του χαρακτηριστικού τους *alt* ή να υπολογίσουμε – μετατρέψουμε τον αύξων αριθμό σε στήλη και γραμμή και να αναζητήσουμε το αντικείμενο μέσω του χαρακτηριστικού *id* και της μεθόδου **getElementById()**. Επειδή ένας μαθηματικός υπολογισμός είναι πολύ γρηγορότερος από τη σάρωση των κελιών κάθε φορά, επιλέχθηκε η μέθοδος αυτή.

Στις τέσσερις λοιπόν πρώτες γραμμές κώδικα της τρέχουσας ρουτίνας, υπολογίζουμε τη στήλη και τη γραμμή στις οποίες αντιστοιχεί ο αύξων αριθμός του ορίσματος. Στην τέταρτη μάλιστα ελέγχουμε ώστε στην περίπτωση που η διαίρεση του αριθμού προς τον αριθμό των στηλών έχει υπόλοιπο, να προσθέσουμε ένα ακόμα στον αριθμό της γραμμής (το κελί δηλαδή βρίσκεται σε κάποια θέση πριν το τέλος της επόμενης γραμμής).

Στο τέλος η ρουτίνα επιστρέφει το αντικείμενο του κελιού, όπως είναι λογικό με τη χρήση της **getElementById()** χρησιμοποιώντας τη δομή του *id* όπως την όρισε η ρουτίνα **fillTable()**.

Ρουτίνα showGameInfo()

Η ρουτίνα αυτή είναι υπεύθυνη να εμφανίσει τις πληροφορίες του τρέχοντος παιχνιδιού, στο στοιχείο * * **currentGameInfo**. Το στοιχείο αυτό βρίσκεται επάνω δεξιά στην οθόνη του παιχνιδιού.

```
148 function showGameInfo() {
149     var level = getGameLevel(gameCols, gameRows, gameBombs);
150     document.getElementById('currentGameInfo').innerHTML =
151         'Τύπος: ' + gameCols + 'X' + gameRows + ' με ' + gameBombs + ' βόμβες' + '<br>' +
152         'Δυσκολία: ' + getGameLevelString(gameCols, gameRows, gameBombs) +
153         ' - ' + getGameType(gameCols, gameRows, gameBombs);
154 }
```

Η δομή της είναι απλούστατη. Στην αρχή παίρνει το λεκτικό του επιπέδου δυσκολίας του παιχνιδιού καλώντας τη ρουτίνα **getGameLevel()** και στη συνέχεια ενημερώνει το περιεχόμενο του **currentGameInfo** μέσω της ιδιότητας **innerHTML** με το κατάλληλα μορφοποιημένο λεκτικό.

Ρουτίνα getGameLevel()

Μέσω της ρουτίνας αυτής βρίσκουμε το επίπεδο δυσκολίας του παιχνιδιού.

```
156 function getGameLevel(cols, rows, bombs) {
157     var level = ((bombs * 100) / (cols * rows)) * 2;
158     return level.toFixed(0);
159 }
```

Η υλοποίησή της είναι απλούστατη. Κάνει έναν απλό μαθηματικό υπολογισμό και επιστρέφει το αποτέλεσμά του. Ο λόγος υλοποίησης της ρουτίνας αυτής είναι ότι, ο μαθηματικός αυτός υπολογισμός θα χρειαστεί σε αρκετά διαφορετικά σημεία του κώδικα. Στην περίπτωση που χρειαστεί για οποιοδήποτε λόγο να αλλάξει, θα πρέπει να ανατρέξουμε σε όλα τα σημεία όπου αναφέρεται και να τον διορθώσουμε. Περικλείοντάς τον όμως μέσα σε μία (απλή κατά τα άλλα) ρουτίνα, μπορούμε γρήγορα αλλά κυρίως με ασφάλεια να τον μεταβάλλουμε, οποτεδήποτε θελήσουμε…

Παρατηρώντας τα έτοιμα επίπεδα δυσκολίας του παιχνιδιού σε συνδυασμό με τα χαρακτηριστικά τους (στήλες, γραμμές, βόμβες), όπως αυτό υπάρχει στο περιβάλλον Windows, βλέπουμε ότι ορίζεται στην πραγματικότητα από το ποσοστό τον βομβών επί του συνόλου των κελιών.

Επειδή όμως με ποσοστό βομβών πάνω από το 50% είναι πρακτικά αδύνατο κάποιος να παίξει, καθώς τα μισά κελιά θα είναι βόμβες, το ποσοστό αυτό πολλαπλασιάζεται επί δύο, ώστε το τελικό νούμερο να

βγαίνει με όριο το εκατό, το οποίο θεωρείται και το μεγαλύτερο επίπεδο στο παιχνίδι.

Χρησιμοποιώντας λοιπόν τα τρία ορίσματα που είναι αντίστοιχα οι στήλες, οι γραμμές και οι βόμβες, βρίσκουμε το ποσοστό αυτό.

Στην επιστροφή της η ρουτίνα αυτή χρησιμοποιεί τη μέθοδο **toFixed(0)** για να επιστρέψει τον πλησιέστερο ακέραιο πάντα αριθμό.

Ρουτίνα getGameLevelString()

Η ρουτίνα αυτή είναι κατά κάποιο τρόπο συμπληρωματική της **getGameLevel()**.

```
161 function getGameLevelString(cols, rows, bombs) {
162     var level = getGameLevel(cols, rows, bombs);
163     return (level < 10) ? '0' + level : level;
164 }
```

Αφού την καλέσει μεταφέροντάς της τα τρία ορίσματα τα οποία και αυτή χρησιμοποιεί επίσης και πάρει το επίπεδο δυσκολίας σε αριθμό, στη συνέχεια ελέγχει αν ο αριθμός αυτός είναι μικρότερος του δέκα και του προσθέτει σε αυτή την περίπτωση στην αρχή το ψηφίο μηδέν.

Στο τέλος επιστρέφει το κατάλληλα αυτό μορφοποιημένο λεκτικό.

Και αυτή η ρουτίνα θα μπορούσε, όπως και η **getGameLevel()** να ενσωματωθεί κατευθείαν στο κάθε σημείο κλήσης, όμως, όπως αναλύθηκε και προηγουμένως, η υλοποίησή της ως ανεξάρτητη ρουτίνα, μας παρέχει την ευκολία μετατροπής χωρίς κίνδυνο.

Επίσης με την διαφοροποίησή της από την **getGameLevel()**, μας δίνεται η δυνατότητα να μπορούμε να πάρουμε το επίπεδο δυσκολίας σε μορφή αριθμού ή κατάλληλα μορφοποιημένου λεκτικού, ανάλογα με το ποιόν τύπο δεδομένων χρειαζόμαστε κάθε φορά.

Ρουτίνα getGameType()

Μέσω της ρουτίνας αυτής λαμβάνουμε το λεκτικό επιπέδου δυσκολίας.

```
function getGameType(cols, rows, bombs) {
    var level = getGameLevel(cols, rows, bombs);
    var gameType = '';
    switch (true) {
        case level < 15:
            gameType = 'Αρχάριο';
            break;
        case level < 30:
            gameType = 'Εύκολο';
            break;
        case level < 40:
            gameType = 'Κανονικό';
            break;
        case level < 50:
            gameType = 'Δύσκολο';
            break;
        case level < 70:
            gameType = 'Πολύ δύσκολο';
            break;
        default:
            gameType = 'Extreme';
            break;
    }
    return gameType;
}
```

Και πάλι καλείται αρχικά η **getGameLevel()** μεταφέροντάς της τα τρία ορίσματα με τα οποία καλέστηκε η τρέχουσα ρουτίνα.

Στη συνέχεια μέσω ενός ελέγχου πολλαπλών επιλογών, ανάλογα με τον αριθμό του επιπέδου δυσκολίας, επιστρέφεται το σχετικό λεκτικό.

Σαφώς, ο ορισμός του αριθμού των επιπέδων σε έξη είναι αυθαίρετος και αντί για τα έξη αυτά επίπεδα θα μπορούσαμε να προσθέσουμε και επιπλέον ή να αφαιρέσουμε κάποια.

Βασικός στόχος επιλογής του πλήθους τους, ήταν να υπακούνε τα επίπεδα αυτά στα ήδη προεπιλεγμένα από το παιχνίδι των Windows και στη συνέχεια να προστεθούν κάποια επιπλέον, ώστε για να καλυφθεί το κενό στην αρχή και το τέλος των ορίων, χωρίς όμως υπερβολές στο τελικό πλήθος των επιπέδων.

Ρουτίνα displayTheTime()

Η ρουτίνα που ορίζεται να εκτελείται ανά ένα δευτερόλεπτο από τη ρουτίνα **startGame()**.

```
function displayTheTime() {
    var nowTime = new Date();

    var timeDiff = nowTime - startTime;
    timeDiff /= 1000; // We don't want the milliseconds

    var timeSeconds = Math.round(timeDiff % 60);
    timeSeconds = (timeSeconds < 10) ? '0' + timeSeconds : timeSeconds;
    timeDiff = Math.floor(timeDiff / 60); // Remove the seconds

    var timeMinutes = Math.round(timeDiff % 60);
    timeMinutes = (timeMinutes < 10) ? '0' + timeMinutes : timeMinutes;
    timeDiff = Math.floor(timeDiff / 60); // Remove the minutes

    var timeHours = Math.round(timeDiff % 24);
    timeHours = (timeHours < 10) ? '0' + timeHours : timeHours;
    timeDiff = Math.floor(timeDiff / 24); // Remove the hours

    var timeDays = (timeDiff > 0) ? timeDiff + ' ' : ''; // Rest is days... if they exists ;-)

    document.getElementById('gameTime').value = timeDays + timeHours + ':' + timeMinutes + ':' + timeSeconds;
}
```

Πριν την έναρξη της επαναλαμβανόμενης εκτέλεσης της τρέχουσας ρουτίνας, είδαμε ότι κρατήθηκε στην καθολική μεταβλητή **startTime** η τρέχουσα (εκείνη τη στιγμή) ημέρα και ώρα σε μορφή αριθμού.

Έτσι τώρα παίρνουμε και πάλι τον αριθμό αυτό της τρέχουσας ημερομηνίας και ώρας και φυσικά τον αφαιρούμε από τον αρχικό.

Για να φτιάξουμε τον αριθμό αυτό σε μορφή αναγνώσιμη πρώτα αφαιρούμε τα milliseconds.

Στη συνέχεια με διαδοχικές διαιρέσεις όπου κρατώντας το υπόλοιπο της διαίρεσης, στη συνέχεια προσθέτουμε στον αριθμό αυτό το χαρακτήρα μηδέν στην αρχή του λεκτικού (αν ο αριθμός που βρήκαμε είναι μικρότερος του δέκα) και στο τέλος αφαιρώντας ότι κρατήσαμε ώστε να πάμε στο επόμενο πεδίο και να επαναλάβουμε τις πράξεις, βρίσκουμε τα δευτερόλεπτα, τα λεπτά, τις ώρες και ίσως και τις μέρες (αν κάποιος ξέχασε τον υπολογιστή του ανοικτό και στον περιηγητή του το παιχνίδι που δεν έχει ακόμα ολοκληρώσει).

Όλα αυτά τα στοιχεία κρατούνται σε τοπικές μεταβλητές τις οποίες θα χρησιμοποιήσουμε κατά τη δόμηση του τελικού λεκτικού.

Τέλος ενημερώνεται το στοιχείο *<input>* **gameTime** μέσω της ιδιότητας *value* για το λεκτικό της ώρας που θα εμφανίσει χρησιμοποιώντας τις παραπάνω μεταβλητές και προσθέτοντας ανάμεσά τους τα κατάλληλα διαχωριστικά χαρακτήρες - λεκτικά.

Ρουτίνα leftClickOnCell()

Η μία από τις δύο βασικές ρουτίνες του παιχνιδιού, καθώς εκτελείται κάθε φορά που ο παίκτης δημιουργεί ένα συμβάν (*event*) τύπου *click* σε κάποιο κελί του πίνακα του παιχνιδιού.

```
function leftClickOnCell(theCell) {
    if (gameIsOn) {
        var confirmOpen = true;
        var theClass = theCell.className;
        var theImgClass = theCell.childNodes[0].className;
        if (theImgClass.indexOf('flagImage') >= 0) {
            alert('Το κελί έχει μαρκαριστεί ως βόμβα!\n\nΓια να το ανοίξετε πρέπει να το ξεμαρκάρετε...\n\n');
            confirmOpen = false;
        } else if (theImgClass.indexOf('checkImage') >= 0) {
            confirmOpen = confirm('Το κελί είναι μαρκαρισμένο ως αμφιλεγόμενο!\n\nΘέλετε σίγουρα να το ανοίξετε;\n\n');
        }
        if (confirmOpen) {
            if (theClass.indexOf('isBomb') < 0) {
                countNearBombs(theCell);
            } else {
                clearTimeout(gameTime);
                showAllBombs();
                theCell.innerHTML = '<img' +
                    ' src="images/thebomb.jpg"' +
                    ' class="cellImage bombImage"' +
                    ' />';
                alert("Μ Π Ο Υ Μ  ! ! !\n\n\Ωχ λάθος! Αυτή ήταν βόμβα !\n\nGame Over !!!\n\n");
            }
        }
    }
}
```

Όπως είδαμε στη ανάλυση της ρουτίνας **fillTable()**, μέσω της μεθόδου **addEventListener()**, κάθε κελί εκτελεί την τρέχουσα ρουτίνα όταν ο χρήστη πατήσει με το αριστερό κουμπί του ποντικιού του ή ασκήσει ένα απλό πάτημα στην οθόνη του κινητού του επάνω σε ένα κελί.

Από την πρώτη γραμμή του κώδικα βλέπουμε ότι η υπόλοιπες εντολές της θα εκτελεστούν μόνο εφόσον το παιχνίδι βρίσκεται σε εξέλιξη (η καθολική μεταβλητή **gameIsOn** έχει την τιμή *true*) και αυτό έχει οριστεί αρχικά μέσα στη ρουτίνα **startGame()**. Εφόσον σε κάποιο άλλο σημείο του κώδικα η μεταβλητή αυτή δεν έχει αλλάξει τιμή εκτελούνται τα παρακάτω.

Να θυμηθούμε επίσης ότι η κλήση της τρέχουσας ρουτίνας μέσω της **addEventListener()** και του συμβάντος (*event*) που δημιουργήθηκε, θα θέσει στο όρισμα (μεταβλητή **theCell**) της ρουτίνας το αντικείμενο του στοιχείου που δημιουργήθηκε το συμβάν, που στην περίπτωσή μας είναι το αντικείμενο του κελιού στο οποίο ο χρήστης έκανε κλικ.

Από το αντικείμενο αυτό και μέσω του πίνακα **childNodes[]** του, ο οποίος περιέχει όλα τα αντικείμενα που κρέμονται ιεραρχικά από κάτω του (φιλοξενούνται στο εσωτερικό του δηλαδή), παίρνουμε το πρώτο από αυτά που είναι μία εικόνα.

Κάθε εικόνα που βλέπει στην οθόνη του ο παίκτης, σε επίπεδο HTML εκτός από την κοινή κλάση **cellImage**, έχει και μία ειδική, ανάλογα με την εικόνα κλάση.

Έτσι, η εικόνα του κανονικού κλειστού κελιού έχει την κλάση **normalImage**, η εικόνα του μαρκαρισμένου ως βόμβα κελιού την κλάση **flagImage**, η εικόνα του αμφιλεγόμενου κελιού την κλάση **checkImage** και η εικόνα του ανοιγμένου κελιού την κλάση **openImage**.

Αφού λοιπόν μέσω της ιδιότητας **className** πάρουμε το λεκτικό των κλάσεων της εικόνας αυτής, γίνεται ένας πρώτος έλεγχος για το αν είναι μαρκαρισμένο το κελί ως βόμβα ή ως αμφιλεγόμενο.

Στην πρώτη περίπτωση εμφανίζεται απλά ένα μήνυμα ειδοποίησης προς τον παίκτη, στη μεν δεύτερη μέσω χρήσης της ρουτίνας **confirm** καλούμε τον παίκτη να πατήσει “Ναι” ή “Άκυρο”, εμφανίζοντάς του μία ερώτηση επιλογής αν θέλει να ανοίξει τελικά το κελί.

Στο επόμενο στάδιο, αν ο χρήστης επέλεξε “Ναι” στη προηγούμενη ερώτηση ή αν το κελί δεν έχει κάποιο μαρκάρισμα επάνω του, ελέγχουμε την ύπαρξη μίας ακόμα κλάσης, αυτή τη φορά όμως στο λεκτικό κλάσεων του κελιού.

Κατά την εκτέλεση της ρουτίνας **fillBombs()** κάποια κελιά ορίστηκαν ως βόμβες προσθέτοντάς τους στην κλάση τους την κλάση **isBomb**. Αν λοιπόν από τον έλεγχο αυτής της κλάσης βρεθεί ότι το κελί δεν είναι βόμβα, εκτελείται η ρουτίνα **countNearBombs()** η οποία θα εμφανίσει έναν αριθμό, αν το κελί αυτό επικοινωνεί με άλλες βόμβες ή θα προκαλέσει το άνοιγμά του καθώς το άνοιγμα των γειτονικών του, αν είναι κενό κελί.

Στη περίπτωση τώρα που το κελί βρεθεί να είναι βόμβα, πρώτα απ’ όλα σταματούμε την εκτέλεση της ρουτίνας **displayTheTime()** η οποία μετρά και εμφανίζει το χρόνο που έχει περάσει από την έναρξη του παιχνιδιού, μέσω της κλήσης της **clearTimeout()**.

Στη συνέχεια καλείται η ρουτίνα **showAllBombs()** η οποία θα εμφανίσει όλες τις βόμβες του παιχνιδιού.

Τέλος, αφού αλλάξει η εικόνα του κελιού μέσω της ιδιότητας **innerHTML** και αντικατασταθεί με την εικόνα της βόμβας σε κίτρινο φόντο, εμφανίζεται το κατάλληλο μήνυμα “Game Over” στον παίκτη.

Ρουτίνα countNearBombs()

Η ρουτίνα αυτή καλείται από τη **leftClickOnCell()** στη περίπτωση που το κελί στο οποίο έκανε κλικ ο παίκτης δεν είναι μαρκαρισμένο ή βόμβα.

```
242 function countNearBombs(theCell) {
243     var totalBombs = 0;
244     var nearCells = getNearCells(theCell);
245     var nearCellsLen = nearCells.length;
246
247     for (var i = 0; i < nearCellsLen; i++) {
248         if (nearCells[i].className.indexOf('isBomb') >= 0) {
249             totalBombs++;
250         }
251     };
252     if (theCell.childNodes[0].className.indexOf('flagImage') >= 0) {
253         gameFlags--;
254         document.getElementById('leftBombs').value = (gameBombs - gameFlags);
255     };
256     theCell.innerHTML = '<img' +
257         ' src="images/near-' + totalBombs + '.jpg"' +
258         ' class="cellImage openImage"' +
259         ' />';
260     if (totalBombs == 0) {
261         for (var i = 0; i < nearCellsLen; i++) {
262             countNearBombs(nearCells[i]);
263         };
264     }
265 }
```

Αρχικά εκτελείται η ρουτίνα **getNearCells()** στην οποία στέλνουμε σαν όρισμα το κελί για το οποίο θα μετρήσουμε και θα εμφανίσουμε τις γειτονικές βόμβες και είναι το όρισμα που αρχικά έχει κληθεί η τρέχουσα ρουτίνα. Επειδή η τρέχουσα ρουτίνα μπορεί να ξανακληθεί εσωτερικά δημιουργώντας μία αλυσίδα εσωτερικών κλήσεων, τότε το όρισμα θα περιέχει το νέο αυτό κελί που θα σταλεί με τη νέα κλήση.

Η **getNearCells()** θα επιστρέψει έναν πίνακα (*array*) από αντικείμενα κελιών που γειτονεύουν με το κελί όρισμα.

Ο πίνακας αυτός μπορεί να περιέχει από τρία μέχρι οκτώ διαφορετικά αντικείμενα κελιών. Μέχρι τρία στην περίπτωση που το τρέχων κελί είναι ένα γωνιακό, μέχρι πέντε σε περίπτωση που είναι κελί πλευράς και μέχρι οκτώ το μέγιστο εάν είναι οποιοδήποτε άλλο κελί, όπου δε θα περιλαμβάνονται τα ήδη ανοιγμένα κελιά.

Στο επόμενο βήμα μετράμε πόσα από τα κελιά αυτά είναι βόμβες, ελέγχοντας όπως και πριν στην κλάση τους την ύπαρξη της **isBomb** και κρατάμε τον αριθμό αυτό στη τοπική μεταβλητή **totalBombs**.

Εάν τώρα, το τρέχων κελί (το οποίο ελέγχουμε) είναι μαρκαρισμένο ως βόμβα από τον παίκτη και επέλεξε να ανοιχτεί στο ανάλογο ερώτημα, αφαιρούμε ένα μαρκάρισμα από τον αριθμό των μαρκαρισμένων κελιών και ενημερώνουμε το ανάλογο στοιχείο *<input>* **leftBombs** μέσω της ιδιότητάς του **value** για να εμφανίσει το νέο σύνολο βομβών προς ανακάλυψη.

Στη συνέχεια με ένα κόλπο εμφανίζουμε τον κατάλληλο αριθμό στο κελί, που δείχνει με πόσες βόμβες επικοινωνεί ή το άδειο, ανοιγμένο κελί.

Είδαμε πιο πάνω ότι στην τοπική μεταβλητή **totalBombs** κρατήσαμε το πλήθος των κελιών βομβών με τα οποία γειτονεύει το τρέχω κελί.

Αν κοιτάξουμε στα περιεχόμενα του φακέλου *images* της εφαρμογής, θα δούμε ότι υπάρχουν εννέα εικόνες με όνομα **near-0** μέχρι **near-8** όπου η **near-0** είναι η εικόνα του ανοικτού, άδειου κελιού, ενώ σε όλες τις υπόλοιπες ο αριθμός τους αντιστοιχεί στον αριθμό των γειτονικών κελιών βομβών.

Σχηματίζοντας λοιπόν το όνομα της εικόνας από το λεκτικό “images/near-“, την τιμή της μεταβλητής **totalBombs** και προσθέτοντας στο τέλος το λεκτικό με την επέκταση “.jpg”, έχουμε την εικόνα του άδειου ή με τον κατάλληλο αριθμό κελιού.

Μέσω της ιδιότητας **innerHTML** του τρέχοντος κελιού αλλάζουμε την εικόνα του στην εικόνα που φτιάξαμε παραπάνω, προσθέτοντας τον κατάλληλο HTML κώδικα στο εσωτερικό του.

Στο τελευταίο βήμα τώρα, εάν το κελί είναι τελικά κενό (η **totalBombs** έχει την τιμή μηδέν), καλούμε επαναλαμβανόμενα για κάθε γειτονικό κελί, ξανά, την **countNearBombs()**, στέλνοντάς της τώρα ως όρισμα το κάθε ένα αυτό γειτονικό κελί.

Με τον τρόπο αυτό, καλώντας δηλαδή επαναλαμβανόμενα την τρέχουσα ρουτίνα για κάθε γειτονικό κελί επιτυγχάνουμε, το κάθε γειτονικό κελί που είναι επίσης άδειο να ανοίξει επίσης, αυτό με τη σειρά του θα καλέσει όλα τα γειτονικά του κελιά και αυτός ο βρόχος θα συνεχιστεί μέχρι να βρεθεί κελί με αριθμό.

Τότε η αλυσίδα των επαναλήψεων θα σταματήσει και σταδιακά θα επιστρέψει στην αρχική κλήση της ρουτίνας, όπου και θα τερματιστεί ο βρόχος αυτός.

Ρουτίνα getNearCells()

Η ρουτίνα καλείται από την **countNearBombs()** και επιστρέφει τα γειτονικά κελιά του κελιού ορίσματος.

```
function getNearCells(theCell) {
    var theID = theCell.id.slice(5).split('x');
    var theRow = parseInt(theID[0]);
    var theCol = parseInt(theID[1]);
    var nearCells = [];
    var checkCell = null;

    if (theRow > 1) {
        if (theCol > 1) {
            checkCell = isValidToAdd((theRow - 1), (theCol - 1));
            if (checkCell) {nearCells.push(checkCell);}
        };
        checkCell = isValidToAdd((theRow - 1), theCol);
        if (checkCell) {nearCells.push(checkCell);}
        if (theCol < gameCols) {
            checkCell = isValidToAdd((theRow - 1), (theCol + 1));
            if (checkCell) {nearCells.push(checkCell);}
        };
    };
    if (theCol > 1) {
        checkCell = isValidToAdd(theRow, (theCol - 1));
        if (checkCell) {nearCells.push(checkCell);}
    };
    if (theCol < gameCols) {
        checkCell = isValidToAdd(theRow, (theCol + 1));
        if (checkCell) {nearCells.push(checkCell);}
    };
    if (theRow < gameRows) {
        if (theCol > 1) {
            checkCell = isValidToAdd((theRow + 1), (theCol - 1));
            if (checkCell) {nearCells.push(checkCell);}
        };
        checkCell = isValidToAdd((theRow + 1), theCol);
        if (checkCell) {nearCells.push(checkCell);}
        if (theCol < gameCols) {
            checkCell = isValidToAdd((theRow + 1), (theCol + 1));
            if (checkCell) {nearCells.push(checkCell);}
        };
    };

    return nearCells;
}
```

Η ρουτίνα επιστρέφει τα γειτονικά κελιά του κελιού ορίσματος σε μορφή πίνακα αντικειμένων.

Σαν γειτονικό ορίζεται κάθε κελί το οποίο ακουμπά στις κάθετες και οριζόντιες πλευρές του τρέχοντος κελιού, αλλά και στις τέσσερις γωνίες του.

Έτσι κάθε κελί μπορεί να έχει μέχρι και οκτώ γειτονικά κελιά.

Στον πίνακα που θα επιστραφεί δεν θα περιέχονται τα κελιά που έχουν ήδη ανοιχτεί, συνθήκη που ελέγχεται μέσω κλήσης της ρουτίνας **isValidToAdd()** και στέλνοντάς της ως ορίσματα τις συντεταγμένες του κελιού που θέλουμε να ελέγξουμε αν θα προστεθεί στον τελικό πίνακα.

Η προσθήκη των αντικειμένων των κελιών στον πίνακα γίνεται αφού αρχικά δημιουργήσουμε έναν άδειο πίνακα, στη συνέχεια κάθε φορά καλείται η μέθοδος **push()** του πίνακα αυτού με όρισμα το προς εισαγωγή αντικείμενο.

Στην αρχή της ρουτίνας παίρνουμε μέσω της ιδιότητας *id* του κελιού που στάλθηκε ως όρισμα, ένα λεκτικό το οποίο περιέχει όπως ξέρουμε τις συντεταγμένες του κελιού στη μορφή *cell-*$^{ΑριθμόςΓραμμής}x^{ΑριθμόςΣτήλης}$, το οποίο του δόθηκε κατά τη δημιουργία του στη ρουτίνα **fillTable()**.

Από το λεκτικό αυτό με χρήση της μεθόδου **slice()** αφαιρούμε τους πέντε πρώτους χαρακτήρες (το “cell-“ δηλαδή) και κρατούμε το υπόλοιπο, που περιέχει τις συντεταγμένες της γραμμής και της στήλης.

Από το υπόλοιπο αυτό λεκτικό, χωρίζοντάς το με χρήση της μεθόδου **split()** με βάση το χαρακτήρα “x” ο οποίος περιέχεται στη μέση των συντεταγμένων, παίρνουμε έναν πίνακα με δύο στοιχεία, τη γραμμή και τη στήλη. Τα στοιχεία αυτά τα αποθηκεύουμε στις τοπικές μεταβλητές **theRow** και **theCol** αντίστοιχα, μετατρέποντάς τα παράλληλα σε αριθμό (από λεκτικό που είναι) με χρήση της ρουτίνας **parseInt()**.

Στη συνέχεια γίνονται οι έλεγχοι της τοποθεσίας του κελιού και με βάση αυτούς ελέγχεται το κάθε γειτονικό κελί αν είναι κλειστό (άρα και μπορούμε να το προσθέσουμε στον τελικό πίνακα) με κλήση της ρουτίνας **isValidToAdd()**.

Το αποτέλεσμα της κλήσης αυτής είναι ή το αντικείμενο του κελιού αν επιτρέπεται να προστεθεί ή η τιμή *null* αν δεν επιτρέπεται.

Εάν η επιστροφή της κλήσης είναι αντικείμενο, αυτό προστίθεται στον πίνακα με χρήση της μεθόδου **push()** όπως προαναφέραμε.

Αφού ολοκληρωθεί ο έλεγχος και των οκτώ (το μέγιστο) γειτονικών κελιών, η ρουτίνα επιστρέφει τον πίνακα με τα αντικείμενα.

Ρουτίνα isValidToAdd()

Η ρουτίνα αυτή όπως είδαμε προηγουμένως, καλείται μέσα από την **getNearCells()**.

```
310 function isValidToAdd(row, col) {
311     var theCell = document.getElementById('cell-' + row + 'x' + col);
312     return (theCell.childNodes[0].className.indexOf('openImage') < 0) ? theCell : null;
313 }
```

Είναι μία πολύ απλή ρουτίνα η οποία δέχεται ως ορίσματα τη γραμμή και τη στήλη κάποιου κελιού.

Στη συνέχεια μέσω της **getElementById()** στην οποία στέλνει ως όρισμα ένα λεκτικό με τις συντεταγμένες του κελιού προς αναζήτηση, δομημένο με τη μορφή *cell-*$^{ΑριθμόςΓραμμής}x^{ΑριθμόςΣτήλης}$, αναζητά και παίρνει το αντικείμενο του κελιού αυτού.

Στη συνέχεια, από το αντικείμενο αυτό καλεί τον πίνακα **childNodes[]**. Ο πίνακας αυτός περιέχει όλα τα αντικείμενα των στοιχείων HTML, που βρίσκονται ιεραρχικά, κάτω από το αντικείμενο αυτό. Από τον πίνακα θα πάρει το πρώτο του αντικείμενο (**childNodes[0]**), το οποίο είναι ένα στοιχείο εικόνας *<img>*, του κελιού.

Τέλος, ελέγχει την κλάση του αντικειμένου αυτού (εικόνα του κελιού), για την ύπαρξη του λεκτικού **openImage**.

Εάν βρεθεί σημαίνει ότι το κελί είναι ήδη ανοιγμένο, άρα και δε θα συμπεριληφθεί στον πίνακα των κελιών προς επεξεργασία, οπότε και επιστρέφει την τιμή *null*, ενώ στην αντίθετη περίπτωση επιστρέφει το αντικείμενο του κελιού, για να προστεθεί στα υπόλοιπα του πίνακα.

Ρουτίνα rightClickOnCell()

Η δεύτερη από τις δύο βασικές ρουτίνες του παιχνιδιού, καθώς εκτελείται κάθε φορά που ο παίκτης δημιουργεί ένα συμβάν (*event*) τύπου *contextmenu* σε κάποιο κελί του πίνακα του παιχνιδιού.

```
function rightClickOnCell(theCell) {
    if (gameIsOn) {
        var theImgClass = theCell.childNodes[0].className;
        if (theImgClass.indexOf('normalImage') >= 0 && (gameBombs - gameFlags) > 0) {
            theCell.innerHTML = '<img' +
                ' src="images/flag.jpg"' +
                ' class="cellImage flagImage"' +
                ' />';
            gameFlags++;
            document.getElementById('leftBombs').value = (gameBombs - gameFlags);
            if (gameBombs == gameFlags) {
                if (allBombsChecked()) {
                    clearTimeout(gameTime);
                    showAllBombs();
                    saveStatistics();
                    alert("Μ Π Ρ Α Β Ο  ! ! !\n\nΌλες οι βόμβες βρέθηκαν σωστά !\n\nGame Over !!!\n\n");
                } else {
                    alert("Χμ, κάποιο λάθος έγινε!\n\nΟι βόμβες δεν είναι σωστά μαρκαρισμένες !\n\nΓια ελέγξτε τις πάλι...\n\n");
                }
            }
        } else if (theImgClass.indexOf('flagImage') >= 0) {
            theCell.innerHTML = '<img' +
                ' src="images/check.jpg"' +
                ' class="cellImage checkImage"' +
                ' />';
            gameFlags--;
            document.getElementById('leftBombs').value = (gameBombs - gameFlags);
        } else if (theImgClass.indexOf('openImage') >= 0) {
        } else {
            theCell.innerHTML = '<img' +
                ' src="images/normal.jpg"' +
                ' class="cellImage normalImage"' +
                ' />';
        }
    }
}
```

Όπως αναφέραμε αντίστοιχα και στην ανάλυση της ρουτίνας **leftClickOnCell()**, από τη ρουτίνα **fillTable()**, μέσω της μεθόδου **addEventListener()**, κάθε κελί εκτελεί την τρέχουσα ρουτίνα όταν ο χρήστη πατήσει με το δεξί κουμπί του ποντικιού του ή ασκήσει ένα παρατεταμένο πάτημα στην οθόνη του κινητού του επάνω σε ένα κελί.

Επίσης βλέπουμε ότι η υπόλοιπες εντολές της ρουτίνας και πάλι θα εκτελεστούν μόνο εφόσον το παιχνίδι βρίσκεται σε εξέλιξη (η καθολική μεταβλητή **gameIsOn** έχει την τιμή *true*) και αυτό έχει όπως προαναφέραμε οριστεί αρχικά μέσα στη ρουτίνα **startGame()**. Εφόσον σε κάποιο άλλο σημείο του κώδικα η μεταβλητή αυτή δεν έχει αλλάξει τιμή εκτελούνται τα παρακάτω.

Να θυμηθούμε επίσης ότι η κλήση της τρέχουσας ρουτίνας μέσω της **addEventListener()** και του συμβάντος (*event*) που δημιουργήθηκε, θα θέσει στο όρισμα (μεταβλητή **theCell**) της ρουτίνας το αντικείμενο του στοιχείου που δημιουργήθηκε το συμβάν, που στην περίπτωσή μας είναι το αντικείμενο του κελιού στο οποίο ο χρήστης έκανε δεξί κλικ.

Από το αντικείμενο αυτό και πάλι μέσω του πίνακα **childNodes[]** του, ο οποίος περιέχει όλα τα αντικείμενα που κρέμονται ιεραρχικά από κάτω του (φιλοξενούνται στο εσωτερικό του δηλαδή), παίρνουμε το πρώτο από αυτά που είναι όπως πλέον γνωρίζουμε μία εικόνα.

Στη συνέχεια ελέγχοντας και πάλι την κλάση της εικόνας αυτής η ρουτίνα ενεργεί ανάλογα με τις περιπτώσεις που ακολουθούν.

Πρώτη περίπτωση, το κελί είναι κλειστό και το πλήθος των μαρκαρισμένων κελιών ως βόμβα είναι μικρότερο του πλήθους των βομβών που καλείται ο παίκτης να βρει.

Σε αυτή την περίπτωση αρχικά αλλάζει η εικόνα του κελιού σε σημαία μαρκαρίσματος, αλλάζει παράλληλα και η κλάση της εικόνας αυτής σε **flagImage**, αυξάνεται κατά ένα ο αριθμός των μαρκαρισμένων κελιών και αλλάζει το περιεχόμενο του στοιχείου <*input*> **leftBombs** με το υπόλοιπο των βομβών προς αναζήτηση.

Στο επόμενο βήμα ελέγχουμε αριθμητικά αν ο βρήκε όλες τις βόμβες και αν αυτό είναι αληθές (αριθμητικά όμως μόνο), καλείται η ρουτίνα **allBombsChecked()**, η οποία επιστρέφει την τιμή *true* αν όντως έχουν βρεθεί όλες οι βόμβες ή *false* αν έστω και ένα κελί είναι λανθασμένα μαρκαρισμένο ως βόμβα.

Στη περίπτωση που όλα τα μαρκαρισμένα κελιά είναι σωστά, πρώτα σταματά η μέτρηση του χρόνου που έχει παρέλθει μέσω της **clearTimeout()**. Έπειτα καλείται η **showAllBombs()** για να εμφανίσει όλες τις βόμβες. Στη συνέχεια καλείται η **saveStatistics()** η οποία θα καταγράψει τον χρόνο και τα στοιχεία του τρέχοντος παιχνιδιού στο ιστορικό και τέλος εμφανίζουμε ένα μήνυμα επιβράβευσης στον παίκτη, πληροφορώντας τον για την επιτυχή ολοκλήρωση του παιχνιδιού.

Στη περίπτωση τώρα που μέσω της **allBombsChecked()** βρεθεί ότι δεν είναι όλα τα μαρκαρισμένα κελιά σωστά, απλά ενημερώνουμε τον παίκτη μέσω ενός μηνύματος στην οθόνη του…

Δεύτερη περίπτωση με βάση την κλάση της εικόνας του κελιού, το κελί είναι μαρκαρισμένο ως βόμβα.

Στην περίπτωση αυτή, απλά αλλάζουμε την εικόνα του κελιού σε λατινικό ερωτηματικό (?) και την κλάση του σε αμφιλεγόμενο (**checkImage**).

Στη συνέχεια αφαιρούμε ένα από το πλήθος των κελιών που ο παίκτης έχει μαρκάρει ως σημαίες και τέλος φυσικά ενημερώνουμε το στοιχείο <*input*> **leftBombs** για να εμφανίσει την αλλαγή αυτή.

Τρίτη περίπτωση με βάση την κλάση της εικόνας του κελιού, το κελί είναι ήδη ανοιγμένο.

Στην περίπτωση αυτή δεν γίνεται καμία ενέργεια.

Στην τελευταία περίπτωση που έχει απομείνει με βάση την κλάση της εικόνας του κελιού, το κελί να είναι μαρκαρισμένο ως αμφιλεγόμενο.

Σε αυτή την περίπτωση το μόνο που κάνουμε είναι να αλλάξουμε την εικόνα του κελιού σε κανονικό, κλειστό κελί και να αλλάξουμε κατάλληλα την κλάση της εικόνας σε **normalImage**.

Ρουτίνα allBombsChecked()

Η ρουτίνα που καλείται από τη **rightClickOnCell()** και ελέγχει αν όλα τα μαρκαρισμένα ως βόμβες κελιά είναι σωστά.

```
352 function allBombsChecked() {
353     var allBombs = document.getElementsByClassName('isBomb');
354     var allBombsLen = allBombs.length;
355     var bombsFound = 0;
356
357     for (var i = 0; i < allBombsLen; i++) {
358         var theImgClass = allBombs[i].childNodes[0].className;
359         bombsFound = (theImgClass.indexOf('flagImage') >= 0) ? bombsFound + 1 : bombsFound;
360     }
361     return (bombsFound == gameBombs) ? true : false;
362 }
```

Στη ρουτίνα αυτή βλέπουμε στη αρχή της μία διαφορετικού τύπου αναζήτηση στοιχείων από τα περιεχόμενα της σελίδας.

Μέχρι τώρα αναζητούσαμε στοιχεία με τη χρήση της μεθόδου **getElementById()** η οποία μας επέστρεφε ένα και μόνο αντικείμενο, του στοιχείου του οποίου η ιδιότητα *id* συμφωνούσε με το λεκτικό που στέλναμε ως όρισμα της μεθόδου.

Εδώ βλέπουμε τη μέθοδο **getElementsByClassName()** από της οποίας το όνομα καταλαβαίνουμε ότι επιστρέφει ένα πλήθος αντικειμένων σε μορφή πίνακα (*array*), με βάση αναζήτησης τώρα την κλάση του κάθε στοιχείου και αν αυτή περιέχει το λεκτικό με την (ή τις) κλάσεις που θα δοθεί ως όρισμα στη μέθοδο.

Με τη χρήση λοιπόν της παραπάνω μεθόδου, γίνεται αναζήτηση όλων των κελιών με ορισμένη την κλάση **isBomb** από τη ρουτίνα **fillBombs()**.

Στη συνέχεια σαρώνονται όλα τα αντικείμενα και ελέγχεται αν η κλάση της εικόνας που περιέχουν (την οποία παίρνουμε μέσω του πρώτου αντικειμένου του πίνακα **childNodes[]** και ελέγχουμε μέσω της ιδιότητας **className**) περιλαμβάνουν την κλάση **flagImage**.

Αν την περιλαμβάνουν σημαίνει ότι ο παίκτης έχει σωστά μαρκάρει τη βόμβα αυτή.

Για κάθε σωστά μαρκαρισμένη βόμβα, ο μετρητής **bombsFound** αυξάνει κατά ένα.

Εάν στο τέλος, η τιμή του μετρητή είναι ίδια με την ποσότητα των προς εύρεση βομβών, η ρουτίνα επιστρέφει την τιμή *true*, στην αντίθετη περίπτωση επιστρέφει την τιμή *false*.

Ρουτίνα showAllBombs()

Η ρουτίνα αυτή καλείται με τον τερματισμό κάθε παιχνιδιού, είτε αυτός είναι επιτυχής, είτε όχι.

```
364 function showAllBombs() {
365     var allBombs = document.getElementsByClassName('isBomb');
366     var allBombsLen = allBombs.length;
367
368     gameIsOn = false;
369     for (var i = 0; i < allBombsLen; i++) {
370         var theImgClass = allBombs[i].childNodes[0].className;
371         if (theImgClass.indexOf('flagImage') >= 0) {
372             allBombs[i].innerHTML = '<img' +
373                 ' src="images/found.jpg"' +
374                 ' class="cellImage bombImage"' +
375                 ' />';
376         } else {
377             allBombs[i].innerHTML = '<img' +
378                 ' src="images/bomb.jpg"' +
379                 ' class="cellImage bombImage"' +
380                 ' />';
381         }
382     }
383 }
```

Στη πρώτη γραμμή της η ρουτίνα αναζητά και πάλι μέσω της **getElementsByClassName()** όπως και προηγουμένως, τα αντικείμενα όλων των στοιχείων που η κλάση τους περιλαμβάνει το **isBomb**.

Αμέσως μετά θέτει στον δείκτη **gameIsOn** την τιμή *false* ορίζοντας έτσι ότι το παιχνίδι πλέον δεν είναι σε εξέλιξη, αλλά τερματίστηκε. Με την αλλαγή της τιμής του δείκτη αυτού, οι ρουτίνες **leftClickOnCell()** και **rightClickOnCell()** θα σταματήσουν πλέον να εκτελούνται, καθώς ελέγχουν την τιμή του δείκτη αυτού στην αρχή τους και προχωράνε στην εκτέλεση των εντολών τους μόνο εάν ο δείκτης αυτός έχει την τιμή *true*.

Στη συνέχεια σαρώνονται όπως και πριν όλα τα αντικείμενα του πίνακα που μας επέστρεψε η **getElementsByClassName()** και ελέγχεται αν η κλάση της εικόνας που περιέχουν (την οποία παίρνουμε και πάλι μέσω του πρώτου αντικειμένου του πίνακα **childNodes[]** και ελέγχουμε μέσω της ιδιότητας **className**) περιλαμβάνουν την κλάση **flagImage**.

Σε κάθε κελί του οποίου η εικόνα περιέχει την παραπάνω κλάση, αντικαθιστάται η εικόνα της σημαίας *flag.jpg* με την *found.jpg*, η οποία είναι μία ασημένια βόμβα, δηλώνοντας έτσι ότι η συγκεκριμένη βόμβα έχει σωστά βρεθεί και μαρκαριστεί από τον παίκτη, ενώ κάθε κελί του οποίου η εικόνα δεν περιέχει την κλάση **flagImage**, αντικαθιστάται η τρέχουσα εικόνα με την *bomb.jpg*, η οποία είναι η κόκκινη βόμβα, δηλώνοντας έτσι ότι η συγκεκριμένη βόμβα δεν βρέθηκε τελικά από τον παίκτη.

Ρουτίνα saveStatistics()

Η ρουτίνα αυτή καλείται από την **rightClickOnCell()** εφόσον διαπιστωθεί ότι όλες οι βόμβες έχουν βρεθεί και μαρκαριστεί σωστά, άρα το παιχνίδι ολοκληρώθηκε με επιτυχία από τον παίκτη.

```
385 function saveStatistics() {
386     historyArray.push(
387         getGameLevelHiddenString(gameCols, gameRows, gameBombs) +
388         document.getElementById('gameTime').value +
389         ' - ' + gameCols + 'X' + gameRows + ' με ' + gameBombs + ' βόμβες' +
390         ' (Δ ' + getGameLevelString(gameCols, gameRows, gameBombs) + ')'
391     );
392     historyArray.sort();
393     var historyCookie = historyArray.slice(0, 20).join('<br>');
394     createCookie('ChrisMinesGame', historyCookie, 5000);
395 }
```

Είναι υπεύθυνη για την αποθήκευση του ιστορικού των παιχνιδιών του χρήστη στο ανάλογο *cookie* του περιηγητή.

Στη ρουτίνα **getStatistics()** η οποία καλείται από την **startUp()** είδαμε ότι η καθολική μεταβλητή **historyArray** κρατάει σε μορφή πίνακα τα παιχνίδια του ιστορικού.

Μέσω της μεθόδου **push()** τώρα, προσθέτουμε ένα ακόμα παιχνίδι στον πίνακα – λίστα.

Αξίζει όμως να αναλύσουμε λίγο το λεκτικό του κάθε παιχνιδιού, όπως αυτό διαμορφώνεται πριν την αποθήκευσή του και που μας χρησιμεύει τελικά η μορφή αυτή.

Το πρώτο μέρος του λεκτικού αυτού αποτελείται από την επιστροφή της εκτέλεσης της ρουτίνας **getGameLevelHiddenString()**.

Το πρώτο αυτό λεκτικό περιέχει αρχικά τη δήλωση ενός στοιχείου *…* με κλάση *hiddenItem* και περιεχόμενο τη διαφορά του επιπέδου δυσκολίας από το 100. Αυτό σημαίνει πως όσο μεγαλύτερο είναι το επίπεδο, τόσο μικρότερος ο αριθμός μέσα στο στοιχείο **.

Το υπόλοιπο μέρος του λεκτικού αποτελείται από τον χρόνο του παιχνιδιού, τον οποίο παίρνουμε από την ιδιότητα *value* του αντικειμένου του στοιχείου *<input>* **gameTime** και το οποίο

ενημερώνεται κατά τη διάρκεια του παιχνιδιού από την επαναλαμβανόμενη εκτέλεση της ρουτίνας **displayTheTime()**, τις τιμές των καθολικών μεταβλητών **gameCols**, **gameRows** και **gameBombs** που περιέχουν τις στήλες, γραμμές και βόμβες του παιχνιδιού, καθώς και το λεκτικό επιπέδου δυσκολίας, το οποίο παίρνομε από την εκτέλεση της ρουτίνας **getGameLevelString()**.

Στην επόμενη εντολή καλούμε τη μέθοδο **sort()** του πίνακα **historyArray**. Με την κλήση αυτής της μεθόδου, προκαλούμε τον πίνακα να κατατάξει τα περιεχόμενά του κατά αύξουσα σειρά.

Καθώς λοιπόν το πρώτο μέρος κάθε λεκτικού του πίνακα περιέχει τη διαφορά του επιπέδου δυσκολίας από το εκατό, όσο πιο δύσκολο το παιχνίδι που έχουμε καταγράψει στο ιστορικό τόσο πιο πάνω θα μπει στις θέσεις του πίνακα. Στη συνέχεια, επειδή το επόμενο μέρος του λεκτικού περιλαμβάνει το χρόνο που παρήλθε και πάλι όσο μικρότερος ο χρόνος τόσο πιο ψηλά θα ανέβει το παιχνίδι αυτό, αλλά πάντα θα είναι πιο κάτω από ένα δυσκολότερο παιχνίδι.

Για το τέλος να σημειώσουμε ότι η κλάση **hiddenItem** του στοιχείου ** που περιλαμβάνει τη διαφορά επιπέδου, λόγω της δήλωσης στο αρχείο *common.css*

```
43  .hiddenItem {
44      display: none;
45  }
```

κάνει το στοιχείο αυτό ** αόρατο κατά την εμφάνισή του μέσα στον πίνακα του ιστορικού!

Με τον τρόπο λοιπόν αυτό καταφέραμε να εμφανίσουμε το ιστορικό των παιχνιδιών ανά χρόνο που παρήλθε, όπου όσο πιο γρήγορα τελείωσε ένα παιχνίδι, τόσο σε πιο ψηλή θέση εμφανίζεται, αλλά συγχρόνως, όσο πιο δύσκολο είναι ένα παιχνίδι, άσχετα με το αν πήρε περισσότερο χρόνο να τελειώσει, να εμφανίζεται πιο ψηλά από κατώτερου επιπέδου δυσκολίας παιχνίδια!

Στη συνέχεια του κώδικα κρατάμε μόνο τα είκοσι πρώτα παιχνίδια - εγγραφές του πίνακα.

Ο λόγος που δεν κρατάμε όλα τα παιχνίδια είναι πρώτον αισθητικός, καθώς μία λίστα με περισσότερα παιχνίδια θα είναι λογικά μεγαλύτερη και θα προκαλούσε τον πίνακα αυτόν να είναι πολύ ψηλότερος από αυτόν του παιχνιδιού.

Ο δεύτερος λόγος είναι ότι το τελικό λεκτικό που κρατάει το ιστορικό και τελικά αποθηκεύεται σε cookie του περιηγητή, καλό είναι να έχει ελεγχόμενο από το πρόγραμμα μήκος, καθώς εάν το αφήσουμε να μεγαλώνει ανεξέλεγκτα, προσθέτοντάς του συνεχώς νέα παιχνίδια, υπάρχει κίνδυνος να γίνει τόσο μεγάλο που κάποιος περιηγητής να μην μπορεί να το φορτώσει ή επεξεργαστεί και το πρόγραμμα να καταρρεύσει κατά την εκτέλεσή του.

Τον περιορισμό λοιπόν αυτό τον επιτυγχάνουμε με τη χρήση την μεθόδου **slice()** αρχικά, η οποία θα μας επιστρέψει ένα νέο πίνακα από τη θέση μηδέν, μέχρι και τη θέση δεκαεννέα του πίνακα (σύνολο δηλαδή είκοσι παιχνίδια).

Στη συνέχεια και με τη χρήση της μεθόδου **join()** ενώνουμε όλες τις γραμμές του νέου αυτού πίνακα σε ένα τελικό μεγάλο λεκτικό, χρησιμοποιώντας ως ενωτικό το λεκτικό *
*, το οποίο είναι μία αλλαγή γραμμής σε κώδικα HTML.

Τέλος με την κλήση της ρουτίνας **createCookie()** αποθηκεύουμε το τελικό αυτό λεκτικό στο κατάλληλο *cookie* του περιηγητή μας, ώστε να διαβαστεί μέσω της ρουτίνας **getStatistics()** την επόμενη φορά που μεταβούμε στην αρχική σελίδα της εφαρμογής μας.

Ρουτίνα getGameLevelHiddenString()

Η βασική ανάλυση της ρουτίνας αυτής έγινε ήδη κατά την ανάλυση της ρουτίνας **saveStatistics()**.

```
397 function getGameLevelHiddenString(cols, rows, bombs) {
398     var level = getGameLevel(cols, rows, bombs);
399     level = 100 - level;
400     level = (level < 10) ? '0' + level : level;
401     return '<span class="hiddenItem">' + level + '</span>';
402 }
```

Όπως λοιπόν ήδη αναφέρθηκε, η ρουτίνα αυτή δημιουργεί και επιστρέφει στην **saveStatistics()** το πρώτο μέρος του λεκτικού με τα στοιχεία του παιχνιδιού, το οποίο τελικά θα αποθηκευτεί στο ιστορικό.

Σαν πρώτο βήμα καλείται η ρουτίνα **getGameLevel()** στέλνοντάς της ως ορίσματα, τα ορίσματα κλήσης της τρέχουσας ρουτίνας.

Ο αριθμός αυτός του επιπέδου δυσκολίας, αφαιρείται στη συνέχεια από το εκατό, ώστε να μας δώσει τη διαφορά και να το μετατρέψουμε έτσι σε κατά κάποιο τρόπο ανάποδο επίπεδο δυσκολίας (μικρότερος αριθμός όσο υψηλότερο είναι το επίπεδο).

Στο τέλος, αφού του προσθέσουμε το χαρακτήρα μηδέν στην αρχή, στην περίπτωση που ο αριθμός αυτός είναι μικρότερος του δέκα, τον επιστρέφουμε μαζί με την HTML δήλωση και κλάση του στοιχείου ** στην αρχή αλλά και το τέλος του λεκτικού.

Ρουτίνα createCookie()

Η ρουτίνα αυτή καλείται από την **saveStatistics()** και αναλαμβάνει να σώσει το ιστορικό τω παιχνιδιών σε ένα *cookie* του περιηγητή.

```
404 function createCookie(name, value, days) {
405     if (days) {
406         var date = new Date();
407         date.setTime(date.getTime() + (days * 24 * 60 * 60 * 1000));
408         var expires = '; expires=' + date.toGMTString();
409     } else {
410         var expires = '';
411     }
412     document.cookie = name + '=' + value + expires + '; path=/';
413 }
```

Κάθε *cookie* που αποθηκεύει ο περιηγητής χρειάζεται τέσσερα στοιχεία.

Το όνομα, τα δεδομένα, το χρόνο διατήρησης και το φάκελο του domain στο οποίο ανήκει.

Τα δύο πρώτα είναι σαφή ως προς τη χρήση τους.

Το τρίτο δηλώνει στον περιηγητή πόσες για μέρες να θεωρεί το *cookie* αυτό ενεργό και κατά συνέπεια να μπορεί η σελίδα να το διαβάσει και γενικότερα να το διαχειριστή. Μετά το πέρας των ημερών αυτών, το *cookie* αυτό θεωρείται ληγμένο από τον περιηγητή και διαγράφεται.

Εάν δε δηλωθεί η παράμετρος αυτή, τότε το *cookie* θα λήξει αμέσως μόλις κλείσει η τρέχουσα σελίδα.

Το τέταρτο και τελευταίο ορίζει το φάκελο του domain για τον οποίο το *cookie* αυτό θα είναι ενεργό.

Αν λοιπόν δηλώσουμε ότι ο φάκελος ενός *cookie* είναι ο αρχικός (/), τιμή που θεωρείται και ως προεπιλεγμένη, τότε σε κάθε αρχείο το οποίο φορτώνεται από την αρχικό φάκελο και κάτω το *cookie* αυτό θα είναι ενεργό. Αν όμως δηλώσουμε ότι ανήκει στο φάκελο για παράδειγμα *alpha*, τότε μόνο για κάθε σελίδα που φορτώνεται από το φάκελο *alpha* και κάτω θα είναι ενεργό.

Έτσι λοιπόν ο πρώτος (και μοναδικός) έλεγχος που κάνει η ρουτίνα αυτή είναι εάν στα ορίσματα έχει δηλωθεί διάρκεια σε μέρες.

Εάν ο έλεγχος είναι αληθής, τότε κρατάει την τρέχουσα ημερομηνία και ώρα μέσω της **new Date()** και στη συνέχεια, στην ημερομηνία αυτή προσθέτει τις ημέρες του ορίσματος με χρήση της μεθόδου **setTime()**, αφού πρώτα μετατρέψει τις ημέρες σε milliseconds.

Στη συνέχεια δομεί το λεκτικό που θα χρησιμοποιήσει κατά τη δημιουργία του *cookie* προσθέτοντάς του το κατάλληλο λεκτικό (*expires=*).

Τέλος μέσω κλήσης της ρουτίνας **cookie** δημιουργεί ή μεταβάλει αν υπάρχει ήδη το *cookie* στον περιηγητή, κάνοντας χρήση των δύο πρώτων ορισμάτων (όνομα και τιμή του *cookie*) καθώς και της τιμής του χρόνου λήξης (εάν αυτό υπάρχει) και προσθέτοντας στο τέλος τη δήλωση ότι το *cookie* ανήκει σε κάθε σελίδα από τον αρχικό φάκελο και κάτω (*path=/*).

Ρουτίνα updateLevel()

Η ρουτίνα αυτή είναι επίσης μία αυτές που καλούνται μέσω κάποιου συμβάντος από το χρήστη.

Η συγκεκριμένη καλείται όταν ο χρήστης αλλάξει τα περιεχόμενα ενός από τα τρία πεδία *<input>* που ορίζουν τα χαρακτηριστικά του εξατομικευμένου επιπέδου δυσκολίας.

```
415 function updateLevel() {
416     var cols = parseInt(document.getElementById('gameCols').value);
417     var rows = parseInt(document.getElementById('gameRows').value);
418     var bombs = parseInt(document.getElementById('gameBombs').value);
419     var level = (cols * rows * bombs > 0) ? getGameLevelString(cols, rows, bombs) : '';
420     document.getElementById('gameLevel').value = level + ' - ' + getGameType(cols, rows, bombs);
421 }
```

Κατά την εκτέλεση της ρουτίνας, αρχικά μέσω της ιδιότητας *value* διαβάζουμε τα περιεχόμενα των τριών πεδίων *<input>* **gameCols**, **gameRows** και **gameBombs**, τα οποία αφού μετατρέψουμε σε ακέραιο αριθμό μέσω της **parseInt()**, τα αποθηκεύουμε στις αντίστοιχες τοπικές μεταβλητές.

Στη συνέχεια με κλήση της ρουτίνας **getGameType()** και περνώντας της ως ορίσματα τις τρεις αυτές τοπικές μεταβλητές, παίρνουμε το λεκτικό του επιπέδου δυσκολίας του παιχνιδιού.

Τέλος ενημερώνουμε το πεδίο *<input>* **gameLevel** μέσω της ιδιότητάς του ώστε να εμφανίσει το νέο λεκτικό επιπέδου.

Ρουτίνα eraseCookie()

Η τελευταία ρουτίνα είναι και αυτή μία από τις οποίες καλούνται μέσω κάποιου συμβάντος από το χρήστη.

Η συγκεκριμένη καλείται όταν ο χρήστης κάνει *click*, πατήσει δηλαδή το κουμπί *<button>* **resetHistory**.

```
423 function eraseCookie(name) {
424     createCookie(name, '', -1);
425     location.reload();
426 }
```

Δέχεται ως όρισμα το όνομα του *cookie* το οποίο καλείται να διαγράψει.

Καλώντας την **createCookie()** που αναλύσαμε παραπάνω και στέλνοντάς της ως ορίσματα το όνομα του *cookie*, στη θέση της τιμής ένα κενό λεκτικό και στη θέση της διάρκειας το μείον ένα, το συγκεκριμένο *cookie* διαγράφεται από τον περιηγητή.

Τέλος προκαλεί τον περιηγητή να κάνει επαναφόρτωση της σελίδας μέσω της **location.reload()**.

Συμπεράσματα

Η υλοποίηση του παιχνιδιού δεν ήταν μία απλή υπόθεση.

Χρειάστηκε μελέτη και ανάλυση της λογικής του παιχνιδιού. Μελετήθηκαν οι αντιδράσεις του παιχνιδιού, όπως αυτό υπάρχει σε περιβάλλον Windows, αλλά και κάποιων σε online μορφή.

Σχεδιάστηκαν τα γραφικά στοιχεία του (κουμπιά, βόμβες κλπ) με τη χρήση του Photoshop, ώστε να δοθεί όσο το δυνατόν μία εικόνα που να θυμίζει μεν το παιχνίδι της Microsoft, αλλά συγχρόνως να είναι διαφορετικό και να έχει τη δική του αισθητική.

Αναλύοντάς το και παίζοντας, τελειοποιήθηκε και υλοποιήθηκε η ιδέα των επιπέδων, που ήταν και ο αρχικός στόχος διαφοροποίησης και αναβάθμισης του τελικού προϊόντος.

Στο κάτω δεξιά μέρος της οθόνης, αναγράφεται ότι η τρέχουσα έκδοση είναι η 4.1. Το νούμερο αυτό δείχνει ότι η υλοποίηση πέρασε από πολλά στάδια, αλλαγές και βελτιώσεις. Το πρώτο νούμερο (το 4) δείχνει ότι τέσσερις μεγάλες – βασικές εκδόσεις φτιάχτηκαν. Κάθε μία από αυτές ήταν τελείως διαφορετική από την προηγούμενη, αισθητικά αλλά και λειτουργικά. Το δεύτερο δείχνει τις υπο-εκδόσεις κάθε κύριας, που ήταν κυρίως διορθωτικές.

Στη πρώτη βασική έκδοση, το παιχνίδι υλοποιήθηκε καθαρά και μόνο μέσω πινάκων και κελιών, χωρίς γραφικά στοιχεία και η λειτουργικότητά του ήταν μόνο δοκιμαστική.

Στη δεύτερη βασική έκδοση προστέθηκαν τα γραφικά στοιχεία και υλοποιήθηκε ο κώδικας ελέγχου των συμβάντων click και δεξί click του χρήστη. Στην έκδοση αυτή ο παίκτης μπορούσε να “παίξει” το παιχνίδι, δεν υπήρχαν όμως έλεγχοι επιτυχής ή όχι ολοκλήρωσης του παιχνιδιού.

Η τρίτη βασική έκδοση ήταν αυτή με τις περισσότερες υποεκδόσεις (12 συνολικά). Σε αυτήν υλοποιήθηκε ο κύριος και τελικός κώδικας του παιχνιδιού. Υλοποιήθηκαν όλοι οι έλεγχοι για την επιτυχή ή όχι ολοκλήρωση του παιχνιδιού. Τελειοποιήθηκαν μέσα από μία μακρά και επίπονη διαδικασία ελέγχου, υλοποίησης, δοκιμής και ξανά ελέγχου και αναζήτησης των σφαλμάτων ή και βελτίωσης των λειτουργιών κάθε ρουτίνας. Δημιουργήθηκαν νέες, διασπάστηκαν όσες κρίθηκε ότι ήταν πολύ μεγάλες και διαγράφτηκαν παλαιές.

Το δυσκολότερο ίσως κομμάτι της διαδικασίας αυτής ήταν η εύρεση των λογικών σφαλμάτων και όχι τόσο των σφαλμάτων του ίδιου του κώδικα.

Ένα σημαντικό εργαλείο της αποσφαλμάτωσης ήταν το Firebug, ένα πρόσθετο για τον περιηγητή Firefox της Mozilla, μέσω του οποίου οριζόντουσαν σημεία παύσης εκτέλεσης του κώδικα JavaScript, ελέγχου των τιμών των μεταβλητών και ανάλυση της λογικής και χρόνου εκτέλεσης κάθε εντολής.

Ένα άλλο επίσης χρήσιμο στοιχείο του συγκεκριμένου πρόσθετου, ήταν η εκπληκτική σε αποτέλεσμα και φιλικότατη σε χρήση δυνατότητα, αλλαγής των τιμών κάθε δήλωσης css, ώστε να ελεγχθεί σε πραγματικές συνθήκες το τελικό αισθητικό, εικαστικό αλλά και λειτουργικό αποτέλεσμα.

Στην τέταρτη και τελευταία έκδοση υλοποιήθηκε το κομμάτι της καταμέτρησης του χρόνου, της εμφάνισης και αποθήκευσης του ιστορικού, καθώς και του εικαστικού της χρωματικής μεταβολής του υπόβαθρου.

Σαφώς ο κώδικας για το παιχνίδι αυτό επιδέχεται διορθώσεις και βελτιώσεις, όπως κάθε κώδικας άλλωστε. Όσο κάποιος παίζει και μελετάει παράλληλα τον κώδικα, θα έχει νέες ιδέες που θα βελτιώνουν σε ταχύτητα, στην εικόνα και σε λειτουργίες το τελικό αποτέλεσμα.

Όμως ο αρχικός στόχος επετεύχθη. Φτιάχτηκε σε καθαρή JavaScript, χωρίς την κλήση καμίας πρόσθετης βιβλιοθήκης (όπως για παράδειγμα η jQuery), ένα πλήρως λειτουργικό και αρκετά ευχάριστο μπορώ να πω παιχνίδι, το οποίο τρέχει είτε ανεβεί και καλεστεί από κάποιο server είτε και σε τοπικό επίπεδο. Μπορεί να εκτελεστεί από οποιαδήποτε συσκευή, είτε σταθερό υπολογιστή και laptop, είτε tablet και κινητό τηλέφωνο και ο χειρισμός του μπορεί να γίνει είτε από ποντίκι είναι μέσω μίας οθόνης αφής.

Πηγές - Βιβλιογραφία

D. Goodman, Javascript and DHTML Cookbook, O'Reilly, 2003

S. Koch, Javascript, Wiley, 2003

D. Goodman, M. Morrison, JavaScript Bible, Wiley, 2010

JavaScript Tutorial - http://www.w3schools.com/

http://stackoverflow.com/questions/tagged/javascript

https://www.tutorialspoint.com/javascript/

https://jsfiddle.net/

Colorful Gradients - http://colorfulgradients.tumblr.com