Ανοικτό Πανεπιστήμιο Κύπρου

Σχολή Θετικών και Εφαρμοσμένων Επιστημών

Μεταπτυχιακή Διατριβή
στα Πληροφοριακά και Επικοινωνιακά Συστήματα

Μεταφορά και Επέκταση της Εφαρμογής RLTankAttack σε Πλατφόρμα Android

Δημήτριος Κεσίδης

Επιβλέπων Καθηγητής
Δημήτριος Καλλές

Μήνας Έτος

Ανοικτό Πανεπιστήμιο Κύπρου

Σχολή Θετικών και Εφαρμοσμένων Επιστημών

Μεταφορά και Επέκταση της Εφαρμογής RLTankAttack σε Πλατφόρμα Android

Δημήτριος Κεσίδης

Επιβλέπων Καθηγητής
Δημήτριος Καλλές

Η παρούσα μεταπτυχιακή διατριβή υποβλήθηκε
προς μερική εκπλήρωση των απαιτήσεων για απόκτηση

μεταπτυχιακού τίτλου σπουδών
στα Πληροφοριακά Συστήματα

από τη Σχολή Θετικών και Εφαρμοσμένων Επιστημών
του Ανοικτού Πανεπιστημίου Κύπρου

Μήνας Έτος

Περίληψη

Η παρούσα Μεταπτυχιακή Διατριβή γίνεται στα πλαίσια επέκτασης της Μεταπτυχιακής Διατριβής του κ. Γκεζερλή Σπύρου με θέμα «Μηχανισμοί Μάθησης και η Χρήση τους στην Επικύρωση της Ποιότητας Νέων Παιγνίων», Αύγουστος 2012.

Στην εν λόγω εργασία επιχειρήθηκε να σχεδιαστεί ένα νέο παιχνίδι μεταξύ δυο αντιπάλων, ανταγωνιστικό, μηδενικού αθροίσματος. Τα χαρακτηριστικά αυτά επιλέχθηκαν γιατί μπορούν να εφαρμοστούν σε λήψεις στρατηγικών αποφάσεων οικονομικής και εμπορικής φύσης, σε μια επιχείρηση, έναν οργανισμό ή ακόμα και ένα κράτος.

Η έννοια παίγνιο μηδενικού αθροίσματος αναφέρεται στην κατάσταση εκείνη όπου τα κέρδη της απόφασης ενός παίκτη Α επηρεάζουν στον ίδιο βαθμό τις ζημιές του παίκτη Β. Σαν παράδειγμα μπορούμε να αναφέρουμε την εισαγωγή ενός νέου προϊόντος στην αγορά από κάποια εταιρία, όπου με αύξηση των πωλήσεων κατά 30%, αυτόματα σημειώνεται ανάλογη μείωση των πωλήσεων για τις υπόλοιπες ανταγωνιστικές εταιρείες.

Σκοπός ενός τέτοιου επιχειρήματος είναι η αξιοποίηση υπάρχουσας τεχνογνωσίας και τεχνολογίας πάνω στην χρήση της ενισχυτικής μάθησης για την αυτόματη ανακάλυψη τακτικής σε ένα παιχνίδι, ώστε ο μηχανισμός μάθησης να μπορεί να επαναχρησιμοποιηθεί.

Αρχικά ο στόχος που έχει τεθεί με την εκπόνηση της Μεταπτυχιακής Διατριβής είναι να υλοποιηθεί το όλο εγχείρημα σε πλατφόρμα Android όπως επίσης και να δίνεται η δυνατότητα σε παίκτη-άνθρωπο να ανταγωνίζεται πράκτορα-υπολογιστή.

Ευχαριστίες

Θα ήθελα να ευχαριστήσω θερμά τον καθηγητή μου κ. Καλλέ Δημητριο για την εμπιστοσύνη που μου έδειξε αναθέτοντας μου την παρούσα Μεταπτυχιακή Διατριβή, καθώς και για την καθοδήγηση και την πολύτιμη βοήθεια του καθ' όλη τη διάρκεια της εκπόνησης της.

Επίσης θα ήθελα να ευχαριστήσω και να αφιερώσω τον Τίτλο Σπουδών στη σύντροφο μου Αναστασία, όπου με την υποστήριξη και την συμπαράσταση της, με βοήθησε να ξεπεράσω κάθε εμπόδιο και να ολοκληρώσω το εγχείρημα.

Περιεχόμενα

Κεφάλαιο 1
Εισαγωγή

Τα τελευταία χρόνια έχουν αυξηθεί σημαντικά οι πωλήσεις έξυπνων συσκευών κινητής τηλεφωνίας λόγω του προσιτού κόστους και σαν συνεπακόλουθο αύξηση παρατηρείται και στη χρήση εφαρμογών κινητής τηλεφωνίας, από χρήστες όλων των ηλικιών και κοινωνικών στρωμάτων.

Τα είδη των εφαρμογών που χρησιμοποιούνται καταλαμβάνουν ένα ευρύ φάσμα, όπως εφαρμογές διασκέδασης, κοινωνικής δικτύωσης, εκπαιδευτικές, εμπορικές, ενημερωτικές, μέχρι προληπτικές ιατρικές εφαρμογές και παρακολούθησής ασθενών.

1.1 Ανάληψη Διπλωματικής Εργασίας

Με βάση τα παραπάνω και σε συνεργασία με τον υπεύθυνο – επιβλέποντα κ. Καλλέ αποφασίσαμε να επεκτείνουμε την Μεταπτυχιακή Διατριβή του κ. Γκεζερλή με θέμα «Μηχανισμοί Μάθησης και η Χρήση τους στην Επικύρωση της Ποιότητας Νέων Παιγνίων» , Αύγουστος 2012.

Στην εν λόγω εργασία επιχειρήθηκε να σχεδιαστεί ένα νέο παιχνίδι μεταξύ δυο αντιπάλων, ανταγωνιστικό, μηδενικού αθροίσματος. Τα χαρακτηριστικά αυτά επιλέχθηκαν γιατί μπορούν να

εφαρμοστούν σε λήψεις στρατηγικών αποφάσεων οικονομικής και εμπορικής φύσης, σε μια επιχείρηση, έναν οργανισμό ή ακόμα και ένα κράτος.

Η έννοια παίγνιο μηδενικού αθροίσματος αναφέρεται στην κατάσταση εκείνη όπου τα κέρδη της απόφασης ενός παίκτη Α επηρεάζουν στον ίδιο βαθμό τις ζημιές του παίκτη Β. Σαν παράδειγμα μπορούμε να αναφέρουμε την εισαγωγή ενός νέου προϊόντος στην αγορά από κάποια εταιρία, όπου με αύξηση των πωλήσεων κατά 30%, αυτόματα σημειώνεται ανάλογη μείωση των πωλήσεων για τις υπόλοιπες ανταγωνιστικές εταιρείες.

Σκοπός ενός τέτοιου επιχειρήματος είναι η αξιοποίηση υπάρχουσας τεχνογνωσίας και τεχνολογίας πάνω στην χρήση της ενισχυτικής μάθησης για την αυτόματη ανακάλυψη τακτικής σε ένα παιχνίδι, ώστε ο μηχανισμός μάθησης να μπορεί να επαναχρησιμοποιηθεί.

Αρχικά ο στόχος που έχει τεθεί με την εκπόνηση της Μεταπτυχιακής Διατριβής είναι να υλοποιηθεί το όλο εγχείρημα σε πλατφόρμα Android όπως επίσης και να δίνεται η δυνατότητα σε παίκτη-άνθρωπο να ανταγωνίζεται πράκτορα-υπολογιστή.

1.2 Minimax

Προκειμένου να δοθεί η δυνατότητα σε παίκτη-υπολογιστή να λαμβάνει στρατηγικές αποφάσεις κινήσεων βασιζόμενες σε λογική, χρησιμοποιήθηκε ο αλγόριθμος Minimax. Ο αλγόριθμος αυτός εφαρμόζει σε παιχνίδια δυο αντιπάλων, όπου ο κάθε αντίπαλος έχει όλη την πληροφορία για την έκβαση του παιχνιδιού και οι κινήσεις πραγματοποιούνται εναλλάξ. Παραδείγματα τέτοιων παιχνιδιών είναι το σκάκι, η ντάμα, η τρίλιζα και άλλα.

Η διαδικασία που ακολουθεί ο αλγόριθμος ώστε να αποφασίσει την κίνηση του είναι η εξής: Λαμβάνοντας υπόψη την παρούσα κατάσταση του παιχνιδιού, εξετάζει όλες τις πιθανές εκβάσεις που μπορούν να δημιουργηθούν για έναν ορισμένο αριθμό επόμενων κινήσεων. Εκτελώντας μια συνάρτηση υπολογισμού βαθμολογίας για κάθε πιθανή έκβαση, επιλέγει την καλύτερη δυνατή. Η διαδικασία αυτή επαναλαμβάνεται μέχρι το πέρας του παιχνιδιού.

Για να εφαρμοστεί ο αλγόριθμος πρέπει να δημιουργηθεί το δέντρο του παιχνιδιού όπου τα φύλλα αναπαριστούν όλες τις πιθανές εκβάσεις του παιχνιδιού με την αντίστοιχη βαθμολογία για κάθε περίπτωση, ενώ εξετάζοντας κάθε επίπεδο προς τη ρίζα βλέπουμε τις πιθανές εναλλαγές κινήσεων μεταξύ των παικτών. Τέλος στη ρίζα εμφανίζεται η τελική κίνηση που είναι η πλέον συμφέρουσα.

Κεφάλαιο 2
Εργαλεία Ανάπτυξης

Στο παρόν κεφάλαιο θα παρουσιάσουμε τα εργαλεία που χρησιμοποιήθηκαν για την ανάπτυξη της εφαρμογής.

2.1 Android

Με τον όρο Android εννοούμε το Λειτουργικό Σύστημα το οποίο αρχικά δημιουργήθηκε για συσκευές κινητής τηλεφωνίας και πλέον μπορεί να υποστηρίξει μια ευρεία γκάμα συσκευών, όπως είναι τα tablets, οι τηλεοράσεις, ρολόγια χειρός κ.α.

Η ψυχή του Android είναι το λειτουργικό σύστημα Linux, από το οποίο έχει δανειστεί τον πυρήνα του, ενώ παράλληλα με τη βοήθεια βιβλιοθηκών λογισμικού που έχουν αναπτυχθεί από την Google και τη γλώσσα προγραμματισμού Java, δίνεται η δυνατότητα σε κατασκευαστές λογισμικού να αναπτύξουν εφαρμογές χωρίς κόστος και να τις διαθέσουν στο επίσημο σημείο διανομής εφαρμογών του Android, το Google Play, ή σε άλλα τρίτα σημεία, όπως είναι το SlideME, το Android Pit και το App Brain.

2.2 Αρχιτεκτονική Android

Η αρχιτεκτονική του Android περιλαμβάνει τα εξής 5 επίπεδα:

- Επίπεδο Εφαρμογών (Applications). Το Android περιλαμβάνει ένα σύνολο από βασικές εφαρμογές όπως περιηγητή ιστού, email client, ημερολόγιο, χάρτες (Google maps), ηχογράφηση και άλλα. Όλες οι εφαρμογές είναι γραμμένες στην γλώσσα προγραμματισμού Java.

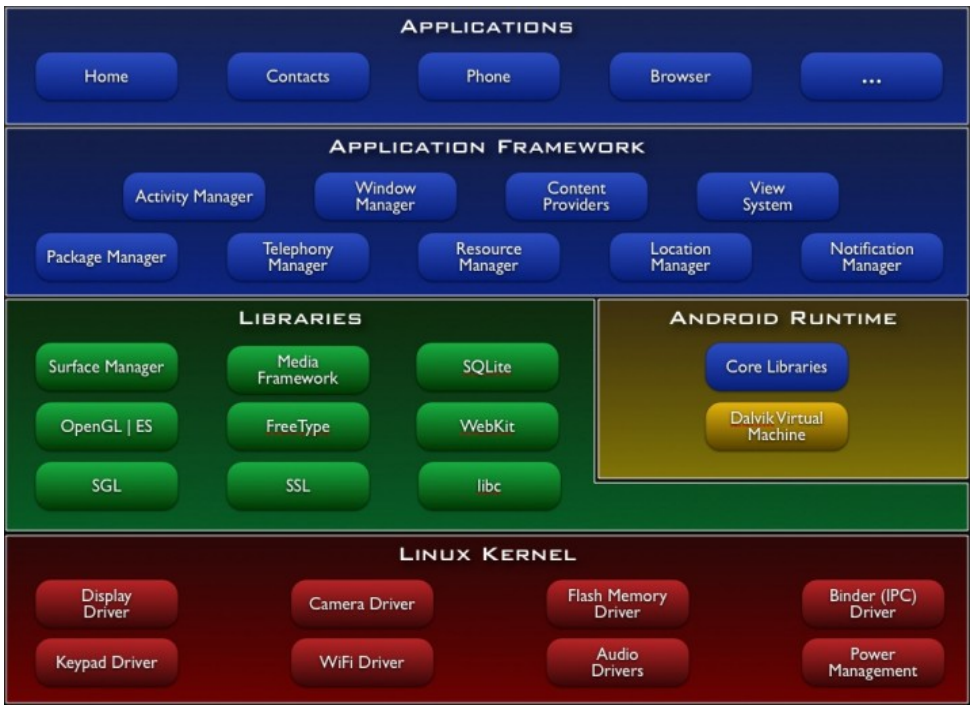
- Επίπεδο Πλαισίου Εφαρμογών (Applications Framework). Το Android προσφέρει στους προγραμματιστές την δυνατότητα να κατασκευάσουν πλούσιες και καινοτόμες εφαρμογές. Οι προγραμματιστές είναι ελεύθεροι να εκμεταλλευτούν πλήρως το υλικό της συσκευών, να έχουν πρόσβαση σε υπηρεσίες εντοπισμού θέσης, να θέσουν χρονοδιακόπτες για εμφάνιση ειδοποιήσεων και πολλά άλλα. Επίσης έχουν πλήρη πρόσβαση στο ίδιο πλαίσιο από APIs που έχουν οι βασικές εφαρμογές Android.

- Επίπεδο Βιβλιοθηκών (Libraries). Το επίπεδο αυτό περιλαμβάνει ένα σύνολο από βιβλιοθήκες γραμμένες σε C/C++ οι οποίες χρησιμοποιούνται από διάφορα στοιχεία του συστήματος Android και είναι διαθέσιμες για χρήση και από τους προγραμματιστές.

- Επίπεδο Εκτέλεσης (Android Runtime). Το οποίο αποτελείται από ένα σύνολο από βασικές βιβλιοθήκες και την Dalvik Virtual Machine.

- Πυρήνας του Linux. Το Android βασίζεται στον πυρήνα Linux έκδοση 2.6 ή νεότερο για βασικές υπηρεσίες συστήματος όπως ασφάλεια, διαχείριση μνήμης, διαχείριση διεργασιών, στοίβα δικτύου και οδηγούς συσκευών.

Στην εικόνα φαίνεται το διάγραμμα της αρχιτεκτονικής Android.



2.3 Eclipse

Πολύ σημαντικό εργαλείο για την ανάπτυξη εφαρμογών Android είναι το Eclipse, ένα ολοκληρωμένο περιβάλλον ανάπτυξης εφαρμογών, όπου μόλις εγκαταστήσουμε τις απαραίτητες προεκτάσεις, μας παρέχει δυνατότητες όπως είναι ο επεξεργαστής του πηγαίου κώδικα μας, μεταγλωττιστή ώστε να μετατρέπεται ο κώδικας σε μορφή κατάλληλη προς εκτέλεση από τον υπολογιστή , εικονικές συσκευές για να ελέγχουμε και να δοκιμάζουμε τις υπό ανάπτυξη εφαρμογές όπως επίσης και δυνατότητες αποσφαλμάτωσης των προγραμμάτων.

2.3.1 Android Development Tools (ADT)

Το Android Development Tools (ADT) είναι η απαραίτητη επέκτασης του Eclipse το οποίο έχει σχεδιαστεί ώστε να παρέχει ένα πλήρη και ενοποιημένο περιβάλλον ανάπτυξης Android εφαρμογών.

Επεκτείνει τις δυνατότητες του Eclipse ώστε εύκολα ακόμα και ένας αρχάριος προγραμματιστής να μπορεί να γράψει και να τρέξει την πρώτη του εφαρμογή. Μας παρέχει γραφικό περιβάλλον για τον εύκολο σχεδιασμό και τοποθέτηση στοιχείων ελέγχου χρήστη πάνω στην οθόνη και φτάνει μέχρι του σημείου εξαγωγής της τελικές εφαρμογής σε μορφή κατάλληλη για διάθεση στην αγορά.

2.3.2 Android Virtual Devices (AVD)

Οι συσκευές που μπορούν να φιλοξενήσουν το λειτουργικό σύστημα Android, θεωρητικά είναι άπειρες. Υπάρχουν διαφορετικές διαστάσεις οθονών, από ένα ρολόι χειρός λίγων εκατοστών, μέχρι μια οθόνη πολλών ιντσών. Επίσης υπάρχουν διάφορες δυνατότητες επεξεργασίας, μνήμης και αποθήκευσης δεδομένων. Για παράδειγμα αλλιώς συμπεριφέρεται ένα κινητό τηλέφωνο τελευταίας τεχνολογίας που είναι εφοδιασμένο με τετραπύρηνο επεξεργαστή και αρκετά MB μνήμης, σε σχέση κάποιο αντίστοιχο πρόγονο του.

Για τους παραπάνω λόγους και επειδή δεν είναι δυνατόν ένας κατασκευαστής λογισμικού να έχει στην κατοχή του εκατοντάδες ή χιλιάδες συσκευές για να δοκιμάζει τις εφαρμογές που αναπτύσσει, υπάρχει το Android Virtual Devices. Μέσω επιλογών αποφασίζουμε το είδος και τις δυνατότητες της συσκευής που θέλουμε να δοκιμάσουμε την εφαρμογή μας και εκτελώντας την εφαρμογή εμφανίζεται ένα εικονικό περιβάλλον συσκευής που συμπεριφέρεται παρόμοια με κάποια αντίστοιχη πραγματική συσκευή με τις δυνατότες που έχουμε εξ' αρχής ορίσει.

2.4 Java

Παρόλο που υπάρχει πολύ στενή σχέση των εφαρμογών του λειτουργικού συστήματος Android με την γλώσσα προγραμματισμού Java, εντούτοις είναι κάτι διαφορετικό και δεν θα πρέπει να τα συγχέουμε. Πράγματι μια εφαρμογή Android γράφεται σε Java, όμως διαφέρει η δομή του πηγαίου κώδικα όπως επίσης και ο τρόπος εκτέλεσης.

Κατά την μεταγλώττιση ενός προγράμματος Java, αρχικά δημιουργούνται τα αρχεία bytecode που χρησιμοποιούνται από τη Java Virtual Machine (JVM). Αν για παράδειγμα έχουμε δημιουργήσει το αρχείο goodmorning.java αυτό θα μετατραπεί σε goodmorning.class και θα είναι διαθέσιμο για εκτέλεση, σε κάθε πλατφόρμα που τρέχει Java.

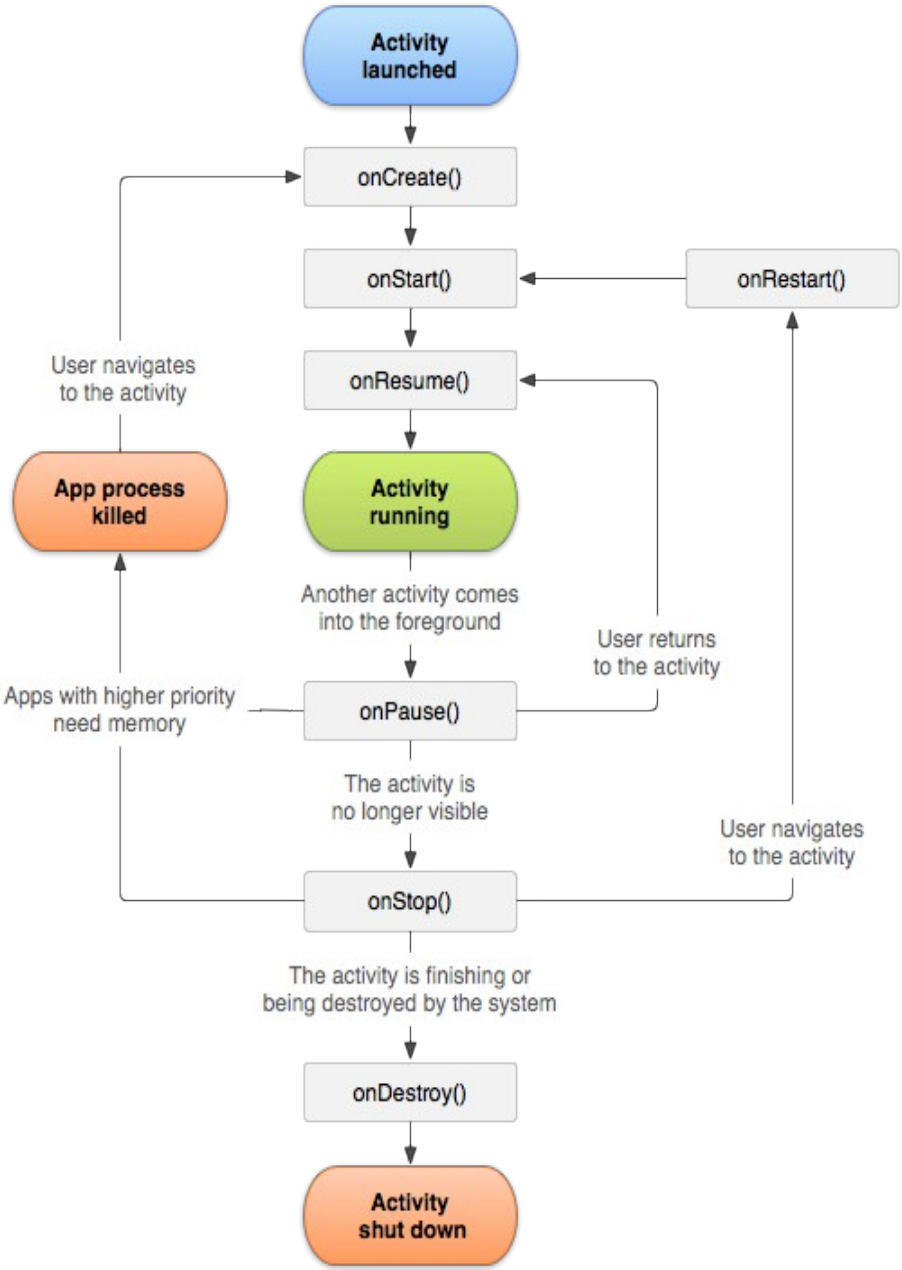
Από την άλλη πλευρά, κατά την μεταγλώττιση ενός αρχείου με πηγαίο κώδικα για Android, από τα αρχεία Java bytecode δημιουργείται το Dalvik bytecode, το οποίο έχει κατάληξη .dex και .odex και περιλαμβάνει όλες τις κλάσεις σε ένα και μόνο αρχείο, διαμορφωμένο έτσι ώστε να μπορεί μια συσκευή με περιορισμένες δυνατότητες σε σχέση με έναν υπολογιστή, να τρέξει χωρίς πρόβλημα το λογισμικό.

2.5 Κύκλος Ζωής Εφαρμογής Android

Μια Δραστηριότητα (Activity) σε εφαρμογή Android είναι μια πλατφόρμα όπου πάνω της στήνονται τα στοιχεία ελέγχου και επιτρέπουν στον τελικό χρήστη να αλληλεπιδρά με την εφαρμογή. Στην ουσία κάθε διαφορετική οθόνη που βλέπουμε, είναι μια δραστηριότητα (Activity).

Κατά την ανάπτυξη μιας εφαρμογής Android, υπάρχουν έτοιμες μέθοδοι που ενεργοποιούνται σε συγκεκριμένες συνθήκες και θα πρέπει να προσθέσουμε το περιεχόμενο τους ώστε να λειτουργούν με τρόπο που καλύπτει τις ανάγκες της εκάστοτε εφαρμογής.

Στο παρακάτω διάγραμμα φαίνεται σχηματικά η ροή των διαφορετικών καταστάσεων ενός κύκλου ζωής μιας εφαρμογής Android, καθώς και η σειρά με την οποία μπορεί να προκύψουν.



Οι μέθοδοι περιληπτικά είναι:

- onCreate() Είναι η πρώτη μέθοδος που καλείται μόλις δημιουργηθεί η δραστηριότητα. Συνήθως στη μέθοδο αυτή αρχικοποιούνται όλα τα στοιχεία που πρόκειται να χρησιμοποιηθούν στη συνέχεια.

- onStart() Καλείται μετά την onCreate() και στην ουσία μόλις εμφανιστεί η δραστηριότητα στον χρήστη.

- onResume() Καλείται μόλις ο χρήστης αρχίζει να αλληλεπιδρά με την εφαρμογή.

- onPause() Καλείται όταν σταματάει η αλληλεπίδραση με την εφαρμογή, είτε από επιλογή του χρήστη, είτε επειδή κάποια άλλη δραστηριότητα έρχεται στο προσκήνιο. Όπως για παράδειγμα όταν κάποιος μα καλεί

- onStop() Καλείται μετά την onPause() και όταν πλέον η δραστηριότητα δεν εμφανίζεται στο χρήστη.

- onRestart() Καλείται όταν η οθόνη εστιάζει και πάλι στη δραστηριότητα μετά από κάποια διακοπή

- onDestroy() Είναι η τελευταία μέθοδος που καλείται και τερματίζει την δραστηριότητα, είτε επειδή ολοκληρώθηκε η εκτέλεση της, είτε επειδή διακόπηκε απρόσκοπτα και απαιτείται μνήμη για την ομαλή λειτουργία του συστήματος.

Κεφάλαιο 3
Περιγραφή του Παιχνιδιού

3.1 Περιγραφή του Παιχνιδιού

Το παιχνίδι διαδραματίζεται σε μια στατική οθόνη, όπου ο χρήστης μπορεί να παρακολουθεί ταυτόχρονα όλους τους συντελεστές – αντικείμενα που το αποτελούν.

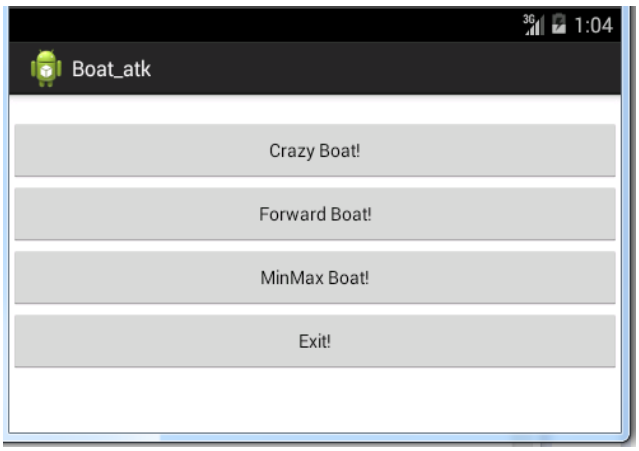
Οι αντίπαλοι είναι δυο καραβάκια, κόκκινου και πράσινου χρώματος. Το πεδίο μάχης είναι δεξαμενή με το νερό του οποίου η στάθμη σε κάθε στήλη είναι δυνατόν να διαφέρει. Η ιδέα για μια τέτοια υλοποίηση είναι η διώρυγα του Παναμά, όπου για να τη διασχίσουν τα πλωτά, η στάθμη του νερού αυξάνεται ή ελαττώνεται.

Έτσι συνοπτικά μπορούμε να ορίσουμε ότι το παιχνίδι αποτελείται από:

	Κόκκινο Καράβι	Ελέγχεται από τον άνθρωπο
	Πράσινο Καράβι	Είναι ο πράκτορας της εφαρμογής
	Δυο Καράβια	Εμφανίζεται όταν οι δυο παίκτες βρεθούν στο ίδιο κελί
	Τοίχος	Αποτελεί τα τοιχώματα δεξιά αριστερά και στο βυθό της δεξαμενής
	Νερό	Το νερό που υπάρχει μέσα στη δεξαμενή
	Κενό	Όπου δεν υπάρχει άλλο αντικείμενο, υπάρχει το κενό

Κατά την εκκίνηση του παιχνιδιού, εμφανίζεται ένα μενού το οποίο μας επιτρέπει να επιλέξουμε τον πράκτορα-υπολογιστή που θέλουμε να έχουμε αντίπαλο. Οι επιλογές είναι Crazy Boat, Forward Boat, MiniMax Boat και Exit για έξοδο από το παιχνίδι. Στη συνέχεια οι πράκτορες αυτοί θα επεξηγηθούν αναλυτικά.

Το screeshot του μενού επιλογών φαίνεται στην παρακάτω εικόνα:



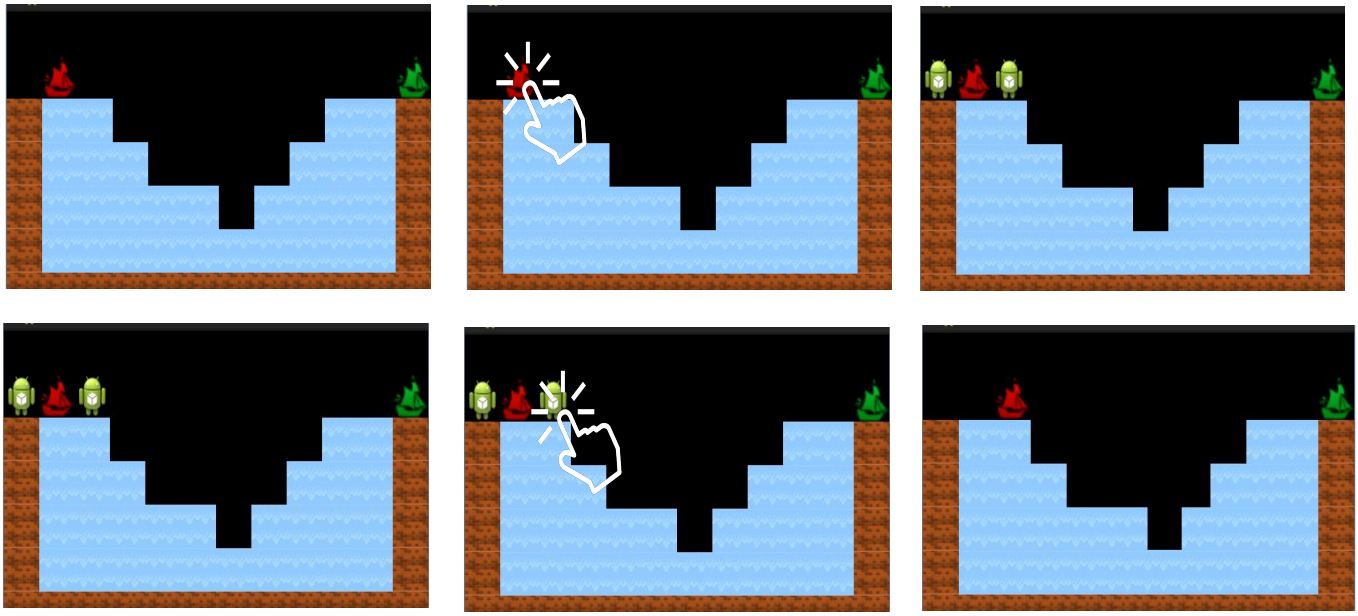
3.2 Κανόνες του Παιχνιδιού

Οι παίκτες παίζουν εναλλάξ. Μια κίνηση κάθε φορά. Εξ' ορισμού, η πρώτη κίνηση γίνεται από το κόκκινο καράβι, δηλαδή τον άνθρωπο. Επιτρεπόμενες κινήσεις θεωρούνται η πορεία εμπρός ή πίσω του καραβιού που χειριζόμαστε, η αύξηση ή η μείωση στάθμης νερού σε μια στήλη.

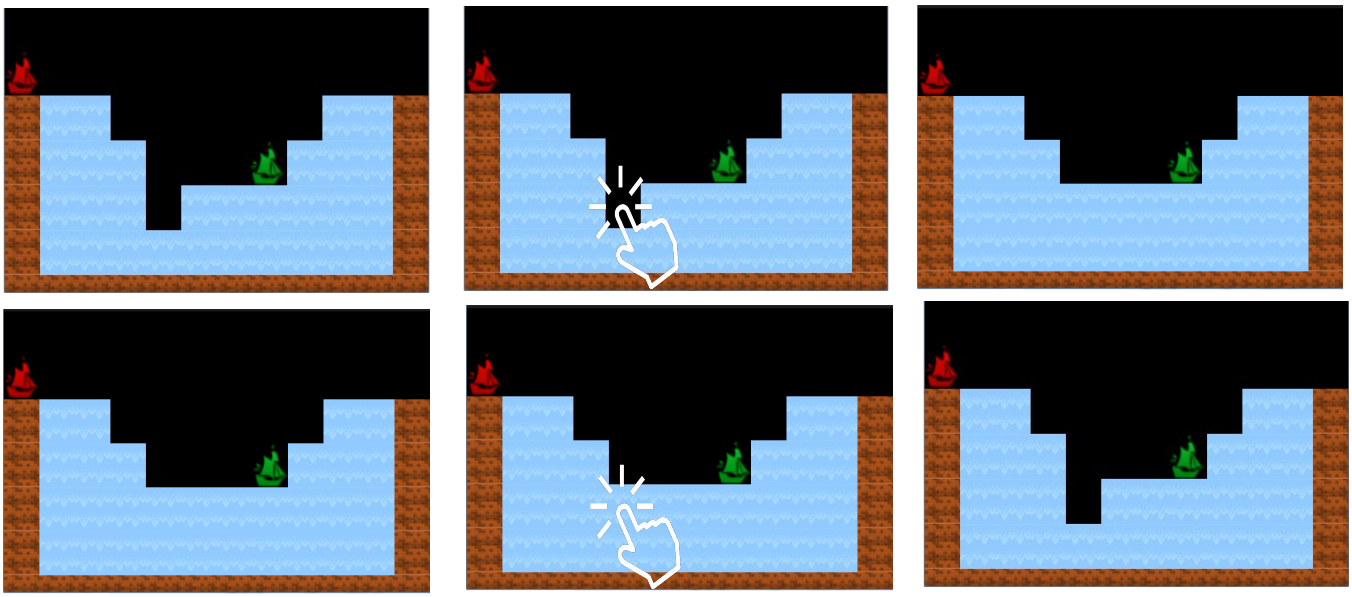
Αντίστοιχα ο αντίπαλος, δηλαδή ο πράκτορας υπολογιστής μπορεί να κινηθεί εμπρός ή πίσω, να αυξήσει ή να μειώσει τη στάθμη του νερού μιας στήλης.

Βέβαια, ταυτόχρονα ισχύουν οι λογικοί περιορισμοί ότι η στάθμη δεν μπορεί να αυξηθεί πάνω από το επίπεδο της δεξαμενής, ούτε να αφαιρεθεί νερό από άδεια στήλη. Αντίστοιχα τα καράβια μπορούν να κινηθούν αν η στάθμη λίμνης στην οποία προτίθενται να βρεθούν είναι ίση ή χαμηλότερη της στάθμης όπου βρίσκονται.

Η διαδικασία για την πραγματοποίηση της κίνησης γίνεται ως εξής: Αν θέλουμε να κινήσουμε το καράβι μας, τότε το αγγίζουμε. Αμέσως εμφανίζεται το ανδροειδές πράσινο ρομπότ, στο σημείο που έχουμε δικαίωμα να μεταφερθούμε. Επιλέγοντας το, ολοκληρώνεται η κίνηση.



Αντίστοιχα για να μειώσουμε στάθμη νερού, πατάμε στο ψηλότερο σημείο της, ενώ για να αυξήσουμε στάθμη αγγίζουμε το κενό σημείο πάνω από το νερό.



3.3 Περιγραφή του πράκτορα crazy_boat

Η ονομασία του πράκτορα crazy_boat έχει εμπνευστεί από τη συμπεριφορά του. Ο λόγος είναι ότι οι κινήσεις που επιλέγει κάθε φορά καθορίζονται από τον παράγοντα τύχη.

Αρχικά ελέγχει ποιες είναι διαθέσιμες κινήσεις που μπορεί να επιλέξει. Με τη βοήθεια της συνάρτησης Random που παράγει τυχαίους αριθμούς, επιλέγεται μια κίνηση και εκτελείται. Για δοκιμαστικούς λόγους αλλά και για να παραμείνει το ενδιαφέρον του χρήστη, οι κινήσεις του καραβιού έχουν διπλάσιες πιθανότητες να επιλεχθούν, σε σχέση με την αυξομείωση της στάθμης.

3.4 Περιγραφή του πράκτορα forward_boat

Σκοπός του πράκτορα forward_boat είναι όπως προδίδει και το όνομα το, η κίνηση που θα εκτελεί να είναι πάντα ένα βήμα μπροστά.

Δεν διεξάγει κανέναν έλεγχο, ως προς τι συμφέρον θα έχει με κάθε κίνηση. Έτσι για παράδειγμα μπορούμε εύκολα να συμπεράνουμε ότι παρόλο που θεωρείται ένας επιθετικός πράκτορας με μοναδικό σκοπό την άμεση νίκη, ταυτόχρονα είναι επιρρεπής στις παγίδες που μπορεί να του καταστρώσει ο αντίπαλος.

3.5 Περιγραφή του πράκτορα minimax_boat

Ο σημαντικότερος πράκτορας που διακατέχεται από λογική και μπορεί να ανταγωνιστεί τον άνθρωπο αντίπαλο είναι minimax_boat. Είναι αρκετά πολυπλοκότερος από τους forward_boat και crazy_boat, καθώς με ένα σύστημα βαθμολόγησης που έχει οριστεί από τους δημιουργούς του, μπορεί να υπολογίσει όλες τις πιθανές τροπές που μπορεί να συμβούν στο παιχνίδι για έναν συγκεκριμένο αριθμό μελλοντικών κινήσεων.

Εφόσον δημιουργηθεί το δέντρο αναζήτησης που προκύπτει από την συνάρτηση βαθμολόγησης, ο αλγόριθμος MiniMax υπολογίζει ποια κίνηση θα μεγιστοποιήσει το κέρδος ενώ ταυτόχρονα θα ελαχιστοποιήσει το κέρδος του αντιπάλου.

Κεφάλαιο 4
Υλοποίηση του Παιχνιδιού

4.1 Κλάσεις του Παιχνιδιού

Όπως έχει αναφερθεί στο κεφάλαιο 2 του παρόντος συγγράμματος, η υλοποίηση εφαρμογών πραγματοποιείται με τη χρήση της γλώσσας προγραμματισμού Java. Κάθε εφαρμογή έχει μια κύρια δραστηριότητα (Activity) η οποία ελέγχει την ροή μέσω των μεθόδων.

4.2 MainActivity.java

MainActivity είναι το όνομα της κύριας κλάσης της εφαρμογής μας, η οποία κληρονομεί τα χαρακτηριστικά μιας δραστηριότητας (Activity) και μας παρέχει την λειτουργικότητα που απαιτείται ώστε να αναπτυχθεί η εφαρμογή.

Μέσα στην κλάση ορίζουμε μεταβλητές , αντικείμενα άλλων κλάσεων καθώς και μεθόδους που απαιτούνται.

Ως παράδειγμα μπορούμε να αναφέρουμε την μέθοδο suffle_water() η οποία εκτελείται κάθε φορά που δημιουργείται η δραστηριότητα και αναλαμβάνει να διαμορφώσει έτσι τη στάθμη της λίμνης ώστε αφενός να μην υπάρχει συμμετρία και αφετέρου να γίνει ποιο ενδιαφέρον το παιχνίδι για τον χρήστη.

Η λίμνη εξ' ορισμού έχει σχήμα ανάποδης πυραμίδας. Με την εκτέλεση του παρακάτω κώδικα της μεθόδου suffe_water() παίρνουμε σαν αποτέλεσμα μια τυχαία επιλογή από τις τέσσερις πιθανές.

```
734
735     private void suffle_water()
736     {
737         Random rand = new Random();
738         int r = rand.nextInt(3);
739
740         if (r==0)
741         {
742             ImageAdapter.mThumbIds[54] = I_WATER;
743             ImageAdapter.mThumbIds[42] = I_WATER;
744         }
745         if (r==1)
746         {
747             ImageAdapter.mThumbIds[53] = I_WATER;
748             ImageAdapter.mThumbIds[41] = I_WATER;
749         }
750         if (r==2)
751         {
752             ImageAdapter.mThumbIds[65] = I_BLACK;
753             ImageAdapter.mThumbIds[54] = I_WATER;
754         }
755         if (r==3)
756         {
757             ImageAdapter.mThumbIds[66] = I_BLACK;
758             ImageAdapter.mThumbIds[53] = I_WATER;
759         }
760
761     } //shuffle end
762
763
```

Μια ακόμα μέθοδος που χρησιμοποιείται είναι η isEnabled. Η μέθοδος αυτή είναι τύπου boolean, δέχεται σαν παράμετρο έναν ακέραιο αριθμό που αναπαριστά το σημείο όπου έχει κάνει κλικ ο χρήστης και ελέγχει αν αυτό είναι ένα νεκρό σημείο, με την έννοια ότι καμία κίνηση δεν μπορεί να πραγματοποιηθεί, όπως για παράδειγμα το κενό που βρίσκεται πάνω από την στάθμη της λίμνης και μπορούμε να θεωρήσουμε ότι είναι ο ορίζοντας. Σε περίπτωση που το κλικ πραγματοποιηθεί σε έγκυρο σημείο, επιστρέφεται η τιμή true, αλλιώς επιστρέφεται η τιμή false.

0	1	2	3	4	5	6	7	8	9	10	11
12	13	14	15	16	17	18	19	20	21	22	23
24	25	26	27	28	29	30	31	32	33	34	35
36	37	38	39	40	41	42	43	44	45	46	47
48	49	50	51	52	53	54	55	56	57	58	59
60	61	62	63	64	65	66	67	68	69	70	71
72	73	74	75	76	77	78	79	80	81	82	83

Οι δύο πρώτες γραμμές, όπως φαίνεται και στο παραπάνω σχήμα, είναι κωδικοποιημένες με τους αριθμούς από το 0 μέχρι το 23. Με εξαίρεση τα σημεία 12 και 23, όπου βρίσκονται τα καράβια κατά την εκκίνηση του παιχνιδιού και καταλήγουν εκεί με σκοπό να κερδίσουν την παρτίδα, όλες οι υπόλοιπες θέσεις μπορούν να χρησιμοποιηθούν από τους παίκτες.

Ο κώδικας της μεθόδου, φαίνεται στην παρακάτω εικόνα

```
782
783
784    protected boolean isEnabled(int position) {
785        // TODO Auto-generated method stub
786        String pos = String.valueOf(position);
787        if (pos.equals("0") || pos.equals("1") || pos.equals("2")
788                || pos.equals("3") || pos.equals("4") || pos.equals("5") || pos.equals("6")
789                || pos.equals("7") || pos.equals("8") || pos.equals("9") || pos.equals("10")
790                || pos.equals("11") || pos.equals("13") || pos.equals("14") || pos.equals("15")
791                || pos.equals("16") || pos.equals("17") || pos.equals("18") || pos.equals("19")
792                || pos.equals("20") || pos.equals("21") || pos.equals("22") )
793        {
794            return false;
795        }
796        else
797        {
798        return true;
799        }
800    }
801
802
```

Στην συνέχεια παρουσιάζονται οι μέθοδοι που αποτελούν τους τρεις πράκτορες του παιχνιδιού και μπορούμε να θεωρήσουμε ότι είναι η ψυχή της εφαρμογής

4.2.1 Μέθοδος crazy_Boat

Ο πράκτορα Crazy_Boat επιλέγει τυχαία τις κινήσεις που θα κάνει. Δεν έχει καμία στρατηγική και φυσικά ούτε λογική. Στην εικόνα φαίνεται ο κώδικας της μεθόδου με τα σχόλια.

```
941
942     private void crazy_Boat(AdapterView<?> parent)
943     {
944         // Αρχικοποιηση θέσης (position)
945         pos.resetPosition();
946         // Ανάθεση τιμών σύμφωνα με το ImageAdapter
947         pos.setPositionFromImageAdapter(null_move, Position.GREEN);
948
949         // Δημιουργία λίστας με όλες τις επιτρεπόμενες κινήσεις
950         Vector<Move> mv_list = Solver.getValidMoves(pos);
951
952         // Επιλογή μιας κίνησης από τη λίστα τυχαία
953         Random r = new Random();
954         int _ix = r.nextInt(mv_list.size());
955         resulting_move.assignMove(mv_list.elementAt(_ix));
956
957         // Εφαρμογή της κίνησης
958         solver.doMove(pos, resulting_move, true);
959
960         // Εγγραφή της νέας θέσης στο ImageAdapter και απεικόνηση
961         pos.writePositionToImageAdapter();
962         redraw(parent);
963
964         // Έλεγχος αν έχει ολοκληρωθεί το παιχνίδι
965         flag_of_game = endOfGameCheck(pos);
966     }
967
```

Αρχικά έχουμε ορίσει μια νέα position με την εντολή `Position pos = new Position();` η οποία βρίσκεται έξω από την μέθοδο.

Στη συνέχεια αρχικοποιούνται οι μεταβλητές του αντικειμένου pos και λαμβάνουν νέες τιμές σύμφωνα με τα δεδομένα του ImageAdapter. Άρα στη φάση αυτή το αντικείμενο pos αντικατοπτρίζει την εικόνα που υπάρχει στην οθόνη μας.

Δημιουργείται η λίστα με όλες τις έγκυρες κινήσεις που μπορεί να πραγματοποιήσει ο πράκτορας crazy_boat και επιλέγεται τυχαία μια. Εφαρμόζεται πρώτα εσωτερικά, δηλαδή αλλάζουν οι τιμές του αντικειμένου pos και στην συνέχεια αποτυπώνεται στο ImageAdapter, μέσω του οποίου βλέπει και ο χρήστης την κίνηση στην οθόνη του.

Τέλος ελέγχει αν με την κίνηση που πραγματοποιήθηκε, το παιχνίδι έχει ολοκληρωθεί, δηλαδή αν υπάρχει κάποιος νικητής ή χαμένος.

4.2.2 Μέθοδος forward_Boat

Η λογική του πράκτορα forward_Boat είναι σε κάθε κίνηση το καράβι που κινεί να κάνει ένα βήμα μπροστά με σκοπό να κερδίσει. Δεν εξετάζει αν κινδυνεύει να καταποντιστεί εκτός λίμνης.

Η μοναδική περίπτωση να μην προχωρήσει μπροστά είναι αν δεν επιτρέπεται η κίνηση λόγω του ότι στο σημείο βρίσκεται νερό. Έτσι αυξάνει την στάθμη της λίμνης όπου βρίσκεται κατά ένα επίπεδο.

Ο κώδικας της μεθόδου forward_boat φαίνεται στην παρακάτω εικόνα.

```
916
917
918     private void forward_Boat(AdapterView<?> parent)
919     {
920         // Αρχικοποιηση θέσης (position)
921         pos.resetPosition();
922         // Ανάθεση τιμών σύμφωνα με το ImageAdapter
923         pos.setPositionFromImageAdapter(null_move, Position.GREEN);
924
925         // ορίζει την κίνηση ένα βήμα αριστερά
926         resulting_move.setMove(Move.MV_GreenLeft, Position.GREEN);
927
928         // Αν η κίνηση δεν είναι έγκυρη, τότε αυξάνεται η στάθμη της λίμνης
929         if (!Solver.isValidMove(pos, resulting_move)) {
930             int _id = Move.MV_Water1Up + pos.getGreen() - 1;
931             resulting_move.setMove(_id, Position.GREEN);
932         }
933
934         // Εφαρμογή της κίνησης
935         solver.doMove(pos, resulting_move, true);
936
937         // Εγγραφή της νέας θέσης στο ImageAdapter και απεικόνηση
938         pos.writePositionToImageAdapter();
939         redraw(parent);
940
941         // Έλεγχος αν έχει ολοκληρωθεί το παιχνίδι
942         flag_of_game = endOfGameCheck(pos);
943     }
944
```

Αρχικά έχουμε ορίσει μια νέα position με την εντολή `Position pos = new Position();` η οποία βρίσκεται έξω από την μέθοδο.

Στη συνέχεια αρχικοποιούνται οι μεταβλητές του αντικειμένου pos και λαμβάνουν νέες τιμές σύμφωνα με τα δεδομένα του ImageAdapter. Άρα στη φάση αυτή το αντικείμενο pos αντικατοπτρίζει την εικόνα που υπάρχει στην οθόνη μας.

Ορίζεται η κίνηση που θα πραγματοποιηθεί, ένα βήμα μπροστά. Δηλαδή αριστερά για το πράσινο καράβι που έλεγχεται από τον πράκτορα forward_boat.

Αν η κίνηση δεν είναι έγκυρη, τότε εφαρμόζεται η εναλλακτική κίνηση, δηλαδή να αυξηθεί η στάθμη της λίμνης κατά ένα επίπεδο.

Εφαρμόζεται πρώτα εσωτερικά, δηλαδή αλλάζουν οι τιμές του αντικειμένου pos και στην συνέχεια αποτυπώνεται στο ImageAdapter, μέσω του οποίου βλέπει και ο χρήστης την κίνηση στην οθόνη του.

Τέλος ελέγχει αν με την κίνηση που πραγματοποιήθηκε, το παιχνίδι έχει ολοκληρωθεί, δηλαδή αν υπάρχει κάποιος νικητής ή χαμένος.

4.2.3 Μέθοδος minmax_Boat

Ο πράκτορας minmax_boat εξετάζει και βαθμολογεί όλες τις πιθανές καταστάσεις που μπορεί να προκύψουν μετά από 4 διαδοχικές κινήσεις των αντιπάλων.

Σύμφωνα με τους υπολογισμούς αυτούς, επιλέγει αυτή που θεωρεί πιο συμφέρουσα για τον ίδιο, ενώ πάντα θεωρεί οτι αντίπαλος θα επιλέξει αντίστοιχα την καλύτερη για αυτόν κίνηση.

Στην εικόνα παρακάτω φαίνεται ο κώδικας του πράκτορα.

```
895
896     private void minmax_Boat(AdapterView<?> parent)
897     {
898         // Αρχικοποιηση θέσης (position)
899         pos.resetPosition();
900         // Ανάθεση τιμών σύμφωνα με το ImageAdapter
901         pos.setPositionFromImageAdapter(move, Position.GREEN);
902
903         // αναζήτηση της καλύερης κίνησης
904         resulting_move.clearMove();
905         int pos_eval = solver.doMinMax(pos, 4, resulting_move, false, 4);
906
907         // Εφαρμογή της κίνησης
908         boolean ok = solver.doMove(pos, resulting_move, true);
909
910         // Εγγραφή της νέας θέσης στο ImageAdapter και απεικόνηση
911         pos.writePositionToImageAdapter();
912         redraw(parent);
913
914         // Έλεγχος αν έχει ολοκληρωθεί το παιχνίδι
915         flag_of_game = endOfGameCheck(pos);
916
917         // Εμφάνιση αποτελεσμάτων για πληροφόρηση του προγραμματιστή
918         Log.i("SOLVER", "e="+pos_eval+" id="+resulting_move.getId()+" ok="+ok);
919         Toast.makeText(getApplicationContext(), "e="+pos_eval + " id="+resulting_move.getId() +
920                                                 " ok="+ok, Toast.LENGTH_SHORT).show();
921     }
922
923
924
```

Όπως όλοι οι πράκτορες, αρχικά ορίζουμε μια νέα position με την εντολή `Position pos = new Position();` η οποία βρίσκεται έξω από την μέθοδο.

Στη συνέχεια αρχικοποιούνται οι μεταβλητές του αντικειμένου pos και λαμβάνουν νέες τιμές σύμφωνα με τα δεδομένα του ImageAdapter. Άρα στη φάση αυτή το αντικείμενο pos αντικατοπτρίζει την εικόνα που υπάρχει στην οθόνη μας.

Εκτελείται ο αλγόριθμος MiniMax με παραμέτρους τη θέση που ισχύει εκείνη τη στιγμή, το βάθος της αναζήτησης καθώς και αν η αναζήτηση θα γίνει αναδρομικά, όπου έχει τιμή false.

Ο αλγόριθμος minimax είναι ένας αναδρομικός αλγόριθμος. Τιμή false σχετικά με την αναδρομή, έχει μόνο κατά την πρώτη κλήση του, που πραγματοποιείται έξω από την μέθοδο αναζήτησης, δηλαδή στη

κλάση MainActivity. Αντίθετα μέσα στην μέθοδο αναζήτησης, δηλαδή την μέθοδο doMinMax της κλασης Solver, όπου η μέθοδος καλεί τον εαυτό της, η παράμετρος είναι true και η αναζήτηση γίνεται με αναδρομικό χαρακτήρα.

Τέλος η συνάρτηση μας επιστρέφει την καλύτερη κίνηση και την βαθμολογία από την οποία προέκυψε. Εφαρμόζεται πρώτα εσωτερικά, δηλαδή αλλάζουν οι τιμές του αντικειμένου pos και στην συνέχεια αποτυπώνεται στο ImageAdapter, μέσω του οποίου βλέπει και ο χρήστης την κίνηση στην οθόνη του.

Ελέγχει αν με την κίνηση που πραγματοποιήθηκε, το παιχνίδι έχει ολοκληρωθεί, δηλαδή αν υπάρχει κάποιος νικητής ή χαμένος και εμφανίζει πληροφορίες στην οθόνη σχετικά με την κίνηση και την βαθμολογία της κίνησης, ώστε να ελέγχουμε τις καταστάσεις κατά την διεξαγωγή του παιχνιδιού.

4.3 Position.java

Η κλάση Position διατηρεί πληροφορίες σχετικά με το περιβάλλον του παιχνιδιού. Ορίζουμε σε ποιο σημείο βρίσκονται οι αντίπαλοι, τα τοιχώματα που αποτελούν την δεξαμενή και την στάθμη του νερού σε κάθε στήλη.

Ο χειρισμός των στοιχείων αυτών, επιτυγχάνεται μέσω των κατάλληλων μεθόδων. Σαν παράδειγμα μπορούμε να αναφέρουμε την μέθοδο resetPosition() η οποία επαναφέρει το περιβάλλον στην αρχική κατάσταση. Ο κώδικας φαίνεται στην παρακάτω εικόνα.

```
// Επανεκκίνηση στην αρχική κατάσταση
public void resetPosition()
{
    red = 0;
    green = 11;
    turn = RED;                    // εξ' ορισμού την πρώτη κίνηση κάνει ο κόκκινος

    // ορίζουμε την στάθμη της λίμνης σε σχήμα ανάποδης πυραμίδας
    water[0] = 4;   water[1] = 4;   water[2] = 3;   water[3] = 2;   water[4] = 1;
    water[5] = 1;   water[6] = 2;   water[7] = 3;   water[8] = 4;   water[9] = 4;

    endofgame_flags = 0;
    moves.clear();

}
```

Ορίζουμε ότι ο κόκκινος παίκτης βρίσκεται στην στήλη 0, ο πράσινος παίκτης στην στήλη 11 και είναι η σειρά του κόκκινου παίκτη να παίξει. Στην συνέχεια δηλώνουμε την στάθμη της λίμνης για κάθε στήλη που την αποτελεί. Τέλος, μηδενίζουμε το ιστορικό των κινήσεων και θέτουμε την τιμή 0 στην μεταβλητή endofgame_flags, που σημαίνει ότι το παιχνίδι δεν έχει τερματιστεί.

Στην συνέχεια παρουσιάζουμε κάποιες ακόμα βοηθητικές μεθόδους

```
88      // Επιστρέφει την στάθμη του νερού για συγκεκριμένη στήλη
89      public int getLevel(int column)
90      {
91          if (column >= 1 && column <= 10) {
92              return water[column-1];
93          }
94
95          return 4;
96      }
97
98      // έλεγχος αν έχει τερματισει το παιχνιδι
99      public int getEndOfGame()
100     {
101         return endofgame_flags;
102     }
103
104
105     // επιστρέφει τον νικητή, αν υπάρχει
106     public int getWinner()
107     {
108         if ((endofgame_flags & EOG_WIN_RED) != 0) {
109             return RED;
110         }
111         else if ((endofgame_flags & EOG_WIN_GREEN) != 0) {
112             return GREEN;
113         }
114
115         return NONE;
116     }
```

Η μέθοδος getLevel(int) λαμβάνει σαν παράμετρο έναν ακέραιο αριθμό που αντιστοιχεί σε στήλη και μας επιστρέφει το επίπεδο της στάθμης για την συγκεκριμένη στήλη. Η μέθοδος getWinner() μας επιστρέφει τον νικητή του παιχνιδιού, αν υπάρχει.

Δυο πολύ σημαντικές μέθοδοι για τη λειτουργία της εφαρμογής είναι η `writePositionToImageAdapter()` και η `setPositionFromImageAdapter(Move last_move, int player_turn)`. Μπορούμε να θεωρήσουμε πως επιτελούν δυο ακριβώς αντίθετες εργασίες. Όπως προκύπτει και από το όνομα τους, η πρώτη διαβάζει τα στοιχεία της κλάσης Position και μεταφέρει τα δεδομένα στον ImageAdapter ώστε τα αποτελέσματα να εμφανιστούν στην οθόνη. Από την άλλη πλευρά, η δεύτερη μέθοδος διαβάζει την κατάσταση της οθόνης και ενημερώνει την κλάση Position.

4.4 Move.java

Η κλάση Move μας επιτρέπει να χειριζόμαστε τις κινήσεις του παιχνιδιού. Κάθε αντικείμενο της κλάσης Move χαρακτηρίζεται από τον αριθμό της κίνησης και τον παίκτη που την πραγματοποιεί, κόκκινο ή πράσινο καράβι.

Συνολικά οι επιτρεπόμενες κινήσεις είναι 25. Ο ορισμός τους φαίνεται στην παρακάτω εικόνα.

```
3  public class Move {
4
5      // σταθερές κινήσεων
6      public static final int MV_RedLeft = 0;
7      public static final int MV_RedRight = 1;
8      public static final int MV_GreenLeft = 2;
9      public static final int MV_GreenRight = 3;
10     public static final int MV_Water1Up = 4;
11     public static final int MV_Water2Up = 5;
12     public static final int MV_Water3Up = 6;
13     public static final int MV_Water4Up = 7;
14     public static final int MV_Water5Up = 8;
15     public static final int MV_Water6Up = 9;
16     public static final int MV_Water7Up = 10;
17     public static final int MV_Water8Up = 11;
18     public static final int MV_Water9Up = 12;
19     public static final int MV_Water10Up = 13;
20     public static final int MV_Water1Down = 14;
21     public static final int MV_Water2Down = 15;
22     public static final int MV_Water3Down = 16;
23     public static final int MV_Water4Down = 17;
24     public static final int MV_Water5Down = 18;
25     public static final int MV_Water6Down = 19;
26     public static final int MV_Water7Down = 20;
27     public static final int MV_Water8Down = 21;
28     public static final int MV_Water9Down = 22;
29     public static final int MV_Water10Down = 23;
30     public static final int MV_null = 24;
31
32     // αντίστροφες κινήσεις
33     private static final int INVERSE_MOVES[] = {
34         MV_RedRight, MV_RedLeft, MV_GreenRight, MV_GreenLeft,
35         MV_Water1Down, MV_Water2Down, MV_Water3Down, MV_Water4Down,
36         MV_Water5Down, MV_Water6Down, MV_Water7Down, MV_Water8Down,
37         MV_Water9Down, MV_Water10Down, MV_Water1Up, MV_Water2Up,
38         MV_Water3Up, MV_Water4Up, MV_Water5Up, MV_Water6Up, MV_Water7Up,
39         MV_Water8Up, MV_Water9Up, MV_Water10Up, MV_null
40     };
41
42     private int m_id;              // αριθμός κίνησης
43     private boolean m_endofagme;   // true, αν η κίνηση τερματίζει το παιχνίδι
44     private int m_player;          // ο παίκτης που κάνει την κίνηση
```

Επίσης η κλάση Move χρησιμοποιεί μεθόδους που μας επιτρέπει να δημιουργούμε και να χειριζόμαστε τα αντικείμενα της.

```
46
47      // κατασκευαστής
48      public Move()
49      {
50          m_id = MV_null;
51          m_player = Position.NONE;
52          m_endofagme = false;
53      }
54
55      // κατασκευαστής με id κίνησης και παίκτη
56      public Move(int id, int player)
57      {
58          if (id >= MV_RedLeft && id <= MV_Water10Down &&
59                  player >= Position.RED && player <= Position.GREEN) {
60              m_id = id;
61              m_player = player;
62          }
63          else {
64              m_id = MV_null;
65              m_player = Position.NONE;
66          }
67          m_endofagme = false;
68      }
69
70      // κατασκευαστής ανάθεσης κίνησης
71      public Move assignMove(Move rhs)
72      {
73          if (this == rhs) return this; // αν πρόκειτε για το ιδιο αντικειμενο
74          // αν αναθέτουμε νέα κίνηση
75          m_id = rhs.m_id;
76          m_player = rhs.m_player;
77          m_endofagme = rhs.m_endofagme;
78
79          return this;
80      }
81
```

Όπως συμβαίνει στις κλάσεις, υπάρχει η μέθοδος κατασκευαστής που δημιουργεί ένα αντικείμενο με αρχικοποίηση τιμών και ένας κατασκευαστής όπου δημιουργεί αντικείμενο σύμφωνα με τις παραμέτρους που δέχεται. Στην περίπτωση μας οι παράμετροι αυτοί είναι το id της κίνησης και του παίκτη.

Τέλος υπάρχει η μέθοδος assignMove όπου μας επιτρέπει να μεταφέρουμε τα δεδομένα μιας κίνησης σε μια άλλη και στην ουσία μας εξυπηρετεί ώστε να διατηρούμε πάντα την καλύτερη κίνηση σε ένα αντικείμενο της κλάσης.

4.5 Solver.java

Η κλάση Solver λειτουργεί περισσότερο σαν ένα container μεθόδων, παρά σαν μια κλάση που μας επιτρέπει να δημιουργήσουμε αντικείμενα.

Η κύρια μέθοδος είναι η doMinMax, η οποία με τη λήψη των κατάλληλων πληροφοριών, αναλαμβάνει να αναζητήσει την καλύτερη κίνηση, σύμφωνα με τον αλγόριθμο MiniMax. Οι παράμετροι που απαιτούνται για να λειτουργήσει είναι η κατάσταση του παιχνιδιού την παρούσα στιγμή και σε ποιο βάθος θα γίνει η αναζήτηση. Στην εικόνα φαίνεται η μέθοδος doMinMax:

```
302    int doMinMax(Position pos, int toplvl, Move bestmove, boolean recursive_call, int lvl)
303    {
304        Vector<Move> mv_list;
305        int best_e, new_e;
306        int nr_moves;
307
308        if (!recursive_call) { lvl = toplvl; }
309
310        if (pos.getEndOfGame()!=0 || lvl == 0)  { return evalPosition(pos); }
311
312        // ο κόκκινος ξεκινά με -INFINITY και προσπαθεί να μεγιστοποιήσει
313        // ο πράσινος ξεκινά με +INFINITY και προσπαθεί να ελαχιστοποιήσει
314        if (pos.getTurn() == Position.RED) { best_e = -INFINITY; }
315        else { best_e = +INFINITY; }
316
317        mv_list = getValidMoves(pos);
318        nr_moves = mv_list.size();
319
320        if (nr_moves == 0) { return best_e; }
321
322        Move new_mv = new Move();
323        for (int i=0; i<nr_moves; i++) {
324            doMove(pos, mv_list.get(i), true);
325            new_e = doMinMax(pos, toplvl, new_mv, true, lvl-1);
326            undoMove(pos, mv_list.get(i));
327            if (pos.getTurn() == Position.RED) {
328                // κόκκινος μεγιστοποιεί
329                if (new_e > best_e) {     // Αν βρεθεί καλύτερη κίνηση
330                    best_e = new_e;
331                    bestmove.assignMove(mv_list.get(i));
332                }
333            }
334            else {
335                // πράσινος ελαχιστοποιεί
336                if (new_e < best_e) {     // Αν βρεθεί καλύτερη κίνηση
337                    best_e = new_e;
338                    bestmove.assignMove(mv_list.get(i));
339                }
340            }
341        }
342
343        return best_e;
344    }
345
```

Παρατηρούμε ότι η μέθοδος doMinMax χρησιμοποιεί επίσης κάποιες βοηθητικές μεθόδους, οι οποίες ορίζονται μέσα στην κλάση Solver.

Η doMove λαμβάνει ένα αντικείμενο Position και εφαρμόζει την κίνηση που λαμβάνει από την παράμετρο. Αντίστοιχα η undoMove αναιρεί την doMove. Στην ουσία εφαρμόζεται μια κίνηση ώστε να υπολογιστεί η συνάρτηση βαθμολογίας χωρίς όμως να μεταβληθεί η παρούσα κατάσταση του παιχνιδιού.

Τέλος για να υπολογίσουμε την τιμή της βαθμολογίας για κάθε περίπτωση, χρησιμοποιούμε την μέθοδο evalPosition με παράμετρο την τρέχουσα κατάσταση του παιχνιδιού. Μας επιστρέφει έναν ακέραιο αριθμό ο οποίος προκύπτει από την διαφορά της απόστασης του πράσινου καραβιού από το καρνάγιο του αντιπάλου όπου βρίσκεται ο τελικός του στόχος, σε σχέση με την απόσταση του κόκκινου καραβιού από το καρνάγιο του πράσινου καραβιού.

Έτσι μπορούμε να συμπεράνουμε ότι όταν η τιμή είναι θετική, το κόκκινο καράβι βρίσκεται σε πλεονεκτικότερη θέση και αντίστοιχα όταν η τιμή είναι αρνητική υπερισχύει το πράσινο καράβι που ελαχιστοποιεί.

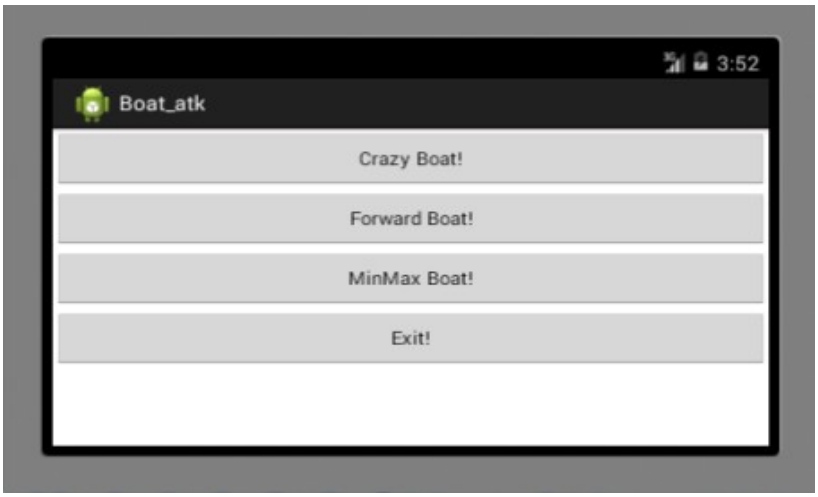
Η υπολογισμός της απόστασης καθώς και όλη η μέθοδος evalPosition φαίνεται στην παρακάτω εικόνα:

```
220
221     // Μέθοδος υπολογισμού βαθμολογίας
222     private int evalPosition(Position pos)
223     {
224         int e = 0;
225
226         if (pos.getWinner() == Position.RED) {
227             e = INFINITY - 1;
228         }
229         else if (pos.getWinner() == Position.GREEN) {
230             e = -INFINITY + 1;
231         }
232         else {
233
234
235             int _red_column = pos.getRed();
236             int _distance_red = 2*(11 - _red_column) + (4 - pos.getLevel(_red_column));
237             if (pos.getTurn() == Position.RED) {
238                 _distance_red--;
239             }
240
241             int _green_column = pos.getGreen();
242             int _distance_green = 2*_green_column + (4 - pos.getLevel(_green_column));
243             if (pos.getTurn() == Position.GREEN) {
244                 _distance_green--;
245             }
246
247             e = _distance_green - _distance_red;
248
249         }
250
251         return e;
252     }
253
```

Κεφάλαιο 5
Παραδείγματα Εκτέλεσης

Στο παρόν κεφάλαιο θα παρατεθούν παραδείγματα εκτέλεσης του παιχνιδιού καθώς και στιγμιότυπα για κάθε έναν από τους τρεις πράκτορες, Crazy_Boat, Forward_Boat και Minimax_Boat.

Εκκινώντας την εφαρμογή, εμφανίζεται το μενού πλοήγησης, με τη βοήθεια του οποίου μπορούμε να επιλέξουμε έναν από τους τρεις πράκτορες για αντίπαλο μας. Η εικόνα δείχνει το μενού πλοήγησης:

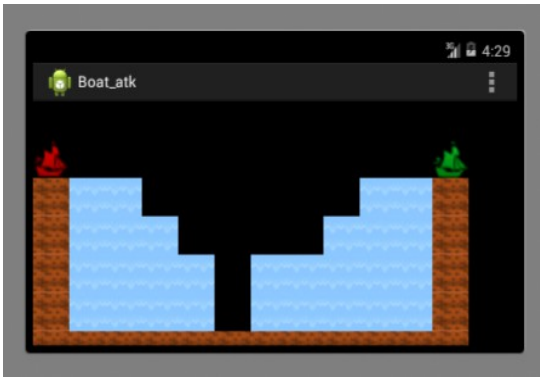


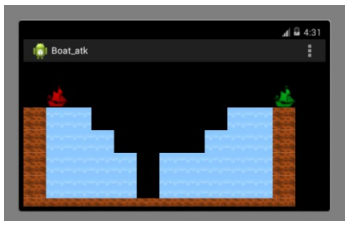
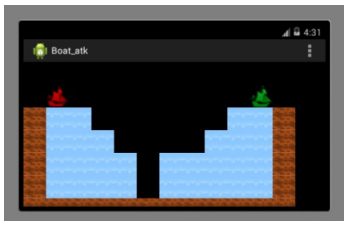
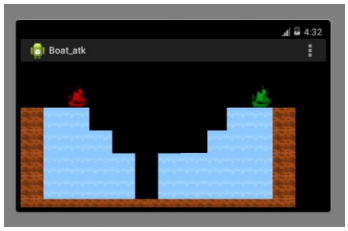
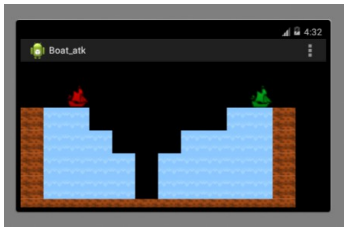
5.1 Crazy_Boat

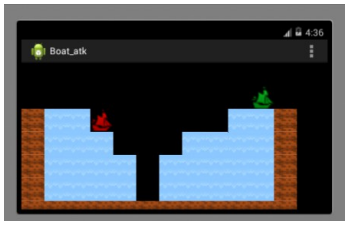
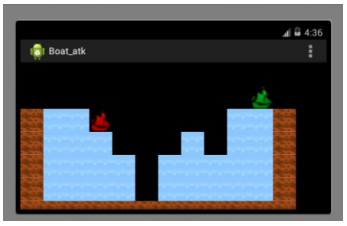
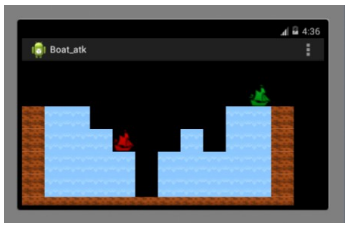
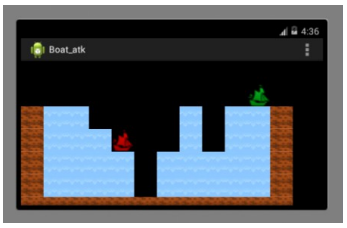
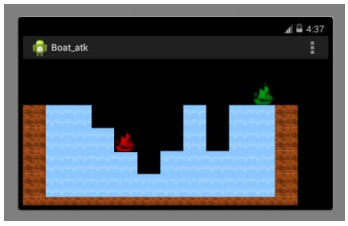
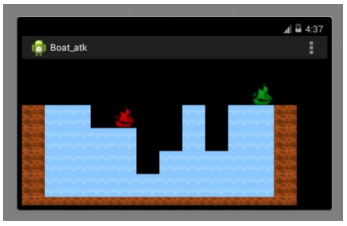
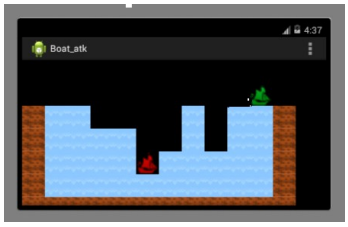
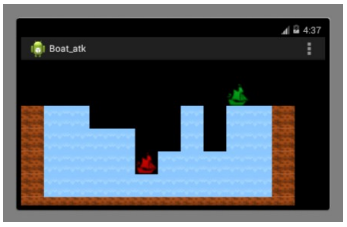
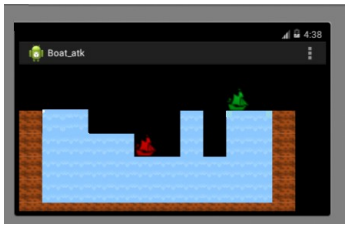
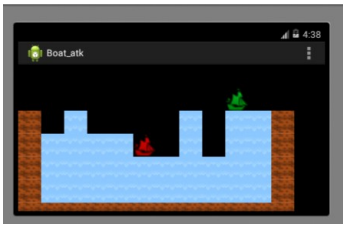
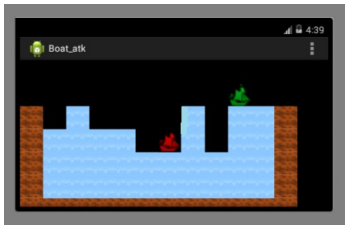
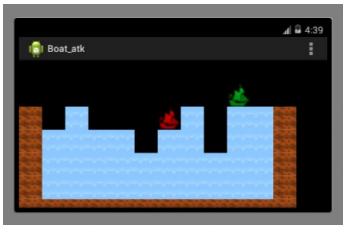
Ο πράκτορας Crazy_Boat είναι ένας τρελός πράκτορας ο οποίος επιλέγει τις κινήσεις που κάνει τυχαία. Έχει δικαίωμα να προσθέσει ή να αφαιρέσει στάθμη της λίμνης από οποιοδήποτε σημείο, όπως επίσης και να μετακινήσει τον εαυτό του, δηλαδή το πράσινο καράβι.

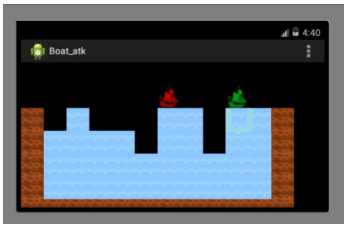
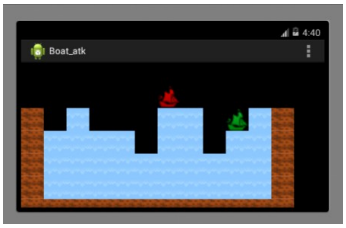
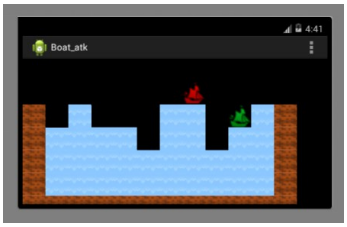
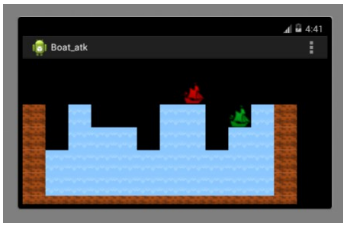
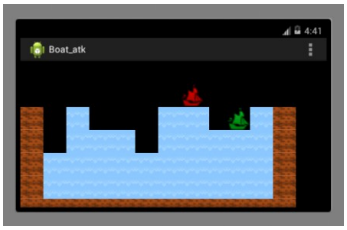
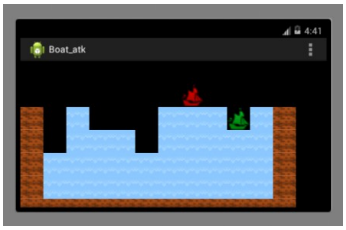
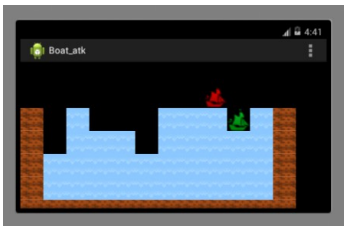
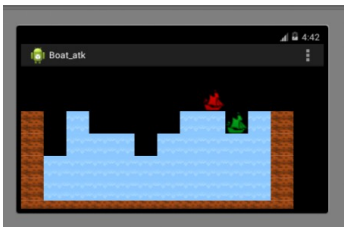
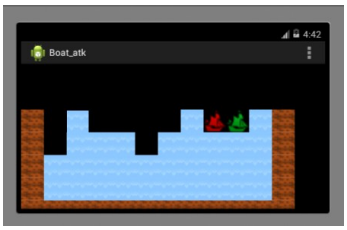
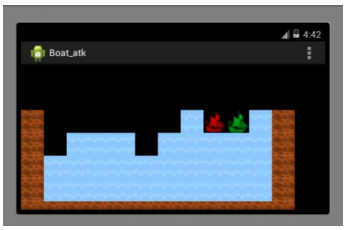
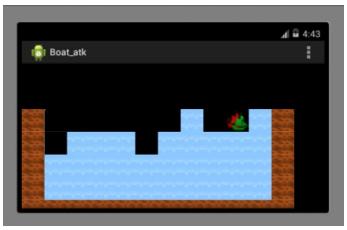
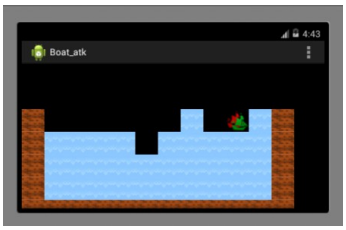
Στις εικόνες που βρίσκονται παρακάτω, υπάρχει ένα ολοκληρωμένο παιχνίδι με επεξήγηση κάθε κίνησης, στο οποίο νικητής είναι ο κόκκινος παίκτης.

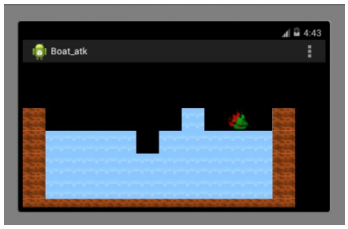
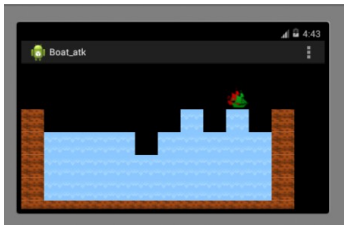
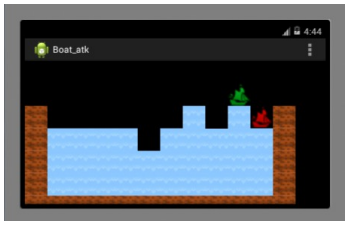
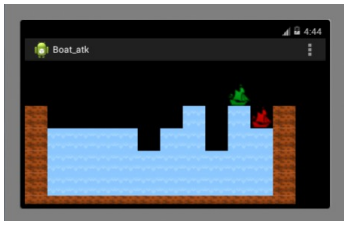
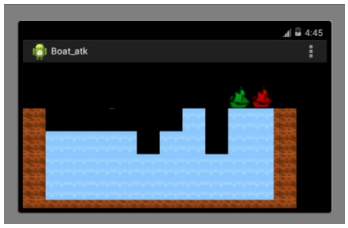
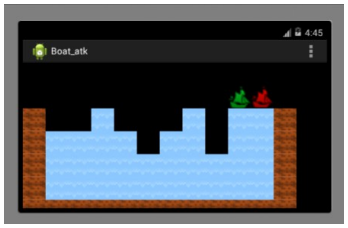
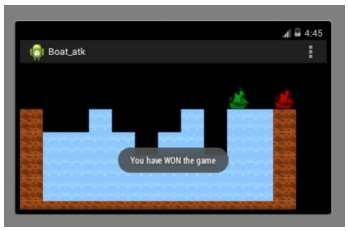
Αρχική κατάσταση



Κόκκινος Παίκτης	Πράσινος Παίκτης
 MV_RedRight	 MV_GreenLeft
 MV_RedRight	 MV_Water7Up

 MV_RedRight	 MV_Water8Down
 MV_RedRight	 MV_Water7Up
 MV_Water5Up	 MV_Water4Down
 MV_RedRight	 MV_GreenLeft
 MV_Water5Up	 MV_Water1Down
 MV_RedRight	 MV_Water6Up

 MV_Water6Up	 MV_Water9Down
 MV_RedRight	 MV_Water1Down
 MV_Water8Up	 MV_Water8Up
 MV_RedRight	 MV_Water6Down
 MV_Water8Down	 MV_Water2Down
 MV_RedRight	 MV_Water1Up

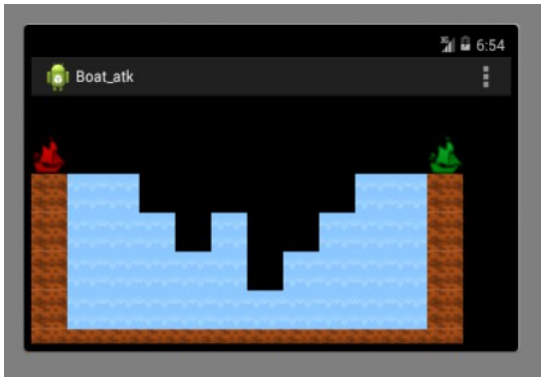
 MV_Water10Down	 MV_Water9Up
 MV_RedRight	 MV_Water8Down
 MV_Water10Up	 MV_Water3Up
 MV_RedRight	

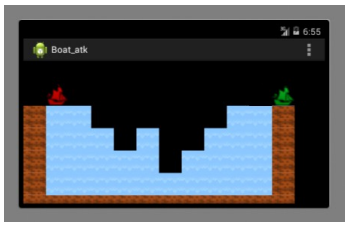
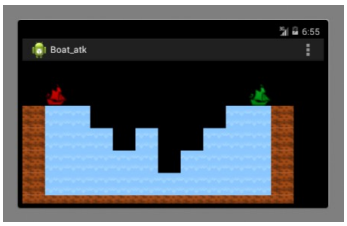
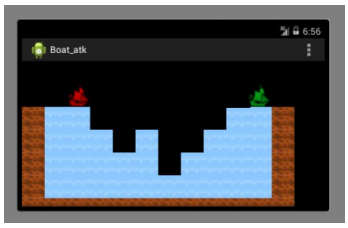
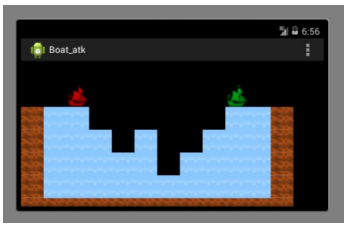
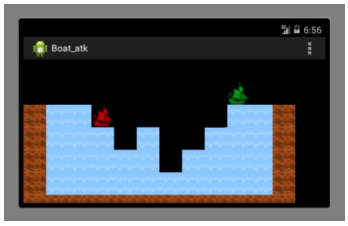
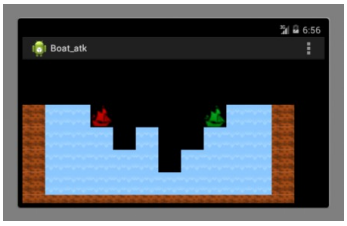
5.2 Forward_Boat

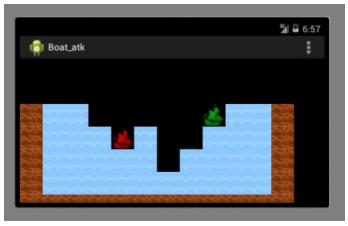
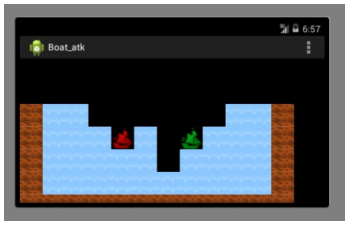
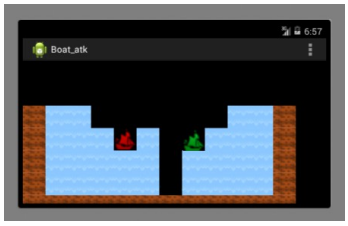
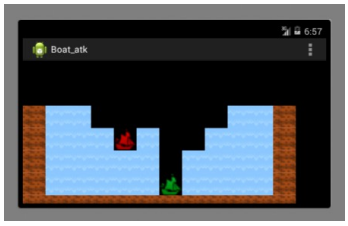
Ο πράκτορας Forward_Boat κάνει κινήσεις πάντα μπροστά με σκοπό την νίκη. Εξαίρεση αποτελεί η περίπτωση που δεν υπάρχει δυνατότητα να κινηθεί μπροστά και έτσι αυξάνει τη στάθμη της λίμνης στο σημείο το οποίο βρίσκεται.

Όπως θα δούμε στο παράδειγμα εκτέλεσης παρακάτω, δεν κάνει κανένα έλεγχο πριν την κίνηση του, με αποτέλεσμα να καταποντίζεται αν στο επόμενο βήμα δεν υπάρχει νερό.

Αρχική κατάσταση



 MV_RedRight	 MV_GreenLeft
 MV_RedRight	 MV_GreenLeft
 MV_RedRight	 MV_GreenLeft

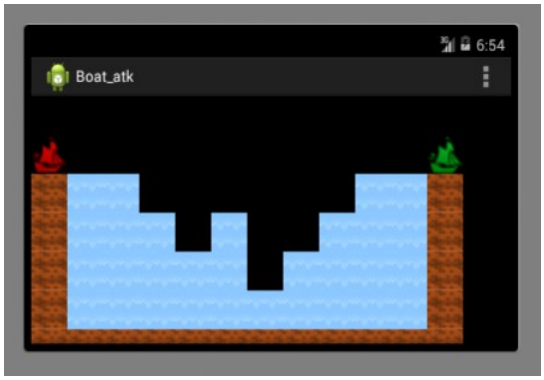
 MV_RedRight	 MV_GreenLeft
 MV_Water6Down	 MV_GreenLeft

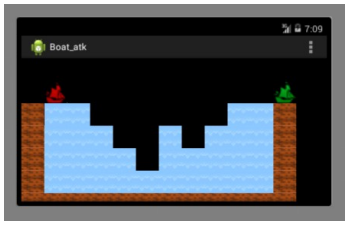
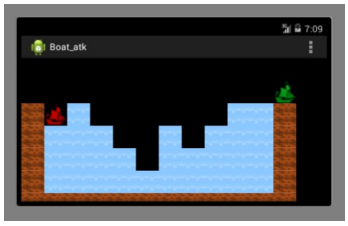
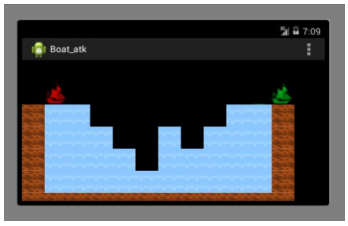
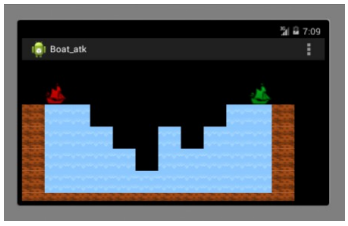
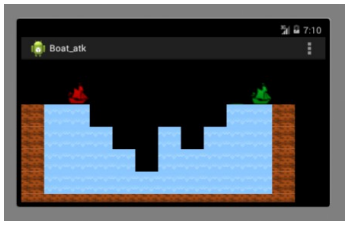
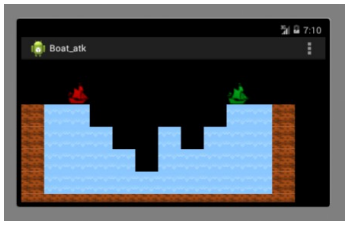
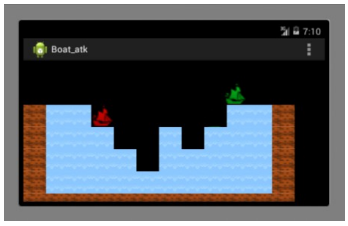
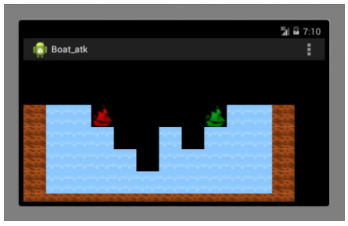
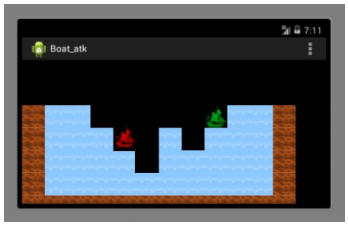
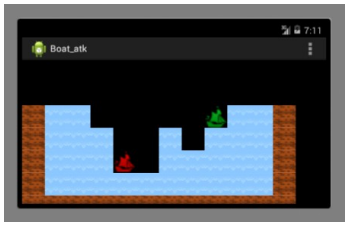
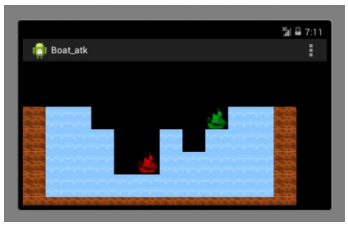
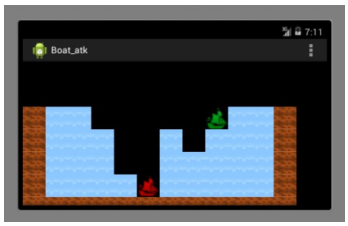
5.3 MiniMax_Boat

Ο πράκτορας MiniMax_Boat λειτουργεί όπως ορίζει η θεωρία του αλγορίθμου του. Το δέντρο για όλες τις πιθανές κινήσεις που δημιουργείται έχει βάθος 4.

Στον πίνακα παρακάτω βλέπουμε ένα παράδειγμα εκτέλεσης, όπου φαίνεται ότι πολύ έξυπνα ο πράκτορας αποφασίζει να μειώσει την στάθμη της λίμνης κάτω από τον αντίπαλο του και τελικά να οδηγηθεί στην νίκη.

Αρχική κατάσταση



Κόκκινος Παίκτης	Πράσινος Παίκτης
 MV_RedRight	 MV_Water1Down
 MV_Water1Up	 MV_GreenLeft
 MV_RedRight	 MV_GreenLeft
 MV_RedRight	 MV_GreenLeft
 MV_RedRight	 MV_Water4Down
 MV_RedRight	 MV_Water5Down

Βιβλιογραφία

[1] http://en.wikipedia.org/wiki/Android_(operating_system)

[2] http://en.wikipedia.org/wiki/Dalvik_(software)

[3] http://developer.android.com/reference/android/app/Activity.html

[4] http://developer.android.com/develop/index.html

[5] http://en.wikipedia.org/wiki/Minimax

[6] Γκεζερλής Σπύρος (2012), "Μηχανισμοί Μάθησης και η χρήση τους στην επικύρωση της Ποιότητας Νέων Παιγνίων", Μεταπτυχιακή Διατριβή, Ανοικτό Πανεπιστήμιο Κύπρου

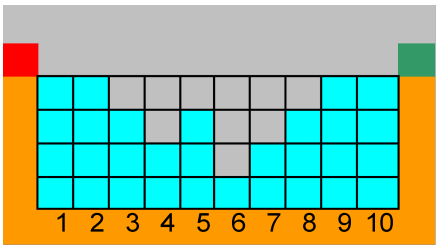
[7] Stuart Russell, Peter Norvig (2009) "Artificial Intelligence: A Modern Approach (3rd Edition)"

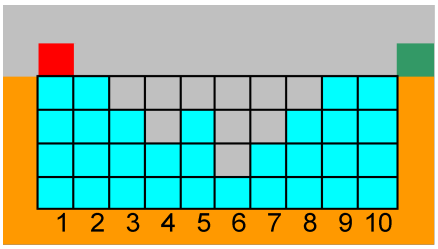
[8] Lauren Darcey, Shane Conder (2010), "Sams Teach Yourself Android Application Development in 24 Hours"

Παράρτημα Α

Επεξήγηση παραδείγματος με εικόνες του πράκτορα MiniMax

Στην πρώτη εικόνα βλέπουμε την αρχική κατάσταση του παιχνιδιού. Έχει προστεθεί ένα βοηθητικό πλέγμα ώστε να είναι ευκολότερο στον αναγνώστη να παρατηρεί τις κινήσεις, από εικόνα σε εικόνα. Όπως έχει προαναφερθεί, κατά σύμβαση στην πρώτη κίνηση δικαίωμα έχει ο κόκκινος παίκτης. Αν υποθέσουμε ότι κινείται ένα βήμα μπροστά, η επόμενη εικόνα απεικονίζει την κατάσταση.

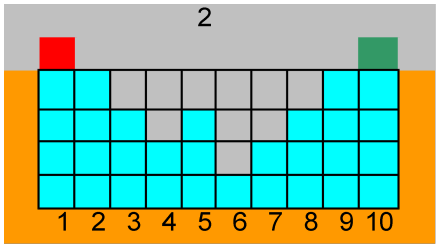


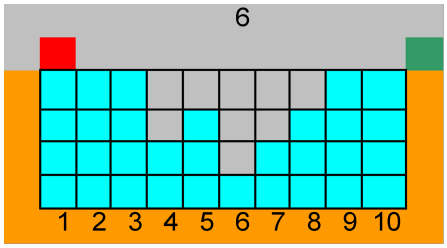


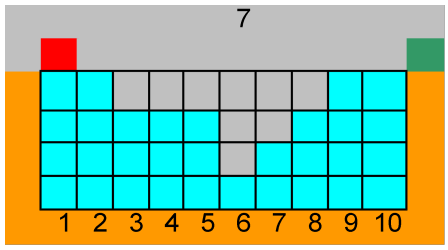
Από το σημείο αυτό, αναλαμβάνει ο πράκτορας αλγόριθμος MiniMax να βρει τη βέλτιστη για αυτόν κίνηση. Έτσι θα εξετάσει όλους τους πιθανούς συνδυασμούς κινήσεων σε βάθος τέσσερα (4). Θεωρώντας ότι η νέα αρχική κατάσταση και ρίζα του δέντρου που θα δημιουργηθεί είναι η εικόνα 2, οι νόμιμες κινήσεις που μπορεί να εκτελέσει σε πρώτο επίπεδο είναι 17 και φαίνονται στην επόμενη σελίδα.

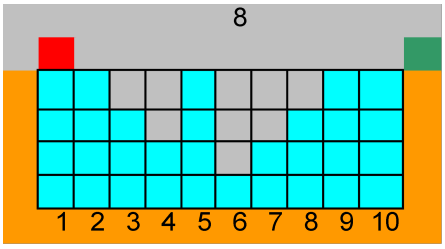
Κάθε εικόνα περιλαμβάνει σαν επικεφαλίδα τον αριθμό της κίνησης. Οι πιθανές κινήσεις που μπορούν να πραγματοποιηθούν καθ' όλη τη διάρκεια του παιχνιδιού είναι 24 και φαίνονται στον πίνακα:

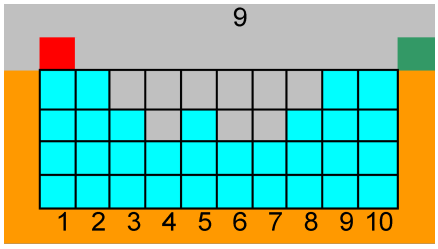
0	MV_RedLeft	6	MV_Water3Up	12	MV_Water9Up	18	MV_Water5Down
1	MV_RedRight	7	MV_Water4Up	13	MV_Water10Up	19	MV_Water6Down
2	MV_GreenLeft	8	MV_Water5Up	14	MV_Water1Down	21	MV_Water7Down
3	MV_GreenRight	9	MV_Water6Up	15	MV_Water2Down	22	MV_Water8Down
4	MV_Water1Up	10	MV_Water7Up	16	MV_Water3Down	23	MV_Water9Down
5	MV_Water2Up	11	MV_Water8Up	17	MV_Water4Down	23	MV_Water10Down

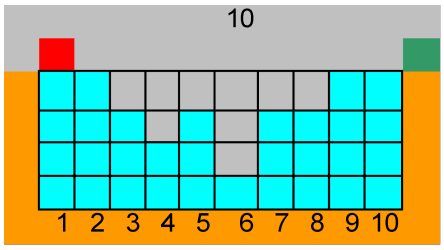

2
1 2 3 4 5 6 7 8 9 10

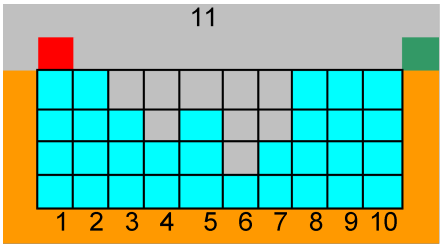

6
1 2 3 4 5 6 7 8 9 10

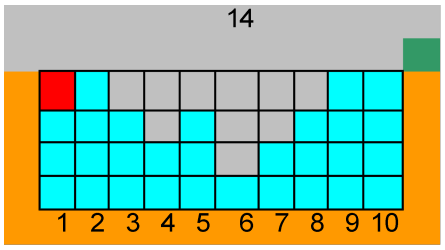

7
1 2 3 4 5 6 7 8 9 10

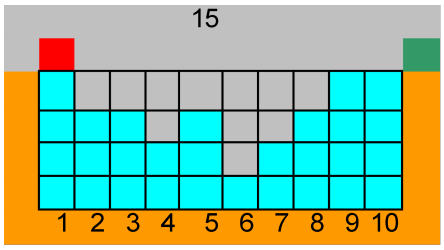

8
1 2 3 4 5 6 7 8 9 10

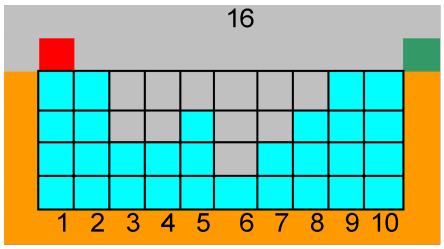

9
1 2 3 4 5 6 7 8 9 10

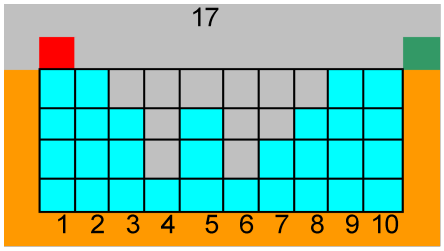

10
1 2 3 4 5 6 7 8 9 10

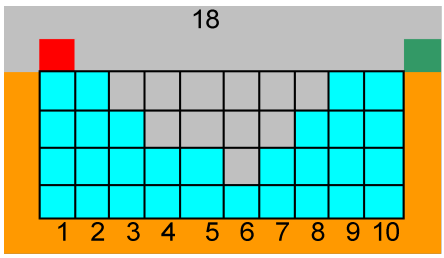

11
1 2 3 4 5 6 7 8 9 10

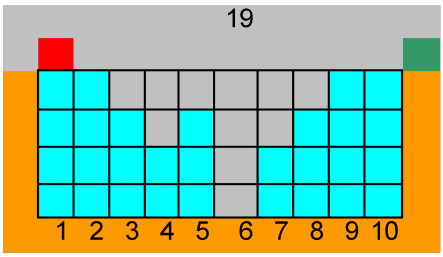

14
1 2 3 4 5 6 7 8 9 10

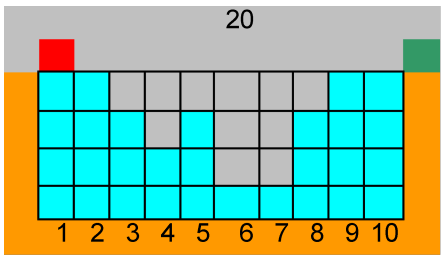

15
1 2 3 4 5 6 7 8 9 10

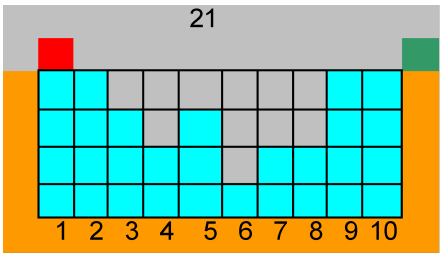

16
1 2 3 4 5 6 7 8 9 10

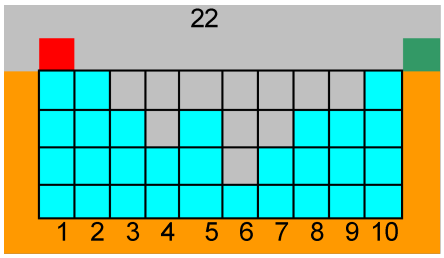

17
1 2 3 4 5 6 7 8 9 10

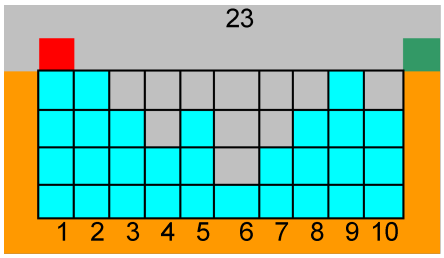

18
1 2 3 4 5 6 7 8 9 10


19
1 2 3 4 5 6 7 8 9 10

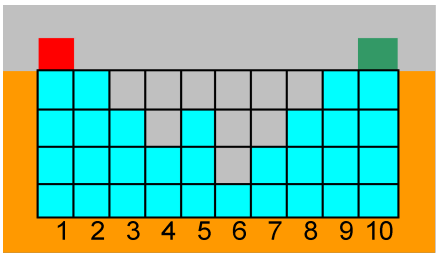

20
1 2 3 4 5 6 7 8 9 10

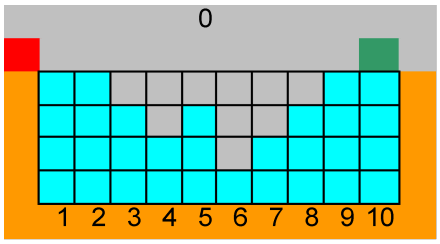

21
1 2 3 4 5 6 7 8 9 10

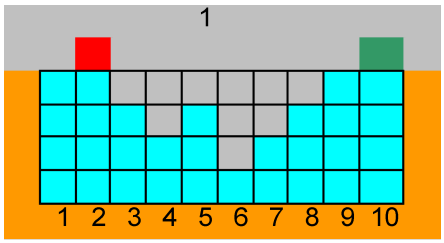

22
1 2 3 4 5 6 7 8 9 10

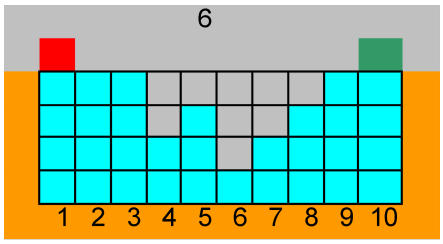

23
1 2 3 4 5 6 7 8 9 10

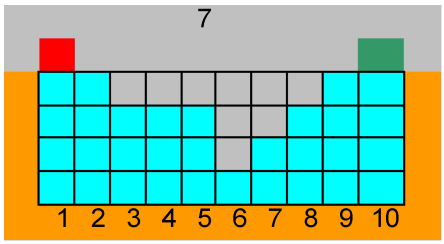
Στη συνέχεια ο πράκτορας MiniMax υπολογίζει όλες τις νόμιμες κινήσεις που θα κάνει ο αντίπαλος για κάθε μια από τις 17 κινήσεις που θεωρητικά θα μπορούσε να εκτελέσει ο ίδιος. Για λόγους απλότητας στην απεικόνιση θα αναπτύξουμε μόνο την πρώτη περίπτωση από τις 17. Έτσι υπόριζο του υπό σύσταση δέντρου ορίζεται το παρακάτω στιγμιότυπο και όλες οι πιθανές καταστάσεις που γεννά και καταλαμβάνουν τις θέσεις των φύλλων.

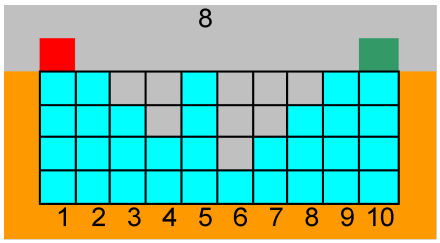


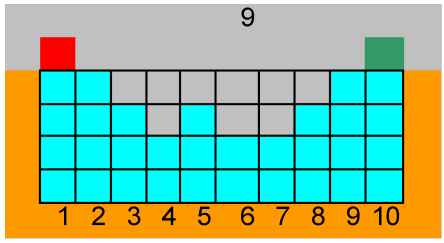


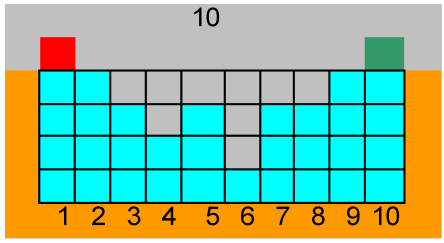


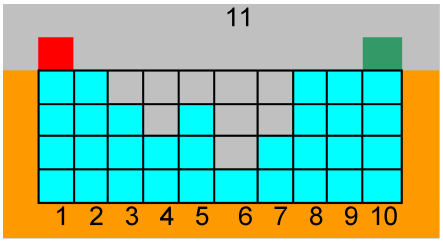


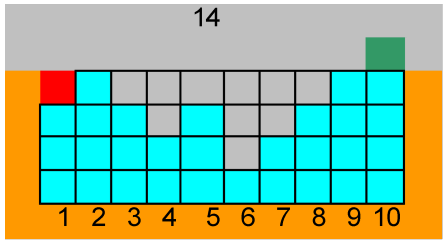


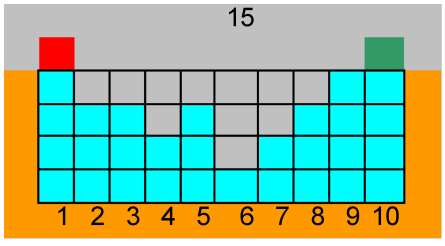


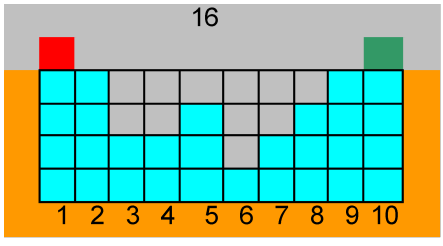


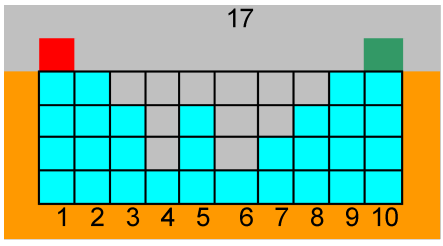


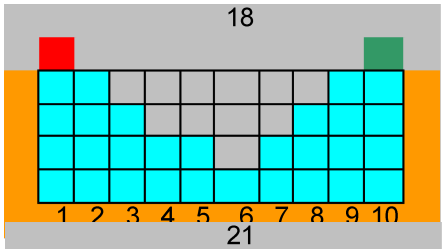


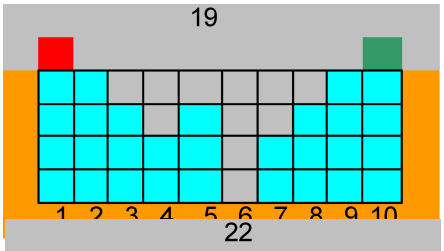


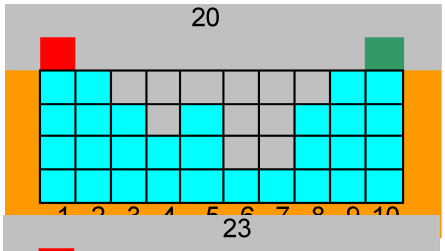


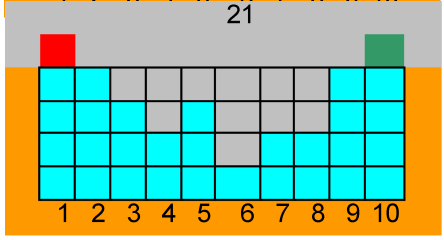


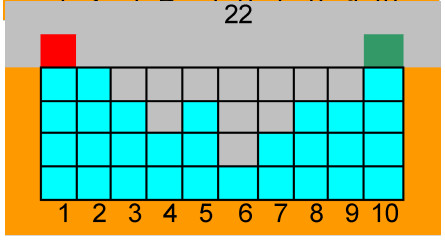


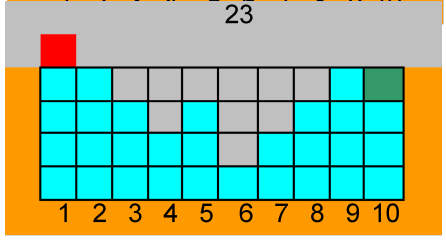




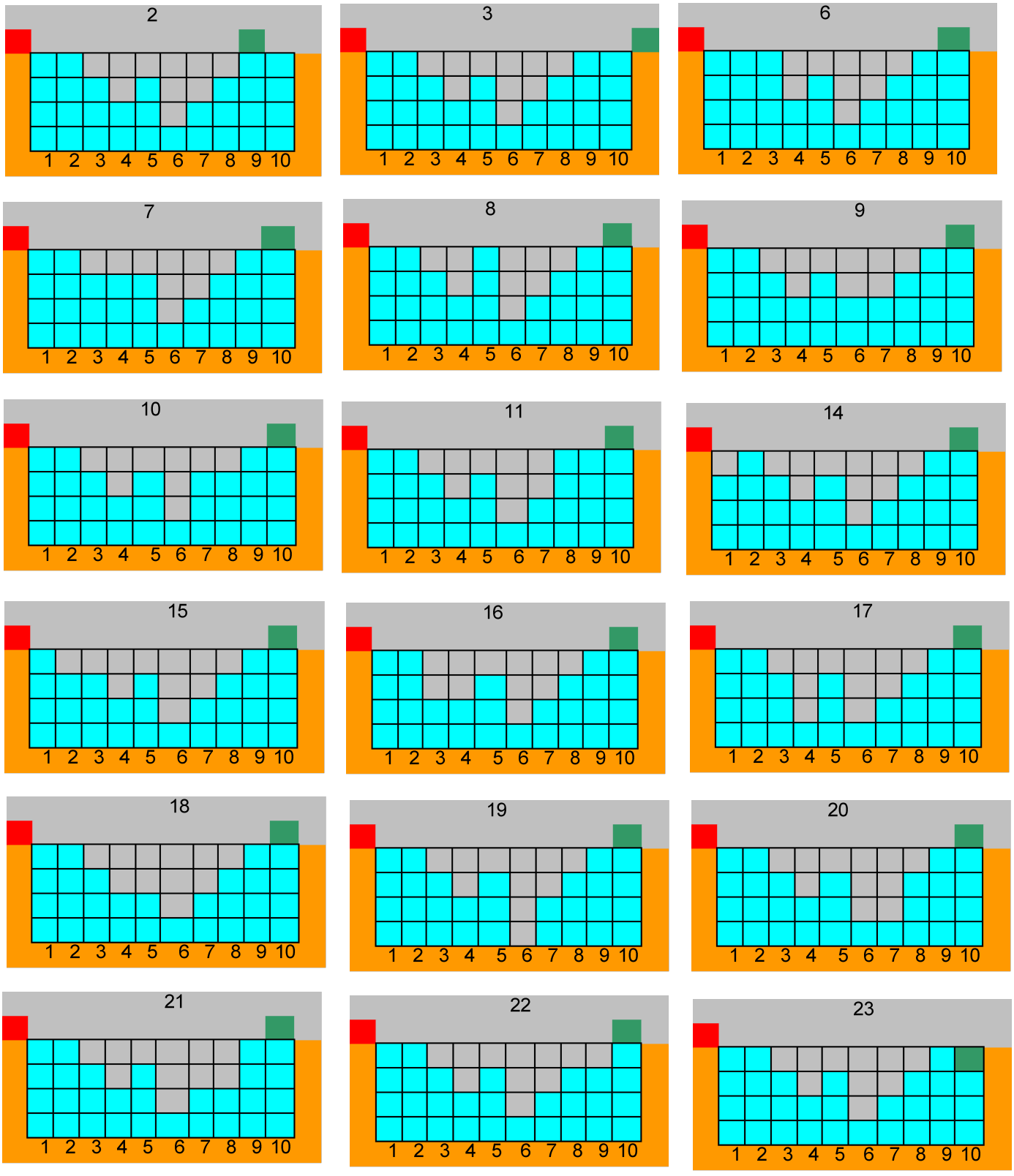




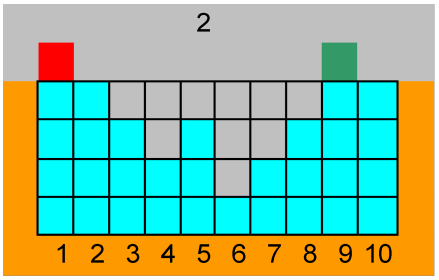


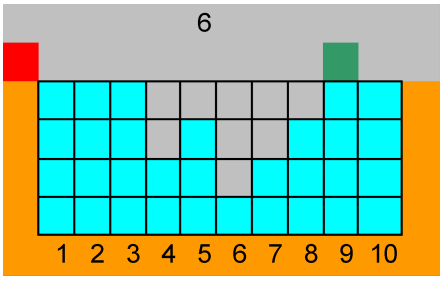


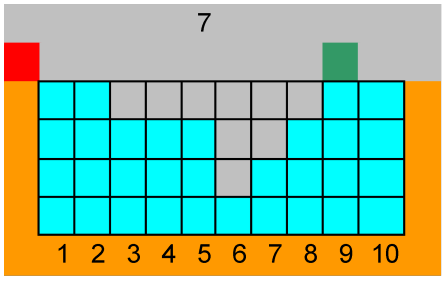
Εφόσον ολοκληρωθεί η διαδικασία για όλες τις πιθανές περιπτώσεις του επιπέδου 2 του δέντρου, σειρά έχει η δημιουργία του επιπέδου 3, όπου κάθε φύλλο του επιπέδου 2 γίνεται νέο υπόριζο και σύμφωνα με αυτό, τα φύλλα του επιπέδου 3 θα αποτελούνται από όλες τις πιθανές κινήσεις του πράκτορα MiniMax. Για λόγους απλότητας θα αναπτύξουμε μόνο την πρώτη περίπτωση από τις 18.

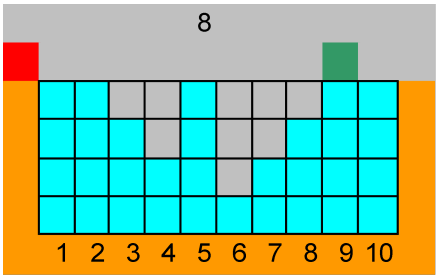


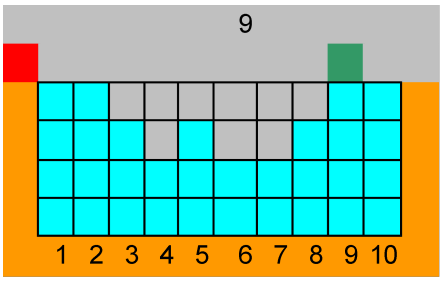
Στο τέταρτο και τελευταίο επίπεδο, όπου τα φύλλα θα περιλαμβάνουν όλες τις πιθανές κινήσεις του αντιπάλου (κόκκινος παίκτης), υπολογίζεται η βαθμολογία της κατάστασης του παιχνιδιού.

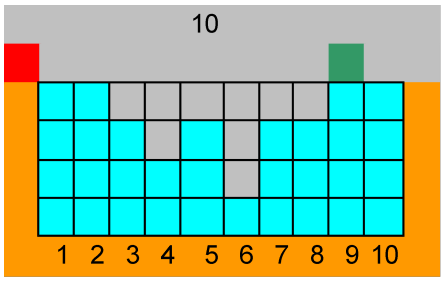


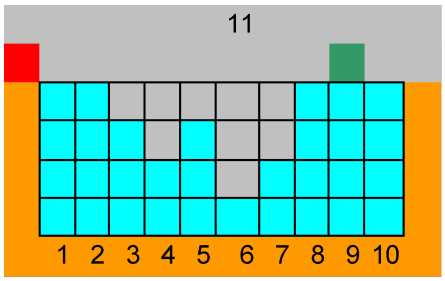


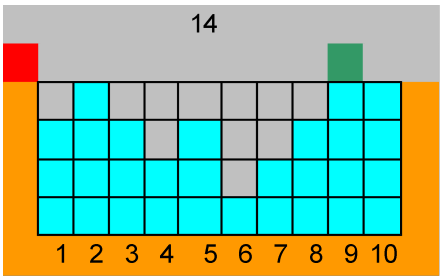


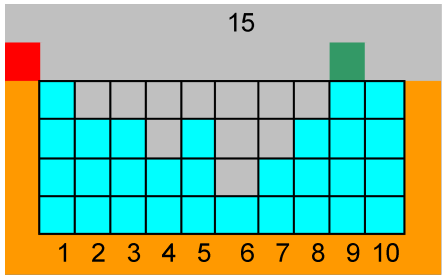


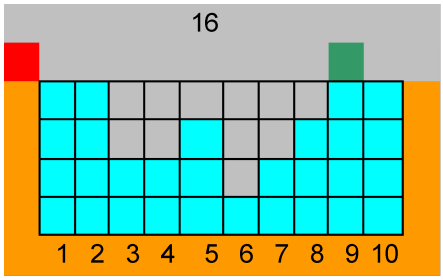


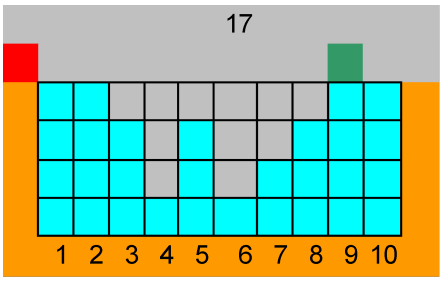


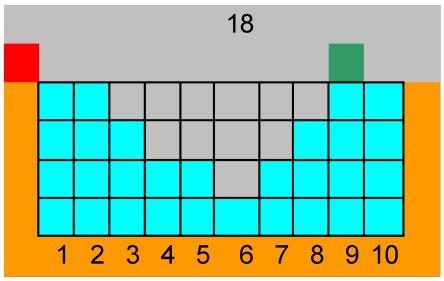


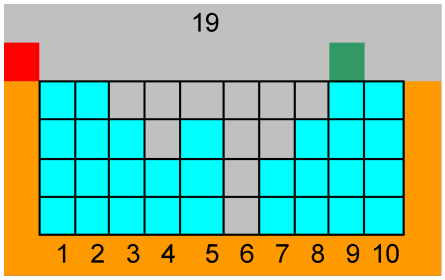


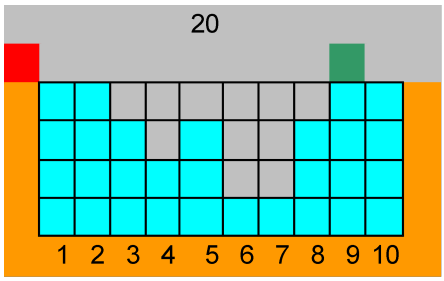


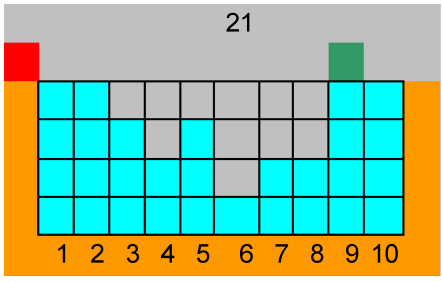


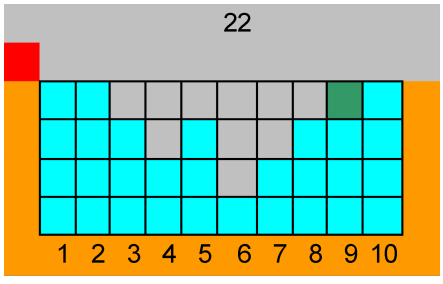


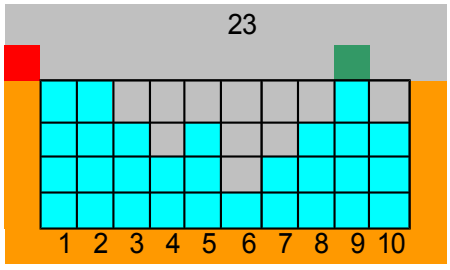










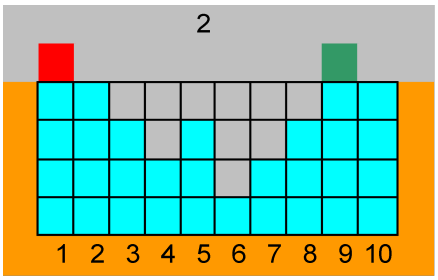


Σύμφωνα με την βαθμολογία αυτή και με την αναδρομική διαδικασία που περιγράφεται στη συνέχεια, ο αλγόριθμος MiniMax θα αποφασίσει ποια είναι η πιο συμφέρουσα κίνηση για τον παίκτη που κινεί, δηλαδή το πράσινο καράβι. Ο υπολογισμός της βαθμολογίας προκύπτει από τη διαφορά της απόστασης των δυο αντιπάλων από το σημείο νίκης τους, δηλαδή το αντίπαλο καρνάγιο.

Αν εξετάσουμε για παράδειγμα το πρώτο φύλλο του δέντρου που δημιουργείται και βρίσκεται αριστερά, τότε η βαθμολογία είναι: `e = _distance_green - _distance_red;` όπου για _distance_green ισχύει `int _distance_green = 2*_green_column + (pos.getLevel(_green_column));` και για _distance_red ισχύει `int _distance_red = 2*(11 - _red_column) + (pos.getLevel(_red_column));` Επίσης μειώνουμε κατά μια μονάδα τη βαθμολογία του παίκτη που έχει σειρά να παίξει.

```
if (pos.getTurn() == Position.RED) { _distance_red—; }
if (pos.getTurn() == Position.GREEN) {_distance_green--;}
```

Συμπερασματικά η τελική βαθμολογία του φύλλου που φαίνεται παρακάτω είναι -3.



Ομοίως η ίδια διαδικασία θα πραγματοποιηθεί για τα επόμενα 16 φύλλα, όπου οι τιμές φαίνονται στον παρακάτω πίνακα.

2	6	7	8	9	10	11	14	15	16	17	18	19	20	21	22	23
-3	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-4	-5

Επειδή ο κόκκινος παίκτης μεγιστοποιεί το κέρδος του, η πιο συμφέρουσα επιλογή είναι η βαθμολογία -3 και έτσι η κίνηση 2 θα περάσει στη ρίζα του επιπέδου 3 και στην συνέχεια στον πρόγονο της ρίζας του επιπέδου 3, δηλαδή στη ρίζα του επιπέδου 2.

Σειρά έχει να λάβει τιμή η δεύτερη ρίζα του επιπέδου 3 και για αυτό τον λόγο θα επαναληφθεί η ίδια διαδικασία για τα 17 φύλλα της.

Από τα 17 φύλλα της επιλέγεται η κίνηση με τη μεγαλύτερη βαθμολογία, καταγράφεται και συγκρίνεται με αυτή της ρίζας 2, δηλαδή το -3. Αν η νέα τιμή είναι μικρότερη από την προηγούμενη καλύτερη, τότε λαμβάνει τη θέση της.

Αφού ολοκληρωθεί η διαδικασία για όλα τα φύλλα του επιπέδου 3, που ανήκουν στην πρώτη ρίζα του επιπέδου 2, σειρά έχουν όλα τα παιδιά της δεύτερης ρίζας του επιπέδου 2, όπου επαναλαμβάνεται όλη αυτή η διαδικασία.

Καθώς ο πράσινος παίκτης ελαχιστοποιεί το κέρδος του, η ρίζα του επιπέδου 2 θα λάβει σαν τιμή την μικρότερη από τις βαθμολογίες του επιπέδου 3.

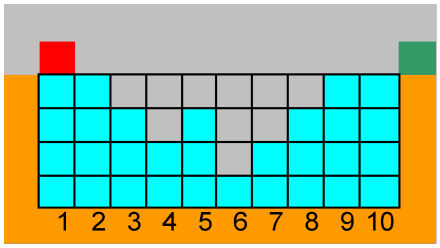
Αντίστοιχα οι ρίζες του επιπέδου 1 θα λάβουν σαν τιμή την μεγαλύτερη βαθμολογία των απογόνων τους που βρίσκονται στο επίπεδο 2 και τελικά η κίνηση που θα πραγματοποιηθεί είναι αυτή με την μικρότερη τιμή του επιπέδου 1.

Παράρτημα Β

Επεξήγηση παραδείγματος με πραγματικές τιμές.

Στο παρόν παράρτημα θα επιχειρήσουμε να εκτελέσουμε τον πράκτορα MiniMax και να παρακολουθήσουμε πως διαμορφώνονται και εναλλάσσονται οι τιμές των μεταβλητών μέχρι να καταλήξει ο αλγόριθμος στην πιο συμφέρουσα για αυτόν τιμή.

Η αρχική κατάσταση που λαμβάνει ο αλγόριθμος για να επιχειρήσει να βρει τη βέλτιστη κίνηση φαίνεται στην εικόνα:

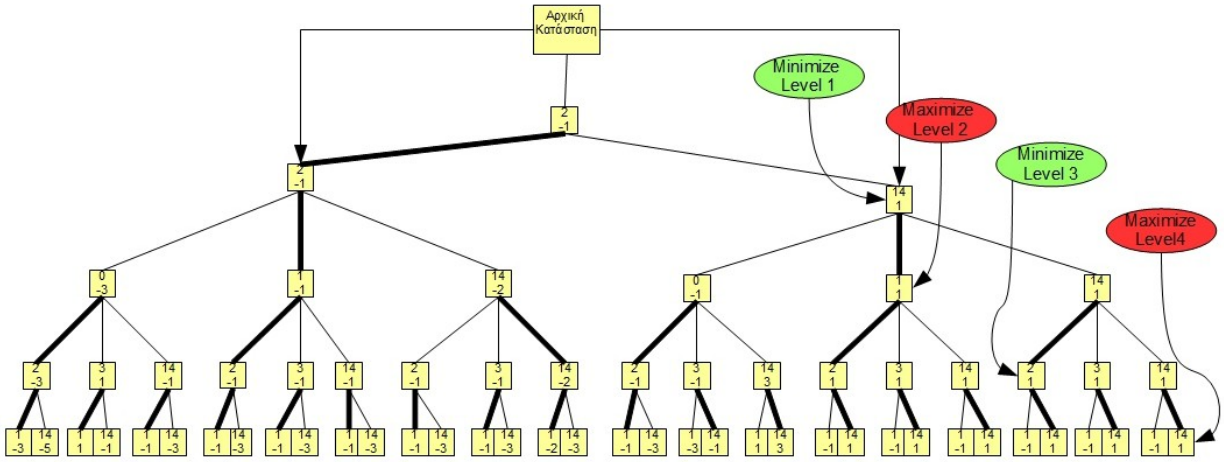


Αμέσως δημιουργείται ένας πίνακας τιμών με όλες τις νόμιμες κινήσεις που μπορεί να εκτελέσει το πράσινο βαρκάκι. Η κάθε κίνηση συμβολίζεται με έναν αριθμό, σύμφωνα με τον πίνακα του παραρτήματος Α.

2	6	7	8	9	10	11	14	15	16	17	19	20	21	22	23

Για λόγους απλότητας kai για να μπορεί να αναπαρασταθεί το δέντρο με βάθος 4 που παράγεται σε μια σελίδα, νόμιμες κινήσεις ua θεωρούνται οι κινήσεις των παικτών, δηλαδή MV_RedLeft, MV_RedRight, MV_GreenLeft, MV_GreenRight, καθώς επίσης η αύξηση και η μείωση της πρώτης στάθμης της λίμνης, δηλαδή MV_Water1Up και MV_Water1Down.

Έτσι στην εικόνα παρακάτω φαίνεται το δέντρο που δημιουργείται:



Σαν αρχική κατάσταση ορίζεται η κίνηση του κόκκινου παίκτη. Έπειτα ο πράκτορα Minimax έχει δικαίωμα να επιλέξει μεταξύ δυο κινήσεων. Δημιουργείται το δέντρο σε βάθος 4 και ξεκινάει αναδρομικά η αναζήτηση της βέλτιστης κίνησης.

Στο επίπεδο 4 και 2 επιλέγεται η μεγαλύτερη τιμή, καθώς ο κόκκινος παίκτης προσπαθεί να μεγιστοποιήσει το κέρδος του. Αντίθετα, στα επίπεδα 3 και 1 επιλέγεται η μικρότερη τιμή, καθώς ο πράσινος παίκτης προσπαθεί να ελαχιστοποιήσει το κέρδος του.

Η έντονη γραμμή δείχνει ποια βαθμολογία υπερισχύει κάθε φορά και τελικά ποια κίνηση επιλέγεται. Έτσι, στο συγκεκριμένο παράδειγμα προκύπτει ότι το πράσινο καράβι θα απαντήσει με μια κίνηση μπροστά.