ΑΝΑΦΟΡΑ 1ΗΣ ΕΡΓΑΣΙΑΣ ΣΤΟ ΜΑΘΗΜΑ ΤΕΧΝΗΤΗΣ ΝΟΗΜΟΣΥΝΗΣ (REVERSI)

Από τους φοιτητες:

ΒΛΑΝΤΗΣ ΔΗΜΗΤΡΙΟΣ - 3180021

ΓΕΩΡΓΙΟΥ ΑΛΕΞΙΟΣ-ΛΑΖΑΡΟΣ - 3180027

ΚΩΤΣΗΣ-ΠΑΝΑΚΑΚΗΣ ΒΑΣΙΛΕΙΟΣ-ΕΚΤΩΡ - 3180094

ΕΙΣΑΓΩΓΗ

Το περιεχόμενο αυτης της εργασίας αφορά την υλοποίηση προγράμματος προσωμοίωσης παιχνιδιού **Reversi (ή Othello)**. Στην παρτίδα, ο ένας παίκτης είναι ο **χρήστης**, και ο άλλος είναι ένας **αυτόματα χειρισμένος παίκτης**, ο οποίος είναι μια δικιά μας αναπτυγμένη τεχνητή νοημοσύνη με τις με προδιαγραφές και τεχνικές που διδάχτηκαν στο μάθημα. Ο χρήστης θα παίζει ενάντια στο AI και θα προσπαθεί να το κερδίσει. Το AI αυτό χρησιμοποιεί τον αλγόριθμο **MiniMax με πριονισμό α-β** προκειμένου να προβλέψει τις κινήσεις του χρήστη, και να λάβει την πιο κατάλληλη κίνηση (τουλάχιστον μέχρι το βάθος που μπορεί να δει). Ο χρήστης θα μπορεί να επιλέγει αν θα ξεκινήσει πρώτος ή δευτερος, καθώς και να επιλέγει την δυσκολία στην οποία θέλει να παίξει. Σε κάθε γύρο, ο τρέχων παίκτης μπορεί να τοποθετήσει ένα δικό του πιόνι μαύρο ή ασπρο όπου επιλέξει (στην εργασία μας X και O αντίστοιχα), υπο την προϋπόθεση ότι η κίνηση του δεν παραβιάζει κάποιον από τους κανόνες του παιχνιδιού. Για περισσότερες πληροφορίες για το παιχνίδι παραπέμπουμε τον παρακάτω σύνδεσμο: https://en.wikipedia.org/wiki/Reversi.

Σε αυτήν την αναφορά θα δώσουμε μια περιληπτική εξήγηση και καθοδήγηση στα βήματα που πάρθηκαν για την υλοποίηση αυτού του προγράμματος. Θα κοιτάξουμε ένα-ένα τα .java αρχεία τα οποία χρησιμοποιούμε και θα εξηγήσουμε την λογική και την λειτουργία των μεθόδων και των διαδικασιών που χρησιμοποιούμε.

1. Αρχείο Move.java

Αυτό το αρχείο περιέχει την κλάση Move, που απαιτείται για την αναπαράσταση μιας κίνησης στο παιχνίδι. Ένα αντικείμενο της κλάσης αυτής χαρακτηρίζεται από την γραμμή της κίνησης **(int row),** την στήλη της κίνησης **(int col)** και την αποτίμιση της ευρεστικης παίχτηκε **(int value)**. Ένα αντικείμενο αυτής της κλάσης θα δημιουργείται κάθε φορα που ο χρήστης ή το AI θα επιλέγει μια κίνηση για να παίξει στο παιχνίδι και κάθε φορα που το AI θα θέλει να αναλύσει τις δυνατές κινήσεις στο Minimax δέντρο.

2. Αρχείο Board.java

Αυτό το αρχείο περιέχει την κλάση με την οποία αναπαριστάνεται ένα ταμπλό στο παιχνίδι. Η κλάση αυτή περιέχει κάποιες στατικές και final μεταβλητές που χρησιμοποιούνται για να διευκολύνουν την αναγνωσιμότητα του κώδικα, οι οποίες είναι η X = 1, O = - 1 και EMPTY = 0. Δηλαδή λέμε πως το 1 θα συμβολίζει τα X, το -1 τα O και το 0 την άδεια θέση. Στη συνέχεια, ένα ταμπλο χαρακτιρίζεται από την κατάσταση στην οποία βρίσκεται (δηλαδή τι πιόνια έχει τις θέσεις του), κάτι που αποθηκεύται σε έναν δισδιάστατο πίνακα **(int[][] gameboard).** Επίσης, για κάθε αντικείμενο ταμπλό αποθηκεύουμε και το τελευταίο γράμμα που παίχτηκε **(int lastLetterPlayed)** και την τελευταία κίνηση που παίχτηκε **(Move lastMove).** Αυτές οι δύο ιδιότητες θα μας βοηθήσουν αργότερα στο να μπορούμε να βρίσκουμε την σειρά του παίκτη και για να μπορεί το AI να προβλέψει κινήσεις.

Το ταμπλό έχει πάντα την ίδια αρχική μορφή πριν αρχίσει το παιχνίδι, οπότε αυτό το δηλώνουμε στον default constructor της κλάσης. Πέρα από τους κλασσικούς getters και setters για κάθε ιδιότητα της κλάσης, η κλάση μας περιέχει και χρήσιμες μεθόδους για την ορθή διεξαγωγή του παιχνιδιού. Ας αναλύσουμε τις πιο σημαντικές μια μια:

- **void makeMove(int row, int col, int letter)**

  Αυτή η μέθοδος καλείται κάθε φορά που γίνεται μια έγκυρη κίνηση. Μέσα απ' αυτήν, το ταμπλό ανανεώνεται, εκχωρώντας το εκάστοτε γράμμα στην άδεια θέση που δόθηκε σαν όρισμα. Επίσης, ανανεώνεται το lastLetterPlayed στην τιμή του νέου που παίχτηκε (letter) και η τελευταία κίνηση lastMove. Τέλος αλλάζουν τιμή τα αντίπαλα πιόνια, το οποία γίνεται χρησιμοποιώντας την βοηθητική συνάρτηση **capture(row, col, letter).**

- **void capture(int row, int col, int letter)**

  Βοηθητική συνάρτηση που αλλάζει τιμή στα πιονια που πιάνονται. Χρησιμοποιεί τις βοηθητικές συναρτήσεις **int [] checkDirection** όπου «Direction» λέμε μια από τις 8 κατευθύνσεις στην πιξίδα του χάρτη. Δηλαδή έχουμε 8 διαφορετικές συναρτήσεις. Για κάθε μία από αυτές τις συναρτήσεις που επιστρέφει κάποια valid θέση για πιόνι που βρέθηκε (αν δεν βρεθει κανένα πιονι η συνάρτηση checkDirection επιστρέφει τον πίνακα [-1,-1]) η συνάρτηση capture αλλάζει τις τιμές όλων των ενδιάμεσων πιονιών του αντιπάλου σε αυτές που αντιστοιχούν στου παίκτη που έκανε το capture σε αυτην την συγκεκριμένη γραμμή (από X σε O ή το αντίθετο).

- **ArrayList<Board> getChildren (int letter)**

  Αυτή η συνάρτηση επιστρέφει σε μορφή λίστα όλα τα ταμπλό που μπορούν να προκύψουν άμεσα με μια κίνηση από το gameboard που την καλεί. Για παράδειγμα, αν στο ταμπλό που την καλεί, ο τρέχων παίκτης

μπορεί να κάνει 7 διαφορετικές κινήσεις, τότε η λίστα που θα επιστραφεί θα έχει ένα διαφορετικό ταμπλό για κάθε κίνηση που μπορεί να συμβεί. Όμως, αν ο παικτης δεν μπορεί να κάνει κίνηση (ανιχνέυεται μέσω της **boolean hasMove**), τότε απλά επιστρέφεται μια λίστα με ένα στοιχείο, όπου το στοιχείο αυτό είναι το εκάστοτε ταμπλό χωρίς αλλαγές. Αυτή η συνάρτηση χρησιμοποιείται αποκλειστικά και μόνο για να μπορεί να λειτουργήσει το AI, καθώς την χρησιμοποιεί για να τρέξει τον MiniMax αλγόριθμο. Αυτό είναι λογικό, εφόσον πρέπει να γνωρίζει κάθε φορά τις επόμενες δυνατές κινήσεις τόσο του εαυτού του, όσο και του αντιπάλου του.

- **int evaluate()**

  Η μέθοδος evaluate είναι η **ευρετική συνάρτηση** που αξιοποιεί το AI προκειμένου να υπολογίσει ποιές καταστάσεις ταμπλό είναι αυτές που θα τον φέρουν πιο κοντά στην καλύτερη δυνατή γι'αυτόν. Στην άσκηση μας, διαλέξαμε τρία κριτηρια που βοηθούν το AI να καταλάβει πότε η κατάσταση του ταμπλό είναι υπερ του ή όχι. Πρώτον, το πόσα περισσότερα πιόνια έχει από τον αντίπαλο (**void evaluateSum**). Δεύτερον, το πόσα περισσότερα πιόνια στις πλευρές του ταμπλό έχει περισσότερα από τον αντίπαλο (**void evaluateSides**). Και τρίτον, πόσα περισσότερα πιόνια έχει στις άκρες του ταμπλό από τον αντίπαλο (**void evaluateCorners**). Κάθε ένα από τα παραπάνω κριτήρια, με αύξουσα σειρά, έχει διαφορετικό βάρος και προτεραιότητα στην ευρετική συνάρτηση (Τα βάρη είναι 1, 2 και 3 αντίστοιχα). Η evaluate επιστρέφει ένα ακέραιο βεβαρυμένο άθροισμα αυτών των τριών βοηθητικών συναρτήσεων. Σε περίπτωση που το AI κάνει evaluate κάποιο ταμπλο τερματικής κατάστασης, η evaluate ελέγχει αν σε αυτήν την τερματική κατάσταση νικάει αυτό ή ο αντίπαλος και. Αν νικάει αυτό, τότε η ευρετική συνάρτηση μεγιστοποιείται (μεγιστή προτεραιότητα). Η evaluate χρησιμοποιείται μόνο στα φύλλα του MiniMax δέντρου.

- **boolean isTerminal()**

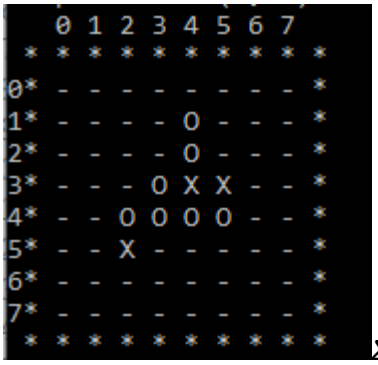
  Η συγκεκριμένη μέθοδος καλείται για να βρεθεί αν η τωρινή κατάσταση του ταμπλό είναι τερματική. Το μόνο που κάνει είναι να ελέγχει αν μπορεί να παίξει έστω ένας από τους δύο παίκτες σε αυτήν την κατάσταση. Αν μπορεί έστω και ένας, επιστρέφει **false**. Αν όχι, τότε **true.** Τον έλεγχο αυτό τον κάνει με την βοηθητική συνάρτηση **boolean canPlay(int letter)**. Η συνάρτηση isTerminal() καλείται κάθε φορά πριν παιξει οποιοσδήποτε από τους δύο παίκτες, καθώς και σαν το αναδρομικό κριτήριο του MiniMax.

- **Boolean canPlay(int letter)**

  Ελέγχει όλες τις θέσεις του ταμπλό και βλέπει αν μπορεί ο συγκεκριμένος παικτης που έχει το letter να παίξει εκει κάποιο γράμμα. Χρησιμοποιείται για την λειτουργία της isTerminal() αλλα και για να ελεγχθεί αν ένας παίκτης πρέπει να χάσει γύρω ή όχι.

- **Void printBoard()**

  Κάνει print σε φιλική προς τον χρήστη μορφή το ταμπλό που την καλεί. Στο σχήμα 1 φαίνεται ένα παράδειγμα της print.



Σχήμα 1

3. Αρχείο GamePlayer.java

Το αρχείο αυτό περιέχει την κλάση που αναπαριστά το AI μας. Οι ιδιότητες του, στις οποίες εκχωρούνται τιμές από τον χρήστη στην αρχή του παιχνιδιού, είναι το μέγιστο βάθος που θα κάνει Minimax (**int maxDepth**) ή αλλιώς η «δυσκολεία» του και το γράμμα με το οποίο θα παίξει το παιχνίδι (**int playerLetter).** Πέρα από αυτές τις ιδιότητες, περιέχει και τον αλγόριθμό **MiniMax** τον οποίο θα καλεί μια φορά για να βρεί την βέλτιστη κίνηση, έχοντας σαν δεδομένο πως ο χρήστης αντίπαλος του παίζει με τον καλύτερο δυνατό τρόπο (ο οποίος καλύτερος τρόπος δίνεται από μια αποδεκτή ευρετική συνάρτηση, σαν την δικιά μας). Οι μέθοδοι έχουν ως εξής:

- **Move MiniMax(Board board)**

  Η συνάρτηση αυτή καλείται για να επιλέξει την βέλτιστη κίνηση. Μέσα σε αυτήν καλούνται οι συναρτήσεις min ή max, ανάλογα με το γράμμα που έχει επιλέξει ο χρήστης για το AI. Όταν το AI έχει το X, τότε καλείται η max (δηλαδή η ευρετική πρέπει να μεγιστοποιηθεί) ενώ όταν έχει το O, τότε καλείται η min (δηλαδή η ευρετική πρέπει να ελαχιστοποιηθεί). Όποια και να καλεστεί γίνεται η έναρξη μιας αναδρομικής αναζήτησης του MiniMax δέντρου, που είναι ένα δέντρο που κάθε κόμβος του είναι μια κατάσταση ταμπλό που μπορεί να είναι προσβάσιμη από το board μετά από το πολύ maxDepth κινήσεις.

- **Move max(Board board, int depth)**

  Η μέθοδος max καλείται όταν η σειρά του παικτη στο MiniMax δέντρο είναι αυτή του X. Γενικά, η max ελέγχει αναδρομικά το δέντρο ώστε να βρει την βέλτιστη κίνηση για να επιλέξει το AI. Ελέγχει αν η τρέχουσα κατάσταση είναι τερματική ή εάν έχουμε φτάσει στο μέγιστο βάθος (maxDepth) του AI. Εάν συμβαίνει έστω και ένα από τα δύο, υπολογίζεται η ευρετική τιμή αυτής

της κατάστασης και επιστρέφεται ώστε να συγκριθεί με άλλες υποψήφιες βέλτιστες κινήσεις. Εάν όχι, αναπτύσσει τα παιδιά της τρέχουσας κατάστασης (δηλαδή όσες καταστάσεις είναι άμεσα προσβάσιμες με μια κίνηση). Το ποιες καταστάσεις θα αναπτυχθούν εξαρτάται άμεσα από τον παίκτη που παίζει αυτήν την στιγμή, καθώς μπορεί να παίξει 2 φορές στην σειρα, εάν δεν μπορεί να παιξει ο αντίπαλος του. Δηλαδή, εάν παίζει πάλι ο X, αναπτύσσονται τα παιδιά του O (που ουσιαστικά αφού δεν έχει παίξει ο O, είναι η ίδια κατάσταση του εκάστοτε ταμπλό) και καλείται ξανα αναδρομικά η max, αλλιώς αν παίζει ο O καλείται η min και αναπτύσσονται τα παιδιά του O. Η μέθοδος επίσης χρησιμοποιεί **πριονισμό α-β** ο οποίος είναι ένας τρόπος για να αποκλείσουμε μονοπάτια πάνω στο δέντρο τα οποία δεν μπορούν να μας δώσουν κάποια κατάσταση με πιο μεγάλη αποτίμηση ευρετικής συνάρτησης.

- **Move min(Board board, int depth)**
  Λειτουργεί παρόμοια με την max, προσπαθώντας όμως να ελαχιστοποιήσει αντί να μεγιστοποιήσει την αποτίμηση της ευρετικής συνάρτησης

4. Αρχείο Main.java

Σε αυτό το αρχείο περιέχεται ο κώδικας που εκτελείται για την διεξαγωγή του παιχνιδιου. Όταν εκτελείται, ο χρήστης ζητείται να πληκτρολογίσει αρχικά την δυσκολία (**int maxDepth**) που θέλει να παίξει καθώς και την σειρά του. Η δυσκολία θα χρησιμοποιηθεί για να κριθεί το πόσο βαθιά θα διανύει ο MiniMax το δέντρο καταστάσεων. Εάν ο χρήστης επιλέξει να παίξει πρώτος, τότε θα πάρει τα X, αλλιώς θα πάρει τα O. Το AI θα πάρει το γράμμα που έμεινε. Για όσο το ταμπλό δεν βρίσκεται σε τερματική κατάσταση το πρόγραμμα δίνει σειρά στον παίκτη που είναι σειρά του να παίξει. Χρησιμοποιώντας την μέθοδο **canPlay(int letter)** κρίνεται αν μπορεί να παίξει ή όχι. Αν δεν μπορεί, ο γύρος του χάνεται και παίζει ο επόμενος παίκτης. Αν μπορεί, τότε επιλέγει (με την μορφή «?,?» απαραίτητα) την γραμμή και τη στήλη που θελει να καταχωρηθεί η κίνηση του. Εάν αυτή η κίνηση δεν είναι νόμιμη, τότε ξαναδιαλέγει κίνηση μέχρις ότου να είναι valid. Στο τέλος κάθε κίνησης εκτυπώνεται με την μέθοδο printBoard() η τρέχουσα κατάσταση του παιχνιδιού. Μόλις βρεθεί τερματική κατάσταση, τότε υπολογίζεται με την μέθοδο evaluate() το πρόσημο του νικητή. Εάν αυτό είναι θετικό, τότε κερδίζει όποιος έχει τα X, ενώ αν είναι αρνητικό, τότε κερδίζει αυτός που έχει τα O. Αν είναι 0, τότε το παιχνίδι είναι ισοπαλία.

Σημείωση: Χρησιμοποιήθηκε κώδικας που έχει διδαχθεί στο εργαστήριο του μαθήματος με τις κατάλληλες τροποποιήσεις