ΤΕΧΝΗΤΗ ΝΟΥΜΟΣΥΝΗ

ΑΝΑΦΟΡΑ ΕΡΓΑΣΙΑΣ 1

ΧΡΙΣΤΙΝΑ ΑΙΜΙΛΙΑ ΦΛΑΣΚΗ 3180195

ΓΕΩΡΓΙΑ ΨΥΧΑ 3190225

Κλάσεις -> ....

Μέθοδοι-> ...

Μεταβλητές-> ...

REVERSI(OTHELLO) (ΠΡΟΒΛΗΜΑ 3)

Σε αυτό το πρόγραμμα υλοποιείται το Othello version του παιχνιδιού αν και αναφερόμαστε μέσα στο πρόγραμμα σε αυτό ως Reversi(λόγω της ονομασίας του προβλήματος στην εκφώνηση).

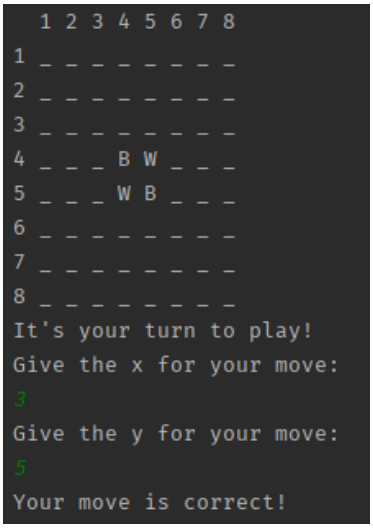
Για την επίλυση του προβλήματος αυτού ξεκινήσαμε με την main να καλεί την μέθοδο Reversi_Othello() όπου πρόγραμμα ζητάει από τον χρήστη να μας δώσει την σειρά με την οποία θέλει να παίξει και το βάθος που θέλει να έχει ο αλγόριθμος MiniMax(Depth()), εμφανίζοντας τα παρακάτω μηνύματα.

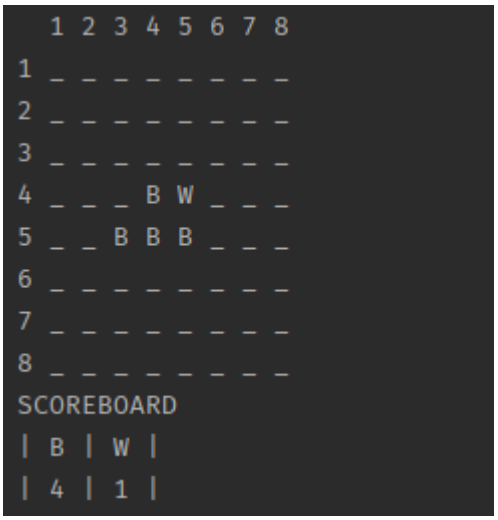
```
Press 1 if you want to play first
else press 2 to play second
1
Enter the depth of MiniMax:
```

Ανάλογα την σειρά που επέλεξε ο χρήστης δημιουργείται ένα αντικείμενο τύπου MiniMax με τα κατάλληλα δεδομένα. Πολύ σημαντικό ρόλο έχει η μεταβλητή current_color η οποία καθορίζει το ποιος παίζει ανάλογα με την τιμή που περιέχει (1 για μαύρο και 2 για άσπρο), αρχικοποιείται με μαύρο αφού τα μαύρα παίζουν πρώτα. Έπειτα δημιουργείται η λίστα myMoves, για την δημιουργία της χρησιμοποιούμε το TheBoard(το ταμπλό του παιχνιδιού) και γεμίζουμε την λίστα με την available_moves(int currentcolor).

Η available_moves(int currentcolor) βρίσκει τις γειτονικές θέσεις πάνω στον πίνακα με βάση την θέση μας, ελέγχει αν αυτές είναι άδειες ή αν έχουν το χρώμα του αντιπάλου και προσθέτει στην λίστα τις κινήσεις που μπορεί να κάνει ο χρήστης σύμφωνα με τους κανόνες του παιχνιδιού.

Για την επιλογή της κίνησης, στην οθόνη του χρήστη θα εμφανιστεί κάποιο αντίστοιχο μήνυμα όπως το παρακάτω μαζί με τον ανανεωμένο πίνακα:





Αν ο παίκτης εισάγει κίνηση η όποια παραβιάζει τους κανόνες του παιχνιδιού και κατ' επέκταση δεν περιλαμβάνεται στην λίστα με τις διαθέσιμες κινήσεις εμφανίζεται κατάλληλο μήνυμα και ζητιούνται ξανά οι συντεταγμένες της κίνησης του.

```
Your move is incorrect, give another one
```

Ο χρήστης πληκτρολογεί τις συντεταγμένες της κίνησης που θέλει να κάνει και το πρόγραμμα καλεί την makeMove(Board temp, Move move, int currentcolor), από την κλάση Board. Η makeMove παίρνει την κίνηση, το χρώμα και τον πίνακα και επιστρέφει τον ανανεωμένο πίνακα αφού κάνει αλλαγές με βάση την κίνηση που της δόθηκε, στο τέλος καλεί την heuristic_evalution(). Η heuristic_evalution() αποτελεί μια συνάρτηση αξιολόγησης, μέσα στην οποία έχει δημιουργηθεί πίνακας board_value με τα βάρη(αξίες) για κάθε θέση*, για κάθε θέση προσθέτει το βάρος της στην αντίστοιχη μεταβλητή για κάθε χρώμα(b για τα μαύρα,w για τα λευκά) με αυτόν τον τρόπο υπολογίζεται πόσο ωφέλιμη είναι μια κίνηση για τον παίκτη.

Μόλις η σειρά του χρήστη τελειώσει τότε η κίνηση του υπολογιστή πραγματοποιείται μέσω του αλγορίθμου MiniMax, καλώντας την μέθοδο MiniMax(Board board), που βρίσκεται στην MiniMax, η οποία υλοποιεί αυτόν τον αλγόριθμο. Ενώ παράλληλα στην οθόνη του χρήστη εμφανίζεται το εξής μήνυμα:

```
It's computer's turn to play!
Loading...
```

Στην περίπτωση μας έχουμε διαλέξει ο χρήστης να παίξει πρώτος όποτε ο υπολογιστής παίζει δεύτερος και η MiniMax χρησιμοποιεί την min(Board board, int a, int b, int depth) για να επιλέξει την κίνηση του υπολογιστή και ανανεώνει τον πίνακα. Αν ο υπολογιστής έπαιζε πρώτος τότε η επιλογή της κίνησης θα γινόταν με την max(Board board, int a, int b, int depth). Μετά από την κίνηση του υπολογιστή

στην οθόνη του χρήστη εμφανίζεται ο ανανεωμένος πίνακας και το νέο σκορ που έχει προκύψει με βάση τον αριθμό από πούλια για κάθε χρώμα.

Η διαδικασία αυτή με την επιλογή της κίνησης του χρήστη και την επιλογή της κίνησης του υπολογιστή επαναλαμβάνεται μέχρι να γεμίσει ο πίνακας με πούλια, αυτό μας γίνεται γνωστό μέσα στο πρόγραμμα όταν η isTerminal() είναι true.

Σε αυτό το σημείου του προγράμματος χρησιμοποιούνται οι εξής μέθοδοι τεχνητής νοημοσύνης: Ο αλγόριθμος MiniMax με πριόνισμα α-β και η ευρετική συνάρτηση αξιολόγησης.

Ο αλγόριθμος αυτός αυτό που κάνει είναι:

- Όταν η σειρά να παίξει είναι του Min τότε πραγματοποιείται πριόνισμα χρησιμοποιώντας το καλύτερο φράγμα α των Max προγόνων του.
- Όταν η σειρά να παίξει είναι του Max τότε πραγματοποιείται πριόνισμα χρησιμοποιώντας το καλύτερο φράγμα β των Min προγόνων του.

Η ευρετική συνάρτηση αξιολόγησης πραγματοποιείται μετά από κάθε κίνηση του παίκτη και η χρήση της είναι ότι προσπαθεί να βγάλει μια εκτίμηση για το πόσο ωφέλιμη είναι κάθε πιθανή κίνηση που βρίσκεται στους κόμβους με μέγιστο βάθος.

Μόλις τελειώσει η while(TheBoard.isTerminal()==false){} επανάληψη καλείται η Winner() που βρίσκεται στην κλάση Board η οποία εκτυπώνει ανάλογο μήνυμα για τον νικητή

⇒ Black checkers win! Αν κέρδισε ο παίκτης με τα μαύρα πούλια.
⇒ White checkers win! Αν κέρδισε ο παίκτης με τα άσπρα πούλια.
⇒ It's a tie! Αν υπολογιστής και χρήστης ήρθαν σε ισοπαλία.

*Τα βάρη για τον πίνακα board_value τα πήραμε από ένα επιστημονικό άρθρο που βρήκαμε στο διαδίκτυο από το University of Washington.

https://courses.cs.washington.edu/courses/cse573/04au/report.html ~ Othellus

```
private int boardWeight[][] = {
    { 100, -10, 11,  6,  6, 11, -10, 100 },
    { -10, -20,  1,  2,  2,  1, -20, -10 },
    {  10,   1,  5,  4,  4,  5,   1,  10 },
    {   6,   2,  4,  2,  2,  4,   2,   6 },
    {   6,   2,  4,  2,  2,  4,   2,   6 },
    {  10,   1,  5,  4,  4,  5,   1,  10 },
    { -10, -20,  1,  2,  2,  1, -20, -10 },
    { 100, -10, 11,  6,  6, 11, -10, 100 }
};
```

Figure 7 – Board weights matrix

The Othellus Heuristics

Finally we arrive to the part we all have been waiting for: the heuristics. What is then a heuristic and how do we use it? A heuristic involves or serves "*as an aid to learning, discovery, or problem-solving by experimental and especially trial-and-error methods*"[1]. In this section, we focus on the (in our game) implemented heuristics. Intuitively, it is likely that not all features have a consistent importance throughout the game. For example, mobility is very important in the middle of the game while it is less significantly at the beginning and the end. But more on this and other small details later.

We have implemented and tested a fairly big number of different heuristics. Most of them are complements to each other and to our own two base ones, which uses position and mobility together with a (in the basic version) static board weight matrix to calculate the current utility.

Random move Before turning our attention to the more advanced heuristics, we would like to start with a very simple method that almost can't even be considered as a real heuristic – taking a *random* move without doing any evaluation. The move is in the most basic version chosen among all possible legal moves available to a player; each one is equally weighted with the same probabilistic value, i.e. every move is a likely to be taken as the next. This can be considered as the play of a very (!) inexperienced human player or as an AI without an implemented intelligence, and our testing of this purely random player gave us an terribly bad computer opponent.

Why do we then even spend time thinking about this seemingly useless heuristic? Well, a random element is useful in as we found at least one situation: to avoid having the same games played over and over again. Playing as human vs. computer (or for testing purposes, computer vs. computer) often leads to an repetitive game with a clear pattern, or a game that ends in a sort of playing deadlock (both computer players doing the same series of moves over and over again). To be able to test our other heuristics in many different situations, a random element therefore needs to be introduced in the test games, and therefore we chose to play human vs. computer for testing, while measuring the heuristics proficiency with an independent statistics evaluator (described in the Evaluation section below).

Board Weights Next step towards building a more complex heuristic, is our simple strategy idea of assigning weights to the board squares according to their position and choose on each turn to play on a square of best weight.

The basic underlying idea is a board matrix containing the weights. But how does one come up with a correctly balanced set of weights for the board weight matrix? Well, combining together some basic game playing strategies we could come up with a good starting position for the weights and then adjusting them as we went along with playing. Ideally, to come up with a near-optimal distribution of board weights, one would like to train the matrix successively by using machine learning. In this case, there just was not time enough for that.

As soon realized by any player (even at the lowest level), having the corners is always good. These discs cannot be re-flipped, as argued earlier. By the same argument, the fields directly adjacent to the corners are usually bad to be the first to take, since this gives your opponent a good chance of taking the corner, while shrinking your options of taking it yourself. Furthermore, the squares even one step further out from the corner are good to take, to gain a good striking position for the corner. These ideas are summarized in Figure 6.

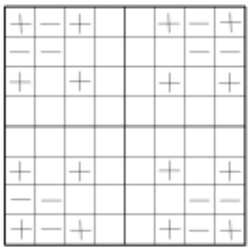


Figure 6 – The board estimation

As can be seen from the above figure, this reasoning still leaves us with 28 fields without any relative value, and the classified ones are only just that; relative to each other without any more precise measure.

So, how to proceed? As the next general advantage rule, we turn our attention to the edges. After the corners (which has protection from flipping in all directions), the edges are second in value to take, since the can only be flipped from two separate directions. Also, they provide a great basis for future moves of flipping discs towards the interior of the board. Therefore, they should be given a higher score than the interior of the field (the center 16 squares). For fields affected both by the corner rule and the edge rule (lying on the edge between those two areas), the scores are higher than for fields just belonging to one rule.

Finally, the interior of the board can be seen as a miniature board of itself. The argument supporting this declaration is given to the reader in the following section, on our Several Stages principle. Of course, being nowhere close to the value of having the "real" corners and edges, this interior "board" has lower scores relative to the more permanently advantageous positions of the field.

Our final board weights are presented in Figure 7. The exact values for each square are a combination of the above reasoning, and evaluation of the extensive testing we have done.