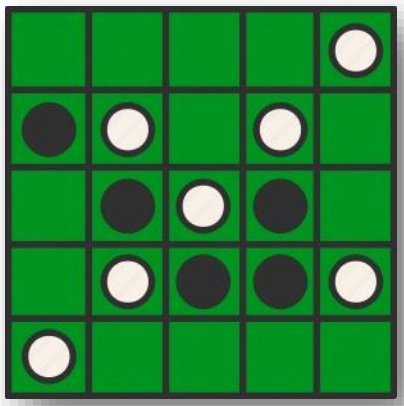
Lebanese University
Faculty of science I

OTHELLO - AI

PROJECT INFO 407 ARTIFICIAL INTELLIGENCE

July 8, 2020



SUPEVISED BY:

DR KIFAH TOUT

DR RAMI BAIDA

PREPARED BY:

FATIMA EZZEDDINE - 87179

FATIMA LAKKIS – 92209

MOHAMMAD KOUTA – 84345

TABLE OF CONTENTS

Table of Contents

SPECIAL THANKS

Special Thanks

A special thought with regard to Doctor Kifah TOUT and Doctor Rami BAIDA at the Lebanese University who accompanied us during the work of this project, providing us with ideas, advice and remarks.

Thank you for enriching our knowledge and for guiding us throughout this semester.

Thank you for showing us the keys to success, having confidence in yourself and your abilities, believing in yourself and always striving to surpass yourself.

SUMMARY

Summary

Othello or **Reversi** is a strategy board **game** played between 2 players on 8x8 uncheckered board. It was invented in 1883. One player plays black and the other white. ... Then the **game** alternates between white and black until: one player cannot make a valid move to outflank the opponent.

In our application, the opponent behavior is based on Alpha-Beta Pruning algorithm with three modes: beginners (no AI), intermediate, and expert.

In the following we present the details of the algorithm, concentrating on the evaluation function used, and our final result…

Algorithmic View

IDEA:

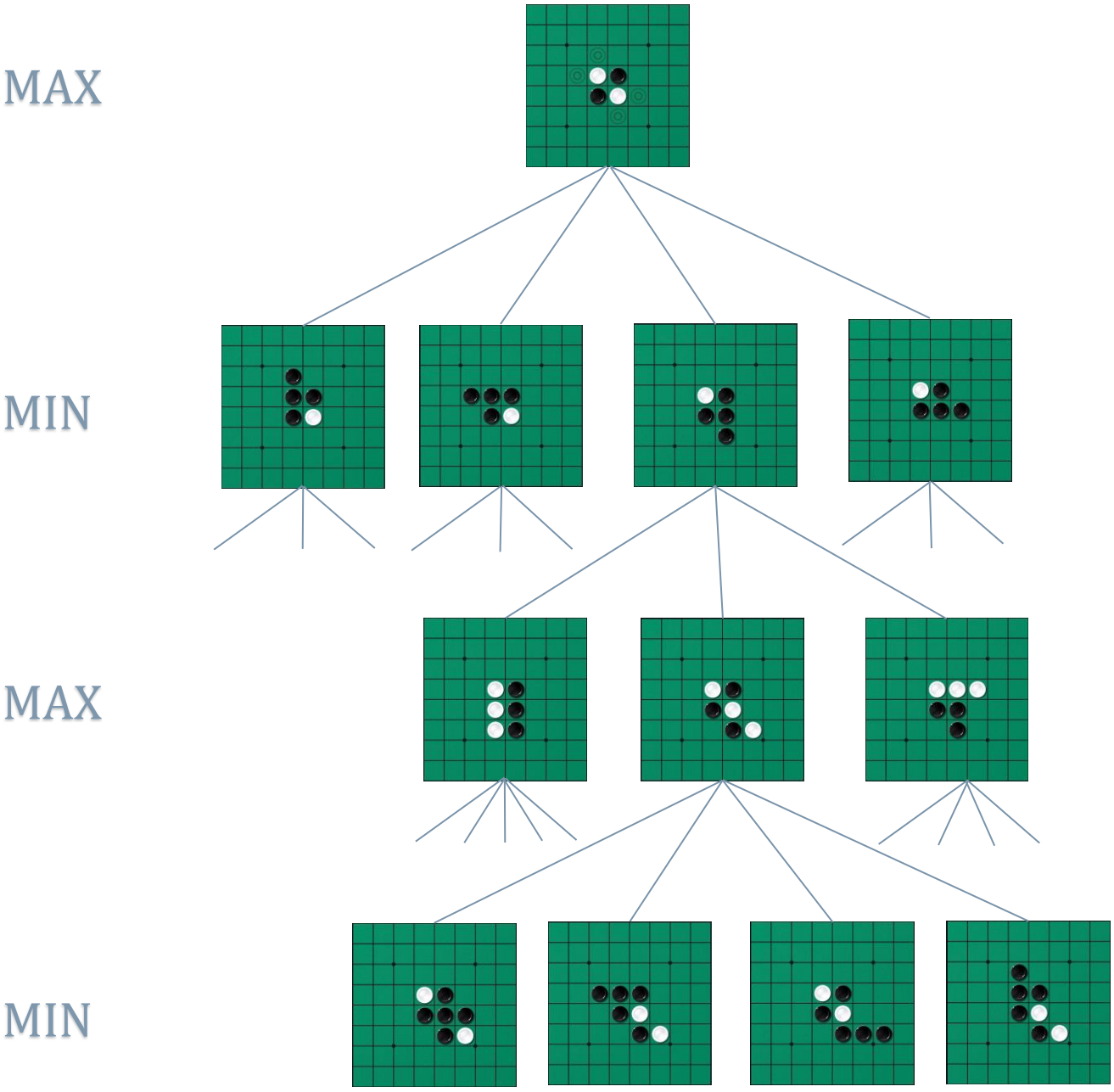
As almost all game playing programs, our Othello player uses a minimax search with alpha-beta pruning. It also uses some move ordering in order to help pruning more quickly.

It stops evaluating a move when at least one possibility has been found that proves the move to be worse than a previously examined move. Such moves need not be evaluated further. When applied to a standard minimax tree, it returns the same move as minimax would, but prunes away branches that cannot possibly influence the final decision.

To illustrate this with a real-life example, suppose somebody is playing chess, and it is their turn. Move "A" will improve the player's position. The player continues to look for moves to make sure a better one hasn't been missed. Move "B" is also a good move, but the player then realizes that it will allow the opponent to force checkmate in two moves. Thus, other outcomes from playing move B no longer need to be considered since the opponent can force a win. The maximum score that the opponent could force after move "B" is negative infinity: a loss for the player. This is less than the minimum position that was previously found; move "A" does not result in a forced loss in two moves.

```
function alphabeta(node, depth, α, β, maximizingPlayer) is
    if depth = 0 or node is a terminal node then
        return the heuristic value of node
    if maximizingPlayer then
        value := −∞
        for each child of node do
            value := max(value, alphabeta(child, depth − 1, α, β, FALSE))
            α := max(α, value)
            if α ≥ β then
                break (* β cut-off *)
        return value
    else
        value := +∞
        for each child of node do
            value := min(value, alphabeta(child, depth − 1, α, β, TRUE))
            β := min(β, value)
            if β ≤ α then
                break (* α cut-off *)
        return value
```

ALGORITHMIC VIEW



ALGORITHMIC VIEW

EVALUATION FUNCTION:

```
public float Evaluate(String[,] board, String color)
{
    if (game.TerminalState(board) == true)
    {
        return 100000 * utility(board, color);
    }

    if (game.DiscsOnBoard(board) <= 20)
    {
        // Opening game
        return 5 * mobility(board, color)
            + 20 * squareWeights(board, color)
            + 10000 * Corners(board, color)
            + 10000 * stability(board, color);
    }
    else if (game.DiscsOnBoard(board) <= 58)
    {
        // Midgame
        return 10 * DiscDifference(board, color)
            + 2 * mobility(board, color)
            + 10 * squareWeights(board, color)
            + 100 * parity(board)
            + 10000 * Corners(board, color)
            + 10000 * stability(board, color);
    }
    else
    {
        // Endgame
        return 500 * DiscDifference(board, color)
            + 500 * parity(board)
            + 10000 * Corners(board, color)
            + 10000 * stability(board, color);
    }
}
```

Our implemented evaluation functions change during three different stages of the game: Beginning, mid-game and end game.

Our heuristic function returns an integer specifying the attractiveness of a potential move.

The greater the number, the more attractive a given move is. The number returned is evaluated through three components: mobility, square weights, corners, stability, disc difference, parity.

✓ Disc Difference:

Since the goal of the game is to have more discs than the opponent, one would think that a good evaluation function would simply compute the two players disc balance at each search node.

In practice, we found that maximizing one's disc count is a very poor strategy and that we could easily beat a program using such an evaluation function.

For this reason, we give very little weight to a player's number of discs.

✓ Mobility or Legal Moves Count:

```
public float mobility(String[,] board, String color)
{
    List<Move> black_moves = findlegalMovesCount(board, "B");
    int bm = black_moves.Count;
    List<Move> white_moves = findlegalMovesCount(board, "W");
    int wm = white_moves.Count;

    if (color == "B")
    {
        return 100 * (bm - wm) / (bm + wm + 1);
    }
    else
    {
        return 100 * (wm - bm) / (bm + wm + 1);
    }
}
```

An interesting tactic to employ is to restrict your opponent's mobility and to mobilize yourself. This ensures that the number of potential moves that your opponent has would drastically decrease, and your opponent would not get the opportunity to place coins that might allow him/her to gain control.

Mobilizing yourself would imply a vast number of moves to choose from, hence indicating that you can exercise power and control the proceeding of the game.

ALGORITHMIC VIEW

✓ Corner Squares:

Corners are the four squares a1, a8, h1, and h8. The specialty of these squares is that once captured, they cannot be flanked by the opponent. They also allow a player to build coins around them and provide stability to the player's coins in the environment. Capturing these corners would ensure stability in the region, and stability is what determines the final outcome to quite a large extent.

There is a high correlation between the number of corners captured by a player and the player winning the game. Of course, it is not true that capturing a majority of the corners would lead to victory, since that clearly need not hold. But capturing a majority of the corners, allows for greater stability to be built.

```
public float Corners(String[,] board, String color)
{
    int blackCorners = 0;
    int whiteCorners = 0;

    if (board[0, 0] == "W")
        whiteCorners++;
    else if (board[0, 0] == "B")
        blackCorners++;

    if (board[0, 7] == "W")
        whiteCorners++;
    else if (board[0, 7] == "B")
        blackCorners++;

    if (board[7, 0] == "W")
        whiteCorners++;
    else if (board[7, 0] == "B")
        blackCorners++;

    if (board[7, 7] == "W")
        whiteCorners++;
    else if (board[7, 7] == "B")
        blackCorners++;

    if (color == "B")
    {
        return 100 * (blackCorners - whiteCorners)
            / (blackCorners + whiteCorners + 1);
    }
    else
    {
        return 100 * (whiteCorners - blackCorners)
            / (blackCorners + whiteCorners + 1);
    }
}
```

✓ Stability:

Stability of coins is a key factor in Othello. The stability measure of a coin is a quantitative representation of how vulnerable it is to being flanked.

```
public int stability(String[,] board,String color)
{
    stableDiscs.Clear();

    stableDiscsFromCorner(board, 0, color);
    stableDiscsFromCorner(board, 7, color);
    stableDiscsFromCorner(board, 56, color);
    stableDiscsFromCorner(board, 63, color);

    int myStables = stableDiscs.Count;
    if (color == "W")
        color = "B";

    else color = "W";
    stableDiscsFromCorner(board, 0, color);
    stableDiscsFromCorner(board, 7, color);
    stableDiscsFromCorner(board, 56, color);
    stableDiscsFromCorner(board, 63, color);

    int opponentStables = stableDiscs.Count;

    return myStables - opponentStables;
}
```

✓ Parity:

Measures who is expected to make the last move/ply of the game. Has zero weight in the opening, but increases to a very large weight in the midgame and endgame.

✓ Square Weights:

Assigns weights to squares so as to avoid giving the opponent a corner. Has moderate weight in the opening and midgame, but has no weight in the endgame.

The static board implicitly captures the importance of each square on the board, and encourages the game play to tend towards capturing corners. Dynamically changing these weights would mean that we would have to use heuristics to calculate the weight of a position based on its stability, offer of mobility and etc.

```
public float squareWeights(String[,] board,String color)
{
    int[] weights={
     200, -100, 100,  50,  50, 100, -100,  200,
    -100, -200, -50, -50, -50, -50, -200, -100,
     100,  -50, 100,   0,   0, 100,  -50,  100,
      50,  -50,   0,   0,   0,   0,  -50,   50,
      50,  -50,   0,   0,   0,   0,  -50,   50,
     100,  -50, 100,   0,   0, 100,  -50,  100,
    -100, -200, -50, -50, -50, -50, -200, -100,
     200, -100, 100,  50,  50, 100, -100,  200,
};
    if (board[0,0] != "E")
    {
        weights[1] = 0;
        weights[2] = 0;
        weights[3] = 0;
        weights[8] = 0;
        weights[9] = 0;
        weights[10] = 0;
        weights[11] = 0;
        weights[16] = 0;
        weights[17] = 0;
        weights[18] = 0;
        weights[24] = 0;
        weights[25] = 0;
    }
```

ALGORITHMIC VIEW

GAME STAGES:

The game is divided to three parts:

1. **Opening (less than 20 discs):**

At the beginning the disc difference and the parity aren't relevant because one move can flip the difference. Yet, the mobility, like square weights, is relatively relevant, to ensure more effective slots, which could give a huge advantage in the rest of the game.

2. **Midgame (between 20 and 58 discs):**

A positional strategy stresses the importance of specific positions and piece configurations on the board. Places such as corners and edges are considered valuable, while others are avoided. Corners are especially valuable because once taken, they can never be recaptured.

Like at the opening of the game, mobility, and square weights still important. In addition, disc difference and parity become more relevant, as the final result started to appear...

3. **Endgame (59 discs or more):**

As the game is ending, and since the goal of the game is to have more discs than the opponent, the most important now is to increase the disc difference and the parity. Unlike the mobility, and the square weights witch became irrelevant at all.

However, at **all stages**, a positional strategy stresses the importance of specific positions and piece configurations on the board. Places such as corners and edges are considered valuable, while others are avoided. Corners are especially valuable because once taken, they can never be recaptured.

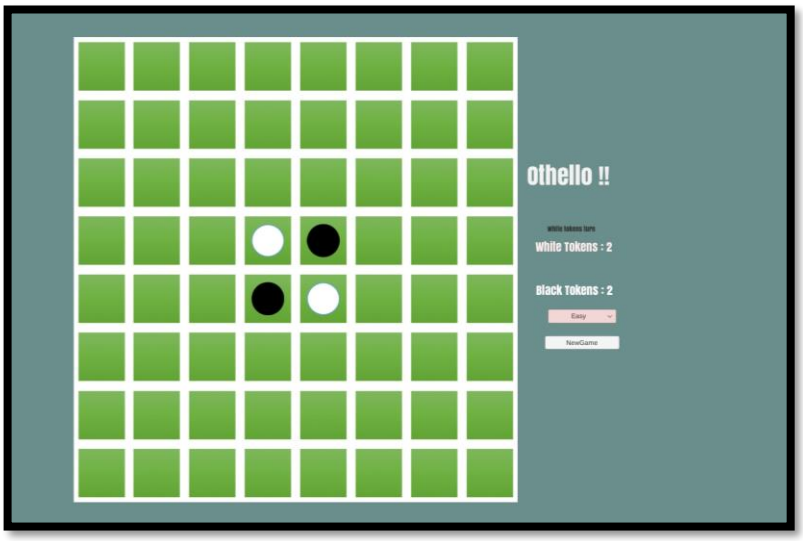
The most natural strategy that many primitive computer Othello players employed was to base their move on a greedy strategy that tried to maximize the number of coins of a player at any point. Such strategies failed miserably, and obviously so. A single move can flank at most 18 coins, which implies that games can swing from the control of one player to another very rapidly. Since a complete exploration of the game tree would not be possible till the very end stages of the game, such a strategy does not incorporate the drastically dynamic nature of the game. Neither does it account for the instability of coins.

Hence, corners and stability are always very important and valuable.

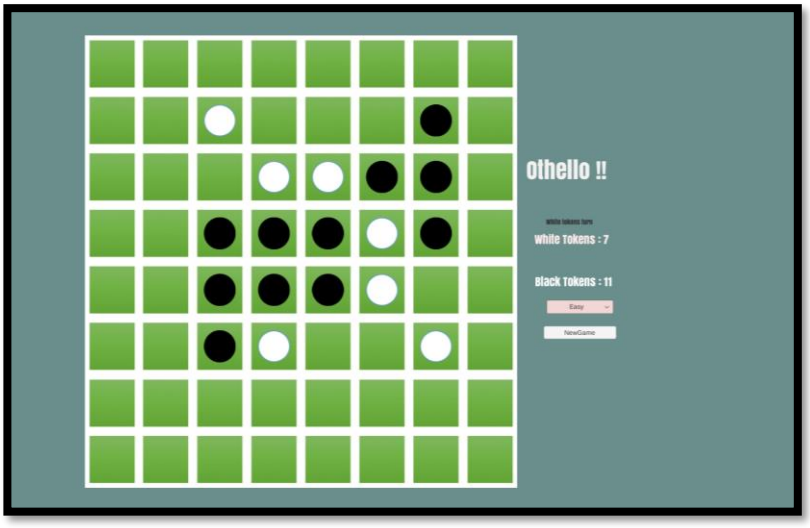
Result

Here is the screen shots of our app exported as android app, link to install:
https://drive.google.com/drive/folders/1Cjvk4dO_6ThAypXGmRDRxRtKC9FE292c?usp=sharing

OPPENING SCENE:



MID GAME:



CONCLUSION

Conclusion

Game playing has always been one of the most attractive fields of artificial intelligence research, and it will continue to be so. It is one of the parts of artificial intelligence that the common man observes and interacts with.

We evaluated the importance of adding heuristics to enhance Othello game play. However, this application can be upgraded by using Evolutionary Neural Network, Negascout, MTD-f…
Several heuristics are also used to reduce the size of the searched tree: good move ordering, transposition table and selective Search.
To speed up the search on machines with multiple processors or cores, a "parallel search" may be implemented.

REFERENCES

References

- ✓ https://www.ultraboardgames.com/othello/tips.php
- ✓ https://www.wikihow.com/Play-Othello
- ✓ https://www.mastersofgames.com/rules/reversi-othello-rules.htm
- ✓ http://en.othellobelgium.be/leer-othello/tips-en-strategie
- ✓ https://courses.cs.washington.edu/courses/cse573/04au/Project/mini1/RUSSIA/Final_Paper.pdf