Contents:

Introduction:

In this project we will learn how to solve Othello game problem by using two different game search algorithm witch is minmax and alpha beta prunning algorithms, the program will ask the user to select an agent between 6 agents for two players and their names then press start bottom to begin, the program will process the game according to the agents chooses at the end of the game the program will shows the result of the two agents that have been chosen witch is the score of each agents and the time spent by each of them and the winner who has the high score.

Problem Formulation:

State Representation: 2D array of the Othello game, each index in the array is the coordinate of the board, which is: board[row][col].

Initial State: The first turn of the game the agent will play to the board.

Goal State: The last turn of the game the agent will play to the board.

Goal Test: Is the current state equal to the goal state?

Actions: The agent move in 8 directions (UP, DOWN, LEFT, RIGHT, UPLEFT, UPRIGHT, DOWNLEFT, DOWNRIGHT).

Path Cost: The number of times played at a turn in the game from the first turn to the last one.

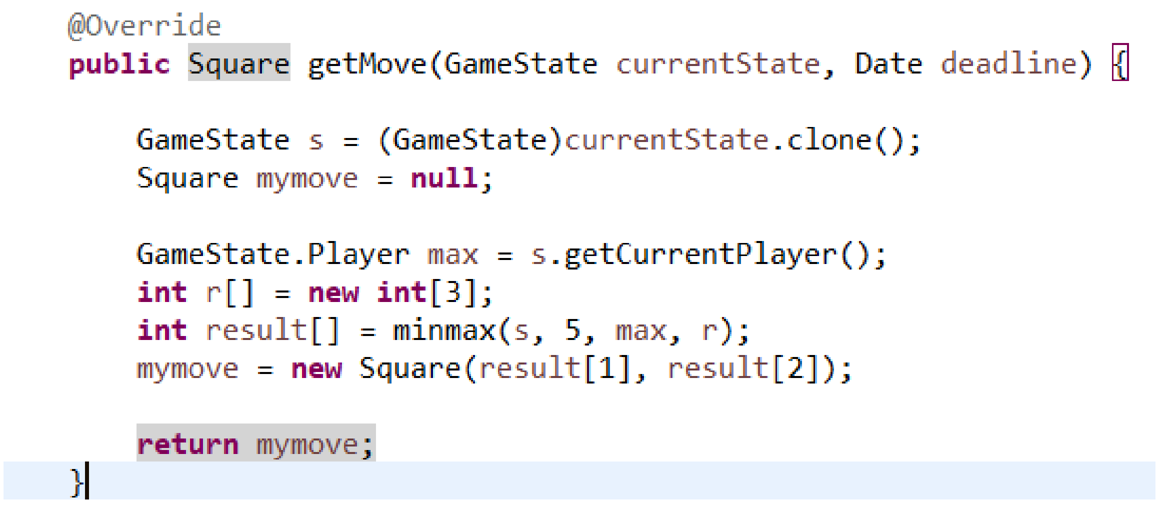
Implementation:

The program is implemented using java language and other library that have been added to the program, there is 16 classes that has already implemented and there is 2 classes that we implemented in the project which is Alpha_beta_prunning_NoHF class and there is Minmax.

1) Minmax class:

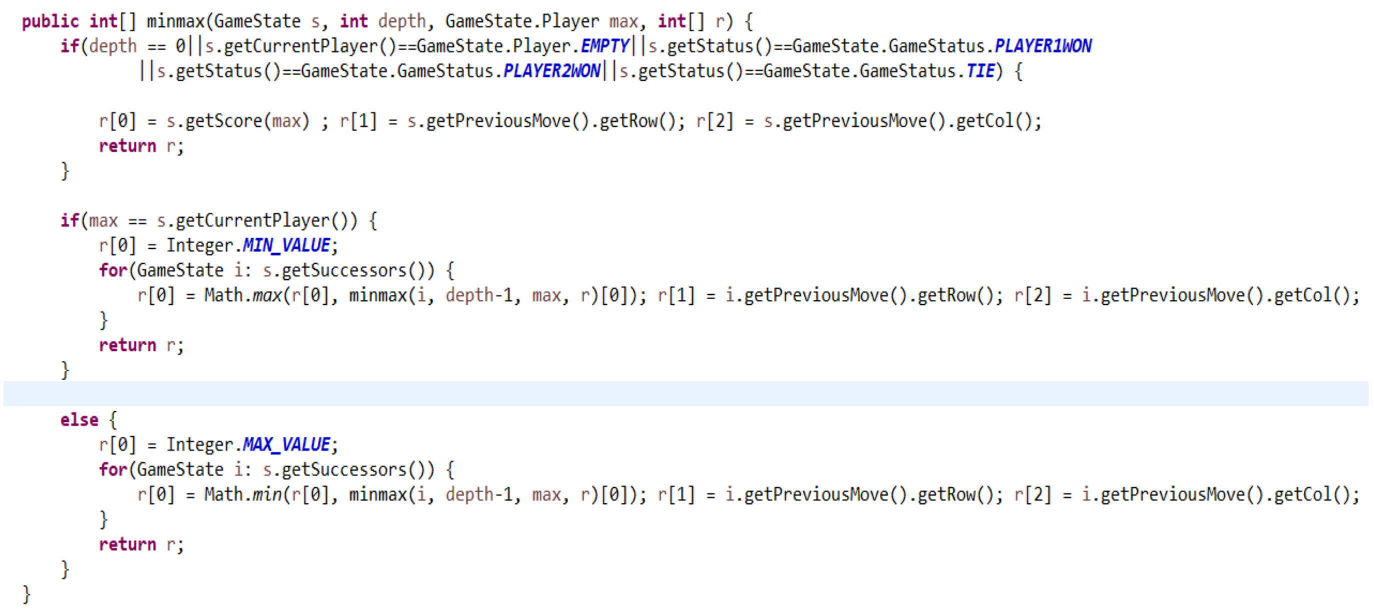
It's the first class we implement and this class is extend the parent abstract class **OthelloPlayer** along with **HumanOthelloPlayer , Alpha_beta_prunning_NoHF,HumanOthelloPlayer, AOthelloPlayer** and **RandomOthelloPlayer** which they all the agents that we want to use in this program, but in this part we will focus on the Minmax class and what's the function that have been implemented.

The function that we implement in this class:



```
@Override
public Square getMove(GameState currentState, Date deadline) {

    GameState s = (GameState)currentState.clone();
    Square mymove = null;

    GameState.Player max = s.getCurrentPlayer();
    int r[] = new int[3];
    int result[] = minmax(s, 5, max, r);
    mymove = new Square(result[1], result[2]);

    return mymove;
}
```

This function will return the object **mymove** of the class **Square**, which the move the player agent will do in the board, but first in this function will create an object of type **GameState** as a clone of the **currentState** object then will create another object of type **Square**, then will set the max as the current player, the array **r** will help us in the **minmax()** function as we will see later, the array **result** will store the result of the recursive function **minmax()** it will have 3 indices we will use the first one as the row and the second as the column of the square then these two indices of the array will store it in the **mymove** object as argument of the class **Square**, and **mymove** object will help the **agent of Minamax** to choose his move.



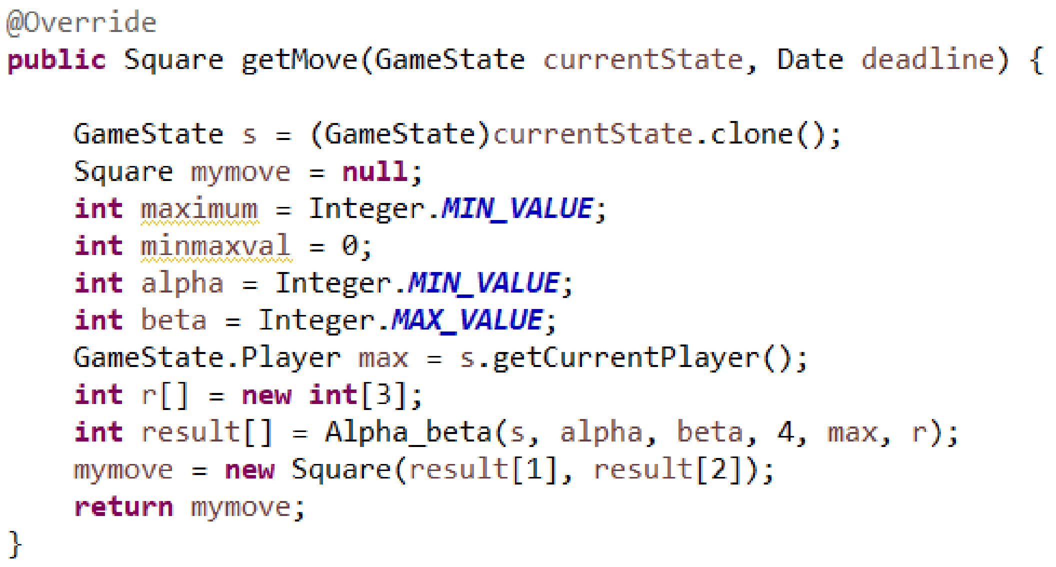
```java
public int[] minmax(GameState s, int depth, GameState.Player max, int[] r) {
    if(depth == 0||s.getCurrentPlayer()==GameState.Player.EMPTY||s.getStatus()==GameState.GameStatus.PLAYER1WON
            ||s.getStatus()==GameState.GameStatus.PLAYER2WON||s.getStatus()==GameState.GameStatus.TIE) {

        r[0] = s.getScore(max) ; r[1] = s.getPreviousMove().getRow(); r[2] = s.getPreviousMove().getCol();
        return r;
    }

    if(max == s.getCurrentPlayer()) {
        r[0] = Integer.MIN_VALUE;
        for(GameState i: s.getSuccessors()) {
            r[0] = Math.max(r[0], minmax(i, depth-1, max, r)[0]); r[1] = i.getPreviousMove().getRow(); r[2] = i.getPreviousMove().getCol();
        }
        return r;
    }

    else {
        r[0] = Integer.MAX_VALUE;
        for(GameState i: s.getSuccessors()) {
            r[0] = Math.min(r[0], minmax(i, depth-1, max, r)[0]); r[1] = i.getPreviousMove().getRow(); r[2] = i.getPreviousMove().getCol();
        }
        return r;
    }
}
```

This recursive function will called by **getMove()** function, it will return array of type int, the first part of the function is the if statement which is the base case of this recursive function it will return the array **r** that received when the depth equal to zero or you reach the terminal state which mean there is one player that win in the game or if there is no more moves which mean there is empty squares in the board.

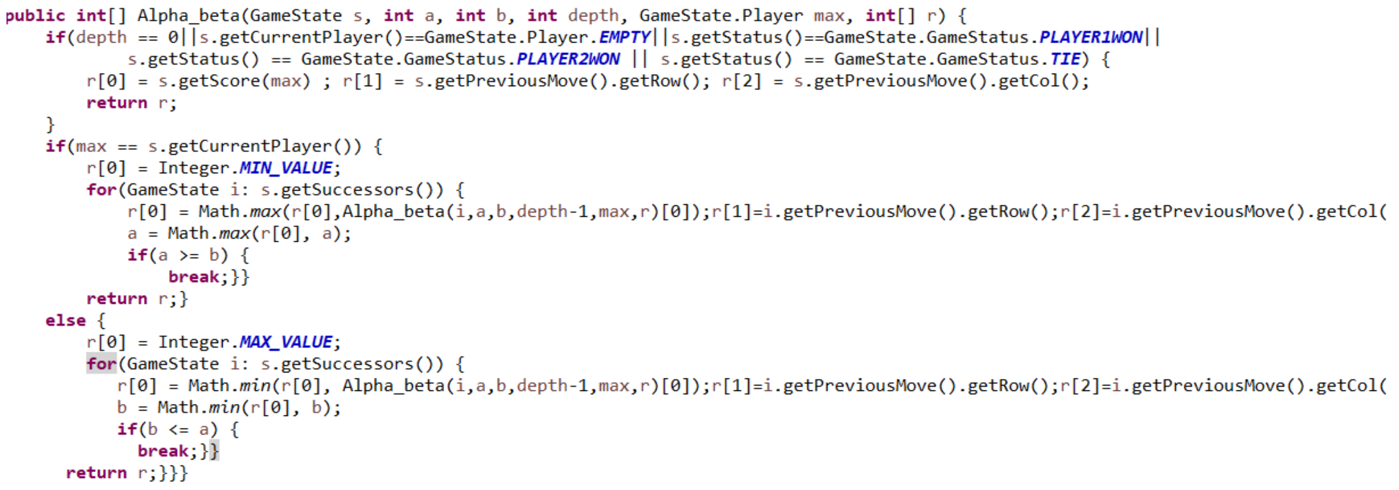
The last part which is the second if and else statements, the if statement will accept if the **max** object which is received from the **getMove()** function which refer to the current player then it will maximize the values inside the if statement, or the else part then it will minimize the values as the idea of the **MINMAX algorithm.**

2) Alpha_beta_prunning_NoHF class:

It's the second function that we implement and this class extends the parent abstract class **OthelloPlayer** same as **Minmax class**, but this class will make better agent than **Minmax** with respect to the speed of the agent but the differences between the two classes are very few as we will see next.



```java
@Override
public Square getMove(GameState currentState, Date deadline) {

    GameState s = (GameState)currentState.clone();
    Square mymove = null;
    int maximum = Integer.MIN_VALUE;
    int minmaxval = 0;
    int alpha = Integer.MIN_VALUE;
    int beta = Integer.MAX_VALUE;
    GameState.Player max = s.getCurrentPlayer();
    int r[] = new int[3];
    int result[] = Alpha_beta(s, alpha, beta, 4, max, r);
    mymove = new Square(result[1], result[2]);
    return mymove;
}
```

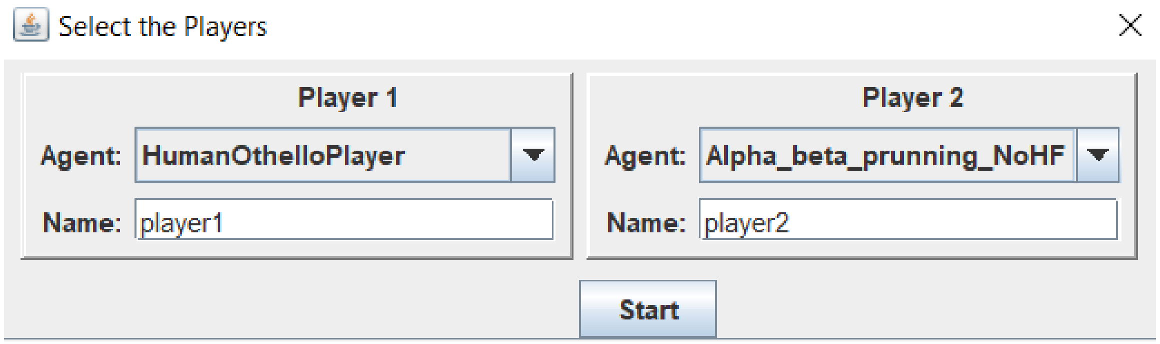


```java
public int[] Alpha_beta(GameState s, int a, int b, int depth, GameState.Player max, int[] r) {
    if(depth == 0||s.getCurrentPlayer()==GameState.Player.EMPTY||s.getStatus()==GameState.GameStatus.PLAYER1WON||
            s.getStatus() == GameState.GameStatus.PLAYER2WON || s.getStatus() == GameState.GameStatus.TIE) {
        r[0] = s.getScore(max) ; r[1] = s.getPreviousMove().getRow(); r[2] = s.getPreviousMove().getCol();
        return r;
    }
    if(max == s.getCurrentPlayer()) {
        r[0] = Integer.MIN_VALUE;
        for(GameState i: s.getSuccessors()) {
            r[0] = Math.max(r[0],Alpha_beta(i,a,b,depth-1,max,r)[0]);r[1]=i.getPreviousMove().getRow();r[2]=i.getPreviousMove().getCol(
            a = Math.max(r[0], a);
            if(a >= b) {
                break;}}
        return r;}
    else {
        r[0] = Integer.MAX_VALUE;
        for(GameState i: s.getSuccessors()) {
           r[0] = Math.min(r[0], Alpha_beta(i,a,b,depth-1,max,r)[0]);r[1]=i.getPreviousMove().getRow();r[2]=i.getPreviousMove().getCol(
           b = Math.min(r[0], b);
           if(b <= a) {
             break;}}
      return r;}}}
```

These functions is almost the same as the two functions in the **Minmax class** but there is two more values which is the **alpha** value is set to the highest value which is close to minus infinity and the second value is **beta** is set to the lowest value which is close to plus infinity and these two will used in the recursive function **Alpha_beta()** as the second image.

Result:

First sample run input:

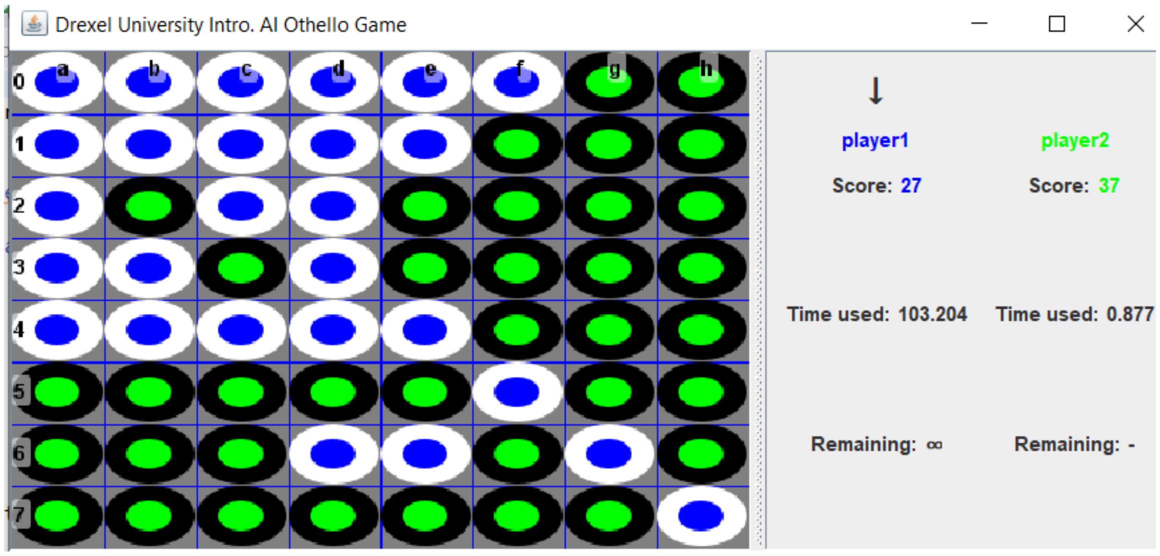


Select the Players

Player 1
Agent: HumanOthelloPlayer
Name: player1

Player 2
Agent: Alpha_beta_prunning_NoHF
Name: player2

First sample run output:



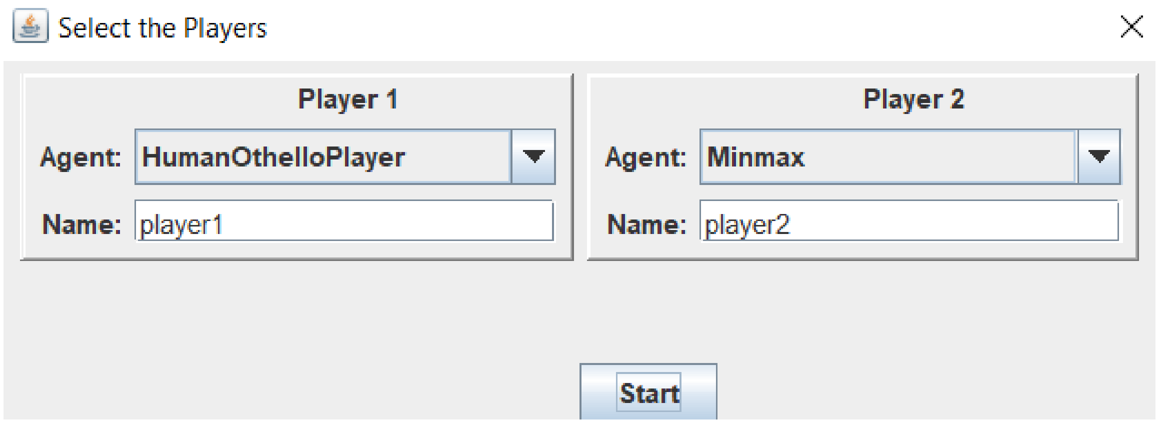
Drexel University Intro. AI Othello Game

player1
Score: 27
Time used: 103.204
Remaining: ∞

player2
Score: 37
Time used: 0.877
Remaining: -

We see that we played against the alpha-beta agent and out of 10 game we won only one game which seems to be smart agent and what we see in the time used that is this agent is so fast.

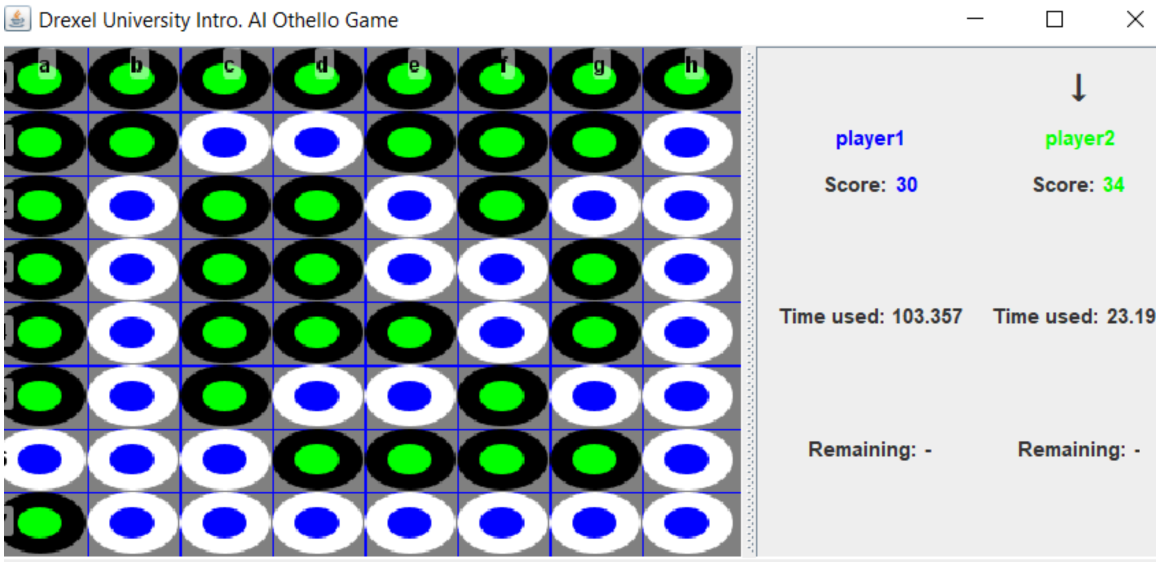
Second sample run input:



Player 1
Agent: HumanOthelloPlayer
Name: player1

Player 2
Agent: Minmax
Name: player2

Second sample run output:



player1
Score: 30
Time used: 103.357
Remaining: -

player2
Score: 34
Time used: 23.19
Remaining: -

We see that we played against the alpha-beta agent and out of 10 game we won 2 times it almost same as the alpha-beta agent but it much slower than alpha-beta agent.

Conclusion:

We created the problem formulation for this Othello game problem so we can get the idea of the structure of the problem like State structure, initial state, final state…etc

After that we started the implementation of the two classes that we created and programed, we use the structure of the Minmax and alpha-beta to implement these two classes and used them as an agent to solve the game problem, After that we analyzed the result by testing the agents that we create depending in three factors which is the speed of the algorithms, the score for each agent and finally by number of winning game for the agent out of 10 games, and we found the desired result for solving the Othello game problem.