**Artificial Intelligence CSCI 264**

Project Report

Implementation of

**Reversi - A Game of Brains**

-By

Yogesh Nageswara Rao(109551409)

Roshan Shivanna(109544467)

**Contents**

1. **Abstract**
2. **Introduction**
3. **About reversi**
4. **Implementation**

Abstract

Reversi is a two player strategy board game established in England in the 1880s. It is also known as Othello. The motivation to implement this game as an AI project comes from adversial search strategies, applying a minimax algorithm with an alpha-beta pruning not taking branches into account that cannot possibly influence the final decision. By adding various heuristic evaluation functions, the search problem can be solved in a more effective way, thus yielding better moves compared to a simple greedy method.

Introduction

**Artificial intelligence** (**AI**) is the [intelligence](https://en.wikipedia.org/wiki/Intelligence) exhibited by [machines](https://en.wikipedia.org/wiki/Machine) or [software](https://en.wikipedia.org/wiki/Software) which is incorporated by humans. It is also the name of the academic [field of study](https://en.wikipedia.org/wiki/Field_of_study) which studies how to create [computers](https://en.wikipedia.org/wiki/Computer) and computer software that are capable of intelligent behavior. Major AI researchers and textbooks define this field as "the study and design of intelligent agents", in which an [intelligent agent](https://en.wikipedia.org/wiki/Intelligent_agent) is a system that perceives its environment and takes actions that maximize its chances of success. [John McCarthy](https://en.wikipedia.org/wiki/John_McCarthy_(computer_scientist)), who coined the term in 1955, defines it as "the science and engineering of making intelligent machines".[[4]](https://en.wikipedia.org/wiki/Artificial_intelligence#cite_note-McCarthy.27s_definition_of_AI-4)

AI research is highly technical and specialized, and is deeply divided into subfields that often fail to communicate with each other. Some of the division is due to social and cultural factors: subfields have grown up around particular institutions and the work of individual researchers. AI research is also divided by several technical issues. Some subfields focus on the solution of specific [problems](https://en.wikipedia.org/wiki/Artificial_intelligence#Goals). Others focus on one of several possible [approaches](https://en.wikipedia.org/wiki/Artificial_intelligence#Approaches) or on the use of a particular [tool](https://en.wikipedia.org/wiki/Artificial_intelligence#Tools) or towards the accomplishment of particular [applications](https://en.wikipedia.org/wiki/Artificial_intelligence#Applications).

The central problems (or goals) of AI research include  [reasoning](https://en.wikipedia.org/wiki/Reasoning),  [knowledge](https://en.wikipedia.org/wiki/Knowledge),  [planning](https://en.wikipedia.org/wiki/Automated_planning_and_scheduling) ,  [learning](https://en.wikipedia.org/wiki/Learning), [natural language processing](https://en.wikipedia.org/wiki/Natural_language_processing), [perception](https://en.wikipedia.org/wiki/Perception) and the ability to move and manipulate objects by machines. [General intelligence](https://en.wikipedia.org/wiki/Artificial_general_intelligence) is still among the field's long-term goals. Currently popular approaches include [statistical methods](https://en.wikipedia.org/wiki/Artificial_intelligence#Statistical), [computational intelligence](https://en.wikipedia.org/wiki/Artificial_intelligence#Sub-symbolic) and [traditional symbolic AI](https://en.wikipedia.org/wiki/Artificial_intelligence#Symbolic). There are a large number of tools used in AI, including versions of [search and mathematical optimization](https://en.wikipedia.org/wiki/Artificial_intelligence#Search_and_optimization), [logic](https://en.wikipedia.org/wiki/Artificial_intelligence#Logic), [methods based on probability and economics](https://en.wikipedia.org/wiki/Artificial_intelligence#Probabilistic_methods_for_uncertain_reasoning), and many others. The AI field is interdisciplinary, in which a number of sciences and professions converge, including [computer science](https://en.wikipedia.org/wiki/Computer_science), [mathematics](https://en.wikipedia.org/wiki/Mathematics), [psychology](https://en.wikipedia.org/wiki/Psychology), [linguistics](https://en.wikipedia.org/wiki/Linguistics), [philosophy](https://en.wikipedia.org/wiki/Philosophy) and [neuroscience](https://en.wikipedia.org/wiki/Neuroscience), as well as other specialized fields such as [artificial psychology](https://en.wikipedia.org/wiki/Artificial_psychology).

Much work has been done in the past on designing Artificial Intelligence (AI) programs to play “classic” board games, such as Chess, Checkers, Othello, and Go. Many of these games have programs that are sufficiently advanced that they beat the best human players. In the last ten to twenty years, however, there has been a rise in “abstract” or “European-style” board games. These differ from the board games many of us have played as children (such as Monopoly or Life) in several areas: (1) the games are typically short, many finishing in 90 minutes or less; (2) the games usually emphasize player interaction in some way (components such as bidding, competing for scarce resources, or trading/negotiation are commonly seen); and (3) the games often are based around hidden information, so that nobody can know the whole state of the game. These factors, especially the last two, make designing an AI for these games a challenge, and so much less has been done analyzing these games, and what has been done has much room for improvement, while other game companies have been developing AI programs for sale, usually as mobile apps).

About Reversi

**Reversi** is a strategic [board game](https://en.wikipedia.org/wiki/Board_game) for two players, played on an 8×8 uncheckered board. There are sixty-four identical game pieces called disks (often spelled "discs"), which are light on one side and dark on the other. Players take turns placing disks on the board with their assigned color facing up. During a play, any disks of the opponent's color that are in a straight line and bounded by the disk just placed and another disk of the current player's color are turned over to the current player's color.

The object of the game is to have the majority of disks turned to display your color when the last playable empty square is filled.

History

The game Reversi was invented in 1883 by either of two English men, Lewis Waterman[[1]](https://en.wikipedia.org/wiki/Reversi#cite_note-1) or John W. Mollett), and gained considerable popularity in England at the end of the nineteenth century. The game's first reliable mention is in the August twenty-first 1886 edition of [The Saturday Review](https://en.wikipedia.org/wiki/Saturday_Review_(London)). Later mention includes an 1895 article in [The New York Times](https://en.wikipedia.org/wiki/The_New_York_Times): "Reversi is something like [Go Bang](https://en.wikipedia.org/wiki/Gomoku), and is played with 64 pieces." In 1893, the well-known German games publisher [Ravensburger](https://en.wikipedia.org/wiki/Ravensburger) started producing the game as one of its first titles. Two 18th-century continental European books dealing with a game that may or may not be Reversi are mentioned on page fourteen of the Spring 1989 [Othello Quarterly](https://en.wikipedia.org/wiki/Othello_Quarterly), and there has been speculation, so far without documentation, that the game has even more ancient origins.

Modern version

The modern version of the game — the most regularly used rule-set, and the one used in international tournaments — is marketed and recognized as Othello, which was perfected[[4]](https://en.wikipedia.org/wiki/Reversi#cite_note-4) by Goro Hasegawa (autonym: Satoshi Hasegawa) originated in Japan in the 1970s. There are two differences from the original game:

* The first four pieces go in the center, but in a standard diagonal pattern, rather than being placed by players
* Each player has (essentially) up to 64 pieces, rather than being limited to 32 pieces each.

Hasegawa established the Japan Othello Association on March 1973, and held the first national Othello championship on April 4, 1973 in Japan.[[5]](https://en.wikipedia.org/wiki/Reversi#cite_note-5) The Japanese game company Tsukuda Original launched Othello in late April, 1973 in Japan under Hasegawa’s license, which led to an immediate commercial success.

Rules

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | | |  |  | | --- | --- | | · | Reversi or Othello takes place between two players, black and white, on an 8x8 board of 64 squares. There are 64 discs colored black on one side and white on the other. | | · | The board is set up initially with two black discs (i.e. a disc with black side uppermost) placed on squares e4 and d5 and two white discs on d4 and e5. | | · | Black always plays first with players then taking alternate turns. | | · | At each turn a player must place a disc with their color face up on one of the empty squares of the board, adjacent to an opponent's disc such that one or more straight lines (horizontal, vertical or diagonal) are formed from the newly placed disc, through one or more of the opponent's discs and up to other discs of their own color already on the board. All the intervening discs of the opponent's color are flipped to the color of the newly laid disc. | | · | Discs may be flipped from one color to the other but once played are not moved from one square to another or removed from the board. | | · | Players may not pass unless there is no valid move available to them in which case they must pass. | | · | Play continues until neither player is able to move, usually when all 64 squares have been played. | |  |  | | --- | |  | |

|  |
| --- |
|  |

Goal of the game

* The winner is the player with most pieces turned to their colour at the close of play. By convention any empty squares at the end are added to the winners score.
* If both players have the same number of discs at the end then the game is tied.

Introduction to Strategy

**Openings**: For relatively inexperienced players the opening (early) part of a game of Reversi is generally not very easy to make sense of. A rule of thumb on the opening is that a good opening is one that leads to a good middle-game. By and large, good moves in the very earliest stages are determined by whether there is a refutation to a move only and few other truly general considerations aside from what exactly constitutes a refutation. The first move (by dark) is no choice at all other than for the purpose of the player's possible sense of ideal visualization.

**Corners**: Corner positions, once played, remain immune to flipping for the rest of the game, being termini of horizontal, vertical and diagonal lines. More generally, a piece is stable when, along all four axes (horizontal, vertical, and each diagonal), it is in terminal position or if from it along the axis one reaches a terminal disk passing only through disks the same color. These are not the only kinds of stable disk, however, and occupying a corner may often be a grave error if one allows one's opponent to create a wedge that results in him or her gathering more stable disks. This can render occupying the corner largely useless, and often much worse than that because of loss of tempo.

### Mobility: An opponent playing with reasonable strategy will not so easily relinquish the corner or any other good moves. So to achieve these good moves, a player must force its opponent to play moves that relinquish those good moves. One of the ways to achieve this involves reducing the number of moves available to the player's opponent. Ideally, this will eventually force the opponent to make an undesirable move.

### Edges: Edge pieces can anchor flips that influence moves to all regions of the board. If played poorly, this can poison later moves by causing players to flip too many pieces and open up many moves for the opponent. However, playing on edges where an opponent can not easily respond drastically reduces possible moves for that opponent.

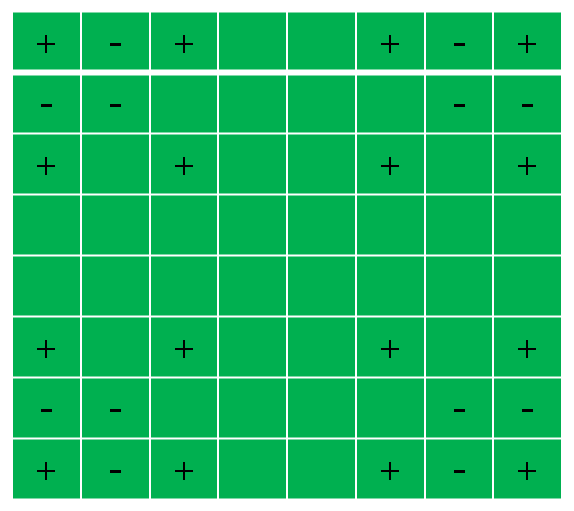
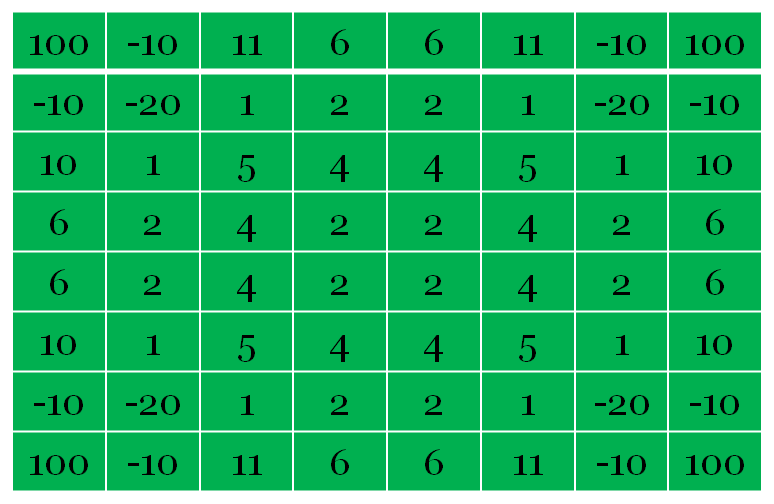
The square immediately diagonally adjacent to the corner (called the X-square), when played in the early or middle game, typically guarantees the loss of that corner. Nevertheless, such a corner sacrifice is sometimes played for some strategic purpose (like retaining mobility). Playing to the edge squares adjacent to the corner (called the C-squares) can also be dangerous if it gives the opponent powerful forcing moves.

### Parity: Parity is one of the most important parts of the strategy. In short, the concept of parity is about getting the last move in every empty region in the end-game, and thereby increasing the number of stable disks. The concept of parity led to a change in the perception of the game, as it led to distinct strategies for playing Black and White. It forced Black to play more aggressive moves and gave White the opportunity to stay calm and focus on keeping the parity. As a result, the opening books and midgame were focused on Black being the "attacker" and White being the "defender". The concept of parity also controls how edge positions are played and how edges interact.

**Maximum Disks Strategy:** Most pieces does not necessarily guarantee a victory.Beginners usually start by interpreting the requirement to end up with the most discs as a strategy for the whole game. It can also be tempting to try to wipe out your opponent earlier in the game by taking as many of their pieces as possible however unless your opponent is carelessly working the [evaporation strategy](http://www.samsoft.org.uk/reversi/strategy.htm#evaporate) and not paying attention then it is unlikely that you will succeed. By the time it becomes apparent that you have failed you are likely to have few good moves left and your opponent will be well placed to drive you to defeat.

**Positional Strategy:** Each square on the board are given certain values for their positions.

Corners are very good. hence planning a move you need to consider your opponents' possible response, then your response to that and so on. At the end of the game it might be possible to calculate all the variations but elsewhere in the game you will need to have some mechanism to avoid having to consider all possible branches more deeply than is practical.



Implementation

In this project, Reversi game has been implemented in Java where the front end has been developed using Java Swings. The color of the disk for the human player is black and that of the computer’s is white.

The execution starts with the main() function( Panel class file) where a window frame with the name “Reversi” is created. And all the necessary functions for the front end has been initialized.

JFrame window = **new** JFrame("Reversi");

window.getContentPane().add(panel);

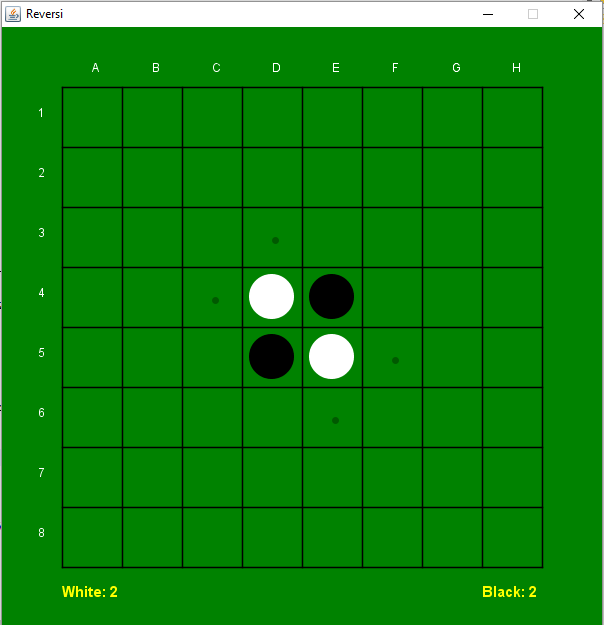
window.setResizable(**false**);

window.pack();

window.setDefaultCloseOperation(JFrame.***EXIT\_ON\_CLOSE***);

window.setVisible(**true**);

When the program is executed the window appears with the initial configuration of 2 black and 2 white pieces.



The drawBoard() function is responsible for displaying the board on the screen:

**private** **void** drawBoard(Graphics g)

{

**int**[] cells = board.getCells();

**int** gameCols = Board.***BOARD\_SIZE***;

**int** hiddenCols = gameCols + 2;

**int** mid = (cellSize - pieceSize)/2;

**for** (**int** i = hiddenCols; i < cells.length - hiddenCols; i++)

{

**int** col = i % hiddenCols;

**int** row = i / hiddenCols;

**if** ((col != 0) && (col != hiddenCols - 1))

{

**int** piece = cells[i];

**if** (piece == Board.***EMPTY***)

**continue**;

Piece p = **new** Piece(cells[i], pieceSize);

p.draw(g, col \* cellSize + mid, row \* cellSize + mid);

}

}

**if** (lastMoveIndex > 0)

{

g.setColor(Color.***yellow***);

**int** col = lastMoveIndex % hiddenCols;

**int** row = lastMoveIndex / hiddenCols;

g.drawRect(col \* cellSize, row \* cellSize, cellSize, cellSize);

}

}

The computer machine will make a move using the function makeMove():

**public** **void** makeMove (Move move)

{

**int** player = move.getPlayer();

ArrayList<Integer> flips = move.getFlipSquares();

**if** (flips != **null**)

{

**int** idx = move.getIdx();

cells[idx] = player;

**for** (Integer flip : flips)

cells[flip] = player;

emptyCells--;

**this**.updatePhase();

}

**this**.toogleCurrentPlayer();

}

In order to make a move against the human player it uses minimax alpha beta pruning to select the most efficient move possible. This makes use of the heuristics functions, In the heuristics class all the strategies are made use of in order to calculate the utility values.

The positional heuristics is given by the following array which is obtained by the reference paper:

**private** **static** **final** **int**[] **squareValues** = **new** **int**[]

{

0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 100, -10, 11, 6, 6, 11, -10, 100, 0,

0, -10, -20, 1, 2, 2, 1, -20, -10, 0,

0, 10, 1, 5, 4, 4, 5, 1, 10, 0,

0, 6, 2, 4, 2, 2, 4, 2, 6, 0,

0, 6, 2, 4, 2, 2, 4, 2, 6, 0,

0, 10, 1, 5, 4, 4, 5, 1, 10, 0,

0, -10, -20, 1, 2, 2, 1, -20, -10, 0,

0, 100, -10, 11, 6, 6, 11, -10, 100, 0,

0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

};

1. The Mobility strategy value is calculated as follows:

**public** **static** **int** mobilityDiff (**int** player, Board board)

{

**return** mobility(player, board) - mobility(board.getOpponent(player), board);

}

2. The Corner position strategy is calculated as follows:

**public** **static** **int** cornerSquares (**int** player, Board board)

{

**int** corners = 0;

**int**[] cells = board.getCells();

**for**(**int** i = 0; i < ***cornerIndexes***.length; i++)

**if** (cells[***cornerIndexes***[i]] == player)

corners++;

**return** corners;

}

3. The mobility strategy is calculated as follows.

**public** **static** **int** mobility (**int** player, Board board)

{

**return** board.getAllMoves(player).size();

}

**public** **static** **int** mobilityDiff (**int** player, Board board)

{

**return** *mobility*(player, board) - *mobility*(board.getOpponent(player), board);

}

**public** **static** **int** potentialMobility(**int** player, Board board)

{

**int** opponent = board.getOpponent(player);

**int**[] cells = board.getCells();

**int** mob = 0;

**for** (**int** i = 10; i < 90; i++)

{

**int** col = i % 10;

**if** (col != 0 && col != 9)

{

**if** (cells[i] == opponent)

{

**for** (Integer dir : Board.***DIRECTIONS***)

{

**int** around = dir + i;

**if** (cells[around] == Board.***EMPTY*** && cells[around] != Board.***WALL***)

{

mob++;

**break**;

}

}

}

}

}

**return** mob;

}

**public** **static** **int** potentialMobilityDiff(**int** player, Board board)

{

**return** *potentialMobility*(player, board) - *potentialMobility*(board.getOpponent(player), board);

}

All the strategies are combined and evaluated to make the best move possible:

**public** **static** **float** eval(**int** player, Board board)

{

**float** value = 0.0f;

**int** phase = board.getPhase();

**if** (phase == Board.**PHASE\_OPENING**)

{

value = 100 \* mobilityDiff(player, board) + 100 \* potentialMobilityDiff(player, board)

+ 800 \* cornerSquaresDiff(player, board)

- 200 \* badXSquaresDiff(player, board) - 200 \* badCSquaresDiff(player, board);

}

**else** **if** (phase == Board.**PHASE\_MIDGAME**)

{

value = 100 \* mobilityDiff(player, board) + 100 \* potentialMobilityDiff(player, board)

+ 900 \* cornerSquaresDiff(player, board)

- 250 \* badXSquaresDiff(player, board) - 200 \* badCSquaresDiff(player, board);

}

**else** **if** (phase == Board.**PHASE\_ENDGAME**)

{

value = discsDiff(player, board);

}

**else**

{

System.**out**.println("Error in calculating the heuristics");

}

**return** value;

}

In order to display the flipping of the disks when a player wins the points the following code snippet is used:

**public** **void** makeMove (Move move)

{

**int** player = move.getPlayer();

ArrayList<Integer> flips = move.getFlipSquares();

**if** (flips != **null**)

{

**int** idx = move.getIdx();

cells[idx] = player;

**for** (Integer flip : flips)

cells[flip] = player;

emptyCells--;

**this**.updatePhase();

}

**this**.toogleCurrentPlayer();

}

The best possible move is created using the algorithm minimax with alphabeta pruning. In order to be efficient and more fast we have made use of multiprocessing element in java processes.

**public** **void** run()

{

**float** val = minimax(board, -INFINITY, INFINITY, 0);

System.**out**.println("calls: " + calls);

System.**out**.println("eval: " + val);

System.**out**.println();

doneSignal.countDown();

}

As we stated the we have used minimax, the code snippet shows the implementation of the minimax algorithm:

**private** **float** minimax(Board board, **float** alpha, **float** beta, **int** depth)

{

calls++;

**int** currentPlayer = board.getCurrentPlayer();

**if** (board.checkEnd())

{

**int** bd = board.countDiscs(Board.**BLACK**);

**int** wd = board.countDiscs(Board.**WHITE**);

**if** ((bd > wd) && currentPlayer == Board.**BLACK**)

{

**return** INFINITY/10;

}

**else** **if** ((bd < wd) && currentPlayer == Board.**BLACK**)

{

**return** -INFINITY/10;

}

**else** **if** ((bd > wd) && currentPlayer == Board.**WHITE**)

{

**return** -INFINITY/10;

}

**else** **if** ((bd < wd) && currentPlayer == Board.**WHITE**)

{

**return** INFINITY/10;

}

**else**

{

**return** 0;

}

}

**if** (!solve)

{

**if** (depth == maxDepth)

**return** Heuristics.eval(currentPlayer, board);

}

ArrayList<Move> moves = board.getAllMoves(currentPlayer);

**if** (moves.size() > 1)

{

Heuristics.scoreMoves(moves);

Collections.sort(moves, comparator);

}

**for** (Move mv : moves)

{

board.makeMove(mv);

**float** score = - minimax(board, -beta, -alpha, depth + 1);

board.undoMove(mv);

**if**(score > alpha)

{

alpha = score;

**if** (depth == 0)

{

**this**.bestFound.setFlipSquares(mv.getFlipSquares());

**this**.bestFound.setIdx(mv.getIdx());

**this**.bestFound.setPlayer(mv.getPlayer());

}

}

**if** (alpha >= beta)

**break**;

}

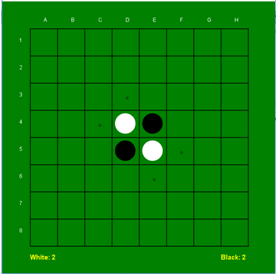
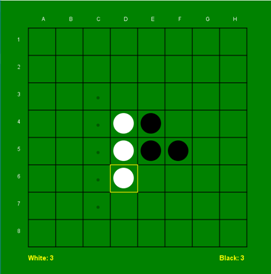
**return** alpha;

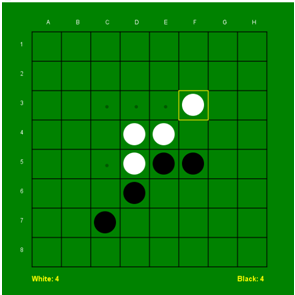
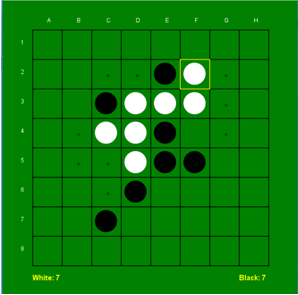
}

Depth is set to 6.

If the board is not solved, i.e, the computer does not get any point the depth is set to max depth and this condition prunes the tree branch which is being created.

The following screen shots gives a better idea of how the game has been implemented and is played:

For the computer the moves are compared with their possible estimation of the points scored. Based on the calculated scores it will decide which move to choose:

**class** MoveComparator **implements** Comparator<Move>

{

**public** **int** compare(Move move1, Move move2)

{

**if** (move1.getEval() > move2.getEval())

**return** 1;

**else** **if** (move1.getEval() < move2.getEval())

**return** -1;

**else**

**return** 0;

}

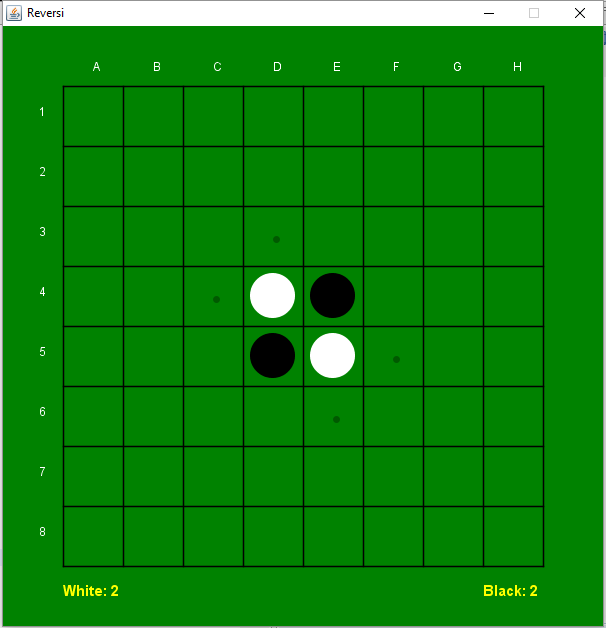
}

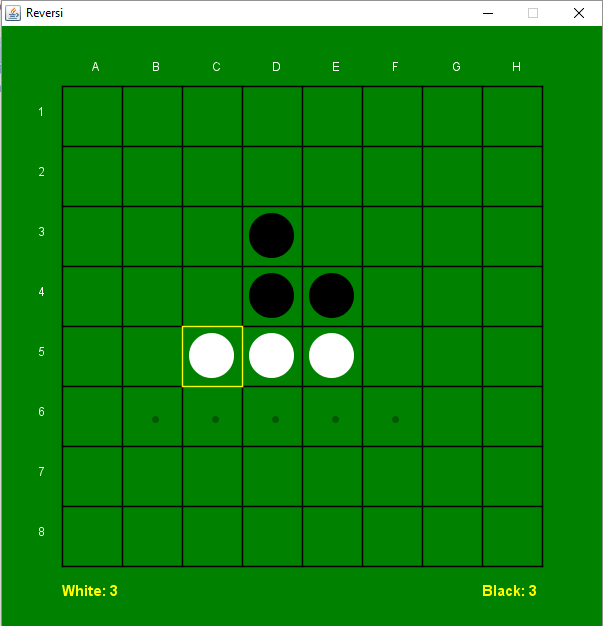
Program Output:

When program is executed, it displays the initial board and wait for player input.

Player - black disc

Computer- White disc



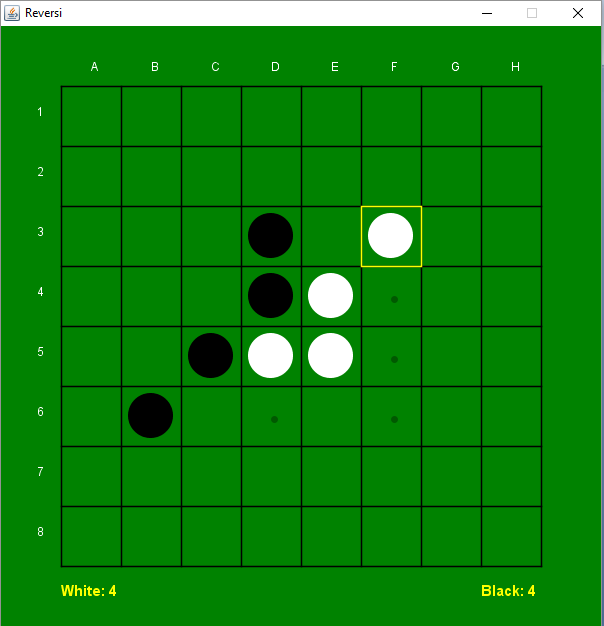


Player Move: Row:3 Column:4

Number of times the minimax algorithm is called: 1453

Obtained highest points for this move: 0.0

Computer's Move: Row:5 Column:3

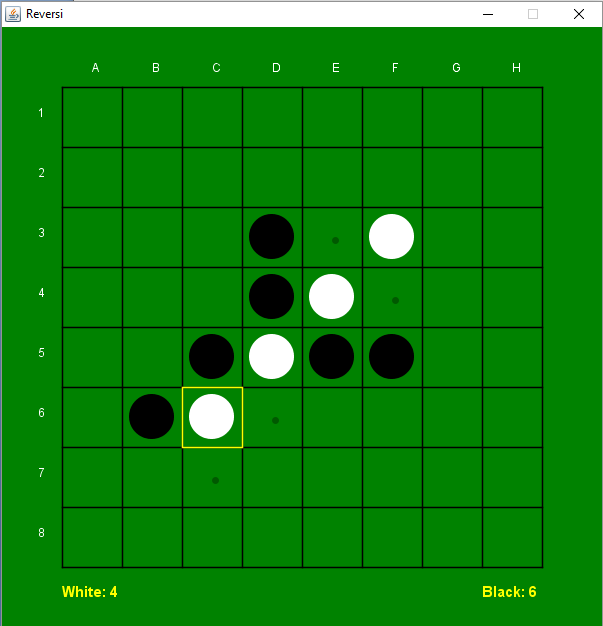


Player Move: Row:6 Column:2

Number of times the minimax algorithm is called: 2247

Obtained highest points for this move: 500.0

Computer's Move: Row:3 Column:6



Player Move: Row:5 Column:6

Number of times the minimax algorithm is called: 595

Obtained highest points for this move: 600.0

Computer's Move: Row:6 Column:3

Play continues until board is completely filled or when both player don't have a legal move.

