# Application of Recursion in Reversi Game using Artificial Intelligence

Nella Zabrina Pramata - 13516025
*Program Studi Teknik Informatika*
*Sekolah Teknik Elektro dan Informatika*
*Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia*
*sandhita.nellazp@gmail.com*

***Abstract*— Reversi is a board game that is like chess and go. Reversi is also called Othello. This game is played by two people. Each person uses black and white pieces to play. In our smart device, this game usually could be played in single player vs PC. The PC will use artificial intelligence to count the best move it could take to have victory. While the counting is using recursion method and this game will be less complicated, simplifying the algorithm so decrease the time it takes.**

***Keywords*—reverse, artificial intelligence, recursion.**

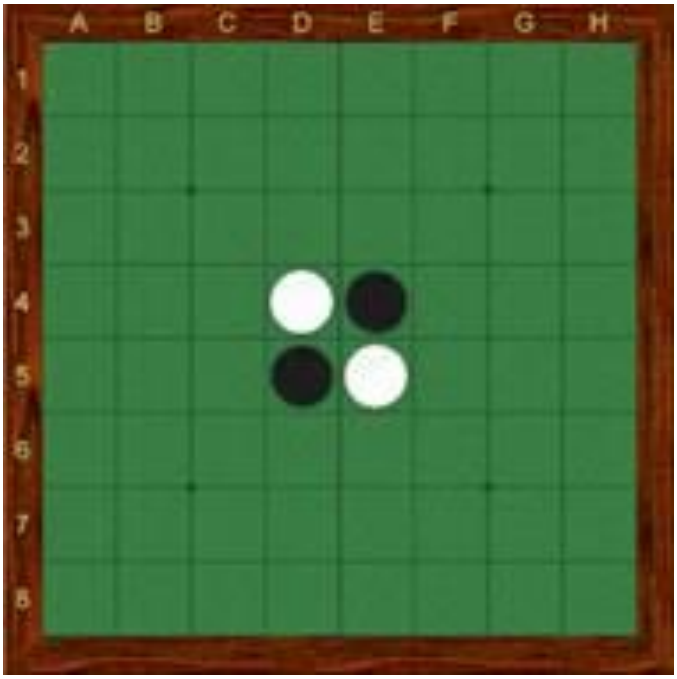
## I. INTRODUCTION

Reversi or Othello originally made by Mr. Goro Hasegawa after the war ended. This game uses black and white pieces because it originally derived from the pieces' color of Go game board. Black and white are represented as good and bad that battle each other to win the game. This game is aimed to have more pieces in the same color to get victory.

This game is very dynamic as a board game. The number of pieces in same color could change at once. Player could change other player piece if we have our color in another side of the row. Othello usually is played on 8x8 board. In the beginning, there will be four pieces reverse. Two for each color that is placed in the middle of the board diagonally.

Long time after war, which was the time when this game was made, Reversi or Othello is known all the world. After its first launching in 1973 by Tsukada Co. it was sold well. The game is so popular as it goes digital. We could play this game in computer or mobile phone, either online or offline. Thus, allowing player to explore more about Reverse or Othello.

In this game, especially in single player mode, the opponent (PC) works by using artificial intelligence. The AI helps so much because it could work like human or the real opponent. The technique to calculate the best step to take for opponent is recursion. Recursion is usually implemented in algorithm to help in solving computational problem.




Picture 1. Reversi Game Board
Source: https://www.yourturnmyturn.com

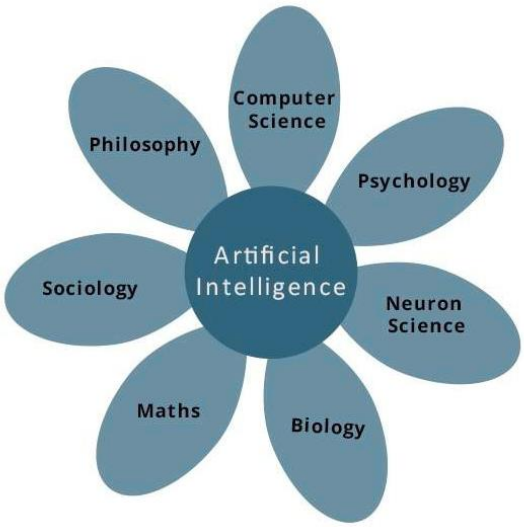
## II. THEORY

### A. Artificial Intelligence

Artificial Intelligence is [The automation of] activities that we associate with human thinking, activities such as decision making, problem solving, learning (Bellman, 1978). AI is a way of making a computer, a computer-controlled robot, or a software thinks intelligently (tutorialspoint). This intelligence is created to have similar intelligence like they way human think. Why must AI? AI is made to create expert systems that could behave, learn, demonstrate, explain, and advice its users, and to implement human intelligence in machines. AI is about how human brain thinks, learn, decide, work while solving problem. Those would be the basis of developing intelligent software and systems.

The disciplines of artificial intelligence consist of philosophy, computer science, psychology, neuron science, biology, maths, and sociology. Those majors are

associated with artificial intelligence, so AI could do reasoning, learning, and problem solving.




Picture 2. Disciplines of Artificial Intelligence
Source: https://www.tutorialspoint.com

AI was made to help the computer work more like human. The different of programming with and without artificial intelligence are:

- Without AI
  Computer could answer specific questions that is meant to solve
  With AI
  Computer could answer generic questions that is meant to solve
- Without AI
  If there is modification in program, there could be some change in its structure
  With AI
  The program could receive modification without change its structure. The method is by put highly independent pieces of information together. So, modification won't change AI programs
- Without AI
  Modification is not quick and easy. It may affect the program (could affect adversely)
  With AI
  Modification could be done quick and easy to program

Before AI use in programming, the knowledge has some unwelcomed properties

- Volume is huge
- Not well-organized
- Keep changing constantly

AI technique is to organize and use knowledge efficiently so

- AI should be perceivable by the people who provide it
- AI should be easily modified so it will help in correcting errors
- AI should be useful in many situations although sometimes it is incomplete or inaccurate

The basic techniques of artificial intelligence are recursion and searching.

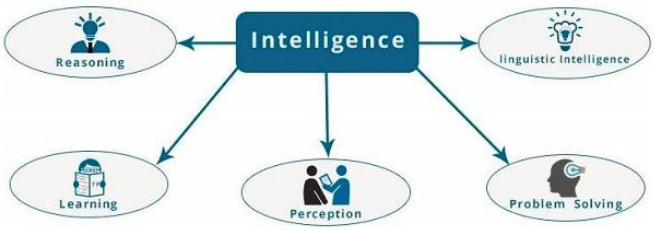
- Recursion
  Recursion is a function that call its function again. This function consists of basis and recurrent.
- Searching
  This technique to search the initial and final point for the solution. Searching consists of two methods, that are
  - Blind un-informed search
  - Informed search

The difference between human and machine intelligence are

- Human perceives by pattern while machine perceives by data and set of rules
- Human store and recall information by pattern, while machine by searching
- Human can figure object even when it has uncompleted part, while machines can't do

From these problem, artificial intelligence is made. It is composed of

- Reasoning
  Activity to enable us to provide basis for judgment, making decisions, and prediction
- Learning
  Activity to increase knowledge or skill by studying, practicing, being taught, or experiencing
- Problem solving
  Activity to perceive and try to have solution by taking some path
- Perception
  Activity to acquiring, interpreting, selecting, and organizing sensory information
- Linguistic intelligence
  Activity to use, comprehend, speak, and write the verbal and written language




Picture 3. Composition of Artificial Intelligence
Source: https://www.tutorialspoint.com

One of the application that use artificial intelligence is gaming. Usually artificial intelligence is used in strategic gamme like chess, poker, tic-tac-toe. Reversi or Othello is one of the example that use artificial intelligence. Machine use artificial intelligence, so it could think large number of possible positions based on heuristic knowledge (informed)

B. Recursion

A function is called recursion if it included in itself and called itself. Sometimes, this function is called recurrence relation.

Recursion consists of two parts:

- Basis
  Basis consists of initial value. This basis won't call itself anymore. In recursion, basis handles the termination of the function and will make the recursion function defined because basis will return value
- Recurrent
  This part is to call the function again. Every time the function calls its function again, the function must be one step closer to basis.

The example of recursion in the real world is Russian doll. If we remove the doll and separate its top and bottom halves, we could see smaller doll inside. This recursion keeps going. This step is called recurrent. Until, it is just one piece, the tiniest one that does not open. This step is called basis.

Picture 4. Recursion in Russian Doll
Source: https://www.khanacademy.org

From this Russian doll, we could get algorithms. This algorithm is designed to solve a problem by solving a smaller one of the same problem, unless the problem is so small that this could solve directly.

Recursion in computer programming has two meanings, that are

- Recursive procedure
  This procedure has ability to call itself. Usually it has capability to save the condition it was in or the particular process it is serving when it calls itself. Typically, recursive procedure will save values in registers or data area stacks before calling itself again or at the beginning of sequence.
- Recursive expression
  This expression could be a function, algorithm, or sequence of instructions that loop back (call itself) until detects some condition (basis).

Another example in gaming is from sudoku. Sudoku could be played by trying every possible combination of numbers. This is called brute force. The code below is one of the possible algorithm to make sudoku. The algorithm is starting from the upper left board and moving across each row one column at a time. Then, move down to start next row. This recursion could be seen in condition of "the current box is filled", then it called its function again, that is solve_sudoku.

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 5 | 3 | | | 7 | | | | |
| 6 | | | 1 | 9 | 5 | | | |
| | 9 | 8 | | | | | 6 | |
| 8 | | | | 6 | | | | 3 |
| 4 | | | 8 | | 3 | | | 1 |
| 7 | | | | 2 | | | | 6 |
| | 6 | | | | | 2 | 8 | |
| | | | 4 | 1 | 9 | | | 5 |
| | | | | 8 | | | 7 | 9 |

Picture 5. Sudoku
Source: https://codemyroad.wordpress.com

```
function solve_sudoku ( current_box_id )

                if all the boxes are filled
                    return true; // solved!
                end

                // ignore already "filled" squares (assume they are correct!)
                if the current box is filled
                    return the result of: solve_sudoku( next box );

                end

                // hypothesize all 9 possible numbers
                for each possible number from 1 to 9
                    if that number is 'valid' (okay to put in box)
// test row,col,square
                        try that number in the box
                        if you can "solve_sudoku" for the rest of the puzzle
                            return true; // success!
                        end
                    end
```

```
                end

                return false; //
failure

            end
```

Picture 6. Example of Pseudocode Sudoku
Source: https://www.cs.utah.edu/

This recursion could go infinite loop. To avoid infinite running of recursive function, there are two properties that recursion must have

- Base criteria
  There must be at least base criteria or condition (basis). This is the condition where the function stops calling itself recursively
- Progressive approach
  The recursive calls should progress in such way, so it will come closer to the base criteria

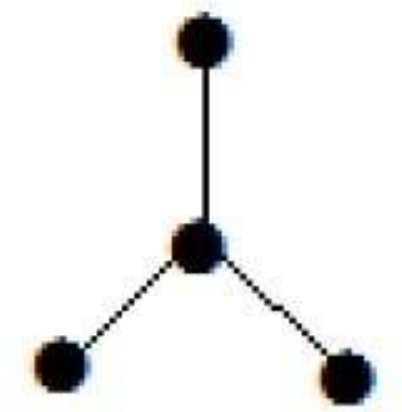
Although program could use iteration instead of recursion, there are reason why some prefers to use recursion. The first reason is recursion makes a program more readable. The next one is it enhanced CPU systems. The last one is recursion is more efficient than iterations.

Iterations takes more time complexity if the number if bigger. Likewise, in recursion, everything is constant. When a call made to function is O(1), hence the (n) number of times a recursive call is made makes the recursive function O(n).

While, space complexity is counted as what amount of extra space is required for a module to execute. In iterations, the compiler hardly requires any extra space. Compiler keeps updating the values of variables used in iterations. If in recursion, the system need to store activation record each time a recursive call is made. So, space complexity, recursion function may higher than iteration.

C. Tree

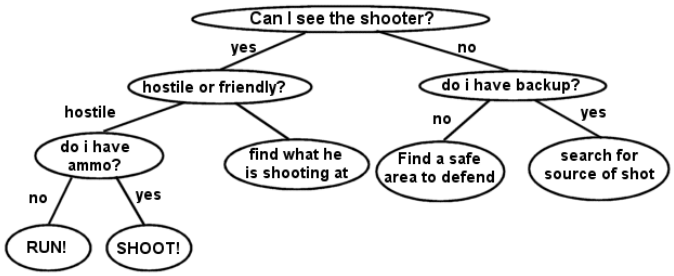
Tree is a special graph. It is indirection graph that is corelated and doesn't form circuit.



Picture 7. Graph Tree
Source: https://www.tutorialspoint.com

The graph is a diagram of vertex (point) and edge (line) that is connected to the points. Vertex is a point where multiple lines meet. It also called node. While edge is line that connects two vertices. There must be a starting vertex and an ending vertex for an edge. The vertex must not empty (with minimum one vertex) and the edge may empty.

Decision tree is a tree in which non-leaf (node) is labeled with an input feature. Some use least depth on the tree to decide or predict unseen data. This tree is used to modelized problems to direct to the solutions. Every node is a decision while leaf is solution.



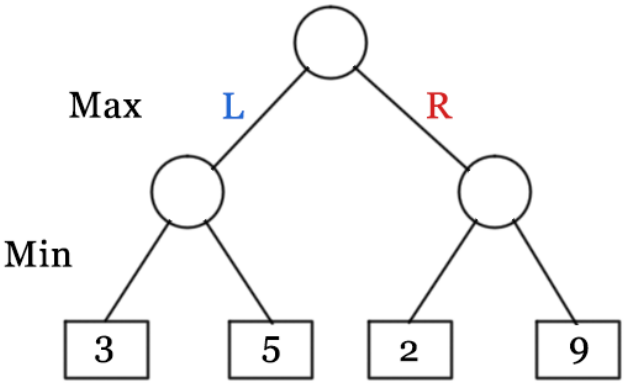

Picture 8. Decision Tree in Shooting Game
Source: http://archive.gamedev.net

D. MiniMax Algorithm

MiniMax algorithm in Game Theory is a backtracking algorithm. This algorithm is used in decision making and to find the optimal move for a player with assumption that the opponent do the optimal step, too. This Minimax is a function that is credit by John von Neumann (1928).

In this algorithm, the two players are calle maximizer and minimizer. The maximizer tries to get the highest score possible while the minimizer tries to get the lowest score possible.

When the maximizer has upper hand, the score of the board will be positive. If the minimizer has the upper hand, the board will be in negative value. This value is calculated by heuristics, and is unique for every board game.

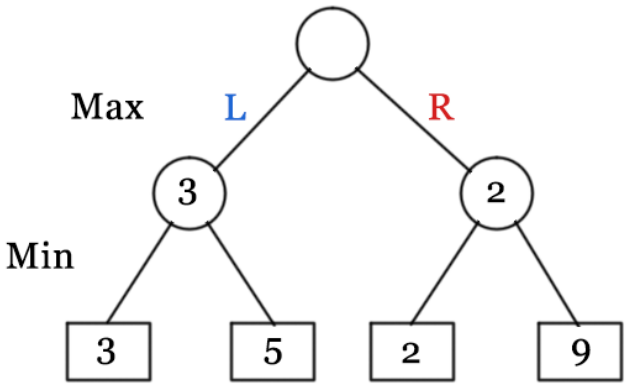



Picture 9. MiniMax Tree Before Evaluating
Source: http://www.geeksforgeeks.org

If a game has four final states and paths to reach final state are from root to four leaves. Assume you are the maximizer and

get the first chance to move (you are at root). Which move you take if the opponent plays optimally?

This algorithm will try all the possible moves and make decision tree. If you go left then the opponent will choose the least, 3. While if you go right, the opponent will choose 2. Then the optimal move for the maximizer is go to left.



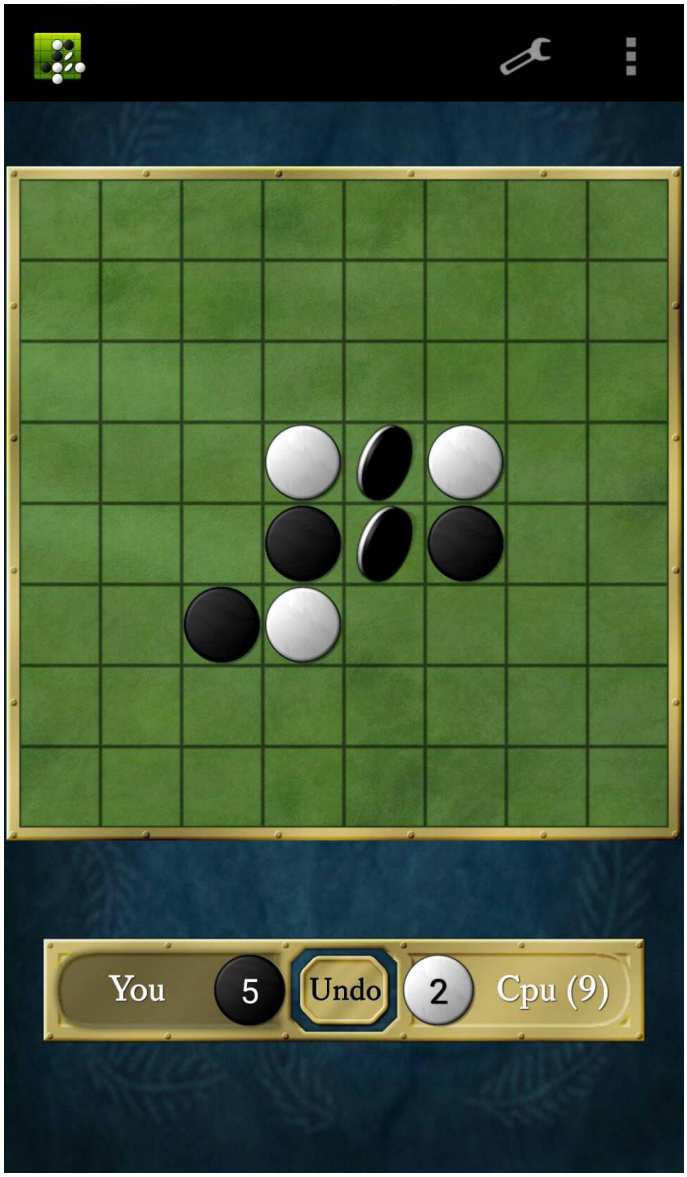

Picture 10. MiniMax Tree Best Move
Source: http://www.geeksforgeeks.org

## III. APPLICATION OF RECURSION IN REVERSI GAME USING ARTIFICIAL INTELLIGENCE

Artificial intelligence is needed to make game of Reversi or Othello. This artificial intelligence usually is applicated in single mode, when we compete with opponent (PC). Opponent will calculate the best step to take, to defeat the player. The difficulty of the step based on the level that the player has chosen. The difficulty could define the chance of the opponent win or anything else. But the value will be calculated as the comparison with the value return by recursion.

To make program of Reversi, we use recursion to make things easier. One of the function is when looking for multiple moves ahead, by maximizing the artificial intelligence's score and minimizing the player's score.

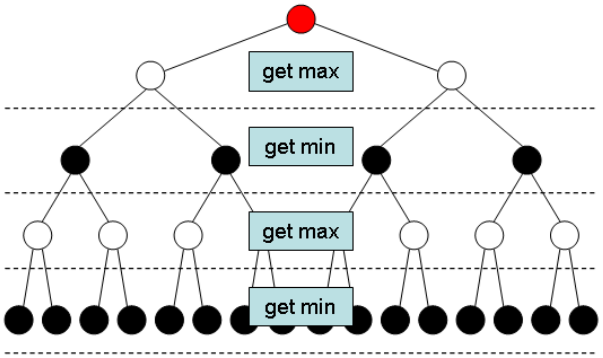
The application recursion in here is called MiniMax. MiniMax is recursion algorithm that evaluates scores over times, making choices based on maximum scores during some steps, while choosing minimum scores of others. This MiniMax explores all possible moves of the opponent and simulating the moves, and explores all of the possible moves of player just like artificial intelligence do.




Picture 11. Artificial Intelligence Takes Step

In the example of the reversion code that applicate MiniMax, tere are two recursive functions, one for simulating possible moves and step ahead "void simMoves", and one for calculating which move yields the best score "function findBestMove".

Void simMoves uses decision tree to see all possible moves that could happen. In this writing, the author won't write a lot about tree.




Picture 12. Decision Tree to Explore the Possible Moves
Source: http://mnemstudio.org

Each level of the tree represents either player or opponent's turn. From this tree, MiniMax algorithm determines the maximum score for the opponent and the minimum score for the player.

```
private void simMoves()
{
    this.mRoot = new Node();
    this.mMovesAhead = 0;

    // Get a list of all immediately available moves.
    ArrayList moves = this.findAllPossibleMoves(this.mComputer, this.mPlayer, this.matrix);

    if(moves.size() > 0){
        // Simulate moves from the immediate list.
        this.simMoves(this.mRoot, moves, this.matrix, this.mComputer, this.mPlayer);
    }
    return;
}

private void simMoves(final Node root,
                      final ArrayList moves,
                      final SquareState[][] aMatrix,
                      final SquareState playerA,
                      final SquareState playerB)
{
    if(++this.mMovesAhead < this.mTotalMovesAhead){
        for(Move aMove: moves)
        {
            Node aNode = new Node(aMove, root);
            root.setChild(GridMath.getID(aMove.X(), aMove.Y()), aNode);

            // Make a copy of the game board.
            SquareState[][] tempMatrix = new SquareState[Globals.GRID_SIZE_INTEGER][Globals.GRID_SIZE_INTEGER];
            for(int y = 0; y < Globals.GRID_SIZE_INTEGER; y++)
                for(int x = 0; x < Globals.GRID_SIZE_INTEGER; x++)
                    tempMatrix[x][y] = aMatrix[x][y];

            // Make a possible prospective move.
            tempMatrix[aMove.X()][aMove.Y()] = playerA;
            // Flip the simulated pieces for the move.
            Traverse t = new Traverse(aMove.X(), aMove.Y(), playerA, playerB, tempMatrix);
            ArrayList flips = t.getFlips();
            this.flipPieces(flips, playerA, tempMatrix, false);

            // Simulate the opponent's possible counter moves.
            ArrayList tempMoves = this.findAllPossibleMoves(playerB, playerA, tempMatrix);
            if(tempMoves.size() > 0){
                this.simMoves(aNode, tempMoves, tempMatrix, playerB, playerA);
            }
        }
    }
    return;
}
```

Picture 13. Algorithm of void simMoves
Source: http://mnemstudio.org

Here the void starts by creating the root node of a decision tree. Then, all possible moves that the opponents could do are passed to the second simMoves void, so do the player's moves.

Most of the important part here, is this void keep calling itself (recursion) by "this.simMoves". This temporary game boards contains the simulated moves that is evaluated by recursion.

Function findBestMove also use recursive to traverse the tree backward from the leaves to the root node While traverse back, it calculates the best move that could happen.

```
public Move findBestMove()
{
    Move bestMove = null;
    ArrayList children = this.mRoot.getChildren();
    if(children.size() > 0){
        findBestMove(this.mRoot, true);
        // Now get the max from the root's children
        int bestIndex = 0;
        for(int i = 0; i < children.size(); i++)
        {
            // Bias is imposed here to simulate more strategic behavior.  Occupying corners and
            // edges of the game board often lead to strategic advantages in the game.
            if(children.get(i).getMove().X() == 0 && children.get(i).getMove().Y() == 0 ||
               children.get(i).getMove().X() == 0 && children.get(i).getMove().Y() == Globals.GRID_SIZE_INTEGER - 1 ||
               children.get(i).getMove().X() == Globals.GRID_SIZE_INTEGER - 1 && children.get(i).getMove().Y() == 0 ||
               children.get(i).getMove().X() == Globals.GRID_SIZE_INTEGER - 1 && children.get(i).getMove().Y() == Globals.GRID_SIZE_INTEGER - 1){
                // Highest bias toward corners.
                children.get(i).setMaxScore(children.get(i).getMaxScore() + this.mCornerBias);
            }else if(children.get(i).getMove().X() == 1 && children.get(i).getMove().Y() == 0 ||
               children.get(i).getMove().X() == 0 && children.get(i).getMove().Y() == 1 ||
               children.get(i).getMove().X() == 1 && children.get(i).getMove().Y() == 1 ||
               children.get(i).getMove().X() == Globals.GRID_SIZE_INTEGER - 2 && children.get(i).getMove().Y() == 0 ||
               children.get(i).getMove().X() == Globals.GRID_SIZE_INTEGER - 1 && children.get(i).getMove().Y() == 1 ||
               children.get(i).getMove().X() == Globals.GRID_SIZE_INTEGER - 2 && children.get(i).getMove().Y() == 1 ||
               children.get(i).getMove().X() == 0 && children.get(i).getMove().Y() == Globals.GRID_SIZE_INTEGER - 2 ||
               children.get(i).getMove().X() == 1 && children.get(i).getMove().Y() == Globals.GRID_SIZE_INTEGER - 1 ||
               children.get(i).getMove().X() == 1 && children.get(i).getMove().Y() == Globals.GRID_SIZE_INTEGER - 2 ||
               children.get(i).getMove().X() == Globals.GRID_SIZE_INTEGER - 1 && children.get(i).getMove().Y() == Globals.GRID_SIZE_INTEGER - 2 ||
               children.get(i).getMove().X() == Globals.GRID_SIZE_INTEGER - 2 && children.get(i).getMove().Y() == Globals.GRID_SIZE_INTEGER - 1 ||
               children.get(i).getMove().X() == Globals.GRID_SIZE_INTEGER - 2 && children.get(i).getMove().Y() == Globals.GRID_SIZE_INTEGER - 2){
                // Bias against Region4.
                children.get(i).setMaxScore(children.get(i).getMaxScore() + this.mRegion4Bias);
            }else if(children.get(i).getMove().X() == 0 ||
                children.get(i).getMove().X() == Globals.GRID_SIZE_INTEGER - 1 ||
                children.get(i).getMove().Y() == 0 ||
                children.get(i).getMove().Y() == Globals.GRID_SIZE_INTEGER - 1){
                // Lower bias toward edges.
                children.get(i).setMaxScore(children.get(i).getMaxScore() + this.mEdgeBias);
            }
            if(children.get(i).getMaxScore() > children.get(bestIndex).getMaxScore()){
                bestIndex = i;
            }
        }
        bestMove = children.get(bestIndex).getMove();
    }

    return bestMove;
}

private void findBestMove(final Node root, final boolean getMaxFromMin)
{
    // The idea behind this recursive method, is to traverse all the way out to the leaves of
    // the tree, then calculate scores for parent nodes while returning back to the root.
    ArrayList children = root.getChildren();
    if(children.size() <= 0){
        for(Node child : children)
        {
            this.findBestMove(root, !getMaxFromMin);
            child.setMaxScore(child.getMinMaxFromChildren(getMaxFromMin));
            child.setMinScore(child.getMinMaxFromChildren(!getMaxFromMin));
        }
    }
    return;
}
```

Picture 14. Algorithm of function findBestMoves
Source: http://mnemstudio.org

Recursion is very important because it could divide problem into smaller and handle the smaller one first. The recursion keeps being every turn called when playing Reversi because the possibility could change over time. This calling function will expand possible moves and countermoves, so do the tree. After calculating the best moves, artificial intelligence will choose the highest score. Although Reversi could be made without recursion, this will make the code more complicated and the time spent to calculate the best move could be more longer.

## V. CONCLUSION

In Reversi or Othello game, artificial intelligence is used to take control as the opponent, for example when we take single player mode. Artificial Intelligence will search for the best solution, so the PC (opponent) could compete with player like a true human being. This artificial intelligence use recursion in the algorithm. In Reversi, recursion is used to find the best move that PC could have by using MiniMax algorithm. To determine the best move, it also use decision tree, then the choosing of best move will be easier. This recursion is very helpful because it solves small piece by piece. Not only will make the code simpler, but also will decrease the time taken to calculate the moves.

## VII. ACKNOWLEDGMENT

First, the author would like to thank God, so the author could finish this writing. Furthermore, the author would like to thank Dr. Ir. Rinaldi Munir, M.T as our lecturer of Discreate Mathematic in K01 that had help us this semester and we could understand those material well. The author would like to thank author's family and friends for the support, too, so author could do this writing well.

## REFERENCES

[1] Rinaldi Munir, *Matematika Diskrit.* Bandung: Informatika. 2005, chapter 3.
[2] Rinaldi Munir, *Matematika Diskrit.* Bandung: Informatika. 2005, chapter 9.
[3] https://channel9.msdn.com/Forums/TechOff/18426-Use-Recursion-in-AI-and-games (accessed on December 3rd 2017).
[4] http://www.kurzweilai.net/a-formula-for-intelligence-the-recursive-paradigm (accessed on December 3rd 2017).
[5] https://ndanul.wordpress.com/2014/11/26/kecerdasan-buatanartificial-intellegence-ai/ (accessed on December 4th 2017).
[6] http://www.o-wc.com/history/ (accessed on December 4th 2017).
[7] https://www.tutorialspoint.com/artificial_intelligence/artificial_intelligence_overview.htm (accessed on December 4th 2017).
[8] https://www.yourturnmyturn.com/rules/reversi.php (accessed on December 4th 2017).
[9] http://eecs.wsu.edu/~cook/ai/lectures/p.html (accessed on December 4th 2017).
[10] http://whatis.techtarget.com/definition/recursion (accessed on December 4th 2017).
[11] https://www.khanacademy.org/computing/computer-science/algorithms/recursive-algorithms/a/recursion (accessed on December 4th 2017).
[12] https://www.cs.utah.edu/~germain/PPS/Topics/recursion.html (accessed on December 4th 2017).
[13] https://www.tutorialspoint.com/data_structures_algorithms/recursion_basics.htm (accessed on December 4th 2017).
[14] http://mnemstudio.org/game-reversi-example-2.htm (accessed on December 4th 2017).
[15] https://www.tutorialspoint.com/graph_theory/graph_theory_trees.htm (accessed on December 4th 2017).

[16] http://artint.info/html/ArtInt_177.html (accessed on December 4th 2017).
[17] http://archive.gamedev.net/archive/reference/programming/features/trees 1/page4.html (accessed on December 4th 2017).
[18] http://www.geeksforgeeks.org/minimax-algorithm-in-game-theory-set-1-introduction/ (accessed on December 4th 2017).

## DECLARATION

I hereby declare that this paper is my original work, not a adapation, plagiarism nor a translation of any existing paper.

Bandung, December 4th 2017

Nella Zabrina Pramata
13516025