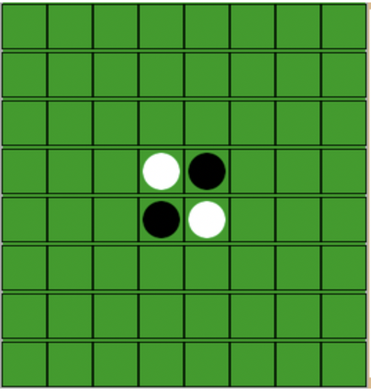
Artificial Intelligence in Playing Reversi

*Background*

Reversi also called Othello, is one of the classical board games, there are over millions of Reversi players in the world, and many players practice with computer programs nowadays. Two players take terms to move discs (game pieces, light on one side and dark on the other) on an 8×8 uncheckered board, in this project one player would be Artificial Intelligence (AI). Usually, start with four discs in the center, two pieces with the light side up, two pieces with the dark side up, with same-colored disks on a diagonal with each other (Wikipedia), the initial state is shown in *fig 1*, and a total of 64 discs could be placed on the game board.

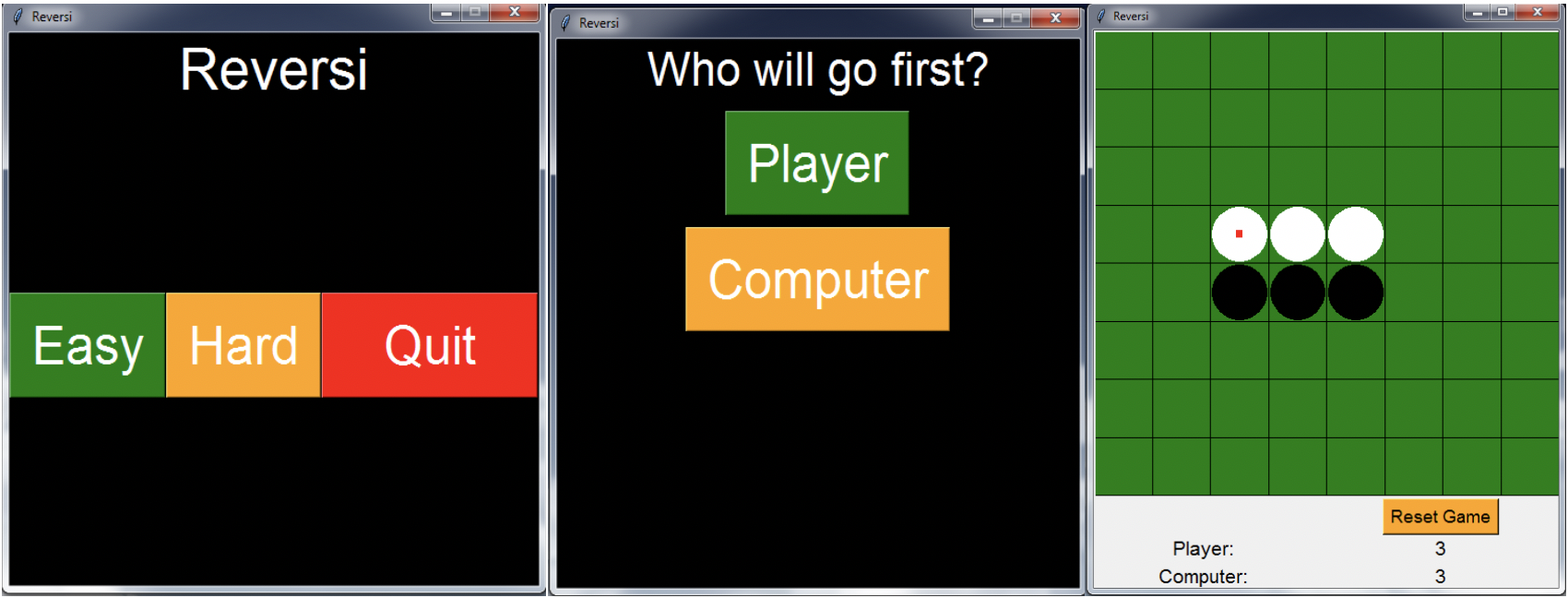


*fig 1*

In the rule of Reversi, dark disc usually moves first. However, the player would be assigned with the dark disc in this project, and the light disc is assigned to AI, if AI moves first, the white disc will move first. Player and AI follow the rest of the rules of the board game, must move at a position that there have at least one horizontal, or vertical, or diagonal occupied line between the new disc and another dark disc, with one or more contiguous light discs between them. When neither player or AI can move, the game over. This occurs when the board has filled up or when neither player can legally place a piece in any of the remaining boxes. This means the game may end before the board is filled. This possibility may occur because one player has no pieces remaining on the board. (Wikipedia) The one with the most pieces on the board at the end of the game will win. There is a possibility that the game could end up with a tied state, which the player and AI both have 32 discs at the end of the game.

*Problem Description*

In this project, I have created a python program which could allow users to play the board game, Reversi against AI. A Tkinter module is imported for the game board interface, and I have implemented two different algorithms for AI. The first GUI page of the program would allow the user to choose the level of difficulty he/she wants to play, or the user could decide to quit the game. After user picks the level of difficulty, the next page asks the user to choose either the player or AI move first. Next, the 8×8 board is be displayed with the initial state of the game; the user would be allowed to click on the board and start to play if the user chooses the player to move first. Otherwise, the AI will move first, and the user would be allowed to click the board to place discs after AI’s tern. The most recent move by the player or AI has an indicator which is a small red box in the center of the disc which the player or AI placed. The play’s and AI’s current scores are displayed under the game board, and a reset button will allow the user to restart the game at any time. Please view fig 2 for the screenshots of the program, and the *Flowchart* at the end of the report for more details.

**

*fig 2*

After the game ends, either the whole board is filled, or disc of one color are all taken by the other color, a message will pop-up and indicate who wins the game, or the game is tied.

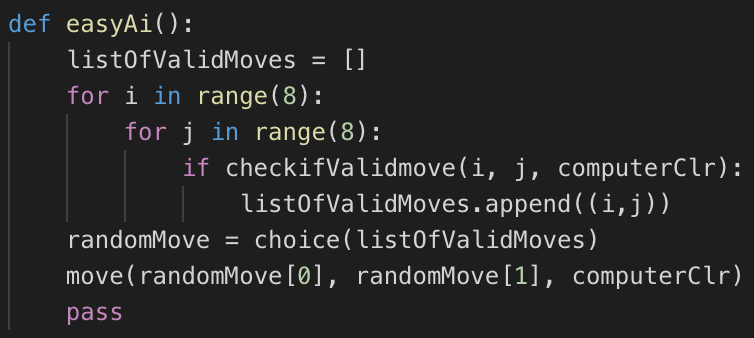
*Objective and Goal*

The goal of this project is to develop a board game-playing program, which can play Reversi. The game implemented with the following properties. It is a two-person game where two players take turns to move in the board, and it is a fair game of perfect, without any hidden information for both players (AI or human player). The primary purpose is to create a game-playing artificial intelligence program that uses two different algorithms. The program will be implemented in python programming language and illustrated in a simple graphical user interface to allow human players to challenge AI. The goal for AI is to win the game against a player, and the player will also try his/her to win to the game against AI.

My ultimate goal for this project is to find and implement an algorithm for AI that makes the AI unbeatable. However, the goal is not accomplished yet, the two algorithms I implemented are not perfect, players still have chances to win the game. Also, one important goal I have accomplished is that that algorithm I used for the hard mode make the human player struggle a lot more compare to the algorithm for the easy mode. Please view the Methodology and Conclusion section for more details about algorithms and program tests.

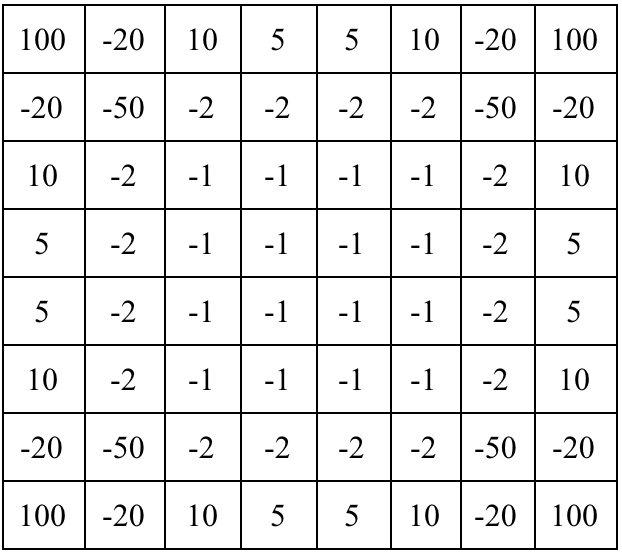
*Methodology*

The User has two different choices of difficulties, Easy and Hard. I implemented two distinct algorithms for varying levels of difficulties. They both shared a function named checkifValidmove, which will do a breadth first search for all possible moves for a particular turn for AI. For the Easy mode, I implemented Random Search. However, it is not completely random; it still follows the rule of the game. In this simple algorithm, all arrays are searched for valid moves, and if any possible valid movement is found, it would be added to another array, which stores a list of valid steps. Next, a random indexes tuple would be selected from this array, and AI would move on that index in this particular term, this algorithm would be repeated every AI’s term. Please view Pseudocode 1 for Random Search algorithm.



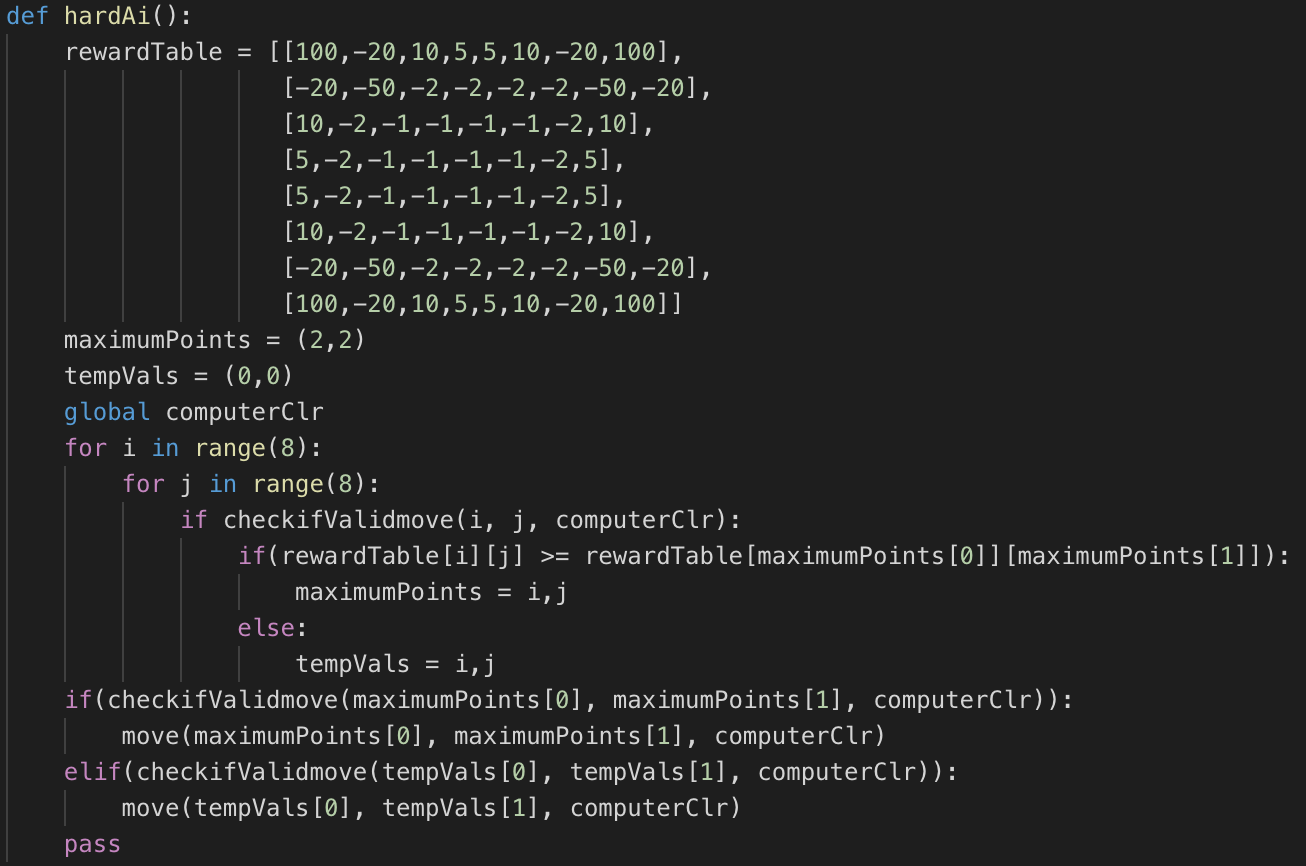
*Pseudocode 1*

The second algorithm I implemented is Local Maximization, which is used for the hard mode in the game. The user would face AI using this algorithm if he/she decided to play the hard mode. In this algorithm, a reward table would be used. View Table 1 for the reward table.



*Table 1*

The reward table is in the form of a two-dimensional array which has reward values according to the positions in the 8 × 8 game board. A dummy maximum value is initialized in a variable, and then it would get the valid moves from the board. Next, the value would compare them all with maximum value; if the maximum value is less than valid move maximum value, the variable would be updated with its indexes. After checking all valid moves, we got the maximum value positions, and then AI would place next step on that position. Please view Pseudocode 2 for Local Maximization algorithm.



*Pseudocode 2*

*Discussion and Conclusion*

*Algorithm Comparison*

The two algorithms I implemented are Random Search algorithm and Local Maximization algorithm. They shared a few similarities. First, the runtime complexity for both algorithms are identical, they both have two for-loops while they check for valid moves and other functions. The complexity would be O(n2) for both cases. Second, both algorithms call the checkifValidmove function at the very beginning, which makes sure both algorithms follow algorithm follows the rule of the Reversi game.

Although they have some specs in common, they result in totally different moves for AI, which would create a huge gap of difficulties between the easy mode, which implemented the Random Search algorithm and hard mode, which implemented the Local Maximization algorithm.

One obvious difference is that the Local Maximization algorithm has a reward table, which used a two-dimensional array which stores values for each index on the game board. Furthermore, the Local Maximization algorithm uses the reward table to figure out the maximum value from all possible valid moves by calling the maximumPoints function, and AI would move accordingly. In contrast, the Random Search algorithm is simply calling the randomMove function which would randomly pick a possible valid move from the result of the checkifValidmove function. By assigning values for each position and calculate the maximum value of each possible move, the Local Maximization algorithm would result in a much more optimal move comparing to the Random Search algorithm.



*Table 2*

I have played total 40 games for testing purpose, ten under easy mode and player moves. First, ten under easy mode and AI moves. First, ten under hard mode and player moves first, and ten under hard mode and AI moves first. The tests result shown in Table 2. The number illustrates the number of discs left on the board for both player and AI; the result column shows the winner of the game.

*Fig 3*

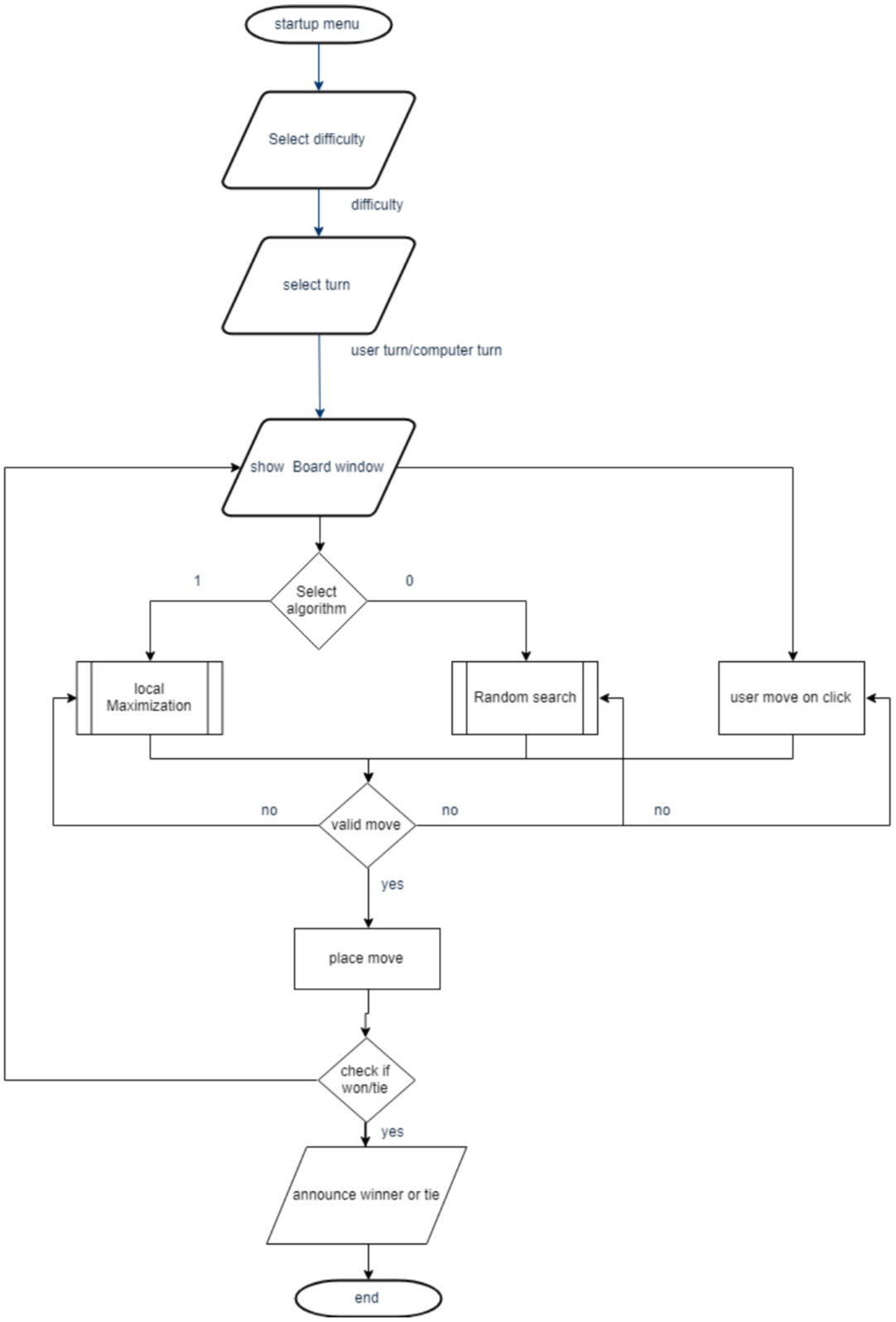
The win rate for the player and AI in each category is shown in Fig 3; it is evident that AI has better win rate in hard mode compare to the easy mode, accordingly to my test result, AI has around 80 percent win rate no matter it goes first or second. In contrast, AI’s win rate is lower than 40 percent in the easy mode.

*Summary*

According to my research, the most intelligent Reversi AI can win the Reversi champion in the world. That is a significant success in AI study. In my project, two different algorithms were used to be applied in the program; they do not have a 100 percent win rate against the human player. Also, they share some other similarities, the time complexity is the same for both algorithms, O(n2), and they share some identical functions, such as checkifValidmove. However, the Local Maximization algorithm implemented a reward table. Therefore, it would result in more optimal moves and have a better win rate.

For future work, I would like implemented mini-max, α-β pruning and some other algorithms to the Reversi AI, try to find an algorithm which the AI unbeatable. Also, try to find an algorithm which has a better time complexity comparing to the two I implemented.

*Flowchart*



References:

<http://web.eecs.utk.edu/~zzhang61/docs/reports/2014.04%20-%20Searching%20Algorithms%20in%20Playing%20Othello.pdf>

<https://en.wikipedia.org/wiki/Reversi>

<http://dhconnelly.com/paip-python/docs/paip/othello.html>

<http://www.radagast.se/othello/howto.html>