# Applying and Comparing UCT and Minimax with Alpha-Beta Pruning on Game Reversi

Zhiyuan He[1] and Zhengxian Lin[2]

*Abstract*— **This paper mainly focus on applying and comparing UCT and minimax search as agents for a game named reversi. This paper is the final project report for CS531 Artificial Intelligence course on Oregon State University. This paper introduce how we implemented Reversi game programs with both Minimax and Upper Confidence Bounds for Trees (UCT) Search algorithm, and discussed the theories and characteristics, experimented performances of the two algorithms.**

## I. INTRODUCTION

Reversi is one of the traditional but challenging strategy board games for two players. In Reversi players take turns placing game pieces with their assigned colors(black or white) onto a 88 uncheckered board. During a play, any game pieces of the opponents color that are in a straight line or a diagonal and bounded by the game piece just placed and another game piece of the current players color are turned over to the current players color. When the board is filled or another player have no position to place next piece, the player whose game pieces are more that anothers would be the winner. And if the two players have the same number of disks, then the game end in a draw.

Upper Confidence Bounds for Trees (UCT) is a variant of Monte Carlo Tree. Monte Carlo Tree (MCT) search is a search method a method for finding optimal decisions in a given domain by taking random samples in the decision space and building a search tree according to the results. [1] The Alpha Go,a famous computer program that plays the board game Go, uses Monte Carlo tree search, guided by a ”value network” and a ”policy network,” both implemented using deep neural network technology.

Minimax is a kind of backtracking algorithm that is used in decision making and game theory to find the optimal move for a player. Minimax assumes that each player plays the best move, one player is trying to maximise the outcome and the other minimise it. It is widely used in two player turn based games such as Tic-Tac-Toe, Backgamon, Mancala, Chess, etc.

Both minimax and UCT are widely used on game theory to determinate the best move to win the game. Minimax are not applied to Go as the domain of each step is too wide even with alphabeta pruning. However, UCT builds a highly selective tree will may miss some crucial moves. Minimax with full width does does not suffer from disadvantages. But with low depth, minimax usually loss its performance. So, for Reversi, how to implemented and applied UCT and minimax on it and how's the performance difference with each other is the main focus of this paper.

## II. DESCRIPTION

### A. Reversi

The Reversi, also called Othello, is a kind of strategy board game which is played by two players on 8 x 8 chessboard. The player reverse opponents chess to win this game. There is a rule to reverse opponents chess. When a chess is putted on the board, the same row, column, diagonal of it will change too. For example, white player play one chess on board, assumes that is $C_{i,j}$, i is its row, and j is its column. If there exist one chess on $C_{k,j}$,( k does not equal i), and no gap between two of them, which is means $C_{n,j}, (k < n < i)$is filled by opponents chess, then all of the $C_{n,j}, (k < n < i)$ will become white. For the same row, same diagonal, it will be same rule. Also, a position cannot be filled by player when this action happening change nothing. There are two ways to win. The first way is that when the chessboard is filled full of chesses, which player has the bigger number of chess of is winner. If both of them are same, the result is draw. Second way is that when one of the player has no chess, its opponent is winner. The pseudocode is showed in Fig.1.

### B. UCT

Upper Confidence Bounds Trees (UCT) is one member of MCTS family. It combines UCB with MCTS. UCT is to calculate an actions value approximately through simulation. The roll out way we used in this paper is playing randomly. There are four steps for UCT:

First, selection: If the current node has been simulated(expanded) then using UCB to calculate its value, which is $\frac{w_i}{n_i} + C * \sqrt{\frac{\ln N_i}{n_i}}$, where $w_i$ is the number of wins, $n_i$ is the number of plays. C is a constant which is $\sqrt{2}$ in our project, $N_i$ is the sum of the number of plays of all nodes which is at same layer with the current node.

Second, expansion: after selection, expands this node and get its children nodes.

Third, simulation: after expansion, random select a leaf node which has not value, which means which has not been simulated, doing simulation. In our project, simulation is random vs random, which means basing on the current node state, keeping doing random play with both player till the game finished.

[1]Zhiyuan He, Computer Science, Oregon State University. Email: `hezhi@oregonstate.edu`

[2]Zhengxian Lin,Computer Science, Oregon State University. Email: `linzhe@oregonstate.edu`

```
function playChess($C_{row,column}$)
if $C_{n,j}$.chesstype equals $C_{row,column}$,
  then for n0 in range(n)
          if n > i
            then Reverse($C_{i+n0,j}$)
                 numOfChange ← numOfChange + 1
            else:
                 Reverse($C_{i-n0,j}$)
                 numOfChange ← numOfChange + 1
if $C_{i,m}$.chesstype equals $C_{row,column}$,
  then for m0 in range(m)
          if m > j
            then Reverse($C_{i,j+m0}$)
                 numOfChange ← numOfChange + 1
            else:
                 Reverse($C_{i,j-m0}$)
                 numOfChange ← numOfChange + 1
if $C_{n,m}$.chesstype equals $C_{row,column}$, and n − i == m − j
  then for k in range(n-i)
          if n > i
            then Reverse($C_{i+k,j+k}$)
                 numOfChange ← numOfChange + 1
          else Reverse($C_{i-k,j-k}$)
            numOfChange ← numOfChange + 1
if numOfChange is not 0
  then return True
  else return False
function Reverse(c)
  if c is selfChessType
          then c ← opponentChessType
  else c ← selfChessType
```

Fig. 1. Pseudocode of Reversi

```
function UCTSearch($s_0$):
    create root node $v_0$ with $s_0$:
    while within time range do
        actions ← allowedActions($v_0$)
        if len(actions) == 0:
          then break
        for $a_0$ in actions
          states ← Result($s_0, a_0$)
        $v_1$ ← selectionPolicy(states,C)
        path, isWin ← simulation($v_1$)
        backpropagation (path, isWin, $v_1$)
function allowedActions(v):
    for i,j in board:
        if playChess($C_{i,j}$):
          then allowedactions ← (i,j)
    return allowedactions
function selectionPolicy(states,C):
    return Max($\frac{w_{si}}{n_{si}} + C * \sqrt{\frac{\ln N_{si}}{n_{si}}}$)
function simulation(v):
    while v is not terminal do
        choose a ∈ allowedActions(v) uniformly
        at random
        v ← PlayChess(v,a)
    return path for v, isWin
function backpropagation(path,isWin,v)
    while v is not null do
        N(v) ← N(v) + 1
        if isWin
          then W(v) ← W(v) + 1
        v ← parent of v
```

Fig. 2. Pseudocode of UCT

Forth, backpropagation: after simulation, the current node state is the final state, go back to the current state which agent is playing now, and update all of the value of the nodes through the path. Our UCT is limited by processing time , for example, 6 seconds. After 6 second it have to stop and decide which action to take. The pseudocode is in Fig.2:

### *C. Minimax*

Minimax with alpha and beta pruning is a classic algorithm for games. It maximizes self-value through simultaneous moves. It assumes that the opponents always choose the minimum value and it choose the maximum value for itself. We have several depths for it, depth means how far it simulate. When it reaches limited depth, which is leaf node, we use two evaluation function to evaluate it, and return to its parents.

Evaluation 1: The first evaluation is quite simple, we called it simple evaluation. It calculates self-chess number and opponent-chess number and return self-chess number minus opponent-chess number. The formula is like:

$$num_{self} - num_{opponent}$$

Evaluation 2: The second evolution consider chess number, and also consider the position of chesses. When design Minimax algorithm, we found the simple evaluation is too weak, and let it paly with our friends who is good at playing Reversi, then they told us that the position is quite important, for example, the four corners on the board are the most important positions, because one a player play it, another player cannot change it forever. It also helps to reverse opponents chess. Second, the positions on the four sides of the board is more important than the position on the center. Thus, we come out a good evaluation, which we called it complex evaluation, to evaluate the state of the board. The value of corner is 100, the value of side is 4, and the value of center is 1, which neither corner or side. Then, return then value of self-value minus opponent-value. The formula of this is like:

$$(num_{self.corner} * 100 + num_{self.side} * 4 + num_{self.center}) \\ -(num_{opponent.corner} * 100 + num_{opponent.side} * 4 + num_{opponent.center})$$

```
function minmax_alpha_beta(S,$\alpha$,$\beta$)
        return MaxValue(S, $\infty$,+$\infty$)
function MaxValue(S,$\alpha$,$\beta$)
        if TERMINAL-TEST(S)
        then return Evaluation(S)
        v $\leftarrow \infty$
        actions $\leftarrow allowedActions(v_0)$
        for each a in actions
           v $\leftarrow$ Max(v,MinValue(Result(s,a),$\alpha$,$\beta$))
           if v $>= \beta$ then return v
           $\alpha \leftarrow$ Max($\alpha$,v)
        return v
function MinValue(S,$\alpha$,$\beta$)
        if TERMINAL-TEST(S)
        then return Evaluation(S)
        v $\leftarrow \infty$
        actions $\leftarrow allowedActions(v_0)$
        for each a in actions
           v $\leftarrow$ Min(v,MaxValue(Result(s,a),$\alpha$,$\beta$))
           if v $<= \alpha$ then return v
           $\beta \leftarrow$ Min($\beta$,v)
        return v
```

Fig. 3. Pseudocode of Minimax with Alpha-Beta Pruning

## III. EXPERIMENTAL SETUP

As mentioned previously, this paper experiment the performance of the UCT and Minimax on Game Reversi. As a kind of backtracking algorithm, the performance of Minimax is largely depend on the max depth it can explore. For UCT, it chose a path till the end to find out the ultimate outcome (win/lose). When all immediate child nodes are explored, it uses UCT to exploit the experience. Because the time of this process is very long which is not we want as a game agent. In this project, this process is repeated within a time frame give. Then, it decides which move is most likely to make the player a winner.

To better measure the performance of UCT and Minimax with alpha-beta pruning on the game Reversi, the score of them got at the end of game is recorded as the judgment of performance. Compared with using the number of game they wins, the score can show more detail of each algorithm. For example, a win with 33 score should be different with a win with 64 score.

The experiment of this project would be arranged to 2 part: The first part will test the performance of Minimax with simple evaluation, Minimax with complex evaluation under different limited depth, UCT under different limited time against random agent. Each test will run 50 times, and the performances are measured by the win rate and the average score.

The second part of this experiment is test the performance of Minimax with simple evaluation against UCT, and Minimax with complex evaluation against UCT. The performances are recorded as former. One thing different from the former test is that the sequence of who play first will change to remove the effect of the advantage of who play first. The process of the experiment is as follow:

```
PART 1:
For each evaluation function,
    For each limited depth,
        1) Test the Minimax against Random agent.
        2) Record needed data.
For each limited time,
        1) Test the UCT against Random agent.
        2) Record needed data.

PART 2:
For each evaluation function,
    For each limited depth,
        For each limited time,
            1) Test the Minimax against UCT.
            2) Test the UCT against Minimax.
            3) Record needed data.
```

Fig. 4. The process of the experiment

## IV. RESULTS

After the experiment, we get the results for PART 1 as follow:

The average Win Rate and score of UCT against random agent is recorded in TABLE I:

TABLE I
THE PERFORMANCE OF UCT WITH LIMITED TIME

| UCT with limited time | Average Win Rate | Average Score |
|---|---|---|
| UCT (0.001) | 0.54 | 33.2 |
| UCT (0.005) | 0.57 | 32.65 |
| UCT (0.01) | 0.66 | 34.36 |
| UCT (0.05) | 0.86 | 38.02 |
| UCT (0.1) | 0.86 | 39.24 |
| UCT (0.5) | 0.98 | 41.7 |
| UCT (1) | 0.98 | 42.72 |
| UCT (2) | 1.0 | 45.46 |
| UCT (4) | 1.0 | 47.26 |
| UCT (6) | 1.0 | 47.64 |
| UCT (8) | 1.0 | 46.36 |

The average win rate of UCT with different limited time against random agent is also showed in Fig.5:

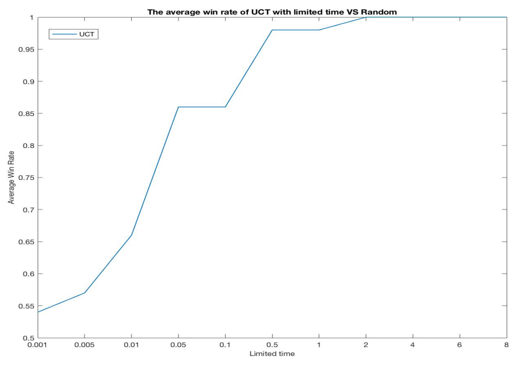

Fig. 5. Average win rate of UCT with limited time VS Random

The average score of UCT with different limited time against random agent is also showed in Fig.6:

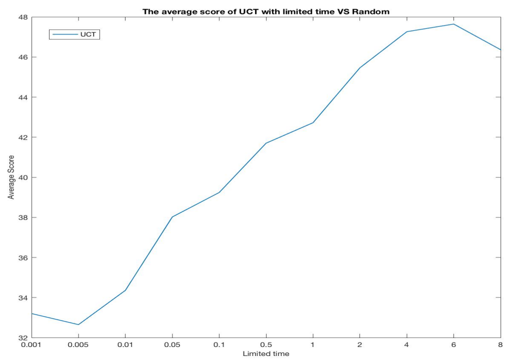

Fig. 6. Average score of UCT with limited time VS Random

The average Win Rate and score of Minimax against random agent is recorded in TABLE II:

TABLE II
THE PERFORMANCE OF MINIMAX WITH LIMITED DEPTH

| Minimax with limited depth | Average Win Rate | Average Score |
|---|---|---|
| Simple Minimax (1) | 0.47 | 32.63 |
| Simple Minimax (2) | 0.73 | 39.64 |
| Simple Minimax (3) | 0.76 | 39.83 |
| Simple Minimax (4) | 0.86 | 41.98 |
| Simple Minimax (5) | 0.8 | 41.78 |
| Simple Minimax (6) | 0.84 | 43.62 |
| Complex Minimax (1) | 0.48 | 32.72 |
| Complex Minimax (2) | 0.96 | 48.68 |
| Complex Minimax (3) | 0.98 | 46.7 |
| Complex Minimax (4) | 0.94 | 49.78 |
| Complex Minimax (5) | 0.96 | 48.6 |
| Complex Minimax (6) | 1.0 | 49.32 |

The average win rate of Minimax with simple evaluation and different limited depth against random agent is also showed in Fig.7:

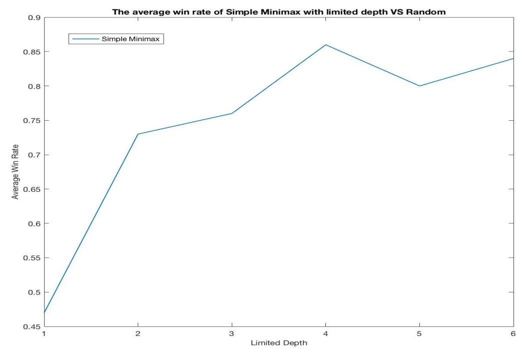

Fig. 7. Average win rate of Simple Minimax VS Random

evaluation and different limited depth against random agent is also showed in Fig.8:

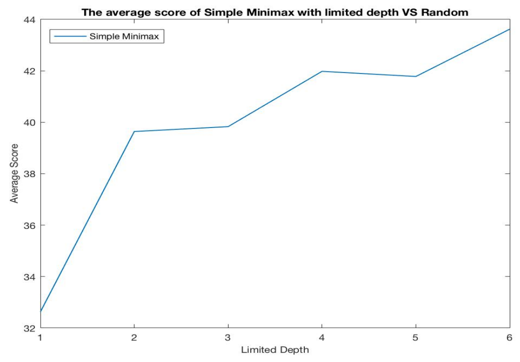

Fig. 8. Average score of Simple Minimax VS Random

The average win rate of Minimax with complex evaluation and different limited depth against random agent is also showed in Fig.9:

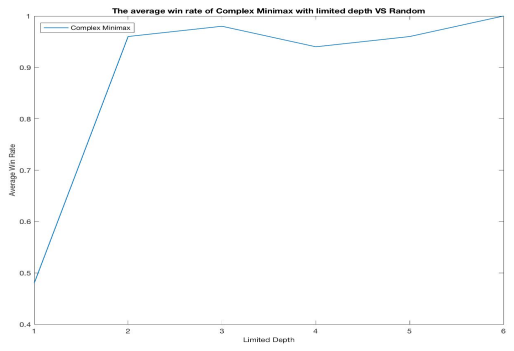

Fig. 9. Average win rate of Complex Minimax VS Random

The average score of Minimax with complex evaluation and different limited depth against random agent is also showed in Fig.10:

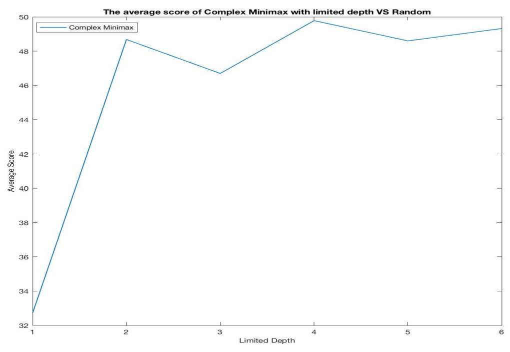

Fig. 10. Average score of Complex Minimax VS Random

For PART 2, we selected UCT with limited time(2, 4, 6, 8) to against simple and complex Minimax with limited depth(3, 4, 5, 6). And the results are recorded in TABLE III:

TABLE III
THE PERFORMANCE OF UCT AGAINST MINIMAX

| UCT against Minimax | Average Win Rate | Average Score |
|---|---|---|
| UCT(2) vs Simple Minimax(3) | 0.88 | 38.68 |
| UCT(2) vs Simple Minimax(4) | 0.96 | 41.64 |
| UCT(2) vs Simple Minimax(5) | 0.92 | 40.92 |
| UCT(2) vs Simple Minimax(6) | 0.68 | 28.4 |
| UCT(2) vs Complex Minimax(3) | 0.68 | 29.92 |
| UCT(2) vs Complex Minimax(4) | 0.16 | 15.48 |
| UCT(2) vs Complex Minimax(5) | 0.16 | 14.52 |
| UCT(2) vs Complex Minimax(6) | 0.08 | 15.08 |
| UCT(4) vs Simple Minimax(3) | 0.96 | 41.52 |
| UCT(4) vs Simple Minimax(4) | 0.88 | 37.56 |
| UCT(4) vs Simple Minimax(5) | 0.88 | 36.12 |
| UCT(4) vs Simple Minimax(6) | 0.8 | 34.52 |
| UCT(4) vs Complex Minimax(3) | 0.56 | 29.52 |
| UCT(4) vs Complex Minimax(4) | 0.32 | 21.12 |
| UCT(4) vs Complex Minimax(5) | 0.36 | 21.76 |
| UCT(4) vs Complex Minimax(6) | 0.12 | 15.52 |
| UCT(6) vs Simple Minimax(3) | 0.96 | 41.92 |
| UCT(6) vs Simple Minimax(4) | 0.96 | 41.24 |
| UCT(6) vs Simple Minimax(5) | 0.96 | 41.92 |
| UCT(6) vs Simple Minimax(6) | 0.8 | 35.16 |
| UCT(6) vs Complex Minimax(3) | 0.68 | 34.64 |
| UCT(6) vs Complex Minimax(4) | 0.32 | 21.16 |
| UCT(6) vs Complex Minimax(5) | 0.52 | 26.2 |
| UCT(6) vs Complex Minimax(6) | 0.08 | 14.76 |
| UCT(8) vs Simple Minimax(3) | 0.8 | 38.8 |
| UCT(8) vs Simple Minimax(4) | 1.0 | 44.4 |
| UCT(8) vs Simple Minimax(5) | 0.9 | 37.3 |
| UCT(8) vs Simple Minimax(6) | 1.0 | 41.9 |
| UCT(8) vs Complex Minimax(3) | 0.8 | 36.52 |
| UCT(8) vs Complex Minimax(4) | 0.12 | 18.48 |
| UCT(8) vs Complex Minimax(5) | 0.2 | 18.64 |
| UCT(8) vs Complex Minimax(6) | 0.16 | 15.36 |

The average win rate of UCT with limited time 2s against Minimax with different evaluation and different limited depth agent is showed in Fig.11:

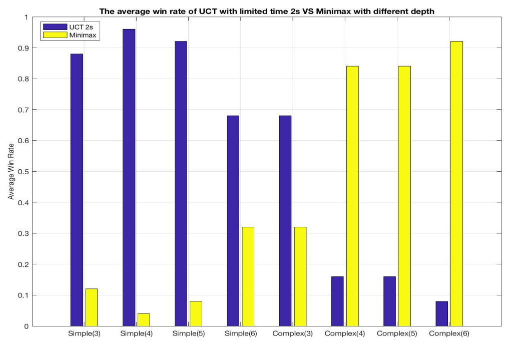

Fig. 11. Average win rate of UCT 2s VS Minimax with different depth

The average score of UCT with limited time 2s against Minimax with different evaluation and different limited depth agent is showed in Fig.12:

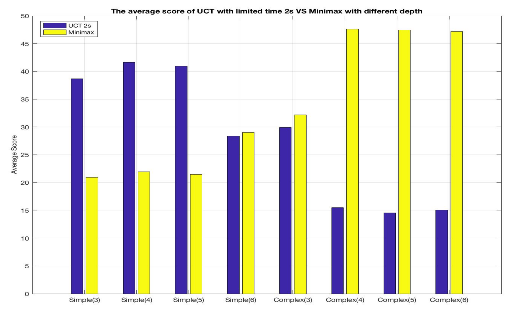

Fig. 12. Average score of UCT 2s VS Minimax with different depth

The average win rate of UCT with limited time 4s against Minimax with different evaluation and different limited depth agent is showed in Fig.13:

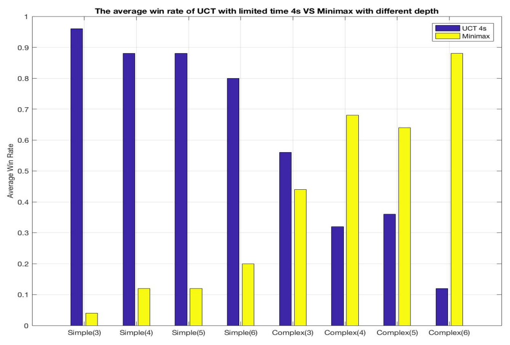

Fig. 13. Average win rate of UCT 4s VS Minimax with different depth

The average score of UCT with limited time 4s against Minimax with different evaluation and different limited depth agent is showed in Fig.14:

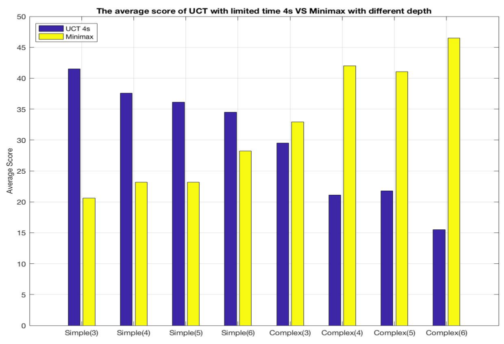

Fig. 14. Average score of UCT 4s VS Minimax with different depth

The average win rate of UCT with limited time 6s against Minimax with different evaluation and different limited depth agent is showed in Fig.15:

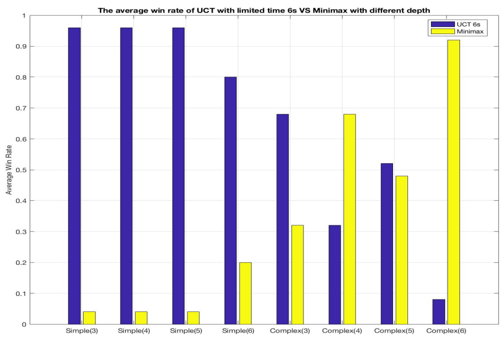

Fig. 15. Average win rate of UCT 6s VS Minimax with different depth

The average score of UCT with limited time 6s against Minimax with different evaluation and different limited depth agent is showed in Fig.16:

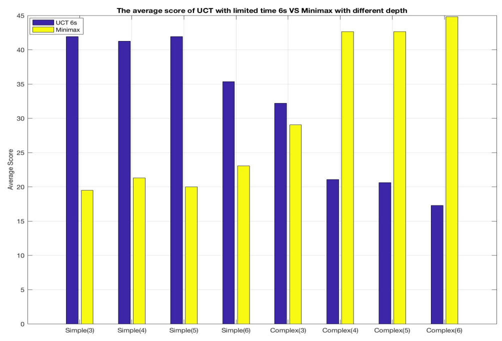

Fig. 16. Average score of UCT 6s VS Minimax with different depth

The average win rate of UCT with limited time 8s against Minimax with different evaluation and different limited depth agent is showed in Fig.17:

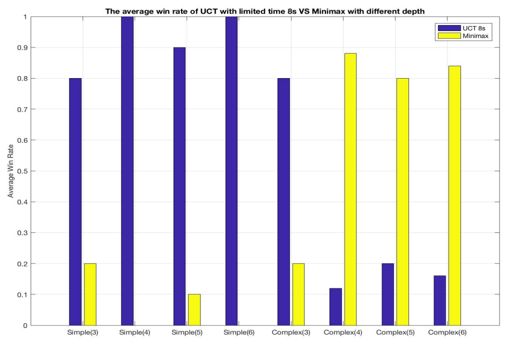

Fig. 17. Average win rate of UCT 8s VS Minimax with different depth

The average score of UCT with limited time 8s against Minimax with different evaluation and different limited depth agent is showed in Fig.18:

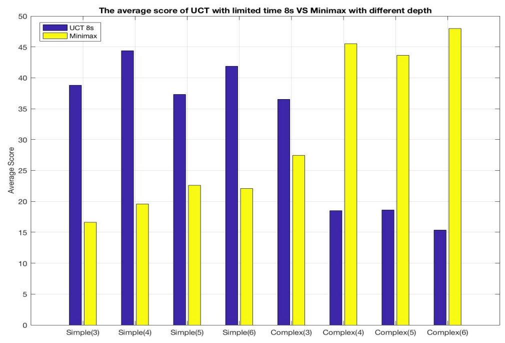

Fig. 18. Average score of UCT 8s VS Minimax with different depth

## V. CONCLUSION

In conclusion, we implemented UCT and Minimax search as agents for the game reversi and compared their performance under different limited time and depth, and we got following findings:

1) The performance of UCT is increasing with limited time increasing. This can be explained by the construction of UCT, with more time that UCT spent to construct itself, the tree is more complete which means it is more likely to find the win action. And the performance of Minimax is increasing with limited depth increasing. Because the more depth that Minimax can explore, the minimax value is more close to the real value. These results are all consistent with the theoretical predictions.
2) One interesting thing is that the increasing speed of UCT's performance is increasing first and then decreasing. The win rate increased 20% from limited time 0.01s to 0.05s, but only increased 2% from limited time 0.5s to 2s. And for Minimax, the biggest increasing happened from limited depth 1 to 2 for both simple and complex evaluation.
3) After this experiment, we found that the UCTs in this experiment have a better performance than the tested

simple Minimax. Even with the small limited time 2s, UCT has a better win rate than simplex agent with depth 6, although the average score of them are close.But for Minimax with complex evaluation, the average win rate is lower than UCT only when the limited depth is 3. For complex Minimax with depth 4, 5 and 6, they always have a better average win rate and score than UCT with limited time not bigger than 8 seconds.

4) For minimax, it seems will perform better than UCT with a good evaluation, because the game like Reversi, they has clearer strategy, like playing the corner and side as possible as it can, than the other game, like Go. Thus, a good evaluation can forecast better and faster than UCT, which using simulation to forecast the future.

There are also limitation of our project. First, we only tested UCT under limited time not bigger than 8 seconds. We used not bigger than 8 seconds as limited time because it cost to long to test. Even for UCT with limited time 8s against random agent, it spent almost 4 hours for testing 50 times. And it cost much longer when test it against Minimax with depth 6. Also because of the long test time, we only test 50 times for per group which is not enough, as in the results, UCT with limited time 2s has a bigger win rate than UCT with limited time 4s when against complex Minimax with depth 3. And after check the detailed data, we think this result is caused by the abnormal data and this effect can be reduced with more test times.

And there are several solutions to solve this problem, such as multi-process and using GPU instead of CPU. But because lack of related knowledge and time, we didn't do this. But this can be our future work.

Besides, for the UCT search, the performance is different with same limited time in different environment. Because its search depends on time, and for the same time, the computation speed is quite different of different CPU. We use 2.9 GHz Intel Core i7 on MacBook Pro 2017 to test it. There is another problem too. Although the testing hardware is same, the result also depend on the CPU is busy or not. We always test it in the midnight when the CPU is not busy, but sometimes the simulation times still was different.

## REFERENCES

[1] C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton, ”A survey of Monte Carlo tree search methods”, IEEE Trans. on Comput. Intel. and Al in Games, vol. 4, no. I, pp. 1-43, 2012.

[2] S.J.Russell & P.Norvig, Artificial Intelligence: a modern approach, 3rd edition.