CSC424: Intro to AI
Report of Project 1: Game Playing

Guofeng(Evan) Cui

September 30, 2018

Abstract

In the report, several searching algorithms will be introduced, including *minimax*, *alphabeta*, *H-minimax* and *Monte Carlo Search Tree*. Moreover such algorithms are applied in three kinds of games, which are simple tic-tac-toe(STTT), advance tic-tac-toe(ATTT) as well as ultimate tic-tac-toe(UTTT). During the experiment, it can be seen that *H-alphabeta* can solve all the three games but the performance will depend heavily on heuristic function. Besides, the time and space consuming of *Monte Carlo Search Tree* depend on the sampling size and the game, and they are more convenient to be controlled, especially in time-limit game. But its performance in game will be hard to predict, especially when the searching space is large.

1 Introduction

In the project, *minimax search tree* with *alphabeta pruning* is applied to play STTT. But problems appear when trying to utilize them in ATTT and UTTT. *H-minimax* and *Monte Carlo* are used instead.

Section 2 will discuss the algorithms and several comprehensions of them. The rules of games as well as application of searching trees will be introduced in section 3. Section 4 shows the structure of codes, several *Unified Modeling Language(UML) diagrams* will be displayed. Results and analysis will be illuminated in Section 5. And section 6 will be future works.

2 Algorithms

2.1 Minimax Search Tree

When dealing with two-player game problem, *Minimax Search Tree* is a considerable choice. Nodes in a same depth are set to be *MAX* or *MIN*, each of which refers to a different player. Nodes in *MAX* attempt to choose a node with highest score from its children, that is the best move in *MAX* player's opinion. On the opposite, *MIN* player tries to choose the node with lowest score from its children, which equals to minimum the probability that *MAX* player can win. In particular, if two players play the game in turn, *MAX* depth and *MIN* depth will also appear in the search tree in turn.

Minimax Search Tree makes use of depth-first search, that is it keeps searching until a leaf node is met. The searching process simulates moving of a player that will change the state of game, and each state that can express the result of game is set as leaf node. After reaching the leaf, parent will compare the results and choose the node they want. After root node chooses the move, that is gets its score, the search ends and such move will be the best move that root player can take.

2.2 Alphabeta Pruning

Actually *alphabeta pruning* is a special strategy of *minimax search tree*. With *alphabeta*, parent node takes the score from each of its children immediately when one of them finishes searching and gets it, instead of comparing all the scores of children and then taking the best score that used by traditional *minimax search tree*. For the reason that *alphabeta* strategy has a larger complexity, it can contribute more action, that is pruning.

Pruning is an useful strategy, that can reduce searching amount. Figure 1 shows a simple process of pruning. Each *MAX* depth has a value of *alpha*, which is always initialized to be negative infinity, and each *MIN* depth has a value of *beta*, which is always initialized to be positive infinity. When the search starts, the node reached in each depth are regarded as a sub-root, the value *alpha* and *beta* are treated as the value of related sub-root. Whenever child node finishes searching and sets its score to its related *alpha* or *beta*, parent would compare its value (*alpha* or *beta*) with child's. When the sub-root finishes searching, related *alpha* or *beta* will turn back to initial value. Pruning will be tested each time when child node updates its score. When parent *alpha* is larger than child's *beta*, or parent *beta* is smaller than child's *alpha*, the search of current child node becomes meaningless, and will be stopped. It is worth to mention that, currently search of current child node does not finish, so score of parent node will not update.

In figure 1(1), when the node in last depth sets its score 0 to *alpha2*, it finishes its search. Then its parent compares it with current *beta* and choose the smaller score, that is the 0. Then the root node compares *alpha1* with *beta*, and knows that *alpha1* is less than *beta*, so pruning will not happen. But current node in second depth has not finish searching, so *alpha1* would not update. At this point, current node of the last depth finishes its search, and the next search will turn out to be its brother. As a result, *alpha2* is initialized to be negative infinity.

In figure 1(2), the second node in the last depth sets score 1 to *alpha2*, but it is larger than the current *beta*. As a result, *beta* will stay 0. At this point, both the node in second depth and the node in third depth finish their search, so the *alpha1* gets the score from *beta*, that is 0, and the *beta* as well as *alpha2* are initialized again.

Then the searching tree comes to the second node in depth 2, as shown in figure1(3). When the leaf node finishes search and gets score -1, *beta* takes the score -1. Currently, value of *alpha1* is larger than that of *beta*, which means whatever current node in second depth will get in the future, the value *beta* will have no way larger than -1, and then the root node will never choose the score of such node. Then the searching of this node becomes meaningless, so such subtree is pruned.

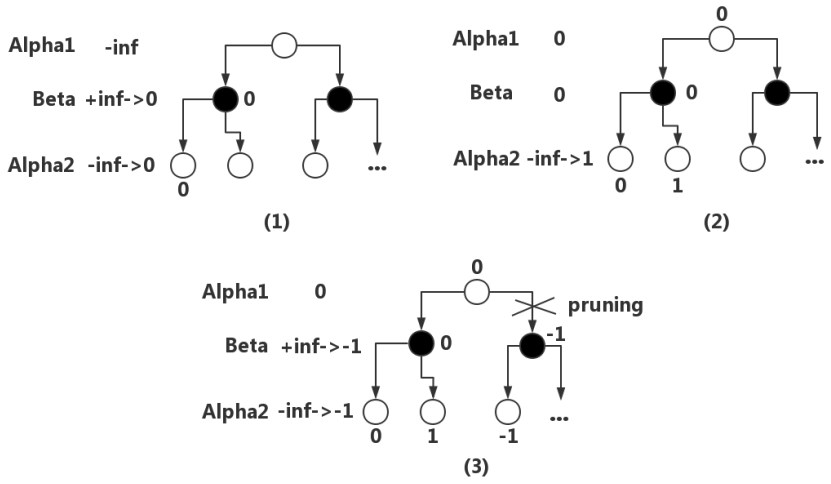

Figure 1: Alphabeta Pruning

2.3 H-minimax

When the searching space of game comes to be large, for example, the maximum search depth of ATTT is 81, that is totally 3^{81} situations has to be searched in the worst case, it is impossible to reach all the leaf in an acceptable time. But stopping the searching process will lead to uncompleted search, that is only part of results can be seen, and may cause the tree to select a bad move.

One way to solve the problem is to limit the searching depth and use heuristic results instead of the true game results. At this point, all the heuristic results that current root can reach are checked, and time consuming can be reduced. Heuristic result is calculated with heuristic function according to current game state. Heuristic function, also called expert function, are formulated by human, which is used to evaluate current game state and expected to predict the true result of game. For example, it is easy for human to evaluate that in X's opinion, the state of figure 2(1) is better that the state of figure 2(2). A related heuristic function will then be formulated to express this judgment.

However, the performance of *H-minimax searching tree* depends on heuristic function heavily, that is if the heuristic function predicts a wrong outcome of current game state, the player will take a wrong move. Moreover, it is hard to do time control with such search tree, for the reason that sometimes the nodes of depths are hard to be analyzed and users have no idea the depth they should limit.

2.4 Monte Carlo Search Tree

Monte Carlo Search Tree (MCT), as shown in its name, makes use of Monte Carlo sampling. There are four steps in *MCT*, selection, expand, simulation and backup, as shown in figure 3. There are two values for each node, that is *score* and *selected time*. *Score* refers to the general simulation result, for example, it

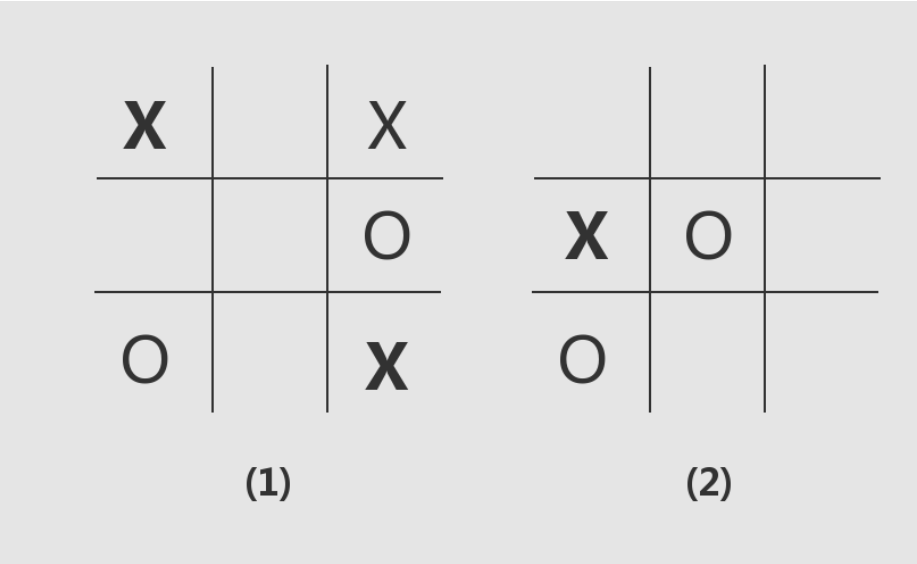

Figure 2: Simple Tic-Tac-Toe

can be set as the time that player wins. And *select time* refers to the total time of simulation for related node.

Selection process is shown in figure 3(1). There are two situations in this step, the figure only shows the first one. That is, if all the children of selected node have been simulated for at least one time, *MCT* will select the expected best node by some statistic function, such as *Upper Confidence Bound 1 applied to trees*, as shown in formulation 1. Where s_i refers to the score of node i, n_i refers to selected time of node i, $n_p i$ refers to the selected time of parent of node i and c is a constant number. Mathematically, the larger the score s_i, the larger the first formulation $\frac{s_i}{n_i}$, which means that if child node has larger score, it will have greater probability to be selected. And if a child node is chosen fewer among its brothers, the value of $\sqrt{\frac{\ln n_{pi}}{n_i}}$ will be larger, that it has greater probability to be chosen. As a result, the first formulation is used to express the performance of a child node, and the second formulation ensures that each child node has similar chance to be selected. In figure 3(1), the node with value 1/1 will be selected. The second situation of selection is that, if there are any children that hasn't been selected yet, *MCT* will select that child at first.

$$UCT = \frac{s_i}{n_i} + c\sqrt{\frac{\ln n_{pi}}{n_i}} \tag{1}$$

Expand process is shown in figure 3(2). After select a node that has not been expand, such as the node with value 1/1, children of such node are added to the tree, the value of which are set to be 0/0. And then, one of the children will be selected.

Simulation process is shown in figure 3(3). After the node with value 0/0 in the figure has been selected, *MCT* will simulate the game randomly to reach the leaf node, that is one result of the game. For STTT, *MCT* will randomly select position in the board to mark, until it wins, or losss, or draws. Randomly play-

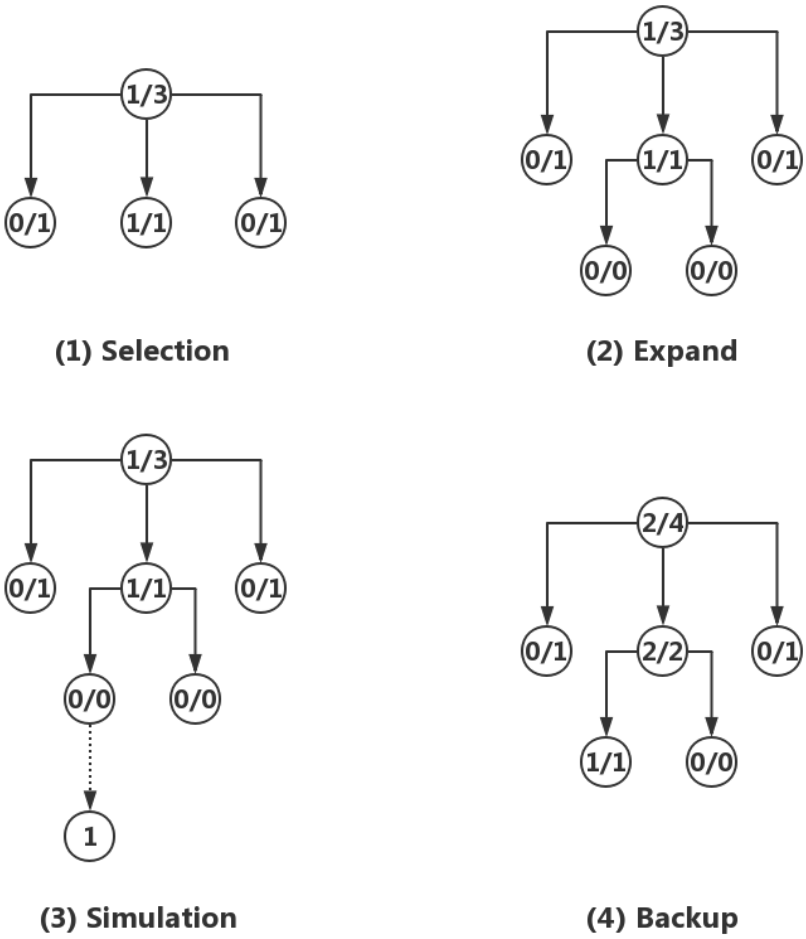

Figure 3: Monte Carlo Search Tree

ing can be time convenient compared with searching process. For this reason, simulation can be performed repeatedly for many times.

Backup process is shown in figure 3(4). Score 1 is gotten from simulation, then *selected time* values of nodes that have been visited are added with 1. And the *score* values of them are added with the score of leaf, that is 1.

For the reason that simulation process is time convenient, the children nodes of root can always be selected for at least one time and get score in a short time. So *MCT* is always safe to be stopped when running out of time. And the performance has little influence from user's heuristic function (the only influence is the value of constant c). If the times of simulation is large enough, the score of selected node can always predict the result of current game state correctly. However, when the searching space of game becomes very large, the required times for simulation becomes enormous. At this point, the score of nodes may be far from perfect.

3 Game Implementation

3.1 Simple Tic-Tac-Toe

STTT is a game that two players mark a position with a 3 times 3 board in turn (The board is shown in figure 2). When there exists a line, which can be row, column and diagonal, with only one kind of mark, the player of that mark wins. When the board is filled, the result will be draw.

The searching depth of such game is only 9, and the total amount of node needed to be selected is 9!, which is a small number for computer. Both *Minimax Searching Tree* and *Alphabeta Pruning* can have a satisfying performance. For the reason that *Alphabeta Pruning* is only a special strategy of *Minimax Search Tree*, *Alphabeta Pruning* is used to implement STTT.

3.2 Advanced Tic-Tac-Toe

Advanced tic-tac-toe is an advanced edition of STTT. 9 blocks are structured in a 3 times 3 board, and each block is an STTT board. The rule is that, when a player choose a position s in a block, the other player should choose a position in block s. If the block s is full, the player can go wherever he wants. When a player wins in a block, he wins the game.

ATTT ends up to be a large searching space. 3^{81} situations can appear regardless of the rule, and the searching space can be far more than that, because of the moving order. As a result, as time consuming of *Alphabeta Pruning* can be unacceptable, *MCT* and *H-minimax* are implemented to play the game.

MCT is simple, so it is more worth to introduce the heuristic function here. The strategy to play ATTT can be highly different compared with STTT which is to create a situation that at least two lines with two marks of the same kind appear. The strategy of ATTT is to occupy blocks. If there is a block s, in which at least one line with two marks of the same kind appear, the other player will no longer mark in position s of other blocks. For example, as shown in figure 4, two marks "O" appear in the third row of the center block. As a result, player with mark "X" can not mark position 5 again, or the third row of center block can be all three marks "O".

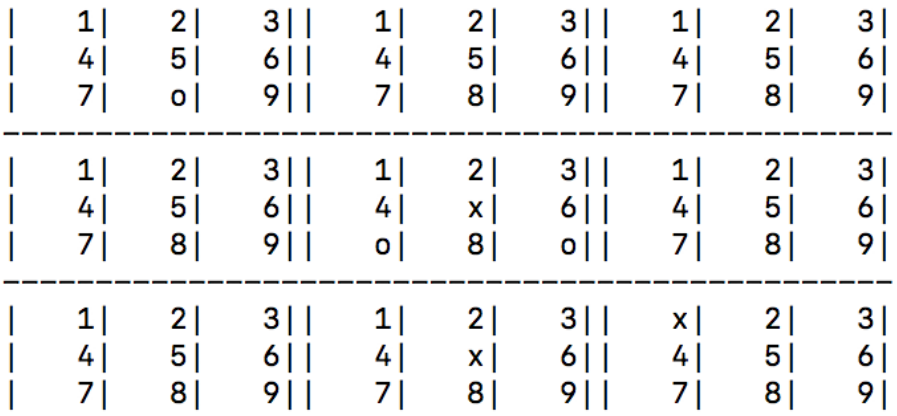

Figure 4: Example of ATTT

Base on such consideration, the first heuristic function is created. Two marks of same kind in a line of block s can ban opposite player to mark position s of other blocks, which is the largest threat. But if there are two lines in block s appear to be such situation, the threat will be similar to that of one line. Moreover, if only one mark in a line of block s appears, this can also be a threat as when the opposite player take position s in other blocks, such line can appear to be two marks of the same kind. It is worth to mention that, if a line appear to have different kinds of mark, that line will be meaningless. Pseudocode of such idea is shown in table 1.

However, heuristic function 1 is insufficient, as it only evaluates the board after current mark, without considering the danger of such mark. For example, a mark in a position s may contribute to a two-mark-in-line threat to opposite player, but there maybe the same threat from opposite player in block s. Marking in position s may lead to the winning of opposite player.So the heuristic function needs to consider a step deeper. The pesudocode of heuristic funtion 2 follows table 1 is shown in table 2. For the reason that the next mark of opposite player will only be a threat to current player, table 2 only considers the threat, that is decreasing the score. Moreover, for the reason that opposite player can choose whether to create a two-mark-in-line threat, only half of two-mark-in-line threat is calculated here.

3.3 Ultimate Tic-Tac-Toe

The board of UTTT is the same as ATTT, but condition to win is different. To win the game, a wining STTT in large board is needed, that is when a player wins a block, a mark will be marked in the large 3 times 3 board. When there is a line with three marks of the same kind appears in the large board, player of such mark wins. It is obvious that in general, UTTT has an even larger searching space than ATTT, although the amounts of possible situations of them are same.

Similarly, *MCT* and *H-minimax Searching Tree* are applied to UTTT. To formulate the heuristic function, the scores of each block are calculated the same as heursitic function 1(table 1), for the reason that it can express the threat of

Heuristic function 1 for ATTT

```
score ← 0
for block in board:
  twoMarkScore1, oneMarkScore1, twoMarkScore1, oneMarkScore2 ← 0

  for line in block:
    markNum1 = getMarkNum(line, player1)
    markNum2 = getMarkNum(line, player2)

  if markNum1 * markNum2 == 0
    if markNum1 == 1: oneMarkScore1 += score with one mark
    else if markNum1 == 2:
      if twoMarkScore1 > 0: oneMarkScore1 += score with one mark
      else: twoMarkScore1 += score with one mark
    else if markNum1 == 3: score += score with three marks

    if markNum2 == 1: oneMarkScore2 -= score with one mark
    else if markNum2 == 2
      if twoMarkScore2 < 0: oneMarkScore2 -= score with one mark
      else: twoMarkScore2 -= score with one mark
    if markNum2 == 3: score += score with three marks

  score += twoMarkScore1 + twoMarkScore2 + oneMarkScore1 + oneMarkScore2
```

Table 1: Heuristic function 1

continue pescudocode of Heuristic function 1

```
...
p = action taken of current mark
if player2 has two mark in a line in block p: score -= score with three marks
else if player2 has one mark in a line in block p:
  if player1 has two mark in a line in block p:
    if player2 can stop player1's threat and get a two mark in a line:
      score -= score with two mark
    else: score -= 0.5 * score with two mark
  else: score -= 0.5 * score with two mark
```

Table 2: Heuristic function 2

pescudocode of Heuristic function 3

```
score = 0 threats = getTheatOfBlocks() //similar as heuristic function 1
for line in largeBoard:
  subScore = 0    if(lineDraw(line)): break

  occupyBlockNum1 = number of block occupied by player 1.
  occupyBlockNum2 = number of block occupied by player 2.
  threatBlockNum1 = number of block occupied by player 1.
  threatBlockNum2 = number of block occupied by player 2.
  if(occupyBlockNum1 * occupyBlockNum2 != 0): break

  subScore = getScoreOccupy(occupyBlockNum1, occupyBlockNum2)
  if subScore != 0:
    subScore += getScoreThreat(threats, threatBlockNum1, threatBlockNum2)
  else:
    subScore += getScoreThreatWeight(threats, threatBlockNum1, threatBlockNum2)
  score += subScore
```

Table 3: Heuristic function 3

each block perfectly. In ATTT, scores of each block are simply added, as each block has the same significance. But when it comes to UTTT, the general threat should be related to the large board, that is STTT. As a result, situations below are considered:
(1) For each line of the large board, if all the three blocks have not been occupied, the threat of blocks are consider. For each player, the threat will be added and multiplied a constant number which is related to the amount of threat blocks come from that player. Then the threat score of the two players will be add.
(2) For each line of the large board, if there are some blocks that has been occupied, the threat of block will only be added without multiply the constant number.
(3) For each line of large board, if there is a block with the state of draw, that line will be meaningless with score 0.
The pesudocode of heuristric function 3 is shown in table 3.

4 Structure of Code

To apply searching trees to the three games, object-orients programming is used. Generally, the codes are divided into three parts, including *Game*, *UI* and *Stratagem*, as shown in the UML diagram in figure 5.

Game is use to implement each games, including the rule, result detection, get the next moves, etc.. Classes *simpleTTT*, *AdvanceTTT* and *UltimateTTT* are detailed rule of each game, which are children classes of Game.

TTTUI is to realize the UI of games, that is to show the board, get players' move, with *STTTUI*, *ATTTUI* and *UTTTUI* to deal with different games.

Stratagem is set to package differemt algorithms, such as *Alphabeta Prunbing*, MCT.

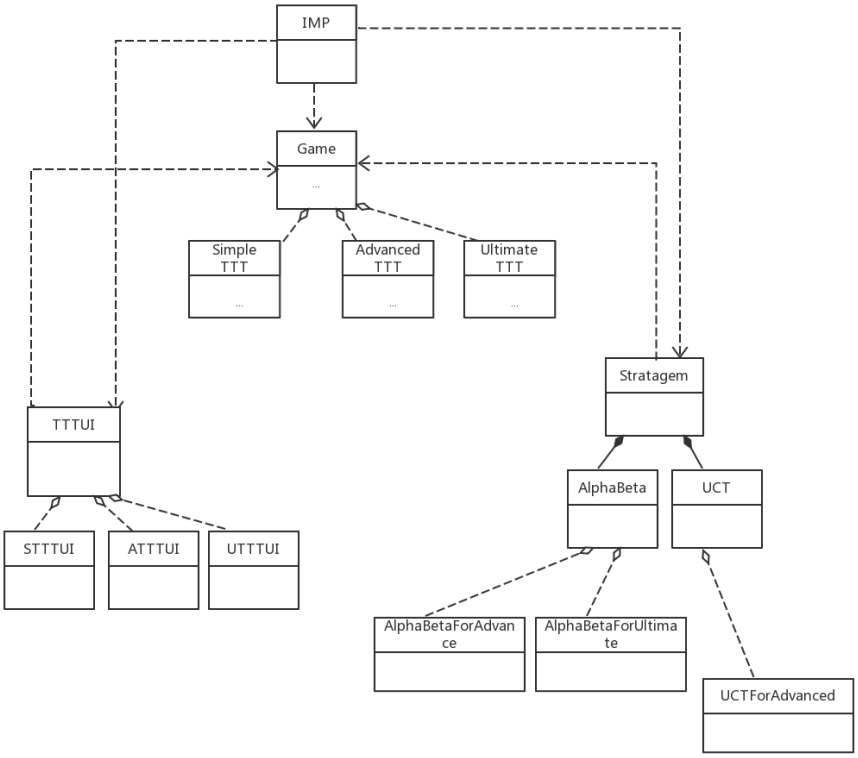

Figure 5: UML diagram

```
---------- Game start ----------
Please select your mark (x or o)(x will go first): o
|  1|  2|  3|
|  4|  5|  6|
|  7|  8|  9|

I am thinking...
Super Computer occpuy position 9
|  1|  2|  3|
|  4|  5|  6|
|  7|  8|  x|

ps. I took 0.016255 seconds to think
```

Figure 6: Experiment of STTT's worst case

And *IMP* is used to deal implement the game, create objects and call functions of classes.

5 Experiment and Analysis

5.1 Simple Tic-Tac-Toe

The searching space of STTT is small, so *Alphabeta Pruning* can handle perfectly. For the worst case, that is the computer takes the first turn, it takes 0.016 seconds to choose a position, as shown in figure 2. In practise, however, the first turn can be random.

Additionally, *Alphabeta* can always choose the best move dur to the small searching space. That is, during the experiment it never loss.

5.2 Advanced Tic-Tac-Toe

ATTT has a much larger searching space than *STTT*, so *H-minimax* and *MCT* are applied.

For *H-minimax Searching Tree*, with limit depth set to be 8, the computer will takes 9 seconds to select the first move, as shown in figure 7. During the experiment, around 10,000 turns are tested, and the average time to get a move is around 2.0 seconds. The reason that first move takes such a long time to select is that the amount of nodes in the first depth is 81, which is the worst case. In practise, such situation will only appear when at least one draw result appear in a block (Note that it can take even longer than 9 seconds in general in such situation, as several steps can choose whatever position they want). But such situation happens rarely, which means that game will end before a block is full.

Notice that two heuristic functions are shown in section 3.3, and manually analyzing, heuristic function 2 is better than heuristic function 3. But during the experiment, they appear similar performance. That is, the first turn will always wins. Such two heuristic functions are also applied to play with other

```
---------- Game start ----------
Please select your mark (x or o)(x will go first): o
|   1|   2|   3||   1|   2|   3||   1|   2|   3|
|   4|   5|   6||   4|   5|   6||   4|   5|   6|
|   7|   8|   9||   7|   8|   9||   7|   8|   9|
-----------------------------------------------
|   1|   2|   3||   1|   2|   3||   1|   2|   3|
|   4|   5|   6||   4|   5|   6||   4|   5|   6|
|   7|   8|   9||   7|   8|   9||   7|   8|   9|
-----------------------------------------------
|   1|   2|   3||   1|   2|   3||   1|   2|   3|
|   4|   5|   6||   4|   5|   6||   4|   5|   6|
|   7|   8|   9||   7|   8|   9||   7|   8|   9|

I am thinking...
Super Computer occpuy position in block 9 with position 5
|   1|   2|   3||   1|   2|   3||   1|   2|   3|
|   4|   5|   6||   4|   5|   6||   4|   5|   6|
|   7|   8|   9||   7|   8|   9||   7|   8|   9|
-----------------------------------------------
|   1|   2|   3||   1|   2|   3||   1|   2|   3|
|   4|   5|   6||   4|   5|   6||   4|   5|   6|
|   7|   8|   9||   7|   8|   9||   7|   8|   9|
-----------------------------------------------
|   1|   2|   3||   1|   2|   3||   1|   2|   3|
|   4|   5|   6||   4|   5|   6||   4|   x|   6|
|   7|   8|   9||   7|   8|   9||   7|   8|   9|

ps. I took 9.12563 seconds to think
```

Figure 7: Experiment of ATTT's first move by *H-minimax*

```
---------- Game start ----------
Please select your mark (x or o)(x will go first): o
|   1|   2|   3||   1|   2|   3||   1|   2|   3|
|   4|   5|   6||   4|   5|   6||   4|   5|   6|
|   7|   8|   9||   7|   8|   9||   7|   8|   9|
-----------------------------------------------
|   1|   2|   3||   1|   2|   3||   1|   2|   3|
|   4|   5|   6||   4|   5|   6||   4|   5|   6|
|   7|   8|   9||   7|   8|   9||   7|   8|   9|
-----------------------------------------------
|   1|   2|   3||   1|   2|   3||   1|   2|   3|
|   4|   5|   6||   4|   5|   6||   4|   5|   6|
|   7|   8|   9||   7|   8|   9||   7|   8|   9|

I am thinking...
100000
Super Computer occpuy position in block 8 with position 5
|   1|   2|   3||   1|   2|   3||   1|   2|   3|
|   4|   5|   6||   4|   5|   6||   4|   5|   6|
|   7|   8|   9||   7|   8|   9||   7|   8|   9|
-----------------------------------------------
|   1|   2|   3||   1|   2|   3||   1|   2|   3|
|   4|   5|   6||   4|   5|   6||   4|   5|   6|
|   7|   8|   9||   7|   8|   9||   7|   8|   9|
-----------------------------------------------
|   1|   2|   3||   1|   2|   3||   1|   2|   3|
|   4|   5|   6||   4|   x|   6||   4|   5|   6|
|   7|   8|   9||   7|   8|   9||   7|   8|   9|

ps. I took 2.23631 seconds to think
```

Figure 8: Experiment of ATTT's first move by *MCT*

kinds of heuristic functions, and it also turns out to be that generally first turn will win. This makes sense, as first turn player has a better chances to occupy more block with two marks.

For *MCT*, with limit sampling amount to be 10,000, it takes 2.2 seconds to select the first move, as shown in figure 8. This is no doubt the time consuming of the worst case. For the reason that the time consuming of *MCT* is $O(nd)$, where n is sampling times and d is the longest depth. As game implementing, the longest depth will decrease, but the sampling times will stay constant. So the time *MCT* takes to select a move will decrease.

However, the performance of *MCT* is hard to predict. During the experiment that *MCT* play against the *H-minimax Searching Tree*, it appears that *MCT* can win the game even when *H-minimax Searching Tree* takes the first turn. But this would not always happened. The reason that *H-minimax Searching Tree* is loss is that the heuristic function is not perfect, and *MCT* takes a better move. And the reason that the game can have different results is that, 10,000 sampling times may not be enough compared with the searching space of ATTT.

Unfortunately, I don't have enough time to test the result with different 10,000 or calcualte the probability that *MCT* can win.

5.3 Ultimate Tic-Tac-Toe

Similar to ATTT, *H-minimax Search tree* and *MCT* are applied to implement the game.

As shown in figure 9, with depth limited to be 8, it takes 3.15 to take the first turn. It is a surprise that it runs faster than ATTT's first step, the reason may be that *MCT* have to handle one more array when implementing *ATTT*. Moreover, may be the pruning is more useful in *UTTT* than in *ATTT*. But as the game implementing, after one block gets a result, the time consuming will be much larger, which can sometimes be more than 10 seconds, for the reason that each time when opposite player marks a related position, the children nodes can reach any position of the board. One more surprise is that, with depth limited to be 8, the result that *H-minimam Search Tree* play against itself would be draw; But when the depth is limited to be 10, the first turn player will always win.

Limiting sampling times to be 10,000, *MCT* will take around 6 seconds to get the first move, as shown in figure 10. One more time, *MCT* sometimes wins the *H-minimax Search Tree* and sometimes losses. Actually, there are some thoughts to improve the performance of *MCT*, which will be shown in section 6 future work.

6 Future Work

Although the games can be handle with *alphabeta*, *H-minimax* and *MCT* as shown above, more improvement can be make. Below are three thoughts that come to my mind, but I don't have time to realize them.
(1)Improve the heuristic function of *UTTT*. The heuristic function 3(table 3) has the same problem of heuristic function 1(table 1), that it only considers current board. More variables can be added to consider the steps deeper.
(2)Improve *MCT*. During the experiment, it comes out that the score of each node gotten by $MonteCarloSampling$ is not stable, which may far from expected value. There may be several reasons, including not enough sampling times compared with large searching space, the result of randomly simulation depends highly on sampling times, etc.. Two thoughts to do the improvement is to combine heuristic function with *MCT* to evaluate a node together to reduce the shortcome of not enough sampling times, or use heuristic function instead of randomly simulation so that the simulation will get to more useful results.
(3)Simulated Annealing to optimize constant value. Heuristic functions above make use several constant number, which may not be an optimal ones. Simulated annealing can be applied to get a better constant number.

```
----------- Game start -----------
Please select your mark (x or o)(x will go first): o
|   1|   2|   3||   1|   2|   3||   1|   2|   3|
|   4|   5|   6||   4|   5|   6||   4|   5|   6|
|   7|   8|   9||   7|   8|   9||   7|   8|   9|
-----------------------------------------------
|   1|   2|   3||   1|   2|   3||   1|   2|   3|
|   4|   5|   6||   4|   5|   6||   4|   5|   6|
|   7|   8|   9||   7|   8|   9||   7|   8|   9|
-----------------------------------------------
|   1|   2|   3||   1|   2|   3||   1|   2|   3|
|   4|   5|   6||   4|   5|   6||   4|   5|   6|
|   7|   8|   9||   7|   8|   9||   7|   8|   9|

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

I am thinking...
Super Computer occpuy position in block 5 with position 5
|   1|   2|   3||   1|   2|   3||   1|   2|   3|
|   4|   5|   6||   4|   5|   6||   4|   5|   6|
|   7|   8|   9||   7|   8|   9||   7|   8|   9|
-----------------------------------------------
|   1|   2|   3||   1|   2|   3||   1|   2|   3|
|   4|   5|   6||   4|   x|   6||   4|   5|   6|
|   7|   8|   9||   7|   8|   9||   7|   8|   9|
-----------------------------------------------
|   1|   2|   3||   1|   2|   3||   1|   2|   3|
|   4|   5|   6||   4|   5|   6||   4|   5|   6|
|   7|   8|   9||   7|   8|   9||   7|   8|   9|

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

ps. I took 3.15213 seconds to think
```

Figure 9: Experiment of UTTT's first move by *H-minimax*

```
---------- Game start ----------
Please select your mark (x or o)(x will go first): o
|   1|   2|   3||   1|   2|   3||   1|   2|   3|
|   4|   5|   6||   4|   5|   6||   4|   5|   6|
|   7|   8|   9||   7|   8|   9||   7|   8|   9|
-----------------------------------------------
|   1|   2|   3||   1|   2|   3||   1|   2|   3|
|   4|   5|   6||   4|   5|   6||   4|   5|   6|
|   7|   8|   9||   7|   8|   9||   7|   8|   9|
-----------------------------------------------
|   1|   2|   3||   1|   2|   3||   1|   2|   3|
|   4|   5|   6||   4|   5|   6||   4|   5|   6|
|   7|   8|   9||   7|   8|   9||   7|   8|   9|

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

I am thinking...
100000
Super Computer occpuy position in block 5 with position 7
|   1|   2|   3||   1|   2|   3||   1|   2|   3|
|   4|   5|   6||   4|   5|   6||   4|   5|   6|
|   7|   8|   9||   7|   8|   9||   7|   8|   9|
-----------------------------------------------
|   1|   2|   3||   1|   2|   3||   1|   2|   3|
|   4|   5|   6||   4|   5|   6||   4|   5|   6|
|   7|   8|   9||   x|   8|   9||   7|   8|   9|
-----------------------------------------------
|   1|   2|   3||   1|   2|   3||   1|   2|   3|
|   4|   5|   6||   4|   5|   6||   4|   5|   6|
|   7|   8|   9||   7|   8|   9||   7|   8|   9|

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

ps. I took 6.38265 seconds to think
```

Figure 10: Experiment of UTTT's first move by *MCT*