

COMP 6721 Project 1 Report

Zhongxu Huang¹ and Yixuan Li²

¹ 40052560 realdonald9@gmail.com

² 40079830 liyixuan4030614@gmail.com

1 Introduction to technical details

1.1 Minimax

Minimax is a state space search approach for game playing. Normally, it is used in zero-sum games, meaning gains and losses of all players sum up to 0 at each step[1]. There are 2 (or 3) attributes in Minimax: state space search tree T , heuristic function $e(n)$, (and maximum search depth in n -ply minimax).

In minimax, each node only contains one value, that is heuristic value. At Max layer, each node will choose the maximum heuristic value of its children; At Min layer, each node will choose the minimum heuristic value of its children. In the end, the root will choose the node based on the values.

However, the minimax algorithm is impractical for a complex problem in real life, because it completely analyzes each node in the game tree. Without affecting the result, the performance can be optimized by alpha-beta pruning.[1]

1.2 Alpha-beta pruning

Alpha-beta pruning is a search algorithm that decreases the number of nodes that are evaluated by the minimax algorithm in state space search tree[2]. Because the number of nodes in state space for zero-sum games will increase exponentially as depth increases, this reason restricts minimax algorithm to look ahead and choose a better move for the root.

In alpha-beta pruning, each node contains two values, alpha and beta. alpha means the lower-bound value, and beta means the upper-bound value. For Max player, it will prune the branch if alpha value \geq beta value of its children; For Min player, it will prune the branch if beta value \leq alpha value of its children. This allows alpha-beta pruning is not going to evaluate the complete game tree by ignoring branches that cannot be beneficial to evaluate the root node.

1.3 Critical point theory

In our project, we also need to consider critical points. As we have known, the double-card game has two sections. Regular section and recycling section. In the regular section, before the 24th step, when AI uses minimax or alpha-beta pruning to select a position, it simulates the human player's selected position with a regular move. However, in the 24th step, when AI do that, it needs to simulate the human player's selected position with recycling move. This is the critical point problem.

Same as above, in the recycling section, before the 60th step, when AI uses minimax or alpha-beta pruning to select a position, the depth is a default value. However, in the 60th step, when AI does that, it needs to set depth to 1. This is because the 60th step is the last step, AI does not need to simulate the human player's selected position anymore.

2 Description and justification of heuristics

2.1 Heuristic function

The heuristic evaluation function is an important part of the AI in the game of the game, and it determines the decision-making steps of the AI. The main task is to evaluate the importance of each node in the state space tree.

In general, the heuristic function evaluates the current form of the game for the AI. It simulates the form of the game after each step of decision for AI and makes the most favorable step.

The most important step in the design of the heuristic function is also a very complicated part, which is like a process of building a mathematical model. Designing the evaluation function for the game Double Card requires a good understanding of the game and some knowledge about specific procedures to derive heuristic.[3]








2.2 Heuristic 1





Before starting describing our first heuristic function, we will make some assumption signs for clarification:

1. O means occupied point by one player.
 2. @ means the other side or boundary (for example, if O means solid point, then @ means a hollow point or boundary).
 3. X means an empty position.
- When there is a “4-connected” on the board, meaning “OOOO”, this situation indicates one of the players has won the game. So, we rate the heuristic value of this situation as 100,000.
 - When there is an “Alive-3”, meaning “XOOOX”, this situation indicates one of the players has a three-segment connection which is not blocked by another player or the boundary. So, we consider this state most likely leads to winning, and we rate the heuristic value of this situation as 10,000.
 - When there is a “Dead-3”, such as “@OOOX”, “@OOXO”, “@OXOO” and so on, this situation indicates one of the players has a three-segment connection which is blocked on one side but is alive on another side. So, we consider this state is advantageous for the current player, and we rate the heuristic value of this situation as 1,000.

- When there is a “Double-Dead-3”, meaning that one side has more than 1 “Dead-3”, then we consider this situation most likely leads to winning, so we rate the heuristic value of this situation as 100,000.
- When there is an “Alive-2”, such as “XOOXX”, “XOXOX”, “XXOOX” etc., this situation indicates one of the players has a two-segment connection which is not blocked on both sides. Because an “Alive-2” can form an “Alive-3” easily, we rate the heuristic value of this situation as same as “Dead-3” which is 1,000.
- When there is a “Dead-2”, such as “@OXOX” etc., this situation indicates one of the players has a two-segment connection which is blocked on one side but is alive on another side. We rate the heuristic value of this situation as 500.
- When there is a “Double-Alive-2”, meaning that one side has more than 1 “Alive-2”, we rate the heuristic value of this situation as 100,000.
- When there is a “Dead-3-and-Alive-2”, meaning that one side has a “Dead-3” and an “Alive-2”, we rate the heuristic value of this situation as 100,000.
- When there is an “Unavailable-3”, meaning “@OOO@”, this situation indicates it is impossible to form a 4-connected segment, so we rate the heuristic value of this situation as -100.
- When there is an “Unavailable-2”, meaning “@OO@”, we rate the heuristic value of this situation as -100.
- When there is an “Unavailable-1”, meaning “@O@”, we rate the heuristic value of this situation as -100.

Table 1. Heuristic value table of heuristic 1

Situation	Diagram (takes ‘color’ as an example)	Heuristic value
“4-connected”		100,000
“Alive-3”		10,000
“Dead-3”		1,000
“Double-Dead-3”		100,000
“Alive-2”		1,000
“Dead-2”		500
“Double-Alive-2”		100,000

“Dead-3-and-Alive-2”		100,000
“Unavailable-3”		-100
“Unavailable-2”		-100
“Unavailable-1”		-100

These are the weight ratios of the evaluation factors we used in heuristic 1. In our program, we have a function called *evaluation* which is a method that computes scores for all the pieces on the board and gives a final score according to the current situation.

Processes in *evaluate* function:

- 1) We named ScoreAI as the sum of unilateral scores for AI player, and *ScoreHuman* as the sum of scores for the human player.
- 2) Then turn the 2-dimensional board into N 1-dimensional arrays in four directions (row, column, positive diagonal, negative diagonal).
- 3) Perform score calculation on every 5 consecutive segments in all 1-dimensional arrays for ScoreAI and ScoreHuman.
- 4) The *evaluation* function return ScoreAI – ScoreHuman, which is the total score for the current situation.

2.3 Heuristic 2[6]

Heuristic 2 is implemented in the function called *evaluation_2*, and this heuristic is a more naïve heuristic than heuristic 1 because instead of evaluating a node by checking “dead” or “alive” same consecutive segments, heuristic 2 only evaluates a node by the occurrence times in every 4 consecutive segments. The detailed process describes below:

- 1) We named danger_color, good_color, danger_dot, good_dot as the sum of the dangerous factor and goodness factor for color and dot.
- 2) Then, as what we did in heuristic 1, we turn the 2-dimensional board into N 1-dimensional arrays in four directions (row, column, positive diagonal, negative diagonal)
- 3) Perform score calculation on all 1-dimensional arrays
 - a. Dangerous factor calculation
In four consecutive segments, if there are more red color or white color segments in the particular line, then the line is potentially dangerous for dot player, and verse visa.

While iterating every four consecutive segments in each row, each column, each positive and negative diagonal, we record the occurrence times of “red”, “white”, “solid”, “hollow”. And dangerous calculation follows this:

$$maxAllowPerc = \frac{Number\ of\ connected\ segments - 1}{Number\ of\ connected\ segments}$$

(Number of connected segments means how many consecutive segments we check each time, in our program, it is equal to 4.)

$$PercWon = \frac{Occurrence\ times}{Number\ of\ connected\ segments}$$

(e.g. for “red”, Occurrence time means how many times does “red” appear in every check.)

$$dangerous = PercWon * \frac{DANGER\ FACTOR}{maxAllowPerc}$$

(we set DANGER FACTOR to 100)

The value of dangerous is the evaluation value for one non-empty position on the board, and we sum dangerous values of all non-empty position to danger_dot and danger_color variables.

b. Goodness factor calculation

Similar to Dangerous evaluation, in four consecutive segments, if there are more red color or white color segments in the particular line, then the line is considered good for the color player. And goodness evaluation follows this:

$$goodFac = \frac{GOOD\ FACTOR}{(Number\ of\ connected\ segment - 1)^2}$$

(we set GOOD FACTOR to 50 since we consider heuristic 2 is a more defensive heuristic even though offensive evaluation may lead to better results.)

$$goodness = Occurrence\ times^2 * goodFac$$

The value of goodness is the evaluation value for one non-empty position on the board, and we sum goodness values of all non-empty positions to good_dot and good_color variables.

- 4) In the end, we calculate ScoreAI and ScoreHuman based on the four values, danger_color, danger_dot, good_color, good_dot, and we give a weight to each value while doing the calculation, then return ScoreAI – ScoreHuman.

3 Analysis of heuristics and results

3.1 Analysis of heuristic 1

Under this heuristic approach, AI becomes smarter. Because when evaluating the overall score of the board, the more the winning situations and losing situations are considered, the difference between the scores produced by different positions becomes more and more obvious. This phenomenon can be found in the following example.

Table 2. An example of heuristic 1

Step	AI moves	Human moves
1	0 2 A 1	
2		0 6 B 1
3	0 1 A 3	
4		0 2 C 1
5	0 4 D 1	

Scenario	Evaluations
['', '', '', '', '', '', '', '']	[-198200, 8, 0, '1']
['', '', '', '', '', '', '', '']	[-99300, 11, 3, '1']
['', '', '', '', '', '', '', '']	[-198300, 11, 5, '1']
['', '', '', '', '', '', '', '']	[-199600, 8, 0, '2']
['', '', '', '', '', '', '', '']	[-100400, 8, 1, '2']
['', '', '', '', '', '', '', '']	[-100400, 9, 2, '2']
['', '', '', '', '', '', '', '']	[-198200, 11, 3, '2']
['', '', '', '', '', '', '', '']	[-199300, 11, 5, '2']
['', '', '', '', '', '', '', '']	[-199300, 11, 7, '2']
['RX', 'W0', '', '', '', '', '', '']	[-200600, 8, 0, '3']
['RX', 'R0', 'RX', '', '', '', '', '']	[-199300, 11, 3, '3']
['W0', 'WX', 'W0', '', '', '', '', '']	[-299300, 11, 4, '3']
Overall steps: 4	[-198800, 11, 6, '3']
	[-199600, 8, 0, '4']
	[-200100, 8, 1, '4']
	[-199100, 9, 2, '4']
	[-300, 11, 3, '4']
	[-197300, 11, 5, '4']
	[-199300, 11, 7, '4']
	[-98800, 8, 0, '5']
	[-99300, 11, 3, '5']
	[-198300, 11, 5, '5']
	[-199600, 8, 0, '6']
	[-100400, 8, 1, '6']
	[-100400, 9, 2, '6']
	[-198200, 11, 3, '6']
	[-199300, 11, 5, '6']
	[-199300, 11, 7, '6']
	[-200600, 8, 0, '7']
	[-98200, 11, 3, '7']
	[-299300, 11, 4, '7']
	[-198800, 11, 6, '7']
	[-199600, 8, 0, '8']
	[-200100, 8, 1, '8']
	[-199100, 9, 2, '8']
	[-300, 11, 3, '8']
	[-197300, 11, 5, '8']
	[-199300, 11, 7, '8']

Assuming human plays color and AI plays dot. Then after step 4, we can clearly see that AI should choose a position that will not let humans win, such as (D,1). With the trace file, we found that under this heuristic method, for all the values of level 2 nodes, the system should select the maximum value of all nodes. Therefore, it may select the value of -300, the corresponding position is (D,1) of the 4th card, or select value is -300, and the corresponding position is (D,1) of the 8th card.

Here is the move that AI chooses. Through the terminal display, the system selects the 4th card with position (D,1). Successfully prevent humans to win.

```
[ '□', '□', '□', '□', '□', '□', '□', '□']
[ '□', '□', '□', '□', '□', '□', '□', '□']
[ '□', '□', '□', '□', '□', '□', '□', '□']
[ '□', '□', '□', '□', '□', '□', '□', '□']
[ '□', '□', '□', '□', '□', '□', '□', '□']
[ '□', '□', '□', '□', '□', '□', '□', '□']
[ '□', '□', '□', '□', '□', '□', '□', '□']
[ '□', '□', '□', '□', '□', '□', '□', '□']
[ '□', '□', '□', '□', '□', '□', '□', '□']
[ 'RX', 'W0', '□', '□', '□', '□', '□', '□']
[ 'RX', 'R0', 'RX', 'W0', '□', '□', '□', '□']
[ 'W0', 'WX', 'W0', 'RX', '□', '□', '□', '□']
Overall steps: 5
```

Time analysis:

We analyze heuristic 1 with heuristic computing time. We consider heuristic computing time has a relationship with the total amount of cards on the board. Due to our program calls the heuristic function in AI's turn, so we draw the diagram to show the relationship between Heuristic time and Number of cards that belong to AI player.

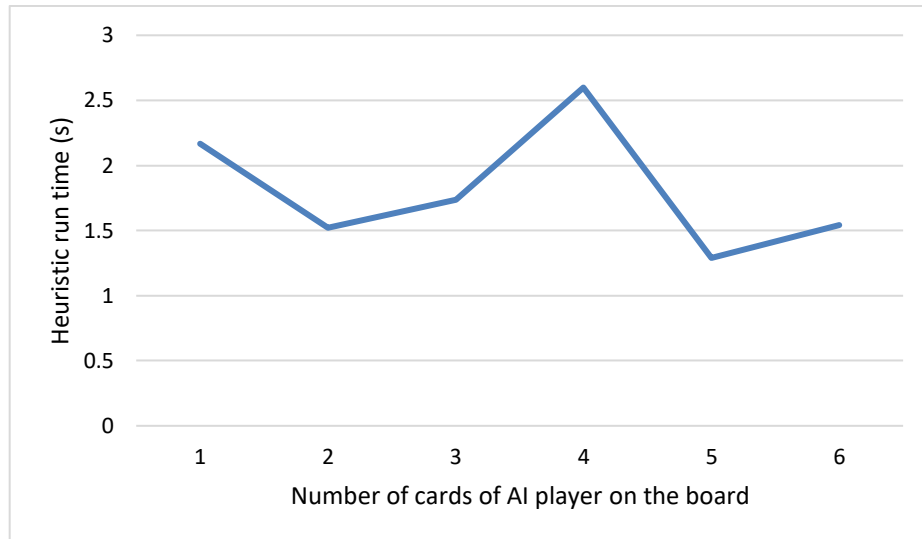


Figure 1. shows the relationship of heuristic 1 computing time and the number of AI cards on the board.

- 1) We use history table (described in 4.1) to avoid re-evaluate positions where the heuristic value is not changed.
- 2) As the number of cards on the board increase, the board will not be “flat” as the very beginning, which means there will be a fewer valid position, so the amount of evaluation call will be fewer depending on the flatness of the board.

Heuristic 2 can perform basic defensive moves to prevent the human player to win and drop a piece that has an advantage to AI player. We shall analyze heuristic 2 using a similar approach in analyzing heuristic 1 by a real scenario.

Step	AI moves	Human moves
1	0 2 A 1	
2		0 8 B 1
3	0 6 E 1	
4		0 2 C 1
5	0 6 D 1	

8

With the trace file, we found that under this heuristic function, for all the evaluation values of level 2 nodes, AI should select the maximum value of all nodes, which is -4633.333, corresponding the move (0 6 D 1).

Here is the move that AI chooses:

```
[ '□', '□', '□', '□', '□', '□', '□', '□']
[ '□', '□', '□', '□', '□', '□', '□', '□']
[ '□', '□', '□', '□', '□', '□', '□', '□']
[ '□', '□', '□', '□', '□', '□', '□', '□']
[ '□', '□', '□', '□', '□', '□', '□', '□']
[ '□', '□', '□', '□', '□', '□', '□', '□']
[ '□', '□', '□', '□', '□', '□', '□', '□']
[ '□', '□', '□', '□', '□', '□', '□', '□']
[ '□', '□', '□', '□', '□', '□', '□', '□']
[ '□', '□', '□', '□', '□', '□', '□', '□']
[ 'RX', 'WX', 'RX', 'R0', 'R0', '□', '□', '□']
[ 'W0', 'R0', 'W0', 'WX', 'WX', '□', '□', '□']
Overall steps: 5
```

Time analysis:

Next, we analyze the heuristic 2 with heuristic computing time. Because heuristic 2 is simpler than heuristic 1, so in practice, the run time of heuristic 2 should be shorter than run time of heuristic 1.

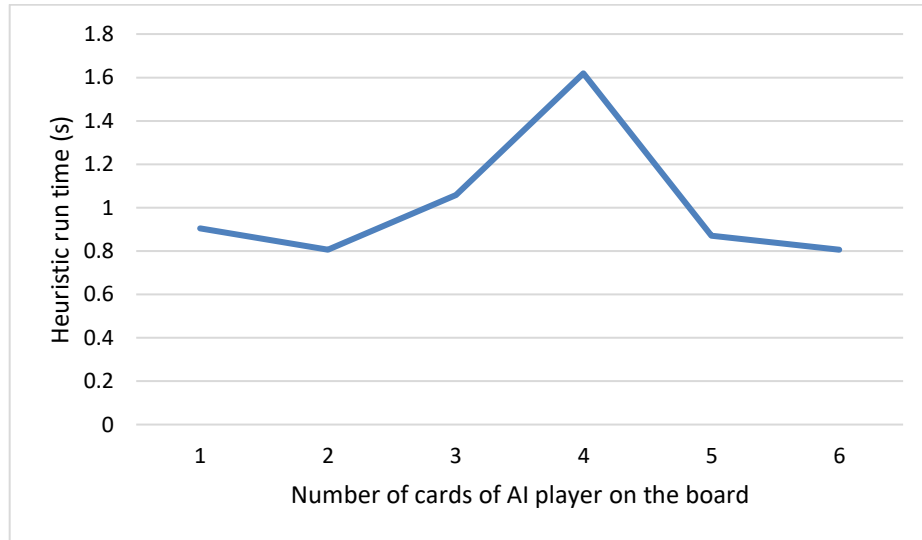


Figure 2. shows the relationship of heuristic 2 run time and number of AI cards on the board.

Similar to time analysis in heuristic 1, the heuristic computing time does not always increase as the number of AI cards increase. The reason for this phenomena is we set a threshold (0.3) in *evaluation_2* function. The threshold guarantees that only the

“red”, “white”, “hollow”, “solid” segment with the percentage of occurrence time of which is more than 0.3 is added to the final good or danger value.

3.3 Comparison of two heuristics

We compare two heuristics in 2 attributes:

- 1) Time used
- 2) Intelligence

For the first attribute, we have analyzed the computing time of heuristic 1 and heuristic 2 in section 3.1 and 3.2. Since heuristic 1 considers more situations than heuristic 2, so the average time used of heuristic 1 is higher than the computing time of heuristic 2 on average.

For the second attribute, we use a real scenario to show which heuristic is more informed than another.

Assume AI player plays color and the human player plays dot. The following moves are performed using heuristic 1:

Table 4. A comparison example of heuristic 1

Step	AI moves	Human moves
1	0 2 A 1	
2		0 8 B 1
3	0 1 A 3	
4		0 2 C 1
5	0 6 D 1	
6		0 4 E 1
7	0 6 D 3	
8		0 6 F 1
9	0 8 G 1	

[illegible]

Assume AI player plays color and the human player plays dot. The following moves are performed using heuristic 2:

Table 5. A comparison example of heuristic 2

Step	AI moves	Human moves
1	0 2 A 1	
2		0 8 B 1
3	0 4 D 1	
4		0 2 C 1
5	0 4 E 1	
6		0 6 F 1
7	0 8 D 3	

[illegible]

As we see in the comparison example above, the card distribution on the board at step 1 is almost the same. The human player takes the move (0 6 F 1) and forms 3 consecutive same segments from (D 2) to (F 2). When we use heuristic 1, AI player will select (0 8 G 1) to prevent human to win, while AI player under heuristic 2 will select (0 8 D 3) which will lead the human player to win in the next turn.

The reason for the bad moves under heuristic 2 may be resulted in the different weights on the four variable, “good_color”, “good_dot”, “danger_color”, “danger_dot”. When computing (ScoreAI - ScoreHuman) at the end, the 2 values would be different and ends with the bad move.

3.4 Alpha-beta pruning effectiveness analysis

Due to the depth of search and the possible situations, the efficiency of the minimax algorithm is not ideal. In fact, it is not necessary for each node to search, and some things are not necessary. The alpha-beta pruning algorithm is just to solve this problem.

The advantage of alpha-beta pruning is to reduce the branches in the search tree and use the search time in a "more promising" subtree, which in turn increases the depth of the search. This algorithm, like the minimax algorithm, is a branch and bound algorithm. If the node search order is optimized or approximated (the best

choice is ranked first in each node), the search depth can be more than twice as large as the minimax algorithm in the same time.

We compare and contrast minimax algorithm and alpha-beta pruning algorithm by showing how many nodes the algorithm evaluates and computing time of two algorithms. Here are the comparison graphics below:

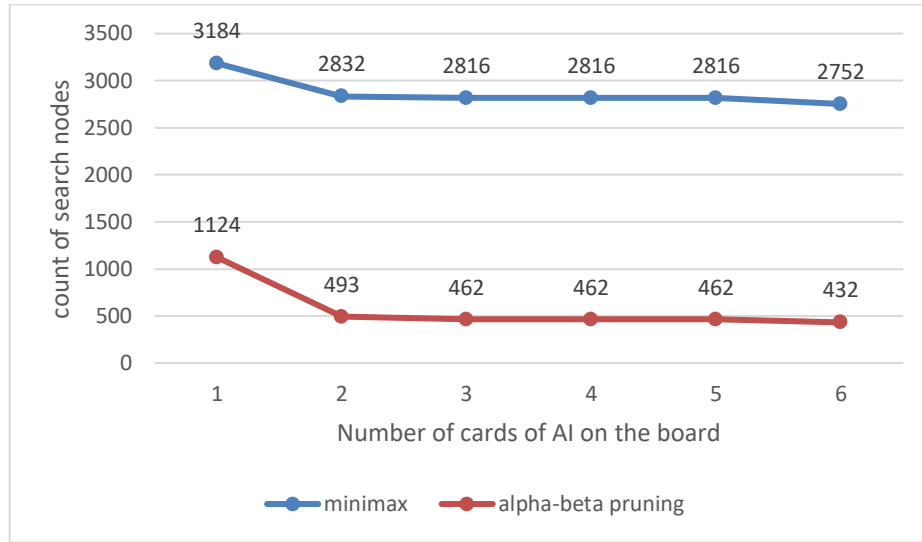


Figure 3. shows the relationship between the number of search nodes and the number of AI cards on the board using minimax and alpha-beta pruning.

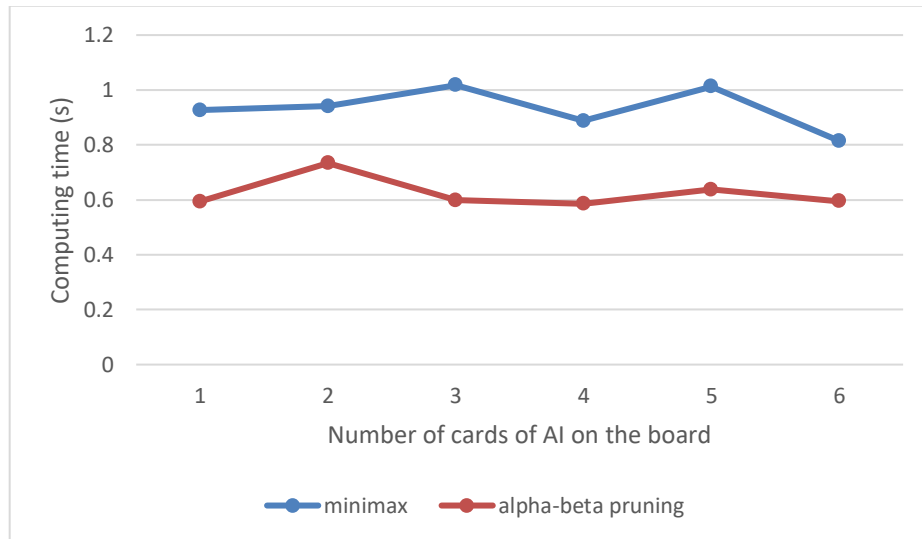


Figure 4. shows the relationship between computing time under the same heuristic and the number of AI cards on the board in minimax and alpha-beta pruning.

As we see in the first figure, after the second move of AI player, the amount of nodes searched by alpha-beta pruning algorithm stays stable at 400-level, which is almost 1/7 of minimax algorithm. And alpha-beta pruning reduces 1/3 of time on average used in minimax algorithm as shown in the second figure.

In this project, it is impractical to traverse the whole tree even with efficient alpha-beta pruning, Because the state space grows exponentially as the depth increases. So we decide to stop looking further ahead at depth 3 and evaluate the current bottom leaves. We reckon that a search algorithm should put more effort on the heuristic value and improve the accuracy of evaluation instead of going deeper by sacrificing the quality of heuristic function.

4 Description of encountered difficulties and solutions

4.1 Difficulty 1: Time complexity of heuristic function

In the heuristic function, the more winning and failing conditions are considered, the more computing time and space it consumes and the lower the CPU utilization is. In minimax approach, the algorithm searches can be approximated to m^n where m means the number of branches from root to its children, n means the number of branches from the children of root to the leaves in the search tree. Although performance is improved if we switch to alpha-beta pruning, the value of m still cannot be greatly reduced.

Solution 1: Discard isolated cards. (implemented)

While searching on the board, we discard isolated points and only search for the position which has other segments in the surrounding area. This avoids searching for obviously useless nodes and greatly improve the overall search speed.

Solution 2: Stop evaluating when 100% win or lose (implemented)

When there is a losing or a winning game in the search process, it will return the evaluation value directly and no longer keep searching.

Solution 3: Use history table (implemented)

In each iteration of alpha-beta pruning function, the pieces on the board only increase by 1, and the scores in most positions of the board are not changed, so it is not required to scan the entire board for each iteration during evaluation. Instead, we can save the score of the board in the previous iteration. Before each iteration, we scan the history table at the very beginning, if one of the positions in history table is the same as the new position in the current iteration, we won't re-evaluate the position. This eliminates the need to rescan the entire board every time and increases efficiency.

Solution 4: Optimize by sorting the nodes (not implemented)

In principle, alpha-beta algorithm can use a simpler heuristic for evaluating a single node in the search tree, then we sort all nodes on the same level based on its heuristic value by the rules:

- Sort children of Min level in an ascending order based on the heuristic value
- Sort children of Max level in a decreasing order based on the heuristic value

Follow this solution, we can prune branches as much as possible and greatly improve the time efficiency of the algorithm and utilization of the CPU. However, due to the time limit, we didn't implement this solution in our program.

Optimization: Randomize the position that AI chooses when there are multiple moves that have the same heuristic value (implemented)

While playing with the AI player, we found that it is possible that, in our heuristic function, there is more than 1 position that has the same heuristic value. However, if the AI player always chooses the same particular position when encountering this situation, the AI player will always lose if human wins at the end and always follows the same moves. In order to avoid this situation, we made AI choose randomly when there are multiple moves with the same heuristic value. This can increase the flexibility of AI selection.

5 Teamwork

For tasks in project 1, we divided them into several parts: coding, optimizing, testing, tournament, and report. In our team, every team member contributes in particular parts and we performed well in tournament ranking (2 wins and 1 honorable loss) and the project.

Table 6. Teamwork contribution

Tasks	Team member	
	ZhongXu Huang	Yixuan Li
Coding in minimax and alpha-beta	√	
Design heuristic functions	√	√
Optimize heuristic functions	√	
Testing minimax and alpha-beta		√
Participation at the Tournament	√	√
Report	√	√

References

1. Wikipedia, zero-sum game, https://en.wikipedia.org/wiki/Zero-sum_game
2. Wikipedia, alpha-beta pruning, https://en.wikipedia.org/wiki/Alpha-beta_pruning
3. Creating admissible heuristics from functions, <https://cs.stackexchange.com/questions/19976/creating-admissible-heuristics-from-functions>
4. Heuristic five-in-a-row minimax, <http://www.voidcn.com/article/p-ykxoybkm-en.html> (Chinese website)
5. AI five-in-a-row, <https://github.com/lihongxun945/myblog/issues/16> (Chinese website)
6. Connect 4 heuristic function, <http://git.mrman.de/ai-c4/blob/master/s3505919.pdf>
7. Improvement for alpha-beta pruning in the game five-in-a-row, <http://blueve.me/archives/825> (Chinese website)
8. Heuristic design in the game five-in-a-row, <https://zhuanlan.zhihu.com/p/25650252> (Chinese website)