COMP 6721 Double Card Report

1 Introduction and technical details

1.1 Initial Set-up

In this project, I use Tkinter to create an interface[5][9]. I use two for loop to create the game board and add lambda event for each label in the game board. Figure 1 shows the game board.

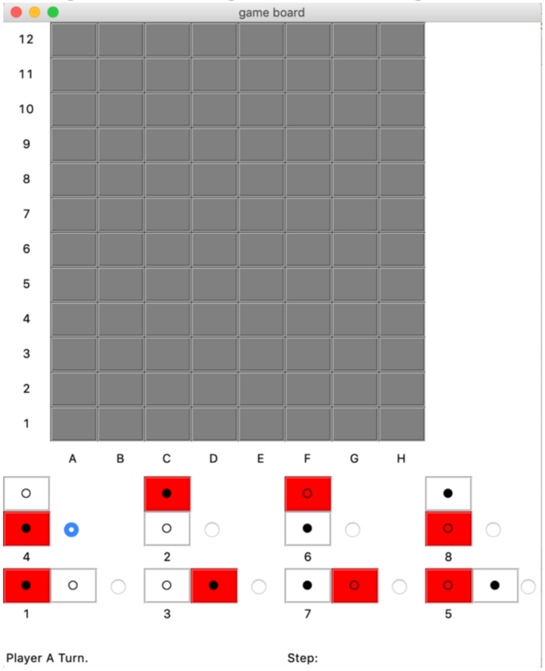
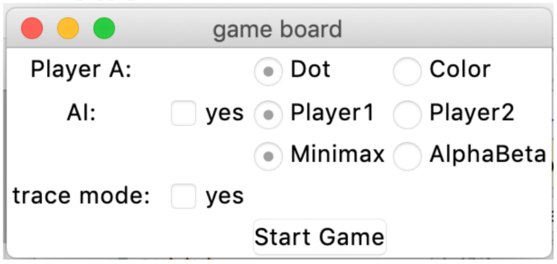


Figure 1. Game board

Before initializing the game board, I create another board to let a user choose to play with AI or play with player2. Figure 2 shows the details[4][9].



game board

Player A: [✓ Dot] Color

AI: yes [✓ Player1] Player2

[✓ Minimax] AlphaBeta

trace mode: yes

Start Game

Figure 2. Play modes

1.2 Moves and Winning

1.2.1 Regular moves

In this part, I use "if" and "elif" to determine the card type first. If card type equals 2,4,6 or 8 then determine the value of x. If x equal 12, it is an illegal move. If there is other card at this position, it also shows an illegal move. If another card type equals 1,3,5 or 7. I have to determine the value of y first. If y equal 8, it is an illegal move. If there are cards already at this position. It also shows an illegal move.

1.2.2 Winning

In this part, I use two functions. One function called “check_winning_result”. In this function, I defined four lists in one list. The four zeros in each list represent a black dot, white dot, red background, and white background. These four lists represent vertical, horizontal, lower left to upper right and lower right to upper left. Each time I call this function. It initializes the list to 0. And then calculate the number of points with the same orientation by adding or subtracting x and y. If they are the same, add one. If they are not the same, stop the operation.
The second function I used in this part is “check_winning”. In this function, it calls the result from the last function, and use a for loop to determine if there is a case that four same dots or color are connected together.

1.2.3 Recycling moves

In this part, I defined a new function called “fillboardafter24”. In this function, I have to determine after24remove equal True or False.
If after24remove is True, it means this step is to remove a card. It detected the card type of the place you choose. And then find another half card. If card type equals 2,4,6 or 8. It checks the point above or below. After finding the same card then check if the point can be canceled. If there are other cards on this card. It will be determined as illegal moves.
If after24remove is False, it means this step is to place a card. First, it checks what is the card type of last removed card. If place the same card at the same position. It shows “can’t put the same card at same position”. If last removed card type equals 1,2,3 or 4 but the place card type is not in 1,2,3 or 4. It shows “Must be rotated card”.

1.2.4 AI

When I initialized the game board. If I choose AI to play as a player. When the game start, if it is AI’s turn, AI will use minimax or alpha beta depends on what did you choose at the beginning. For AI part, I write 10 functions only used for AI part.

- “AI_card_list” this function returns a list contains all the available moves.
- “Minimax_tmpboard” based on the available moves from the last step, this function gives a tmp_board, but don’t refresh the original board and the GUI interface. The following steps use the board generated from this function.
- “AI_check_winning_result”, “cgetpoint” and “cget”
  For each tmp_board generated from the last step. We need to check if there is anyone to win the game. But the GUI interface didn’t refresh, so we can’t use the function we used in manual mode. So I use “cgetpoint” and “cget” to get the text or background of the position. And I also write a new function to get the list result. And give the result to the next function to analysis if there is someone to win the game.
- “AI_check_winning” use the result from last function to analysis which player wins.
- “AI_removecardlist”, “AIafter24checkremoveable”
  This function is for AI recycling moves. During the recycling moves, the first step is to know which step can I remove from the board. This function gives you a list of removable card type and position. The second function is to check the availability of every single position.
- “AI_card_list_after24” and “AI_check_card_after24”
  The first function gives back a list of available position after removing a card. It uses two for loop to analyse the availability of each position. The second function is used to analyze the availability of every single position.

1.2.4 Minimax & Alpha Beta pruning

For the minimax algorithm and Alpha Beta algorithm. I divided it into two separate parts. The first one is when the step count is smaller than 24. I use the functions above and recursive algorithm to implement the minimax and alpha beta pruning. I saw some code on GitHub, but they all build tree in advance[7]. Because this program has a three seconds limitation. It would waste a lot of time to build a tree. I didn’t choose this way. My program is to build the tree while doing the analysis. So it saves a lot of time[2].

2 Description and justification of heuristics

2.1 First heuristics

The first heuristics I choose is to calculated how many identical points are near this point. If there are four same card type. The heuristics return 1000. If there are three same card type, it returns 100. If there are two same card type, it returns 10.
Because when I saw this topic, I thought of the same card put together, there should be a greater chance to win. I use minimax and Alpha-Beta to test this heuristic.
For the speed, before I write this heuristic, I use the naïve heuristic to test the minimax and alpha-beta pruning algorithm. In the first test, I used the for loop to calculate the score of the heuristic of the current step at each node, but because I use for loop to scan all the gameboards very slow and there are many points to be calculated at level 3, so the first test is really slow. To improve speed, I found that each decision is based on the current game board. It will not change. After each player finishes the current step, the program calculates the current heuristics value outside the minimax or alpha-beta function. When operating on the value of each available list, I add the value of the current point in the minimax or alpha-beta pruning function, so it will improve the efficiency.
At last for the naïve heuristics, it only takes less than one second to decide which is the next step with the process of level 3.
When the step counts lager than 24, we need the recycling moves. I use a for loop to decide which step should be canceled. And then calculate the score of the heuristic of the current game board to save time for the next step.
To balance the speed of evaluation function with performance, I try to run with depth 3. It works and then I try to increase the depth to 4 and see if the time is still below 3 seconds. Most times it is below 3 seconds, but sometimes it is more than 3 seconds. So I applied a dictionary in this function, because reading data directly from the memory will waste some memory, but it will save some time.

2.2 Second heuristics

The second heuristics I choose is to get the value of "AI_check_winning_result" and then use a for loop to get the final score. The result contains four integers, which means "black dot", "white dot", "red background" and "white background". If the current player is playing dots and the dots result is larger or equal to 4, the final score plus 100000. If the current player is playing dots but the background result is larger or equal to 4, the final score reduced by 100000. If the current player is playing dots and the dots result is equal to 3, the final score plus 1000. If the current player is playing dots and the color results are equal to 3, the final score reduced by 1000. If the current player is playing dots and the dots result is equal to 2, the final score plus 10. If the current player is playing dots and the color result is equal to 2, the final score reduces 10. If the current player is playing dots and dots result is 1, the final score plus 1. If the current player is playing dots and the color result is 1, the final score reduces 1.
I thought out this heuristic because I was inspired by the Minimax and Alpha-beta pruning lab experiment. At the same time, in my program I also have a function can calculate the value of winning result. So, I can just use the result to the analysis of the situation.
For the speed, I try to build a tree before analysis. But it takes at least 5 seconds. I try another way to implement the tree structure. So in my final code my minimax and Alpha-beta don't have to build tree before analysis, so if I use Minimax I can at least explore level 4 with less than 6 seconds. But for Alpha-Beta pruning, because it prunes the useless point. So most time it can explore level 4 with less than 3 seconds. But if I try to explore level 5, it will take a very long time.
When I test the heuristic by myself. I found that there is a situation based on the game board for now if I can win the game with only one move. But if we don't add any judgment conditions, the program will process the next step with the whole level. For the entire depth value, if you don't do the judge, the steps you take for the next move may make the opponent win or miss the change to win.
To solve this problem, I add another judge function, if the current level equal to begin level plus one. It means the current level is the entire legal moves. And then I use the "AI_Check_winning" function to check if this move can make the current player win this game. If it returns the value of the current player. I just break the minimax or Alpha-Beta pruning function. And put the right card in the right position.
To balance the speed of evaluation function with performance, I changed some judgment functions, before the for loop, I use "AI_check_winning_result" to get the raw result list. I use "cget" to implement the same

function of “lablelist.cget”, but found it will take more time than if function. So I just rewrite the whole function and use “if” and “elif” to do the judge instead of using a function. And it increases some efficiency.

3 A description and analysis of my results

3.1 My one test

Before the tournament, I test my heuristics by myself and I found that the second heuristic is better than the first one. Here is a score sheet that I recorded.

	win	lose
Heuristics1	2	15
Heuristics2	15	2

In this test, I found that the first heuristics is not good enough for the tournament. The second one is better. It can get the exact result from the current board. We can know which player has a greater chance.
The second heuristics can find the position to win directly. So when AI is player1, I set the max value when AI is played, and the player plays the minimum value, so AI wins the most.
For the first heuristics, although in the first heuristics, it is reasonable to judge whether the card type is the same, there are still cases where the card type is different and can win, so in the second heuristics, more complicated detection is performed to ensure the accuracy of results. The card type is as shown in Figure 3.

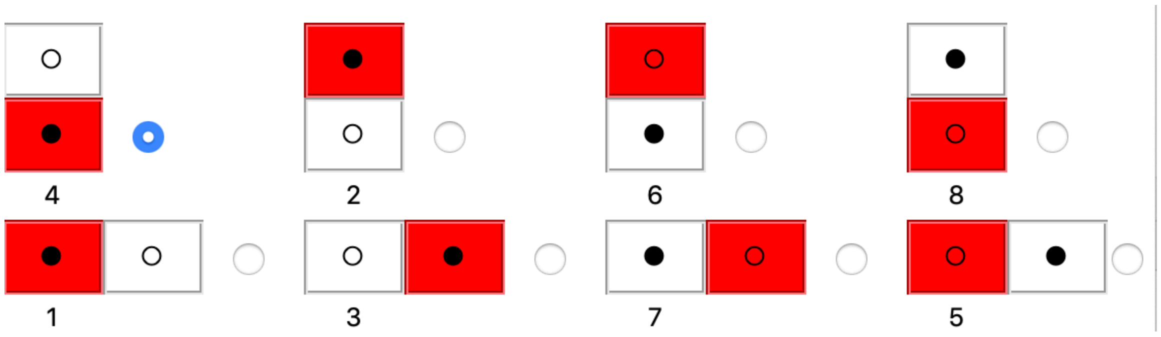
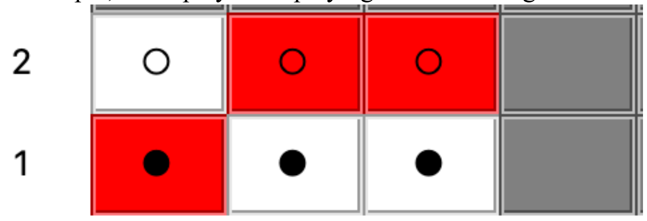


Figure 3, eight card types

For example, if the player1 is playing dots. But the game board is like this.



If I use heuristics two. It will directly give me the correct moves. For example, put card type 4 at position D1, or card type 6 at position D1 or card type 1 or 7 at position D1. But if I use the first heuristics. I cannot get the correct move and card type according to the heuristic. Because the card type if not the same. The program will not give the most available moves.

3.2 Tournament

Tournament1:
Because my heuristics can explore at least level 4 and it is based on the most probable steps in each step. But the heuristic of my opponent seems not to be very accurate. It can only use card 4, and can only put cards up, so my heuristic won two games.

Tournament2:
During this tournament, I found there is a situation, my AI will generate a card type and a position that will give three same color or same dots for the opponents. The opponents will win if they put the same card next to this one. During the tournament2, I found that my heuristic is not worse than the opponent, but when I select the position that I can place card. I should cancel the position and card type that can cause the opponent

to get a three-point connection. Try to avoid the situation like after preventing this step, the opponent will win in the next step. To do so, I should get an opponent's score at the same time with getting the score of my own step. Remove those steps which can give opponents a three-point connection from the available list.

Tournament3:
During the tournament3, I win one time and lose one time. For the one I won was because the opposite gamer made my AI win. But the one lost was because my AI generated a step to connect the three positions on the opposite side.

In conclusion, my biggest inspiration during the tournament is that in the case of selection, we should make an early judgment on each step. If this step directly allows the opponent to win or give the opponent a big advantage, this step should be removed from the list.

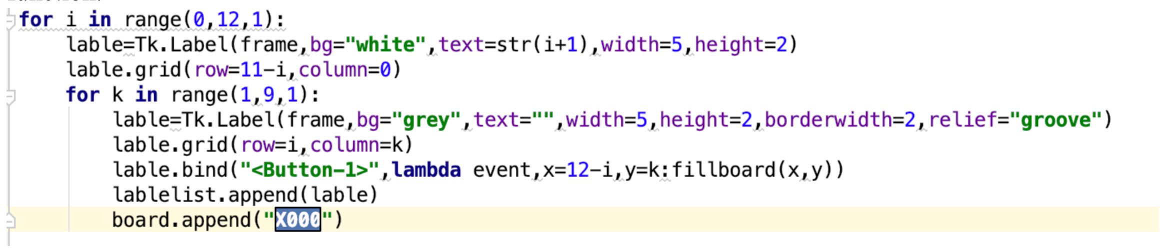
4 A description of difficult

4.1 Time limitation of AI make decision

Build the tree first and then scan and analysis the tree will waste a lot of time. So in order to make AI make decisions faster, I used a recursive method to parse the tree as well as the spanning tree. In the process of spanning tree, I provide a separate function, based on the chessboard situation, find all the points that can play chess and his card type. Based on the returned list, I can use the same function to recursively layer by layer and complete the tree.
According to the recursive way of working, when recursing to the largest layer, the node at this time is the final node, doing heuristics on these nodes, returning the required values according to minimax, and ending the recursive function because of the recursive way of working. I will always go back to the beginning of the node, so the final decision will be automatically brought back to the original node so that the correct result is obtained.

4.2 How to organize data

Because in the game, every point has his coordinates and card type and player information, how to store and block this data is a problem I encountered. So in order to solve this problem, I used a list to store all the information, I designed a calculation method, you can convert the coordinates into a subscript of the list.
This solves the problem of coordinates. In order to organize the data more conveniently, I created a string format to store which player the card is placed, and step count and card type. If there is nothing in this place, the 0th digit of the string is X. If it is player A, the 0th digit is A. If it is player b, the 0th bit is b. If this game requires more players in the future, I only need to modify the 0th bit to change the entire data structure.
This is how I implement the game board when I initialize the game board, I put grey as a background for each label and add lambda event for each label that means if you click the game board, it will call "fillboard" function.



```
for i in range(0,12,1):
    lable=Tk.Label(frame,bg="white",text=str(i+1),width=5,height=2)
    lable.grid(row=11-i,column=0)
    for k in range(1,9,1):
        lable=Tk.Label(frame,bg="grey",text="",width=5,height=2,borderwidth=2,relief="groove")
        lable.grid(row=i,column=k)
        lable.bind("<Button-1>",lambda event,x=12-i,y=k:fillboard(x,y))
        lablelist.append(lable)
        board.append("X000")
```

4.3 How to check which player win the game

Due to the complexity of the rules of the game and the variety of physiological conditions of the game, how to check the win or loss of the game is a problem I encountered.
To solve this problem, I designed two functions. The first function can return a list, which can give me counts in four different physiological conditions in different directions. The second function is based on the result of first function and analysis the result. If the data is greater than or equal to 4, it means that the data satisfies the victory condition in this direction. At this time, according to the structure of the stored information I

designed, which player is read under this step, Read the game's choice of dot or color to determine whether to win or lose[1].

5 Contribution

Because I did this project independently. So I have done everything from UI design, data structure design, to the logic building, code implementation, game testing, and AI implementation, heuristics writing, and report.

6 Compare the heuristics

6.1 Heuristics 1

The advantage of the first heuristics is the speed and easy to implement. Because this heuristic they don't have many things to consider, just the type of card, you can get the type of the card just by reading the data of the board.
The disadvantage of the first heuristics is simple logic, as I said before, this is still some situation occurs we don't need to put the same card type together to win this game. There are also many combinations can win the game with different card type.

6.2 Heuristics2

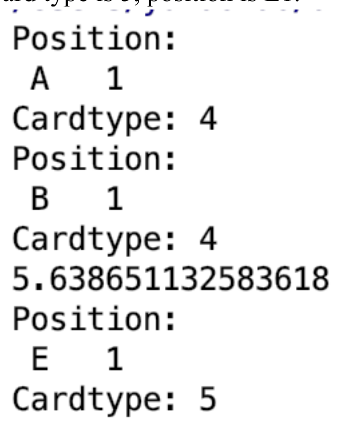
The advantage of the first heuristics is good logic. AI wins more chances, these heuristics takes into account a variety of situations. It also returns the value of color when you play dot, so based on this heuristic you can get everything you want, but it takes a lot of preprocessing to get very satisfactory results.

The disadvantage is speed. Because in this heuristic it uses a lot of functions. If we call many functions in one function, it will slow down, in this heuristic I didn't put every information in the board parameter. So when I implement AI functions, I need to write another function to get some information from the game board. So this may be another disadvantage when I design the structure.

7 Analyze the effectiveness of alpha-beta pruning

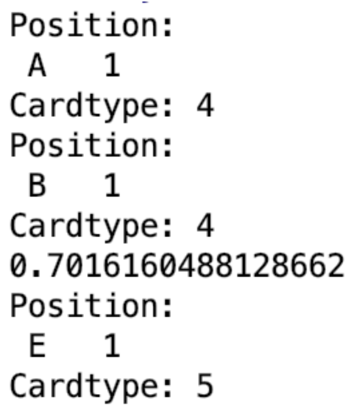
Alpha-beta pruning saves a lot of time. Because I want to test the time each step of AI runs. I added a timer for AI[7], so I can have more intuition results. Here are some screenshot from my command line. They all traverse the same depth and produce the same results, but the time varies greatly.

For the minimax, depth = 4, AI first, AI play dots.
Human play card type 4 and put on position B1. It takes 5.6 seconds to generate the next step. And next step card type is 5, position is E1.



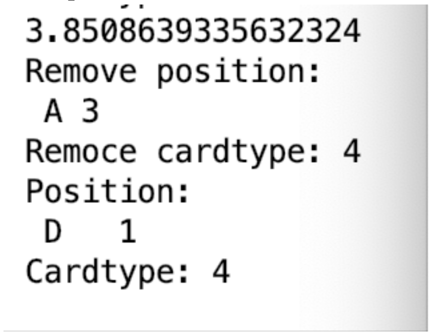
```
Position:
 A   1
Cardtype: 4
Position:
 B   1
Cardtype: 4
5.638651132583618
Position:
 E   1
Cardtype: 5
```

But for Alpha-Beta pruning, depth = 4, AI first, AI play dots.
Human play card type 4 and put on position B1. It takes only 0.7 seconds to generate the next step. And next step card type is 5, position is E1. Same with minimax but use less than five times the time.

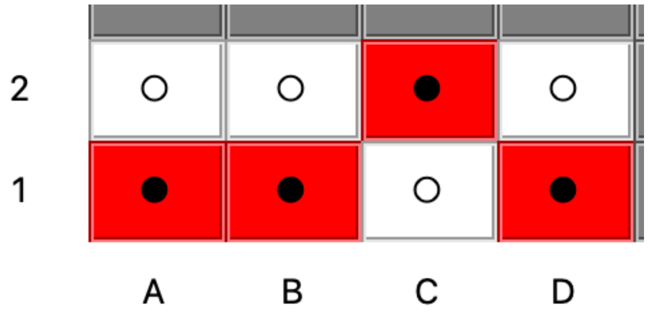


```
Position:
 A   1
Cardtype: 4
Position:
 B   1
Cardtype: 4
0.7016160488128662
Position:
 E   1
Cardtype: 5
```

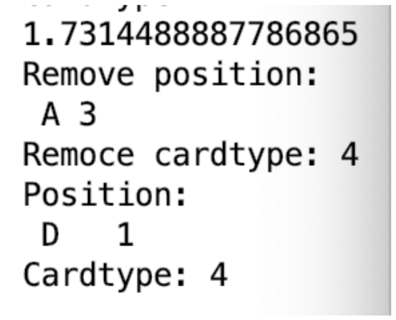
I initialize the step count to 20, and test the time spend, this is the time spend for the first recycling move for minimax, in this step, it remove card type 4 from position A3.and put the same card type to D1. The time spend is 3.8 seconds.



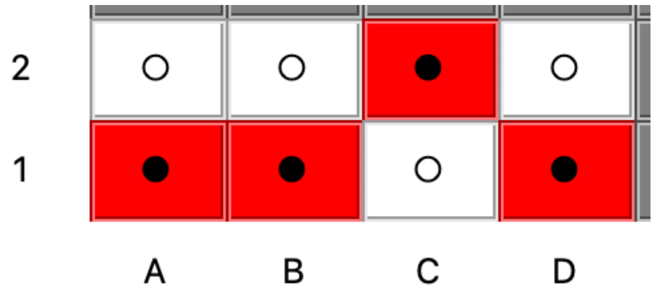
```
3.8508639335632324
Remove position:
 A 3
Remoce cardtype: 4
Position:
 D   1
Cardtype: 4
```



This is the time spend for Alpha-Beta pruning with the same operation, but the time spend is only 1.7 seconds.



```
1.7314488887786865
Remove position:
 A 3
Remoce cardtype: 4
Position:
 D   1
Cardtype: 4
```



So the alpha-beta pruning can use less time than minimax. And Alpha-beta and minimax returns the same answer. All Alpha-Beta does is prevent minimax from making calculations that are 100% guaranteed to NOT be an optimal state for the current player [3].

Reference:

1. Jones, M. (2019). Check for checkmate. [online] Chess Stack Exchange. Available at: https://chess.stackexchange.com/questions/13871/check-for-checkmate [Accessed 28 Mar. 2019].
2. Korf, R. (1994). Best-First Minimax Search: Othello Results. [online] Aaai.org. Available at: https://www.aaai.org/Papers/AAAI/1994/AAAI94-210.pdf [Accessed 28 Mar. 2019].
3. Stack Overflow. (2019). Minimax vs Alpha Beta Pruning algorithms. [online] Available at: https://stackoverflow.com/questions/40492848/minimax-vs-alpha-beta-pruning-algorithms [Accessed 28 Mar. 2019].
4. Python-textbok.readthedocs.io. (2019). Introduction to GUI programming with tkinter — Object-Oriented Programming in Python 1 documentation. [online] Available at: https://python-textbok.readthedocs.io/en/1.0/Introduction_to_GUI_Programming.html [Accessed 28 Mar. 2019].
5. Razvan, S. (2019). Tkinter binding a function with arguments to a widget. [online] Stack Overflow. Available at: https://stackoverflow.com/questions/7299955/tkinter-binding-a-function-with-arguments-to-a-widget [Accessed 28 Mar. 2019].
6. Aima.cs.berkeley.edu. (2019). AIMA Python file: games.py. [online] Available at: http://aima.cs.berkeley.edu/python/games.html [Accessed 28 Mar. 2019].
7. Haapala, A. (2019). Creating a timer in python. [online] Stack Overflow. Available at: https://stackoverflow.com/questions/18406165/creating-a-timer-in-python/18406263 [Accessed 28 Mar. 2019].
8. Russell, S. and Norvig, P. (n.d.). Artificial intelligence.
9. Effbot.org. (2019). The Tkinter Button Widget. [online] Available at: http://effbot.org/tkinterbook/button.htm [Accessed 28 Mar. 2019].