CSC442: Introduction to AI

Project 1: Game Playing

Ziyi Kou

Collaboration: Ziyi Kou & Ziqiu Wu

1. Basic Tic Tac Toe

The basic Tic Tac Toe game is a game with two players playing on a 3*3 board. The player who succeeds in placing three of their marks in a horizontal, vertical, or diagonal row firstly wins the game.[1]

1.1 Formulating

From the rule, we could see Tic Tac Toe game is a deterministic, turn-taking, two-player, perfect informational zero-sum game, which means we could regard two players as two agents competing in a fully observable environment. The utility values of two agents in the end of the game are always equal or opposite.

Such a type of game could be denoted as a tree (see Fig.1). The nodes are game states, the edges are moves. With the root node as the initial state, each layer below represents the possible moves of one of two layers alternately while leaf nodes represent the end of the game. There are three types of results and different result will return different score.

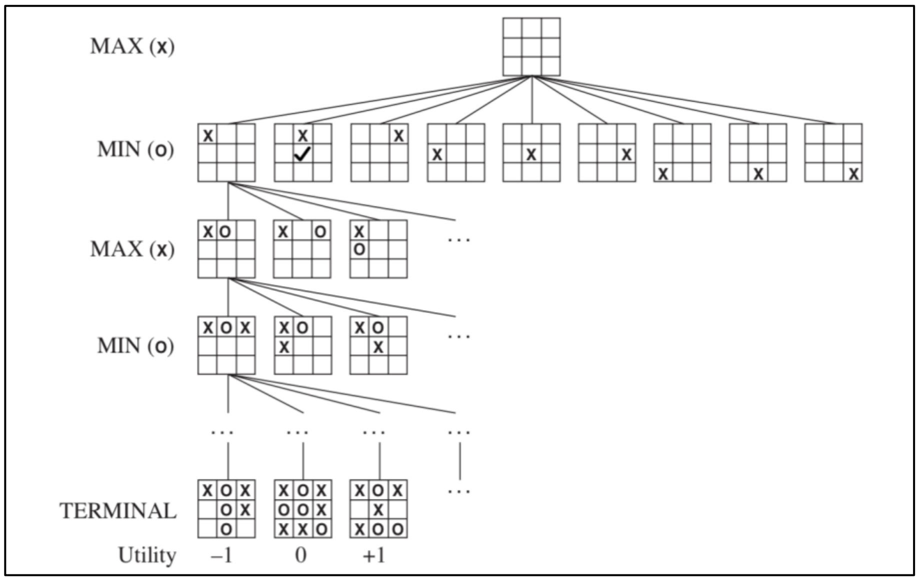


Fig. 1 zero-sum game could be denoted as a tree

To simplify such a problem, we could define it formally by several components:

- **States**: the state could be determined by three different marks:

$$mark_i \in \{'X', 'O', ''\} \\ S=\{mark_1, mark_2, \ldots, mark_8\} \quad (1.1)$$

- **Initial state**: the state that nobody has marked on the board, which means all the mark on the board is $''$:

$$S_{initial}=\{'','','','','','','',''\} \quad (1.2)$$

- **Actions**: two players put their marks on valid space in the board, so we could define it as follows:

$$action=[S[i], PlayerMark] \\ PlayerMark \in \{'X', 'O'\} \\ i \in [1,9] \quad (1.3)$$

- **Results**: given a state and an action, a result state is returned:

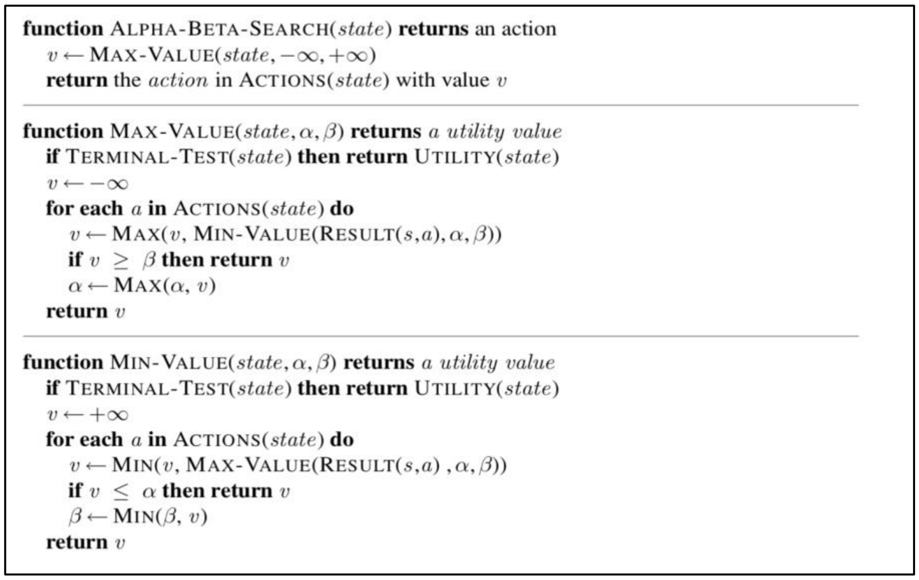
$$results(S, action)=S' = [mark_1, mark_2, \ldots, mark_8] \\ \exists mark_i \neq mark_i \in S \quad (1.4)$$

- **Goal Test**: every turn when AI or human player completes, we check if someone has won
- **Utility**: for basic Tic Tac Toe, we define the utility function of human agent is U(human)=-1 while U(AI)=1, and U(tie)=0

1.2 Strategy/Algorithm

To help AI to find best move, we need to use *ai_minmax* function. That is for every time when AI need to mark, we call this function, using depth first function to return max score for AI from bottom. In the procedure of this function, AI tries to return max value in each AI layer while human player tries to return min value in each human operating layer. With this function, in the environment that both players want to win the other, the specific move of AI could always keep its score best.

However, it's costly for *ai_minmax* function to consider all the depth and all the leaf nodes (see Fig.2). To reduce times it runs and save time, we add *alpha-beta pruning* into *ai_minmax* to disregard worse choices for AI when the later has found better one. There are α and β two parameters in *alpha-beta pruning.* α represents the best choice for MAX agent while β represents the best choice for MIN agent.



```
function ALPHA-BETA-SEARCH(state) returns an action
  $v \leftarrow$ MAX-VALUE($state, -\infty, +\infty$)
  return the action in ACTIONS(state) with value $v$

function MAX-VALUE($state, \alpha, \beta$) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
  $v \leftarrow -\infty$
  for each $a$ in ACTIONS(state) do
    $v \leftarrow$ MAX($v$, MIN-VALUE(RESULT($s,a$), $\alpha, \beta$))
    if $v \geq \beta$ then return $v$
    $\alpha \leftarrow$ MAX($\alpha, v$)
  return $v$

function MIN-VALUE($state, \alpha, \beta$) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
  $v \leftarrow +\infty$
  for each $a$ in ACTIONS(state) do
    $v \leftarrow$ MIN($v$, MAX-VALUE(RESULT($s,a$) , $\alpha, \beta$))
    if $v \leq \alpha$ then return $v$
    $\beta \leftarrow$ MIN($\beta, v$)
  return $v$
```

Fig. 2 minimax with alpha-beta pruning

1.3 Functions

We could divide all functions to three parts which are help functions, controlling functions and algorithm functions (see Table.1). Help functions are some that help the program to output different information on terminal. Controlling functions control the running of the game, letting two agents put on right positions according the rule. Algorithm functions let AI perform best on each turn.

help functions	controlling functions	algorithm functions
plot_block *empty_pos* *time*	*check_first* *is_win* *mark* *clear_mark* *turn*	*ai_minmax*

Table. 1 different types of functions

We explain each function in Table2,3,4,5,6,7.

function name	*plot_block*
function description	this function outputs the current board to terminal which is often called after each agent complete its mark.
return value	N/A

Table. 2 function plot_block

function name	*check_first*

function description	check which agent is the first one to go. The mark of first one is 'X' while the second one if 'O'.
return value	N/A

Table. 3 function check_first

function name	*is_win*
function description	this function check if any agent has won or there is no place for both agents to play. It's usually called in *ai_minmax* to return final score and also after each agent's action.
return value	'Human win'/'AI win'/'draw'

Table. 4 function is_win

function name	*mark*
function description	this function helps each player to mark on the board.
return value	N/A

Table. 5 function mark

function name	*clear_mark*
function description	when AI uses *ai_minmax* to get its best move, it continuously marks on each valid position representing AI and human alternatively to bottom of the tree. The function clears the mark after each node returns its score.
return value	N/A

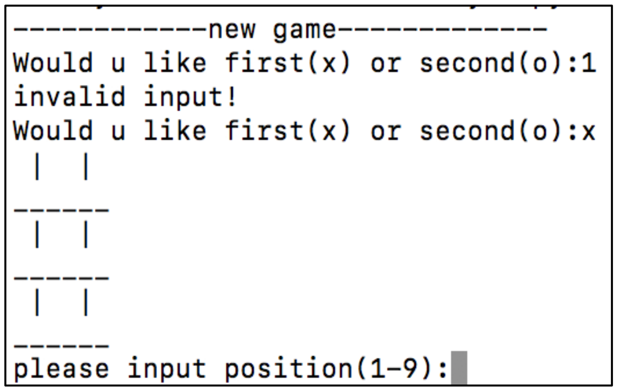
Table. 6 function *clear_mark*

function name	*turn*
function description	This function controls the turn of each agent and also the inputs of them. When it's AI turn, this function calls *ai_minmax* to get best choice for it.
return value	N/A

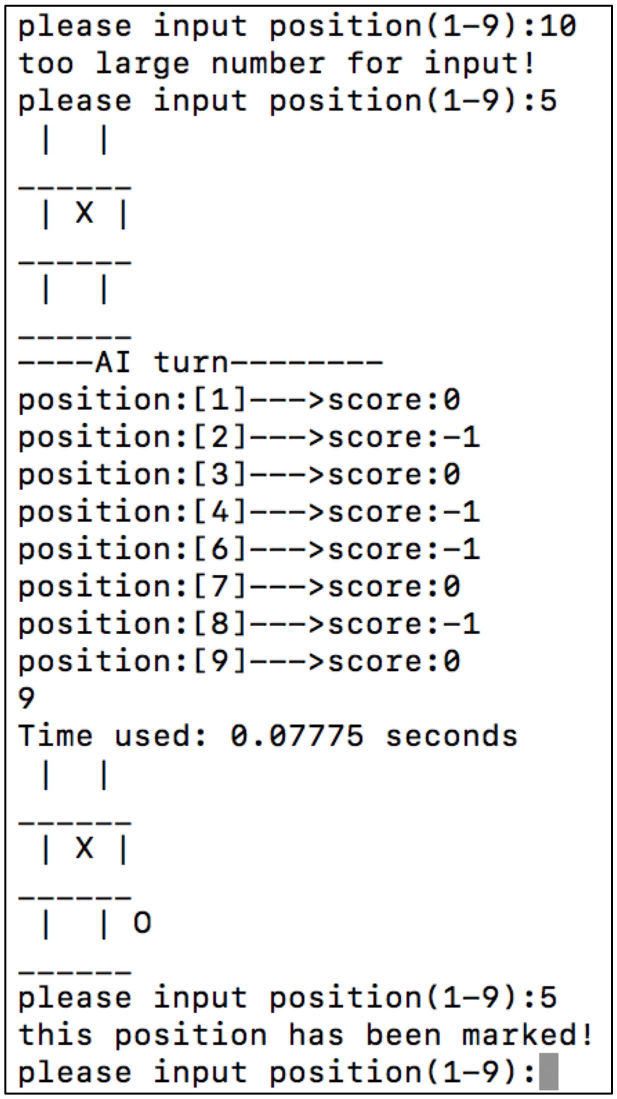
Table. 7 function *turn*

1.4 Demo of game

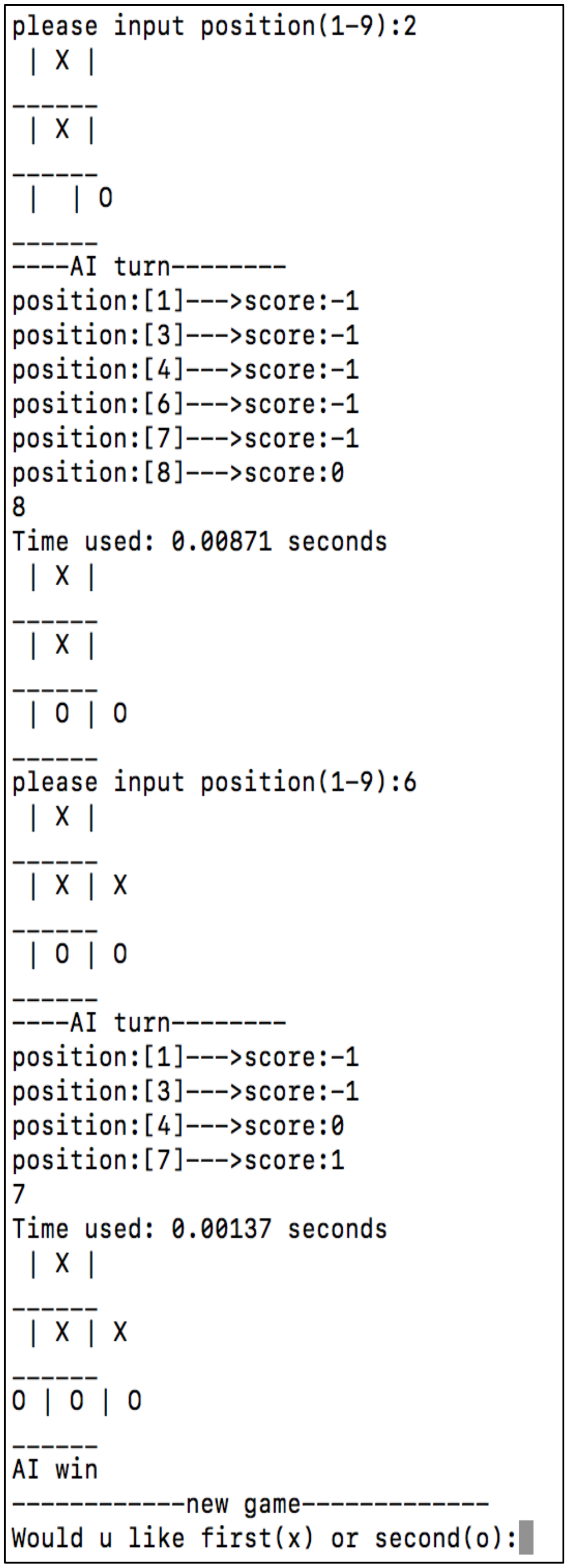
Firstly, we need to choose first or second to play. Invalid input will result in reinput.


```
------------new game-------------
Would u like first(x) or second(o):1
invalid input!
Would u like first(x) or second(o):x
 |  |
------
 |  |
------
 |  |
------
please input position(1-9):
```

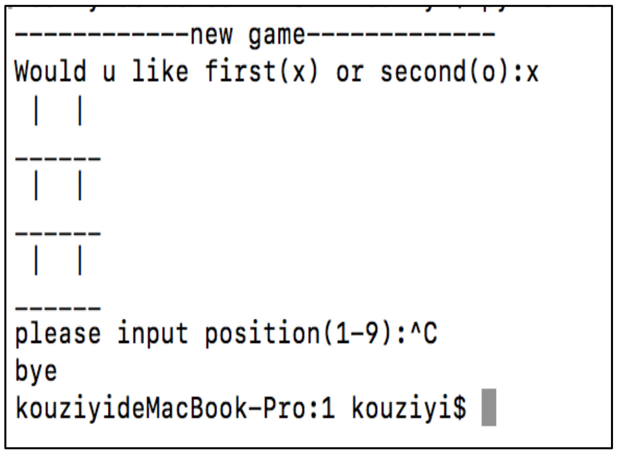
We choose a position to input. Invalid input will result in reinput. Then AI uses *ai_minmax* function to calculate the score of each possible mark. Finally, the whole 'thinking' time will be displayed.


```
please input position(1-9):10
too large number for input!
please input position(1-9):5
 |   |
------
 | X |
------
 |   |
------
----AI turn--------
position:[1]--->score:0
position:[2]--->score:-1
position:[3]--->score:0
position:[4]--->score:-1
position:[6]--->score:-1
position:[7]--->score:0
position:[8]--->score:-1
position:[9]--->score:0
9
Time used: 0.07775 seconds
 |   |
------
 | X |
------
 |   | O
------
please input position(1-9):5
this position has been marked!
please input position(1-9):
```

With *ai_minmax* function, AI could always get the best choice until the end of the game.


```
please input position(1-9):2
 | X |

------
 | X |

------
 |  | O

------
----AI turn--------
position:[1]--->score:-1
position:[3]--->score:-1
position:[4]--->score:-1
position:[6]--->score:-1
position:[7]--->score:-1
position:[8]--->score:0
8
Time used: 0.00871 seconds
 | X |

------
 | X |

------
 | O | O

------
please input position(1-9):6
 | X |

------
 | X | X

------
 | O | O

------
----AI turn--------
position:[1]--->score:-1
position:[3]--->score:-1
position:[4]--->score:0
position:[7]--->score:1
7
Time used: 0.00137 seconds
 | X |

------
 | X | X

------
O | O | O

------
AI win
------------new game-------------
Would u like first(x) or second(o):
```

Once we don't want to play. Just type Ctrl+C to terminate the program.



```
------------new game-------------
Would u like first(x) or second(o):x
 | |
------
 | |
------
 | |
------
please input position(1-9):^C
bye
kouziyideMacBook-Pro:1 kouziyi$ 
```

2. Advanced Tic Tac Toe

The basic idea of advanced Tic Tac Toe is similar to basic Tic Tac Toe. The change is that now there is nine 3*3 board. Each time when one agent marks on a position, another agent need to mark on the board whose index is same as the former position.

2.1 Formulating

For advanced Tic Tac Toe, it's also a zero-sum game. But for every time AI and human player mark, the input needs to be a two-digits input. The first one is the index of board while the second one is the index of position in targeted board.

Besides, it's a too large board for AI to use basic *ai_minmax* function to consider to the bottom of tree. So with limited number of depths approaching, we need to prevent AI to go deeper and return a evaluated score by another utility function, instead of 1/-1/0 before.

We could simplify the problem as below:

- **States**: the state could be determined by three different marks:

$$mark_i \in \{'X','O',''\} \qquad (1.1)$$
$$S=\{mark_{[1,1]}, mark_{[1,2]}, \dots, mark_{[9,9]}\}$$

- **Initial state**: the state that nobody has marked on the board, which means all the mark on the board is $''$:

$$S_{initial}=\{'*','*','*' \dots ,'*'\} \qquad (1.2)$$

- **Actions**: two players put their marks on valid space in the board, so we could define it as follows:

$$action=[S[i], PlayerMark]$$
$$PlayerMark \in \{'X','O'\} \qquad (1.3)$$
$$i \in [1,81]$$

- **Results**: given a state and an action, a result state is returned:

$$results(S, action){=}S' = [mark_1, mark_2, \ldots, mark_8] \\ \exists mark_i \neq mark_i \in S \tag{1.4}$$

- **Goal Test**: every turn when AI or human player completes, we check if someone has won
- **Utility**: for advanced Tic Tac Toe, we use another utility function mentioned below to evaluate the score.

2.2 Strategy/Algorithm

Firstly we introduce our new heuristic function with alpha-beta pruning to estimate the scores even when the there is no winner or loser, nor no tie. Similarly to the part one, the heuristic function called by the minimax will return an estimated score. For every board in 9 boards, there is 8 ways to win. In each way, there is three positions. Different marks on the positions result in different scores (see Fig. 3).

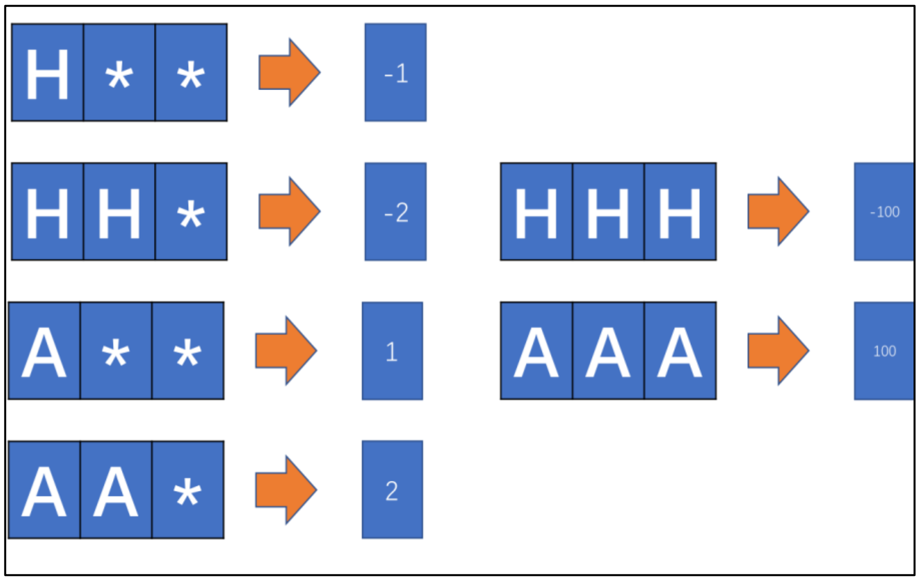


Fig. 3 heuristic function which is used on every win way.

The total score is calculated for 81 times.

we add this score to the traditional minimax function. Instead of calculating the bottom score anyway, we calculate the score of heuristic function when the minimax go into 5th layer but still does not get the final win/draw score, which is shown below:

```
# if there is a win/draw result, just return score
        if self.is_win() and self.is_win() == 'HUMAN win':
            return -100
        if self.is_win() and self.is_win() == 'AI win':
            return 100
        if self.is_win() and self.is_win() == 'draw':
            return 0

        # if depth goes to 5 and no result, use depth_scores() to calculate eval
uation score
        if depth == self.MAX_DEPTH:
            if self.HUMAN_TURN:
                return -1 * self.depth_scores()
            else:
                return self.depth_scores()
```

2.3 Functions

We add some new functions to change basic Tic Tac Toe to advanced one (see Table.8).

help functions	controlling functions	algorithm functions
plot_block *empty_pos* *time* *is_single_chess_empty*	*check_first* *is_win* *mark_pos* *clear_chess_pos* *turn*	*ai_minmax* *depth_scores*

Table. 8 functions in advanced Tic Tac Toe

For most functions, we change the input from one digit to two digits. And also we need to consider the empty positions returned since there is some situation that the targeted board is full and we need to return all valid positions in the whole board. The below are some main changes.

function name	*empty_pos*
function description	If there is empty positions on targeted board, return them. Else return empty positions on the whole board.
return value	List: empty positions

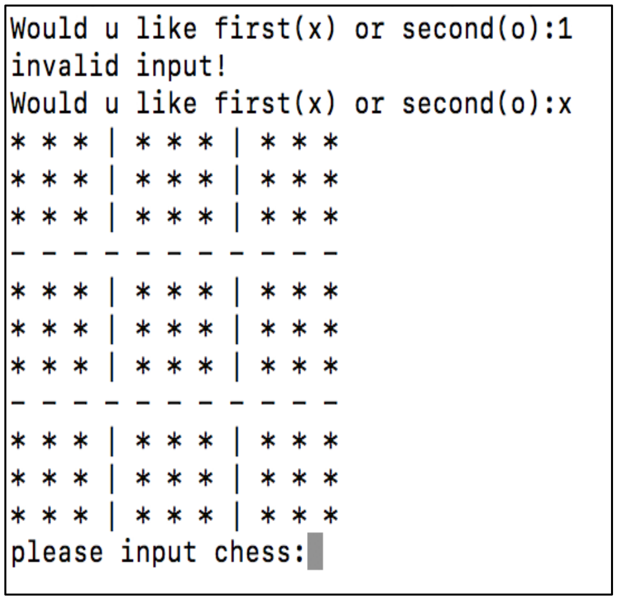
Table. 9 function empty_pos

function name	*is_single_chess_empty*
function description	If there is empty positions on targeted board, return them. Else return zero. This is used to check if human player need to input 2 digits or just 1position with determined board by previous move.
return value	List: empty positions

Table. 10 function is_single_chess_empty

2.4 Demo of game

Firstly, we need to choose first or second to play. Invalid input will result in reinput.

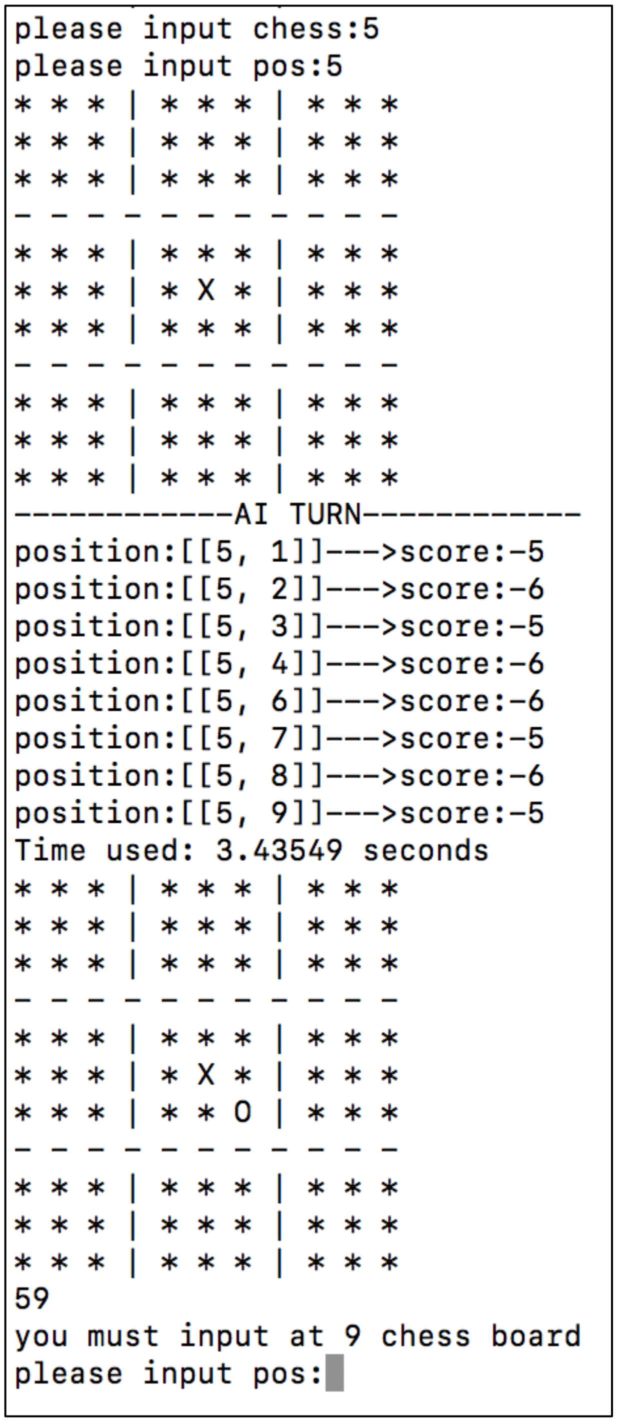


```
Would u like first(x) or second(o):1
invalid input!
Would u like first(x) or second(o):x
* * * | * * * | * * *
* * * | * * * | * * *
* * * | * * * | * * *
- - - - - - - - - - -
* * * | * * * | * * *
* * * | * * * | * * *
* * * | * * * | * * *
- - - - - - - - - - -
* * * | * * * | * * *
* * * | * * * | * * *
* * * | * * * | * * *
please input chess:
```

We choose a position to input. Invalid input will result in reinput. Then AI uses *ai_minmax* function to calculate the score of each possible mark on targeted board. Finally, the whole 'thinking' time will be displayed.

In the later game, when there is no position on the targeted board, human player need to input two digits while AI would consider the empty position on the whole board.

Other parts are similiar as the basic Tic Tac Toe.



```
please input chess:5
please input pos:5
* * * | * * * | * * *
* * * | * * * | * * *
* * * | * * * | * * *
- - - - - - - - - - -
* * * | * * * | * * *
* * * | * X * | * * *
* * * | * * * | * * *
- - - - - - - - - - -
* * * | * * * | * * *
* * * | * * * | * * *
* * * | * * * | * * *
------------AI TURN------------
position:[[5, 1]]--->score:-5
position:[[5, 2]]--->score:-6
position:[[5, 3]]--->score:-5
position:[[5, 4]]--->score:-6
position:[[5, 6]]--->score:-6
position:[[5, 7]]--->score:-5
position:[[5, 8]]--->score:-6
position:[[5, 9]]--->score:-5
Time used: 3.43549 seconds
* * * | * * * | * * *
* * * | * * * | * * *
* * * | * * * | * * *
- - - - - - - - - - -
* * * | * * * | * * *
* * * | * X * | * * *
* * * | * * O | * * *
- - - - - - - - - - -
* * * | * * * | * * *
* * * | * * * | * * *
* * * | * * * | * * *
59
you must input at 9 chess board
please input pos:
```

3. Super Tic Tac Toe

The idea of super Tic Tac Toe is similar with advanced Tic Tac Toe. Instead of winning on one board, this time we need to win on 3 board ruled the same as basic Tic Tac Toe.

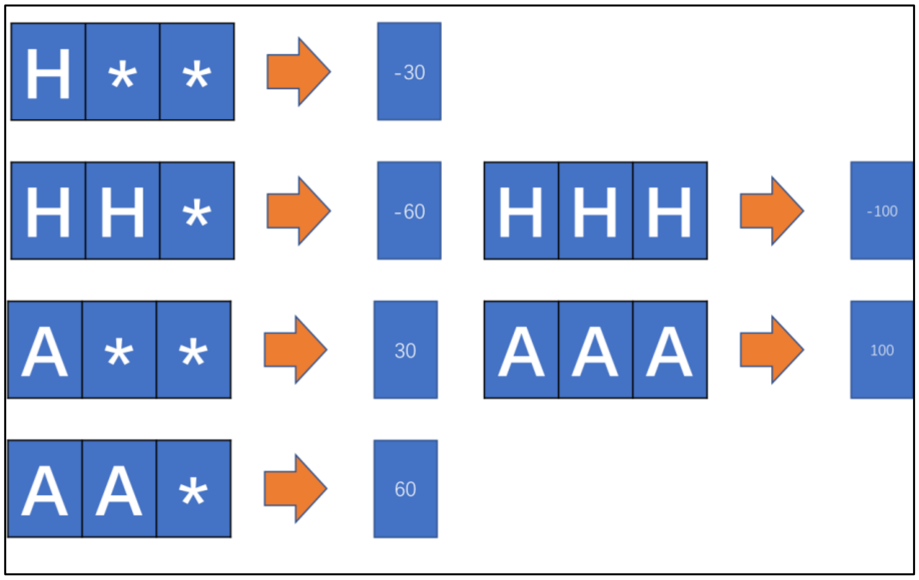
3.1 Formulating

The only change to the super Tic Tac Toe is changing the heuristic function and the way to win.

We need to define two boards. One is 9*3*3 which is the whole large board while the other is 3*3 that each of it denotes a single board on large board. We call it small board.

3.2 Strategy/Algorithm

In the heuristic function , after calculating the heuristic score like advanced Tic Tac Toe, we need to continue to calculate the score on the small board with the same way. That means we need to calculate 9*8 times on large board and then 8 times on small board. In the small board, the score is larger (see Fig. 4).



3.3 Functions

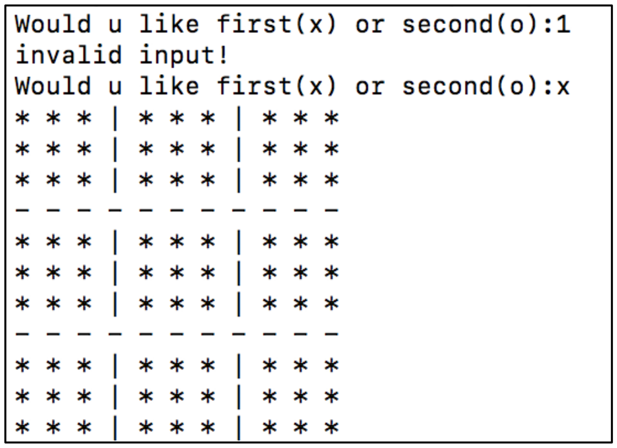
There are only some small changes compared with advanced Tic Tac Toe. We add a new function *check_single_chess_win* which is used to update the state of the small board after each mark by human and AI players. If there is one single board wins, we set empty position on that board to be '-' instead of '*', which means it's no longer valid to input.

3.4 Demo of game

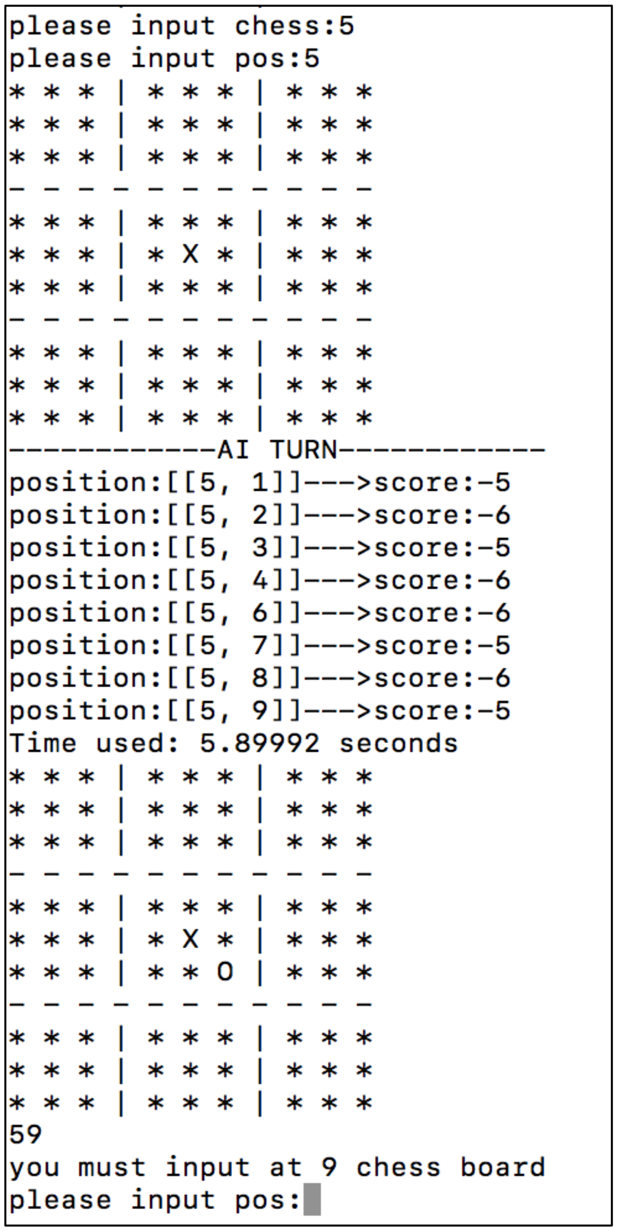
Firstly, we need to choose first or second to play. Invalid input will result in reinput.

We choose a position to input. Invalid input will result in reinput. Then AI uses *ai_minmax* function to calculate the score of each possible mark on targeted board. Finally, the whole 'thinking' time will be displayed.

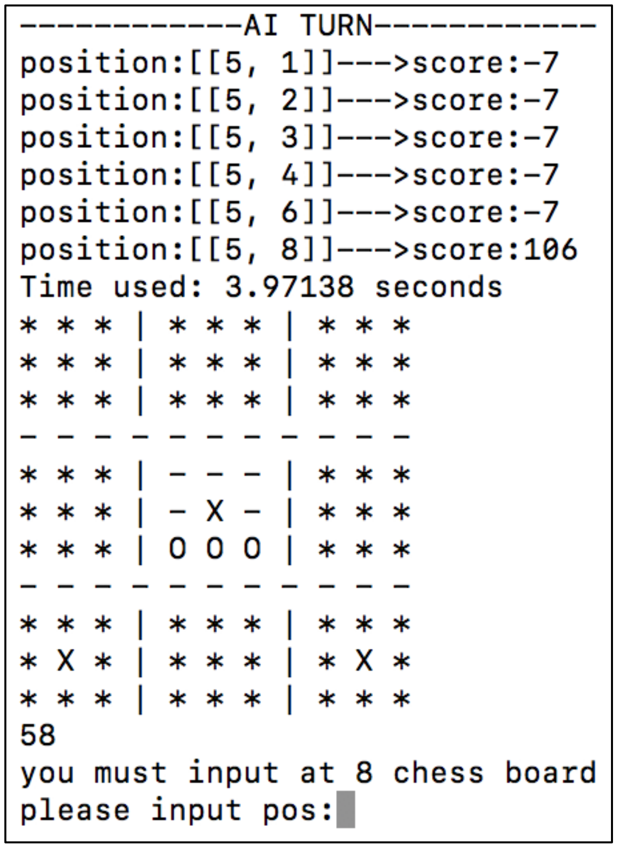
When there is single board win, empty positions will be set to '-'.



```
Would u like first(x) or second(o):1
invalid input!
Would u like first(x) or second(o):x
* * * | * * * | * * *
* * * | * * * | * * *
* * * | * * * | * * *
- - - - - - - - - - -
* * * | * * * | * * *
* * * | * * * | * * *
* * * | * * * | * * *
- - - - - - - - - - -
* * * | * * * | * * *
* * * | * * * | * * *
* * * | * * * | * * *
```



```
please input chess:5
please input pos:5
* * * | * * * | * * *
* * * | * * * | * * *
* * * | * * * | * * *
- - - - - - - - - - -
* * * | * * * | * * *
* * * | * X * | * * *
* * * | * * * | * * *
- - - - - - - - - - -
* * * | * * * | * * *
* * * | * * * | * * *
* * * | * * * | * * *
------------AI TURN------------
position:[[5, 1]]--->score:-5
position:[[5, 2]]--->score:-6
position:[[5, 3]]--->score:-5
position:[[5, 4]]--->score:-6
position:[[5, 6]]--->score:-6
position:[[5, 7]]--->score:-5
position:[[5, 8]]--->score:-6
position:[[5, 9]]--->score:-5
Time used: 5.89992 seconds
* * * | * * * | * * *
* * * | * * * | * * *
* * * | * * * | * * *
- - - - - - - - - - -
* * * | * * * | * * *
* * * | * X * | * * *
* * * | * * O | * * *
- - - - - - - - - - -
* * * | * * * | * * *
* * * | * * * | * * *
* * * | * * * | * * *
59
you must input at 9 chess board
please input pos:
```



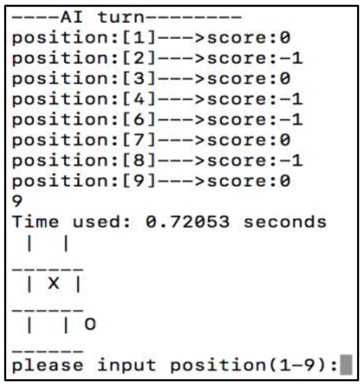
```
------------AI TURN------------
position:[[5, 1]]--->score:-7
position:[[5, 2]]--->score:-7
position:[[5, 3]]--->score:-7
position:[[5, 4]]--->score:-7
position:[[5, 6]]--->score:-7
position:[[5, 8]]--->score:106
Time used: 3.97138 seconds
* * * | * * * | * * *
* * * | * * * | * * *
* * * | * * * | * * *
- - - - - - - - - - -
* * * | - - - | * * *
* * * | - X - | * * *
* * * | O O O | * * *
- - - - - - - - - - -
* * * | * * * | * * *
* X * | * * * | * X *
* * * | * * * | * * *
58
you must input at 8 chess board
please input pos:
```

4. Evaluation

Now we need to evaluate the difference in using *alpha-beta pruning* between without using.

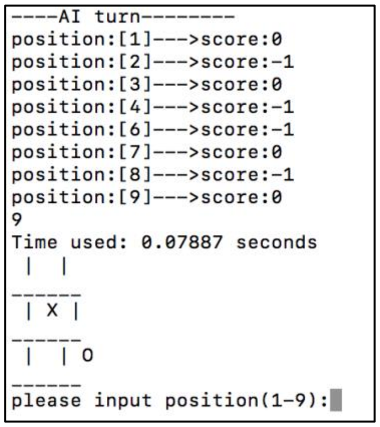
4.1 alpha-beta pruning could fasten the speed of minimax

Basic tic tac toe:
Without *alpha-beta pruning*



```
----AI turn--------
position:[1]--->score:0
position:[2]--->score:-1
position:[3]--->score:0
position:[4]--->score:-1
position:[6]--->score:-1
position:[7]--->score:0
position:[8]--->score:-1
position:[9]--->score:0
9
Time used: 0.72053 seconds
 |  |
------
 | X |
------
 |  | O
------
please input position(1-9):
```

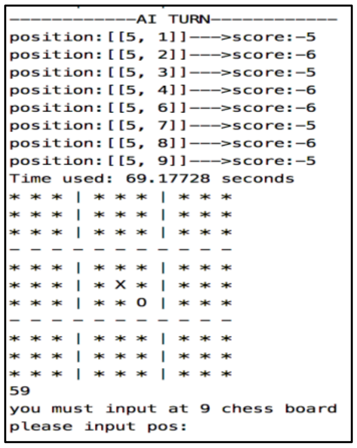
With alpha-beta pruning



```
----AI turn--------
position:[1]--->score:0
position:[2]--->score:-1
position:[3]--->score:0
position:[4]--->score:-1
position:[6]--->score:-1
position:[7]--->score:0
position:[8]--->score:-1
position:[9]--->score:0
9
Time used: 0.07887 seconds
 |  |
------
 | X |
------
 |  | O
------
please input position(1-9):
```

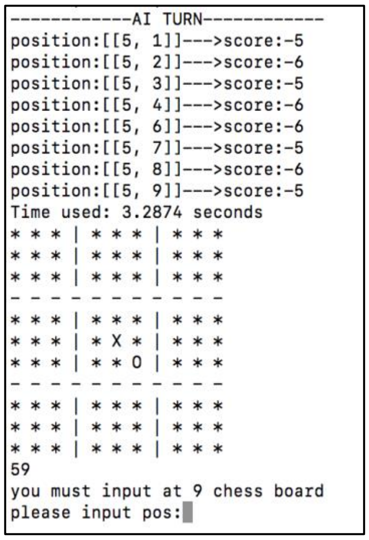
Advanced tic tac toe:

Without *alpha-beta pruning*



```
------------AI TURN------------
position:[[5, 1]]--->score:-5
position:[[5, 2]]--->score:-6
position:[[5, 3]]--->score:-5
position:[[5, 4]]--->score:-6
position:[[5, 6]]--->score:-6
position:[[5, 7]]--->score:-5
position:[[5, 8]]--->score:-6
position:[[5, 9]]--->score:-5
Time used: 69.17728 seconds
* * * | * * * | * * *
* * * | * * * | * * *
* * * | * * * | * * *
- - - - - - - - - - -
* * * | * * * | * * *
* * * | * X * | * * *
* * * | * * O | * * *
- - - - - - - - - - -
* * * | * * * | * * *
* * * | * * * | * * *
* * * | * * * | * * *
59
you must input at 9 chess board
please input pos:
```

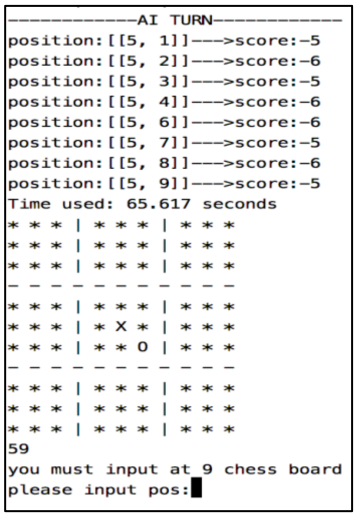
With alpha-beta pruning



```
------------AI TURN------------
position:[[5, 1]]--->score:-5
position:[[5, 2]]--->score:-6
position:[[5, 3]]--->score:-5
position:[[5, 4]]--->score:-6
position:[[5, 6]]--->score:-6
position:[[5, 7]]--->score:-5
position:[[5, 8]]--->score:-6
position:[[5, 9]]--->score:-5
Time used: 3.2874 seconds
* * * | * * * | * * *
* * * | * * * | * * *
* * * | * * * | * * *
- - - - - - - - - - -
* * * | * * * | * * *
* * * | * X * | * * *
* * * | * * O | * * *
- - - - - - - - - - -
* * * | * * * | * * *
* * * | * * * | * * *
* * * | * * * | * * *
59
you must input at 9 chess board
please input pos:
```

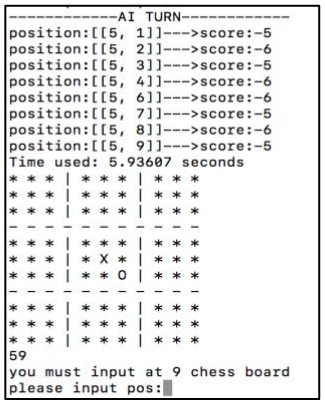
Super tic tac toe:

Without *alpha-beta pruning*



```
------------AI TURN------------
position:[[5, 1]]--->score:-5
position:[[5, 2]]--->score:-6
position:[[5, 3]]--->score:-5
position:[[5, 4]]--->score:-6
position:[[5, 6]]--->score:-6
position:[[5, 7]]--->score:-5
position:[[5, 8]]--->score:-6
position:[[5, 9]]--->score:-5
Time used: 65.617 seconds
* * * | * * * | * * *
* * * | * * * | * * *
* * * | * * * | * * *
- - - - - - - - - - -
* * * | * * * | * * *
* * * | * X * | * * *
* * * | * * O | * * *
- - - - - - - - - - -
* * * | * * * | * * *
* * * | * * * | * * *
* * * | * * * | * * *
59
you must input at 9 chess board
please input pos:
```

With alpha-beta pruning



```
------------AI TURN------------
position:[[5, 1]]--->score:-5
position:[[5, 2]]--->score:-6
position:[[5, 3]]--->score:-5
position:[[5, 4]]--->score:-6
position:[[5, 6]]--->score:-6
position:[[5, 7]]--->score:-5
position:[[5, 8]]--->score:-6
position:[[5, 9]]--->score:-5
Time used: 5.93607 seconds
* * * | * * * | * * *
* * * | * * * | * * *
* * * | * * * | * * *
- - - - - - - - - - -
* * * | * * * | * * *
* * * | * X * | * * *
* * * | * * O | * * *
- - - - - - - - - - -
* * * | * * * | * * *
* * * | * * * | * * *
* * * | * * * | * * *
59
you must input at 9 chess board
please input pos:
```

5. Future work

5.1 there are another two projects on part3 project that we could choose. Maybe in the future we could implement it.
5.2 in our code, when facing the same scores from minimax function, we choose the last one. But there is a better way, we could add one depth and then calculate these positions with same scores. After that, if we still have positions with same scores, we could do this again until there is only one best position.