Explanation of Project1 : Minimax tic tac toe

20191097 배호용

1.Minimax algorithm

Search most advantageous choice when the other always does disadvantageous behaviors for me.

Supposition of minimax:

-Game of 2 player(named 'max' and 'min') taking turns in order.

-Max is computer, min is human

-Each player does best choice.

-One's win means the other one's lose(zero sum game).

-Max's turn is first.

Process

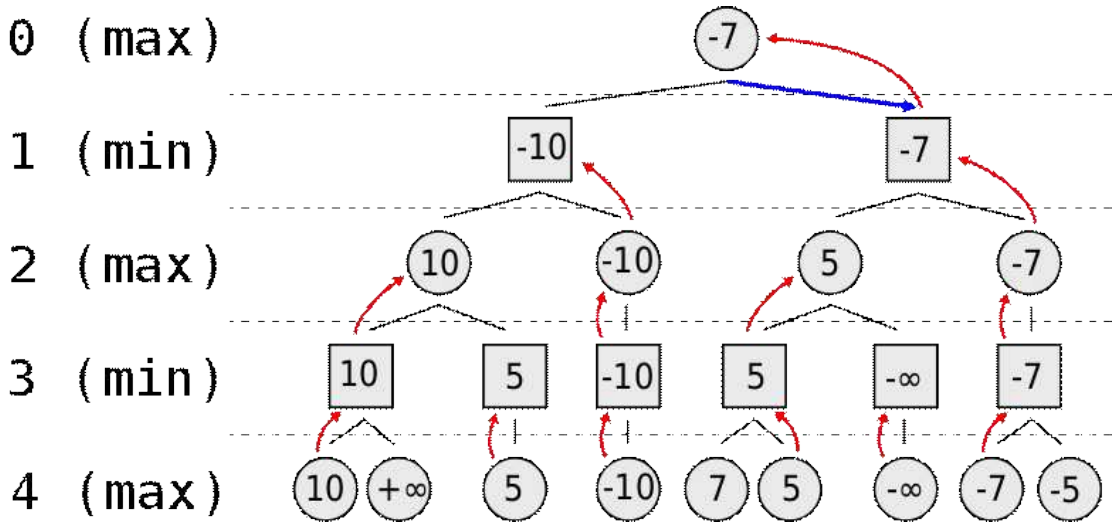
-Make game trees from current status.

-Calculate heuristic of terminal node. Terminal nodes are the ends of the game tree. Value of heuristic means 'how much it's good for computer'. High value is better for computer.

-Starting from terminal, do decision making.

-Because each player does best choice, computer will choose node with max value among possible things. And human will choose min value(=bad for com = good for human).

-By repeating this choice in each depths, find the choice that have max heuristic value.

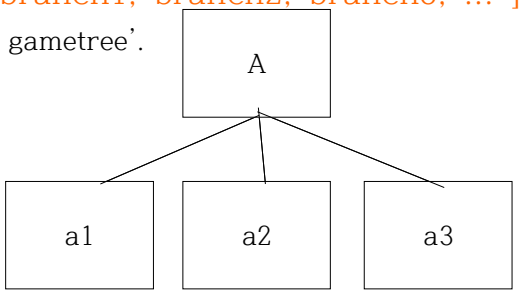


2.Developing minimax for TTT

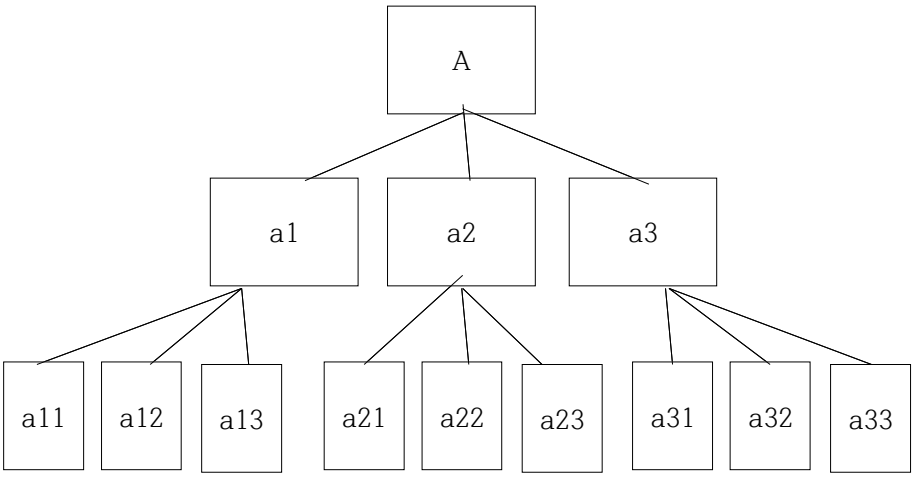
First step is to determine how to define gametree. I'm gonna save it by list.

Format is [original, [branch1, branch2, branch3, ...]]

I'll call it as 'one unit of gametree'.



In this case, gametree is [A, [a1, a2, a3]]. And this continues inside the list.



In this case, game tree is

[A, [[a1, [a11, a12, a13]], [a2, [a21, a22, a23]], [a3, [a31, a32, a33]]]]

In short, there's some gametrees inside the gametree. By doing so, I can store branches and their stem.

```
135 def getgametree(currentboard, letter):
136     if letter == 'O':
137         counterletter = 'X'
138     else:
139         counterletter = 'O'
140     trees = [currentboard,[]]
141     for i in range(1,10):
142         branch = currentboard.copy()
143         if not (isWinner(branch, letter) or isWinner(branch, counterletter)):
144             if branch[i] == ' ': #make move in only empty spaces
145                 makeMove(branch, letter,i)
146                 trees[1].append(branch)
147     for b in range(len(trees[1])):
148         b_branch=trees[1][b]
149         trees[1][b]=getgametree(b_branch, counterletter)
150     return trees
```

This is code of making gametree. Defined as function named 'getgametree'. Parameter is currentboard and computer's letter. Since this program is based on Homework3(Graphical TTT), board is defined as list form such as [' ', 'O', ' ', ' ', 'X', ' ', ' ', ' ', 'O', ' ']. This function returns one unit of gametree.

*136 ~ 139 : Get counterpart's letter.

*140 : Basic form of game tree. Stem is current board and branches will be added soon.

*141 ~ 146 : Making one unit of gametree. Only one stem and its branches(without no more branches).

*142 : Get copy of current board. Because game tree is imaginary process, real board must not be changed. makeMove function(145) really changes the board so copying is needed.

*143 : Check if the game is already over. If it's over, don't make branches. This will reduce processing time.

*144 ~ 146 : makeMove if that space is empty. And after each moves, store the result at trees.

*147 ~ 149 : Making branches' branches over and over.

*147 ~ 148 : Repeat for each branches.

*149 : Change the branch into that branches one unit of game tree. By calling gametree function recursively, making the whole gametree is possible. By giving parameter 'counterletter', OX is added on the imaginary board by turns.

Next step is approaching terminal nodes and calculate heuristic values. At first, I tried to find all terminal nodes in the generated gametree. But this could be quite inefficient. So, I determined to evaluate terminal nodes when making game tree. So, I modified gametree function.

```
154    def evaluate(currentboard):
155        if isWinner(currentboard, computerLetter):
156            heuristic = 5
157        elif isWinner(currentboard, playerLetter):
158            heuristic = 0
159        elif isBoardFull(currentboard):
160            heuristic = 2
161        else:
162            heuristic = ' '
163        return heuristic
```

I defined function that calcuate heuristic named ‘evaluate’. Parameter is current board and check wins. If computer won, value is 5 and if lose, 0. Draw is 2. And if the game is not ended, value is ‘ ’. This value will be stored in each branches. Board’s data type was a List of 10 strings. That’s because original developer wanted to define functions with no confuse. But TTT has only 9 spaces. So, one space of list is always empty (index 0). So I’ll store heuristic there.

```
143            if not (isWinner(branch, letter) or isWinner(branch, counterletter)):
144                if branch[i] == ' ': #make move in only empty spaces
145                    makeMove(branch, letter,i)
146                    branch[0]=str(evaluate(branch)) + letter
147                    trees[1].append(branch)
```

And I added code in the line 146. Branches now have their heuristic and what shape was moved. For example, [‘2X’, 'X', 'X', 'X', 'O', 'O', ' ', 'O', 'X', 'O']. And branch that have number in first index means it is terminal node that doesn’t have their branches.

In sun, gametree carrys datas like below.

↓Last moved shape
↓ So, Board now ↓ ← Same rule ↓

[[‘5O’, ‘ ’, ‘O’, ‘X’, ‘O’, …], [[‘2X’, ‘X’, ‘O’, ‘X’, …] , [‘5X’, ‘ ’, ‘O’, …]]]

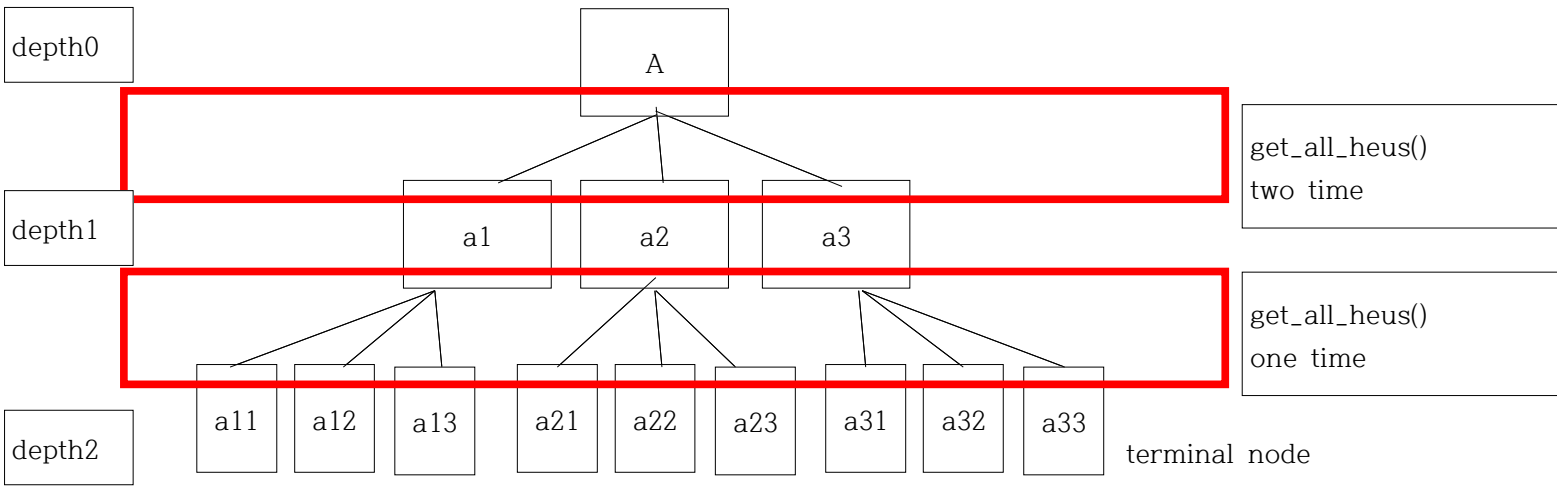
↑ possible next boards

heuristic
5=win
2=draw
0=lose
‘ ’=not yet selected

```
165  def choose(tree):
166      heus=[]
167      for branches in tree[1]:
168          heus.append(branches[0][0][0])
169      if heus != [] and ' ' not in heus:
170          if tree[0][0][1]==computerLetter:#previous turn was player
171              tree[0][0]=str(min(heus)) + computerLetter
172          elif tree[0][0][1]==playerLetter:#previous turn was com
173              tree[0][0]=str(max(heus)) + playerLetter
174      return heus
175
176  def get_all_heus(tree):
177      tree2=tree.copy()
178      if ' ' not in choose(tree2) and len(choose(tree2))>0:
179          choose(tree)
180      else:
181          for subtree in tree[1]:
182              get_all_heus(subtree)
183
```

'choose' function does min max decision. Collect hues from each branches and determine max or min value. And don't choose one if there's any branche that has ' ' heu value. Chosen value will be stored in stem. This function directly changes the gametree and returns heus list.

'get_all_heus' repeats 'choose' over and over inside the gametree. If stem's all branches have heu, stem gets min/max value of heu. If not, approach to its branches and repeat get_all_heus recursively. Somewhen can reach a stem that has terminal nodes only. Because all terminal nodes have heu, that stem can choose heu. One use of this function means getting min / max value from the bottom and store it to upper nodes. By using this function several times, can get value for depth 0. That time equals number of depths and it is same as number of ' 's of current board.



```
184 def get_com_move(gametree):
185     possibles=[]
186     for i in range(len(gametree[1])):#tree(branch[])
187         if gametree[0][0][0] == gametree[1][i][0][0][0]:
188             possibles.append(gametree[1][i][0])#branches
189     print(possibles)
190     commove=possibles[np.random.randint(0,len(possibles))]
191     return commove
192
```

And if computer got heu for depth 0, it's time to select what to do in computer's turn. Several branches have same max heu values. It means whatever computer selects, it doesn't change game result. So computer will collect branches that have max heu and select one of them randomly. And this returned thing will be assigned as current board.

I added line 189 to figure out what's computer's possible moves and if computer can win.

```
395                 else:
396                     if lastclick + 800 < pygame.time.get_ticks():
397                         # Computer's turn.
398                         theBoard[0]=' '+playerLetter
399                         treeofgame=getgametree(theBoard,computerLetter)
400                         for i in range(len(treeofgame[1])):
401                             get_all_heus(treeofgame)
402                         move = get_com_move(treeofgame)
403                         theBoard=move
404                         if computerLetter=='O':
405                             o_sound.play()
406                         else:
407                             x_sound.play()
408                         if isWinner(theBoard, computerLetter):
409                             winner=turn
410                             one_game_end = True
411                         else:
412                             if isBoardFull(theBoard):
413                                 winner='draw'
414                                 one_game_end = True
415                             else:
416                                 turn = 'player'
```

In the actual game running code, I only changed line 398 ~ 403.

*398 : Mark the last moved shape in theBoard. That's because min/max is selected by last moved shape.

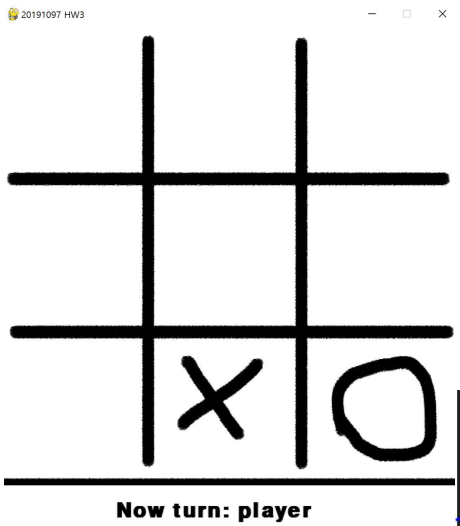
*399 : Generate gametree.

*400 ~ 401 : derive hue from terminal to depth 0.

*402 : Select random move and update theBoard.

3.Run TTT and check it works well.

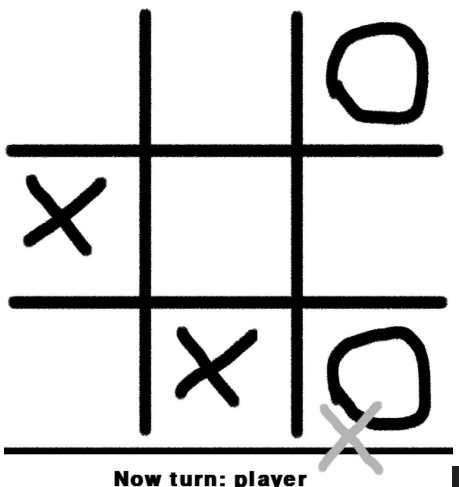
Now let's run and check if it works.



7	8	9
4	5	6
1	2	3

```
[['20', 'O', 'X', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '],
 ['20', ' ', 'X', 'O', ' ', ' ', ' ', ' ', ' ', ' ', ' '],
 ['20', ' ', 'X', ' ', ' ', 'O', ' ', ' ', ' ', ' ', ' '],
 ['20', ' ', 'X', ' ', ' ', ' ', ' ', ' ', 'O', ' ', ' ']]
```

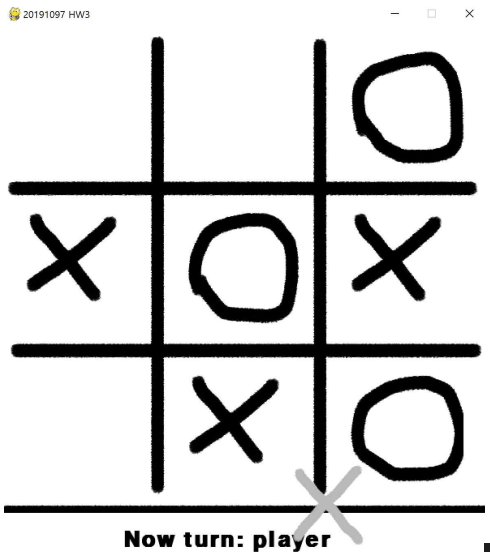
The first turn was me and I'm X. I moved to 2. Then, computer get 4 possible roots of draw(heu value 2), and selected 3. That means my best result is draw. I cannot win computer forever.



7	8	9
4	5	6
1	2	3

```
[['50', ' ', 'X', 'O', 'X', ' ', ' ', ' ', ' ', 'O']]
```

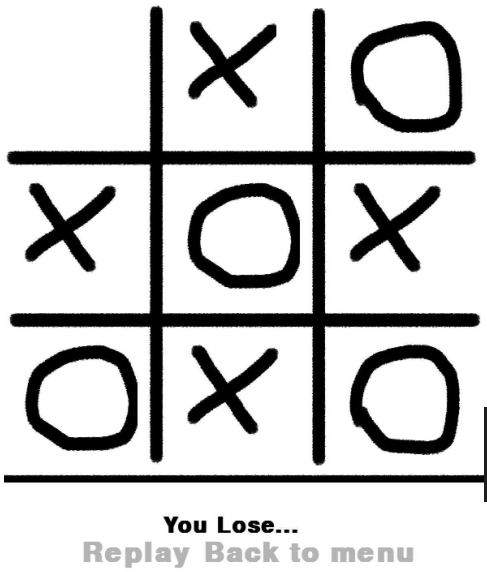
Next, I moved to 4, and computer found only one way to win(heu 5).



7	8	9
4	5	6
1	2	3

```
[['50', ' ', 'X', 'O', 'X', 'O', 'X', ' ', ' ', 'O']]
```

I defended by move to 6. Computer found 1 way to win, and selected to move to 5. If computer didn't move to 5, then I'll move to 5 and computer cannot win.



7	8	9
4	5	6
1	2	3

```
[['50', 'O', 'X', 'O', 'X', 'O', 'X', ' ', 'X', 'O'],
 ['50', ' ', 'X', 'O', 'X', 'O', 'X', 'O', 'X', 'O']]
```

I moved to 8, and computer found 2 ways to win. Computer selected to move to 1 and computer won.