For Ultimate Tic Tac Toe, the BEST implementation uses the minimax algorithm with alpha-beta pruning. The heuristic looks at the overall board and each of the sub boards. The way for evaluating a particular configuration (i.e. an arrangement of X and Os) is the same, and that function is score\_subboard(). The score is a linear combination of weights for corners and center pieces with the difference in the number possessed by each player. If a player has captured three corners, then the count is doubled. A victory returns a value of winning player \* VICTORY. The value of VICTORY chosen to be about 2 or 3 times larger than possible mid-game states. A victory in the overall game board returns a large number (100000)

To determine the score of the entire board, the score of the overall board configuration of X’s and O’s is multiplied by 10 and then added to 0.2 \* (sum of weights \* score of sub-board). The score of each sub-board is multiplied by a weight for the location of the sub-board. The AGGRESSIVE agent evaluates the score of all possible next moves and chooses the move that would give it the highest score. The RANDOM agent returns a random valid score.

Heuristic

def scoreBoard(*board*,*player*):

"""

1 - player win

0 - Draw

-1 - player lost

# higher score indicates better outcome for passed player

"""

other = "O"

*if* *player* == "O":

other = "X"

big\_board = getbigBoard(*board*,*player*)

endGame = gameOver(big\_board,*player*)

*if* endGame is not None:

*if* endGame > 0:

*return* 100000

*elif* endGame == 0:

*return* 100\*(big\_board.count(*player*)-big\_board.count(other))

*else*:

*return* -100000

*else*:

*# advantage for: having a two in a row on a big board, getting the center, getting corner squares*

*# small board weighting: the completeness of the board(number open/9 \* score of small board)*

*# total score: large board score+score of each small board*

big\_Score = score\_subboard(big\_board,*player*)\*10

smallScore = 0

weights = [10,1,10,

1,20,1,

10,1,10] *# corners have weight of 10, points, center is 20, edges are 1*

*for* b *in* range(len(*board*)):

smallScore += score\_subboard(*board*[b],*player*) \* weights[b]

*return* big\_Score + (smallScore)\*0.2

def score\_subboard(*board*,*player*):

VICTORY = 500

CORNER = 25

CENTER = 30

TWO = 20

other = "O"

*if* *player* == "O":

other = "X"

end = gameOver(*board*,*player*)

*if* end is not None:

*return* end \* VICTORY

*else*:

score = 0

myTwo = 0

otherTwo = 0

*for* i *in* range(3):

*if* (seq:=*board*[i:i+3]) == *player*+*player*+"." or seq == "."+*player*+*player*:

myTwo += 1

*if* seq == other+other+"." or seq == "."+other+other:

otherTwo += 1

*# columns*

*for* col *in* range(3):

seq = *board*[col:col+3\*3:3]

*if* seq == *player*+*player*+"." or seq == "."+*player*+*player*:

myTwo += 1

*if* seq == other+other+"." or seq == "."+other+other:

otherTwo += 1

*# diagonals*

posDiag = *board*[0]+*board*[4]+*board*[8]

negDiag = *board*[2]+*board*[4]+*board*[6]

*if* posDiag == *player*+*player* + "." or posDiag == "."+*player*+*player*:

myTwo += 1

*if* posDiag == other+other+"." or posDiag == "."+other+other:

otherTwo += 1

*if* negDiag == *player*+*player* + "." or negDiag == "."+*player*+*player*:

myTwo += 1

*if* negDiag == other+other+"." or negDiag == "."+other+other:

otherTwo += 1

score += TWO\*(myTwo-otherTwo)

*# look for corners*

myCorners = 0

otherCorners = 0

*for* c *in* [0,2,6,8]:

*if* *board*[c] == *player*:

myCorners += 1

*elif* *board*[c] == other:

otherCorners+= 1

*if* myCorners == 3:

myCorners \*= 2

*if* otherCorners == 3:

otherCorners \*= 2

score += CORNER\*(myCorners - otherCorners)

*# add weighting for centers*

*if* *board*[len(*board*)//2] == *player*:

score += CENTER

*elif* *board*[len(*board*)//2] == other:

score -= CENTER

*return* score