

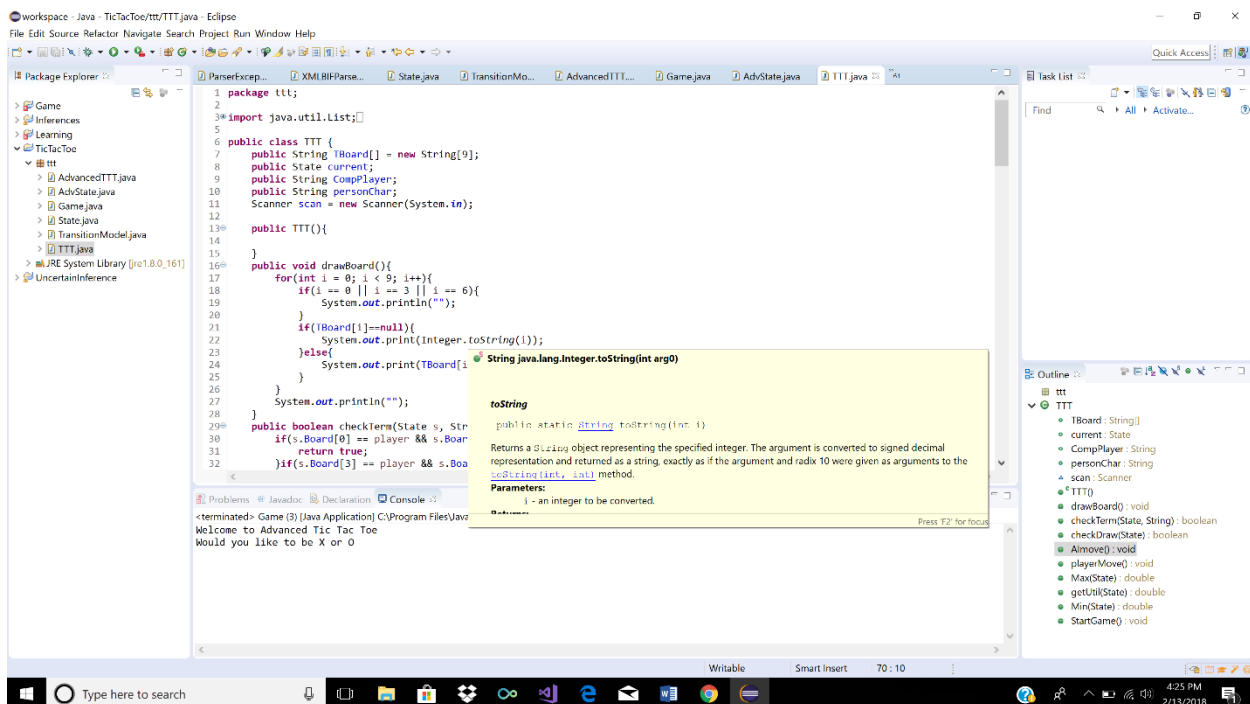
Thomas Burt

CSC 242

February 13, 2018

Writup

To start off the project, the first thing I had to take into account was how to represent the basic tic tac toe problem. My first thought was that I needed to implement the right data structures for the board, the basic game, states, and actions. First, I started off by making a State class, which the coding is for a page or two below. This just contained the String name of the player and a String array of length 9, which represented a possible board combination. After, I then made a TTT class for Basic Tic-tac-toe. This contains the main board and functions that help with the basic tic tac toe program. The following are my global variables for the class.



```
1 package ttt;
2
3 import java.util.List;
4
5
6 public class TTT {
7     public String TBoard[] = new String[9];
8     public State current;
9     public String CompPlayer;
10    public String personChar;
11    Scanner scan = new Scanner(System.in);
12
13    public TTT(){
14    }
15
16    public void drawBoard(){
17        for(int i = 0; i < 9; i++){
18            if(i == 0 || i == 3 || i == 6){
19                System.out.println("");
20            }
21            if(TBoard[i] == null){
22                System.out.print(Integer.toString(i));
23            }else{
24                System.out.print(TBoard[i]);
25            }
26        }
27        System.out.println("");
28    }
29    public boolean checkTerm(State s, String player) {
30        if(s.Board[0] == player && s.Board[3] == player && s.Board[6] == player) return true;
31        if(s.Board[1] == player && s.Board[4] == player && s.Board[7] == player) return true;
32        if(s.Board[2] == player && s.Board[5] == player && s.Board[8] == player) return true;
33        if(s.Board[0] == player && s.Board[4] == player && s.Board[8] == player) return true;
34        if(s.Board[2] == player && s.Board[4] == player && s.Board[6] == player) return true;
35        if(s.Board[0] == player && s.Board[6] == player && s.Board[8] == player) return true;
36        if(s.Board[2] == player && s.Board[6] == player && s.Board[4] == player) return true;
37        if(s.Board[4] == player && s.Board[0] == player && s.Board[6] == player) return true;
38        if(s.Board[4] == player && s.Board[2] == player && s.Board[6] == player) return true;
39        if(s.Board[6] == player && s.Board[0] == player && s.Board[2] == player) return true;
40        if(s.Board[6] == player && s.Board[2] == player && s.Board[4] == player) return true;
41        if(s.Board[0] == player && s.Board[4] == player && s.Board[2] == player) return true;
42        if(s.Board[4] == player && s.Board[6] == player && s.Board[0] == player) return true;
43        if(s.Board[2] == player && s.Board[4] == player && s.Board[0] == player) return true;
44        if(s.Board[2] == player && s.Board[6] == player && s.Board[4] == player) return true;
45        if(s.Board[4] == player && s.Board[0] == player && s.Board[2] == player) return true;
46        if(s.Board[6] == player && s.Board[0] == player && s.Board[4] == player) return true;
47        if(s.Board[6] == player && s.Board[4] == player && s.Board[2] == player) return true;
48        return false;
49    }
50    public boolean checkDraw(State s) {
51        for(int i = 0; i < 9; i++){
52            if(s.Board[i] == null) return false;
53        }
54        return true;
55    }
56    public void playerMove() {
57        int move = -1;
58        while(move < 0 || move > 8) {
59            move = scan.nextInt();
60        }
61        if(TBoard[move] != null) {
62            System.out.println("That spot is already taken. Please choose another spot.");
63            playerMove();
64        } else {
65            TBoard[move] = personChar;
66        }
67    }
68    public void compMove() {
69        // TODO: Implement computer move logic
70    }
71    public void startGame() {
72        current = new State("Player", new String[9]);
73        CompPlayer = "Computer";
74        personChar = "X";
75        scan = new Scanner(System.in);
76        drawBoard();
77    }
78}
```

toString
public static String toString(int i)
Returns a String object representing the specified integer. The argument is converted to signed decimal representation and returned as a string, exactly as if the argument and radix 10 were given as arguments to the `toString(int, int)` method.
Parameters:
i - an integer to be converted.

Outline:
TTT
TBoard: String[]
current: State
CompPlayer: String
personChar: String
scan: Scanner
TTT()
drawBoard(): void
checkTerm(State, String): boolean
checkDraw(State): boolean
playerMove(): void
compMove(): void
startGame(): void

I thought about making a search tree at first that way I could try to make it all at once so that it didn't need to use an algorithm to search for the best move. However, I quickly found out that this was not the way to go. Following the MiniMax algorithm from the book and using the basic general idea of Professor Ferguson's state space search algorithm, or best algorithm ever. The idea was to take the initial state expand it using depth first search, meaning to expand the initial state, then its field, and that child's first child, etc. This would then setup a layout of the search tree without creating it thus saving space efficiency. This stopped me from making a search tree, but instead I made a State class in my project. The State class would consist of the board, player

symbol (meaning which player's turn it was) and the successors of the state. I thought about making a transition model class, which was mentioned in the project instructions, but I felt that it was overdoing it and that, if needed, I could easily implement it later. I saw no need for the transition model. Following this I made my MiniMax algorithm functions in the TTT class called Min and Max. These functions had helper functions like checkTerm(state s, String player) which checked if the state was a terminal state with the specified player. This helped a lot with the getUtil functions as well.

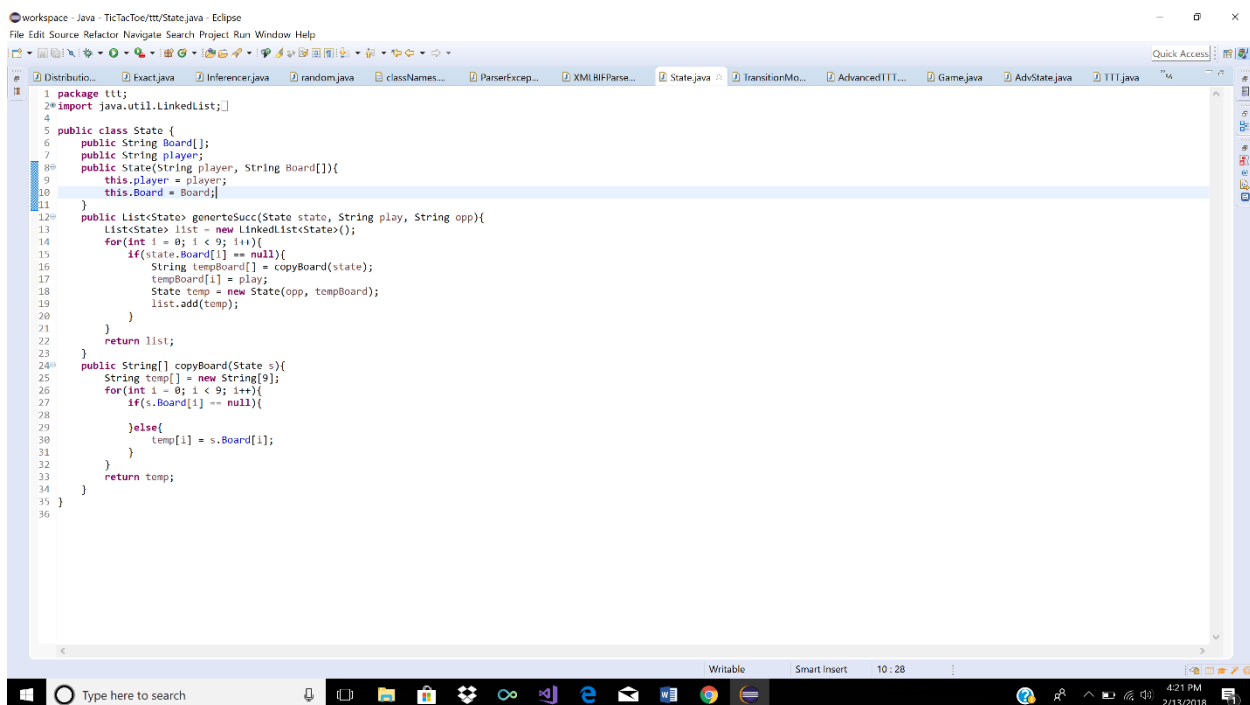
Once I was done with the MiniMax algorithm I started testing it. At first it was being completely crazy. It would output in the first move:

OOOOOO

OOOO

XOO

This was very funky, but I found out it was my generateSucc function in my State class which was causing it. I was referencing the state whose successors I was trying to generate; thus, its table was changed when exploring the options. So, I made a new function called copyBoard(), which made a brand new board all copied from the root State's board. This solved the problem with the crazy board. The changes and present State class coding is in the picture below.



```
1 package ttt;
2 import java.util.LinkedList;
3
4
5 public class State {
6     public String Board[];
7     public String player;
8     public State(String player, String Board[]){
9         this.player = player;
10        this.Board = Board;
11    }
12
13    public List<State> generateSucc(State state, String play, String opp){
14        List<State> list = new LinkedList<State>();
15        for(int i = 0; i < 9; i++){
16            if(state.Board[i] == null){
17                String tempBoard[] = copyBoard(state);
18                tempBoard[i] = play;
19                State temp = new State(opp, tempBoard);
20                list.add(temp);
21            }
22        }
23        return list;
24    }
25
26    public String[] copyBoard(State s){
27        String temp[] = new String[9];
28        for(int i = 0; i < 9; i++){
29            if(s.Board[i] == null){
30            }else{
31                temp[i] = s.Board[i];
32            }
33        }
34        return temp;
35    }
36 }
```

The next problem I ran into was that my MiniMax was not stopping easy wins on my part. This was an easy fix because I realized I forgot to implement function that checks if there is a draw. So, I made a checkDraw function, added it to my Min, Max and, most importantly, getUtil function so that the computer would go for a draw rather than seeking out a win when I am clearly about to win. This solve the problem and the program was almost perfect, however, I was still able to use

strategies that people players don't see, but the program should. An example of a board where I played this strategy is:

From

O2X

O56

78X

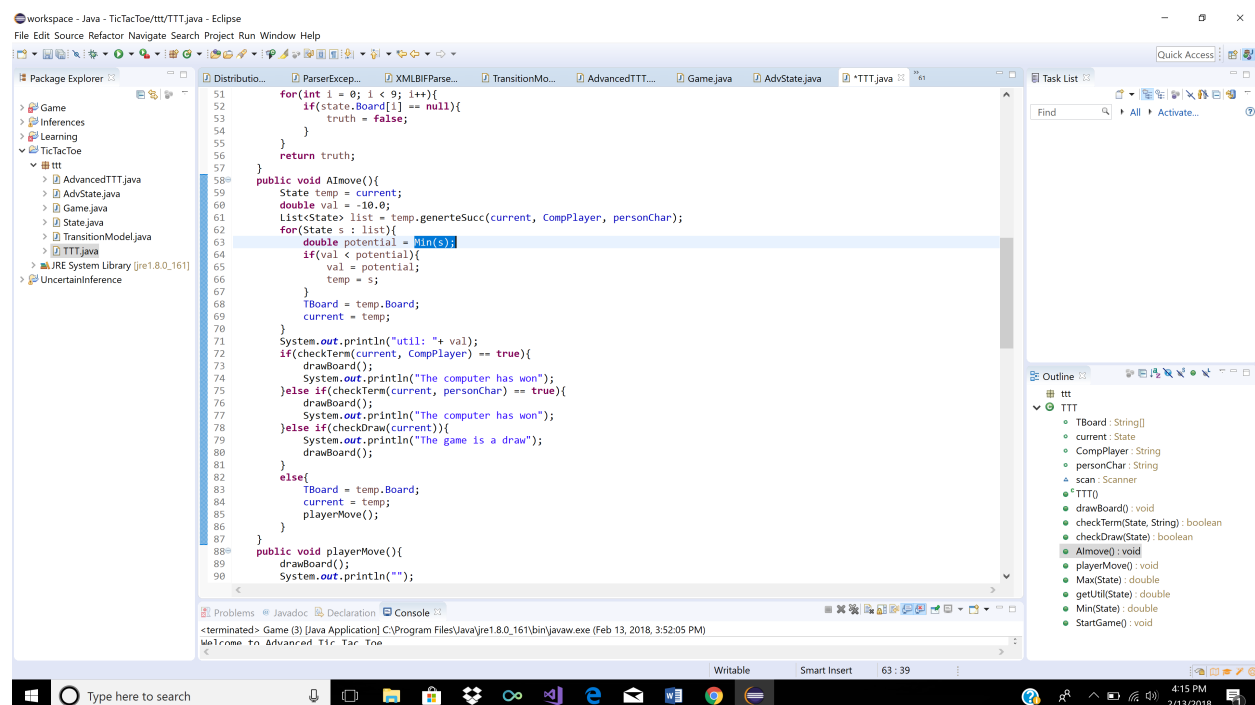
to

O2X

O56

X8X

I am the X in this instance and this got the computer stuck because there is no conceivable way the computer could stop my win; thus, the program was not ideal. I took a break for a couple days and then looked back into it and found something that I did wrong. In the function `AImove()`, the computer would take the max value of the max function called on all successors. This is incorrect because the algorithm takes the max value of the min value function called on all successors. I fixed this error by changing the max call to the min call which is highlighted in the picture below.



After this error was solved my Basic Tic Tac Toe program was complete here the first picture below is the program never letting me win. The second picture is the program stopping me from

doing my strategy that was beating it before. Please note that this is not the final way I printed out the project but shows that my program is at its peak in terms of MiniMax.

```
Command Prompt
Microsoft Windows [Version 10.0.16299.125]
(c) 2017 Microsoft Corporation. All rights reserved.

C:\Users\tburt>cd workspace
C:\Users\tburt\workspace>cd TicTacToe
C:\Users\tburt\workspace\TicTacToe>java ttt.Game
Welcome to Tic Tac Toe
Would you like to be X or O
O

X12
345
678

Where would you like to go next
4

XX2
305
678

Where would you like to go next
2

XXO
305
X78

Where would you like to go next
3

XXO
00X
X78

Where would you like to go next
8
The game is a draw

XXO
00X
XXO

C:\Users\tburt\workspace\TicTacToe>
```

```
Command Prompt
C:\Users\tburt\workspace\TicTacToe>java ttt.Game
Welcome to Tic Tac Toe
Would you like to be X or O
X

012
345
678

Where would you like to go next
2

01X
305
678

Where would you like to go next
6

00X
305
X78

Where would you like to go next
7

00X
305
XXO

Where would you like to go next
0

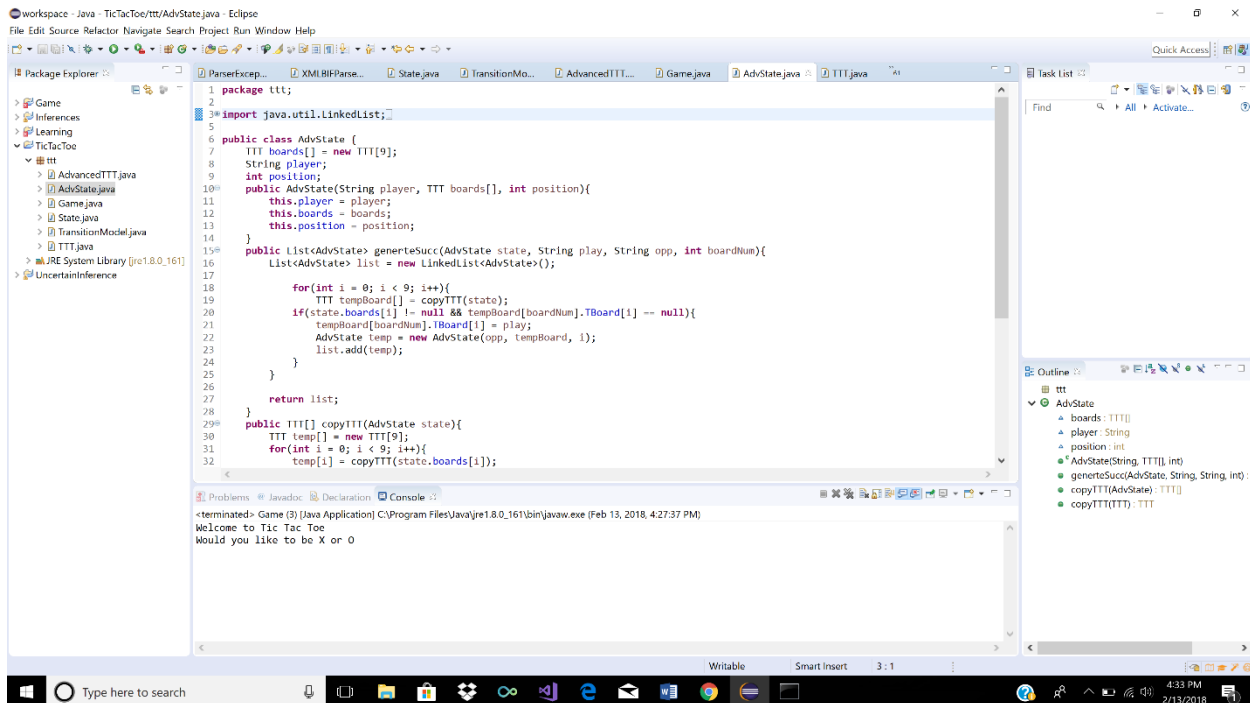
XXO
005
XXO

Where would you like to go next
5
The game is a draw

XXO
00X
XXO

C:\Users\tburt\workspace\TicTacToe>
```

For Advanced Tic-tac-toe, the process of representing the 9 tic tac toe boards was easy. I reused the TTT class that was used for Basic Tic-tac-toe and made a length 9 array of TTT[]. This gave me my 9 boards, which was stored as a global variable. I also created an AdvState class which had the following global variables:



On top of the length 9 TTT[] I also had String player and int position. At first, I did not have int position in the code at first, but will explain a page later why I needed to add that in.

Once the representation was done, I needed to then implement a player move function, AI move function, and the Heuristic MiniMax functions. Following what I did in the basic Tic-tac-toe class, I essentially had the same function just stopping once the search hit a depth of four or hit a terminal state. This was fairly easy and after looking at the book, easily implemented alpha-beta pruning into the code as well. This is where my first problem came into play however. After the AI made its first move, it never made a move after when I made my second, third, fourth moves, etc. It was only doing one move the entire game at Board 0, position 0. I found the error inside my generateSucc function. If you see the parameter or the if function, I added in tempBoard[boardNum].Tboard[i] == null. Once I added that in, the problem was solved. The thing was, the generateSucc function was counting the current state itself as a successor thus continually choosing it. Keep in mind, at this point in the project I did not have my heuristic function down, so I was mainly just looking to see if the Computer was making moves at least. It was after solving this error.

Afterwards, I made my heuristic function which would add up all the heuristics of every board. Each board heuristic would be as follows:

- Adding up all possible ways to win for the computer
 - o Every way with two symbols already positioned there would add 10 to the sum of the board heuristic, and every way with 1 symbol in that way thus far would add 5 to the sum. So, if there are 3 ways to win with two symbols in each way, the heuristic value of the board would be 30

- Once the value of a table for the computer was summed up, the program would then call the same function but instead value of the table for the player and subtract it from the value that was just summed up for the computer.
- If $g(n)$ was the function and $f(n)$ was the util of the board then the $f(n)$ would be
 - o $g(\text{CompChar}) - g(\text{playerChar}) = f(n)$
- a screen shot of both the function that takes each board's individual utility for a player and the function that computes the total utility of the state are shown below

workspace - Java - TicTacToe/ttt/AdvancedTTT.java - Eclipse

File Edit Source Refactor Navigate Search Project Run Window Help

157 }
158 return draw;
159 }
160 public double getUtil(AdvState s){
161 double util = 0.0;
162 if(checkAdvTerm(s, compChar)){
163 util = 1000.0;
164 }
165 else if(checkAdvTerm(s, playerChar)){
166 util = -1000.0;
167 }
168 else if(checkAdvDraw(s)){
169 util = 0.0;
170 }
171 else{
172 TTT t[] = s.boards;
173 for(int i = 0; i < 9; i++){
174 util += boardVal(t[i], compChar, playerChar);
175 util -= boardVal(t[i], playerChar, compChar);
176 }
177 }
178 return util;
179 }
180 public double boardVal(TTT t, String max, String opponent){
181 double sum = 0.0;
182 if(t.TBoard[0] == max && t.TBoard[1] != opponent && t.TBoard[2] != opponent){
183 sum += 5.0;
184 }
185 if(t.TBoard[0] == max && t.TBoard[3] != opponent && t.TBoard[6] != opponent){
186 sum += 5.0;
187 }
188 if(t.TBoard[0] == max && t.TBoard[4] != opponent && t.TBoard[8] != opponent){
189 sum += 5.0;
190 }
191 if(t.TBoard[1] == max && t.TBoard[4] != opponent && t.TBoard[7] != opponent){
192 sum += 5.0;
193 }
194 if(t.TBoard[2] == max && t.TBoard[5] != opponent && t.TBoard[8] != opponent){
195 sum += 5.0;
196 }
197 if(t.TBoard[2] == max && t.TBoard[4] != opponent && t.TBoard[6] != opponent){
198 sum += 5.0;
199 }
200 if(t.TBoard[3] == max && t.TBoard[4] != opponent && t.TBoard[5] != opponent){
201 sum += 5.0;
202 }
203 if(t.TBoard[6] == max && t.TBoard[7] != opponent && t.TBoard[8] != opponent){
204 sum += 5.0;
205 }
206 if(t.TBoard[0] == max && t.TBoard[1] == max && t.TBoard[2] != opponent){
207 sum += 10.0;
208 }
209 if(t.TBoard[0] == max && t.TBoard[3] == max && t.TBoard[6] != opponent){
210 sum += 10.0;
211 }
212 if(t.TBoard[0] == max && t.TBoard[4] == max && t.TBoard[8] != opponent){
213 sum += 10.0;
214 }
215 if(t.TBoard[1] == max && t.TBoard[4] == max && t.TBoard[7] != opponent){
216 sum += 10.0;
217 }
218 if(t.TBoard[2] == max && t.TBoard[5] == max && t.TBoard[8] != opponent){
219 sum += 10.0;
220 }
221 if(t.TBoard[2] == max && t.TBoard[4] == max && t.TBoard[6] != opponent){
222 sum += 10.0;
223 }
224 if(t.TBoard[3] == max && t.TBoard[4] == max && t.TBoard[5] != opponent){
225 sum += 10.0;
226 }
227 if(t.TBoard[6] == max && t.TBoard[7] == max && t.TBoard[8] != opponent){
228 sum += 10.0;
229 }
230 if(t.TBoard[0] == max && t.TBoard[1] != opponent && t.TBoard[2] == max){
231 sum += 10.0;
232 }
233 if(t.TBoard[0] == max && t.TBoard[3] != opponent && t.TBoard[6] == max){
234 sum += 10.0;
235 }
236 if(t.TBoard[0] == max && t.TBoard[4] != opponent && t.TBoard[8] == max){
237 sum += 10.0;
238 }
239 if(t.TBoard[1] == max && t.TBoard[4] != opponent && t.TBoard[7] == max){
240 sum += 10.0;
241 }
242 if(t.TBoard[2] == max && t.TBoard[5] != opponent && t.TBoard[8] == max){
243 sum += 10.0;
244 }
245 if(t.TBoard[2] == max && t.TBoard[4] == max && t.TBoard[6] != opponent){
246 sum += 10.0;
247 }
248 if(t.TBoard[3] == max && t.TBoard[4] == max && t.TBoard[5] != opponent){
249 sum += 10.0;
250 }
251 if(t.TBoard[6] == max && t.TBoard[7] == max && t.TBoard[8] != opponent){
252 sum += 10.0;
253 }
254 if(t.TBoard[0] == max && t.TBoard[1] != opponent && t.TBoard[2] == max){
255 sum += 10.0;
256 }
257 if(t.TBoard[0] == max && t.TBoard[3] != opponent && t.TBoard[6] == max){
258 sum += 10.0;
259 }
260 if(t.TBoard[0] == max && t.TBoard[4] != opponent && t.TBoard[8] == max){
261 sum += 10.0;
262 }
263 if(t.TBoard[1] == max && t.TBoard[4] != opponent && t.TBoard[7] == max){
264 sum += 10.0;
265 }
266 if(t.TBoard[2] == max && t.TBoard[5] != opponent && t.TBoard[8] == max){
267 sum += 10.0;
268 }
269 if(t.TBoard[2] == max && t.TBoard[4] == max && t.TBoard[6] != opponent){
270 sum += 10.0;
271 }
272 if(t.TBoard[3] == max && t.TBoard[4] == max && t.TBoard[5] != opponent){
273 sum += 10.0;
274 }
275 if(t.TBoard[6] == max && t.TBoard[7] == max && t.TBoard[8] != opponent){
276 sum += 10.0;
277 }
278 if(t.TBoard[0] == max && t.TBoard[1] != opponent && t.TBoard[2] == max){
279 sum += 10.0;
280 }
281 if(t.TBoard[0] == max && t.TBoard[3] != opponent && t.TBoard[6] == max){
282 sum += 10.0;
283 }
284 if(t.TBoard[0] == max && t.TBoard[4] != opponent && t.TBoard[8] == max){
285 sum += 10.0;
286 }
287 if(t.TBoard[1] == max && t.TBoard[4] != opponent && t.TBoard[7] == max){
288 sum += 10.0;
289 }
289 }
290 return sum;
291 }

Updates Available
Updates are available for your software.
Click to review and install updates.
Set up Reminder options

workspace - Java - TicTacToe/ttt/AdvancedTTT.java - Eclipse

File Edit Source Refactor Navigate Search Project Run Window Help

178 }
179 public double boardVal(TTT t, String max, String opponent){
180 double sum = 0.0;
181 if(t.TBoard[0] == max && t.TBoard[1] != opponent && t.TBoard[2] != opponent){
182 sum += 5.0;
183 }
184 if(t.TBoard[0] == max && t.TBoard[3] != opponent && t.TBoard[6] != opponent){
185 sum += 5.0;
186 }
187 if(t.TBoard[0] == max && t.TBoard[4] != opponent && t.TBoard[8] != opponent){
188 sum += 5.0;
189 }
190 if(t.TBoard[1] == max && t.TBoard[4] != opponent && t.TBoard[7] != opponent){
191 sum += 5.0;
192 }
193 if(t.TBoard[2] == max && t.TBoard[5] != opponent && t.TBoard[8] != opponent){
194 sum += 5.0;
195 }
196 if(t.TBoard[2] == max && t.TBoard[4] != opponent && t.TBoard[6] != opponent){
197 sum += 5.0;
198 }
199 if(t.TBoard[3] == max && t.TBoard[4] != opponent && t.TBoard[5] != opponent){
200 sum += 5.0;
201 }
202 if(t.TBoard[6] == max && t.TBoard[7] != opponent && t.TBoard[8] != opponent){
203 sum += 5.0;
204 }
205 if(t.TBoard[0] == max && t.TBoard[1] == max && t.TBoard[2] != opponent){
206 sum += 10.0;
207 }
208 if(t.TBoard[0] == max && t.TBoard[3] == max && t.TBoard[6] != opponent){
209 sum += 10.0;
210 }
211 if(t.TBoard[0] == max && t.TBoard[4] == max && t.TBoard[8] != opponent){
212 sum += 10.0;
213 }
214 if(t.TBoard[1] == max && t.TBoard[4] == max && t.TBoard[7] != opponent){
215 sum += 10.0;
216 }
217 if(t.TBoard[2] == max && t.TBoard[5] == max && t.TBoard[8] != opponent){
218 sum += 10.0;
219 }
220 if(t.TBoard[2] == max && t.TBoard[4] == max && t.TBoard[6] != opponent){
221 sum += 10.0;
222 }
223 if(t.TBoard[3] == max && t.TBoard[4] == max && t.TBoard[5] != opponent){
224 sum += 10.0;
225 }
226 if(t.TBoard[6] == max && t.TBoard[7] == max && t.TBoard[8] != opponent){
227 sum += 10.0;
228 }
229 if(t.TBoard[0] == max && t.TBoard[1] != opponent && t.TBoard[2] == max){
230 sum += 10.0;
231 }
232 if(t.TBoard[0] == max && t.TBoard[3] != opponent && t.TBoard[6] == max){
233 sum += 10.0;
234 }
235 if(t.TBoard[0] == max && t.TBoard[4] != opponent && t.TBoard[8] == max){
236 sum += 10.0;
237 }
238 if(t.TBoard[1] == max && t.TBoard[4] != opponent && t.TBoard[7] == max){
239 sum += 10.0;
240 }
241 if(t.TBoard[2] == max && t.TBoard[5] != opponent && t.TBoard[8] == max){
242 sum += 10.0;
243 }
244 if(t.TBoard[2] == max && t.TBoard[4] == max && t.TBoard[6] != opponent){
245 sum += 10.0;
246 }
247 if(t.TBoard[3] == max && t.TBoard[4] == max && t.TBoard[5] != opponent){
248 sum += 10.0;
249 }
250 if(t.TBoard[6] == max && t.TBoard[7] == max && t.TBoard[8] != opponent){
251 sum += 10.0;
252 }
253 if(t.TBoard[0] == max && t.TBoard[1] != opponent && t.TBoard[2] == max){
254 sum += 10.0;
255 }
256 if(t.TBoard[0] == max && t.TBoard[3] != opponent && t.TBoard[6] == max){
257 sum += 10.0;
258 }
259 if(t.TBoard[0] == max && t.TBoard[4] != opponent && t.TBoard[8] == max){
260 sum += 10.0;
261 }
262 if(t.TBoard[1] == max && t.TBoard[4] != opponent && t.TBoard[7] == max){
263 sum += 10.0;
264 }
265 if(t.TBoard[2] == max && t.TBoard[5] != opponent && t.TBoard[8] == max){
266 sum += 10.0;
267 }
268 if(t.TBoard[2] == max && t.TBoard[4] == max && t.TBoard[6] != opponent){
269 sum += 10.0;
270 }
271 if(t.TBoard[3] == max && t.TBoard[4] == max && t.TBoard[5] != opponent){
272 sum += 10.0;
273 }
274 if(t.TBoard[6] == max && t.TBoard[7] == max && t.TBoard[8] != opponent){
275 sum += 10.0;
276 }
277 if(t.TBoard[0] == max && t.TBoard[1] != opponent && t.TBoard[2] == max){
278 sum += 10.0;
279 }
280 if(t.TBoard[0] == max && t.TBoard[3] != opponent && t.TBoard[6] == max){
281 sum += 10.0;
282 }
283 if(t.TBoard[0] == max && t.TBoard[4] != opponent && t.TBoard[8] == max){
284 sum += 10.0;
285 }
286 if(t.TBoard[1] == max && t.TBoard[4] != opponent && t.TBoard[7] == max){
287 sum += 10.0;
288 }
289 if(t.TBoard[2] == max && t.TBoard[5] != opponent && t.TBoard[8] == max){
290 sum += 10.0;
291 }
292 if(t.TBoard[2] == max && t.TBoard[4] == max && t.TBoard[6] != opponent){
293 sum += 10.0;
294 }
295 if(t.TBoard[3] == max && t.TBoard[4] == max && t.TBoard[5] != opponent){
296 sum += 10.0;
297 }
298 if(t.TBoard[6] == max && t.TBoard[7] == max && t.TBoard[8] != opponent){
299 sum += 10.0;
300 }
301 if(t.TBoard[0] == max && t.TBoard[1] != opponent && t.TBoard[2] == max){
302 sum += 10.0;
303 }
304 if(t.TBoard[0] == max && t.TBoard[3] != opponent && t.TBoard[6] == max){
305 sum += 10.0;
306 }
307 if(t.TBoard[0] == max && t.TBoard[4] != opponent && t.TBoard[8] == max){
308 sum += 10.0;
309 }
309 }
310 return sum;
311 }

Updates Available
Updates are available for your software.
Click to review and install updates.
Set up Reminder options

```
190     sum += 5.0;
191     }if(t.TBoard[2] == max && t.TBoard[4] != opponent && t.TBoard[6] != opponent){
192         sum += 5.0;
193     }if(t.TBoard[3] == max && t.TBoard[4] != opponent && t.TBoard[5] != opponent){
194         sum += 5.0;
195     }if(t.TBoard[6] == max && t.TBoard[7] != opponent && t.TBoard[8] != opponent){
196         sum += 5.0;
197     }
198     if(t.TBoard[0] == max && t.TBoard[1] == max && t.TBoard[2] != opponent){
199         sum += 10.0;
200     }if(t.TBoard[0] == max && t.TBoard[3] == max && t.TBoard[6] != opponent){
201         sum += 10.0;
202     }if(t.TBoard[0] == max && t.TBoard[4] == max && t.TBoard[8] != opponent){
203         sum += 10.0;
204     }if(t.TBoard[1] == max && t.TBoard[4] == max && t.TBoard[7] != opponent){
205         sum += 10.0;
206     }if(t.TBoard[2] == max && t.TBoard[5] == max && t.TBoard[8] != opponent){
207         sum += 10.0;
208     }if(t.TBoard[2] == max && t.TBoard[4] == max && t.TBoard[6] != opponent){
209         sum += 10.0;
210     }if(t.TBoard[3] == max && t.TBoard[4] == max && t.TBoard[5] != opponent){
211         sum += 10.0;
212     }if(t.TBoard[6] == max && t.TBoard[7] == max && t.TBoard[8] != opponent){
213         sum += 10.0;
214     }
215     if(t.TBoard[0] == max && t.TBoard[1] != opponent && t.TBoard[2] == max){
216         sum += 10.0;
217     }if(t.TBoard[0] == max && t.TBoard[3] != opponent && t.TBoard[6] == max){
218         sum += 10.0;
219     }if(t.TBoard[0] == max && t.TBoard[4] != opponent && t.TBoard[8] == max){
220         sum += 10.0;
221     }if(t.TBoard[1] == max && t.TBoard[4] != opponent && t.TBoard[7] == max){
222         sum += 10.0;
223     }if(t.TBoard[2] == max && t.TBoard[5] != opponent && t.TBoard[8] == max){
224         sum += 10.0;
225     }if(t.TBoard[2] == max && t.TBoard[4] != opponent && t.TBoard[6] == max){
226         sum += 10.0;
227     }if(t.TBoard[3] == max && t.TBoard[4] != opponent && t.TBoard[5] == max){
228         sum += 10.0;
229     }if(t.TBoard[6] == max && t.TBoard[7] != opponent && t.TBoard[8] == max){
230         sum += 10.0;
231     }
232     return sum;
233 }
234 }
```

These heuristics were quite good. The computer never let me go to a board where I would be able to win, it always won when the possibility was given, and played above how I thought it would. Because of this I never tried any other heuristics just because this one was a pretty good one. I still have never beaten the computer after implementing this heuristic in.

After this was done I ran into a problem with the speed. The speed was terrible when I was running this at first. It took 5-10 minutes to complete the first move and I knew there was something wrong with the efficiency. I tried to make the depth limit at to 3, but this just dropped the time to 2-3 minutes. This problem was found when I realized you could only choose what board you wish to play on if you are the first to go. Thus, I just needed to save the board number in the state, which is why I created int position in AdvState class. This significantly boosted my efficiency because I was generating successors for all possible positions on all possible boards. That means it takes polynomial time or $O(n^2)$ to generate successors, which takes quite a while. Once I changed it to only generate successors for the board provided the time went from polynomial to linear making a ton faster and the heuristic still worked just as well. I could then increase the depth limit to 4 and it honestly go even deeper. After this was done, I had completed both projects and fit in any specifics like validating moves, checking if a board is full in Advanced TTT and filling in an system.err statements as needed.