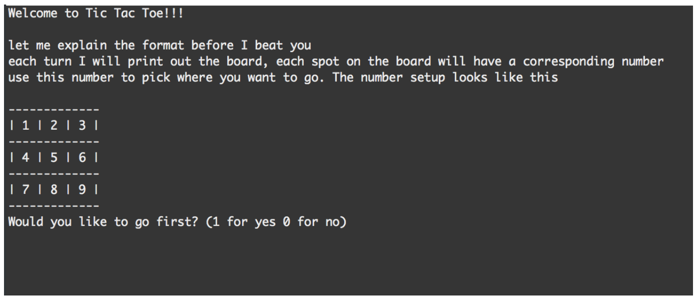
Project One Write Up

Thomas Kaizer

February 13, 2018

I began my project by formatting the simple tic tac toe board. My thought process in doing so was that it would be easier to troubleshoot any issues with the game later on if I had a visual representation of it. I first made a class called board and began coding up the user interface in a method called `printBoard`. I printed the board by making vertical lines out of dashes(-), and horizontal lines out of `or` bars (—).
Next I filled the board initially with numbers `1-9`, which the players would use to select spaces they would like to fill. I also programmed in some basic instructions for the user as to how to play the game in the class `Game`.
Next I had to program a way for the user to input moves. At this point I



```
Welcome to Tic Tac Toe!!!

let me explain the format before I beat you
each turn I will print out the board, each spot on the board will have a corresponding number
use this number to pick where you want to go. The number setup looks like this

-------------
| 1 | 2 | 3 |
-------------
| 4 | 5 | 6 |
-------------
| 7 | 8 | 9 |
-------------
Would you like to go first? (1 for yes 0 for no)
```

Figure 1: The output of my tic tac toe layout.

had to make a data structure to represent the board. I chose a string array to do this for simplicity sake. I believe I could also have used a character array. I personally chose to use strings however because I am more knowledgeable about `int to String` conversion, and have had problems using character representations in the past.
Next came the hard part. Creating my `minimax` algorithm so the computer

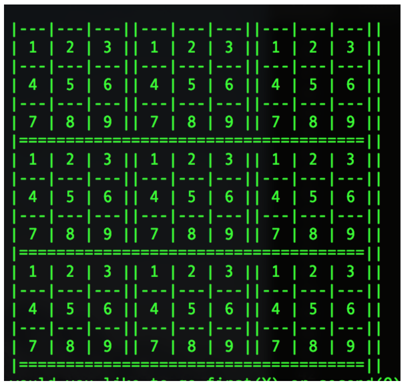
could generate a response.

To do this I followed the guidelines given by the pseudo code in the textbook. I first implemented `min` and `max` functions, which recursively called each other, using terminal states as base case.

On the topic of terminal states, I also implemented a `terminal` function, which would return *true* if the AI won, the player won, or the board was full. To check if either the player of AI won I also implemented a `winning` function, which just checked if the board had any of the winning combinations filled with either X or O.

Returning to the topic of the `minimax` function however, this function saved a copy of the board then used a temporary variable `z` to check all the possible outcomes, by running it through the `min` function, to start. Each time it found an optimal move it would save its index in variable `saveMove`. After it was done iterating through all possible moves, it would would input the optimal move into the original board, and then print it to the user using the `printBoard` function.
Next came my white whale. Ultimate tic tac toe, which I called Super tic tac toe incorrectly throughout the entire project.
I tried my best to follow a similar process as when I was developing normal tic tac toe, so the first thing I did was format the board. It was SUPER tedious.
I started by making another class called `superBoard`. In this class I defined a method `printSuperBoard()` which prints the $9x9$ tic tac toe board using similar symbols as were used in the original tic tac toe board. The superBoard structure itself was just a one dimensional array of Boards. I used the indexes of this array and nested for loops to fill the user interface with numbers corresponding to spaces that they could fill in. In the end it looked something like this.

***HOPEFULLY ONLY TEMPORARILY IN WRITEUP* My regular Tic Tac Toe Board was working, but now it seems not to be. It is now beatable if you go first and fill spots 1, then 3, then 2.**
it fills in spots sequentially (in numerical order) if you choose to go second. I don't know why it fills numbers in sequentially, but



```
|---|---|---||---|---|---||---|---|---||
| 1 | 2 | 3 || 1 | 2 | 3 || 1 | 2 | 3 ||
|---|---|---||---|---|---||---|---|---||
| 4 | 5 | 6 || 4 | 5 | 6 || 4 | 5 | 6 ||
|---|---|---||---|---|---||---|---|---||
| 7 | 8 | 9 || 7 | 8 | 9 || 7 | 8 | 9 ||
|=====================================||
| 1 | 2 | 3 || 1 | 2 | 3 || 1 | 2 | 3 ||
|---|---|---||---|---|---||---|---|---||
| 4 | 5 | 6 || 4 | 5 | 6 || 4 | 5 | 6 ||
|---|---|---||---|---|---||---|---|---||
| 7 | 8 | 9 || 7 | 8 | 9 || 7 | 8 | 9 ||
|=====================================||
| 1 | 2 | 3 || 1 | 2 | 3 || 1 | 2 | 3 ||
|---|---|---||---|---|---||---|---|---||
| 4 | 5 | 6 || 4 | 5 | 6 || 4 | 5 | 6 ||
|---|---|---||---|---|---||---|---|---||
| 7 | 8 | 9 || 7 | 8 | 9 || 7 | 8 | 9 ||
|=====================================||
```

Figure 2: My Ultimate Tic Tac Toe board

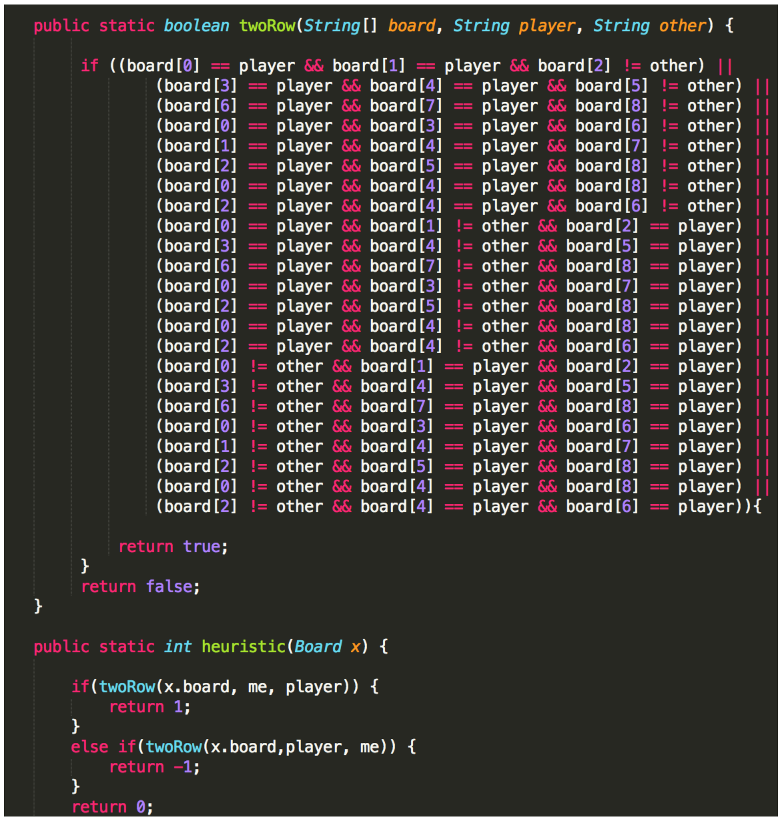
from what I can see it seems to not be finding a move with a more optimal utility value than $Integer.MINVALUE$...Literally any move should be greater than this number.

Around this point in my development process eclipse started acting buggy, which is why from now on, my pictures come from sublime. I also decided to move my code to sublime anyway to simplify the process of saving the project to files in a way that could be easily zipped and submitted.

Gonna leave that part in because I think it's relevant to the development process...I implemented the function `twoRow` into my code as a heuristic that would prioritize blocking a player. So the board now seems to work when the player chooses to go first.

Anywho, I continued on to make a series of distinct methods for `SuperGame` that more or less corresponded to the methods int `Game`. These included `ABmin`, `ABmax`,`superUtility`, `superYourMove`, and `superTerminal`. In place of minimax I used a function called `ABsearch`, AB standing for alpha and beta because the Ultimate tic tac toe game makes use of alpha beta pruning to cut down on the number of states in the search tree.

ABsearch once again made use of the two in a row heuristic and at the time of this writing does not seem to work. From what I can tell it seems that the AI is repeatedly attempting to place itself in the same space, and is therefore doing nothing because the space is full.

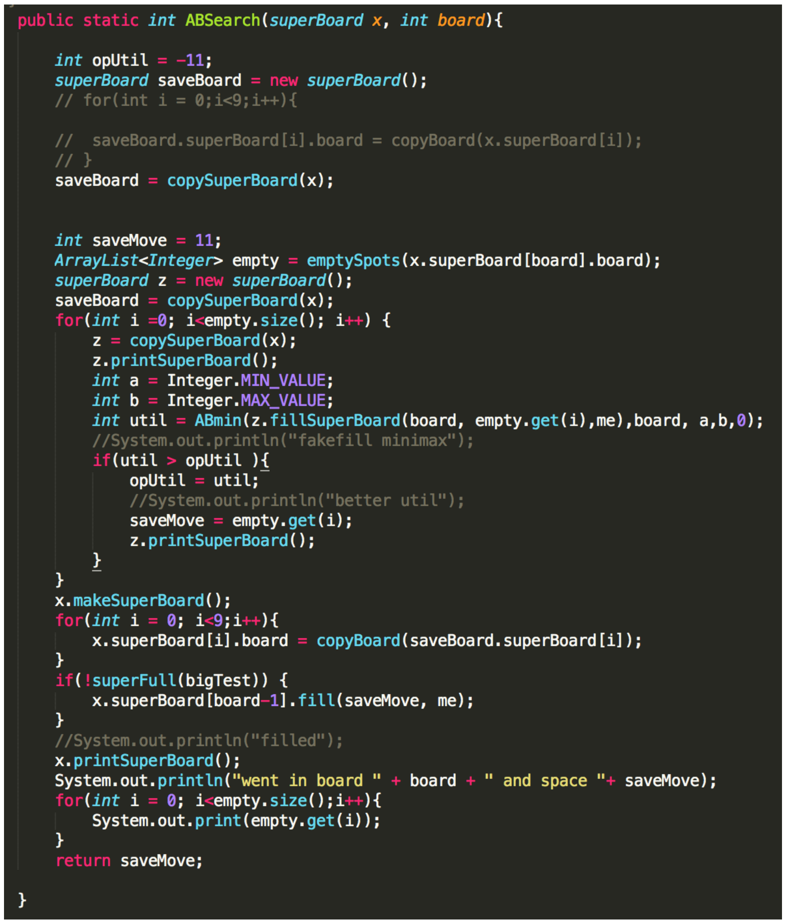


```java
public static boolean twoRow(String[] board, String player, String other) {

    if ((board[0] == player && board[1] == player && board[2] != other) ||
            (board[3] == player && board[4] == player && board[5] != other) ||
            (board[6] == player && board[7] == player && board[8] != other) ||
            (board[0] == player && board[3] == player && board[6] != other) ||
            (board[1] == player && board[4] == player && board[7] != other) ||
            (board[2] == player && board[5] == player && board[8] != other) ||
            (board[0] == player && board[4] == player && board[8] != other) ||
            (board[2] == player && board[4] == player && board[6] != other) ||
            (board[0] == player && board[1] != other && board[2] == player) ||
            (board[3] == player && board[4] != other && board[5] == player) ||
            (board[6] == player && board[7] != other && board[8] == player) ||
            (board[0] == player && board[3] != other && board[7] == player) ||
            (board[2] == player && board[5] != other && board[8] == player) ||
            (board[0] == player && board[4] != other && board[8] == player) ||
            (board[2] == player && board[4] != other && board[6] == player) ||
            (board[0] != other && board[1] == player && board[2] == player) ||
            (board[3] != other && board[4] == player && board[5] == player) ||
            (board[6] != other && board[7] == player && board[8] == player) ||
            (board[0] != other && board[3] == player && board[6] == player) ||
            (board[1] != other && board[4] == player && board[7] == player) ||
            (board[2] != other && board[5] == player && board[8] == player) ||
            (board[0] != other && board[4] == player && board[8] == player) ||
            (board[2] != other && board[4] == player && board[6] == player)){

        return true;
    }
    return false;
}

public static int heuristic(Board x) {

    if(twoRow(x.board, me, player)) {
        return 1;
    }
    else if(twoRow(x.board,player, me)) {
        return -1;
    }
    return 0;
```

Figure 3: My heuristic function

Another issue I've run into is that when opting to go second in Ultimate tic tac toe, the AI always commands the user to fill in a spot in space five. This is the default first move was set to *5,5*, and no room was left for the `space` parameter to be updated. I attempted to fix this by nesting the `ABsearch` function inside the `superYourMove` function, which actually seems to have made things worse. Now the same problem exists, and the board fills in two spots now.

Unfortunately at this point I am running fairly low on time, and must put a good amount of my focus into this write up. So I do not anticipate that my program will end up working perfectly before the due date. That being said, I believe I have learned enough that if I were to restart the project, and reformat how I made my program, I could make everything work. Though



```java
public static int ABSearch(superBoard x, int board){

    int opUtil = -11;
    superBoard saveBoard = new superBoard();
    // for(int i = 0;i<9;i++){

    //  saveBoard.superBoard[i].board = copyBoard(x.superBoard[i]);
    // }
    saveBoard = copySuperBoard(x);


    int saveMove = 11;
    ArrayList<Integer> empty = emptySpots(x.superBoard[board].board);
    superBoard z = new superBoard();
    saveBoard = copySuperBoard(x);
    for(int i =0; i<empty.size(); i++) {
        z = copySuperBoard(x);
        z.printSuperBoard();
        int a = Integer.MIN_VALUE;
        int b = Integer.MAX_VALUE;
        int util = ABmin(z.fillSuperBoard(board, empty.get(i),me),board, a,b,0);
        //System.out.println("fakefill minimax");
        if(util > opUtil ){
            opUtil = util;
            //System.out.println("better util");
            saveMove = empty.get(i);
            z.printSuperBoard();
        }
    }
    x.makeSuperBoard();
    for(int i = 0; i<9;i++){
        x.superBoard[i].board = copyBoard(saveBoard.superBoard[i]);
    }
    if(!superFull(bigTest)) {
        x.superBoard[board-1].fill(saveMove, me);
    }
    //System.out.println("filled");
    x.printSuperBoard();
    System.out.println("went in board " + board + " and space "+ saveMove);
    for(int i = 0; i<empty.size();i++){
        System.out.print(empty.get(i));
    }
    return saveMove;

}
```

Figure 4: AB search

this project might not turn out well, I will make a point to get a working ultimate tic tac toe AI on my github at some point this semester.

The final edit I made to my code was that I changed the main method in my `Game` class to be the method `playTTT`, I then implemented a scanner with if/else if statements in the main method of `SuperGame`, to allow the user to play either form of Tic-Tac-Toe through one executable.