**Foundations of Programming 2 Programming Project 4**

**CSC6002 Spring 1 - 2022 (Grzyb) Due Date: Wednesday, Feb. 16, 2022, 11:59pm**

Please solve the problem(s) alone. Remember to test your solution thoroughly. Code that does not compile will lose credit. Please submit a digital copy, by the due date and time, via Blackboard. The digital copy of your code should be a zip file containing the entire project folder named with your last name followed by the project number. For example, grzyb4.zip would be my zip file for project 4. Good luck!

In this assignment, you will write a program that plays Tic Tac Toe and develop an algorithm using recursion for a computer opponent that will pick the best possible move based on the current state of the board.

This assignment has two main parts. In the first part, you must write a version of Tic Tac Toe that is to be played by humans only (i.e. you always enter the square for ‘X’ or ‘O’). In the second part of the assignment, you will allow the user to play as ‘X’ or ‘O’ and the computer opponent will play the best possible move each turn for itself. ‘X’ always goes first.

The main method for your program should be in a class called TicTacToe and when it runs it should present the following menu and loop until option 4 is chosen to end the program:

Tic Tac Toe

1. Play against human

2. Play against computer (Human: X, Computer: O)

3. Play against computer (Computer: X, Human: O)

4. End program

For each part of the program, you will need the following classes:

# Board

|  |
| --- |
| Board |
| -square: char[3][3] |
|  |
| +Board(): |
| +getSquare(column: int, row: int): char  +setSquare(column: int, row: int, c: char): void  +draw(): void  +determineBoardStatus(): BoardStatus  +makeAMove(currentPlayer: char): Move |
|  |

The Board represents the Tic Tac Toe board with the following attributes and methods:

* square – a 3x3 character array that represents the 9 squares on the Tic Tac Toe board. When the game starts, each square should be initialized to a blank space ‘ ‘. When the square is selected by a player, the value should change to an ‘X’ or an ‘O’ based on the current player
* getSquare – returns the character value of the column/row on the board
* setSquare – sets the character value of the column/row on the board
* determineBoardStatus – checks the state of the board and returns one of the BoardStatus enum values:
  + UNFINISHED – if the board is not full and there is not a winner
  + X\_WINS – if there are three ‘X’ squares in a row (horizontally, vertically or diagonally)
  + O\_WINS – if there are three ‘O’ squares in a row (horizontally, vertically or diagonally)
  + DRAW – if the board is full and there is no winner
* makeAMove – this is the method that will be recursively called for the computer opponent as you simulate all possible outcomes of the game and determine the best possible Move to make. It should behave as follows
  + the currentPlayer argument is the player that is currently making a move ‘X’ or ‘O’
  + the makeAMove method should return the best possible ‘Move’ for the computer to make
  + this method should iterate through each square on the board
    - if the square is empty, set the square to the currentPlayer (e.g. ‘X’ or ‘O’)
    - determine the status of the Board
    - if the game is unfinished, make a move for the **opponent** of the player that just moved (i.e. if ‘O’ just went, recursively call with ‘X’ to simulate players taking turns)
    - if the current player is ‘X’ and the status is X\_WINS, that is the best move so return a Move object that represents that square and status
    - if the current player is ‘O’ and the status is O\_WINS, that is the best move so return a Move object that represents that square and status
    - if the status is a draw then we will temporarily store that and return from the current call to makeAMove
  + As you go through each empty square on the board and place an ‘X’ or an ‘O’ in your simulation, you must be sure to reset each square back to blank before you return from makeAMove as each call to makeAMove recursively is a **simulation**, not the actual move
  + The actual move is what makeAMove returns to the main game when it is called
  + The main game will update the board based on the ‘Move’ object that makeAMove returns
* draw – Draws the current state of the Tic Tac Toe Board. For example:

O | X | O

---+---+---

O | O | X

---+---+---

X | X | X

The board positions are by row and column:

[0][0] | [1][0] | [2][0]

---------+----------+----------

[0][1] | [1][1] | [2][1]

---------+----------+----------

[0][2] | [1][2] | [2][2]

# Move

|  |
| --- |
| Move |
| -column: int  -row: int  -boardStatus: BoardStatus |
|  |
| +Move(): |
| +getColumn(): int  +getRow(): int  +getBoardStatus(): BoardStatus  +setColumn(column: int): void  +setRow(row: int): void  +setBoardStatus(boardStatus: BoardStatus): void  +equals() |
|  |

The Move class represents a possible move to make (i.e. if an ‘X’ or ‘O’ is placed at the given square) and the BoardStatus when that move is made

The BoardStatus is an enumeration of the state of the game at that time based on the squares

* UNFINISHED – if the board is not full and there is not a winner
* X\_WINS – if there are three ‘X’ squares in a row (horizontally, vertically or diagonally)
* O\_WINS – if there are three ‘O’ squares in a row (horizontally, vertically or diagonally)
* DRAW – if the board is full and there is no winner

|  |
| --- |
| <<enumeration>>  BoardStatus |
| UNFINISHED  X\_WINS  O\_WINS  DRAW |

# Option 1 – Human Opponent

The first part of the assignment deals with all the aspects of the game except for the recursion (drawing the board, accepting input from the human player(s) and determining the status of the Board after each move). (See appendix for example output).

If option 1 is selected (Play against human), the program should loop until the game is finished doing the following through each turn:

* Prompt for X to enter a column and a row (each between 0-2)
* If the column and row is valid and empty, place the ‘X’
* Draw the board
* Determine if the game is over, if so, print out the status (X\_WINS, O\_WINS or DRAW)
* If not, repeat with ‘O’ then ‘X’ then ‘O’ etc.

# Option 2 – Computer Opponent (Human: X)

If option 2 is selected (Computer Opponent with Human being X), the program should loop until the game is finished doing the following through each turn:

* Prompt for X to enter a column and a row (each between 0-2)
* If the column and row is valid and empty, place the ‘X’
* Draw the board
* Determine if the game is over, if so, print out the status (X\_WINS, O\_WINS or DRAW)
* Now, you must implement the makeAMove method for ‘O’ and determine the best square to choose based on the board with the ‘X’ that the human opponent places.
* Place an ‘O’ on the board for the ‘Move’ that was returned by makeAMove
* Draw the board
* Determine if the game is over, if so, print out the status (X\_WINS, O\_WINS or DRAW)
* Go back to the human opponent ‘X’ and repeat from the top

# Option 3 – Computer Opponent (Computer: X)

If option 3 is selected (Computer Opponent with the Computer being X), the program should loop until the game is finished doing the following through each turn:

* Use the makeAMove method for ‘X’ first and determine the best square to choose for the computer based on the board
* Place an ‘X’ on the board for the ‘Move’ that was returned by makeAMove
* Draw the board
* Determine if the game is over, if so, print out the status (X\_WINS, O\_WINS or DRAW)
* Prompt for O to enter a column and a row (each between 0-2)
* If the column and row is valid and empty, place the ‘O’
* Draw the board
* Determine if the game is over, if so, print out the status (X\_WINS, O\_WINS or DRAW)
* Go back to the computer opponent ‘X’ and repeat from the top

Some tests will be provided to help you test your program and some starter code for the menu.

Grading

Your grade on this assignment (100 points) will be defined as follows:

* TicTacToe board is drawn properly (10 points)
* An ‘X’ or ‘O’ can only be placed on a valid square and returns an appropriate error message otherwise (10 points)
* determineBoardStatus properly returns the state of the Board at any given time (10 points)
* Play against Human is implemented properly (15 points)
* Play against Computer (Human is X) is implemented properly (20 points)
* Play against Computer (Computer is X) is implemented properly (15 points)
* All provided tests pass (10 points)
* Program follows course coding standard guidelines (10 points)

# Appendix

## Option 1 – Sample output:

Tic Tac Toe

1. Play against human

2. Play against computer (Human: X, Computer: O)

3. Play against computer (Computer: X, Human: O)

4. End program

1

Human opponent

| |

---+---+---

| |

---+---+---

| |

Enter a column (0-2) for 'X':

0

Enter a row (0-2) for 'X':

0

X | |

---+---+---

| |

---+---+---

| |

Enter a column (0-2) for 'O':

1

Enter a row (0-2) for 'O':

1

X | |

---+---+---

| O |

---+---+---

| |

Enter a column (0-2) for 'X':

0

Enter a row (0-2) for 'X':

1

X | |

---+---+---

X | O |

---+---+---

| |

Enter a column (0-2) for 'O':

2

Enter a row (0-2) for 'O':

2

X | |

---+---+---

X | O |

---+---+---

| | O

Enter a column (0-2) for 'X':

0

Enter a row (0-2) for 'X':

2

X | |

---+---+---

X | O |

---+---+---

X | | O

XWINS

## Option 2 – Sample output:

Tic Tac Toe

1. Play against human

2. Play against computer (Human: X, Computer: O)

3. Play against computer (Computer: X, Human: O)

4. End program

2

Human: X, Computer: O

| |

---+---+---

| |

---+---+---

| |

Enter a column (0-2) for 'X':

1

Enter a row (0-2) for 'X':

0

| X |

---+---+---

| |

---+---+---

| | O

Enter a column (0-2) for 'X':

0

Enter a row (0-2) for 'X':

0

X | X | O

---+---+---

| |

---+---+---

| | O

Enter a column (0-2) for 'X':

2

Enter a row (0-2) for 'X':

1

X | X | O

---+---+---

| | X

---+---+---

| O | O

Enter a column (0-2) for 'X':

0

Enter a row (0-2) for 'X':

2

X | X | O

---+---+---

O | | X

---+---+---

X | O | O

Enter a column (0-2) for 'X':

1

Enter a row (0-2) for 'X':

1

X | X | O

---+---+---

O | X | X

---+---+---

X | O | O

DRAW

## Option 3 – Sample output:

Tic Tac Toe

1. Play against human

2. Play against computer (Human: X, Computer: O)

3. Play against computer (Computer: X, Human: O)

4. End program

3

Computer: X, Human: O

| |

---+---+---

| |

---+---+---

| | X

Enter a column (0-2) for 'O':

1

Enter a row (0-2) for 'O':

1

| |

---+---+---

| O | X

---+---+---

| | X

Enter a column (0-2) for 'O':

2

Enter a row (0-2) for 'O':

0

| | O

---+---+---

| O | X

---+---+---

X | | X

Enter a column (0-2) for 'O':

0

Enter a row (0-2) for 'O':

0

O | | O

---+---+---

| O | X

---+---+---

X | X | X

XWINS