UNIVERSITI UTARA MALAYSIA

SECOND SEMESTER SESSION 2021/2022 (A212)

UUM SCHOOL OF ART AND SCIENCES

STIA1113 PROGRAMMING 1

GROUP A

GROUP PROJECT: REPORT

PREPARED FOR:

DR. NORADILA BINTI NORDIN

PREPARED BY:

GROUP 5

NAME	MATRIC NUMBER
LEE JUAN	280027
MUHAMMAD ZUHAIR AFHAM BIN MOHD NASIR	280782
EDWIN LIM WEI BIN	281775

SUBMISSION DATE:

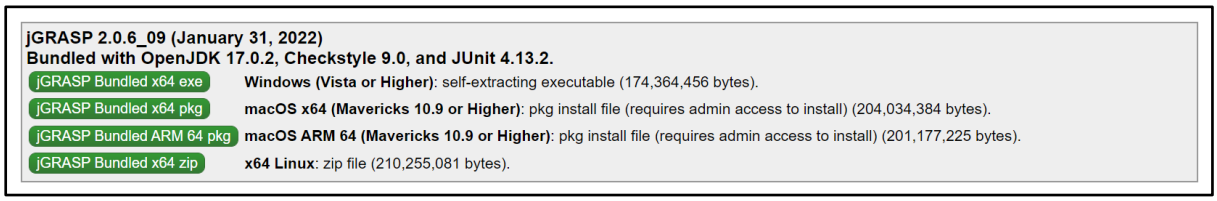
14 JULY 2022

TABLE OF CONTENTS

1.0 USER MANUAL

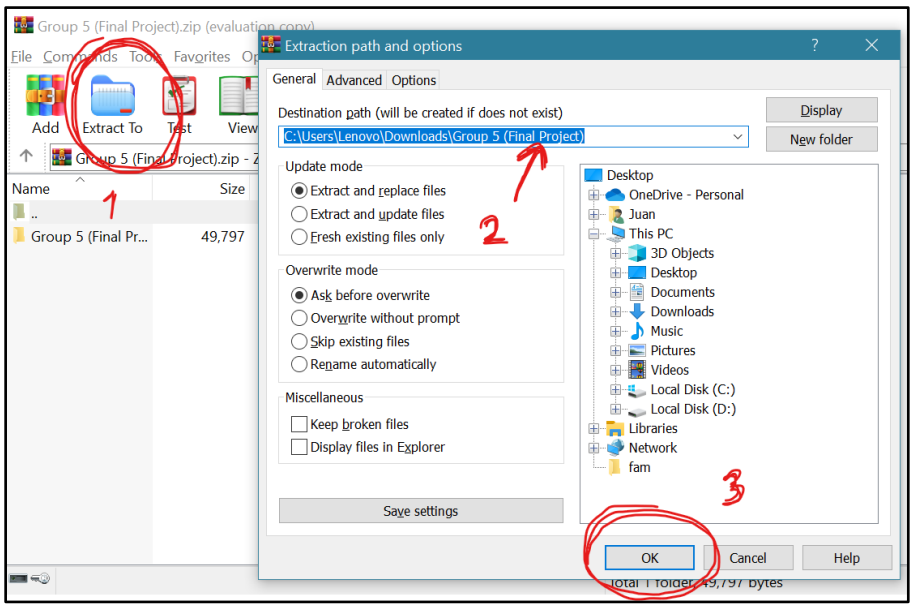
Below is a simple manual for users to access, open, and run the programme:

1. To have *jGRASP* on your computer, click here, and scroll to find the section as shown in the screenshot below. Download the bundle for your operating system, e.g. "*jGRASP Bundled x64 exe*" for Windows, or "*jGRASP Bundled x64 pkg*" for macOS. After it is done, open the executable to start installing the application.



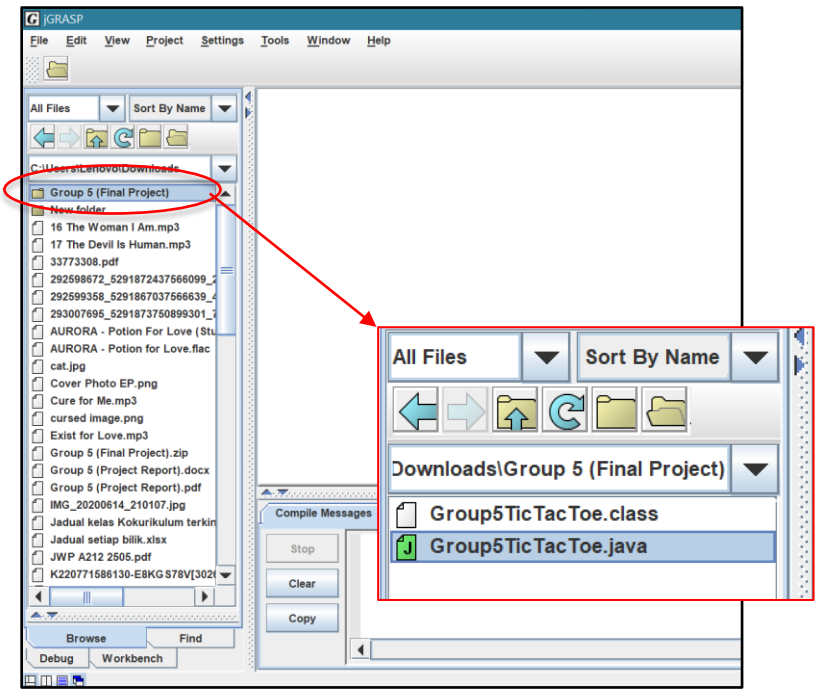
2. Next, after downloading the zipped folder of our programme, extract it using a folder extract tool e.g. *WinRAR*. You may download WinRAR here.

3. In *WinRAR*, click "Extract To", then select the folder or destination path that you want to save the programme in. Click "OK".

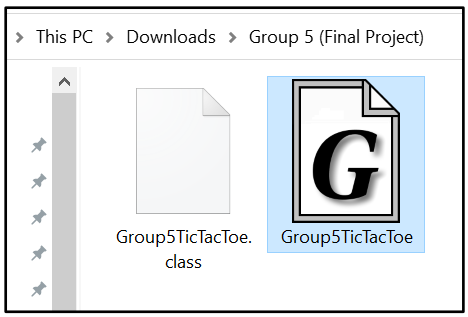


4. Once *jGRASP* has done installed and the zipped folder extraction has completed, there are two ways to open the file on *jGRASP:*

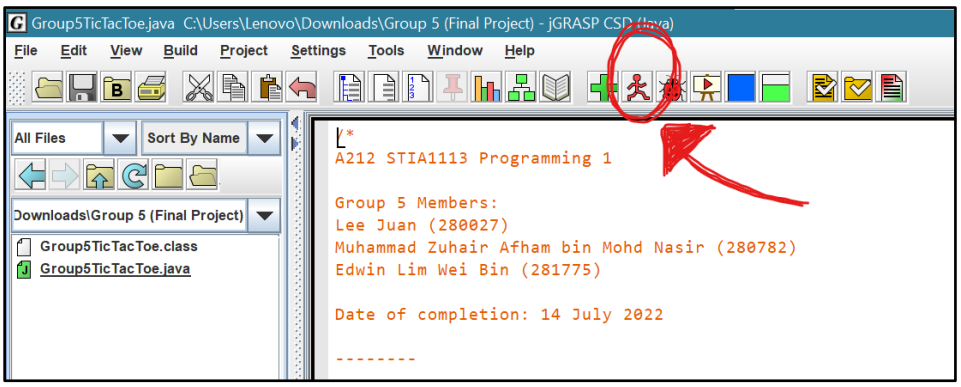
a. (Option1): Locate the extracted folder on the left side of the application as shown below, then double click on *Group5TicTacToe.java*.



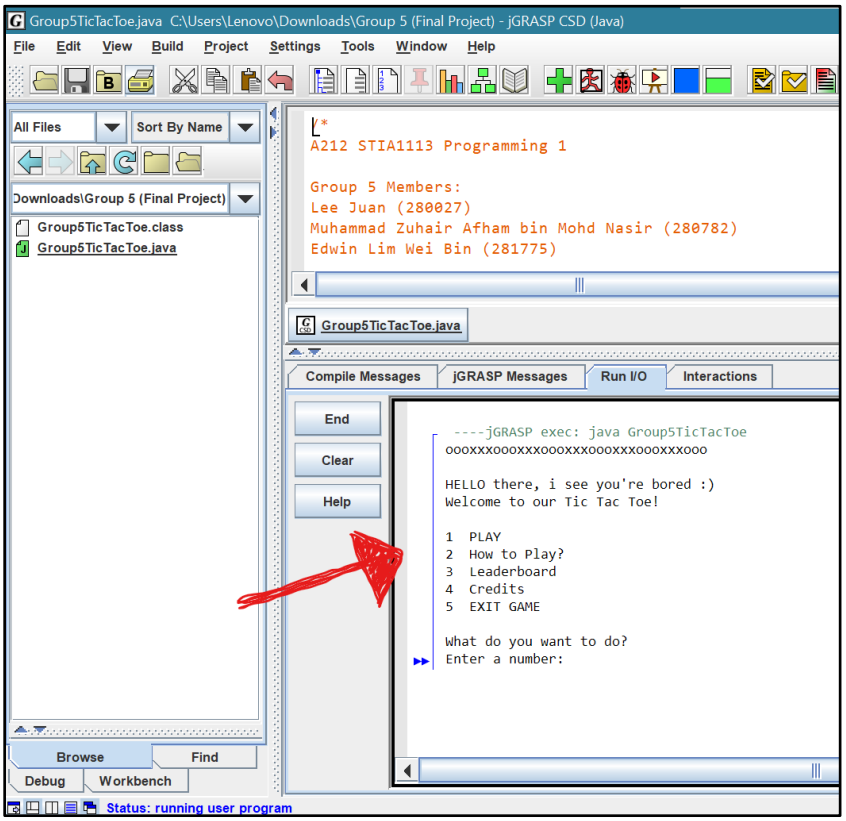
b. (Option 2): Find the folder on your computer, then double click the *Group5TicTacToe.java* file (the icon of the file should look like the icon for jGRASP since you have it installed on your computer). jGRASP should open automatically once you do so.



5. The codes for the programme should show up in the application. Finally, navigate to the top of the application and select the “Find and run main method or applet” button (button with a red stickman) as circled in red in the picture below.



6. The file should be running after compiling successfully. User may then interact with the programme at the terminal section (bottom of the application) as shown in the arrow below:



7. Have fun!

2.0 INTRODUCTION

Tic Tac Toe is a game in which two players alternately put ‘X’ or ‘O’ in two vertical lines crossing two horizontal lines. Players need to complete each row with three ‘X’ or three ‘O’ before their opponent does. The row can be horizontal, vertical or diagonal. Coded in Java, this programme is a Tic Tac Toe game that consists of two 2-player modes: easy and medium, where two players compete with each other to victory in easy mode, and the players will face a challenge by having to answer a maths question before making their move in medium mode. It also has a CPU mode where users will play against the computer. This programme allows players to enable a tips function that will display some game suggestions or funny commentaries during gameplay. Additionally, it has a simple high score and leaderboard function that would let players view their rankings in the game.

2.1 Brief Introduction to Interface and Controls

```
HELLO there, i see you're bored :)
Welcome to our Tic Tac Toe!

1  PLAY
2  How to Play?
3  Leaderboard
4  Credits
5  EXIT GAME

What do you want to do?
Enter a number:
```

When we run the program, it will bring us to the main menu where it shows “Play”, “How to play”, “Leaderboard”, “Credits” and “Exit Game”. There is a number in front representing each of those options. Below, the programme will ask for the user's input before going into another part of the programme, e.g. in this case what we want to do. Users will then need to enter the number that represents the option. To put an example, if we want to play the game, we need to enter the number ‘1’ so it can proceed to the next display which shows us “2-Player: Easy mode”, “2-Player: Normal mode”, “CPU mode” and “Back to main menu” as shown below:

```
Enter a number: 1

1  2-PLAYER: EASY MODE      ('>')/*\('<')
2  2-PLAYER: MEDIUM MODE    (">")/***\("<")
3  CPU MODE                 (@_@)
4  Back to main menu

Enter the mode that you want to play, or back to menu:
```

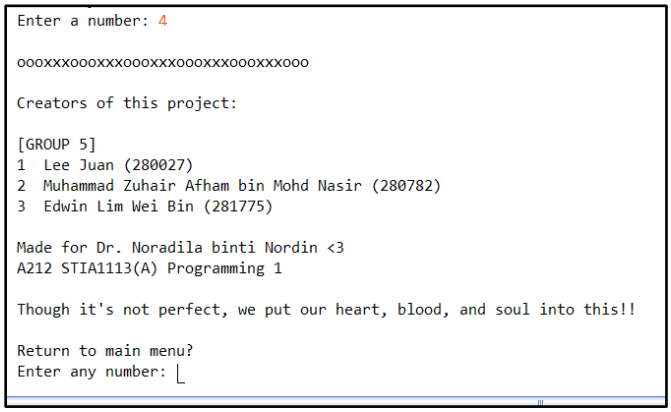
Back at the main menu's display, if we choose to enter the number '2' instead, it will bring us to the "How to play" menu. In this menu, it will show us the rules of the game:

```
Enter a number: 2

oooxxxoooxxxoooxxxoooxxxoooxxxooo

--- RULES ---

The game is played on a 3x3 grid.
Assuming you are 'X', and your friend (or the computer) is 'O'.
The first player to get 3 of thier marks in a row, either HORIZONTALLY, VERTICALLY, or DIAGONALLY; is the winner.
However, when all 9 squares are full, the game is tied and nobody wins.
It's that simple. but ARE YA READY?! ;)
```

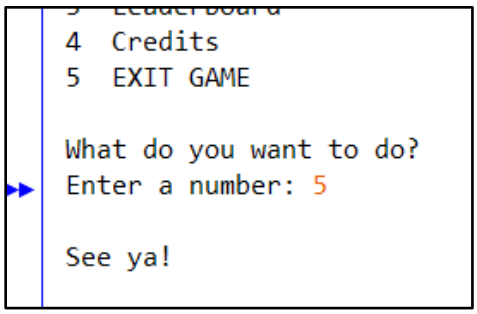
If we enter the number '3', it will bring us to the "Leaderboard" menu. In the leaderboard mennu, it will show us the player's current place, name and score. Besides that, users can also search for a player and reset the leaderboard.

```
Enter a number: 3

oooxxxoooxxxoooxxxoooxxxoooxxxooo

--- LEADERBOARD ---

Place    Name        Score
1        CPU           0
1        null          0
1        null          0

--------------
1  Search name in leaderboard
2  Reset leaderboard
3  Return to main menu
```

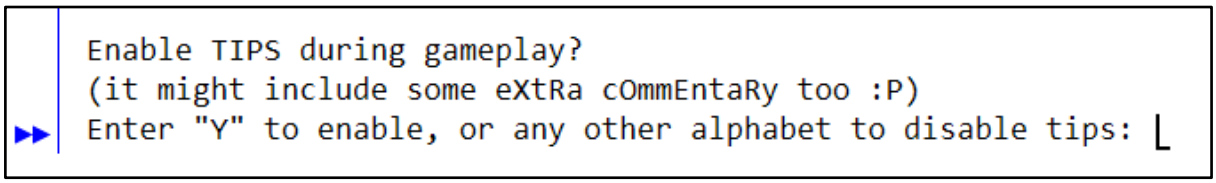
When we enter the number ‘4’, it will show us the creators of the program:



Lastly, if we enter the number ‘5’, the programme terminates.



At some occasions, users are prompted to enter an alphabet instead, as shown in the screenshot below where the programme asks if user wants to enable tips:



3.0 REQUIREMENTS

In this chapter, we will explain some important codes in our programme that might need further clarification.

3.1 Importing Java Utilities and Declaring Global Variables

The *Scanner* utility is imported to provide convenient methods for reading user input of various types, whereas the *Random* utility is needed to generate random numbers for CPU decisions in CPU mode as well as the random addition maths questions in 2-Player Medium Mode. They are both part of the java.util class library.

The *input* variable is declared as a global variable so that its Scanner function can be used anywhere within the class regardless of any method. *option* is always used to collect different kinds of input from users, hence it is also declared globally.

```
1  import java.util.Scanner;
2  import java.util.Random;     //"Random" utility needed for CPU to make decisions in CPU mode
3
4
5  public class Group5TicTacToe {
6
7
8     public static Scanner input = new Scanner(System.in);          //global variable for creating input function
9     public static int option;                                      //global variable for option input
10    public static char tipsCheck;                                  //tipsCheck = Y if tips are enabled
11    public static String name1, name2, nameCPU = "CPU";
12    public static int score1 = 0, score2 = 0, tipsCount;
13
```

3.2 Main Method

In the *main()* method, the for loop is being used as an infinite loop to repeat the *menu()* method endlessly. The purpose is to let users return to the main menu every time a mode ends.

```
//main method
public static void main(String[] myVariable) {

   for (;;) {       //infinite loop to always return to menu whenever a mode ends
      menu();
   }

} //end main
```

The do-while loop as shown below will repeat the codes within the curly bracket when the user inputs an invalid number, given the specified conditions.

```
do {
   System.out.println("\nWhat do you want to do?");
   System.out.print("Enter a number: ");
   option = input.nextInt();

   if ((option != 1) && (option != 2) && (option != 3) && (option != 4) && (option != 5)) {
      System.out.println("Invalid number! Try again.");
   }

} while ((option != 1) && (option != 2) && (option != 3) && (option != 4) && (option != 5));    //do-while loop if user input is invalid
```

The only way users will end the programme properly is when they enter '5' on the main menu page or *menu()* where they are prompted to select an option.

```
1  PLAY
2  How to Play?
3  Leaderboard
4  Credits
5  EXIT GAME

What do you want to do?
Enter a number: 5

See ya!

 ----jGRASP: operation complete.
```

```
else {
    System.out.println("\nSee ya!");
    System.exit(0);                    //end programme
}
```

Above is the code to terminate the programme.

3.3 Two-Player Easy Mode and Tips Function

Below is the beginning of the method *play2Easy()* or 2-player easy mode, where variables are being declared.

```
do {

   System.out.println("\noooxxxoooxxxoooxxxoooxxxoooxxxooo\n");
   System.out.println("--- 2-PLAYER: EASY MODE  ('>')/*\\('<') ---\nIt's more fun playing with friends, right!\n");

   boolean player1 = true;             //player 1 or 2's turn
   boolean endGame = false;            //boolean that is false in default, otherwise game ends
   char mark;                          //Xs or Os
   int row, col;                       //row and column number, input by user
   char[][] board = new char[3][3];    //array for 2D-board

   tipsCount = 0;              //reset tipsCount
```

The boolean variable *player1* is true meaning that it's Player 1's turn, and that it's Player 2's turn if otherwise. *endGame* variable is set to false by default. *tipsCount* is a variable that increments when the tips function is enabled by the user. In all modes, 2D arrays are used to store 3x3 values that form the values in each cell within the Tic Tac Toe board.

```
//method to display tips when enabled by user
public static void tips() {

   if (tipsCheck == 'Y') {
      tipsCount++;
   }

   switch (tipsCount) {

      case 1:
         System.out.println("Tip: Secure the corners first and you'll decre
         break;
      case 2:
         System.out.println("Tip: Take the center spot if your opponent too
         break;
      case 3:
         System.out.println("Tip: Always be aware of your opponent's move,
         break;
      case 4:
```

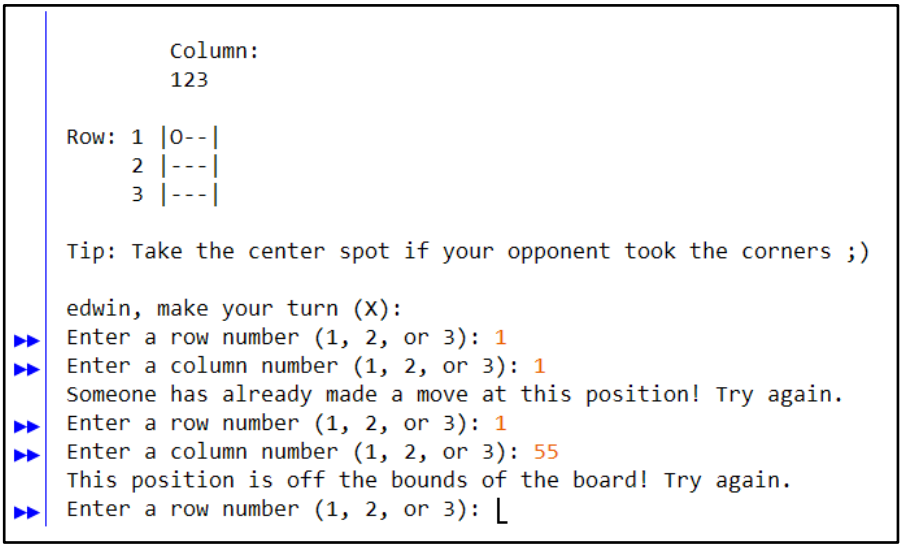
If the user enables tips before proceeding to the game modes, the *tips()* method will verify it and print out different tips for each passing turn that is calculated with the *tipsCount* variable as shown above.

Below, we assigned the values *i* and *j* to symbolise the rows and columns in the *board[][]* array. This *drawBoard(board)* method utilises a for loop to read the values in the said array and prints them out in a specific order.

```
public static void drawBoard(char[][] board) {

   int r = 1;
   System.out.println("\n\t\t  Column:");
   System.out.println("\t\t  123\n");
   System.out.print("Row: " + r + " |");

   for (int i=0; i<3; i++) {
      for (int j=0; j<3; j++) {
         System.out.print(board[i][j]);
      }

      System.out.print("|");
      r++;
      if (r<=3) {
         System.out.print("\n\t  " + r + " |");
      }
      else {
         System.out.println("\n");
      }

   } //end for loop
```

The code below uses the if-else statement to check the validity of the user's input on *row* and *column*. It is considered invalid when a player marks a spot that has been taken or inserts a value that is beyond the rows and columns set for the board. Notice the global variable *input* is always used to collect input from users.

```
while (true) { //loop if user input is invalid
   System.out.print("Enter a row number (1, 2, or 3): ");
   row = input.nextInt();
   System.out.print("Enter a column number (1, 2, or 3): ");
   col = input.nextInt();

   //check validity of row and col input
   if (row<1 || col<1 || row>3 || col>3) {
      System.out.println("This position is off the bounds of the board! Try again.");
   }
   else if (board[row-1][col-1] != '-') {
      System.out.println("Someone has already made a move at this position! Try again.");
   }
   else {
      break;
   }

}//end while(true)
```



Once a user has entered the right values for *row* and *col*, they are then assigned to represent a single mark on the board, which is 'O' for Player 1 and 'X' for Player 2. Before a turn ends, the programme will check whether the board has a completed row, either horizontally, vertically, or diagonally by reading the *win(board)* method. If any condition is true, the programme announces the respective winner of the game, and increments the variable *score1* or *score2* (which represents the score for each player), depending on the winner. Or, if the programme reads that the 3x3 board is full, as in all players have made their turn and there are still no completed lines, the game will conclude the match with a tie. Either way, the variable *endGame* will turn "true", hence forcing the do-while loop that loops the entire mode to end.

However, if neither of those conditions, with players winning or tie, are true; the boolean for *player1* will switch, indicating the turn has switched to another player. The codes are shown as below:

```
    //assign the position of row and col to mark Xs or Os to the board
    board[row-1][col-1] = mark;

    //check if any player has won or if the game is a tie
    if (win(board) == 'O') {
       System.out.println("\n***" + name1 + " HAS WON!!***");
       score1++;
       endGame = true;
    }
    else if (win(board) == 'X') {
       System.out.println("\n***" + name2 + " HAS WON!!***");
       score2++;
       endGame = true;
    }
    else if (boardFull(board)) {
       System.out.println("\n***GAME IS A TIE! Nobody won :\\***");
       endGame = true;
    }
    else {
       player1 = !player1;           //continue to next player's turn if either conditions are true
    }

 }//end while (!endGame)
```

^ Screenshot above: Position assignation and end game check

```
//method to check whether someone has won in 2-player mode
public static char win(char[][] board) {

   //check for each row (horizontally)
   for (int i=0; i<3; i++) {
      if (board[i][0] == board[i][1] && board[i][1] == board[i][2] && board[i][0] != '-') {
         return board[i][0];
      }
   }

   //check for each column (vertically)
   for (int j=0; j<3; j++) {
      if (board[0][j] == board[1][j] && board[1][j] == board[2][j] && board[0][j] != '-') {
         return board[0][j];
      }
   }

   //check for diagonals
   if (board[0][0] == board[1][1] && board[1][1] == board[2][2] && board[0][0] != '-') {
      return board[0][0];
   }
   if(board[2][0] == board[1][1] && board[1][1] ==  board[0][2] && board[2][0] != '-') {
      return board[2][0];
   }

   //otherwise nobody wins yet, game continues
   return '-';

} //end win()
```

^ Screenshot above: *win(board)* method where it checks for the winning condition

```
//method to check if entire board is filled up to end game as a tie in 2-player mode
public static boolean boardFull(char[][] board) {

   for (int i=0; i<3; i++) {
      for (int j=0; j<3; j++) {
         if (board[i][j] == '-') {
            return false;
         }
      }
   }
   return true;

} //end boardFull()
```

^ Screenshot above: *boardFull(board)* method where it checks if the board is filled up

3.4 Two-Player Medium Mode

With this mode, though it uses the same methods used in 2-Player Easy Mode: with *drawBoard(board)* to print out the board, *win(board)* to check whether someone has won, and boardFull(board) to check if the entire board is filled up to end game as a tie; and while its main method is similar as of easy mode, there were some add ons. Before making a turn, users are prompted with a general maths addition question, where users need to answer correctly in order to proceed with their move, else their turn is forfeited and skipped to the other player, and so forth if that player does the same.

```
Random randomNumber = new Random();
int num1 = randomNumber.nextInt(100), num2 = randomNumber.nextInt(100);    //initialise random numbers between 0-100 to num1 and num2
int answer, realAnswer;

System.out.println("Answer a question correctly to make a move, otherwise, you skip a turn :D");
System.out.println("What is " + num1 + " + " + num2 + "?");               //question on addition of two random numbers

if (player1) {
   System.out.print(name1 + ", enter your answer: ");
}
else {
   System.out.print(name2 + ", enter your answer: ");
}
answer = input.nextInt();

realAnswer = num1 + num2;

if (answer == realAnswer) {
   System.out.println("You got it right!");
   correct = true;
}
else {
   System.out.println("Oops, wrong answer. You lost a turn :(");
   correct = false;
}

//if answer is correct, proceed with turn
if (correct) {
   if (player1) {
      System.out.println(name1 + ", make your turn (O): ");
```

From the screenshot above, after importing the *Random* utility, the *Random* function is then assigned to *num1* and *num2*, where it generates a value from 0 to 100 to each of the numbers for every alternating turn. The correct addition of those two numbers are assigned to the variable *realAnswer*, and is then compared to the user input *answer*. If they are the same, the boolean variable *correct* turns to "true", the programme will then proceed with their respective turns using the if-else statement. Otherwise when it's "false", the programme will ignore everything that is contained in the condition *(correct)*, and straightaway proceed to the next player's turn (as shown in the picture below):

```
}
//lose turn if answer is wrong, switch to next player's turn
else {
   player1 = !player1;
}
```

3.5 Computer Mode

```
System.out.println("\nooooxxxoooxxxoooxxxoooxxxoooxxxooo\n");
System.out.println("--- CPU MODE  (@_@) ---\nCan you defeat a non-living thing?\n");
System.out.println("The numbers representing each cell:");
System.out.println("1" + "|" + "2"  + "|" + "3");
System.out.println("-----");
System.out.println("4" + "|" + "5"  + "|" + "6");
System.out.println("-----");
System.out.println("7" + "|" + "8"  + "|" + "9");
System.out.println("\n************\n");
```

```
The numbers representing each cell:
1|2|3
-----
4|5|6
-----
7|8|9

************
```

In the first part of the computer mode code, we have displayed a Tic Tac Toe game board and show the game board cells with respect to numbers from 1 to 9. This can remind the player about the number that represents each cell.

The 2D array: *board* to create the game board is shown below. The blank space that exists before we fill in any letter (‘X’ or ‘O’) will be represented by the space between the single quotes. That is, we will have three rows and each row will have three spaces for the player or computer to fill in their move.

```
char [] [] board = {{' ', ' ', ' '},
                    {' ', ' ', ' '},
                    {' ', ' ', ' '}}; //create board
```

```
public static void printBoard(char [] [] board) {

   System.out.println(board [0][0] + "|" + board [0][1] + "|" + board [0][2] );
   System.out.println("-----");
   System.out.println(board [1][0] + "|" + board [1][1] + "|" + board [1][2] );
   System.out.println("----");
   System.out.println(board [2][0] + "|" + board [2][1] + "|" + board [2][2] + "\n" );

}// end printBoard();
```

For this part, we have used the method technique since we will use it again and again in our following code. Here, we have used [0][0] to represent the cell 1 which will be placed at the upper left of the game board. The purpose of this process can help us link the number with the cells in our following code:

```
//method to check number input from user in CPU mode
public static void playerMove(char [][] board) {

    String userInput;

    while (true) {
        System.out.print("X, make a move by entering a number (1~9): ");
        userInput = input.next();

        if (emptyCells(board, userInput)) {
            break;
        }
        else {
            System.out.println("************");
            System.out.println(userInput + " is not a valid move.");
            System.out.println("************");
        }

     } //end while
```

Player's move was also coded using method technique since every move of the player will reuse the same code. At this point, the user can enter any number from 1 until 9 into the empty cells. If the user inputs a number that already exists or the number that is out of the range of the board (which is smaller than 1 or bigger than 9), the system will display "The user input is not a valid move." which represents what user has input to the system.

```
System.out.println(userInput);

switch (userInput) {
    case "1":
        board[0][0] = 'X';
        break;
    case "2":
        board[0][1] = 'X';
        break;
    case "3":
        board[0][2] = 'X';
        break;
    case "4":
        board[1][0] = 'X';
        break;
    case "5":
        board[1][1] = 'X';
        break;
    case "6":
        board[1][2] = 'X';
        break;
    case "7":
        board[2][0] = 'X';
        break;
    case "8":
        board[2][1] = 'X';
        break;
    case "9":
        board[2][2] = 'X';
        break;
     default:
        System.out.println();
} //end switch
```

From above, to link the input numbers to the respective cells, switch control structure is used. For example, if the user input number '1', the switch case '1' is valid, the system will then access case '1' and show the letter 'X' in the cell [0][0]. If the user inputs number '9', the system will display 'X' in the cell [2][2].

Next, it's the computer's turn to make a move. Therefore, we need to declare the *Random* function as shown below ("*Random randomNumber = new Random();*"). Then, we assign *computerMove* as a *randomNumber* with a range of 1 to 9. This will make every step for the CPU to be unpredictable and random:

```
public static void computerRandNo (char [][] board) {

   Random randomNumber = new Random(); //declare Random function
   int computerMove;

   while(true) {
      computerMove = randomNumber.nextInt(9) + 1; //obtain random number for computer's move

      if (emptyCells(board,Integer.toString(computerMove))) {
         break;
      }
   } //end while

   System.out.println("Computer (O) choose to move at " + computerMove + "\n");
   placeMove(board, Integer.toString(computerMove), 'O');
```

Since we need to make sure the CPU will not put a mark or 'O' in a cell that has already been used before, we will use switch control structure to exclude all the used cells so the CPU will not reuse the cells.

```
//method to display empty cells when necessary in CPU mode
public static boolean emptyCells(char[][] board, String place) {

   switch (place) {
      case "1":
         return (board[0][0] == ' ');
      case "2":
         return (board[0][1] == ' ');
      case "3":
         return (board[0][2] == ' ');
      case "4":
         return (board[1][0] == ' ');
      case "5":
         return (board[1][1] == ' ');
      case "6":
         return (board[1][2] == ' ');
      case "7":
         return (board[2][0] == ' ');
      case "8":
         return (board[2][1] == ' ');
      case "9":
         return (board[2][2] == ' ');
      default:
         return false;
      } //end switch
```

In this part, we have shown eight winning conditions: 3 vertical lines, 3 horizontal lines and 2 diagonal lines. *letter* represents either 'X' or 'O', which means when the player or the computer completes a line, it will return as true and if not, it will continue the game until there is no empty cell on the game board.

```
//method for CPU mode to check if someone has won the game, otherwise game continues
public static boolean someoneWon (char[][] board, char letter) {

    if((board[0][0] == letter && board[0][1] == letter && board[0][2] == letter ) ||
      (board[1][0] == letter && board[1][1] == letter && board[1][2] == letter ) ||
      (board[2][0] == letter && board[2][1] == letter && board[2][2] == letter ) ||

      (board[0][0] == letter && board[1][0] == letter && board[2][0] == letter ) ||
      (board[0][1] == letter && board[1][1] == letter && board[2][1] == letter ) ||
      (board[0][2] == letter && board[1][2] == letter && board[2][2] == letter ) ||

      (board[0][0] == letter && board[1][1] == letter && board[2][2] == letter ) ||
      (board[0][2] == letter && board[1][1] == letter && board[2][0] == letter ) )
      {
         return true;
      }

      return false;
```

The screenshot below shows when *letter* 'X' completes a row which fulfils the requirement above, it will show "Congratulations! You are the winner!!!". If it's the letter 'O' that completes a line, the computer wins and it will display "Computer wins! You lose. T.T". Otherwise, the game will continue until there is a winner, or it will stop when there is no more empty cell for players to fill in. The latter will display "The game ended in a draw."

```
public static boolean gameOver(char[][] board){

    if (someoneWon(board, 'X')) {
          printBoard(board);
          System.out.println("Congratulations! You are the winner!!!");
          return true;
    }

    if (someoneWon(board,'O')) {
          printBoard(board);
          System.out.println("Computer wins! You lose T.T");
          return true;
    }


    for (int i = 0;i<board.length; i++ ) {
       for (int j = 0; j < board[i].length; j++) {
           if (board[i][j] == ' ') {
             return false;
           }
       }
    }
    printBoard(board);
    System.out.println("The game ended in a draw.");
    return true;
```

After the game is over, the system will ask the player whether they want to play again. They will be requested to type '1' to play the game again or type any other number to go back to the main menu.

```
        System.out.println("Play again?");
        System.out.print("Enter 1 to continue playing or any number to exit to the main menu: ");
        option = input.nextInt();

        //if option != 1, return to main menu; otherwise loop cuurent method (game mode)
        if (option != 1) {
            System.out.println();
            return;
        }

    } while (option == 1);
    //end do-while for method loop
```

CPU mode does not contribute any final score to users.

3.6 Leaderboard

For the leaderboard to function, we used the array method to link *name1*, *name2* and *nameCPU* (which is just "CPU") to form an array *names[]*; and *score1*, *score2* and 0 into array *scores[]*.

```
String names[] = {name1, name2, nameCPU};
int scores[] = {score1, score2, 0};

do {
```

```
if ((score1 == 0) && (score2 == 0)) {
   System.out.printf(1 + "\t\t\t%-10s%5d%n", names[2], scores[2]);
   System.out.printf(1 + "\t\t\t%-10s%5d%n", name1, score1);
   System.out.printf(1 + "\t\t\t%-10s%5d%n", name2, score2);
}
else if (score1 > score2) {
   System.out.printf(1 + "\t\t\t%-10s%5d%n", name1, score1);
   System.out.printf(2 + "\t\t\t%-10s%5d%n", name2, score2);
   System.out.printf(3 + "\t\t\t%-10s%5d%n", names[2], scores[2]);
}
else if (score2 > score1) {
   System.out.printf(1 + "\t\t\t%-10s%5d%n", name2, score2);
   System.out.printf(2 + "\t\t\t%-10s%5d%n", name1, score1);
   System.out.printf(3 + "\t\t\t%-10s%5d%n", names[2], scores[2]);
}
else {
   System.out.printf(1 + "\t\t\t%-10s%5d%n", name1, score1);
   System.out.printf(1 + "\t\t\t%-10s%5d%n", name2, score2);
   System.out.printf(3 + "\t\t\t%-10s%5d%n", names[2], scores[2]);
}
```

From above, we use the if…else method to arrange the ranking. The first if-statement is the default arrangement when the programme has just started to run, where the leaderboard will print *names[2]* or "CPU" as the name and *scores[2]* or CPU score in first place, second goes to *name1* (*player1*'s name) and *name2* (*player2*'s name). After a game is played, and when *score1* is bigger than *score2* (*player1* wins more than *player2*), the rankings will display *name1* and *score1* at first place and *name2* and *score2* at second place. When *score2* is bigger than *score1* (*player2* wins more than *player1*), the rankings will display *name2* at first place and *name1* at second place. Moreover, when *score1* is the same as *score2*, it will display both players' names as first place and computer will be at third place.

The next part of the leaderboard is the search name function. The for loop that we used is to check if the user input number is the same with the integer *i* in the loop. If the input number is

found to be the same as *i* in the loop, the player number is found and it will print out the name of the player. Otherwise, the input is considered invalid.

```
//method to search the name of the selected player
public static void searchName(String[] names) {

   int flag = 0;
   int num;

   System.out.println("\n1\tPlayer 1");
   System.out.println("2\tPlayer 2");
   System.out.print("Enter the player number that you want to search: ");
   num = input.nextInt();

   for (int i=1; i <= names.length; i++) {
      if (i == num) {
         System.out.println("\nYou searched for Player " + i + ", their name is " + names[i-1]);
         flag = 1;
         break;
      }
   }

   if (flag == 0) {
      System.out.println("\nInvalid number!");
   }

   System.out.println("\nReturn to leaderboard?");
   System.out.print("Enter any number to proceed: ");
   num = input.nextInt();

} //end searchName()
```

Below is the code that will execute when users choose to reset the leaderboard. Again using the same concept, a for loop is used to read each value in the arrays *names[]* and *scores[]*. Then, for each passing value, it sets the existing values back to their default values, which are ‘null’ and ‘0’ respectively.

```
//method to reset the leaderboard
public static void resetLeaderboard(String[] names, int[] scores) {

   for (int i=0; i < names.length; i++) {
      names[i] = "null";
      scores[i] = 0;
   }

} //end resetLeaderboard()
```

4.0 APPENDIX

4.1 Workload

Workload was delegated to each member of the group from time to time. Full details of the place and time where tasks are divided, do refer to section **4.2 (Meeting Logs)**. Anyhow, below is the overall tasks that each member has done throughout the time period of the project's completion:

Name	Matric Number	Task

Lee Juan	280027	- Code:
		  - 2-Player Easy Mode
		  - Leaderboard and high score related functions
		- Report writing
		  - Requirements
		  - Flowchart

Muhammad Zuhair Afham bin Mohd Nasir	280782	- Code:
		  - 2-Player Medium Mode
		  - Tips function
		- Report writing
		  - User manual
		  - Introduction
		- Video editing

Edwin Lim Wei Bin	281775	- Code:
		  - Menu
		  - CPU Mode
		- Report writing
		  - Requirements
		  - Flowchart

* Further miscellaneous adjustments in both the code, video, and report were being made by all members of the group.

4.2 Meeting Logs

Three official group meetings or discussions have taken place: first two meetings were being held online, and the final meeting was held physically on campus. Below are the meeting logs that further explain details about the meetings conducted:

GROUP DISCUSSION 1

Thursday, 26 May 2022, 12:30 - 2:30 pm, WhatsApp

Attendees:
Lee Juan (280027)
Muhammad Zuhair Afham Bin Mohd Nasir (280782)
Edwin Lim Wei Bin (281775)

Absentees:
None

Discussed:

1. Plan
    a. No set future meetings as for now, but all members will have to text the WhatsApp group for any updates or questions in regard to the project.
    b. Created a CodeCollab project for every member to view and modify the code together.
    c. Planned to begin the project by coding 2-player mode since it is the easiest to code compared to the other modes.
2. Work division
    a. No tasks are delegated at the end of the meeting, which will be done after the midterm test on 31 May so that all members can concentrate on the test first.

GROUP DISCUSSION 2

Friday, 10 June 2022, 9:30 - 10:00 pm, WhatsApp

Attendees:

Lee Juan (280027)

Muhammad Zuhair Afham Bin Mohd Nasir (280782)

Edwin Lim Wei Bin (281775)

Absentees:

None

Discussed:

1. Main tasks
    a. Divided tasks to all members according to the leader's instructions.
    b. Started to write the codes on easy mode and medium of the Tic Tac Toe game at the CodeCollab.
2. Work division
    a. Juan - will code 2-player easy mode
    b. Zuhair - will code 2-player medium mode
    c. Edwin - will code CPU mode
3. Miscellaneous
    - All members should update their coding progress into the CodeCollab.io project every Thursday until further instructions are made.

GROUP DISCUSSION 3

Friday, 8 July 2022, 5:30 – 7:00 pm, Juan's dorm

Attendees:

Lee Juan (280027)

Muhammad Zuhair Afham Bin Mohd Nasir (280782)

Edwin Lim Wei Bin (281775)

Absentees:

None

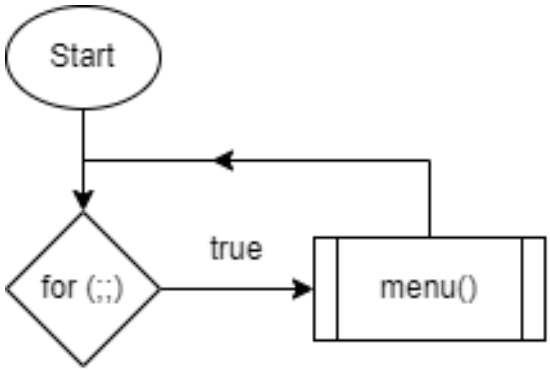
Discussed:

1. Progress
    a. The flow for the menu and 2-player easy mode are done, and will plan on working with the interface to make it look nicer.
    b. The coding of CPU mode and leaderboard are still in progress.
    c. Enable and disable tips function is now working.

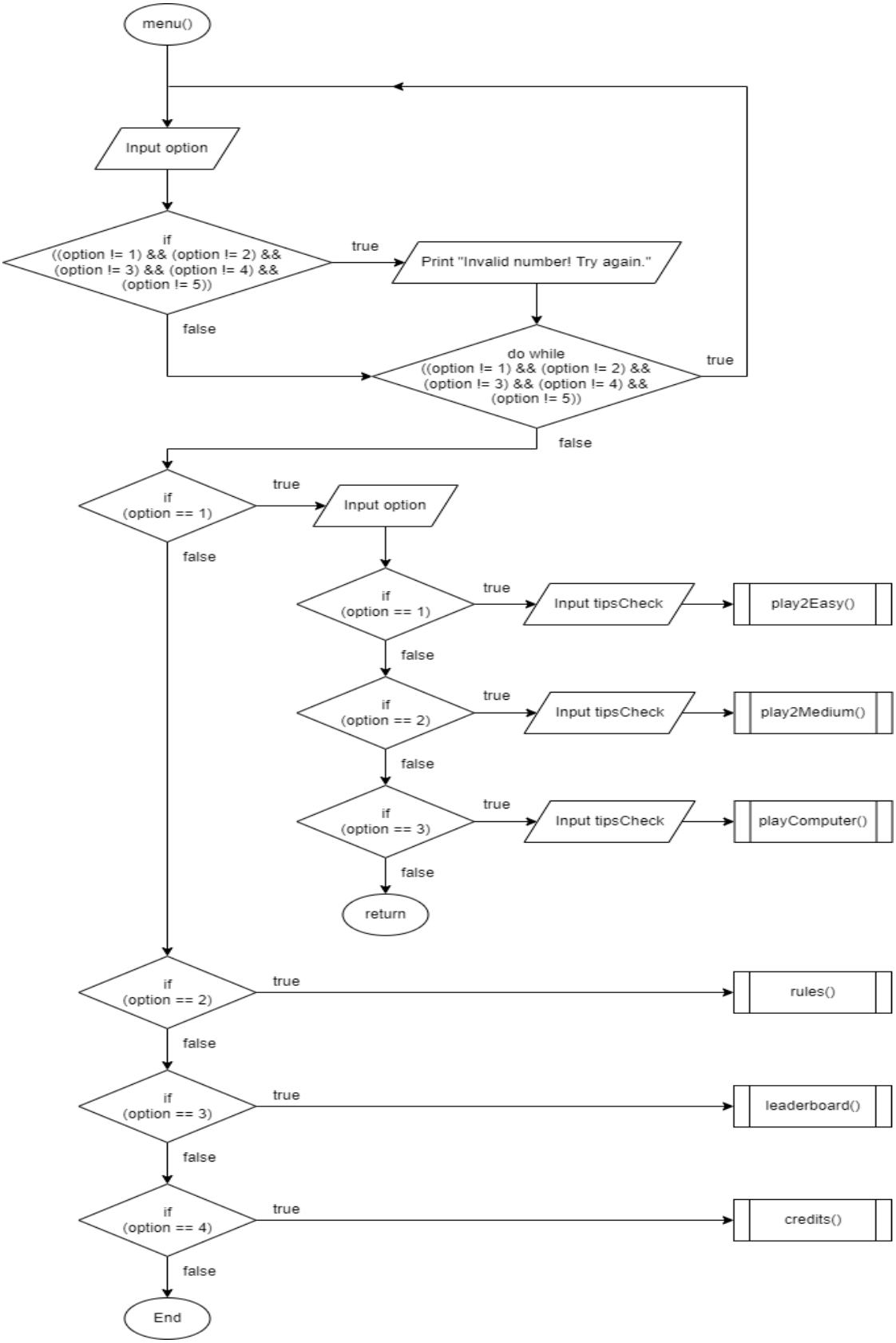
2. Work division
    a. Juan - Continue with interface and overall programme execution flow
    b. Zuhair - Prepare outline for video presentation and begin writing introduction and user manual for report
    c. Edwin - Prepare outline for report and construct flowcharts for completed methods

4.3 Flowchart

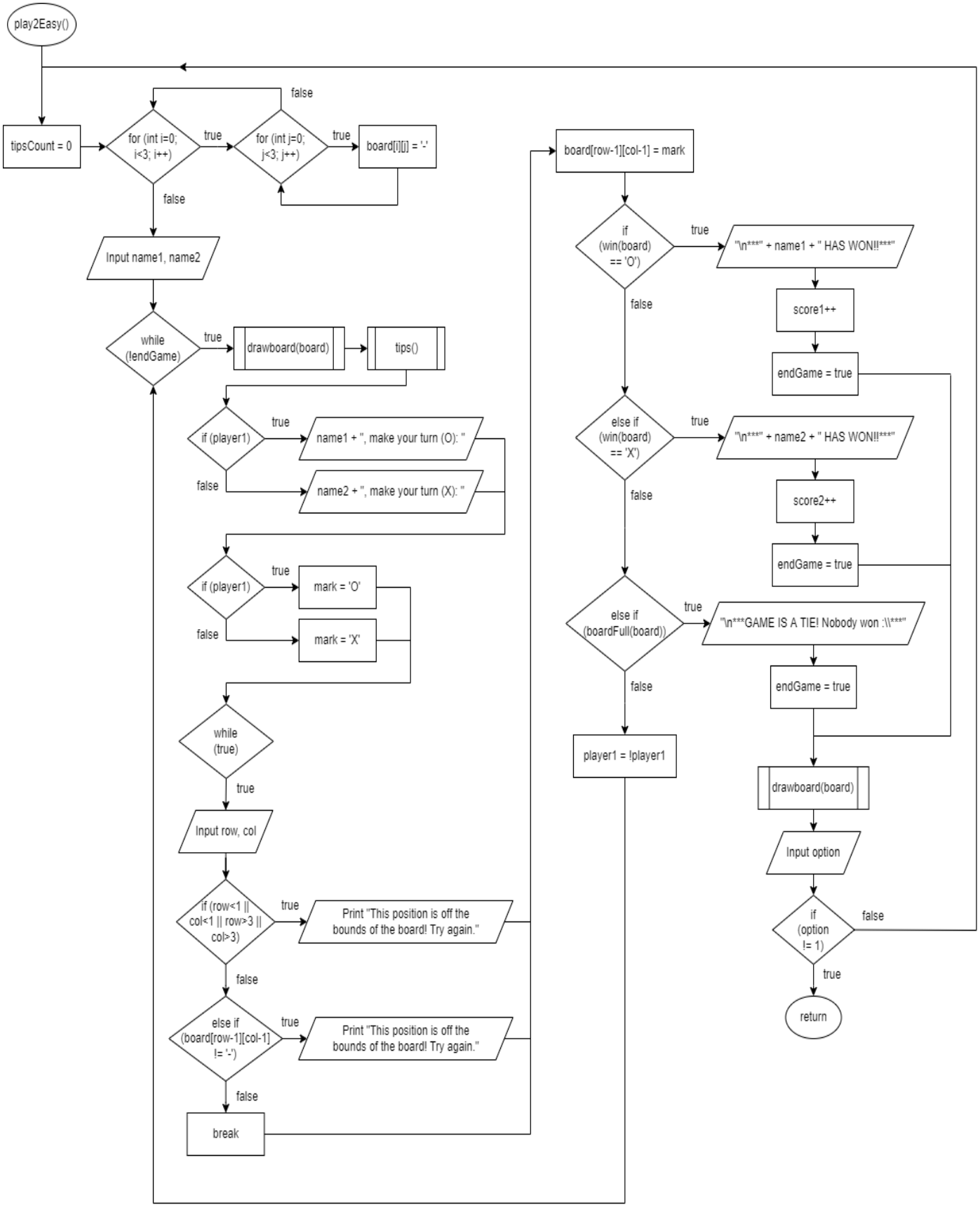
Figures attached below are the flowcharts for our Tic Tac Toe project, separated and categorised depending on each method.



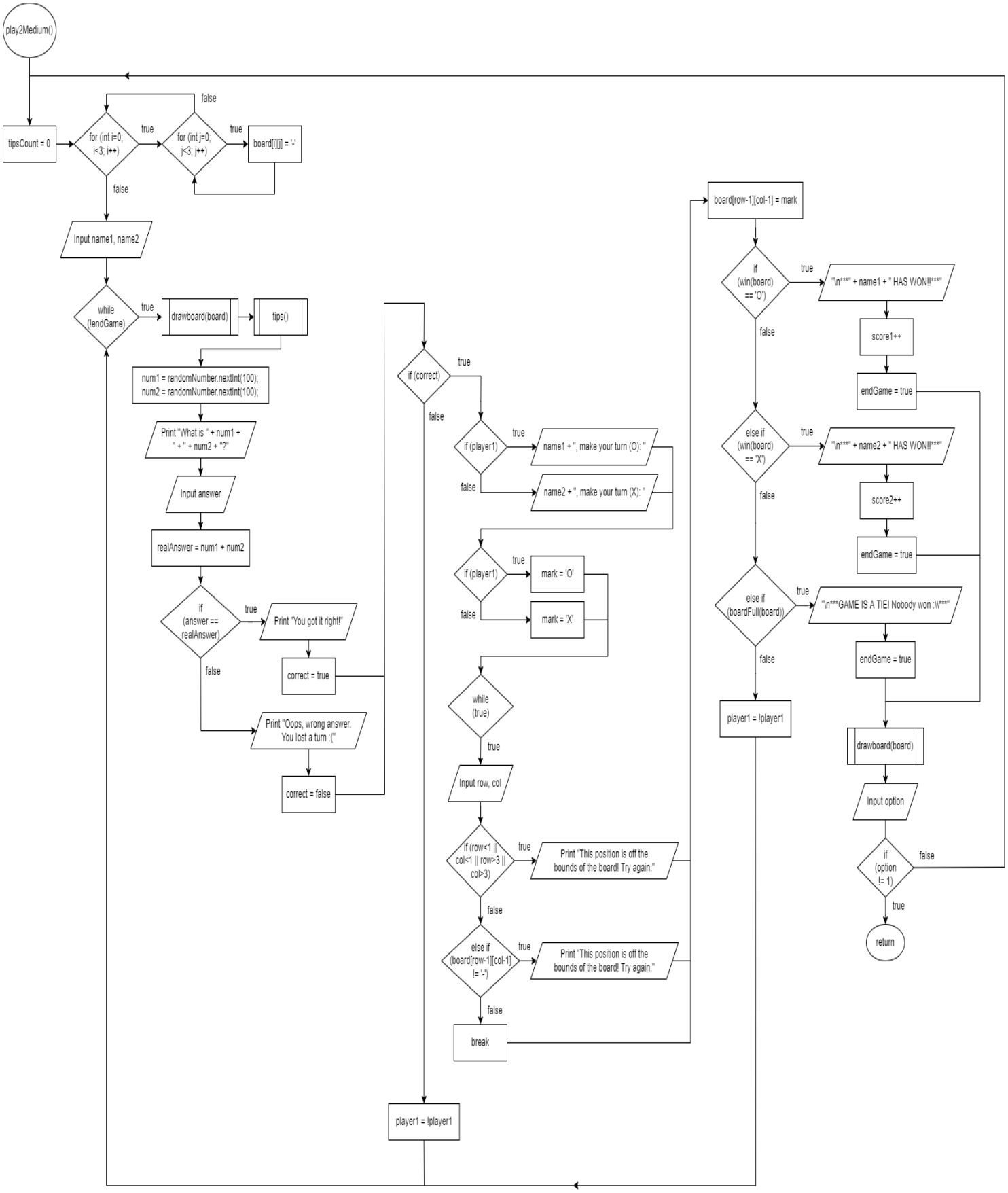
Flowchart for *main()* method



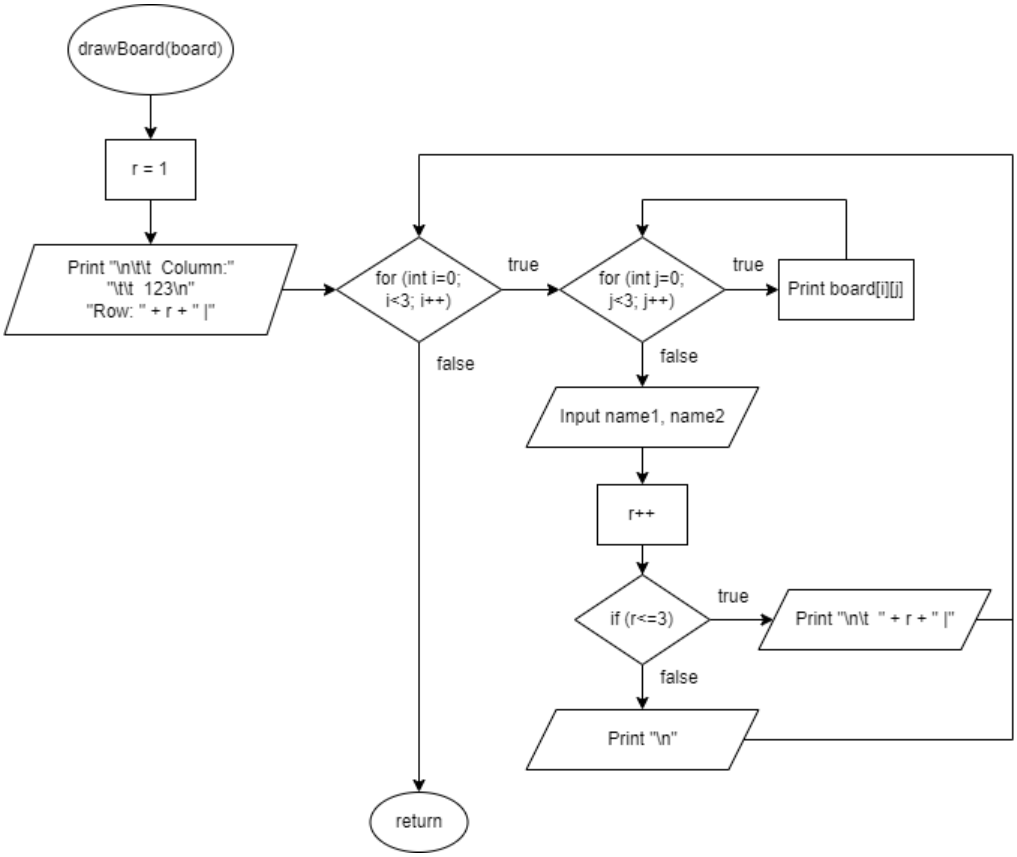
Flowchart for *menu()* method



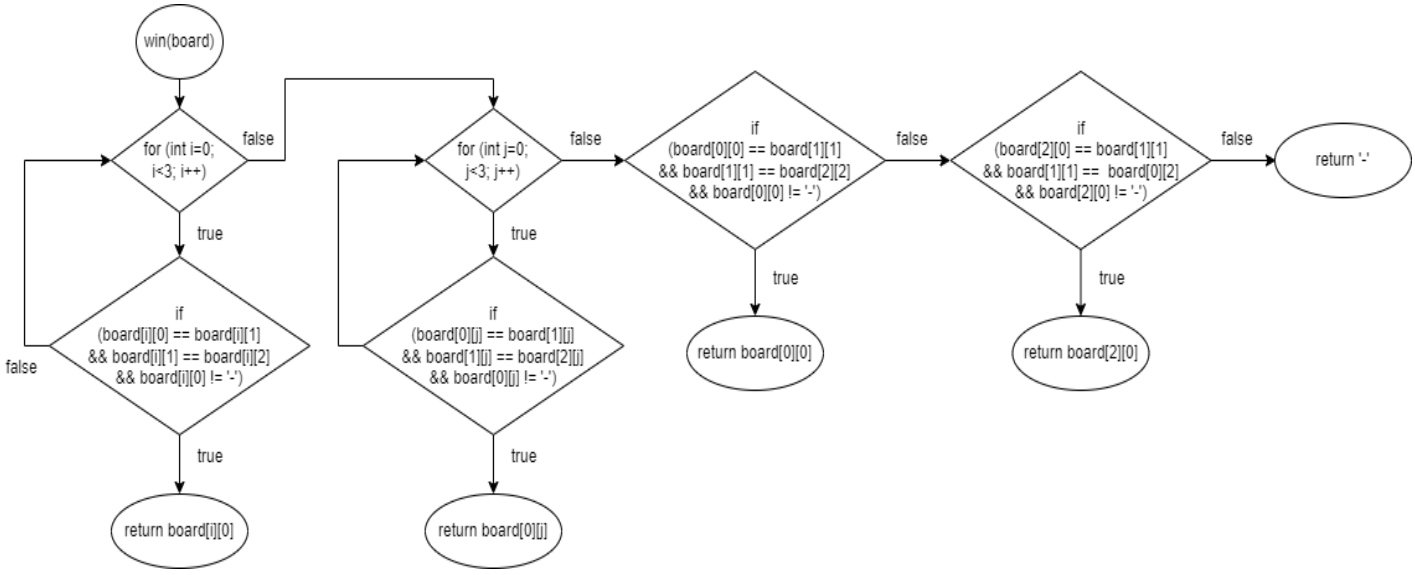
Flowchart for *play2Easy()* method



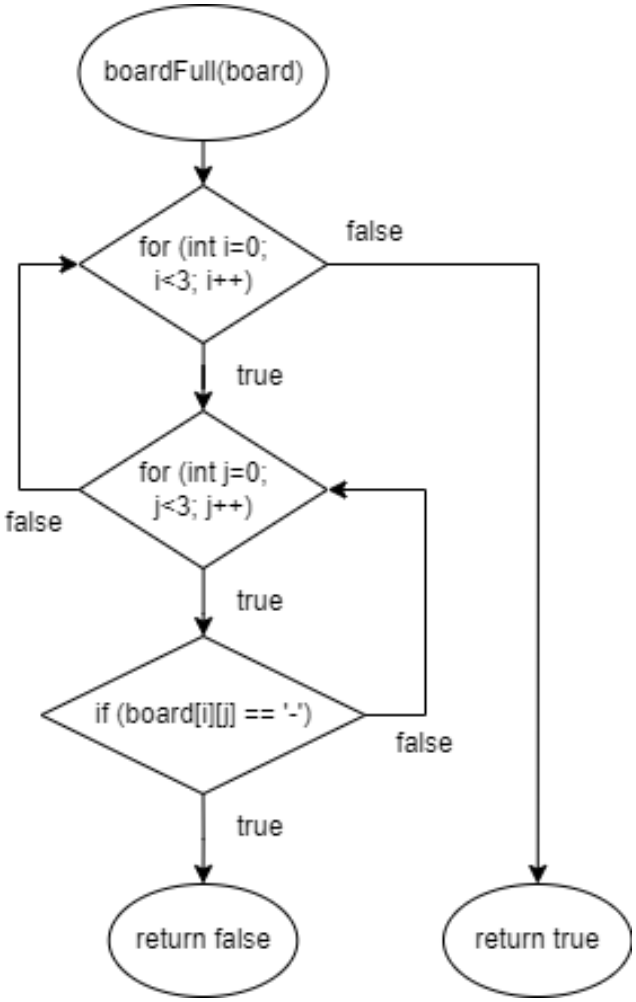
Flowchart for *play2Medium()* method



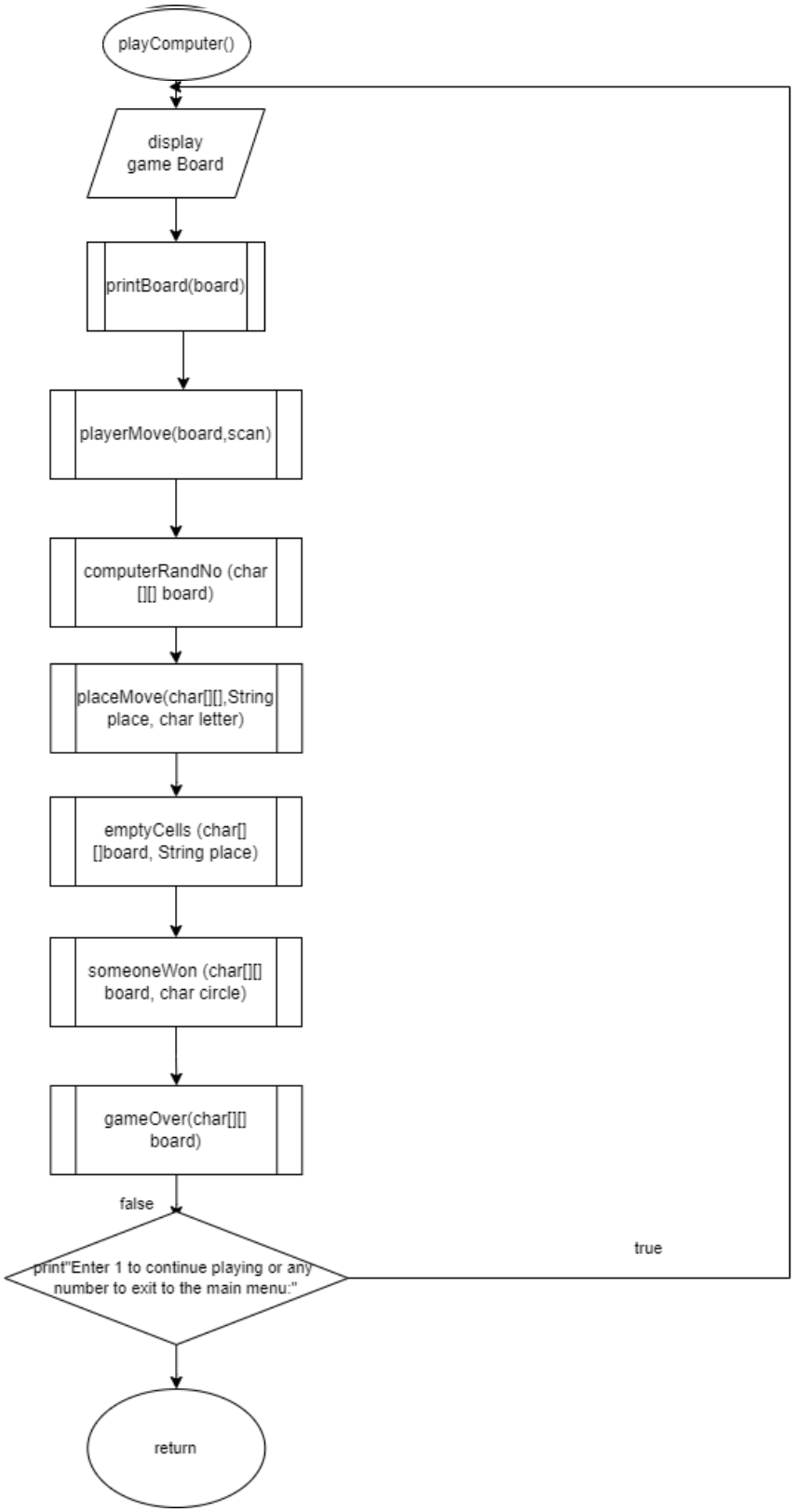
Flowchart for *drawBoard(board)* method



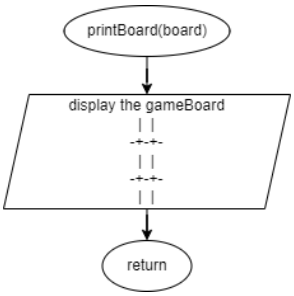
Flowchart for *win(board)* method



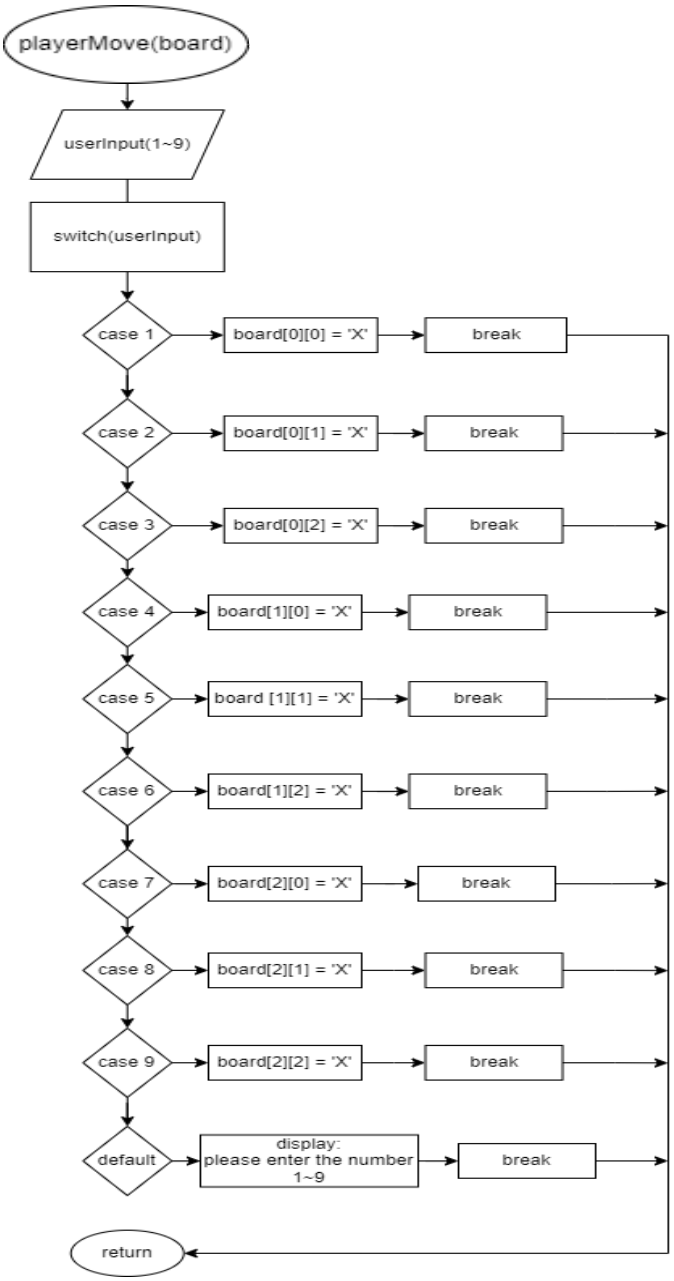
Flowchart for *boardFull(board)* method



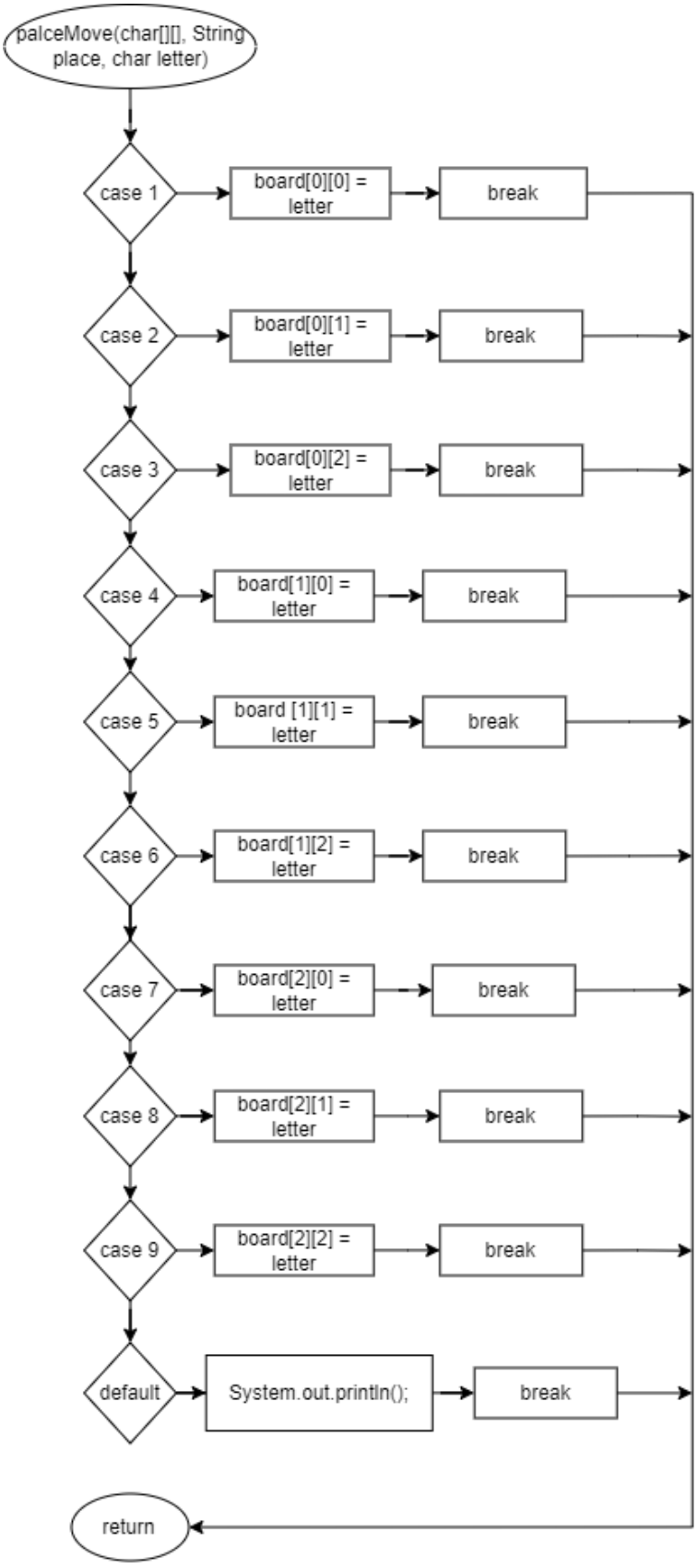
Flowchart for *playComputer()* method



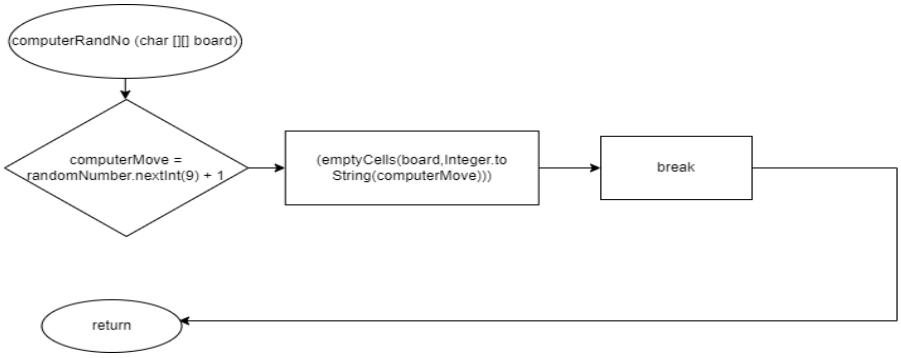
Flowchart for *printBoard(board)* method



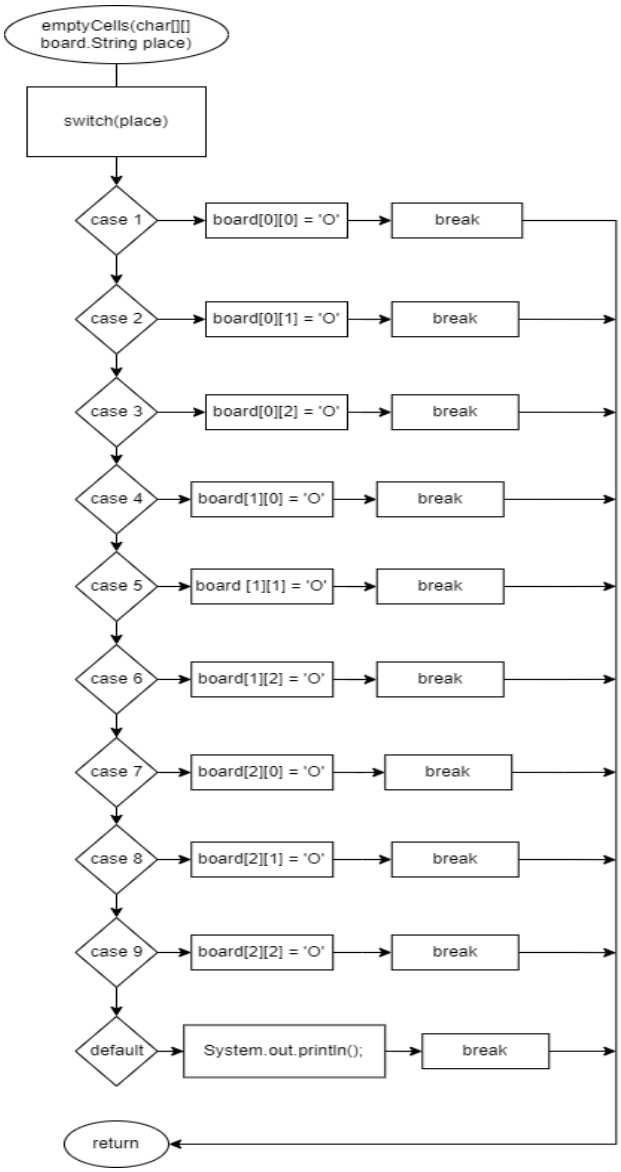
Flowchart for *playerMove(board)* method



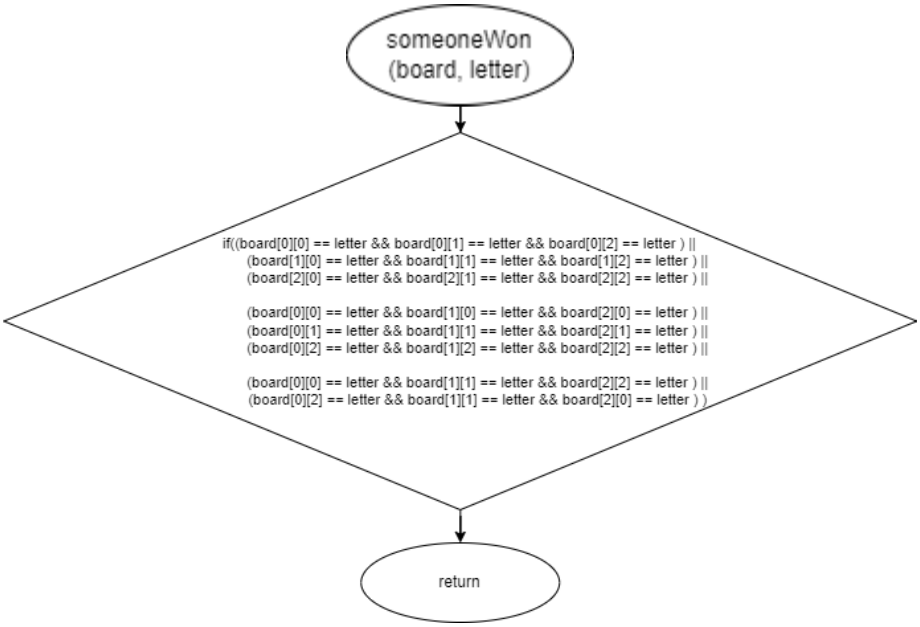
Flowchart for *placeMove(board, place, letter)* method



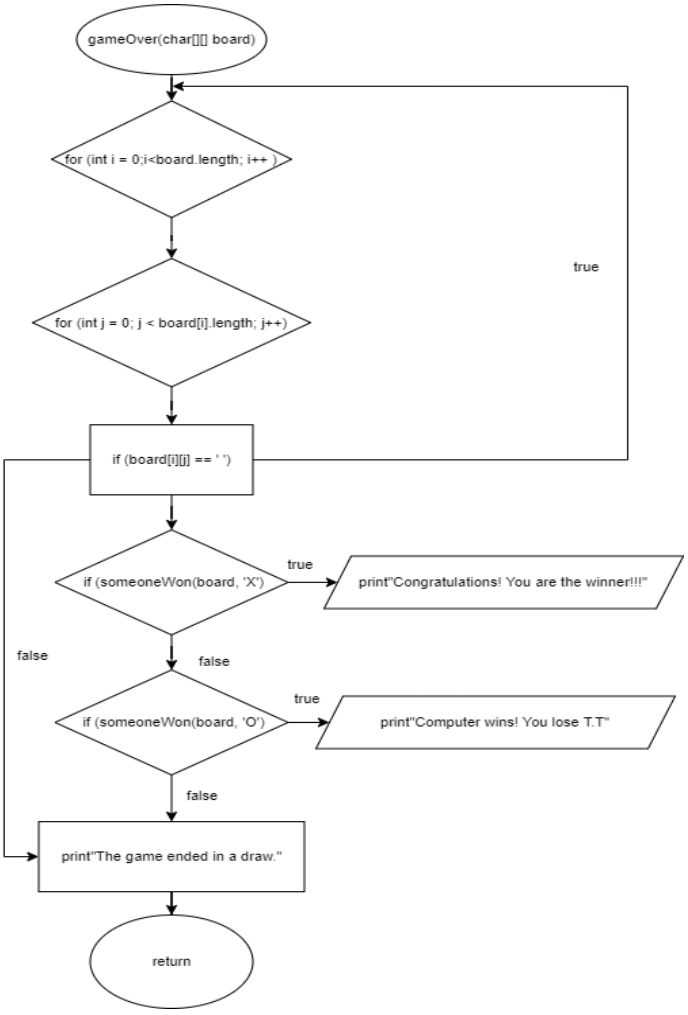
Flowchart for *computerRandNo(board)* method



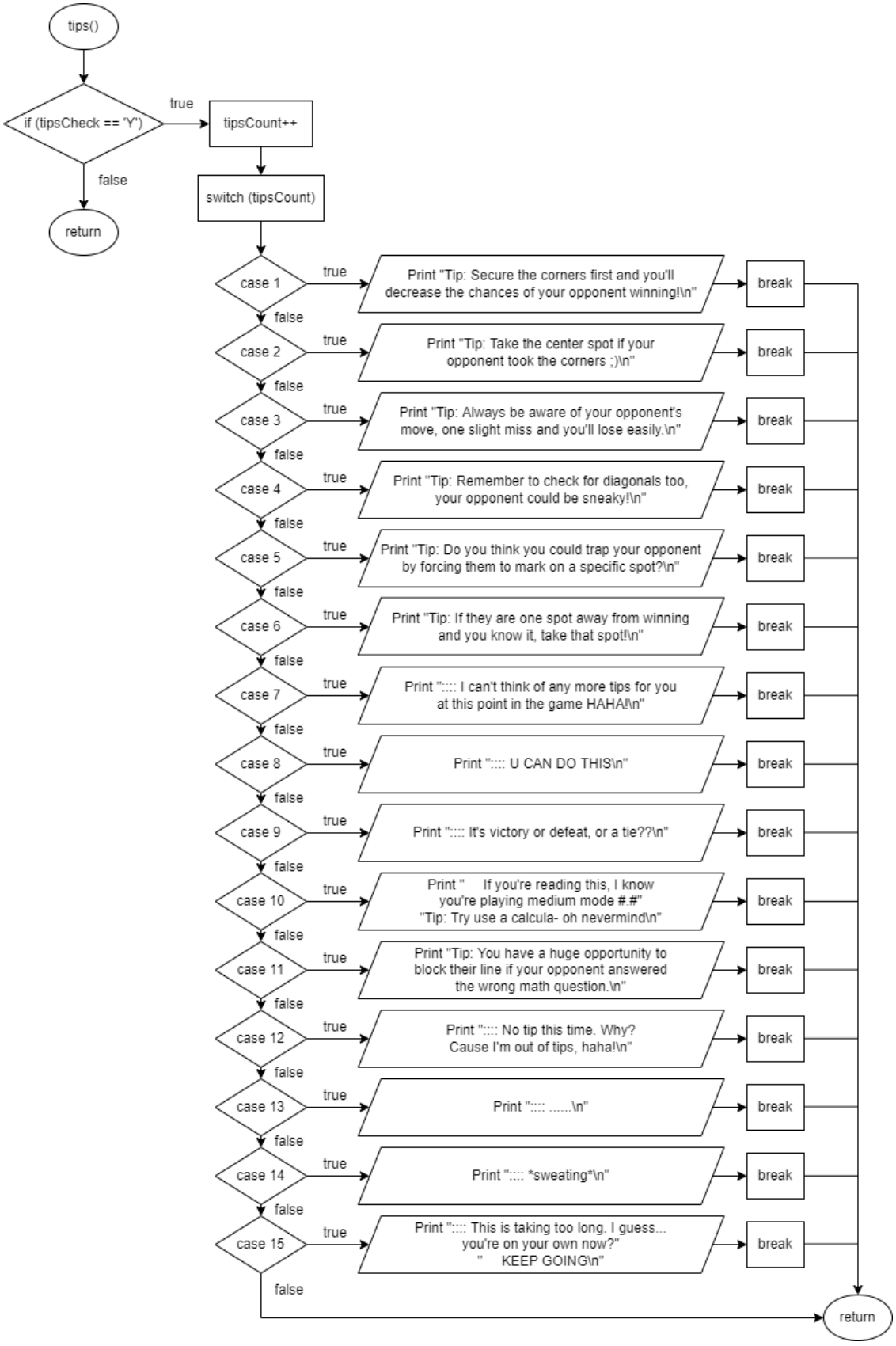
Flowchart for *emptyCells(board, place)* method



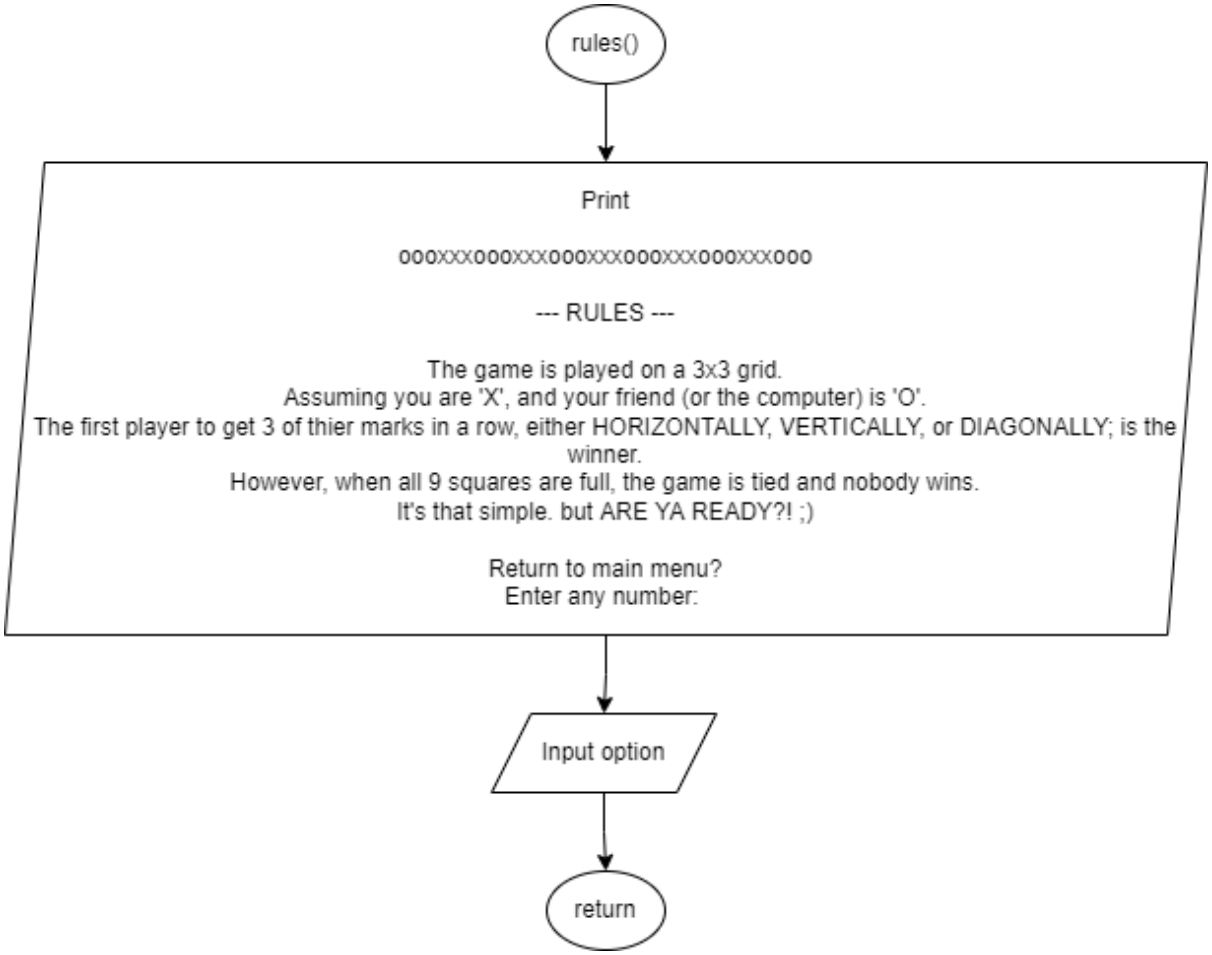
Flowchart for *someoneWon(board, letter)* method



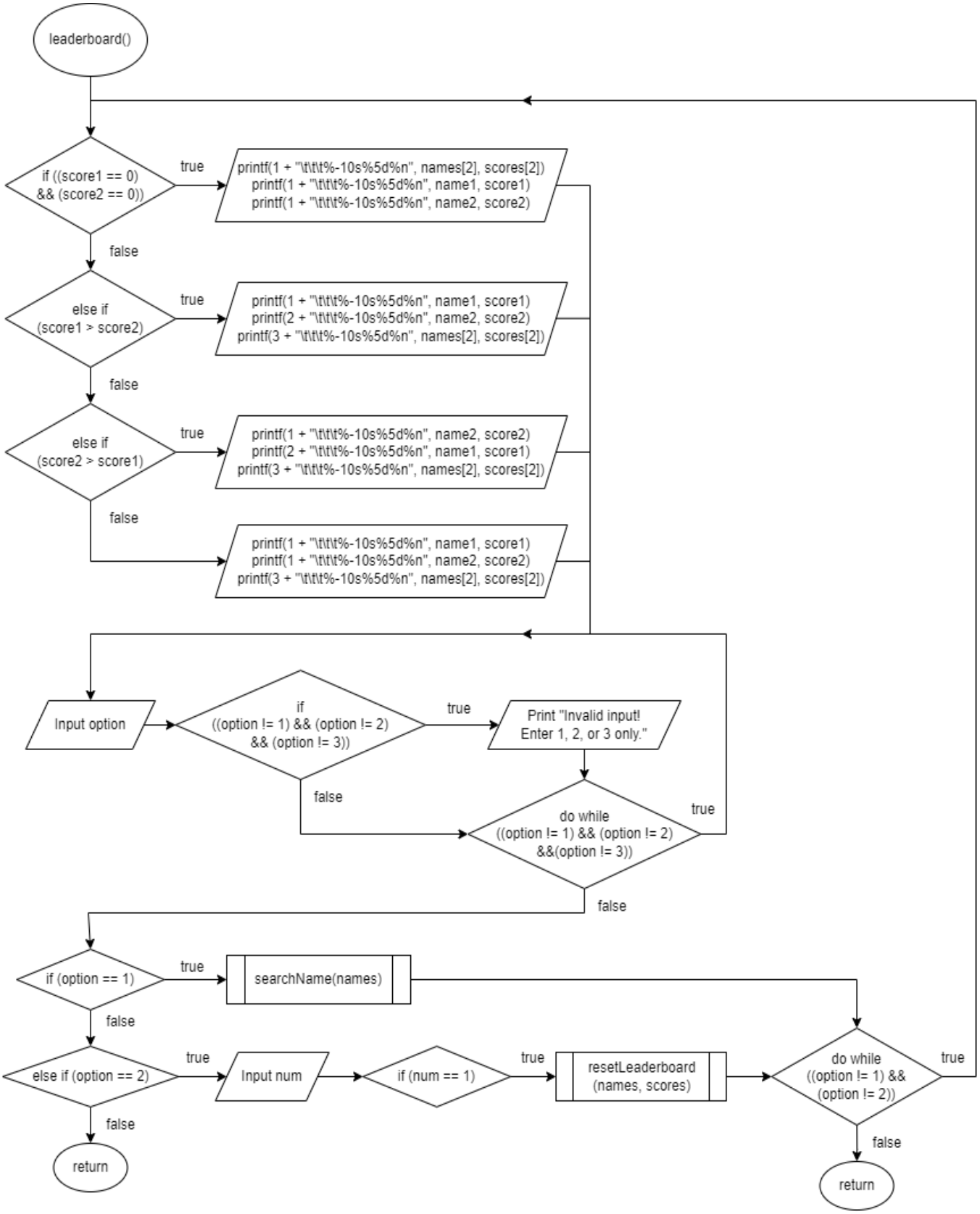
Flowchart for *gameOver(board)* method



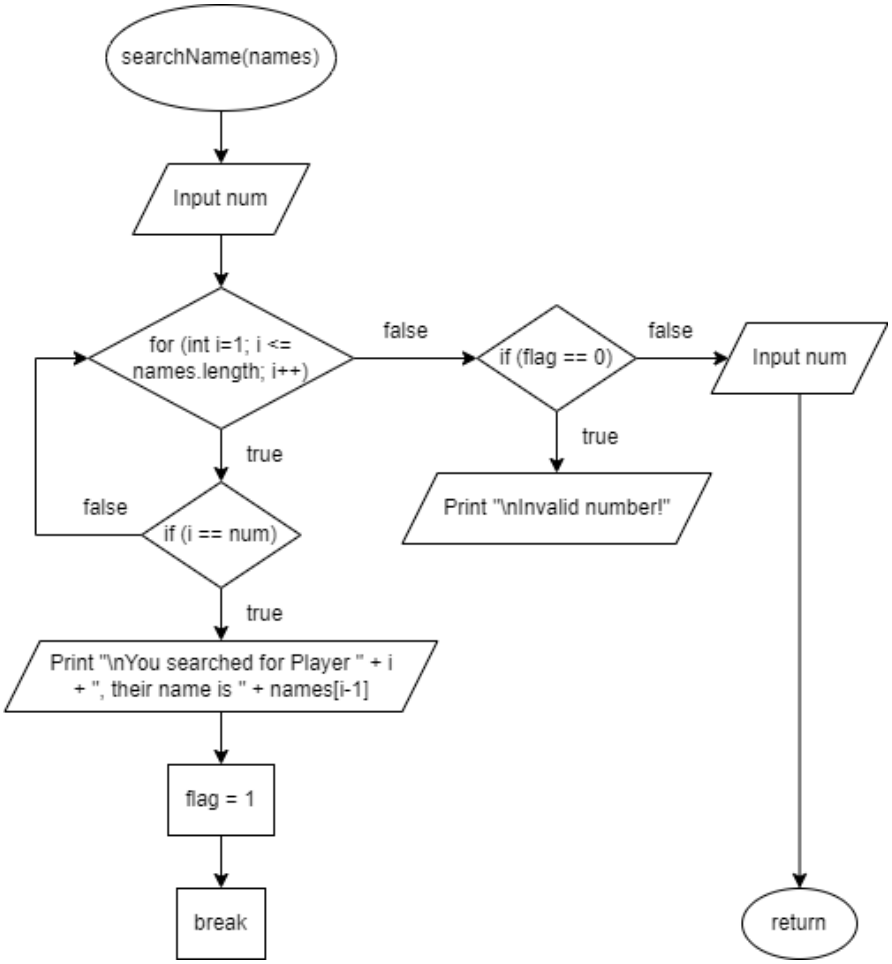
Flowchart for *tips()* method



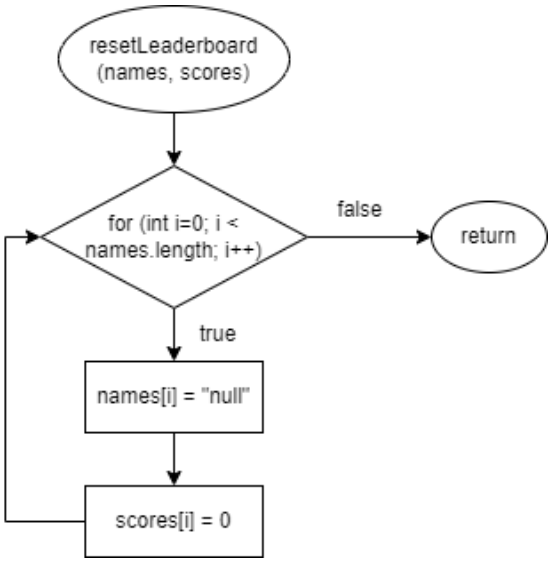
Flowchart for *rules()* method



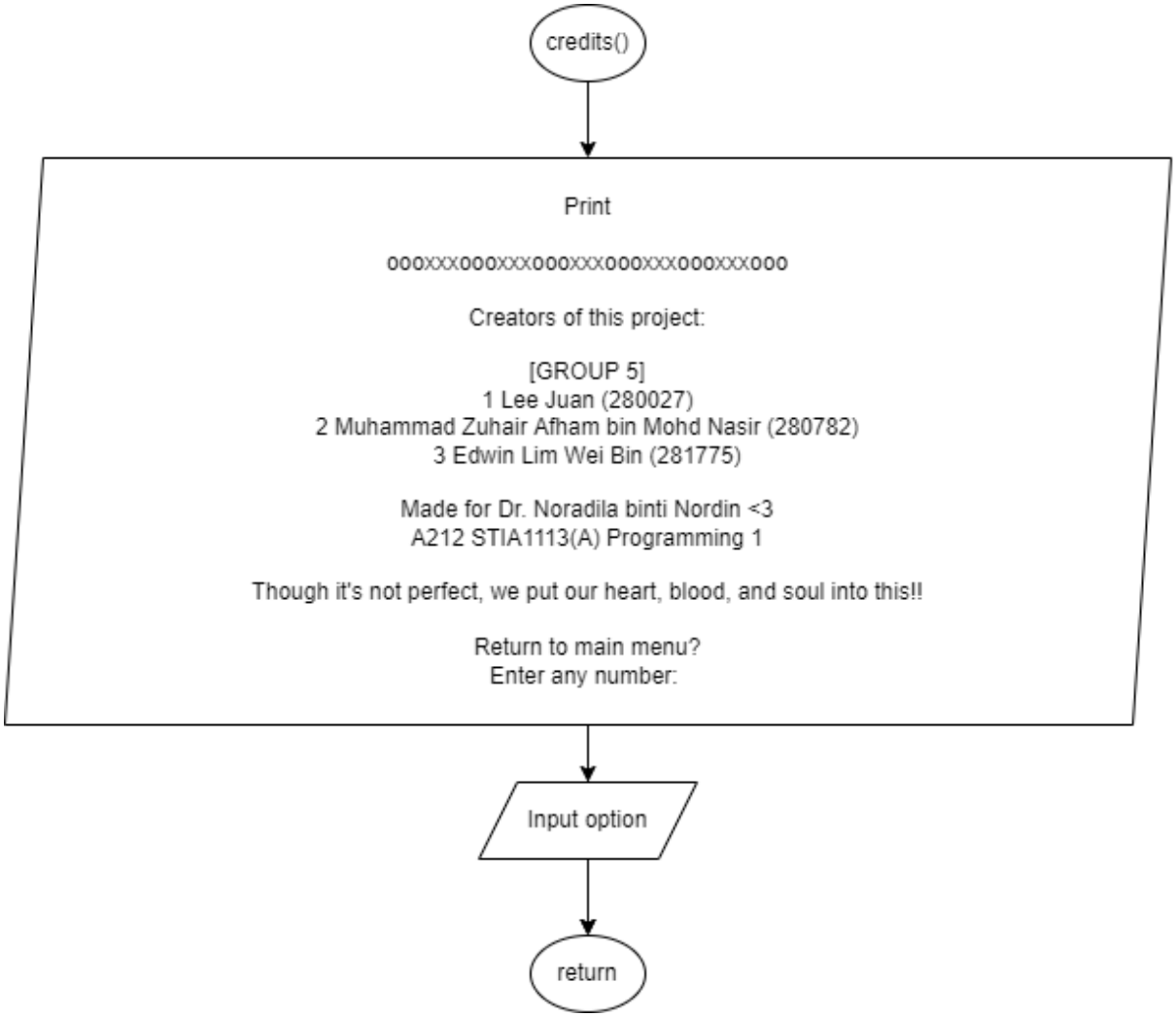
Flowchart for *leaderboard()* method



Flowchart for *searchName(names)* method



Flowchart for *resetLeaderboard(names, scores)* method



Flowchart for *credits()* method

4.4 Complete Source Code

Attached below is the complete source code for our Tic Tac Toe project:

```
/*
A212 STIA1113 Programming 1

Group 5 Members:
Lee Juan (280027)
Muhammad Zuhair Afham bin Mohd Nasir (280782)
Edwin Lim Wei Bin (281775)

Date of completion: 14 July 2022

--------

This programme is a Tic Tac Toe game that consists of two 2-player (easy and medium)
modes and CPU mode along with
a tips function that could provide some suggestions to the player during gameplay.
This programme also has a
simple high score and leaderboard function that would let players view their
rankings in the game.

*/


import java.util.Scanner;
import java.util.Random;   //"Random" utility needed for CPU to make decisions in
CPU mode


public class Group5TicTacToe {


   public static Scanner input = new Scanner(System.in);           //global
variable for creating input function
   public static int option;                                       //global
variable for option input
   public static char tipsCheck;                                   //tipsCheck = Y
if tips are enabled
   public static String name1, name2, nameCPU = "CPU";
   public static int score1 = 0, score2 = 0, tipsCount;


   //main method
   public static void main(String[] myVariable) {

       for (;;) {    //infinite loop to always return to menu whenever a mode ends
       menu();
       }

   } //end main


   //------------------------------------------- MENU -------------------------
-------------------
```

```
   //method for main menu display
   public static void menu() {

       System.out.println("oooxxxoooxxxoooxxxoooxxxoooxxxooo\n");
       System.out.println("HELLO there, i see you're bored :)\nWelcome to our Tic
Tac Toe!\n");
       System.out.println("1\tPLAY");
       System.out.println("2\tHow to Play?");
       System.out.println("3\tLeaderboard");
       System.out.println("4\tCredits");
       System.out.println("5\tEXIT GAME");

       do {
       System.out.println("\nWhat do you want to do?");
       System.out.print("Enter a number: ");
       option = input.nextInt();

       if ((option != 1) && (option != 2) && (option != 3) && (option != 4) &&
(option != 5)) {
              System.out.println("Invalid number! Try again.");
       }

       } while ((option != 1) && (option != 2) && (option != 3) && (option != 4) &&
(option != 5));     //do-while loop if user input is invalid

       //where user will proceed for each input option
       if (option == 1) {

       System.out.println("\n-------------\n\n1\t2-PLAYER:        EASY        MODE
\t('>')/*\\('<')");
       System.out.println("2\t2-PLAYER: MEDIUM MODE \t(\">\")/***\\(\"<\")");
       System.out.println("3\tCPU MODE  \t\t\t\t\t(@_@)");
       System.out.println("4\tBack to main menu");
       System.out.print("\nEnter the mode that you want to play, or back to menu:
");
       option = input.nextInt();

       if (option == 1) {
              System.out.println("\nEnable TIPS during gameplay?");
              System.out.println("(it might include some eXtRa cOmmEntaRy too :P)");
              System.out.print("Enter \"Y\" to enable, or any other alphabet to
disable tips: ");
              tipsCheck = input.next().toUpperCase().charAt(0);

              play2Easy();
       }
       else if (option == 2) {
              System.out.println("\nEnable TIPS during gameplay?");
              System.out.println("(it might include some eXtRa cOmmEntaRy too :P)");
              System.out.print("Enter \"Y\" to enable, or any other alphabet to
disable tips: ");
              tipsCheck = input.next().toUpperCase().charAt(0);

              play2Medium();
       }
         else if (option == 3) {
              System.out.println("\nEnable *TIPS* during gameplay?");
              System.out.print("Enter \"Y\" to enable, or any other alphabet to
disable tips: ");
              tipsCheck = input.next().toUpperCase().charAt(0);

              playComputer();
```

```
        }
        else {
                System.out.println();
                return;
        }

        }
        else if (option == 2) {
        rules();
        }
        else if (option == 3) {
        leaderboard();
        }
        else if (option == 4) {
        credits();
        }
        else {
        System.out.println("\nSee ya!");
        System.exit(0);                    //end programme
        }
    } //end menu()


    //------------------------------------------- METHODS FOR 2-PLAYER MODES ----
----------------------------------------


    //method for 2-player easy mode
    public static void play2Easy() {

        do {

        System.out.println("\nooooxxxoooxxxoooxxxoooxxxoooxxxooo\n");
        System.out.println("--- 2-PLAYER: EASY MODE   ('>')/*\\('<') ---\nIt's more
fun playing with friends, right!\n");

        boolean player1 = true;            //player 1 or 2's turn
        boolean endGame = false;           //boolean  that  is  false  in  default,
otherwise game ends
        char mark;                         //Xs or Os
        int row, col;                      //row and column number, input by user
        char[][] board = new char[3][3]; //array for 2D-board

        tipsCount = 0;              //reset tipsCount

        //assign initial char '-' to board
        for (int i=0; i<3; i++) {
                for (int j=0; j<3; j++) {
                board[i][j] = '-';
                 }
        }

        System.out.println("You ready? LET'S GO!");
        System.out.print("Player 1, what is your name?: ");
        name1 = input.next();
        System.out.print("Player 2, what is your name?: ");
        name2 = input.next();

        //while loop when endGame is not true
        while (!endGame) {

                drawBoard(board);
                tips();
```

```
            if (player1) {
            System.out.println(name1 + ", make your turn (O): ");
            }
            else {
            System.out.println(name2 + ", make your turn (X): ");
            }

            if (player1) {
            mark = 'O';
            }
            else {
            mark = 'X';
          }

            while (true) { //loop if user input is invalid
            System.out.print("Enter a row number (1, 2, or 3): ");
            row = input.nextInt();
            System.out.print("Enter a column number (1, 2, or 3): ");
            col = input.nextInt();

            //check validity of row and col input
            if (row<1 || col<1 || row>3 || col>3) {
                  System.out.println("This position is off the bounds of the
board! Try again.");
            }
            else if (board[row-1][col-1] != '-') {
                  System.out.println("Someone has already made a move at this
position! Try again.");
            }
            else {
                  break;
            }

            }//end while(true)

            //assign the position of row and col to mark Xs or Os to the board
            board[row-1][col-1] = mark;

            //check if any player has won or if the game is a tie
            if (win(board) == 'O') {
            System.out.println("\n***" + name1 + " HAS WON!!***");
            score1++;
            endGame = true;
            }
            else if (win(board) == 'X') {
            System.out.println("\n***" + name2 + " HAS WON!!***");
            score2++;
            endGame = true;
            }
            else if (boardFull(board)) {
            System.out.println("\n***GAME IS A TIE! Nobody won :\\***");
            endGame = true;
            }
            else {
            player1 = !player1;        //continue to next player's turn if either
conditions are true
            }

      }//end while (!endGame)

          drawBoard(board);
```

```
        System.out.println("Play again?");
        System.out.print("Enter 1 to continue playing or any number to exit to the
main menu: ");
        option = input.nextInt();

        //if option != 1, return to main menu; otherwise loop cuurent method (game
mode)
        if (option != 1) {
                System.out.println();
                return;
        }

        } while (option == 1);
        //end do-while for method loop

    } //end play2Easy() or 2-player easy mode


    //method for 2-player medium mode
    public static void play2Medium() {

        do {

        System.out.println("\nooooxxxoooxxxoooxxxoooxxxoooxxxooo\n");
        System.out.println("---  2-PLAYER:  NORMAL  MODE    (\">\")/***\\(\"<\")  ---
\nReady for a challenge?\n");

        boolean player1 = true;
        boolean endGame = false;
        char mark;
        int row, col;
        char[][] board = new char[3][3];
        boolean correct;                    //added a boolean to check if input answer
is correct

        tipsCount = 0;              //reset tipsCount

        //assign initial char '-' to board
        for (int i=0; i<3; i++) {
                for (int j=0; j<3; j++) {
                board[i][j] = '-';
                }
        }

        System.out.println("You ready? LET'S GO!");
        System.out.print("Player 1, what is your name?: ");
        name1 = input.next();
        System.out.print("Player 2, what is your name?: ");
        name2 = input.next();

        //while loop when endGame is not true
        while (!endGame) {

                drawBoard(board);
                tips();

                Random randomNumber = new Random();
                int      num1      =      randomNumber.nextInt(100),      num2      =
randomNumber.nextInt(100); //initialise random numbers between 0-100 to num1 and
num2
                int answer, realAnswer;

                System.out.println("Answer  a  question  correctly  to  make  a  move,
```

```
otherwise, you skip a turn :D");
              System.out.println("What is " + num1 + " + " + num2 + "?");
       //question on addition of two random numbers

           if (player1) {
              System.out.print(name1 + ", enter your answer: ");
              }
              else {
              System.out.print(name2 + ", enter your answer: ");
              }
              answer = input.nextInt();

              realAnswer = num1 + num2;

              if (answer == realAnswer) {
              System.out.println("You got it right!");
              correct = true;
              }
              else {
              System.out.println("Oops, wrong answer. You lost a turn :(");
              correct = false;
              }

              //if answer is correct, proceed with turn
              if (correct) {
              if (player1) {
                   System.out.println(name1 + ", make your turn (O): ");
              }
              else {
                   System.out.println(name2 + ", make your turn (X): ");
              }

              if (player1) {
                     mark = 'O';
              }
              else {
                  mark = 'X';
              }

              while (true) { //loop if user input is invalid
                   System.out.print("Enter a row number (1, 2, or 3): ");
                     row = input.nextInt();
                     System.out.print("Enter a column number (1, 2, or 3): ");
                     col = input.nextInt();

                     //check validity of row and col input
                     if (row<1 || col<1 || row>3 || col>3) {
                     System.out.println("This position is off the bounds of the
board! Try again.");
                     }
                     else if (board[row-1][col-1] != '-') {
                     System.out.println("Someone has already made a move at this
position! Try again.");
                     }
                     else {
                     break;
                     }

              }//end while(true)

              //assign the position of row and col to mark Xs or Os to the board
              board[row-1][col-1] = mark;
```

```
            //check if any player has won or if the game is a tie
            if (win(board) == 'O') {
                  System.out.println("\n***" + name1 + " HAS WON!!***");
                  score1++;
                  endGame = true;
            }
            else if (win(board) == 'X') {
                  System.out.println("\n***" + name2 + " HAS WON!!***");
                  score2++;
                  endGame = true;
            }
            else if (boardFull(board)) {
                  System.out.println("\n***GAME IS A TIE! Nobody won :\\***");
                  endGame = true;
            }
            else {
                player1 = !player1;          //continue to next player's turn if
either conditions are true
            }

            }
            //lose turn if answer is wrong, switch to next player's turn
            else {
            player1 = !player1;
            }

      }//end while (!endGame)

      drawBoard(board);

      System.out.println("Play again?");
      System.out.print("Enter 1 to continue playing or any number to exit to the
main menu: ");
      option = input.nextInt();

      //if option != 1, return to main menu; otherwise loop cuurent method (game
mode)
      if (option != 1) {
            System.out.println();
            return;
      }

      } while (option == 1);
      //end do-while for method loop

   } //end play2Medium() or 2-player medium mode


//method to draw board in 2-player mode
   public static void drawBoard(char[][] board) {

      int r = 1;
      System.out.println("\n\t\t  Column:");
      System.out.println("\t\t  123\n");
      System.out.print("Row: " + r + " |");

      for (int i=0; i<3; i++) {
      for (int j=0; j<3; j++) {
            System.out.print(board[i][j]);
      }

      System.out.print("|");
      r++;
```

```
        if (r<=3) {
                System.out.print("\n\t  " + r + " |");
        }
        else {
                System.out.println("\n");
        }

        } //end for loop

    } //end drawBoard()


    //method to check whether someone has won in 2-player mode
    public static char win(char[][] board) {

        //check for each row (horizontally)
        for (int i=0; i<3; i++) {
        if   (board[i][0]   ==   board[i][1]   &&   board[i][1]   ==   board[i][2]   &&
board[i][0] != '-') {
                return board[i][0];
        }
        }

        //check for each column (vertically)
        for (int j=0; j<3; j++) {
        if   (board[0][j]   ==   board[1][j]   &&   board[1][j]   ==   board[2][j]   &&
board[0][j] != '-') {
                return board[0][j];
        }
        }

        //check for diagonals
        if   (board[0][0]   ==   board[1][1]   &&   board[1][1]   ==   board[2][2]   &&
board[0][0] != '-') {
        return board[0][0];
        }
        if(board[2][0]   ==   board[1][1]   &&   board[1][1]   ==     board[0][2]   &&
board[2][0] != '-') {
        return board[2][0];
        }

        //otherwise nobody wins yet, game continues
        return '-';

    } //end win()

    //method to check if entire board is filled up to end game as a tie in 2-player
mode
    public static boolean boardFull(char[][] board) {

        for (int i=0; i<3; i++) {
        for (int j=0; j<3; j++) {
                if (board[i][j] == '-') {
                return false;
                }
        }
        }
        return true;

    } //end boardFull()


    //---------------------------------------- METHODS FOR CPU MODE ----------
```

```
-----------------------------------


   //method for CPU mode
   public static void playComputer() {

       do {

       System.out.println("\nooooxxxoooxxxoooxxxoooxxxoooxxxooo\n");
       System.out.println("--- CPU MODE   (@_@) ---\nCan you defeat a non-living
thing?\n");
       System.out.println("The numbers representing each cell:");
       System.out.println("1" + "|" + "2"  + "|" + "3");
       System.out.println("-----");
       System.out.println("4" + "|" + "5"  + "|" + "6");
       System.out.println("-----");
       System.out.println("7" + "|" + "8"  + "|" + "9");
       System.out.println("\n************\n");

       tipsCount = 0;              //reset tipsCount

       char [] [] board = {{' ', ' ' , ' '},
                           {' ', ' ' , ' '},
                           {' ', ' ' , ' '}}; //create board

       printBoard(board); //print board

       while (true) {

              tips();
              playerMove(board);

              if (gameOver(board)){
              break;
              }
              printBoard(board);
              computerRandNo(board);

              if (gameOver(board)){
              break;
              }

              printBoard(board);
       } //end while

       System.out.println("Play again?");
       System.out.print("Enter 1 to continue playing or any number to exit to the
main menu: ");
       option = input.nextInt();

       //if option != 1, return to main menu; otherwise loop cuurent method (game
mode)
       if (option != 1) {
              System.out.println();
              return;
       }

       } while (option == 1);
       //end do-while for method loop

   }//end playComputer() or CPU mode
```

```
   //method to print board for CPU mode
   public static void printBoard(char [] [] board) {

       System.out.println(board [0][0] + "|" + board [0][1] + "|" + board [0][2] );
       System.out.println("-----");
       System.out.println(board [1][0] + "|" + board [1][1] + "|" + board [1][2] );
       System.out.println("----");
       System.out.println(board [2][0] + "|" + board [2][1] + "|" + board [2][2] +
"\n" );

   }// end printBoard();


   //method to check number input from user in CPU mode
   public static void playerMove(char [][] board) {

       String userInput;

       while (true) {
       System.out.print("X, make a move by entering a number (1~9): ");
       userInput = input.next();

       if (emptyCells(board, userInput)) {
              break;
       }
       else {
              System.out.println("************");
              System.out.println(userInput + " is not a valid move.");
              System.out.println("************");
       }

       } //end while

       System.out.println(userInput);

       switch (userInput) {
       case "1":
              board[0][0] = 'X';
              break;
       case "2":
              board[0][1] = 'X';
              break;
       case "3":
              board[0][2] = 'X';
              break;
       case "4":
              board[1][0] = 'X';
              break;
       case "5":
              board[1][1] = 'X';
              break;
       case "6":
              board[1][2] = 'X';
              break;
       case "7":
              board[2][0] = 'X';
              break;
       case "8":
              board[2][1] = 'X';
              break;
       case "9":
              board[2][2] = 'X';
              break;
```

```
            default:
            System.out.println();
        } //end switch

    } // end playerMove()


    //method to insert symbol (Xs or Os) into board in CPU mode
    public static void placeMove(char[][] board, String place,char letter) {

        switch (place) {
        case "1":
               board[0][0] = letter;
               break;
           case "2":
               board[0][1] = letter;
               break;
        case "3":
               board[0][2] = letter;
               break;
        case "4":
               board[1][0] = letter;
               break;
        case "5":
            board[1][1] = letter;
               break;
        case "6":
               board[1][2] = letter;
               break;
        case "7":
               board[2][0] = letter;
               break;
        case "8":
               board[2][1] = letter;
               break;
        case "9":
               board[2][2] = letter;
               break;
        default:
               System.out.println();
        }//end switch

    } //end placeMove()


    //method to create move for CPU
    public static void computerRandNo (char [][] board) {

        Random randomNumber = new Random(); //declare Random function
        int computerMove;

        while(true) {
        computerMove = randomNumber.nextInt(9) + 1; //obtain random number for
computer's move

        if (emptyCells(board,Integer.toString(computerMove))) {
               break;
        }
        } //end while

        System.out.println("Computer (O) choose to move at " + computerMove + "\n");
       placeMove(board, Integer.toString(computerMove), 'O');
```

```
   } //end computerRandNo();


   //method to display empty cells when necessary in CPU mode
   public static boolean emptyCells(char[][] board, String place) {

       switch (place) {
       case "1":
              return (board[0][0] == ' ');
       case "2":
              return (board[0][1] == ' ');
       case "3":
              return (board[0][2] == ' ');
       case "4":
              return (board[1][0] == ' ');
           case "5":
              return (board[1][1] == ' ');
       case "6":
              return (board[1][2] == ' ');
       case "7":
              return (board[2][0] == ' ');
       case "8":
              return (board[2][1] == ' ');
       case "9":
             return (board[2][2] == ' ');
       default:
              return false;
       } //end switch

   } //end emptyCells()


   //method for CPU mode to check if someone has won the game, otherwise game
continues
   public static boolean someoneWon (char[][] board, char letter) {

       if((board[0][0] == letter && board[0][1] == letter && board[0][2] == letter )
||
       (board[1][0] == letter && board[1][1] == letter && board[1][2] == letter )
||
       (board[2][0] == letter && board[2][1] == letter && board[2][2] == letter )
||

       (board[0][0] == letter && board[1][0] == letter && board[2][0] == letter )
||
       (board[0][1] == letter && board[1][1] == letter && board[2][1] == letter )
||
       (board[0][2] == letter && board[1][2] == letter && board[2][2] == letter )
||

       (board[0][0] == letter && board[1][1] == letter && board[2][2] == letter )
||
       (board[0][2] == letter && board[1][1] == letter && board[2][0] == letter ) )
       {
              return true;
       }

       return false;

   } //end someoneWon()


   //method to display end game for CPU mode
```

```
   public static boolean gameOver(char[][] board){

       if (someoneWon(board, 'X')) {
              printBoard(board);
              System.out.println("Congratulations! You are the winner!!!");
              return true;
       }

       if (someoneWon(board,'O')) {
              printBoard(board);
              System.out.println("Computer wins! You lose T.T");
              return true;
       }


       for (int i = 0;i<board.length; i++ ) {
       for (int j = 0; j < board[i].length; j++) {
              if (board[i][j] == ' ') {
              return false;
              }
       }
       }
       printBoard(board);
       System.out.println("The game ended in a draw.");
       return true;

   } //end gameOver()


   //------------------------------------------- TIPS -------------------------
-------------------


   //method to display tips when enabled by user
   public static void tips() {

       if (tipsCheck == 'Y') {
       tipsCount++;
       }

       switch (tipsCount) {

       case 1:
              System.out.println("Tip: Secure the corners first and you'll decrease
the chances of your opponent winning!\n");
              break;
       case 2:
              System.out.println("Tip: Take the center spot if your opponent took
the corners ;)\n");
              break;
       case 3:
              System.out.println("Tip: Always be aware of your opponent's move, one
slight miss and you'll lose easily.\n");
              break;
       case 4:
              System.out.println("Tip: Remember to check for diagonals too, your
opponent could be sneaky!\n");
              break;
       case 5:
              System.out.println("Tip: Do you think you could trap your opponent by
forcing them to mark on a specific spot?\n");
              break;
       case 6:
```

```
            System.out.println("Tip: If they are one spot away from winning and
you know it, take that spot!\n");
            break;
      case 7:
            System.out.println(":::: I can't think of any more tips for you at
this point in the game HAHA!\n");
            break;
      case 8:
            System.out.println(":::: U CAN DO THIS\n");
            break;
      case 9:
            System.out.println(":::: It's victory or defeat, or a tie??\n");
            break;
      case 10:
            System.out.println("      If you're reading this, I know you're
playing medium mode #.#");
            System.out.println("Tip: Try use a calcula- oh nevermind\n");
            break;
      case 11:
            System.out.println("Tip: You have a huge opportunity to block their
line if your opponent answered the wrong math question.\n");
            break;
      case 12:
            System.out.println(":::: No tip this time. Why? Cause I'm out of tips,
haha!\n");
            break;
      case 13:
            System.out.println(":::: ......\n");
            break;
      case 14:
            System.out.println(":::: *sweating*\n");
            break;
         case 15:
            System.out.println(":::: This is taking too long. I guess... you're
on your own now?");
            System.out.println("       KEEP GOING\n");
            break;
      default:
            ;

      } //end switch for tips display

   } //end tips()


   //------------------------------------------- RULES -------------------------
--------------------


   //method to view rules
   public static void rules() {

      System.out.println("\nooooxxxoooxxxoooxxxoooxxxoooxxxooo\n");
      System.out.println("--- RULES ---\n");
      System.out.println("The game is played on a 3x3 grid.");
      System.out.println("Assuming you are 'X', and your friend (or the computer)
is 'O'.");
      System.out.println("The first player to get 3 of thier marks in a row, either
HORIZONTALLY, VERTICALLY, or DIAGONALLY; is the winner.");
      System.out.println("However, when all 9 squares are full, the game is tied
and nobody wins.");
      System.out.println("It's that simple. but ARE YA READY?! ;)");
```

```
        System.out.println("\nReturn to main menu?");
        System.out.print("Enter any number: ");
        option = input.nextInt();
        System.out.println();
        return;

    } //end rules()


    //------------------------------------------ METHODS FOR LEADERBOARD -------
-------------------------------------


    //method to view leaderboard
    public static void leaderboard() {

        String names[] = {name1, name2, nameCPU};
        int scores[] = {score1, score2, 0};

        do {

        System.out.println("\nooooxxxoooxxxoooxxxoooxxxoooxxxooo\n");
        System.out.println("--- LEADERBOARD ---\n");
        System.out.println("Place\t\tName\t\t\tScore");

        if ((score1 == 0) && (score2 == 0)) {
                System.out.printf(1 + "\t\t\t%-10s%5d%n", names[2], scores[2]);
                System.out.printf(1 + "\t\t\t%-10s%5d%n", name1, score1);
                System.out.printf(1 + "\t\t\t%-10s%5d%n", name2, score2);
        }
        else if (score1 > score2) {
                System.out.printf(1 + "\t\t\t%-10s%5d%n", name1, score1);
                System.out.printf(2 + "\t\t\t%-10s%5d%n", name2, score2);
                System.out.printf(3 + "\t\t\t%-10s%5d%n", names[2], scores[2]);
        }
        else if (score2 > score1) {
                System.out.printf(1 + "\t\t\t%-10s%5d%n", name2, score2);
                System.out.printf(2 + "\t\t\t%-10s%5d%n", name1, score1);
                System.out.printf(3 + "\t\t\t%-10s%5d%n", names[2], scores[2]);
        }
        else {
                System.out.printf(1 + "\t\t\t%-10s%5d%n", name1, score1);
                System.out.printf(1 + "\t\t\t%-10s%5d%n", name2, score2);
                System.out.printf(3 + "\t\t\t%-10s%5d%n", names[2], scores[2]);
        }

        System.out.println("\n--------------");
        System.out.println("1\tSearch name in leaderboard");
        System.out.println("2\tReset leaderboard");
        System.out.println("3\tReturn to main menu");

        do {
                System.out.print("\nEnter a number: ");
                option = input.nextInt();

                if ((option != 1) && (option != 2) && (option != 3)) {
                  System.out.println("Invalid input! Enter 1, 2, or 3 only.");
                }

        } while ((option != 1) && (option != 2) && (option != 3));

        if (option == 1) {
                searchName(names);
```

```
        }
        else if (option == 2) {
                int num;

                System.out.println("\nAre you sure to reset the leaderboard?");
                System.out.print("Enter 1 to proceed or any number to cancel: ");
                num = input.nextInt();

                if (num == 1) {
                resetLeaderboard(names, scores);
                }
        }
        else {
                System.out.println();
                return;
        }
        } while ((option == 1) || (option == 2));

    } //end leaderboard()


    //method to search the name of the selected player
    public static void searchName(String[] names) {

        int flag = 0;
        int num;

        System.out.println("\n1\tPlayer 1");
        System.out.println("2\tPlayer 2");
       System.out.print("Enter the player number that you want to search: ");
        num = input.nextInt();

        for (int i=1; i <= names.length; i++) {
        if (i == num) {
                System.out.println("\nYou searched for Player " + i + ", their name
is " + names[i-1]);
                flag = 1;
                break;
        }
        }

        if (flag == 0) {
        System.out.println("\nInvalid number!");
        }

        System.out.println("\nReturn to leaderboard?");
        System.out.print("Enter any number to proceed: ");
        num = input.nextInt();

    } //end searchName()


    //method to reset the leaderboard
    public static void resetLeaderboard(String[] names, int[] scores) {

        for (int i=0; i < names.length; i++) {
        names[i] = "null";
        scores[i] = 0;
        }

    } //end resetLeaderboard()
```

```
   //------------------------------------------- CREDITS ----------------------
----------------------


   //method to display credits
   public static void credits() {

       System.out.println("\nooooxxxoooxxxoooxxxoooxxxoooxxxooo\n");
       System.out.println("Creators of this project:\n");
       System.out.println("[GROUP 5]");
       System.out.println("1\tLee Juan (280027)");
       System.out.println("2\tMuhammad Zuhair Afham bin Mohd Nasir (280782)");
       System.out.println("3\tEdwin Lim Wei Bin (281775)");
       System.out.println("\nMade for Dr. Noradila binti Nordin <3");
       System.out.println("A212 STIA1113(A) Programming 1");

       String justForFun = "heart, blood, and soul";
       System.out.printf("%nThough it's not perfect, we put our %s into this!!%n",
justForFun);

       System.out.println("\nReturn to main menu?");
       System.out.print("Enter any number: ");
       option = input.nextInt();
       System.out.println();
       return;

   } //end credits()

} //end class
```

4.5 Video Presentation

YouTube link to our video presentation:

https://youtu.be/6XxROfIrJI4

Embed of the video, this will direct you to the YouTube link as well: