5COSC019W - Tutorial 7 Exercises

1 A Tic-Tac-Toe Game

In this exercise we will develop a simple text version of the classic tic-tac-toe game.

The game is played with two players, **X** (user) and **O** (computer), who take turns marking the spaces in a 3×3 grid. The **X** player goes first. The player who succeeds in placing three respective marks in a horizontal, vertical, or diagonal row wins the game (see Figure 1 below).

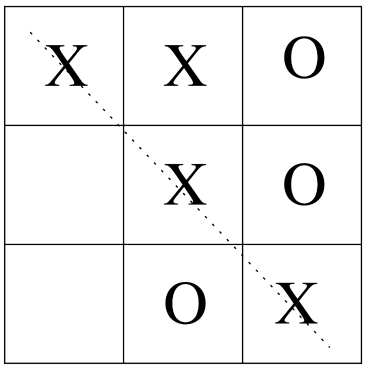

Figure 1: An instance of the tic-tac-toe board where the X player has won since he managed to place three 'X' in a diagonal.

1. The following classes implement the game. Create a new project in Netbeans (or your favourite IDE, or use the command line if you prefer) include the classes and run the code.

   File `Player.java`:

```java
abstract class Player {
    Board board;
    char symbol;

    // player plays with symbol 'X' or 'O'
    Player(Board b, char symbol1) {
        board = b;
        symbol = symbol1;
    }


    // takes (returns) an action based on the current state of the Board
    public abstract String act();
}
```

   File `Board.java`:

```java
class Board {
    String current_state = "---------";  // the board is empty initially
    boolean turnX = true;  // true if it is X's turn


    //converts a string description of a board to a char graphical view
    public String str2Board(String str) {
        String s = "| " + charOut(str.charAt(0)) + " | " +
                          charOut(str.charAt(1)) + " | " +
                          charOut(str.charAt(2)) + " |" +
                          "\n----|---|----\n" +
                          "| " + charOut(str.charAt(3)) + " | " +
                          charOut(str.charAt(4)) + " | " +
                          charOut(str.charAt(5)) + " |" +
                          "\n----|---|----\n" +
                          "| " + charOut(str.charAt(6)) + " | " +
                          charOut(str.charAt(7)) + " | " +
                          charOut(str.charAt(8)) + " |\n";

        return s;
    }


    @Override
    public String toString() {
        return str2Board(current_state);
    }


    // returns an blank space if s is "-" otherwise the char itself
    public char charOut(char s) {
        if (s == '-')
            return ' ';
        else
            return s;
    }


    // displays in char mode a board with keys indices for player's next move
    public void displayKeys() {
        String keys = "123456789";

        System.out.println(str2Board(keys));
        System.out.println("*****************");
    }


    /* translates a player's action to an index that the X
       will be placed on the board - returns -1 if not valid int */
```

```java
public int action2Index(String action) {
    int index = -1;

    try {
        index = Integer.valueOf(action) - 1;
    }
    catch (NumberFormatException ex) {
        // user entered a non integer
        System.out.println("Invalid move");
    }

    if (index < 0 || index > 8) {
        System.out.println("Invalid move");
        index  = -1;
    }
    return index;
}


// applies a (human/machine) player's move and moves to the next state
public boolean step(String action) {
    int index = action2Index(action);

    if (index != -1 && index < current_state.length() &&
                current_state.charAt(index) != 'X' &&
                current_state.charAt(index) != 'O') {
        if (turnX)
            current_state = current_state.substring(0, index) + 'X' +
                        current_state.substring(index+1, current_state.length());
        else
            current_state = current_state.substring(0, index) + 'O' +
                        current_state.substring(index+1, current_state.length());

        // switch to other player's move
        turnX = !turnX;

        return true;
    }
    else {
        System.out.println("Action chosen by Player is not valid!");
        return false;
    }
}


/* returns 'X' or 'O' if either is a winner or '-' otherwise
   if a state is passed then that state is checked */
public char checkWinner() {
    // check rows
```

```
int countX = 0;    // how many X
int countO = 0;    // how many O

for (int pos=0; pos <= 6; pos = pos+3) {
    for (int col=0; col < 3; col++) {
        if (current_state.charAt(pos+col) == 'X')
            ++countX;
        else if (current_state.charAt(pos+col) == 'O')
            ++countO;
    }

    if (countX == 3)
        return 'X';
    else if (countO == 3)
        return 'O';
    else {
        countX = 0;
        countO = 0;
    }
}

// check columns
countX = 0;
countO = 0;
for (int pos=0; pos < 3; ++pos) {
    for (int row=0; row <= 6; row = row+3) {
       if (current_state.charAt(row+pos) == 'X')
           ++countX;
       else if (current_state.charAt(row+pos) == 'O')
           ++countO;
    }

    if (countX == 3)
        return 'X';
    else if (countO == 3)
        return 'O';
    else {
        countX = 0;
        countO = 0;
    }
}

// check top-left bottom-right diagonal
countX = 0;
countO = 0;
for (int i=0; i < 9; i=i+4) {
    if (current_state.charAt(i) == 'X')
        ++countX;
    else if (current_state.charAt(i) == 'O')
```

```
                ++countO;
        }

        if (countX == 3)
            return 'X';
        else if (countO == 3)
            return 'O';

        // check top-right bottom-left diagonal
        countX = 0;
        countO = 0;
        for (int i=2; i < 8; i=i+2) {
            if (current_state.charAt(i) == 'X')
                ++countX;
            else if (current_state.charAt(i) == 'O')
                ++countO;
        }

        if (countX == 3)
            return 'X';
        else if (countO == 3)
            return 'O';

        // No winner!
        return '-';
    }


    // returns true if there are no moves left on the board
    public boolean isFull() {
        int empty = 0;

        for (int i=0; i < current_state.length(); i++)
            if (current_state.charAt(i) == '-')
                return false;

        System.out.println("*** The board is full! ***");
        return true;
    }

}
```

File `HumanPlayer.java`:

```
import java.util.*;

class HumanPlayer extends Player {
    HumanPlayer(Board board, char symbol) {
        super(board, symbol);
    }
    // takes an action based on the current state of the Board
    public String act() {
        System.out.println(board);  // display the current state of the board
        board.displayKeys();   // display choices for the user

        Scanner sc = new Scanner(System.in);
        System.out.print("Enter a move: ");
        String action = sc.next();

        return action;
    }
}
```

File RandomPlayer.java:

```
import java.util.*;

class RandomPlayer extends Player {
    RandomPlayer(Board board, char symbol) {
        super(board, symbol);
    }

    // takes an action based on the current state of the Board
    public String act() {
        System.out.println(board);  // display the current state of the board
        System.out.println("Machine is thinking...");
        // simulate thinking by delaying a bit
        try {
            Thread.sleep(500);
        }
        catch(InterruptedException ex) {
            ex.printStackTrace();
        }

        // find all available actions (empty slots)
        ArrayList<Integer> available = new ArrayList<>();
        for (int i=0; i < board.current_state.length(); i++)
            if (board.current_state.charAt(i) == '-')
                available.add(i);

        // choose a random action among the available empty slots
        Random gen = new Random();
        int index = gen.nextInt(available.size());
        System.out.println("*** Played position-> " +
                                (1+available.get(index)) + "\n");

        // numbering scheme starts at 1
        return String.valueOf(1+available.get(index));
    }
}
```

File TicTacToeGame.java:

```
class TicTacToeGame {
    Board board;
    Player human;
    Player computer;


    TicTacToeGame(Board board1, Player player1, Player actor1) {
        board = board1;
        human = player1;
        computer = actor1;
    }


    public void playGame() {
        boolean finished = false;
        char winner;

        while (!finished) {
            play_human_move();

            winner = board.checkWinner();
            if (winner == 'X' || winner == 'O' || board.isFull()) {
                System.out.println(" *** The winner is: " + winner + " ***");
                break;
            }

            play_machine_move();
            winner = board.checkWinner();
            if (winner == 'X' || winner == 'O' || board.isFull()) {
                System.out.println(" *** The winner is: " + winner + " ***");
                break;
            }
        }

        System.out.println(board);
    }


    public void play_human_move() {
        String action = human.act();
        boolean success = board.step(action);

        // check if the human's move was valid
        while (!success) {
            action = human.act();
            success = board.step(action);
        }
    }


    public void play_machine_move() {
        // choose a valid action
        String action = computer.act();

        // apply the action to the plant (game)
        board.step(action);
    }


    public static void main(String[] args) {
        Board board = new Board();
        Player human = new HumanPlayer(board, 'X');
        Player computer = new RandomPlayer(board, 'O');  // Replace with your player

        TicTacToeGame game = new TicTacToeGame(board, human, computer);
        game.playGame();
    }
}
```

2. Study the code and try to understand it. Your tutor will go through it after some time, but meanwhile you can ask any questions to your tutor.

2 Extending the Game with an Intelligent Player

1. Implement an intelligent version of the computer player (i.e. replacing the `RandomPlayer` class) by extending class `Player`. The intelligent computer player is able to defend itself, i.e. if the human player has already placed 2 `X` in a row, column or diagonal, the computer player will choose the free slot which will prevent the human to win in their next move.

2. Extend your intelligent player class so that it is able to choose winning positions, i.e. if the computer player has already placed 2 `O` is a row, column or diagonal then it places the next `O` in the slot which completes 3 `O` to win the game.

   If there is no winning move, but there is a defending move, the intelligent computer player will choose the defending one. If there is neither a winning or defending move, the intelligent player will choose a random valid move (i.e. a random slot among all empty slots).