5COSC019W - Solutions to Tutorial 7 Exercises

1 A Tic-Tac-Toe Game

Follow the instructions and make sure you ask any questions to your tutor.

2 Extending the Game with an Intelligent Player

The intelligent player (called `LogicalPlayer`) for both the defending and the winning moves can be found below.

You should **MODIFY** the line of your `main` which creates a `RandomPlayer` to create an object of the newly developed `LogicalPlayer` instead. Your `main` method should be:

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

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

The `LogicalPlayer.java` file:

```
import java.util.*;

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

    /* Returns a winning move or one which prevents defeat in the next move
       of the opponent - Otherwise it chooses a random move */
    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();
    }

    // check if a move will lead to winning or prevent defeat
    int index = get_winning_or_defend_index();
    if (index != -1) {  // win or defend action was found
        System.out.println("*** Played position-> " + (1+index) + "\n");
        return String.valueOf(1+index);
    }
    else {
        // 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();
        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));
    }
}


// Returns an index which wins or prevents a defeat or -1 otherwise
public int get_winning_or_defend_index() {
    String current_state = board.current_state;

    /* these indices contain the empty slot that if played will
       lead in winning or prevent the defeat, i.e. the row. column
       or diagonal that contain 2 Xs or 2 Os. */
    int winning_index = -1;
    int defend_index = -1;
    int empty_slot = -1;

    /* 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;
        else
            empty_slot = pos + col;
    }

    if (countX == 2 && empty_slot != -1) {
        if (symbol == 'X')  // if computer plays with 'X'
            winning_index = empty_slot;
        else
            defend_index = empty_slot;
    }
    else if (countO == 2 && empty_slot != -1) {
        if (symbol == 'O')  // if computer plays with 'O'
            winning_index = empty_slot;
        else
            defend_index = empty_slot;
    }

    countX = 0;
    countO = 0;
    empty_slot = -1;
}

/* check columns */
countX = 0;
countO = 0;
empty_slot = -1;

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;
       else
           empty_slot = row + pos;
    }

    if (countX == 2 && empty_slot != -1) {
        if (symbol == 'X')  // if computer plays with 'X'
            winning_index = empty_slot;
        else
            defend_index = empty_slot;
    }
    else if (countO == 2 && empty_slot != -1) {
        if (symbol == 'O')  // if computer plays with 'O'
            winning_index = empty_slot;
        else
            defend_index = empty_slot;
    }
    countX = 0;
    countO = 0;
    empty_slot = -1;
}

/* check top-left bottom-right diagonal */
countX = 0;
countO = 0;
empty_slot = -1;

for (int i=0; i < 9; i=i+4) {
    if (current_state.charAt(i) == 'X')
        ++countX;
    else if (current_state.charAt(i) == 'O')
        ++countO;
    else
        empty_slot = i;
}

if (countX == 2 && empty_slot != -1) {
    if (symbol == 'X')  // if computer plays with 'X'
        winning_index = empty_slot;
    else
        defend_index = empty_slot;
}
else if (countO == 2 && empty_slot != -1) {
    if (symbol == 'O')  // if computer plays with 'O'
        winning_index = empty_slot;
    else
        defend_index = empty_slot;
}

countX = 0;
countO = 0;
empty_slot = -1;

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

if (countX == 2 && empty_slot != -1) {
    if (symbol == 'X')  // if computer plays with 'X'
```

```
                winning_index = empty_slot;
            else
                defend_index = empty_slot;
        }
        else if (countO == 2 && empty_slot != -1) {
            if (symbol == 'O')  // if computer plays with 'O'
                winning_index = empty_slot;
            else
                defend_index = empty_slot;
        }

        /* return the winning index if any, otherwise a defend one */
        if (winning_index != -1)
            return winning_index;
        else if (defend_index != -1)
            return defend_index;
        else
            return -1;
    }
}
```