Data Structures and Algorithms

Assignment 2

Connect 4 game with AI

Submission document

**DUE DATE: Friday 5th April 2024**

**This Project is worth 10%**

**YOU MUST UPLOAD THIS COMPLETED DOCUMENT IN MICROSOFT WORD FORMAT ONLY.**

**Project Group Size: 4 to 5 people (Complete the area below with each team member’s details)**

|  |  |  |
| --- | --- | --- |
| **First Name: Anna**  **Last Name: Shibanova**  **Student ID: 101399925** | **First Name: Learwinn Ianjo**  **Last Name: Suaner**  **Student ID: 101394258** | **First Name: Jonathan**  **Last Name: Weir**  **Student ID: 101181715** |

|  |  |
| --- | --- |
| **First Name: Oleg**  **Last Name: Sanitskii**  **Student ID: 101466133** | **First Name:**  **Last Name:**  **Student ID:** |

**NOTE:**

**YOU MUST ONLY USE BASIC ARRAYS TO MODEL THE BOARD STATE – if this is not done, the maximum mark that can be attained is 5/10.**

**The non-AI part (2 human players) is worth 7 marks**

**The recursive minimax version (1 human vs AI) is worth 3 marks**

**This assignment loses 30% each day (based on the date) after the 5thApril.**

**This means if it is submitted 12:01 AM (1 minute late) on the 6th, the maximum mark that can be got is 7/10.**

**This means if it is submitted 12:01 AM on the 7th, the maximum mark that can be got is 4/10.**

**It is therefore better to submit a non-working AI version on time than an AI version late.**

## Submission Format

This is a group assignment, and one report and code will be submitted per group.

Your group must submit two Items:

1. This completed submission document with ALL the group member’s names inserted at the top and code pasted neatly into it.
2. The completed Java source files with all student’s names and numbers commented at the top of the code[not the whole project, just all the java classes you created]. **To be clear, do NOT upload a whole solution**.

Please note your code must be properly documented.

Code is pasted after this point in with neat classes.

:

// Anna Shibanova 101399925  
// Learwinn Ianjo Suaner 101394258  
// Jonathan Weir 101181715  
// Oleg Sanitskii 101466133  
  
import java.util.Arrays;  
import java.util.Scanner;  
  
public class Game {  
 private final int winningScore = 100;  
 private final int boardColumns = 7;  
 private final int boardRows = 6;  
 private String[] board;  
  
 private String playerName1;  
 private String playerName2;  
 private String symbol1;  
 private String symbol2;  
  
 private boolean isAIGame;  
  
 // Constructor to set up game  
 public Game(){  
 board = new String[boardColumns \* boardRows];  
 Arrays.*fill*(board, "O");  
  
 Scanner input = new Scanner(System.*in*);  
 System.*out*.println("Hello gamer! Is it a versus AI (1) or 2 player game (2)?");  
 String choiceGame = input.nextLine();  
 isAIGame = choiceGame.equals("1");  
  
 System.*out*.println("Hello gamer! Please, enter player 1 name: ");  
 playerName1 = input.nextLine();  
  
 System.*out*.println(playerName1 + ", choose symbol R (for Red) or Y (for Yellow): ");  
 boolean symbolEntered = false;  
 while (!symbolEntered) {  
 symbol1 = input.nextLine().toUpperCase();  
  
 if (!symbol1.equals("R") && !symbol1.equals("Y")) {  
 System.*out*.println("Only R (for Red) or Y (for Yellow) allowed! Again: ");  
 } else {  
 symbolEntered = true;  
 }  
 }  
  
 if (symbol1.equals("R")) {  
 symbol2 = "Y";  
 } else {  
 symbol2 = "R";  
 }  
  
 if (isAIGame) {  
 System.*out*.println("Do you want to go first (Y/N)?");  
 String choiceTurn = input.nextLine().toUpperCase();  
  
 if (choiceTurn.equals("Y")) {  
 playerName2 = "Computer";  
 } else {  
 // swap players according to turn order  
 playerName2 = playerName1;  
 playerName1 = "Computer";  
 String swap = symbol1;  
 symbol1 = symbol2;  
 symbol2 = swap;  
 }  
 } else {  
 System.*out*.println("Please, enter player 2 name: ");  
 playerName2 = input.nextLine();  
 System.*out*.println(playerName2 + " your symbol is " + symbol2);  
 }  
 }  
  
 // show board state  
 public void printBoard(String[] board) {  
 System.*out*.println("Game board:");  
 for (int i = 0; i < board.length; i++) {  
 System.*out*.print(" " + board[i] + " ");  
 if (i % 7 == 6) {  
 System.*out*.println("");  
 }  
 }  
 }  
  
 // function to check if the cell is empty  
 public boolean isEmpty(int position) {  
 return board[position].equals("O");  
 }  
  
 public void playGame() {  
 if (isAIGame) {  
 playAIGame();  
 } else {  
 playMultiplayer();  
 }  
 }  
  
 // Game between 2 players  
 public void playMultiplayer() {  
 while (true) {  
 if (playerAction(playerName1, symbol1)) {  
 System.*out*.println("Congratulations " + playerName1 + "! You win!");  
 break;  
 }  
 if (playerAction(playerName2, symbol2)) {  
 System.*out*.println("Congratulations " + playerName2 + "! You win!");  
 break;  
 }  
 }  
  
 }  
  
 // Game with AI  
 public void playAIGame(){  
 printBoard(board);  
 while (true) {  
 if (playerName1.equals("Computer")) {  
 if (computerAction(symbol1)) {  
 break;  
 }  
 if (playerAction(playerName2, symbol2)) {  
 System.*out*.println("Congratulations " + playerName2 + "! You win!");  
 break;  
 }  
 } else {  
 if (playerAction(playerName1, symbol1)) {  
 System.*out*.println("Congratulations " + playerName1 + "! You win!");  
 break;  
 }  
  
 if (computerAction(symbol2)) {  
 break;  
 }  
 }  
 }  
 }  
  
 // Logic for player moves  
 public boolean playerAction(String playerName, String symbol) {  
 Scanner input = new Scanner(System.*in*);  
  
 System.*out*.println(playerName + " pick a column");  
  
 int position = 0;  
 boolean isValid = false;  
 while(!isValid) {  
 try {  
 position = input.nextInt() - 1;  
  
 if (position < 0 || position > boardColumns) {  
 System.*out*.println("Must be a value from 1 to 7! Pick again ");  
 continue;  
 }  
  
 if (!dropDisk(board, symbol, position)) {  
 System.*out*.println("This column is already full!");  
 } else {  
 isValid = true;  
 }  
  
 } catch (Exception e) {  
 input.nextLine();  
 System.*out*.println("Must be an integer value ");  
 }  
 }  
 printBoard(board);  
 return checkWinner(board);  
 }  
  
 public boolean dropDisk(String[] board, String symbol, int position) {  
 boolean isEmpty = isEmpty(position);  
 if (!isEmpty) {  
 return false;  
 }  
  
 while (isEmpty && position + boardColumns < board.length) {  
 int newPosition = position + boardColumns;  
 isEmpty = isEmpty(newPosition);  
  
 if (isEmpty) {  
 position = newPosition;  
 }  
 }  
  
 board[position] = symbol;  
 return true;  
 }  
  
 public boolean checkWinner(String[] board){  
 // horizontal  
 for (int row = 0; row < boardRows; row++) {  
 for (int column = 0; column < boardColumns - 3; column++) {  
 int cell = row \* boardColumns + column;  
 if (checkLine(board, cell, 1)) {  
 return true;  
 }  
 }  
 }  
  
 // vertical  
 for (int row = 0; row < boardRows - 3; row++) {  
 for (int column = 0; column < boardColumns; column++) {  
 int cell = row \* boardColumns + column;  
 if (checkLine(board, cell, boardColumns)) {  
 return true;  
 }  
 }  
 }  
  
 // diagonal ascending  
 for (int row = 0; row < boardRows - 3; row++) {  
 for (int column = 3; column < boardColumns; column++) {  
 int cell = row \* boardColumns + column;  
 if (checkLine(board, cell, boardColumns - 1)) {  
 return true;  
 }  
 }  
 }  
  
 // diagonal descending  
 for (int row = 0; row < boardRows - 3; row++) {  
 for (int column = 0; column < boardColumns - 3; column++) {  
 int cell = row \* boardColumns + column;  
 if (checkLine(board, cell, boardColumns + 1)) {  
 return true;  
 }  
 }  
 }  
 return false;  
 }  
  
 public boolean checkLine(String[] board, int cell, int change){  
 String symbol = board[cell];  
 if (symbol.equals("O")) {  
 return false;  
 }  
  
 return checkLine(board, cell+change, change, symbol, 1);  
 }  
  
 public boolean checkLine(String[] board, int cell, int change, String symbol, int count){  
 if (!symbol.equals(board[cell])) {  
 return false;  
 }  
  
 count++;  
 if (count == 4) {  
 return true;  
 }  
 return checkLine(board,cell+change, change, symbol, count);  
 }  
  
 // Logic for AI moves  
 public boolean computerAction(String symbol) {  
 int bestMove = -1;  
 int bestScore = -100;  
 for (int i = 0; i < boardColumns; i++ ) {  
 if (!isEmpty(i)) {  
 continue;  
 }  
  
 int score = max(board, symbol, i, 7); // use MINMAX to find score  
  
 if (score > bestScore) {  
 bestMove = i;  
 bestScore = score;  
 }  
 }  
  
 if (bestMove == -1) {  
 System.*out*.println("We ran out of valid moves. Goodbye.");  
 return true;  
 }  
 if (dropDisk(board, symbol, bestMove)) {  
 printBoard(board);  
 if (checkWinner(board)) {  
 System.*out*.println("Computer wins today! Try again!");  
 return true;  
 }  
 }  
 return false;  
 }  
  
 // Find best score for AI player move  
 public int max(String[] board, String symbol, int column, int depth) {  
 String[] newBoard = board.clone();  
 boolean validMove = dropDisk(newBoard, symbol, column);  
 int score = scoreBoard(board, symbol);  
  
 if (validMove && depth > 0) {  
 int maxValue = -100;  
 for (int i = 0; i < boardColumns; i++) {  
 int otherScore = min(newBoard, symbol, i, depth - 1);  
 if (score > maxValue) {  
 maxValue = otherScore;  
 }  
 }  
 return maxValue;  
 }  
 return score;  
 }  
  
 // Find worse score for human opponent move  
 public int min(String[] board, String symbol, int column, int depth) {  
 String[] newBoard = board.clone();  
 String oppositeSymbol;  
 if (symbol.equals("R")) {  
 oppositeSymbol = "Y";  
 } else {  
 oppositeSymbol = "R";  
 }  
  
 boolean validMove = dropDisk(newBoard, oppositeSymbol, column);  
 int score = scoreBoard(board, symbol);  
  
 if (validMove && depth > 0) {  
 int maxValue = -100;  
 for (int i = 0; i < boardColumns; i++) {  
 int otherScore = max(newBoard, symbol, i, depth - 1);  
 if (score > maxValue) {  
 maxValue = otherScore;  
 }  
 }  
 return maxValue;  
 }  
 return score;  
 }  
  
 // Calculate value of current position in game  
 public int scoreBoard(String[] board, String playerSymbol){  
 int totalScore = 0;  
 // horizontal  
 for (int row = 0; row < boardRows; row++) {  
 for (int column = 0; column < boardColumns - 3; column++) {  
 int cell = row \* boardColumns + column;  
 int score = scoreLine(board, cell, 1);  
 if (!board[cell].equals(playerSymbol)) {  
 score \*= -1;  
 }  
 totalScore += score;  
 }  
 }  
  
 // vertical  
 for (int row = 0; row < boardRows - 3; row++) {  
 for (int column = 0; column < boardColumns; column++) {  
 int cell = row \* boardColumns + column;  
 int score = scoreLine(board, cell, boardColumns);  
 if (!board[cell].equals(playerSymbol)) {  
 score \*= -1;  
 }  
 totalScore += score;  
 }  
 }  
  
 // diagonal ascending  
 for (int row = 0; row < boardRows - 3; row++) {  
 for (int column = 3; column < boardColumns; column++) {  
 int cell = row \* boardColumns + column;  
 int score = scoreLine(board, cell, boardColumns - 1);  
 if (!board[cell].equals(playerSymbol)) {  
 score \*= -1;  
 }  
 totalScore += score;  
 }  
 }  
  
 // diagonal descending  
 for (int row = 0; row < boardRows - 3; row++) {  
 for (int column = 0; column < boardColumns - 3; column++) {  
 int cell = row \* boardColumns + column;  
 int score = scoreLine(board, cell, boardColumns + 1);  
 if (!board[cell].equals(playerSymbol)) {  
 score \*= -1;  
 }  
 totalScore += score;  
 }  
 }  
 return totalScore;  
 }  
  
  
 public int scoreLine(String[] board, int cell, int change){  
 String symbol = board[cell];  
 if (symbol.equals("O")) {  
 return 0;  
 }  
  
 return scoreLine(board, cell+change, change, symbol, 1, 1);  
 }  
  
 // Use recursion to calculate value of a line  
 public int scoreLine(String[] board, int cell, int change, String symbol, int count, int score) {  
 if (isEmpty(cell)) {  
 return score;  
 }  
 if (!symbol.equals(board[cell])) {  
 return 0;  
 }  
  
 count++;  
 if (count == 4) {  
 return winningScore;  
 }  
 return scoreLine(board,cell+change, change, symbol, count, score \* 2);  
 }  
}