Data Structures and Algorithms

Gomoku (Five in a Row) Game with AI

Submission document

**DUE DATE: Friday 4th APRIL 2025**

**This Project is worth 15%**

**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:** Henrique  **Last Name:** Custodio  **Student ID:** 101497015 | **First Name:** Jinah  **Last Name:** Ahn  **Student ID:** 100902591 | **First Name:** Fitsum  **Last Name:** Asgedom  **Student ID:** 101510623 |
| --- | --- | --- |

| **First Name:**  **Last Name:**  **Student ID:** | **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 4thApril.**

**This means if it is submitted 12:01 AM (1 minute late) on the 5th, the maximum mark that can be got is 7/10.**

**This means if it is submitted 12:01 AM on the 6th, 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 neat classes.

:

# Main.java

import java.util.Scanner;

public class Main

{

public static void main(String[] args)

{

Scanner scanner = new Scanner(System.in);

System.out.println("------------------------------");

System.out.println("Welcome to Console 9x9 Gomoku!");

System.out.println("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_");

// Selecting game mode menu

boolean running = true;

while (running)

{

System.out.println("Select Game Mode: ");

System.out.println("1. 1 Player (Human vs AI)");

System.out.println("2. 2 Player (Human vs Human)");

System.out.println("3. Exit Game");

System.out.println("Enter your choice: ");

int userChoice = 0;

try

{

userChoice = Integer.parseInt(scanner.nextLine());

}

catch (NumberFormatException e)

{

System.out.println("You did not enter in a number (1-3).");

}

switch (userChoice)

{

case 1:

System.out.print("Enter player name: ");

String playerName = scanner.nextLine();

char playerSymbol = ' ';

// Choosing symbols menu

while (playerSymbol != 'B' && playerSymbol != 'W')

{

System.out.print("Enter player symbol (B for Black and W for White): ");

playerSymbol = scanner.next().toUpperCase().charAt(0);

scanner.nextLine();

}

char aiSymbol;

if (playerSymbol == 'B')

{

aiSymbol = 'W';

}

else

{

aiSymbol = 'B';

}

GameLogic gameBoard = new GameLogic(playerName, playerSymbol, aiSymbol);

gameBoard.startSinglePlayer();

break;

case 2:

System.out.print("Enter player 1's name: ");

String player1Name= scanner.nextLine();

System.out.print("Enter player 2's name: ");

String player2Name= scanner.nextLine();

char player1Symbol = ' ';

char player2Symbol = ' ';

while (player1Symbol != 'B' && player1Symbol != 'W')

{

System.out.print("Enter Player 1's symbol. Player 2 will be the other color (B for Black and W for White): ");

player1Symbol = scanner.next().toUpperCase().charAt(0);

scanner.nextLine();

}

if (player1Symbol == 'B')

{

player2Symbol = 'W';

}

else

{

player2Symbol = 'B';

}

GameLogic gameBoard2 = new GameLogic(player1Name, player1Symbol, player2Name, player2Symbol);

gameBoard2.startTwoPlayer();

break;

case 3:

System.out.println("Thank you for playing!");

running = false;

break;

default:

System.out.println("You did not enter a valid option.");

break;

}

}

scanner.close();

}

}

# MinimaxAlgo.java

import java.util.Random;

public class MinimaxAlgo

{

private final GameBoard gameBoard = new GameBoard();

private final int DEPTH = 3;

public int minimax(char[][] board, int depth, boolean max, char aiSymbol, char playerSymbol, int alpha, int beta)

{

if (winningMove(board, aiSymbol))

{

return Integer.MAX\_VALUE - depth;

}

if (winningMove(board, playerSymbol))

{

return Integer.MIN\_VALUE + depth;

}

if (depth == 0 || gameBoard.isBoardFull(board))

{

return evaluateBoard(board, aiSymbol, playerSymbol);

}

if (max)

{

int maxEval = Integer.MIN\_VALUE;

for (int i = 0; i < board.length; i++)

{

for (int j = 0; j < board[0].length; j++)

{

if (board[i][j] == '.')

{

board[i][j] = aiSymbol;

int eval = minimax(board, depth - 1, false, aiSymbol, playerSymbol, alpha, beta);

board[i][j] = '.';

maxEval = Math.max(maxEval, eval);

alpha = Math.max(alpha, eval);

if (beta <= alpha)

{

break;

}

}

}

}

return maxEval;

}

else

{

int minEval = Integer.MAX\_VALUE;

for (int i = 0; i < board.length; i++)

{

for (int j = 0; j < board[0].length; j++)

{

if (board[i][j] == '.')

{

board[i][j] = playerSymbol;

int eval = minimax(board, depth - 1, true, aiSymbol, playerSymbol, alpha, beta);

board[i][j] = '.';

minEval = Math.min(minEval, eval);

beta = Math.min(beta, eval);

if (beta <= alpha)

{

break;

}

}

}

}

return minEval;

}

}

public int[] findBestMove(char[][] board, int depth, boolean isMax, char aiSymbol, char playerSymbol)

{

// Check for any immediate winning moves

for (int i = 0; i < board.length; i++)

{

for (int j = 0; j < board[0].length; j++)

{

if (board[i][j] == '.')

{

board[i][j] = aiSymbol;

if (gameBoard.checkWin(board, i, j, aiSymbol))

{

board[i][j] = '.';

return new int[]{i, j};

}

board[i][j] = '.';

}

}

}

// Check for player winning move and block it

for (int i = 0; i < board.length; i++)

{

for (int j = 0; j < board[0].length; j++)

{

if (board[i][j] == '.')

{

board[i][j] = playerSymbol;

if (gameBoard.checkWin(board, i, j, playerSymbol))

{

board[i][j] = '.';

return new int[]{i, j};

}

board[i][j] = '.';

}

}

}

// Minimax to find best scoring move

int bestScore = Integer.MIN\_VALUE;

int[][] bestMove = new int[81][2];

int count = 0;

for(int row = 0; row < board.length; row++)

{

for(int col = 0; col < board[0].length; col++)

{

if(board[row][col] == '.')

{

board[row][col] = aiSymbol;

int tempScore = minimax(board, DEPTH, false, aiSymbol, playerSymbol, Integer.MIN\_VALUE, Integer.MAX\_VALUE);

board[row][col] = '.';

if (tempScore > bestScore)

{

bestScore = tempScore;

bestMove[0][0] = row;

bestMove[0][1] = col;

count = 1;

}

else if (tempScore == bestScore)

{

bestMove[count][0] = row;

bestMove[count][1] = col;

count++;

}

}

}

}

int randomIndex = new Random().nextInt(count);

return new int[]{bestMove[randomIndex][0], bestMove[randomIndex][1]};

}

private boolean winningMove(char[][] board, char symbol)

{

for (int row = 0; row < board.length; row++)

{

for (int col = 0; col < board[0].length; col++)

{

if (board[row][col] == symbol && gameBoard.checkWin(board, row, col, symbol))

{

return true;

}

}

}

return false;

}

private int evaluateBoard(char[][] board, char aiSymbol, char playerSymbol)

{

int score = 0;

for (int row = 0; row < board.length; row++)

{

for (int col = 0; col < board[0].length; col++)

{

if (board[row][col] != '.')

{

char currentSymbol = board[row][col];

int value = evaluatePosition(board, row, col, currentSymbol);

if (currentSymbol == aiSymbol)

{

// Favour the AI's move more heavily

score += value \* 2;

}

else if (currentSymbol == playerSymbol)

{

score -= value;

}

}

}

}

return score;

}

private int evaluatePosition(char[][] board, int row, int col, char symbol)

{

int score = 0;

int[][] directions = {{0, 1}, {1, 0}, {1, 1}, {1, -1}};

for (int[] direction : directions)

{

int count = 1;

int openEnds = 0;

int r = row + direction[0];

int c = col + direction[1];

while (inBounds(board, r, c) && board[r][c] == symbol)

{

count++;

r = r + direction[0];

c = c + direction[1];

}

if (inBounds(board, r, c) && board[r][c] == '.')

{

openEnds++;

}

r = row - direction[0];

c = col - direction[1];

while (inBounds(board, r, c) && board[r][c] == symbol)

{

count++;

r = r - direction[0];

c = c - direction[1];

}

if (inBounds(board, r, c) && board[r][c] == '.')

{

openEnds++;

}

score += scorePositions(count, openEnds);

}

return score;

}

private int scorePositions(int count, int openEnds)

{

if (count >= 5)

return 10000;

else if (count == 4 && openEnds == 2)

return 500;

else if (count == 4 && openEnds == 1)

return 200;

else if (count == 3 && openEnds == 2)

return 100;

else if (count == 3 && openEnds == 1)

return 50;

else if (count == 2 && openEnds == 2)

return 25;

else if (count == 2 && openEnds == 1)

return 5;

else

return 0;

}

private boolean inBounds(char[][] board, int row, int col)

{

return row >= 0 && row < board.length && col >= 0 && col < board[0].length;

}

}

# GameLogic.java

import java.util.Scanner;

public class GameLogic

{

private final int BOARD\_WIDTH = 9;

private final int BOARD\_HEIGHT = 9;

private char[][] gameBoard;

private boolean isSinglePlayer;

private String[] playerNames = new String[2];

private char[] playerSymbols = new char[2];

private int[] playerScores = new int[2];

private GomokuAI ai;

private GameBoard gameBoardRender;

private Scanner scanner;

public GameLogic(String name, char symbol, char aiSymbol)

{

this.isSinglePlayer = true;

playerNames[0] = name;

playerSymbols[0] = symbol;

playerScores[0] = 0;

playerNames[1] = "Computer";

playerSymbols[1] = aiSymbol;

playerScores[1] = 0;

this.ai = new GomokuAI(aiSymbol, symbol);

this.scanner = new Scanner(System.in);

this.gameBoardRender = new GameBoard();

this.gameBoard = new char[BOARD\_HEIGHT][BOARD\_WIDTH];

gameBoardRender.initBoard(gameBoard);

}

public GameLogic(String name1, char symbol1, String name2, char symbol2)

{

this.isSinglePlayer = false;

playerNames[0] = name1;

playerNames[1] = name2;

playerSymbols[0] = symbol1;

playerSymbols[1] = symbol2;

playerScores[0] = 0;

playerScores[1] = 0;

this.scanner = new Scanner(System.in);

this.gameBoardRender = new GameBoard();

this.gameBoard = new char[BOARD\_HEIGHT][BOARD\_WIDTH];

gameBoardRender.initBoard(gameBoard);

}

public void startTwoPlayer()

{

boolean player1Turn = (playerSymbols[0] == 'B');

while (true)

{

gameBoardRender.drawBoard(gameBoard);

if (player1Turn)

{

System.out.println(playerNames[0] + "'s turn (" + playerSymbols[0] + ")");

int[] move = makeValidMove(playerSymbols[0]);

if (gameBoardRender.checkWin(gameBoard, move[0], move[1], playerSymbols[0]))

{

gameBoardRender.drawBoard(gameBoard);

System.out.println("Player 1 Wins!");

break;

}

}

else

{

System.out.println(playerNames[1] + "'s turn (" + playerSymbols[1] + ")");

int[] move = makeValidMove(playerSymbols[1]);

if (gameBoardRender.checkWin(gameBoard, move[0], move[1], playerSymbols[1]))

{

gameBoardRender.drawBoard(gameBoard);

System.out.println("Player 2 Wins!");

break;

}

}

if (gameBoardRender.isBoardFull(gameBoard))

{

gameBoardRender.drawBoard(gameBoard);

System.out.println("Its a draw!");

break;

}

player1Turn = !player1Turn;

}

}

public void startSinglePlayer()

{

boolean player1Turn = (playerSymbols[0] == 'B');

while (true)

{

gameBoardRender.drawBoard(gameBoard);

if (player1Turn)

{

System.out.println(playerNames[0] + "'s turn (" + playerSymbols[0] + ")");

int[] move = makeValidMove(playerSymbols[0]);

if (gameBoardRender.checkWin(gameBoard, move[0], move[1], playerSymbols[0])) {

gameBoardRender.drawBoard(gameBoard);

System.out.println("Player 1 Wins!");

break;

}

}

else

{

System.out.println("Computers turn (" + playerSymbols[1] + ")");

System.out.println("Please wait for Computers Turn.");

int[] move = ai.getBestNextMove(gameBoard);

gameBoard[move[0]][move[1]] = playerSymbols[1];

if (gameBoardRender.checkWin(gameBoard, move[0], move[1], playerSymbols[1]))

{

gameBoardRender.drawBoard(gameBoard);

System.out.println("Player 2 Wins!");

break;

}

}

if (gameBoardRender.isBoardFull(gameBoard))

{

gameBoardRender.drawBoard(gameBoard);

System.out.println("Its a draw!");

break;

}

player1Turn = !player1Turn;

}

}

private int[] makeValidMove(char symbol)

{

int row, col;

while (true)

{

try

{

System.out.print("Enter row (0-8): ");

row = Integer.parseInt(scanner.nextLine());

System.out.print("Enter col (0-8): ");

col = Integer.parseInt(scanner.nextLine());

if (row >= 0 && row < 9 && col >= 0 && col < 9 && gameBoard[row][col] == '.')

{

gameBoard[row][col] = symbol;

return new int[]{row, col};

}

else

{

System.out.println("Invalid Move. Try again.");

}

}

catch (NumberFormatException e)

{

System.out.println("Please enter valid integers");

}

}

}

public char[][] getGameBoardCopy()

{

char[][] copy = new char[BOARD\_HEIGHT][BOARD\_WIDTH];

for (int i = 0; i < BOARD\_HEIGHT; i++)

{

System.arraycopy(gameBoard[i], 0, copy[i], 0, BOARD\_WIDTH);

}

return copy;

}

}

# GameBoard.java

public class GameBoard

{

private final int BOARD\_WIDTH = 9;

private final int BOARD\_HEIGHT = 9;

public void initBoard(char[][] gameBoard)

{

for (int i = 0; i < BOARD\_HEIGHT; i++)

{

for (int j = 0; j < BOARD\_WIDTH; j++)

{

gameBoard[i][j] = '.';

}

}

}

public void drawBoard(char[][] gameBoard)

{

System.out.print(" ");

for (int i = 0; i < BOARD\_HEIGHT; i++)

{

System.out.print(i + " ");

}

System.out.println("\n -----------------------");

for (int i = 0; i < BOARD\_HEIGHT; i++)

{

System.out.print(i + " | ");

for (int j = 0; j < BOARD\_WIDTH; j++)

{

System.out.print(gameBoard[i][j] + " ");

}

System.out.println();

}

}

public boolean isBoardFull(char[][] gameBoard)

{

for (int i = 0; i < BOARD\_HEIGHT; i++)

{

for (int j = 0; j < BOARD\_WIDTH; j++)

{

if (gameBoard[i][j] == '.')

{

return false;

}

}

}

return true;

}

public boolean checkWin(char[][] gameBoard, int row, int col, char symbol)

{

return checkDirection(gameBoard, row, col, 0, 1, symbol)

|| checkDirection(gameBoard, row, col, 1, 0, symbol)

|| checkDirection(gameBoard, row, col, 1, 1, symbol)

|| checkDirection(gameBoard, row, col, -1, 1, symbol);

}

private boolean checkDirection(char[][] gameBoard, int row, int col, int dRow, int dCol, char symbol)

{

int count = 1;

int r = row + dRow, c = col + dCol;

while (inBounds(r, c) && gameBoard[r][c] == symbol)

{

count++;

r += dRow;

c += dCol;

}

r = row - dRow;

c = col - dCol;

while (inBounds(r, c) && gameBoard[r][c] == symbol)

{

count++;

r -= dRow;

c -= dCol;

}

return count >= 5;

}

private boolean inBounds(int row, int col)

{

return row >= 0 && row < BOARD\_HEIGHT && col >= 0 && col < BOARD\_WIDTH;

}

}

# 

# GomokuAI.java

/\*

This class acts as an interface between the Game Logic and Minimax algo.

Takes both symbols and uses algo to determine the best move.

\*/

public class GomokuAI

{

private final MinimaxAlgo minimaxAlgo;

private final char aiSymbol;

private final char playerSymbol;

public GomokuAI(char aiSymbol, char playerSymbol)

{

this.minimaxAlgo = new MinimaxAlgo();

this.aiSymbol = aiSymbol;

this.playerSymbol = playerSymbol;

}

// Param here is current state of the game board, and returns an array with best move [row, col]

public int[] getBestNextMove(char[][] board)

{

return minimaxAlgo.findBestMove(board, 3, true, aiSymbol, playerSymbol);

}

}