

|  |
| --- |
| ICS 202  Data Structures  2018 |
| Lab Project |
| Mustafa Alsadah  201913690 |

Source Code:

Code for GameNode class:

public class GameNode {

//static int currentNumberOfPieces;

int currentNumberOfPieces;

int initialNumberOfPieces;

int level;

int minMaxValue;

String strRepresentation = "";

GameNode oneMoveChildren;

GameNode twoMovesChildren;

public GameNode(int currentNumberOfPeices){

this.currentNumberOfPieces = currentNumberOfPeices;

this.initialNumberOfPieces = currentNumberOfPieces;

for (int i = 0; i<this.currentNumberOfPieces; i++){

strRepresentation+="● ";

}

}

public void setCurrentNumberOfPieces(int currentNumberOfPieces) {

//GameNode.currentNumberOfPieces = currentNumberOfPieces;

}

public int getCurrentNumberOfPieces() {

return currentNumberOfPieces;

}

public String toString(){

return this.strRepresentation;

}

public boolean isWinning(){

return currentNumberOfPieces<=0?true:false;

}

public boolean isTerminalNode(){

return isWinning()?true:false;

}

}

Code for GameTree class:

import java.util.Scanner;

public class GameTree {

String treeStatistics;

GameNode root;

static int numberOfNodes = 0;

public int getNumberOfNodes() {

return numberOfNodes;

}

public GameTree(int numberOfPices){

root = new GameNode(numberOfPices);

root.level = 0;

root.minMaxValue = 0;

numberOfNodes++;

}

public void generateNextMoves(GameNode currentMoveNode){

if (!currentMoveNode.isTerminalNode()){

//currentMoveNode.currentNumberOfPieces-=1;

currentMoveNode.oneMoveChildren= new GameNode(currentMoveNode.currentNumberOfPieces-1);

currentMoveNode.oneMoveChildren.level = currentMoveNode.level+1;

numberOfNodes++;

generateNextMoves(currentMoveNode.oneMoveChildren);

///////////////////////////////////////////

currentMoveNode.twoMovesChildren = new GameNode(currentMoveNode.currentNumberOfPieces-2);

currentMoveNode.twoMovesChildren.level = currentMoveNode.level+1;

numberOfNodes++;

generateNextMoves(currentMoveNode.twoMovesChildren);

}

}

public void assignMinMaxValues(boolean playerIsGoingFirst){

assignMinMaxUsingDFS(root, playerIsGoingFirst);

}

private int assignMinMaxUsingDFS(GameNode node, boolean playerIsGoingFirst){

if (node == null){

System.out.println("UNREACHABLE");

return 0;

}else if (node.isWinning()){

if (playerIsGoingFirst)

//MAX ==> Player

//MIN ==> CPU

node.minMaxValue = node.level%2==0?-1:1;

else

//MAX ==> CPU

//MIN ==> Player

node.minMaxValue = node.level%2==0?1:-1;

return node.minMaxValue;

}else { //Game is still carrying on

int index = 0;

int[] minMaxValues = new int[2];

if (node.oneMoveChildren!=null) minMaxValues[index++] = assignMinMaxUsingDFS(node.oneMoveChildren, playerIsGoingFirst);

if (node.twoMovesChildren!=null) minMaxValues[index++] = assignMinMaxUsingDFS(node.twoMovesChildren, playerIsGoingFirst);

int overAllMin = minMaxValues[0];

int overAllMax = minMaxValues[0];

for (int i = 1; i < index; i++) {

if (overAllMin > minMaxValues[i]) overAllMin = minMaxValues[i];

if (overAllMax < minMaxValues[i]) overAllMax = minMaxValues[i];

}

if (node.level%2==0){

node.minMaxValue = overAllMax;

return overAllMax;

}else {

node.minMaxValue = overAllMin;

return overAllMin;

}

}

}

public void CPUTurn(GameNode currentNode, boolean isPerfectStrategy){

//Runs on Perfect strategy (deterministic with minimax, remove 2 if no move is optimal)

if (currentNode!=null){

if (currentNode.oneMoveChildren.minMaxValue==-1){

if (currentNode.oneMoveChildren.strRepresentation==""){

System.out.println("CPU has played and took 1.");

System.out.println("Hardluck, CPU won!");

System.exit(1);

}

System.out.println("CPU has played and took 1.");

playerTurn(currentNode.oneMoveChildren, isPerfectStrategy);

}

else if (currentNode.twoMovesChildren.minMaxValue==-1){

if (currentNode.twoMovesChildren.strRepresentation==""){

System.out.println("CPU has played and took 2.");

System.out.println("Hardluck, CPU won!");

System.exit(1);

}

System.out.println("CPU has played and took 2.");

playerTurn(currentNode.twoMovesChildren, isPerfectStrategy);

}else {

//No optimal move case

if (currentNode.twoMovesChildren.strRepresentation==""){

System.out.println("CPU has played and took 2.");

System.out.println("Hardluck, CPU won!");

System.exit(1);

}

System.out.println("CPU has played and took 2.");

playerTurn(currentNode.twoMovesChildren, isPerfectStrategy);

}

}

}

public void CPUTurnRandom(GameNode currentNode, boolean isPerfectStrategy){

//Runs on Random Strategy

if (currentNode!=null){

int rand = (int)(1+Math.random()\*2);

GameNode nextMoveNode = rand==1?currentNode.oneMoveChildren:currentNode.twoMovesChildren;

if (rand==1){

if (currentNode.oneMoveChildren.strRepresentation==""){

System.out.println("CPU has played and took 1.");

System.out.println("Hardluck, CPU won!");

System.exit(1);

}

System.out.println("CPU has played and took 1.");

playerTurn(currentNode.oneMoveChildren, isPerfectStrategy);

}

else{

if (currentNode.twoMovesChildren.strRepresentation==""){

System.out.println("CPU has played and took 2.");

System.out.println("Hardluck, CPU won!");

System.exit(1);

}

System.out.println("CPU has played and took 2.");

playerTurn(currentNode.twoMovesChildren, isPerfectStrategy);

}

}

}

public void playerTurn(GameNode currentNode, boolean isPerfectStrategy){

if (currentNode!=null){

Scanner sc = new Scanner(System.in);

String hint;

System.out.println("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n> Current state: "+ currentNode);

if (currentNode.oneMoveChildren.minMaxValue==1){

hint = "HINT: Take out 1";

}

else if (currentNode.twoMovesChildren.minMaxValue==1){

hint = "HINT: Take out 2";

}else {

hint = "There's no optimal move at this turn";

}

int choice;

do {

System.out.print("1.Take 1 || 2.Take 2 || 3.Show hint || 4.Options || enter:");

choice = sc.nextInt();

switch (choice){

case 1:

if (currentNode.oneMoveChildren.strRepresentation==""){

System.out.println("CONGRATS!, player won!");

System.exit(1);

}

System.out.println("You took one, current state: "+currentNode.oneMoveChildren);

if (isPerfectStrategy)

CPUTurn(currentNode.oneMoveChildren, isPerfectStrategy);

else

CPUTurnRandom(currentNode.oneMoveChildren, isPerfectStrategy);

break;

case 2:

if (currentNode.twoMovesChildren.strRepresentation==""){

System.out.println("CONGRATS!, player won!");

System.exit(1);

}

System.out.println("You took two, current state: "+currentNode.twoMovesChildren);

if (isPerfectStrategy)

CPUTurn(currentNode.twoMovesChildren, isPerfectStrategy);

else

CPUTurnRandom(currentNode.twoMovesChildren, isPerfectStrategy);

break;

case 3:

System.out.println(hint);

break;

case 4:

System.out.println("0.Exit || 1.Show tree statistics");

Scanner inputter = new Scanner(System.in);

int optionsListChoice = inputter.nextInt();

if (optionsListChoice==0)

System.exit(-1);

else if (optionsListChoice==1) {

treeStatistics = "Number of nodes = " + this.getNumberOfNodes() + " || Height = " + getTreeHeight(root);

System.out.println(treeStatistics);

}

break;

}

}while (choice!=1 || choice!=2);

}

}

public int getTreeHeight(GameNode node){

if (node==null){

return 0;

}else {

int lDepth = getTreeHeight(node.oneMoveChildren);

int rDepth = getTreeHeight(node.twoMovesChildren);

if (lDepth>rDepth)

return lDepth+1;

else

return rDepth+1;

}

}

}

Code for TestClass class:

import java.util.Scanner;

public class TestClass {

public static void main(String[] args){

Scanner sc = new Scanner(System.in);

System.out.print("Enter the number of elements (0 to exit): ");

int nOfElements = sc.nextInt();

if (nOfElements==0){

System.exit(-1);

}

GameTree nimTree = new GameTree(nOfElements);

nimTree.generateNextMoves(nimTree.root);

System.out.println("> Created tree with "+nOfElements+" elements: "+nimTree.root);

System.out.println("Choose the CPU strategy 1.Perfect || 2.Random");

int strategyChoice = sc.nextInt();

if (strategyChoice==1)

System.out.println("CPU strategy is: Perfect strategy (deterministic with minimax, remove 2 if no move is optimal)");

else

System.out.println("CPU strategy is: Random strategy (Taking random move 1 or 2)");

boolean isPerfectStrategy = strategyChoice==1?true:false;

System.out.print("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\nDo you want to go first? (true/false)");

boolean playerGoesFirst = sc.nextBoolean();

if (playerGoesFirst){

System.out.println("> Player is first");

nimTree.assignMinMaxValues(true);

nimTree.playerTurn(nimTree.root, isPerfectStrategy);

}else {

System.out.println("> Computer is first");

nimTree.assignMinMaxValues(false);

if (isPerfectStrategy)

nimTree.CPUTurn(nimTree.root, isPerfectStrategy);

else

nimTree.CPUTurnRandom(nimTree.root, isPerfectStrategy);

}

}

}

Output screen shots:

1.Start the game with perfect strategy

A screenshot of a computer

Description automatically generated

2. CPU plays first then the player plays with aid of hint:

A screenshot of a computer

Description automatically generated

3.Show tree statistics:

A screenshot of a computer

Description automatically generated

4.CPU Won!:

Text

Description automatically generated

Supplementary info: Diagram

Description automatically generated with medium confidence

Diagram

Description automatically generated