HW1

|  |  |
| --- | --- |
| Student Name | Student Number |
| Mazen Mohsen Tohari | 411003806 |
| Yazan Mahmoud Barakat | 411010698 |

Q1: Write/develop a software program using any programming language to simulate 8 puzzle problem (Game).

A1:

using System;  
using System.Collections.Generic;  
  
namespace EightPuzzleProblem  
{  
 class Program  
 {  
 static void Main(string[] args)  
 {  
 int[,] initialState = new int[3, 3]  
 {  
 { 7, 2, 0 },  
 { 5, 6, 4 },  
 { 1, 8, 3 }  
 };  
  
 int[,] goalState = new int[3, 3]  
 {  
 { 0, 1, 2 },  
 { 3, 4, 5 },  
 { 6, 7, 8 }  
 };  
  
 List<int[,]> solution = GeneralSearch(initialState, goalState);  
  
 if (solution == null)  
 {  
 Console.WriteLine("No solution found.");  
 }  
 else  
 {  
 Console.WriteLine("Solution found:");  
 foreach (int[,] state in solution)  
 {  
 for (int i = 0; i < 3; i++)  
 {  
 for (int j = 0; j < 3; j++)  
 {  
 Console.Write(state[i, j] + " ");  
 }  
 Console.WriteLine();  
 }  
 Console.WriteLine();  
 }  
 }  
 }  
  
 static List<int[,]> GeneralSearch(int[,] initialState, int[,] goalState)  
 {  
 Queue<List<int[,]>> queue = new Queue<List<int[,]>>();  
 List<int[,]> initialPath = new List<int[,]> { initialState };  
 queue.Enqueue(initialPath);  
 HashSet<string> visitedStates = new HashSet<string>();  
 visitedStates.Add(ConvertToString(initialState));  
  
 while (queue.Count > 0)  
 {  
 List<int[,]> path = queue.Dequeue();  
 int[,] currentState = path[path.Count - 1];  
  
 if (IsEqual(currentState, goalState))  
 {  
 return path;  
 }  
  
 int[] zeroIndex = GetZeroIndex(currentState);  
 int i = zeroIndex[0];  
 int j = zeroIndex[1];  
  
 if (i > 0)  
 {  
 int[,] up = (int[,])currentState.Clone();  
 up[i, j] = up[i - 1, j];  
 up[i - 1, j] = 0;  
 if (!visitedStates.Contains(ConvertToString(up)))  
 {  
 List<int[,]> newPath = new List<int[,]>(path);  
 newPath.Add(up);  
 queue.Enqueue(newPath);  
 visitedStates.Add(ConvertToString(up));  
 }  
 }  
  
 if (i < 2)  
 {  
 int[,] down = (int[,])currentState.Clone();  
 down[i, j] = down[i + 1, j];  
 down[i + 1, j] = 0;  
 if (!visitedStates.Contains(ConvertToString(down)))  
 {  
 List<int[,]> newPath = new List<int[,]>(path);  
 newPath.Add(down);  
 queue.Enqueue(newPath);  
 visitedStates.Add(ConvertToString(down));  
 }  
 }  
  
 if (j > 0)  
 {  
 int[,] left = (int[,])currentState.Clone();  
 left[i, j] = left[i, j - 1];  
 left[i, j - 1] = 0;  
 if (!visitedStates.Contains(ConvertToString(left)))  
 {  
 List<int[,]> newPath = new List<int[,]>(path);  
 newPath.Add(left);  
 queue.Enqueue(newPath);  
 visitedStates.Add(ConvertToString(left));  
 }  
 }  
  
 if (j < 2)  
 {  
 int[,] right = (int[,])currentState.Clone();  
 right[i, j] = right[i, j + 1];  
 right[i, j + 1] = 0;  
 if (!visitedStates.Contains(ConvertToString(right)))  
 {  
 List<int[,]> newPath = new List<int[,]>(path);  
 newPath.Add(right);  
 queue.Enqueue(newPath);  
 visitedStates.Add(ConvertToString(right));  
 }  
 }  
 }  
  
 return null;  
 }  
  
 static string ConvertToString(int[,] state)  
 {  
 string str = "";  
 for (int i = 0; i < 3; i++)  
 {  
 for (int j = 0; j < 3; j++)  
 {  
 str += state[i, j];  
 }  
 }  
 return str;  
 }  
  
 static bool IsEqual(int[,] state1, int[,] state2)  
 {  
 for (int i = 0; i < 3; i++)  
 {  
 for (int j = 0; j < 3; j++)  
 {  
 if (state1[i, j] != state2[i, j])  
 {  
 return false;  
 }  
 }  
 }  
 return true;  
 }  
  
 static int[] GetZeroIndex(int[,] state)  
 {  
 int[] index = new int[2];  
 for (int i = 0; i < 3; i++)  
 {  
 for (int j = 0; j < 3; j++)  
 {  
 if (state[i, j] == 0)  
 {  
 index[0] = i;  
 index[1] = j;  
 return index;  
 }  
 }  
 }  
 return index;  
 }  
 }  
}

Q2: Write/develop a software program using any programming language to simulate XOR problem (tic tac toe).

*using* System;  
*using* System.Collections.Generic;  
  
*namespace* TicTacToe  
{  
 *class* Program  
 {  
 *static char*[,] board = *new char*[*3*, *3*]  
 {  
 { ' ', ' ', ' ' },  
 { ' ', ' ', ' ' },  
 { ' ', ' ', ' ' }  
 };  
 *static char* currentPlayer = 'X';  
  
 *static void Main*(*string*[] args)  
 {  
 *DrawBoard*();  
  
 *while* (*true*)  
 {  
 *if* (currentPlayer == 'X')  
 {  
 Console.*WriteLine*("Player X, enter row and column (0-2) separated by a space:");  
 *int* row = *int*.*Parse*(Console.*ReadLine*());  
 *int* col = *int*.*Parse*(Console.*ReadLine*());  
  
 *if* (board[row, col] == ' ')  
 {  
 board[row, col] = currentPlayer;  
 *DrawBoard*();  
 }  
 *else* {  
 Console.*WriteLine*("Cell is already occupied, try again.");  
 *continue*;  
 }  
 }  
 *else* {  
 List<Tuple<*int*, *int*>> moves = *GetValidMoves*();  
   
 *if* (moves.Count == *0*)  
 {  
 Console.*WriteLine*("It's a draw.");  
 *break*;  
 }  
   
 Random rnd = *new* Random();  
 *int* moveIndex = rnd.*Next*(moves.Count);  
 Tuple<*int*, *int*> move = moves[moveIndex];  
 Console.*WriteLine*("AI's turn:");  
 Console.*WriteLine*("AI's possible moves:");  
 *foreach* (*var* mov *in* moves)  
 {  
 Console.*WriteLine*($"Row: {mov.Item1} Column: {mov.Item2}");  
 }  
 board[move.Item1, move.Item2] = currentPlayer;  
 *DrawBoard*();  
   
   
 }  
 *if* (*CheckWin*(currentPlayer))  
 {  
 Console.*WriteLine*($"Player {currentPlayer} wins!");  
 *break*;  
 }  
 currentPlayer = currentPlayer == 'X' ? 'O' : 'X';  
 }  
 }  
  
 *static void DrawBoard*()  
 {  
 Console.*WriteLine*(" 0 1 2");  
 Console.*WriteLine*("0 " + board[*0*, *0*] + "|" + board[*0*, *1*] + "|" + board[*0*, *2*]);  
 Console.*WriteLine*(" ----");  
 Console.*WriteLine*("1 " + board[*1*, *0*] + "|" + board[*1*, *1*] + "|" + board[*1*, *2*]);  
 Console.*WriteLine*(" ----");  
 Console.*WriteLine*("2 " + board[*2*, *0*] + "|" + board[*2*, *1*] + "|" + board[*2*, *2*]);  
 }  
  
 *static bool CheckWin*(*char* player)  
 {  
 *for* (*int* i = *0*; i < *3*; i++)  
 {  
 *if* (board[i, *0*] == player && board[i, *1*] == player && board[i, *2*] == player)  
 {  
 *return true*;  
 }  
 }  
  
 *for* (*int* j = *0*; j < *3*; j++)  
 {  
 *if* (board[*0*, j] == player && board[*1*, j] == player && board[*2*, j] == player)  
 {  
 *return true*;  
 }   
 }  
   
 *if* (board[*0*, *0*] == player && board[*1*, *1*] == player && board[*2*, *2*] == player)  
 {  
 *return true*;  
 }  
  
 *if* (board[*0*, *2*] == player && board[*1*, *1*] == player && board[*2*, *0*] == player)  
 {  
 *return true*;  
 }  
  
 *return false*;  
 }  
  
 *static bool* IsDraw()  
 {  
 *for* (*int* i = *0*; i < *3*; i++)  
 {  
 *for* (*int* j = *0*; j < *3*; j++)  
 {  
 *if* (board[i, j] == ' ')  
 {  
 *return false*;  
 }  
 }  
 }  
 *return true*;  
 }  
  
 *static* List<Tuple<*int*, *int*>> *GetValidMoves*()  
 {  
 List<Tuple<*int*, *int*>> moves = *new* List<Tuple<*int*, *int*>>();  
 *for* (*int* i = *0*; i < *3*; i++)  
 {  
 *for* (*int* j = *0*; j < *3*; j++)  
 {  
 *if* (board[i, j] == ' ')  
 {  
 moves.*Add*(*new* Tuple<*int*, *int*>(i, j));  
 }  
 }  
 }  
 *return* moves;  
 }  
 }  
}

Q3: Write/develop a software program using any programming language to simulate missionaries and cannibals problem.

A3:

using System;  
using System.Collections.Generic;  
  
class MissionariesAndCannibals  
{  
 static int missionCount = 3;  
 static int canniCount = 3;  
 static int boatSize = 2;  
  
 static void Main(string[] args)  
 {  
 Node startNode = new Node(missionCount, canniCount, 0, 0, true);  
 Node endNode = new Node(0, 0, missionCount, canniCount, false);  
  
 GeneralSearch(startNode, endNode);  
 }  
  
 static void GeneralSearch(Node start, Node end)  
 {  
 Queue<Node> queue = new Queue<Node>();  
 HashSet<Node> visited = new HashSet<Node>();  
 Dictionary<Node, Node> parent = new Dictionary<Node, Node>();  
  
 queue.Enqueue(start);  
 visited.Add(start);  
 parent[start] = null;  
  
 while (queue.Count > 0)  
 {  
 Node current = queue.Dequeue();  
  
 if (current.IsEqual(end))  
 {  
 Console.WriteLine("Solution found!");  
 PrintSolution(parent, start, end);  
 return;  
 }  
  
 List<Node> children = current.GenerateChildren();  
 foreach (Node child in children)  
 {  
 if (!visited.Contains(child))  
 {  
 queue.Enqueue(child);  
 visited.Add(child);  
 parent[child] = current;  
 }  
 }  
 }  
  
 Console.WriteLine("No solution found.");  
 }  
  
 static void PrintSolution(Dictionary<Node, Node> parent, Node start, Node end)  
 {  
 List<Node> path = new List<Node>();  
 Node current = end;  
 while (current != null)  
 {  
 path.Add(current);  
 current = parent[current];  
 }  
  
 path.Reverse();  
 foreach (Node node in path)  
 {  
 Console.WriteLine(node);  
 }  
 }  
  
  
 class Node  
 {  
 int missionLeft;  
 int canniLeft;  
 int missionRight;  
 int canniRight;  
 bool boatOnLeft;  
  
 public Node(int missionLeft, int canniLeft, int missionRight, int canniRight, bool boatOnLeft)  
 {  
 this.missionLeft = missionLeft;  
 this.canniLeft = canniLeft;  
 this.missionRight = missionRight;  
 this.canniRight = canniRight;  
 this.boatOnLeft = boatOnLeft;  
 }  
  
 public List<Node> GenerateChildren()  
 {  
 List<Node> children = new List<Node>();  
  
 int missionOnBoat = boatOnLeft ? -1 : 1;  
 int canniOnBoat = boatOnLeft ? -1 : 1;  
  
 for (int i = 0; i <= boatSize; i++)  
 {  
 for (int j = 0; j <= boatSize - i; j++)  
 {  
 if (i == 0 && j == 0) continue;  
 int newMissionLeft = missionLeft + missionOnBoat \* i;  
 int newCanniLeft = canniLeft + canniOnBoat \* j;  
 int newMissionRight = missionRight - missionOnBoat \* i;  
 int newCanniRight = canniRight - canniOnBoat \* j;  
  
 if (newMissionLeft >= 0 && newMissionLeft <= missionCount && newCanniLeft >= 0 &&  
 newCanniLeft <= canniCount &&  
 newMissionRight >= 0 && newMissionRight <= missionCount && newCanniRight >= 0 &&  
 newCanniRight <= canniCount)  
 {  
 if ((newMissionLeft >= newCanniLeft || newMissionLeft == 0) &&  
 (newMissionRight >= newCanniRight || newMissionRight == 0))  
 {  
 Node child = new Node(newMissionLeft, newCanniLeft, newMissionRight, newCanniRight,  
 !boatOnLeft);  
 children.Add(child);  
 }  
 }  
 }  
 }  
  
 return children;  
 }  
  
 public override string ToString()  
 {  
 string boatPos = boatOnLeft ? "left" : "right";  
 return "Missionaries: " + missionLeft + " left, " + missionRight + " right; Cannibals: " + canniLeft +  
 " left, " + canniRight + " right; Boat is on the " + boatPos;  
 }  
  
 public bool IsEqual(Node other)  
 {  
 return missionLeft == other.missionLeft && canniLeft == other.canniLeft &&  
 missionRight == other.missionRight && canniRight == other.canniRight &&  
 boatOnLeft == other.boatOnLeft;  
 }  
  
 public override bool Equals(object obj)  
 {  
 Node other = obj as Node;  
 if (other == null) return false;  
 return IsEqual(other);  
 }  
  
 public override int GetHashCode()  
 {  
 int hash = 13;  
 hash = (hash \* 7) + missionLeft.GetHashCode();  
 hash = (hash \* 7) + canniLeft.GetHashCode();  
 hash = (hash \* 7) + missionRight.GetHashCode();  
 hash = (hash \* 7) + canniRight.GetHashCode();  
 hash = (hash \* 7) + boatOnLeft.GetHashCode();  
 return hash;  
 }  
 }  
}