**SHERIDAN COLLEGE**

Student Name:

Ngoc Phuong Thao Doan 991466176

Instructor: Dario Guiao

Assignment 1

PROG37721 Web Services Using .NET and C# Programming

June 7, 2019

**TABLE OF CONTENTS**

1 – Problem definition 2

2 – Modularity 2

3 – Data structure 3

4 – Flow Chart 3

5 – Test program 4

6 – Code 5

1. **Requirement**

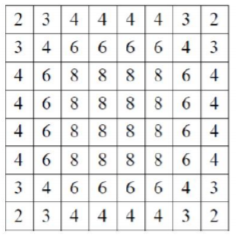
You should have the Notepad application to click the “Open file” button

1. **Problem definition**

The program is about the Knight Tour game, the application should be written in C# using Window forms. The chess board game has 64 squares and the knight can make an L-shaped moves as many as it can, and it can only move to every square once, no repeating squares are allowed

There are two methods for this game:

* Non-intelligent method: the knight to move to a random accessible square.
* Heuristics method: the knight should move the lowest accessible number from the current position as the board below describes.



Once the game is over, based on the method you have selected, print the results to either the file *ThaoDoanNonIntelligentMethod.txt* or *ThaoDoanHeuristicsMethod.txt*

1. **Modularity**

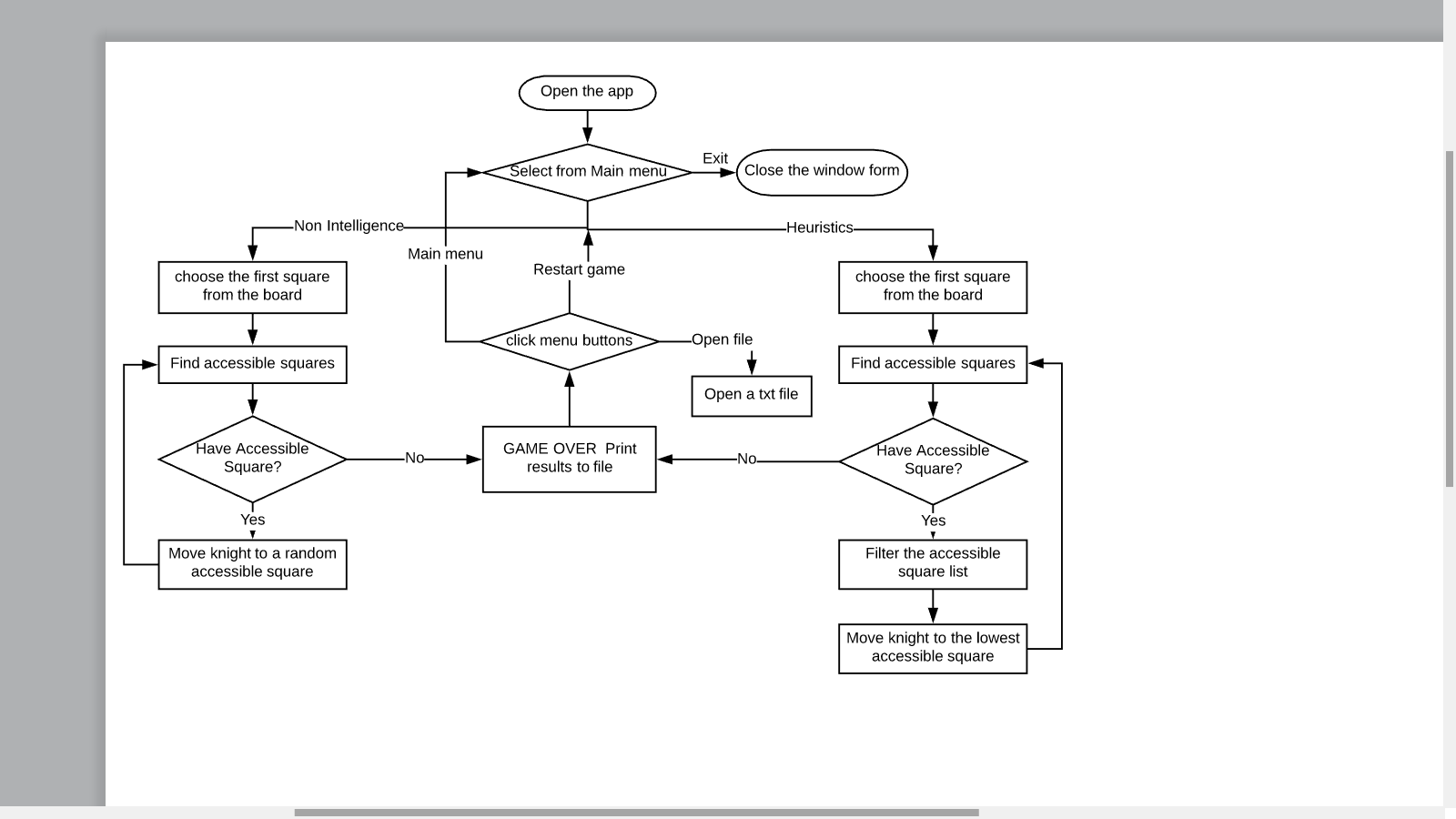
|  |  |
| --- | --- |
| Type | Class Names |
| Interface | IBoardPosition.cs |
| Parent class | Possition.cs |
| Child classes | Square1.cs  Square2.cs |
| Partial classes | Square1.cs  Square2.cs |
| Properties | *(In Position.cs class)*   * Int X (line 11) * Int Y (Line 15) |
| Others | Form1.cs |

1. **Data structure**

The data structure used in this program including:

1. **- Flow chart**

This chart below describes the flow of program processes:



1. **Test program**

In this program, you do not have to tell how many times to run the game beforehand, you can click the “restart” button to restart the game with the same method or click “main menu” to go back to main menu and choose another method

|  |  |
| --- | --- |
| 1. Choose an option from the main Menu | 1. Click any square from the board |
| 1. Once the game is over, select either to restart the game, go back to menu or open a file for that method | 1. The txt file for the corresponding method will be shown on NotePad |

1. **Code**

In the parent class: **Possition.cs**

namespace ThaoDoanAssignment1

{

class Possition

{

public int X

{

get; set;

}

public int Y

{

get; set;

}

}

}

In the Interface: **IBoardPosition.cs**

namespace ThaoDoanAssignment1

{

interface IBoardPosition

{

bool IsLocValid();

bool IsLocAvailable(int[,] board);

}

}

In the partial class: **Square1.cs**

namespace ThaoDoanAssignment1

{

partial class Square : Possition, IBoardPosition

{

public Square() { }

public Square(int y, int x)

{

X = x;

Y = y;

}

public bool IsLocValid()

{

return (Y >= 0 && X >= 0 && Y < 8 && X < 8);

}

}

}

In the partial class: **Square2.cs**

namespace ThaoDoanAssignment1

{

partial class Square : Possition, IBoardPosition

{

public bool IsLocAvailable(int[,] board)

{

return board[Y, X] == 0;

}

}

}

In the **Form1.cs**

namespace ThaoDoanAssignment1

{

public partial class Form1 : Form

{

static Random r = new Random();

static Button[,] buttons = new Button[8, 8];

static private int[,] board;

static Square currLoc;

static int moveNumber, trialCount;

static private string fileName, methodName;

public Form1()

{

board = new int[8, 8];

trialCount = 0;

moveNumber = 0;

currLoc = new Square();

InitializeComponent();

CreateBoard();

}

/\*

\* This function will create a chessboard and add it on the main form in run time

\* each square will contain a value of 0

\*/

public void CreateBoard()

{

for (int i = 0; i < 8; i++)

{

for (int j = 0; j < 8; j++)

{

board[i, j] = 0;

buttons[i, j] = new Button();

buttons[i, j].Margin = new Padding(0, 0, 0, 0);

buttons[i, j].Size = new Size(50, 50);

buttons[i, j].Tag = i + "," + j;

buttons[i, j].Click += BtnFirstMove\_Click;

if ((i + j) % 2 == 0)

buttons[i, j].BackColor = Color.DimGray;

else

buttons[i, j].BackColor = Color.White;

tableLayoutPanel1.Controls.Add(buttons[i, j], j, i);

}

}

}

/\* When you select the Heauristics Button, it will automatically switch to the form

\* that contain the chessboard

\* the label on the top display the method name that you selected

\* and the relative path name of the ThaoDoanHeuristicsMethod.txt file will be set as well

\*/

private void HeuristicsBtn\_Click(object sender, EventArgs e)

{

methodName = "Heuristics";

MethodLabel.Text = methodName + " method ";

fileName = "..\\..\\ThaoDoanHeuristicsMethod.txt";

switchForms();

}

/\* When you select the Non intelligent Button, it will automatically switch to the other form

\* the label on the top display the method name that you selected

\* and the relative path name of the ThaoDoanNonIntelligentMethod.txt file will be set

\*/

private void NonIntelligentBtn\_Click(object sender, EventArgs e)

{

methodName = "Non-Intelligent";

MethodLabel.Text = methodName + " method";

fileName = "..\\..\\ThaoDoanNonIntelligentMethod.txt";

switchForms();

}

/\*

\* Exit from the application

\*/

private void ExitBtn\_Click(object sender, EventArgs e)

{

Application.Exit();

}

/\* Choose the first move from the board

\*/

private void BtnFirstMove\_Click(object sender, EventArgs e)

{

Score.Text = "";

//get the y and x cordinates from the button(y,x)

var button = (Button)sender;

string[] indexes = button.Tag.ToString().Split(',');

//After the first move, user can not click any more buttons

foreach (Button b in buttons)

b.Enabled = false;

moveKnight(Convert.ToInt32(indexes[0]), Convert.ToInt32(indexes[1]));

trialCount++;

//execute non-intelligent or Heuristics method based on user's selection

if (methodName == "Heuristics")

HeuristicsMethod();

else

NonIntelligentMethod();

LbTrialCount.Text = "Trial " + trialCount;

}

/\* move knight to the new location

\* and set the new location to currLoc (current location)

\* Mark on the board a different color for any cells that the knight have visited

\*/

static void moveKnight(int newY, int newX)

{

moveNumber++;

currLoc.Y = newY;

currLoc.X = newX;

//update the board array

board[currLoc.Y, currLoc.X] = moveNumber;

buttons[currLoc.Y, currLoc.X].Text = "" + moveNumber;

buttons[currLoc.Y, currLoc.X].BackColor = Color.Brown;

buttons[currLoc.Y, currLoc.X].Update();

buttons[currLoc.Y, currLoc.X].Refresh();

}

/\* Move the knight until there is no accessible moves and write to the coressponding file

\*/

public void NonIntelligentMethod()

{

while (moveNumber <= 64)

{

List<Square> moveOptions = FindAccessibleMoves(currLoc);

// GAME OVER if there is no accessible squares

if (moveOptions.Count == 0)

{

Score.Text = "YOUR SCORE: " + moveNumber;

WriteToFile();

break; //break the loop

}

else

{

//choose a random option from accessible squares

int random = r.Next(moveOptions.Count);

int randomX = moveOptions[random].X;

int randomY = moveOptions[random].Y;

moveKnight(randomY, randomX);

}

}

}

/\* find accessible moves for each time time the knight move to new location

\* filter the list of squares to choose only smallest accessible ones

\* Move the knight until there is no accessible squares and write to the coressponding file

\*/

public void HeuristicsMethod()

{

int[,] board2 = new int[8, 8] { {2,3,4,4,4,4,3,2 },

{3,4,6,6,6,6,4,3 },

{4,6,8,8,8,8,6,4 },

{4,6,8,8,8,8,6,4 },

{4,6,8,8,8,8,6,4 },

{4,6,8,8,8,8,6,4 },

{3,4,6,6,6,6,4,3 },

{2,3,4,4,4,4,3,2 } };

while (moveNumber <= 64)

{

List<Square> moveOptions = FindAccessibleMoves(currLoc);

// GAME OVER if there is no accessible squares

if (moveOptions.Count == 0) {

Score.Text = "YOUR SCORE: " + moveNumber;

WriteToFile();

break;

}

else

{

//filter the move options and only select good squares

List<Square> goodSquares = new List<Square>();

//find the lowest accessible number

int x1 = moveOptions[0].X;

int y1 = moveOptions[0].Y;

foreach (Square l in moveOptions) {

if (board2[l.Y, l.X] < board2[y1, x1]){

x1 = l.X;

y1 = l.Y;

}

}

// Add to the list all squares that have same lowest accessible number

foreach (Square l in moveOptions)

if (board2[l.Y, l.X] == board2[y1, x1]){

goodSquares.Add(new Square(l.Y, l.X));

}

//choose a random option from good squares from goodSquares list

int random = r.Next(goodSquares.Count);

int randomX = goodSquares[random].X;

int randomY = goodSquares[random].Y;

moveKnight(randomY, randomX);

}

}

}

/\* Find all accessible squares that the knight to move to

\* return a list of Accessible squares

\*/

static List<Square> FindAccessibleMoves(Square l)

{

List<Square> accessibilities = new List<Square>();

Square tmp = new Square(l.Y + 2, l.X + 1);

if (tmp.IsLocValid() && tmp.IsLocAvailable(board))

accessibilities.Add(tmp);

tmp = new Square(l.Y + 2, l.X - 1);

if (tmp.IsLocValid() && tmp.IsLocAvailable(board))

accessibilities.Add(tmp);

tmp = new Square(l.Y - 2, l.X + 1);

if (tmp.IsLocValid() && tmp.IsLocAvailable(board))

accessibilities.Add(tmp);

tmp = new Square(l.Y - 2, l.X - 1);

if (tmp.IsLocValid() && tmp.IsLocAvailable(board))

accessibilities.Add(tmp);

tmp = new Square(l.Y + 1, l.X + 2);

if (tmp.IsLocValid() && tmp.IsLocAvailable(board))

accessibilities.Add(tmp);

tmp = new Square(l.Y + 1, l.X - 2);

if (tmp.IsLocValid() && tmp.IsLocAvailable(board))

accessibilities.Add(tmp);

tmp = new Square(l.Y - 1, l.X + 2);

if (tmp.IsLocValid() && tmp.IsLocAvailable(board))

accessibilities.Add(tmp);

tmp = new Square(l.Y - 1, l.X - 2);

if (tmp.IsLocValid() && tmp.IsLocAvailable(board))

accessibilities.Add(tmp);

return accessibilities;

}

/\* Switch between the front and the back form

\*/

public void switchForms() {

if (panel1.Visible == true){

panel1.Visible = false;

panel2.Visible = true;

}

else{

panel2.Visible = false;

panel1.Visible = true;

}

}

/\* reset all components related to the game back to begining

\*/

public void restartGame()

{

LbTrialCount.Text = "";

for (int i = 0; i < 8; i++)

{

for (int j = 0; j < 8; j++)

{

//reset color

if ((i + j) % 2 == 0)

buttons[i, j].BackColor = Color.DimGray;

else

buttons[i, j].BackColor = Color.White;

board[i, j] = 0;

buttons[i, j].Text = "";

Score.Text = "";

buttons[i, j].Enabled = true;

moveNumber = 0;

}

}

}

/\* Check if the file exist

\* print output of each trial to the output file or display error message if it does not exist

\* If this is the first trial, overwrite the output file to empty (in case it was not empty)

\*/

public static void WriteToFile()

{

if (!File.Exists(fileName)){

MessageBox.Show("File not found\n", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

else {

if (trialCount == 1)

using (StreamWriter fileWriter = new StreamWriter(fileName))

{

fileWriter.Write("");

}

using (StreamWriter fileWriter = File.AppendText(fileName))

{

fileWriter.WriteLine("Trial {0}: The Knight was successully touch {1} squares ", trialCount, moveNumber);

for (int i = 0; i < 8; i++) {

for (int j = 0; j < 8; j++)

fileWriter.Write("{0,4}", board[i, j]);

fileWriter.WriteLine();

}

fileWriter.WriteLine();

}

}

}

private void RestartBtn\_Click(object sender, EventArgs e)

{

restartGame();

}

/\*

\* Reset game

\* switch to the main menu

\* the number of trial is reset to 0

\*/

private void MainMenuBtn\_Click\_1(object sender, EventArgs e)

{

trialCount = 0;

switchForms();

restartGame();

}

/\* Check if the file exist

\* Otherwise, process to open the file on the notepad app

\*/

private void OpenFile\_Click(object sender, EventArgs e)

{

if (!File.Exists(fileName))

MessageBox.Show("File not found\n", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);

else

Process.Start("notepad.exe", fileName);

}

}

}