ФЕДЕРАЛЬНОЕ государственное БЮДЖЕТНОЕ

образовательное учреждение

высшего образования

«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Кафедра защиты информации

**

**ОТЧЁТ**

**по лабораторной работе № 5**

**«Наследование. Полиморфизм»**

**по дисциплине: «***Программирование***»**

Выполнил:Проверил:

Студент гр. «АБ-121», «АВТФ» *доцент кафедры ЗИ*

*Новиков Втюрин Александр Романович Архипова А. Б.*

«21» декабря 2022г«\_\_\_» \_\_\_\_\_\_ 2022 г.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись) (подпись)

Новосибирск 2022

**Задание к работе**: Написать программу по созданию абстрактного базового класса с виртуальными методами и производных классов в контексте понятий наследования и полиморфизма в ООП. Работа выполняется на двух ЯП (С++ и другой). Работа может быть выполнена только на одном ЯП только в случае, если  реализована графическая оболочка (вес ЛР = 1).

**Вариант 12:**

Реализовать игру «Сапер», работающую по следующим правилам:

• В начале игры случайным образом генерируется поле;

• Количество мин каждый раз случайное или задается перед началом игры;

• После окончания игры игрок узнает, сколько мин было найдено и сколько мин было всего сгенерировано;

• При нажатии правой кнопки мыши, флагом помечается клетка.

Игрок открывает ячейки, стараясь не открыть ячейку с миной. Открыв ячейку с миной, он проигрывает. Если под открытой ячейкой мины нет, то в ней появляется число, показывающее, сколько ячеек, соседствующих с только что открытой, «заминировано» (в каждом варианте игры соседство определяется по-своему); используя эти числа, игрок пытается рассчитать расположение мин, однако иногда даже в середине и в конце игры некоторые ячейки всё же приходится открывать наугад. Если под соседними ячейками тоже нет мин, то открывается некоторая «не заминированная» область до ячеек, в которых есть цифры или случайные поля. «Заминированные» ячейки игрок может пометить, чтобы случайно не открыть их. Открыв все «не заминированные» ячейки или отметив флагом все ячейки, где находятся мины, игрок выигрывает. Дополнительные задания:

• Поле генерируется после первого открывания ячейки, чтобы игрок не проиграл при первом же ходе;

• Возможность выбирать уровень сложности (количество мин и размер поля);

• Реализация текстур.

**С#**

**Programm.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace Lab5

{

static class Program

{

/// <summary>

/// Главная точка входа для приложения.

/// </summary>

[STAThread]

static void Main()

{

Application.EnableVisualStyles();

Application.SetCompatibleTextRenderingDefault(false);

Application.Run(new GameWindow());

}

}

}

**GameWindow.cs**

using Miner.Controllers;

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace Miner

{

public partial class GameWindow : Form

{

public GameWindow()

{

InitializeComponent();

this.Size = new Size(400, 500);

Difficulty.Show();

Easy.Show();

Medium.Show();

Hard.Show();

}

private void Form1\_Load(object sender, EventArgs e)

{

}

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

{

Game.n = 0;

Difficulty.Hide();

Easy.Hide();

Medium.Hide();

Hard.Hide();

Game.Init(this);

}

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

{

Game.n = 1;

Difficulty.Hide();

Easy.Hide();

Medium.Hide();

Hard.Hide();

Game.Init(this);

}

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

{

Game.n = 2;

Difficulty.Hide();

Easy.Hide();

Medium.Hide();

Hard.Hide();

Game.Init(this);

}

}

}

**Game.cs**

using System;

using System.Collections.Generic;

using System.Drawing;

using System.IO;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace Lab5

{

public class Game

{

public static readonly int[] mapSize = {8, 16, 24};

public const int cellSize = 50;

public static int n;

private static int currentPictureToSet = 0;

public static int[,] map;

public static int[,] bombFlag;

public static Button[,] buttons;

public static Image spriteSet;

private static bool isFirstStep;

private static Point firstCoord;

public static Form form;

private static void ConfigureMapSize(Form current, int n)

{

current.Width = mapSize[n] \* cellSize + 20;

current.Height = (mapSize[n] + 1) \* cellSize;

}

private static void InitMap(int n)

{

for (int i = 0; i < mapSize[n]; i++)

{

for (int j = 0; j < mapSize[n]; j++)

{

map[i, j] = 0;

bombFlag[i, j] = 0;

}

}

}

public static void Init(Form current)

{

form = current;

map = new int[mapSize[n], mapSize[n]];

buttons = new Button[mapSize[n], mapSize[n]];

bombFlag = new int[mapSize[n], mapSize[n]];

currentPictureToSet = 0;

isFirstStep = true;

spriteSet = new Bitmap(Path.Combine(new DirectoryInfo(Directory.GetCurrentDirectory()).Parent.Parent.FullName.ToString(), "Sprites\\tiles.png"));

ConfigureMapSize(current, n);

InitMap(n);

InitButtons(current, n);

}

private static void InitButtons(Form current, int n)

{

for (int i = 0; i < mapSize[n]; i++)

{

for (int j = 0; j < mapSize[n]; j++)

{

Button button = new Button();

button.Location = new Point(j \* cellSize, i \* cellSize);

button.Size = new Size(cellSize, cellSize);

button.Image = FindNeededImage(0, 0);

button.MouseUp += new MouseEventHandler(OnButtonPressedMouse);

current.Controls.Add(button);

buttons[i, j] = button;

}

}

}

private static void OnButtonPressedMouse(object sender, MouseEventArgs e)

{

Button pressedButton = sender as Button;

switch (e.Button.ToString())

{

case "Right":

OnRightButtonPressed(pressedButton);

break;

case "Left":

OnLeftButtonPressed(pressedButton);

break;

}

}

private static void OnRightButtonPressed(Button pressedButton)

{

currentPictureToSet++;

currentPictureToSet %= 2;

int posX = 0;

int posY = 0;

int iButton = pressedButton.Location.Y / cellSize;

int jButton = pressedButton.Location.X / cellSize;

switch (currentPictureToSet)

{

case 0:

posX = 0;

posY = 0;

bombFlag[iButton, jButton] = 0;

break;

case 1:

posX = 0;

posY = 2;

bombFlag[iButton, jButton] = 1;

isWin();

break;

}

pressedButton.Image = FindNeededImage(posX, posY);

}

private static void isWin()

{

int bomb = 0;

int findbomb = 0;

for (int i = 0; i < mapSize[n]; i++)

{

for (int j = 0; j < mapSize[n]; j++)

{

if (map[i, j] == -1){

bomb++;

if (bombFlag[i, j] == 1)

{

findbomb++;

}

}

}

}

if (bomb == findbomb)

{

MessageBox.Show("Победа!");

form.Controls.Clear();

Init(form);

}

}

private static void OnLeftButtonPressed(Button pressedButton)

{

pressedButton.Enabled = false;

int iButton = pressedButton.Location.Y / cellSize;

int jButton = pressedButton.Location.X / cellSize;

if (isFirstStep)

{

firstCoord = new Point(jButton,iButton);

SeedMap(n);

CountCellBomb(n);

isFirstStep = false;

}

OpenCells(iButton, jButton);

if (map[iButton, jButton] == -1)

{

ShowAllBombs(iButton,jButton, n);

MessageBox.Show("Поражение!");

form.Controls.Clear();

Init(form);

}

}

private static void ShowAllBombs(int iBomb,int jBomb, int n)

{

for (int i = 0; i < mapSize[n]; i++)

{

for (int j = 0; j < mapSize[n]; j++)

{

if (i == iBomb && j == jBomb)

continue;

if(map[i, j] == -1)

{

buttons[i, j].Image = FindNeededImage(3, 2);

}

}

}

}

public static Image FindNeededImage(int xPos,int yPos)

{

Image image = new Bitmap(cellSize, cellSize);

Graphics g = Graphics.FromImage(image);

g.DrawImage(spriteSet, new Rectangle(new Point(0, 0), new Size(cellSize, cellSize)), 0 + 32 \* xPos, 0 + 32\* yPos,33,33,GraphicsUnit.Pixel);

return image;

}

private static void SeedMap(int n)

{

Random r = new Random();

int number = r.Next(mapSize[n] , (mapSize[n]\* mapSize[n])/2);

MessageBox.Show("На поле обнаруженно " + number.ToString()+ " бомб");

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

{

int posI = r.Next(0, mapSize[n] - 1);

int posJ = r.Next(0, mapSize[n] - 1);

while (map[posI, posJ] == -1 || (Math.Abs(posI-firstCoord.Y)<=1 && Math.Abs(posJ - firstCoord.X) <= 1))

{

posI = r.Next(0, mapSize[n] - 1);

posJ = r.Next(0, mapSize[n] - 1);

}

map[posI, posJ] = -1;

}

}

private static void CountCellBomb(int n)

{

for(int i = 0; i < mapSize[n]; i++)

{

for(int j = 0; j < mapSize[n]; j++)

{

if (map[i, j] == -1)

{

for(int k = i - 1; k < i + 2; k++)

{

for(int l = j - 1; l < j + 2; l++)

{

if (!IsInBorder(k, l, n) || map[k, l] == -1)

continue;

map[k, l] = map[k, l] + 1;

}

}

}

}

}

}

private static void OpenCell(int i,int j)

{

buttons[i, j].Enabled = false;

switch (map[i, j])

{

case 1:

buttons[i, j].Image = FindNeededImage(1, 0);

break;

case 2:

buttons[i, j].Image = FindNeededImage(2, 0);

break;

case 3:

buttons[i, j].Image = FindNeededImage(3, 0);

break;

case 4:

buttons[i, j].Image = FindNeededImage(4, 0);

break;

case 5:

buttons[i, j].Image = FindNeededImage(0, 1);

break;

case 6:

buttons[i, j].Image = FindNeededImage(1, 1);

break;

case 7:

buttons[i, j].Image = FindNeededImage(2, 1);

break;

case 8:

buttons[i, j].Image = FindNeededImage(3, 1);

break;

case -1:

buttons[i, j].Image = FindNeededImage(1, 2);

break;

case 0:

buttons[i, j].Image = FindNeededImage(0, 0);

break;

}

}

private static void OpenCells(int i,int j)

{

OpenCell(i, j);

if (map[i, j] > 0)

return;

for(int k = i - 1; k < i + 2; k++)

{

for(int l = j - 1; l < j + 2; l++)

{

if (!IsInBorder(k, l, n))

continue;

if (!buttons[k, l].Enabled)

continue;

if (map[k, l] == 0)

OpenCells(k, l);

else if (map[k, l] > 0)

OpenCell(k, l);

}

}

}

private static bool IsInBorder(int i,int j, int n)

{

if(i<0 || j<0 || j>mapSize[n] -1 || i > mapSize[n] - 1)

{

return false;

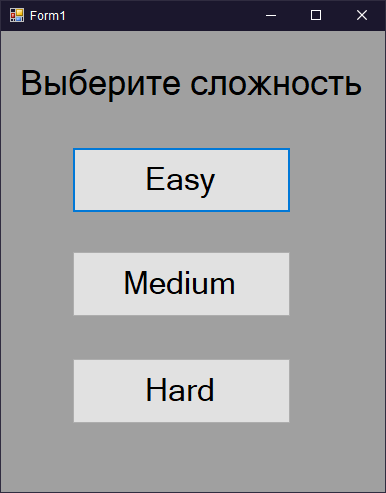
}

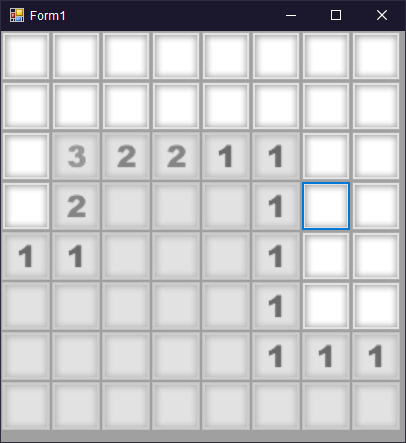
return true;

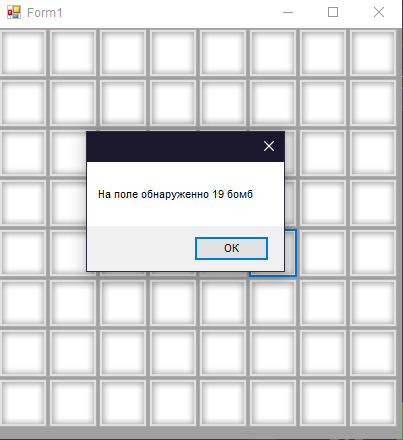
}

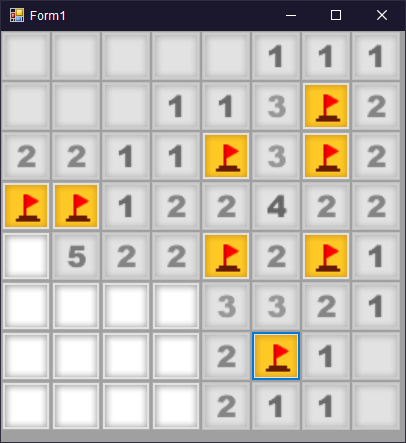
}

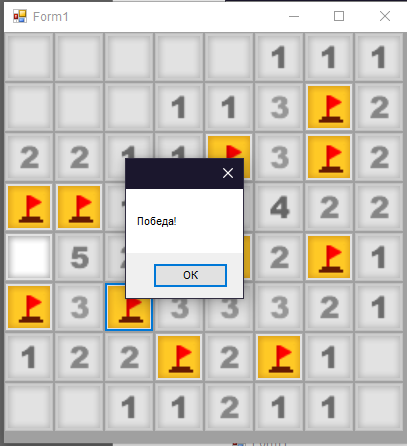
}

****

****

****

****

****