МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ

(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

Факультет №3. Системы управления, информатика и электроэнергетика.

Кафедра 302

Лабораторная работа по дисциплине «ООП»

Выполнили студенты группы: 3О-208Б:

Чернышевич Е.А.

Макурин В.М.

Принял:

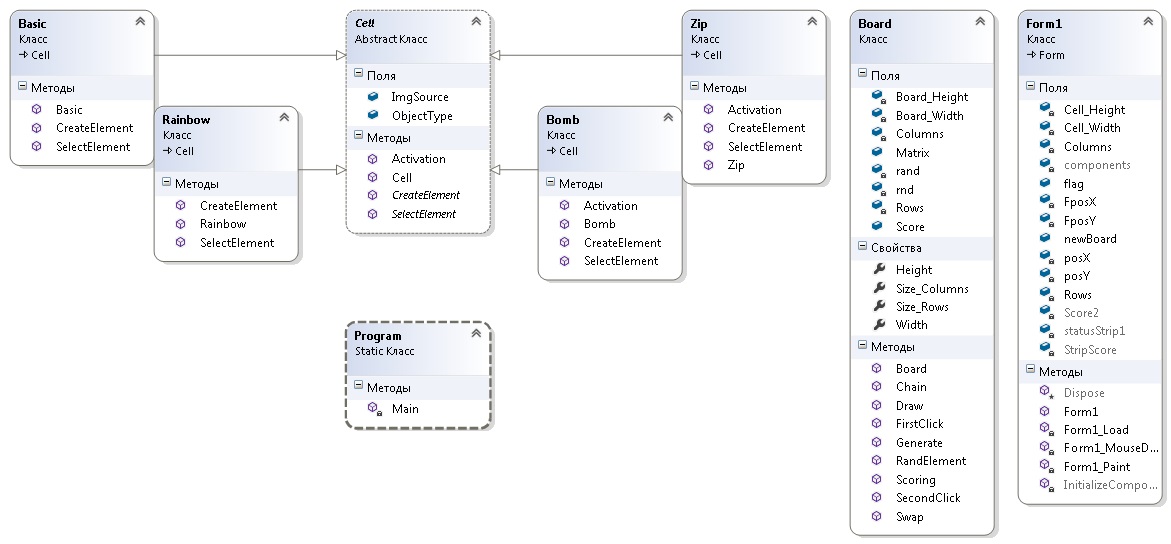
Фетисов А.А.

Москва 2014 год

**Содержание**

1. Формализованная постановка
2. Описание основных алгоритмов
3. UML-диаграммы классов
4. Сведения о реализации
5. Руководство пользователя
6. Тесты
7. Исходный код
8. **Формализованная постановка**

Написать программу, реализующую игру «три в ряд»

*Диаграмма классов:*

1. **Описание основных алгоритмов**

*Метод Generate*

// генерация элементов игрового поля

Начало

Создаем матрицу размером 10 на 8 из объектов (цветная ячейка, молния, бомба и радужный квадрат)

Присваиваем ячейке матрицы случайный объект (цветная ячейка, молния, бомба или радужный квадрат)

Конец

*Метод Draw*

// рисование игрового поля

Начало

Рисуем на форме объекты матрицы

Конец

*Метод Scoring*

// подсчитывание количества набранных очков

Начало

Запоминаем текущее количество очков

Подсчитываем очки по горизонтали

Проверяем ячейки справа и слева от текущей ячейки на одинаковость (или на радужный квадрат)

Если проверяемые ячейки равны между собой,

то заменяем найденные одинаковые ячейки другими случайными объектами

и увеличиваем очки на одну единицу

Подсчитываем очки по вертикали

Проверяем ячейки снизу и сверху от текущей ячейки на одинаковость (или на радужный квадрат)

Если проверяемые ячейки равны между собой,

то заменяем найденные одинаковые ячейки другими случайными объектами

и увеличиваем очки на одну единицу

Конец

*Метод FirstClick*

// клик по первому элементу игрового поля

Начало

Проверяем возможность активации при клике на ячейку

Если выбранная ячейка не является бомбой или молнией,

то выделяем выбранную ячейку(затемняем)

иначе подсчитываем очки после активации

Конец

*Метод SecondClick*

// клик по второму элементу игрового поля

Начало

Проверяем корректность второго клика

Если выбранная ячейка является цветной ячейкой (желтой, голубой, зеленой или красной) и выбранная ячейка находится в той же строке или столбце, что и ячейка, которую выбрали при первом клике

то проверяем возможность образования цепочки в результате перемены мест

Если цепочки не образовалось, то снимаем выделение с ячейки из первого клика

Конец

*Метод Chain*

// удаление цепочки из трех или более одинаковых по цвету (в том числе радужного квадрата)

Начало

Запоминаем текущее количество очков

Меняем местами две ячейки с координатами первого и второго клика

Считаем количество очков после перемены мест

Если количество очков не изменилось,

то меняем выбранные ячейки обратно и снимаем выделение с первой ячейки

иначе сохраняем перемену мест и снимаем выделение со второй ячейки

Подсчитываем все очки

Конец

*Метод Swap*

// перестановка местами двух элементов игрового поля

Начало

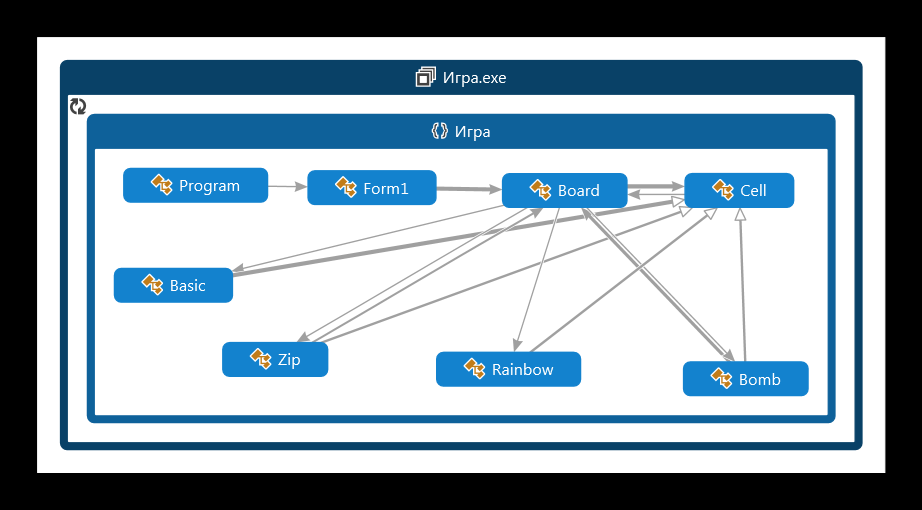
Сохраняем в буфер первую ячейку

Меняем первую ячейку на вторую

Меняем вторую ячейку на первую (из буфера)

Конец

1. **UML-диаграммы классов**



1. **Сведения о реализации**

*Информация о языке*

Программная реализация выполнена на языке C#

Система программирования: Microsoft Visual Studio 2013

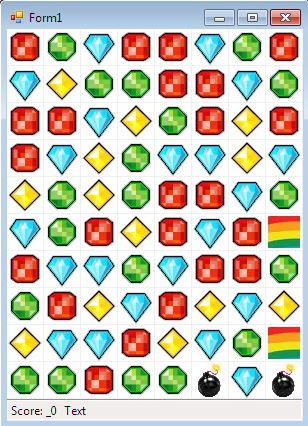
*Системные требования*

* 1 Мб свободного места на жестком диске;
* 64 Мб свободной оперативной памяти;
* Процессор 32-разрядный (x86) / 64-разрядный (x64).
* Операционная система Microsoft Windows 7 (или выше);
* .NET Framework 4.5 (или выше);

1. **Руководство пользователя**

Сначала пользователь открывает папку с приложением. Затем запускает файл «Игра».

На экране появляется следующее окно:



- в нижней строчке пишется количество очков

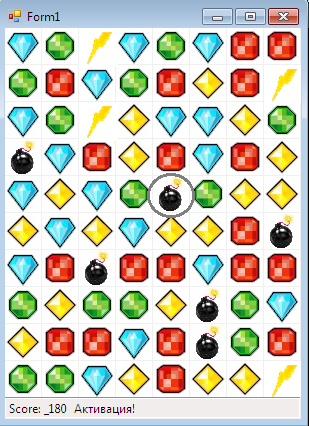
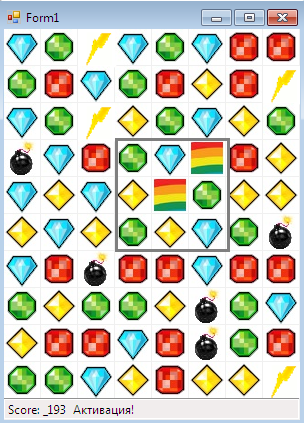
- при составлении цепочки из 3-х и более элементов, эта строчка удаляется

- при нажатии на молнию, элементы перемешиваются

- при нажатии на бомбу, происходит удаление квадрата размером 3х3

1. **Тесты**
2. *Работа программы при нажатии на бомбу*

*Ожидаемый результат: произойдет удаление квадрата размером 3х3*

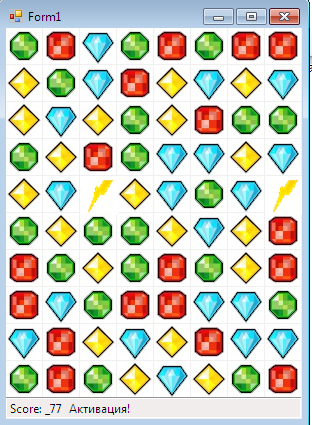
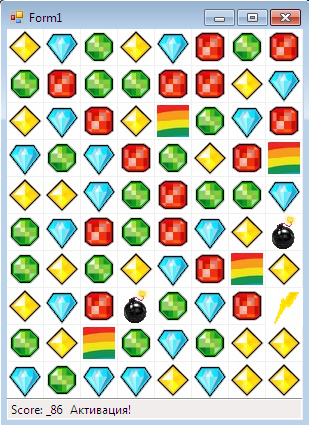


**==>**

*Ожидаемый результат совпал с действительным.*

1. *Работа программы при нажатии на молнию.*

*Ожидаемый результат: все элементы перемешаются*

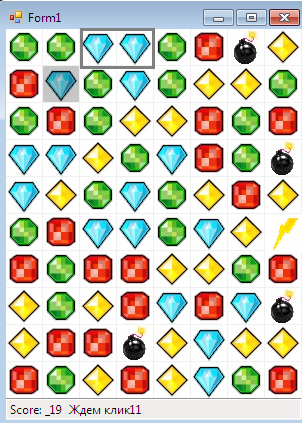
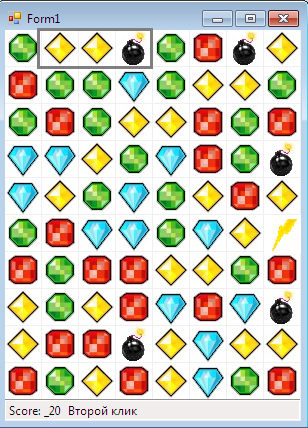


**==>**

*Ожидаемый результат совпал с действительным.*

1. *Работа программы при составлении цепочки из 3-х элементов*

*Ожидаемый результат: вся цепочка удалится*

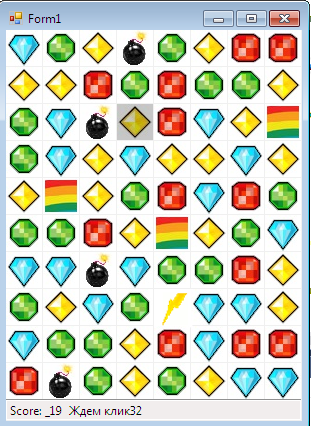
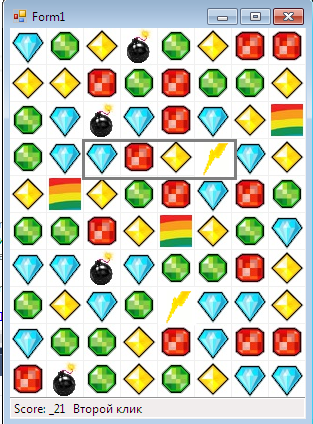


**==>**

*Ожидаемый результат совпал с действительным.*

1. *Работа программы при составлении цепочки из 4-х элементов*

*Ожидаемый результат: вся цепочка удалится*



**==>**

**5. Листинги**

**Класс Program**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using System.Windows.Forms;

using System.Drawing;

namespace Игра

{

static class Program

{

/// <summary>

/// Главная точка входа для приложения.

/// </summary>

[STAThread]

static void Main()

{

Application.EnableVisualStyles();

Application.SetCompatibleTextRenderingDefault(false);

Application.Run(new Form1());

}

}

}

**Класс Form1**

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 Игра

{

public partial class Form1 : Form

{

//public Game newGame; // Объект - Игра

public Board newBoard;

int Rows = 10, Columns = 8, Cell\_Width = Properties.Resources.blue.Width, Cell\_Height = Properties.Resources.blue.Height;

public bool flag = false; // Флаг состояния кликов (False - не было первого клика, True - был первый клик)

int posX = 0, posY = 0;

int FposX = 0, FposY = 0; // Координаты клика

/// <summary>

/// Конструктор - инициализация формы

/// </summary>

public Form1()

{

InitializeComponent();

}

/// <summary>

/// Метод при загрузке формы

/// </summary>

/// <param name="e">Аргументы события</param>

/// <param name="sender">Объект обращения</param>

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

{

newBoard = new Board(Columns, Rows, Columns \* Cell\_Width, Rows \* Cell\_Height + this.statusStrip1.Height); // создаем новую игру

newBoard.Generate();

do // доводим игровое поле до состояния готовности путем обнуления очков на игровом поле

{

newBoard.Score = 0;

newBoard.Scoring();

} while (newBoard.Score != 0);

this.ClientSize = new System.Drawing.Size(newBoard.Width, newBoard.Height); // корректируем размеры формы

}

/// <summary>

/// Отрисовка формы

/// </summary>

/// <param name="e">Событие - рисование</param>

/// <param name="sender">Объект обращения</param>

private void Form1\_Paint(object sender, PaintEventArgs e)

{

this.StripScore.Text = "Score: \_" + Convert.ToString(newBoard.Score); // Выводит кол-во очков в поле Score

newBoard.Draw(e); // Рисует игровое поле - обращение к методу Draw из класса Board

}

/// <summary>

/// Обработка клика мыши

/// </summary>

/// <param name="e">Событие - клик мыши</param>

/// <param name="sender">Объект обращения</param>

private void Form1\_MouseDown(object sender, MouseEventArgs e)

{

if(!flag) // проверяем состояние флага

{

if ((e.X < newBoard.Width) && (e.Y < newBoard.Height)) // защита от кликов вне игрового поля

{

posX = (int)(e.X / Cell\_Width); // находит номер ячейки матрицы

posY = (int)(e.Y / Cell\_Height);

if (flag = newBoard.FirstClick(posX, posY)) // проверяем характер первого клика - True - ожидание 2 клика, False - была активация

{

FposX = posX; // сохраняем коодинаты первого клика

FposY = posY;

this.Score2.Text = "Ждем клик" + Convert.ToString(FposX) + Convert.ToString(FposY); // выводим в поле Score2 текст

this.Refresh(); // обновляем форму

}

else

{

this.Score2.Text = "Активация!"; // выводим в поле Score2 текст

this.Refresh(); // обновляем форму

}

}

}

else // если находимся в состоянии ожидании второго клика (flag = True)

{

this.Score2.Text = "Второй клик"; // выводим в поле Score2 текст

if ((e.X < newBoard.Width) && (e.Y < newBoard.Height)) // защита от клика вне границ игрового поля

{

posX = (int)(e.X / Cell\_Width); // получает номер столбца и строки ячейки, по которой кликнули

posY = (int)(e.Y / Cell\_Height);

newBoard.SecondClick(FposX, FposY, posX, posY);

this.Refresh(); // обновляем форму

}

flag = false; // снимаем флаг (False - не было первого клика)

}

}

}

}

**Класс Board**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace Игра

{

/// <summary>

/// Класс Board - создает игровое поле

/// </summary>

public class Board

{

private readonly int Columns; // Кол-во столбцов игрового поля

private readonly int Rows; // Кол-во строк игрового поля

private readonly int Board\_Height;

private readonly int Board\_Width;

public int Size\_Columns

{

get

{

return Columns;

}

}

public int Size\_Rows

{

get

{

return Rows;

}

}

public int Width

{

get

{

return Board\_Width;

}

}

public int Height

{

get

{

return Board\_Height;

}

}

public Cell[,] Matrix; // Матрица объектов класса Cell

public int Score; // Количество набранных очков

Random rnd = new Random(); // Инициализируем генератор случайных чисел

int rand = 0; // Счетчик итераций генератора случайных чисел

/// <summary>

/// Конструктор класса

/// </summary>

/// <param name="Columns"></param>

/// <param name="Rows"></param>

public Board(int Columns, int Rows, int Width, int Height)

{

this.Columns = Columns;

this.Rows = Rows;

this.Board\_Width = Width;

this.Board\_Height = Height;

}

/// <summary>

/// Генерирует элементы игрового поля

/// </summary>

public void Generate()

{

Matrix = new Cell[Size\_Rows, Size\_Columns]; // создает матрицу размером SizeN x SizeM из объектов класса Cell

for (int i = 0; i < Size\_Rows; i++) // заполняем матрицу объектами класса Cell

for (int j = 0; j < Size\_Columns; j++)

{

Matrix[i, j] = RandElement(); // ячейке матрицы присваиваем случайный объект класса Cell

}

}

/// <summary>

/// Рисует игровое поле

/// </summary>

/// <param name="e">Событие - рисование</param>

public void Draw(PaintEventArgs e)

{

for (int i = 0; i < this.Size\_Rows; i++) // рисуем на форме объекты матрицы

for (int j = 0; j < this.Size\_Columns; j++)

e.Graphics.DrawImage(this.Matrix[i, j].ImgSource, j \* Properties.Resources.blue.Width, i \* Properties.Resources.blue.Height); // рисуем на форме рисунок элемента

}

/// <summary>

/// Подсчитывает количество набранных очков

/// </summary>

public void Scoring()

{

int k = 1; // счетчик доп. ячеек

int StScore = Score; // запоминает текущее кол-во очков

for (int i = 0; i < Size\_Rows; i++) // подсчет очков по горизонтали

{

for (int j = 1; j < Size\_Columns - 1; j++) // от первого до предпоследнего, т.к. у них нет соседа слева (справа)

{

// проверяет соседей справа и слева от текущ. ячейки на одинаковость (или на радужный квадрат)

if (((Matrix[i, j - 1].ObjectType == Matrix[i, j].ObjectType) || (Matrix[i, j - 1] is Rainbow))

&& ((Matrix[i, j + 1].ObjectType == Matrix[i, j].ObjectType) || (Matrix[i, j + 1] is Rainbow)))

{

k = 1;

// проверяем ещё соседей справа

while ((j + 1 + k < Size\_Columns) && ((Matrix[i, j + 1 + k].ObjectType == Matrix[i, j].ObjectType) || (Matrix[i, j + 1 + k] is Rainbow)))

k++;

Matrix[i, j - 1] = RandElement(); // заменяем найденные одинаковые ячейки (соседей)

Matrix[i, j] = RandElement();

Matrix[i, j + 1] = RandElement();

if (k != 1) // если были найдены доп. соседи справа

{

for (int l = 1; l < k; l++)

Matrix[i, j + 1 + l] = RandElement(); // меняем доп. соседи на новые

}

Score++; // увеличиваем счетчик очков на 1

}

}

}

if (Score == StScore)

{

for (int i = 1; i < Size\_Rows - 1; i++) // подсчет очков по вертикали

{

for (int j = 0; j < Size\_Columns; j++)

{

// проверяет соседей сверху и снизу от текущ. ячейки на одинаковость (или на радужный квадрат)

if (((Matrix[i - 1, j].ObjectType == Matrix[i, j].ObjectType) || (Matrix[i - 1, j] is Rainbow))

&& ((Matrix[i + 1, j].ObjectType == Matrix[i, j].ObjectType) || (Matrix[i + 1, j] is Rainbow)))

{

// проверяем ещё соседей снизу

k = 1;

while ((i + 1 + k < Size\_Rows) && ((Matrix[i + 1 + k, j].ObjectType == Matrix[i, j].ObjectType) || (Matrix[i + 1 + k, j] is Rainbow)))

k++;

Matrix[i - 1, j] = RandElement(); // заменяем найденные одинаковые ячейки (соседей)

Matrix[i, j] = RandElement();

Matrix[i + 1, j] = RandElement();

if (k != 1) // если были найдены доп. соседи снизу

{

for (int l = 1; l < k; l++)

Matrix[i + 1 + l, j] = RandElement(); // меняем доп. соседи на новые

}

Score++; // увеличиваем счетчик очков на 1

}

}

}

}

}

/// <summary>

/// Анализирует клик по первому элементу игрового поля (квадратику)

/// </summary>

/// <param name="e">Событие - клик мыши</param>

/// <returns>True - не было активации, False - была активация</returns>

public bool FirstClick(int posX, int posY)

{

// posX, posY;

int StScore; // запоминает текущее кол-во очков

if ((posX < Width) && (posY < Height))

{

if (!Matrix[posY, posX].Activation(posX, posY, this)) // проверка на возможность активации при однократном клике

{

Matrix[posY, posX].SelectElement(); // выделяем данную ячейку (затемняем)

return true;

}

else

{

do // подсчитываем очки после активации

{

StScore = Score;

Scoring();

} while (StScore != Score); // до тех пор, пока не останется неподсчитанных очков

}

}

return false;

}

/// <summary>

/// Анализирует клик по второму элементу игрового поля (квадратику)

/// </summary>

/// <param name="FposX">Номер столбца ячейки из первого клика</param>

/// <param name="FposY">Номер строки ячейки из первого клика</param>

/// <param name="e">Событие - клик мыши</param>

public void SecondClick(int FposX, int FposY, int posX, int posY)

{

// проверяет корректность второго клика

if ((Matrix[posY, posX] is Basic)

&&

(((Math.Abs(posX - FposX) == 1) && (Math.Abs(posY - FposY) == 0)) || ((Math.Abs(posY - FposY) == 1) && (Math.Abs(posX - FposX) == 0))))

{

Chain(FposX, FposY, posX, posY); // проверяем возможность образования цепочки в реультате перемены мест

}

Matrix[FposY, FposX].CreateElement(); // снимаем выделение с ячейки из первого клика

}

/// <summary>

///Уничтожает цепочки из 3 и более элементов одинаковых по цвету (в том числе Радужного квадрата) по столбцам или строкам игрового поля

/// </summary>

/// <param name="FposX">Номер столбца ячейки из первого клика</param>

/// <param name="FposY">Номер строки ячейки из первого клика</param>

/// <param name="SposX">Номер столбца ячейки из второго клика</param>

/// <param name="SposY">Номер строки ячейки из второго клика</param>

public void Chain(int FposX, int FposY, int SposX, int SposY)

{

int ChScore = Score; // запоминаем кол-во очков на текущий момент

Swap(FposX, FposY, SposX, SposY); // меняем местами 2 ячейки с координатами из вх. аргументов

Scoring(); // считаем кол-во очков на игровом поле

if (Score == ChScore) // если кол-во очков не изменилось

{

Swap(SposX, SposY, FposX, FposY); // меняем 2 ячейки местами обратно

Matrix[FposY, FposX].CreateElement(); // снимаем выделение с первой ячейки

}

else // если кол-во очков изменилось, сохраняем перемену мест

{

Matrix[SposY, SposX].CreateElement(); // снимаем выделение со второй ячейки

do // производим подсчет очков до тех пор, пока не подсчитаем все

{

ChScore = Score;

Scoring();

} while (ChScore != Score);

}

}

/// <summary>

/// Перестановка местами двух элементов игрового поля

/// </summary>

/// <param name="FposX">Номер столбца ячейки из первого клика</param>

/// <param name="FposY">Номер строки ячейки из первого клика</param>

/// <param name="posX">Номер столбца ячейки из второго клика</param>

/// <param name="posY">Номер строки ячейки из второго клика</param>

public void Swap(int FposX, int FposY, int posX, int posY)

{

Cell SwCell = Matrix[FposY, FposX]; // сохраняем в буфер первую ячейку

Matrix[FposY, FposX] = Matrix[posY, posX]; // меняем первую ячейку на вторую

Matrix[posY, posX] = SwCell; // меняем вторую ячейку на первую (из буфера)

}

/// <summary>

/// Генерирует случайный элемент игрового поля

/// </summary>

/// <param name="prand">Увеличивает счетчик итераций генератора случайных чисел на значение prand</param>

/// <returns>Сгенерированный объект класса Cell (ячейку)</returns>

public Cell RandElement()

{

int op;

rand++; // увеличиваем счетчик итераций на 1

if (rand <= 10) // пока счетчик итераций < 10

op = rnd.Next(1); // генерируем только базовые элементы (0, 1, 2, 3)

else // счетчик достигает 10

{

op = rnd.Next(7); // генерируем любой случайный элемент

rand = 0; // обнуляем счетчик итераций

}

switch (op)

{

case 0:

case 1:

case 2:

case 3: return new Basic(op);

case 4: return new Bomb();

case 5: return new Rainbow();

default: return new Zip();

}

}

}

}

**Класс Cell**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Drawing;

namespace Игра

{

/// <summary>

/// Класс Cell - ячейка

/// </summary>

abstract public class Cell

{

public Image ImgSource; // Хранит изображение объекта

public int ObjectType;

public Cell() { }

/// <summary>

/// Создает изображение элемента игрового поля

/// </summary>

public abstract void CreateElement();

/// <summary>

/// Выделяет изображение элемента игрового поля

/// </summary>

public abstract void SelectElement();

/// <summary>

/// Активация

/// </summary>

public virtual bool Activation(int posX, int posY, Board myBoard) { return false; }

}

}

**Класс Basic**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Игра

{

// 0 - «Базовый»,

public class Basic : Cell

{

public Basic(int op)

{

ObjectType = op;

switch (ObjectType)

{

case 0: ImgSource = Properties.Resources.yellow; break;

case 1: ImgSource = Properties.Resources.red; break;

case 2: ImgSource = Properties.Resources.blue; break;

case 3: ImgSource = Properties.Resources.green; break;

}

}

public override void CreateElement()

{

switch (ObjectType)

{

case 0: ImgSource = Properties.Resources.yellow; break;

case 1: ImgSource = Properties.Resources.red; break;

case 2: ImgSource = Properties.Resources.blue; break;

case 3: ImgSource = Properties.Resources.green; break;

}

}

public override void SelectElement()

{

switch (ObjectType)

{

case 0: ImgSource = Properties.Resources.yellow\_1; break;

case 1: ImgSource = Properties.Resources.red\_1; break;

case 2: ImgSource = Properties.Resources.blue\_1; break;

case 3: ImgSource = Properties.Resources.green\_1; break;

}

}

}

}

**Класс Bomb**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Игра

{

// 4 - «Бомба»,

public class Bomb : Cell

{

public Bomb()

{

ImgSource = Properties.Resources.bomb;

}

/// <summary>

/// Создает изображение элемента игрового поля

/// </summary>

public override void CreateElement()

{}

/// <summary>

/// Выделяет изображение элемента игрового поля

/// </summary>

public override void SelectElement()

{}

public override bool Activation(int posX, int posY, Board myBoard)

{

int lb = posX - 1, rb = posX + 1, tb = posY - 1, bb = posY + 1; // запоминаем координаты границ вокруг ячейки

if (posX == 0) lb = posX; // корректируем границ, чтобы избежать выхода за границы поля

if (posX == myBoard.Size\_Columns - 1) rb = posX;

if (posY == 0) tb = posY;

if (posY == myBoard.Size\_Rows - 1) bb = posY;

for (int i = tb; i <= bb; i++) // проходим по всем ячейкам вокруг ячейки

for (int j = lb; j <= rb; j++)

{

myBoard.Score++; // увелииваем счетчик очков на 1

myBoard.Matrix[i, j] = myBoard.RandElement(); // заменяем на новый

}

return true;

}

}

}

**Класс Zip**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Игра

{

// 6 - «Молния»

public class Zip : Cell

{

public Zip()

{

ImgSource = Properties.Resources.zip;

}

/// <summary>

/// Создает изображение элемента игрового поля

/// </summary>

public override void CreateElement() { }

/// <summary>

/// Выделяет изображение элемента игрового поля

/// </summary>

public override void SelectElement() { }

public override bool Activation(int posX, int posY, Board myBoard)

{

myBoard.Generate(); // генерируем новое игровое поле

return true;

}

}

}

**Класс Rainbow**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Игра

{

// 5 - «Радужный квадрат»,

public class Rainbow : Cell

{

public Rainbow()

{

ImgSource = Properties.Resources.rainbow;

}

/// <summary>

/// Создает изображение элемента игрового поля

/// </summary>

public override void CreateElement()

{ }

/// <summary>

/// Выделяет изображение элемента игрового поля

/// </summary>

public override void SelectElement()

{ }

}

}