**Озеров Ростислав 4-ИП**

**Отчёт по практическому заданию**

Необходимо реализовать игру “Жизнь”.

**В проекте задачи содержится файлы:**

**Game.cs**

Файл содержит основной код со всеми функциями, необходимыми для реализации данного задания.

**Глобальные переменные**

Grafica – Содержит экземпляр класса Graphics для рисования

CellSize – Содержит размер клетки

RedCells – Массив красных клеток

BlueCells – Массив синих клеток

Rows – Содержит кол-во строк

Columns – Содержит кол-во столбцов

ArraysForCheckEquals - Список для хранения предыдущих состояний

**Описание конструктора класса:**

**Алгоритм:**

После инициализации компонентов программы запускается функция расчёта стартовых необходимых переменных.

**Описание конструктора формы:**

У формы были изменены свойства: text, name и size.

В конструкторе были созданы следующие объекты:

4 Button – Переименованные в StartButton, PauseButton, CleanButton, ExitButton. К каждой привязано событие Click и изменено св-во text.

PictureBox – Переименованный в GamePlace. Изменено свойство size. Привязано событие MouseClick.

1 ComboBox- sizeBox.

**Описание основных методов и функций:**

1. **Private void StartGame()**

Функция запускает таймер, благодаря чему начинается генерация поколений.

**Алгоритм:**

Если таймер включен выйти из функции, иначе запустить таймер.

1. **Private int CountNeighbors(int x, int y, bool[,] a)**

Функция подсчитывает кол-во соседей клетки.

**Переменные:**

CountNeighbors – Хранит количество соседей

NeighboringCol – Хранит номер столбца обрабатываемой клетки

NeighboringRow – Хранит номер строки обрабатываемой клетки

Samoproverka – Содержит логическое значение, отражающее не проверяется ли сама клетка.

IsAlive – Содержит логическое значение, отражающее жива ли клетка

**Алгоритм:**

Создаём переменную для хранения кол-ва соседей.

В двух циклах перебираем соседей.

Проверяем на самопроверку.

Проверяем жива ли клетка.

Если клетка жива и это не самопроверка, то прибавить соседа.

Затем вернуть итоговое кол-во соседей.

1. **Private void NextGeneration()**

Функция рассчитывает следующее поколение.

**Переменные:**

NewRedCells – Массив следующего поколения красных

NewBlueCells – Массив следующего поколения синих

NeighborsRed – Содержит кол-во красных соседей

IsAliveRed – Содержит значение, отражающее красная ли живая клетка

NeighborsBlue – Содержит кол-во синих соседей

IsAliveBlue – Содержит значение, отражающее синяя ли живая клетка

**Алгоритм:**

Очищаем поле.

Создаём массивы для хранения синих и красных клеток следующего поколения.

В двух циклах проходимся по всем клеткам.

Считаем кол-во красных соседей клетки.

Проверяем красная ли клетка.

Считаем кол-во синих соседей клетки.

Проверяем синяя ли клетка.

Если синяя клетка появляется на красной, красная остаётся.

Если красная клетка появляется на синей, синяя остаётся.

Если синяя и красная на пустой, синяя остаётся.

Если клетка пустая и 3 соседа красных, становится красной.

Иначе если красная жива и соседей больше 3 или меньше 2, то умирает.

Иначе клетка сохранит цвет.

Если пустая клетка имеет 3 синих соседа, становится синей.

Иначе если синяя жива и имеет больше 3 или меньше 2 соседей, то умирает.

Иначе сохраняет значение.

Если после всех вычислений клетка красная, то покрасить её на поле в красный, если синяя, то покрасить в синий.

После отработки циклов присвоить текущим массивам клеток, новые массивы и отрисовать изменения на экран.

1. **Private void TimerTick(object sender, EventArgs e)**

Каждый тик таймера рассчитывается новое поколение.

**Алгоритм:**

Запустить функцию следующего поколения.

1. **Private void PauseClick(object sender, EventArgs e)**

Инвертирует состояние таймера.

**Алгоритм:**

Включить таймер, если выключен и наоборот.

1. **Private void PrintColor(object sender, MouseEventArgs e)**

Функция реализует возможность нарисовать или стереть клетку красного или синего цвета по нажатию на клетку.

**Переменные:**

MousePosX – Содержит координату нажатия по x

MousePosY – Содержит координату нажатия по y

**Алгоритм:**

Если игра работает, то рисовать нельзя – выйти из функции.

Поместить в переменные позицию клика пользователя.

Проверяем находится ли клик в пределах поля.

Если нажата левая кнопка мыши, то при наличии красной клетки она стирается, а при отсутствии рисуется.

Если нажата правая кнопка мыши, то при наличии синей клетки она стирается, а при отсутствии рисуется.

В конце отобразить изменения на поле.

1. **ClickPositionCheck(int MousePosX, int MousePosY)**

Возвращает true, если клик находится в пределах координат игрового поля.

**Алгоритм:**

Вернуть true, если позиция по x больше 0, позиция по y больше 0, позиция по x меньше кол-ва столбцов и позиция по y меньше кол-ва рядов.

1. **Start(object sender, EventArgs e)**

При нажатии на кнопку, запустить функцию начала игры.

**Алгоритм:**

Запустить функцию начала игры.

1. **Exit(object sender, EventArgs e)**

При нажатии на кнопку, запустить функцию закрытия приложения.

**Алгоритм:**

Запустить функцию закрытия приложения.

1. **StartCalcVariable(object sender, EventArgs e)**

Функция высчитывает необходимые для программы переменные при старте.

**Алгоритм:**

Вычислить по формулам необходимые переменные.

Инициализировать массивы.

Создать сетку игры и заполнить её белым.

1. **Private void ClearColors(object sender, EventArgs e)**

Функция очищает игровое поле.

**Алгоритм:**

Остановить таймер.

Создать новое игровое поле и заполнить его белым.

В двух циклах стереть из массивов красных и синих клеток данные.

**Код программы:**

using System;

using System.Drawing;

using System.Windows.Forms;

namespace Praktika11

{

public partial class Game : Form

{

private Graphics Grafica;

private int CellSize = 10;

private bool[,] RedCells;

private bool[,] BlueCells;

private int Rows;

private int Columns;

public Game() //создание формы

{

InitializeComponent();

StartCalcVariable(new object(), new EventArgs());

}

private void StartGame() //При начале игры запустить таймер

{

if (Timer.Enabled)

{

return;

}

Timer.Start();

}

private int CountNeighbors(int x, int y, bool[,] a) //Подсчет кол-ва соседей клетки

{

int CountNeighbors = 0;

for (int i = -1; i < 2; i++) //В циклах обрабатываем всех соседей

{

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

{

int NeighboringCol = (x + i + Columns) % Columns; //Нахождение соседних столбцов

int NeighboringRow = (y + j + Rows) % Rows; //Нахождение соседних строк

bool Samoproverka = NeighboringCol == x && NeighboringRow == y; //является ли проверка соседа самопроверкой

bool IsAlive = a[NeighboringCol, NeighboringRow];

if (IsAlive && !Samoproverka) //Если клетка имеет жизнь и не самопроверка, увеличить кол-во соседей

{

CountNeighbors++;

}

}

}

return CountNeighbors;

}

private void NextGeneration() //Просчет следующего поколения

{

Grafica.Clear(Color.White); //Очистить поле

var NewRedCells = new bool[Columns, Rows];

var NewBlueCells = new bool[Columns, Rows];

for (int x = 0; x < Columns; x++) //2 цикла for для прохода по всем клеткам массивов

{

for (int y = 0; y < Rows; y++)

{

int NeighborsRed = CountNeighbors(x, y, RedCells);

bool IsAliveRed = RedCells[x, y];

int NeighborsBlue = CountNeighbors(x, y, BlueCells);

bool IsAliveBlue = BlueCells[x, y];

if (IsAliveRed && !IsAliveBlue && NeighborsBlue == 3) //если синяя клетка появляется там, где жива красная

{

NewRedCells[x, y] = true;

NewBlueCells[x, y] = false;

Grafica.FillRectangle(Brushes.Blue, x \* CellSize, y \* CellSize, CellSize - 1, CellSize - 1);

continue;

}

if (IsAliveBlue && !IsAliveRed && NeighborsRed == 3) //если красная клетка появляется там, где жива синяя

{

NewBlueCells[x, y] = true;

NewRedCells[x, y] = false;

Grafica.FillRectangle(Brushes.Red, x \* CellSize, y \* CellSize, CellSize - 1, CellSize - 1);

continue;

}

if ((!IsAliveRed && !IsAliveBlue) && (NeighborsRed == 3 && NeighborsBlue == 3)) //если синяя и красная на пустой

{

NewBlueCells[x, y] = true;

Grafica.FillRectangle(Brushes.Red, x \* CellSize, y \* CellSize, CellSize - 1, CellSize - 1);

continue;

}

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

if (!IsAliveRed && NeighborsRed == 3) //если клетка пуста и 3 соседа красных

{

NewRedCells[x, y] = true;

}

else if (IsAliveRed && (NeighborsRed < 2 || NeighborsRed > 3)) //если красная жива и рядом меньше 2 или больше 3 соседей

{

NewRedCells[x, y] = false;

}

else //Если ни одно условие не сработало, клетка остаётся такой же

{

NewRedCells[x, y] = RedCells[x, y];

}

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

if (!IsAliveBlue && NeighborsBlue == 3) //если клетка пуста и имеет 3 соседа синих

{

NewBlueCells[x, y] = true;

}

else if (IsAliveBlue && (NeighborsBlue < 2 || NeighborsBlue > 3)) //если синяя клетка жива и рядом меньше 2 или больше 3 соседей

{

NewBlueCells[x, y] = false;

}

else //Если ни одно условие не сработало, клетка остаётся такой же

{

NewBlueCells[x, y] = BlueCells[x, y];

}

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

if (NewRedCells[x, y]) //Если клетка по координатам красная, то покрасить клетку

{

Grafica.FillRectangle(Brushes.Red, x \* CellSize, y \* CellSize, CellSize - 1, CellSize - 1);

}

else if (NewBlueCells[x, y]) //Если клетка по координатам синяя, то покрасить клетку

{

Grafica.FillRectangle(Brushes.Blue, x \* CellSize, y \* CellSize, CellSize - 1, CellSize - 1);

}

}

}

RedCells = NewRedCells; //Новые поколения становятся текущими

BlueCells = NewBlueCells;

GamePlace.Refresh(); //Отрисовываются изменения

}

private void TimerTick(object sender, EventArgs e) //Каждый тик расчитывать новое поколение

{

NextGeneration();

}

private void PauseClick(object sender, EventArgs e) //Пауза

{

Timer.Enabled = !Timer.Enabled;

}

private void PrintColor(object sender, MouseEventArgs e) //Закрасить клетку

{

if (Timer.Enabled) //Если таймер включен, то рисовать нельзя

{

return;

}

int MousePosX = e.Location.X / CellSize; //координаты клика

int MousePosY = e.Location.Y / CellSize;

if (ClickPositionCheck(MousePosX, MousePosY)) //если координата клетки в пределах поля

{

if (e.Button == MouseButtons.Left) //если нажата ЛКМ

{

if (RedCells[MousePosX, MousePosY]) //если клетка уже существует - стереть

{

RedCells[MousePosX, MousePosY] = false;

Grafica.FillRectangle(Brushes.White, MousePosX \* CellSize, MousePosY \* CellSize, CellSize - 1, CellSize - 1);

}

else //если клетки ещё не существует - создать

{

RedCells[MousePosX, MousePosY] = true;

Grafica.FillRectangle(Brushes.Red, MousePosX \* CellSize, MousePosY \* CellSize, CellSize - 1, CellSize - 1);

}

}

if (e.Button == MouseButtons.Right) //если нажата ПКМ

{

if (BlueCells[MousePosX, MousePosY]) //если клетка уже существует - стереть

{

BlueCells[MousePosX, MousePosY] = false;

Grafica.FillRectangle(Brushes.White, MousePosX \* CellSize, MousePosY \* CellSize, CellSize - 1, CellSize - 1);

}

else //если клетки ещё не существует - создать

{

BlueCells[MousePosX, MousePosY] = true;

Grafica.FillRectangle(Brushes.Blue, MousePosX \* CellSize, MousePosY \* CellSize, CellSize - 1, CellSize - 1);

}

}

}

GamePlace.Refresh(); //Отобразить изменения

}

private bool ClickPositionCheck(int MousePosX, int MousePosY) //Проверка нахождение клика в границах поля

{

return MousePosX >= 0 && MousePosY >= 0 && MousePosX < Columns && MousePosY < Rows;

}

private void Start(object sender, EventArgs e) //Запустить игру при нажатии кнопки

{

StartGame();

}

private void Exit(object sender, EventArgs e) //Закрыть приложение

{

Application.Exit();

}

private void StartCalcVariable(object sender, EventArgs e)

{

Rows = GamePlace.Height / CellSize;

Columns = GamePlace.Width / CellSize;

RedCells = new bool[Columns, Rows]; //Инициализируем новый размер массивов для 2-х цветов

BlueCells = new bool[Columns, Rows];

GamePlace.Image = new Bitmap(GamePlace.Width, GamePlace.Height); //Создаём сетку игры

Grafica = Graphics.FromImage(GamePlace.Image); //Переносим сетку в изображение

Grafica.Clear(Color.White); //Заполняем графику белым цветом

}

private void ClearColors(object sender, EventArgs e)

{

Timer.Enabled = false;

GamePlace.Image = new Bitmap(GamePlace.Width, GamePlace.Height); //Создать новое поле

Grafica = Graphics.FromImage(GamePlace.Image); //Инициализировать переменную Grafica полем

Grafica.Clear(Color.White); //заполнение поля белым цветом

for (int i = 0; i < Columns; i++) //В двойном цикле пройтись по всему массиву и очистить его.

{

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

{

RedCells[i, j] = false;

BlueCells[i, j] = false;

}

}

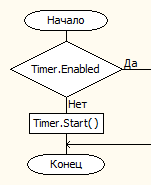
}

}

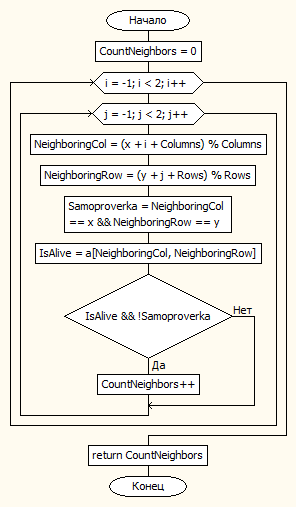
}

**Блок-схемы функций**

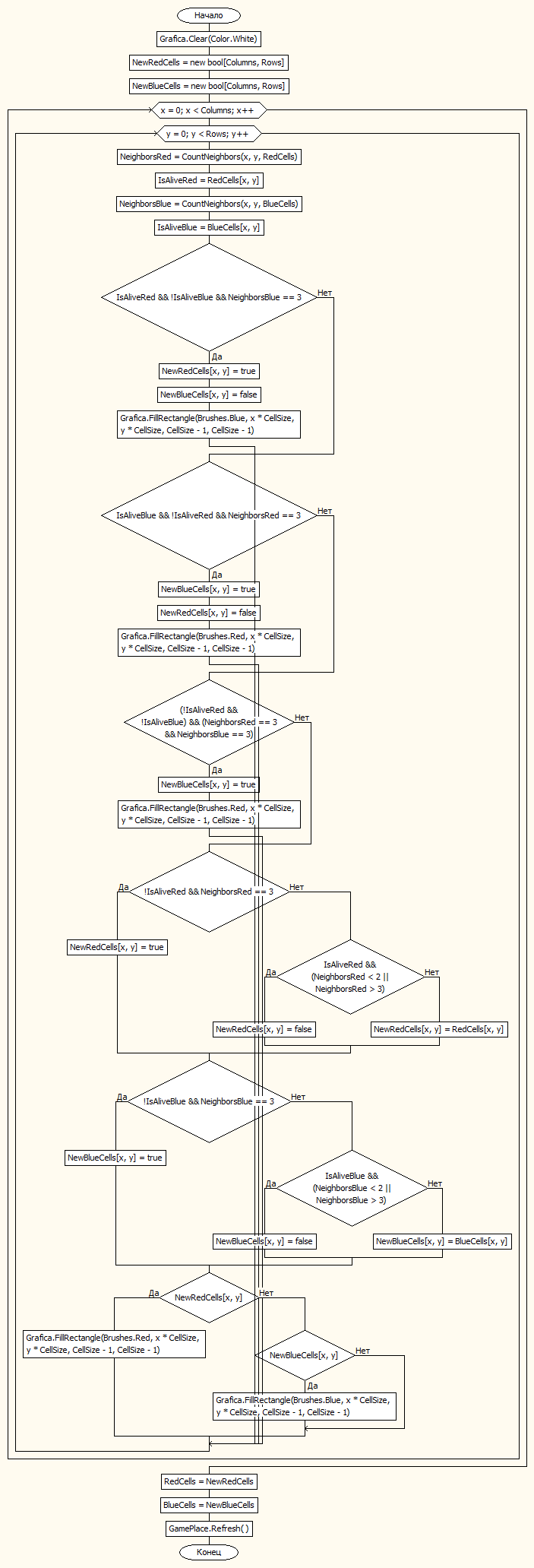
**Private void StartGame()**

****

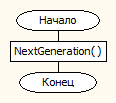
**Private int CountNeighbors(int x, int y, bool[,] a)**

****

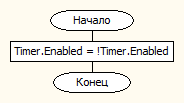
**Private void NextGeneration()**

****

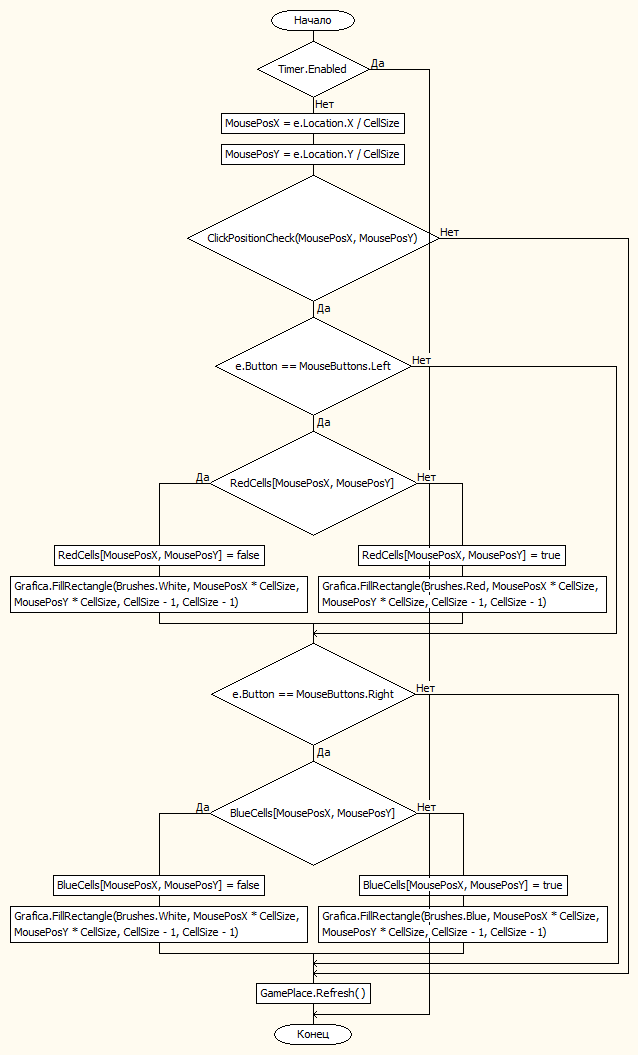
**Private void TimerTick(object sender, EventArgs e)**

****

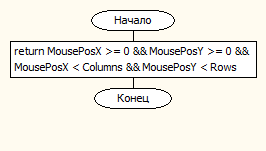
**Private void PauseClick(object sender, EventArgs e)**

****

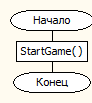
**Private void PrintColor(object sender, MouseEventArgs e)**

****

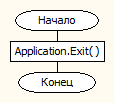
**Private bool ClickPositionCheck(int MousePosX, int MousePosY)**

****

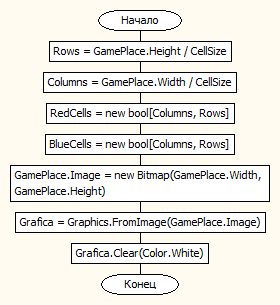
**Private void Start(object sender, EventArgs e)**

****

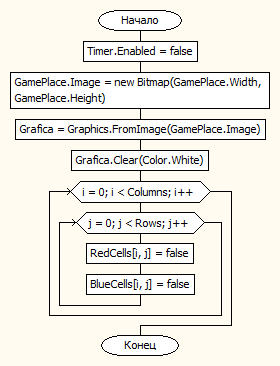
**Private void Exit(object sender, EventArgs e)**

****

**Private void StartCalcVariable(object sender, EventArgs e)**

****

**Private void ClearColors(object sender, EventArgs e)**

****