Федеральное государственное бюджетное образовательное учреждение высшего образования «Омский государственный технический университет»

Факультет информационных технологий и компьютерных систем

Кафедра «Информатика и вычислительная техника»

Отчет по расчётно-графической работе

по дисциплине Проектирование и тестирование программного обеспечения

на тему «Разработка приложения-игры «Битва площадей»»

Выполнил: ст. гр. ИВТ-183

Бабанов А.Л.

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

Проверил: ст.п.

Поздниченко Н.Н.

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

Омск 2020

Содержание

[1 Описание задания 1](#_Toc40569601)

[2 Выбранные средства выполнения 2](#_Toc40569602)

[3 Разработка алгоритмов 3](#_Toc40569603)

[3.1 Основа программы 3](#_Toc40569604)

[3.2 Загрузка контента 4](#_Toc40569605)

[3.3 Обновление 4](#_Toc40569606)

[3.4 Отрисовка 8](#_Toc40569607)

[3.5 Выгрузка контента 9](#_Toc40569608)

[4. Скриншоты выполнения работы с описанием 10](#_Toc40569609)

[5. Результаты тестирования с описанием 12](#_Toc40569610)

[Заключение 13](#_Toc40569611)

[Библиографический список 14](#_Toc40569612)

[Приложение А(код программы) 1](#_Toc40569613)

## 1 Описание задания

Задачей расчетно-графической работы было написание программы на любом объектно-ориентированном языке программирования с составлением тестов для нее. В качестве программы была выбрана игра.

Более конкретно задачу можно описать следующим образом: нужно реализовать функционал, который позволял бы играть в игру на одном ПК. Из особенностей: предусмотреть графическое меню, расписать все алгоритмы для правил игры (нужно ставить квадрат, который генерируется с помощью кидания «кубиков», только возле своих квадратов, и заставить ими все поле так, чтобы противник не смог поставить, в конечном итоге, свой квадрат), сделать управление мышью и графику для расширения мониторов FullHD (1920\*1080).

## 2 Выбранные средства выполнения

В качестве графического фреймворка был выбран Monogame – реализация фрейморка XNA от Microsoft. Данный фреймворк облегчает написание игры, и помогает написать одну версию игры для многих платформ.

В качестве IDE был выбран Visual Studio 2015 в виду того, что шаблонов и редактора контента Monogame нет под более новые версии.

Для тестов ПО был выбран фреймворк NUnit – один из самых популярных и простых в освоении.

Для разработки алгоритмов был использован метод восходящего программирования. На первом этапе разрабатываются модули самого низкого уровня. На следующем этапе к ним подключаются модули более высокого уровня и проверяется их работоспособность. На завершающем этапе проектирования разрабатывается головной модуль, отвечающий за логику работы всего программного комплекса.

## 3 Разработка алгоритмов

### ****3.1 Основа программы****

Благодаря использованию реализации фреймворка Microsoft XNA 4 Monogame, весь процесс создания игры сводится к реализации нескольких функций:

1. Инициализация объекта игры

2. Загрузка контента

3. Обновление

4. Отрисовка

5. Выгрузка контента

Упрощение состоит в том, что нужно просто реализовать эти функции. Основной цикл игры – это цикл между 3 и 4 пунктами – обновление данных (что рисовать, логическая часть), и отрисовка (собственно рисование). Более подробно алгоритм будет описан ниже, а схема отображена на рисунке 1.



Рисунок 1 – схема основного алгоритма программы

В инициализации происходит инициализация объектов, которые понадобятся в будущем. В моем случае, это объекты:

типа GridSystem – управление и хранение информацией об прямоугольниках (дальнейшем, под «квадраты» будет подразумеваться именно «любые прямоугольники»);

два объекта типа Dice – для каждой «кости» - информация о кости, отрисовка и генерация случайных чисел;

объект типа Square – квадрат, который будет курсором, когда выбирается где ставить квадрат.

И так же другие инициализации для настройки экрана – ширина, высота, полноэкранный режим.

### 3.2 Загрузка контента

На этом этапе происходит загрузка контента в переменные класса с помощью загрузчика контента. Подразумевается, что все изображения (кнопки и другие текстуры) для стартового меню, меню «как играть» и меню по концу игры. Так же и другие текстуры (поля, текста), в циклах загружаются в списки текстуры квадратов и костей, которые в дальнейшем из других частей программы можно будет получить через специально отведенные для этого Get-теры.

### 3.3 Обновление

Основа всей логики – обновление данных. Этот этап будет происходить в среднем 16 раз в секунду, чтобы отрисовывать 60 кадров в секунду. Для всех меню оно содержит чтение местоположения курсора при клике мыши и определении кнопки, и выполнения действий, связанных в этой кнопкой.

А в игре же процесс обновления происходит иначе. Стоит уточнить, что ради удобства, большинство графических элементов имеют свои функции для обновления данных, так что здесь пока рассмотрим именно логическую часть игры. После череды вспомогательных операций (определение конца игры, итерация счетчика времени отрисовки «ошибочного» квадрата, изменение координат мыши), производится проверка нажатий клавиш клавиатуры, чтобы вывести подсказки или повернуть прямоугольник. А так же, при нажатии на левую клавишу мыши, и происходит основная часть логики игры. Сначала, используя метод isItFit() в GridSystem, передав данные о квадрате, который мы ставим, проверяется, возможно ли вообще его поставить (если нельзя поставить именно здесь, то квадрат станет красным).

Отдельно стоит остановится на isItFit(). Это функция, которая возвращает -1, если поставлен квадрат не в том месте, 1 если возле чужого квадрата, и 0, если там где можно. По сути, сама эта функция определяет только не ставится ли квадрат на другие, что делается просто перебором всех ячеек массива (который так и формируется – поле, которое занимает квадрат, записывается цифрами его номера, который, если четный, то относится к синим, иначе розовым). И если квадрат вообще можно поставить, то проверяется, а на верном ли он месте, вызывается ifItOnRightPlace(). Тут уже проверяется для каждой стороны, находится ли первый квадрат на углу (левый верхний для синих, правый нижний для розовых). После, опять же, перебором, проверяется, а есть ли в границах квадрата другие квадраты такого же цвета. В положительном варианте возвращается 0 и проверка завершена. Схема алгоритма приведена на рисунках 2 и 3



**Рисунок 2 – схема алгоритма метода isItFit()**



Рисунок 3 - схема алгоритма метода **ifItOnRightPlace().**

**После того, как отрисовка анимации курсора закончилась, и место для установки квадрата подходящее (подразумевается, что квадрат можно ставить только возле своих квадратов или в углу для первого, и что нельзя ставить на другие квадраты), квадрат добавляется в «систему» GridSystem как элемент списка информации SquareInfo и как набор цифр в массиве, который представляет сетку поля в виде числовых значений. После выполняется генерация новых размеров (кидаются кубики), и меняется команда.**

**Для других обновляющих методов, в основном смысл одинаков – нужно рисовать анимацию. Это анимация курсора (загрузки) и анимация костей. По сути, просто меняются значения на отрисовку кадра, и после выполняется отрисовка в отведенном для этого методе.**

### ****3.4 Отрисовка****

Для всех меню отрисовка состоит в том, что в одном упаковщике спрайтов рисуются все текстуры кнопок (по нажатию на кнопку рисуется другой набор кнопок – с одной нажатой).

Для самого игрового процесса иначе.

В одном упаковщике спрайтов (стоит упомянуть, что это важно. Иначе сборщик мусора не будет работать, и оперативная память будет сильно забита) рисуются, опять же, все элементы, однако, не так просто, как было описано выше с меню. Да, текстуры поля и «точек старта» так же – просто Draw(). Но прочее – иначе. Как было выше сказано, чтобы программа работала с памятью более-менее адекватно, нужно использовать всем функциям, связанным с отрисовкой, один и тот же упаковщик спрайтов. Итак, с помощью метода DrawAll() у GridSystem отрисовываются все квадраты, находящиеся на поле, информация о которых «система» хранит в типизированном списке List, состоящем из SquareInfo – главной информации о расположенных квадратах. После чего отрисовываются «кости» (точнее сказать, происходит последовательно процесс обновления о анимации и отрисовка по новым данным). В случае, если рисуется анимация костей, то рисуется и курсор. Иначе рисуется курсор в виде квадрата «1-1». После отрисовывается текст подсказки.

По сути, так и создается каждый новый кадр. 60 раз в секунду отрисовывается все заново, что создает ощущение плавности анимации.

Стоит более подробно описать в целом процесс отрисовки. Все операции отрисовки (для простоты, вывод текста тоже назовем отрисовкой, т.к. и в этом случае почти все тоже самое используется) сводятся к одной функции – Draw() у объекта упаковщика спрайтов SpriteBatch. В функцию передается текстура (это может быть, как и создаваемая программным методом текстура, так и текстура, загружаемая из файла при инициализации (я использую только этот вариант)), координаты, в каком месте рисовать, цвет, масштаб, поворот, и эффекты. Для отрисовки анимации используется тоже обычная отрисовка, только с измененными координатами (в теории, можно изменять и цвет, и размер, и поворот, и эффекты). Для того, чтобы сделать интерфейс игры независящим от размеров экрана, нужно использовать масштабирование, что в виду ограниченности сроков разработки, не было сделано, однако это используется, например, в анимации курсора (чтобы не создавать лишнюю текстуру, используется просто текстура “1-1”). Изменение цвета позволяет рисовать кубики (да и что угодно), нужного цвета, используя только одну текстуру. Поворот используется при повороте фигуры (так же при повороте во всех функциях приходится пересчитывать координаты угла, смещать их, иначе будет неправильно рисовать).

### 3.5 Выгрузка контента

В теории, если проект достаточно увесистый, или ресурсы использует несколько программ, то этот пункт обязателен. В моем случае, я этот пункт пропустил, что и не мешает программе работать правильно, как и не засоряет систему.

## 4. Скриншоты выполнения работы с описанием

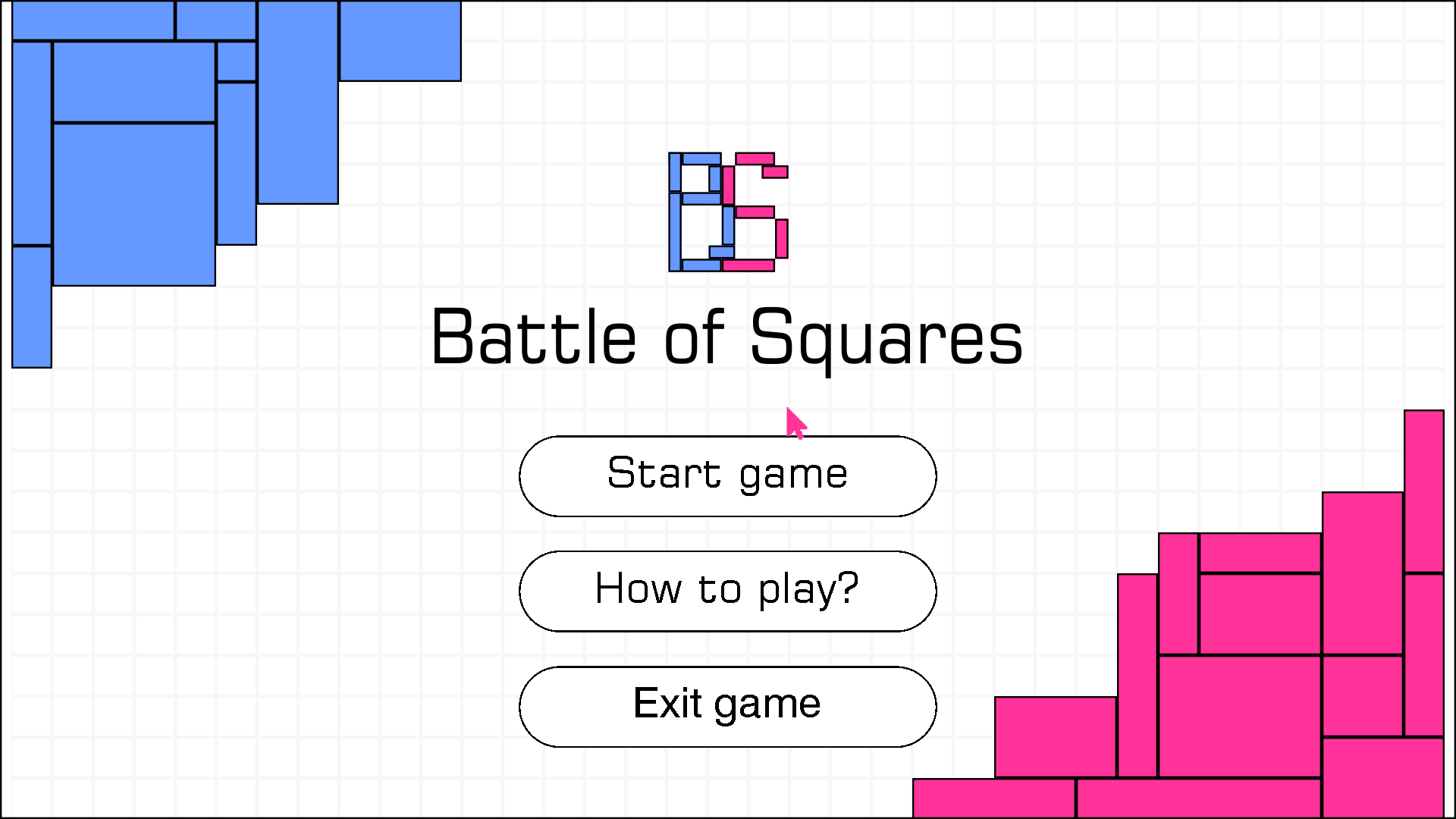


Рисунок 4 – стартовое меню игры. Start game запускает игру

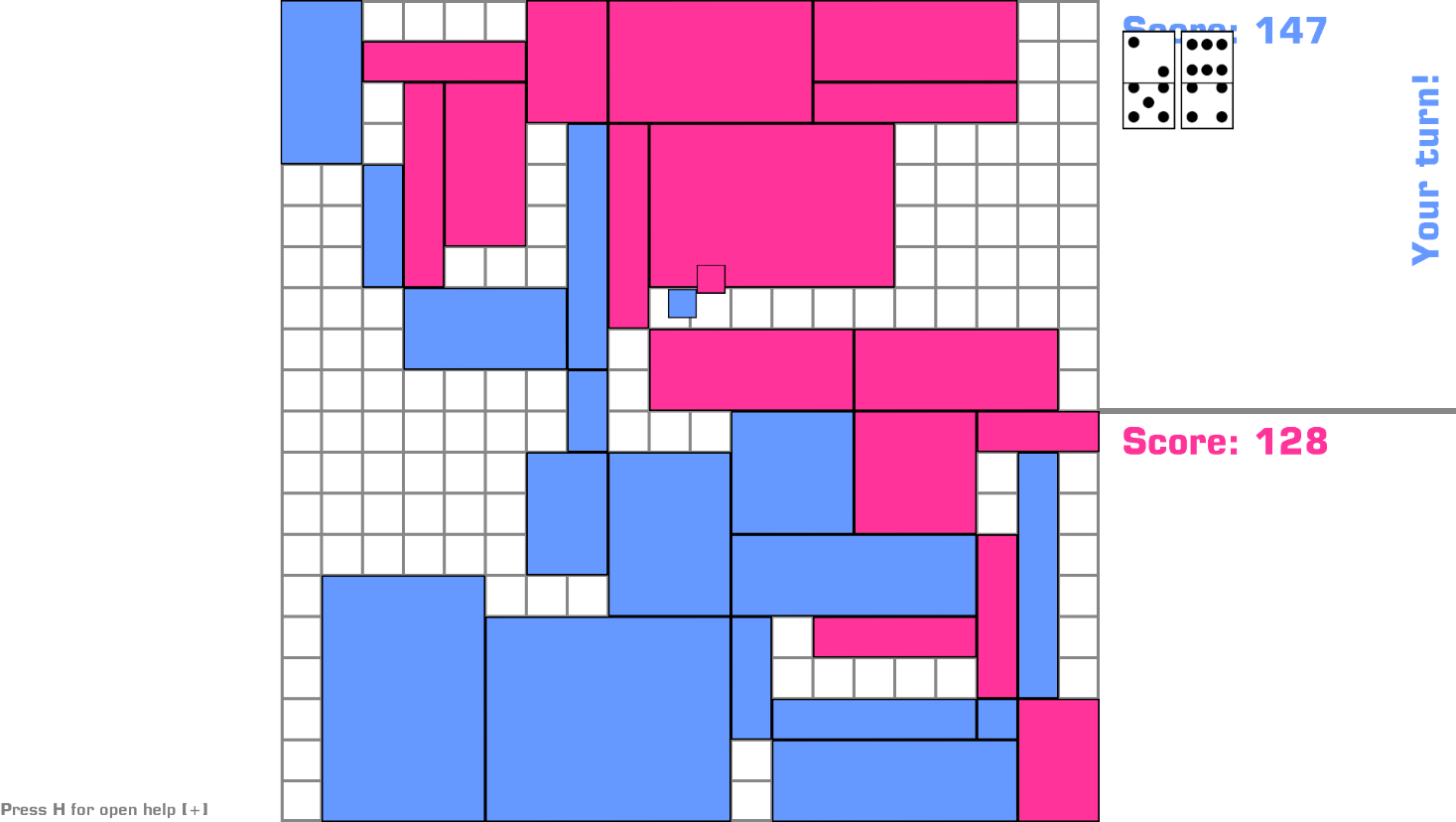


Рисунок 5 – анимации курсора и кубиков

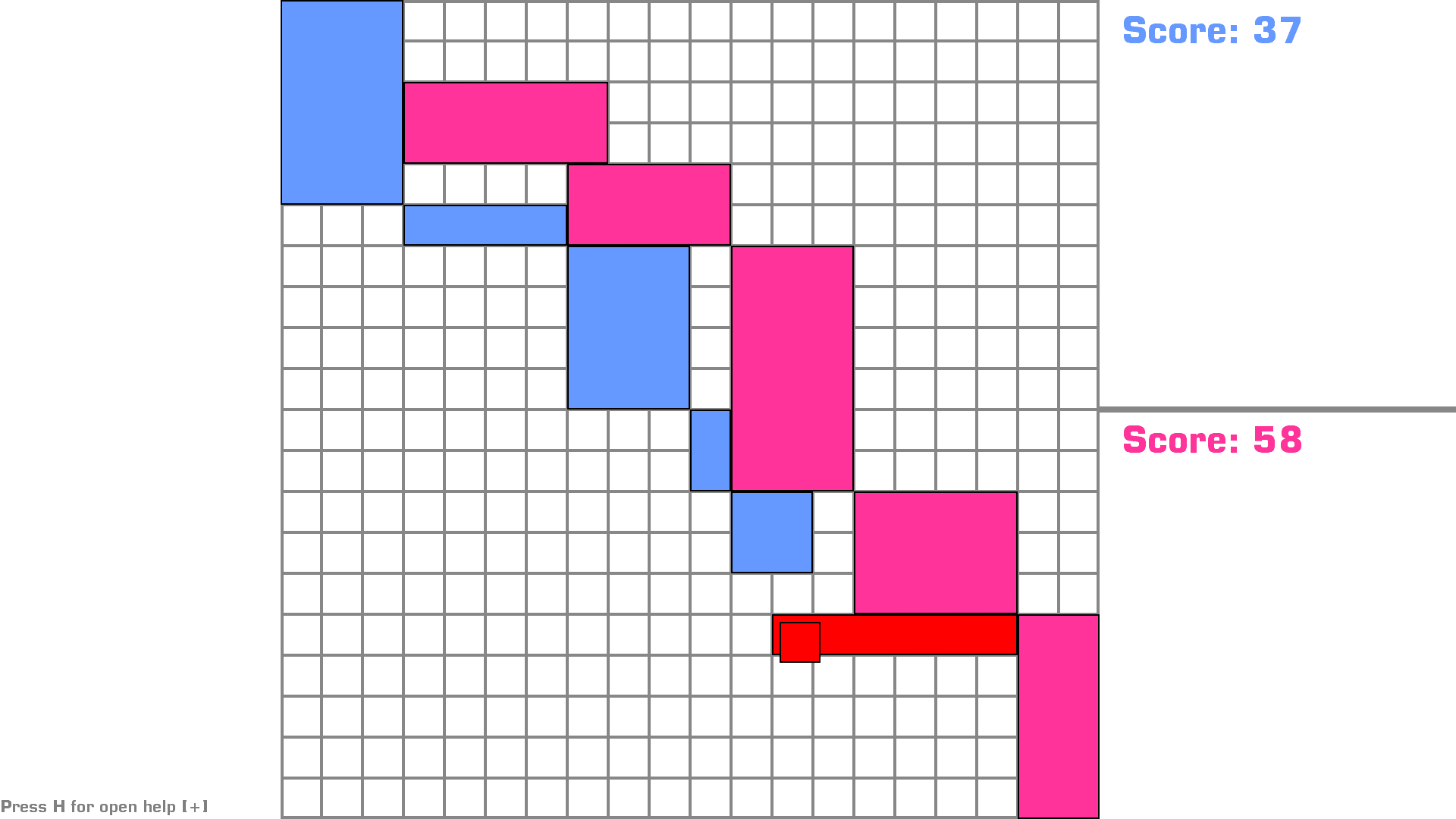


Рисунок 6 – попытка расположить прямоугольник в неправильном месте

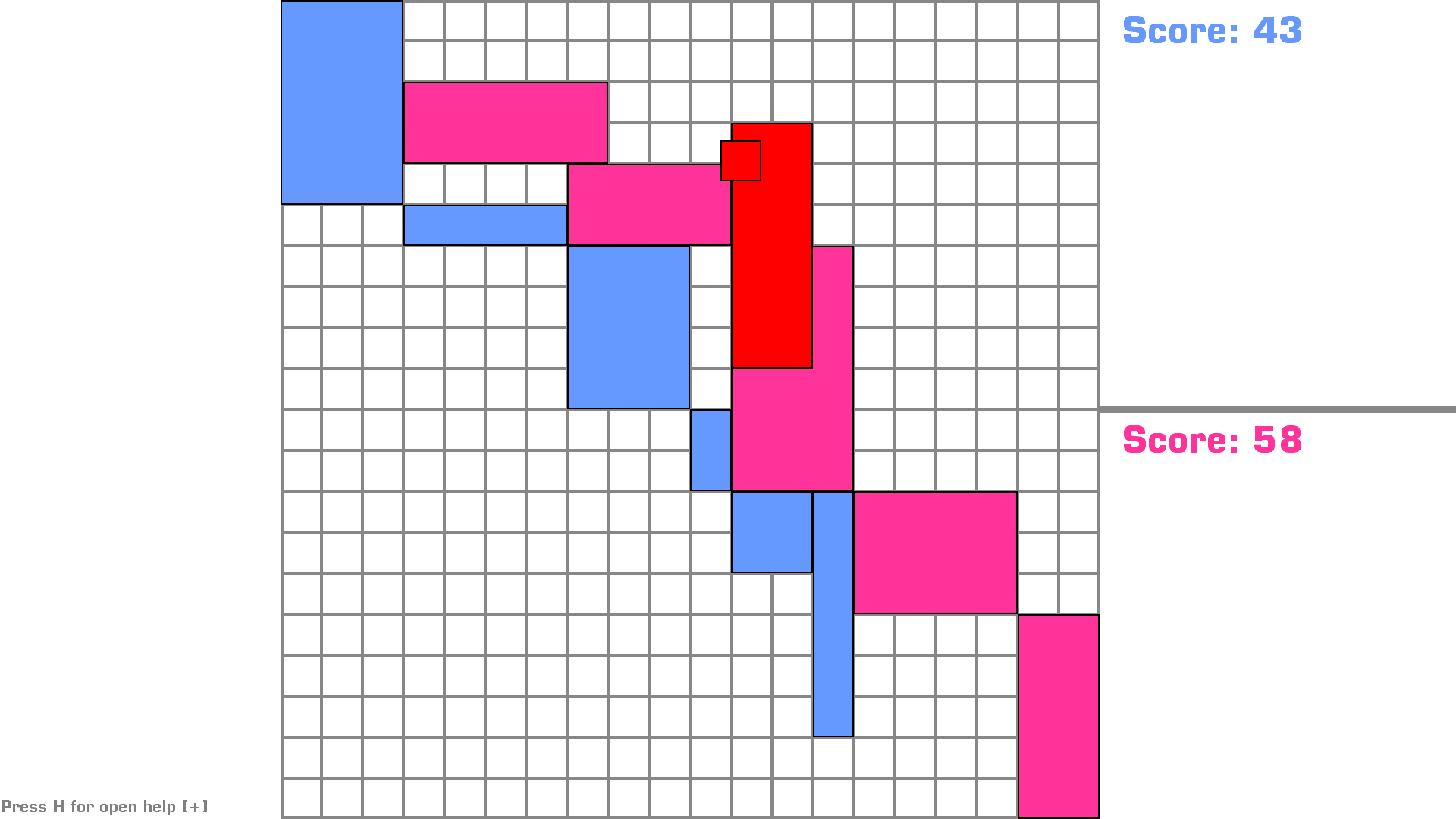


Рисунок 7 – попытка расположить прямоугольник на другой прямоугольник

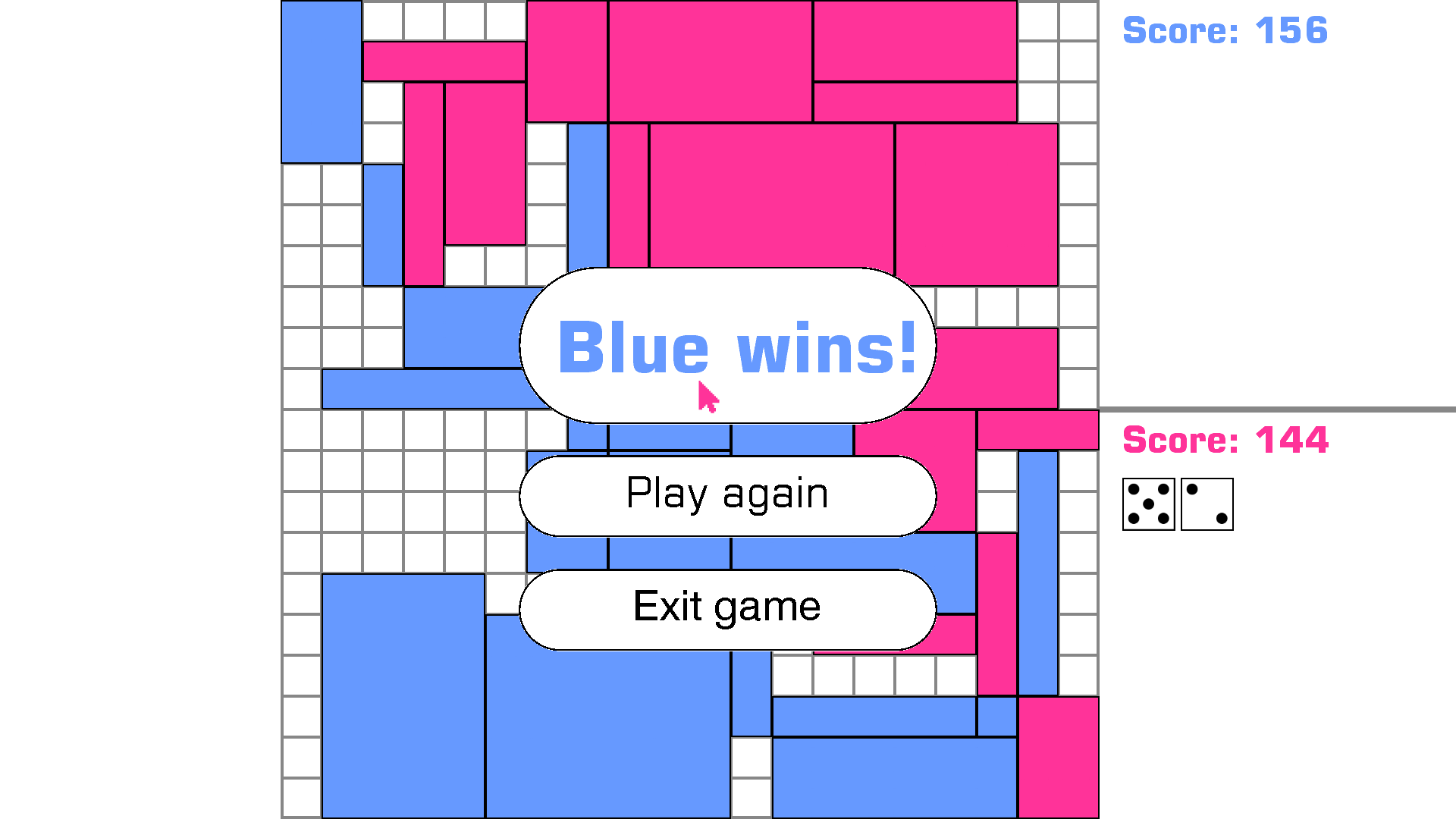


Рисунок 8 – определение конца игры – розовая команда не смогла бы поставить прямоугольник 5 на 2 на своей стороне

## 5. Результаты тестирования с описанием

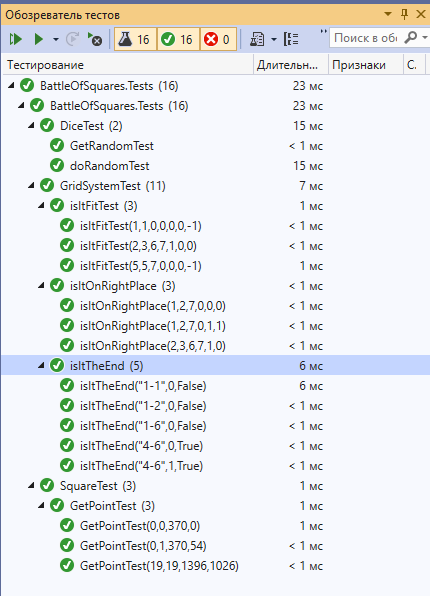


Рисунок 9 – тестирование программы с помощью фреймворка NUnit

На рисунке 9 представлены результаты тестирования. Для тестирования были выбраны основные логические модули:

1. Генерация случайных чисел (doRandomTest())

2. Получение случайного числа и связь с другим «кубиком» (GetRandomTest())

3. Проверка, можно ли разместить квадрат в целом в определенном месте (isItFitTest())

4. Проверка, можно ли разместить квадрат, исходя из расположения других квадратов того же цвета(isItOnRightPlaceTest())

5. Перевод относительных координат в абсолютные(GetPointTest())

После написания тестов и проверки программы, исправления ошибок, все тесты были пройдены успешно.

## Заключение

В данной расчетно-графической работе были реализованы все поставленные цели и задачи. В конечном итоге, были реализованы:

1. Графическое меню с управлением мышью

2. Все основные правила перенесены в игру как алгоритмы

3. Интерфейс игры рассчитан на мониторы с расширением FullHD

4. Написаны тесты, которые в конечном итоге были пройдены успешно.

## Библиографический список

1. Основы объектно-ориентированного программирования на C# : метод. указания по практике / Минобрнауки России, ОмГТУ ; [сост.: Н. Н. Поздниченко и др]. – Омск : Изд-во ОмГТУ, 2019.

2. Восходящее и нисходящее программирование: метод. указания/ ОмГТУ; сост.О.П.Шафеева.-Омск: Изд-во ОмГТУ, 2015. -1 эл.опт.диск(CD-ROM)

3. Руководство по игростроению на платформе MonoGame [Электронный ресурс]. – Режим доступа: <https://metanit.com/sharp/monogame/>, свободный

4. MonoGame [Электронный ресурс]. – Режим доступа <https://www.monogame.net/>, свободный

## Приложение А(код программы)

**Game1.cs**

using Microsoft.Xna.Framework;

using Microsoft.Xna.Framework.Graphics;

using Microsoft.Xna.Framework.Input;

using System;

using System.Collections.Generic;

namespace BattleOfSquares

{/// <summary>

/// A main Game class

/// </summary>

public class Game1 : Game

{

public static Color blueTeamColor = new Color(102, 153, 255, 255);

public static Color pinkTeamColor = new Color(255, 51, 153, 255);

GraphicsDeviceManager graphics;//главный графический менеджер

SpriteBatch spriteBatch;//упаковщик спрайтов

MouseState lastMouseState;//положение мыши для сравнения на изменения ее пололжения

Texture2D fieldTexture;//текстура поля

Texture2D fieldStartPointsTexture;//текстура поля

Texture2D startMenuTexture;//текстура стартового меню

Texture2D startMenuStButton;//текстура кнопки старт стартового меню

Texture2D startMenuStPrButton;//текстура кнопки старт стартового меню в нажатом состоянии

Texture2D startMenuCoursor;//текстура курсора стартового меню

Texture2D startMenuhTButton;//текстура кнопки howTo стартового меню

Texture2D startMenuhTPrButton;//текстура кнопки howTo стартового меню в нажатом состоянии

Texture2D startMenuExButton;//текстура кнопки выход стартового меню

Texture2D startMenuExPrButton;//текстура кнопки выход стартового меню в нажатом состоянии

Texture2D endMenuBackground;//текстура фона меню end

Texture2D endMenuPlButton;//текстура кнопки play afgain меню end

Texture2D endMenuPlPrButton;//текстура кнопки play afgain меню end в нажатом состоянии

Texture2D endMenuExButton;//текстура кнопки старт стартового меню

Texture2D endMenuExPrButton;//текстура кнопки старт стартового меню в нажатом состоянии

Texture2D howToMenu;//текстура меню how to

Texture2D howTobackButton;//текстура кнопки back меню how to

Texture2D howToBackPrButton;//текстура кнопки back меню how to в нажатом состоянии

SpriteFont controlHelp;//SprtieFont для хелпа

SpriteFont turnDisplay;//SpriteFont для your turn!

SpriteFont whoWins;//SpriteFont для определения победителя

public static SpriteFont scoreDisplay;//SpriteFont для отображение очков

bool hideHelp = false;//скрывать подсказку

int currentTimeKeyboard = 0; // сколько времени прошло, для клавиатуры

int periodKeyboard = 150; // частота обновления в миллисекундах

int currentTimeWrong = 0;//текущее время в отрисовке ошибочного квадрата

int periodWrong = 300;//время отрисовки ошибочного квадрата

int pressed = 0;//мышка нажата? 0 - нет. 1 - какая-то первая кнопка, 2 - какая-то вторая и тп

Point mousePosition;//абсолютные координаты мыши

Point positionPoint = new Point(0, 0);//относительные координаты мыши

GridSystem gridSystem;//управление сеткой и прямоугольниками на ней

Dice dice;//кости, анимация и рандом

Dice dice2;//кости, анимация и рандом, 2 кость

CousorLoadingAnimation coursorLoading = new CousorLoadingAnimation();

Square.SquareInfo placingSquare;//квадратик, который будем ставить, информация о нем

static List<Texture> squareTextures = new List<Texture>();//текстуры прямоугольников

static List<Texture> diceTextures = new List<Texture>();//текстуры костей

static List<SpriteFont> scoreSpriteFont = new List<SpriteFont>();//spriteFont для очков

public static Vector2 startPoint = new Vector2(GraphicsAdapter.DefaultAdapter.CurrentDisplayMode.Width / 2 - 590, 0);//начальная точка отрисовки поля

int pageNumber = 0;//номер страницы - определяет рисовать меню - 0, игру - 1, выход - 2

/// <summary>

/// constructor of class

/// </summary>

public Game1()

{

graphics = new GraphicsDeviceManager(this);

Content.RootDirectory = "Content";

}

/// <summary>

/// initialize variables and some display settings

/// </summary>

protected override void Initialize()

{

graphics.PreferredBackBufferWidth = GraphicsAdapter.DefaultAdapter.CurrentDisplayMode.Width;

graphics.PreferredBackBufferHeight = GraphicsAdapter.DefaultAdapter.CurrentDisplayMode.Height;

graphics.ApplyChanges();

graphics.ToggleFullScreen();

gridSystem = new GridSystem();

dice = new Dice();

dice2 = new Dice();

GraphicsDevice.Clear(Color.White);

placingSquare = new Square.SquareInfo(new Point(0, 0), "1-1", 0, 0);

base.Initialize();

}

/// <summary>

/// loading a content

/// </summary>

protected override void LoadContent()

{

fieldTexture = Content.Load<Texture2D>("field");

fieldStartPointsTexture = Content.Load<Texture2D>("fieldStartPoints");

startMenuTexture = Content.Load<Texture2D>("StartMenu\\startMenu");

startMenuStButton = Content.Load<Texture2D>("StartMenu\\button");

startMenuStPrButton = Content.Load<Texture2D>("StartMenu\\pressedButton");

startMenuCoursor = Content.Load<Texture2D>("StartMenu\\coursor");

startMenuhTButton = Content.Load<Texture2D>("StartMenu\\howTo");

startMenuhTPrButton = Content.Load<Texture2D>("StartMenu\\howToPr");

startMenuExButton = Content.Load<Texture2D>("StartMenu\\ex");

startMenuExPrButton = Content.Load<Texture2D>("StartMenu\\exPr");

endMenuBackground = Content.Load<Texture2D>("EndMenu\\background");

endMenuPlButton = Content.Load<Texture2D>("EndMenu\\pl");

endMenuPlPrButton = Content.Load<Texture2D>("EndMenu\\plPr");

endMenuExButton = Content.Load<Texture2D>("EndMenu\\ex");

endMenuExPrButton = Content.Load<Texture2D>("EndMenu\\exPr");

howToMenu = Content.Load<Texture2D>("HowToMenu\\howToMenu");

howTobackButton = Content.Load<Texture2D>("HowToMenu\\back");

howToBackPrButton = Content.Load<Texture2D>("HowToMenu\\backPr");

for (int i = 1; i <= 6; i++) //заполняем список текстур квадратиков

{

for (int j = i; j <= 6; j++)

{

string name = i.ToString() + "-" + j.ToString();

string reverseName = j.ToString() + "-" + i.ToString();

string place = "squares\\" + name;

squareTextures.Add(new Texture(Content.Load<Texture2D>(place), name + " " + reverseName)); //записываем в спискок текстуру и название прямоугольника в двух видах, чтобы успешно осуществлять поиск в GetSquareTexture

}

}

for (int i = 1; i <= 6; i++)//заполняем список костей

{

string place = "dices\\" + i.ToString();

diceTextures.Add(new Texture(Content.Load<Texture2D>(place), i));

}

scoreDisplay = Content.Load<SpriteFont>("score");

controlHelp = Content.Load<SpriteFont>("controlHelp");

turnDisplay = Content.Load<SpriteFont>("turn");

whoWins = Content.Load<SpriteFont>("whoWins");

}

/// <summary>

/// unloading a content

/// </summary>

protected override void UnloadContent()

{/\*it is actually empty\*/ }

/// <summary>

/// main update method. It's choose exactly what we need to update

/// </summary>

/// <param name="gameTime">Just a GameTime</param>

protected override void Update(GameTime gameTime)

{

currentTimeKeyboard += gameTime.ElapsedGameTime.Milliseconds;

if ( Keyboard.GetState().IsKeyDown(Keys.Escape)) Exit();

switch (pageNumber)

{

case 0:

{

UpdateMenu();

break;

}

case 1:

{

UpdateGame();

break;

}

case 2:

{

UpdateEnd();

break;

}

case 3:

{

UpdateHowTo();

break;

}

}

base.Update(gameTime);

}

/// <summary>

/// Update method for menu

/// </summary>

void UpdateMenu()

{

MouseState currentMouseState = Mouse.GetState();

if (currentMouseState.X != lastMouseState.X || currentMouseState.Y != lastMouseState.Y)//мышка сдвинулась вообще

{

mousePosition = new Point(currentMouseState.X, currentMouseState.Y);

}

if ((currentMouseState.X > 696 && currentMouseState.X < 1223) && (currentMouseState.Y > 574 && currentMouseState.Y < 683))

{

if (currentMouseState.LeftButton == ButtonState.Pressed)

{

if (pressed == 0)

{

pressed = 1;

}

}

if (currentMouseState.LeftButton == ButtonState.Released)

{

if (pressed == 1)//клавиша start была нажата

{

pressed = 0;

pageNumber = 1;

int prCount = dice.NewRoll(1, 0);

int count = dice2.NewRoll(2, prCount);

placingSquare.ChangeDices(dice.GetRandom(), dice2.GetRandom());

}

}

}

if ((currentMouseState.X > 696 && currentMouseState.X < 1223) && (currentMouseState.Y > 725 && currentMouseState.Y < 834))

{

if (currentMouseState.LeftButton == ButtonState.Pressed)

{

if (pressed == 0)

{

pressed = 2;

}

}

if (currentMouseState.LeftButton == ButtonState.Released)

{

if (pressed == 2)//клавиша how to была нажата

{

pressed = 0;

pageNumber = 3;

}

}

}

if ((currentMouseState.X > 696 && currentMouseState.X < 1223) && (currentMouseState.Y > 878 && currentMouseState.Y <978))

{

if (currentMouseState.LeftButton == ButtonState.Pressed)

{

if (pressed == 0)

{

pressed = 3;

}

}

if (currentMouseState.LeftButton == ButtonState.Released)

{

if (pressed == 3)//клавиша exit была нажата

{

pressed = 0;

Exit();

}

}

}

}

/// <summary>

/// Update method for game

/// </summary>

void UpdateGame()

{

if (placingSquare.team!=2 && gridSystem.isItTheEnd(placingSquare.name, placingSquare.team) && dice.needToDraw == false)

{

pageNumber = 2;

}

if (placingSquare.wrong)

{

currentTimeWrong += 16;

}

if (currentTimeWrong >= periodWrong)

{

currentTimeWrong = 0;

placingSquare.WrongPlace(0);

}

MouseState currentMouseState = Mouse.GetState();

if (currentMouseState.X != lastMouseState.X || currentMouseState.Y != lastMouseState.Y)//мышка сдвинулась вообще

{

if ((currentMouseState.X > startPoint.X && currentMouseState.X < startPoint.X + 1080) && (currentMouseState.Y > 0 && currentMouseState.Y < 1080))

{

positionPoint = new Point((int)((currentMouseState.X - startPoint.X) / Square.sizeOfGrid.X), (int)((currentMouseState.Y - startPoint.Y) / Square.sizeOfGrid.Y));//относительные координаты

mousePosition = new Point(currentMouseState.X - 27, currentMouseState.Y - 27);//возможно стоит переделать для удобства

}

}

lastMouseState = currentMouseState;

if (currentTimeKeyboard > periodKeyboard)

{

currentTimeKeyboard -= periodKeyboard;

KeyboardState keyboardState = Keyboard.GetState();

if (keyboardState.IsKeyDown(Keys.R))

{

placingSquare.ChangeRotate();

}

if (keyboardState.IsKeyDown(Keys.H))

{

if (hideHelp == false)

hideHelp = true;

else hideHelp = false;

}

}

if (currentMouseState.LeftButton == ButtonState.Pressed)

{

if (pressed == 0)

{

pressed = 1;

}

}

if (currentMouseState.LeftButton == ButtonState.Released)

{

if (pressed == 1)//клавиша была нажата

{

if (gridSystem.isItFit(placingSquare.name, placingSquare.rotate, positionPoint, placingSquare.team) == 0)//место подходит для установки

{

if (dice.needToDraw == false)

{

gridSystem.addSquare(placingSquare.name, placingSquare.rotate, placingSquare.team, positionPoint);//добавляем в систему

//а почему просто placingSquare не передавать то...

int prCount = dice.NewRoll(1, 0);

int count = dice2.NewRoll(2, prCount);

placingSquare.ChangeDices(dice.GetRandom(), dice2.GetRandom());

placingSquare.ChangeTeam();

placingSquare.rotate = 0;

pressed = 0;

}

}

else

{

if (placingSquare.wrong == false) placingSquare.WrongPlace(1);

pressed = 0;

}

}

}

}

/// <summary>

/// Update method for ending screen

/// </summary>

private void UpdateEnd()

{

MouseState currentMouseState = Mouse.GetState();

if (currentMouseState.X != lastMouseState.X || currentMouseState.Y != lastMouseState.Y)//мышка сдвинулась вообще

{

mousePosition = new Point(currentMouseState.X, currentMouseState.Y);

}

if ((currentMouseState.X > 696 && currentMouseState.X < 1223) && (currentMouseState.Y > 605 && currentMouseState.Y < 703))//кнопка again

{

if (currentMouseState.LeftButton == ButtonState.Pressed)

{

if (pressed == 0)

{

pressed = 1;

}

}

if (currentMouseState.LeftButton == ButtonState.Released)

{

if (pressed == 1)//клавиша была нажата

{

pressed = 0;

pageNumber = 1;

gridSystem.Clear();

int prCount = dice.NewRoll(1, 0);

int count = dice2.NewRoll(2, prCount);

placingSquare.ChangeDices(dice.GetRandom(), dice2.GetRandom());

}

}

}

if ((currentMouseState.X > 696 && currentMouseState.X < 1223) && (currentMouseState.Y > 755 && currentMouseState.Y < 853))//кнопка exit

{

if (currentMouseState.LeftButton == ButtonState.Pressed)

{

if (pressed == 0)

{

pressed = 2;

}

}

if (currentMouseState.LeftButton == ButtonState.Released)

{

if (pressed == 2)//клавиша была нажата

{

pressed = 0;

Exit();

}

}

}

}

/// <summary>

/// Update method for help screen

/// </summary>

private void UpdateHowTo()

{

MouseState currentMouseState = Mouse.GetState();

if (currentMouseState.X != lastMouseState.X || currentMouseState.Y != lastMouseState.Y)//мышка сдвинулась вообще

{

mousePosition = new Point(currentMouseState.X, currentMouseState.Y);//возможно стоит переделать для удобства

}

if ((currentMouseState.X > 696 && currentMouseState.X < 1223) && (currentMouseState.Y > 834 && currentMouseState.Y < 932))

{

if (currentMouseState.LeftButton == ButtonState.Pressed)

{

if (pressed == 0)

{

pressed = 1;

}

}

if (currentMouseState.LeftButton == ButtonState.Released)

{

if (pressed == 1)//клавиша была нажата

{

pressed = 0;

pageNumber = 0;

}

}

}

}

/// <summary>

/// main draw method. It's choose exactly what we need to draw

/// </summary>

/// <param name="gameTime">Just a GameTime</param>

protected override void Draw(GameTime gameTime)

{

GraphicsDevice.Clear(Color.White);//clears screen

switch (pageNumber)

{

case 0:

{

DrawMenu();

break;

}

case 1:

{

DrawGame();

break;

}

case 2:

{

DrawEnd();

break;

}

case 3:

{

DrawHowTo();

break;

}

}

base.Draw(gameTime);

}

/// <summary>

/// Draw method for menu

/// </summary>

private void DrawMenu()

{

using (spriteBatch = new SpriteBatch(GraphicsDevice))

{

spriteBatch.Begin(SpriteSortMode.Immediate);

spriteBatch.Draw(startMenuTexture, new Vector2(0, 0), Color.White); //фон

switch (pressed)

{

case 0:

{

spriteBatch.Draw(startMenuStButton, new Vector2(684, 574), Color.White); //кнопка start не нажата

spriteBatch.Draw(startMenuhTButton, new Vector2(684, 725), Color.White);//кнопка how to не нажата

spriteBatch.Draw(startMenuExButton, new Vector2(684, 878), Color.White);//кнопка exit не нажата

break;

}

case 1:

{

spriteBatch.Draw(startMenuStPrButton, new Vector2(696, 579), Color.White); //кнопка start нажата

spriteBatch.Draw(startMenuhTButton, new Vector2(684, 725), Color.White);//кнопка how to не нажата

spriteBatch.Draw(startMenuExButton, new Vector2(684, 878), Color.White);//кнопка exit не нажата

break;

}

case 2:

{

spriteBatch.Draw(startMenuStButton, new Vector2(684, 574), Color.White); //кнопка start не нажата

spriteBatch.Draw(startMenuhTPrButton, new Vector2(696, 730), Color.White);//кнопка how to нажата

spriteBatch.Draw(startMenuExButton, new Vector2(684, 878), Color.White);//кнопка exit не нажата

break;

}

case 3:

{

spriteBatch.Draw(startMenuStButton, new Vector2(684, 574), Color.White); //кнопка start не нажата

spriteBatch.Draw(startMenuhTButton, new Vector2(684, 725), Color.White);//кнопка how to не нажата

spriteBatch.Draw(startMenuExPrButton, new Vector2(696, 883), Color.White);//кнопка exit нажата

break;

}

}

spriteBatch.Draw(startMenuCoursor, mousePosition.ToVector2(), Color.White); //курсор

spriteBatch.End();

}

}

/// <summary>

/// Draw method for help screen

/// </summary>

private void DrawHowTo()

{

using (spriteBatch = new SpriteBatch(GraphicsDevice))

{

spriteBatch.Begin(SpriteSortMode.Immediate);

spriteBatch.Draw(howToMenu, new Vector2(0, 0), Color.White); //фон

if (pressed == 0)

{

spriteBatch.Draw(howTobackButton, new Vector2(684, 834), Color.White); //кнопка не нажата

}

else spriteBatch.Draw(howToBackPrButton, new Vector2(696, 839), Color.White); //кнопка нажата

spriteBatch.Draw(startMenuCoursor, mousePosition.ToVector2(), Color.White); //курсор

spriteBatch.End();

}

}

/// <summary>

/// Draw method for game

/// </summary>

private void DrawGame()

{

using (spriteBatch = new SpriteBatch(GraphicsDevice))

{

spriteBatch.Begin(SpriteSortMode.Immediate);

spriteBatch.Draw(fieldTexture, startPoint, null, new Color(255, 255, 255, 120), 0, Vector2.Zero, 1f, SpriteEffects.None, 0); //поле

spriteBatch.Draw(fieldStartPointsTexture, startPoint, null, Color.White, 0, Vector2.Zero, 1f, SpriteEffects.None, 0); //поле, точки старта

gridSystem.DrawAll(spriteBatch);//все квадратики

dice.Draw(spriteBatch, 1, placingSquare.team);

dice2.Draw(spriteBatch, 2, placingSquare.team);

if (dice.needToDraw == false)

{

Square sq = new Square(spriteBatch);

sq.Draw(placingSquare.name, placingSquare.rotate, placingSquare.team, positionPoint);

sq.DrawInPixel("1-1", placingSquare.rotate, placingSquare.team, mousePosition, spriteBatch);

}

else

{

coursorLoading.Draw(spriteBatch, mousePosition);

}

DrawText();//help, your turn!

spriteBatch.End();

}

}

/// <summary>

/// Draws text for DrawGame

/// </summary>

private void DrawText()

{

string helpInfo = "Press R to rotate a rectangle\n\n"

+ "Press LMB to place a rectangle\n\n"

+ "If you have 1\*1 or 6\*6 rectangle\n"

+ "Use scroll wheel to increase or\n"

+ "decrease your rectangle\n\n"

+ "Press F for pay respect";

if (hideHelp == false)

{

spriteBatch.DrawString(controlHelp, "Press H for open help [+]", new Vector2(0, 1050), Color.Gray);

}

else spriteBatch.DrawString(controlHelp, helpInfo, new Vector2(0, 840), Color.Gray);

if (placingSquare.team == 0)

{

spriteBatch.DrawString(turnDisplay, "Your turn!", new Vector2(1850, 350), blueTeamColor, 4.712f, Vector2.Zero, 1f, SpriteEffects.None, 1f);

}

else if (placingSquare.team == 1)

{

spriteBatch.DrawString(turnDisplay, "Your turn!", new Vector2(1850, 910), pinkTeamColor, 4.712f, Vector2.Zero, 1f, SpriteEffects.None, 1f);

}

}

/// <summary>

/// Draw method for end screen

/// </summary>

private void DrawEnd()

{

using (spriteBatch = new SpriteBatch(GraphicsDevice))

{

spriteBatch.Begin(SpriteSortMode.Immediate);

spriteBatch.Draw(fieldTexture, startPoint, null, new Color(255, 255, 255, 120), 0, Vector2.Zero, 1f, SpriteEffects.None, 0); //поле

spriteBatch.Draw(fieldStartPointsTexture, startPoint, null, Color.White, 0, Vector2.Zero, 1f, SpriteEffects.None, 0); //поле, точки старта

gridSystem.DrawAll(spriteBatch);//все квадратики и очки

dice.Draw(spriteBatch, 1, placingSquare.team);

dice2.Draw(spriteBatch, 2, placingSquare.team);

spriteBatch.Draw(endMenuBackground, new Vector2(684, 352), Color.White); //фон

if (placingSquare.team == 1)

{

spriteBatch.DrawString(whoWins, "Blue wins!", new Vector2(732, 400), blueTeamColor);

}

if (placingSquare.team == 0)

{

spriteBatch.DrawString(whoWins, "Pink wins!", new Vector2(732, 400), pinkTeamColor);

}

if (pressed == 0)

{

spriteBatch.Draw(endMenuPlButton, new Vector2(684, 600), Color.White); //кнопка again не нажата

spriteBatch.Draw(endMenuExButton, new Vector2(684, 750), Color.White); //кнопка exit не нажата

}

if (pressed == 1)

{

spriteBatch.Draw(endMenuPlPrButton, new Vector2(696, 605), Color.White); //кнопка again нажата

spriteBatch.Draw(endMenuExButton, new Vector2(684, 750), Color.White); //кнопка exit не нажата

}

if (pressed == 2)

{

spriteBatch.Draw(endMenuPlButton, new Vector2(684, 600), Color.White); //кнопка again не нажата

spriteBatch.Draw(endMenuExPrButton, new Vector2(696, 755), Color.White); //кнопка exit нажата

}

spriteBatch.Draw(startMenuCoursor, mousePosition.ToVector2(), Color.White); //курсор

spriteBatch.End();

}

}

/// <summary>

/// Returns Texture2D by its name

/// </summary>

/// <param name="sqName">squares name with the template "width-height"</param>

/// <returns>Texture2D by its name</returns>

public static Texture2D GetSquareTexture(string sqName)

{

for (int i = 0; i < squareTextures.Count; i++)

{

if (squareTextures[i].name.IndexOf(sqName) != -1)

{

Texture2D texture = (squareTextures[i]).texture;

return (squareTextures[i]).texture;

}

}

return null;

}

/// <summary>

/// Returns Texture2D by its number

/// </summary>

/// <param name="num">dice's number</param>

/// <returns>Texture2D by its number</returns>

public static Texture2D GetDiceTexture(int num)

{

if (num > 0 && num<7)

return (diceTextures[num - 1]).texture;

else return null;

}

}

}

**CousorLoadingAnimation.cs**

using Microsoft.Xna.Framework;

using Microsoft.Xna.Framework.Graphics;

namespace BattleOfSquares

{

/// <summary>

/// A Class for draw a coursor loading animation

/// </summary>

class CousorLoadingAnimation

{

public bool needToDraw=true;//нужно ли рисовать

static float scale = 0.7f;//масштаб

static Vector2 size = new Vector2(54, 54) \* scale;

static Vector2 addTo = new Vector2(5, 5) \* scale;//вектор, на сколько изменять координаты

Vector2 additionVector1 = Vector2.Zero;//куда добавлять для первого

Vector2 additionVector2 = Vector2.Zero;//куда добавлять для второго

int currentCorner = 1;//текущая сторона у голубого

//1-------->2

//| |

//3<\_\_\_\_\_\_\_\_4

/// <summary>

/// Main method that draws a animation

/// </summary>

/// <param name="sb">SpriteBatch</param>

/// <param name="mousePoint">Point where needs to draw</param>

public void Draw(SpriteBatch sb, Point mousePoint)

{

Vector2 placeVector = mousePoint.ToVector2();

Texture2D square = Game1.GetSquareTexture("1-1");

if (needToDraw == true)

{

switch (currentCorner)

{

case 1:

{

if (additionVector1.X <= size.X)

{

additionVector1 += new Vector2(addTo.X, 0);

additionVector2 += new Vector2(-addTo.X, 0);

}

else currentCorner++;

break;

}

case 2:

{

if (additionVector1.Y <= size.Y)

{

additionVector1 += new Vector2(0, addTo.Y);

additionVector2 += new Vector2(0, -addTo.Y);

}

else currentCorner++;

break;

}

case 3:

{

if (additionVector1.X > 0)

{

additionVector1 += new Vector2(-addTo.X, 0);

additionVector2 += new Vector2(addTo.X, 0);

}

else currentCorner++;

break;

}

case 4:

{

if (additionVector1.Y >0)

{

additionVector1 += new Vector2(0, -addTo.Y);

additionVector2 += new Vector2(0, addTo.Y);

}

else currentCorner=1;

break;

}

}

sb.Draw(square, placeVector + additionVector1, null, Game1.blueTeamColor, 0f, Vector2.Zero, scale, SpriteEffects.None, 1f);

sb.Draw(square, placeVector + size + additionVector2, null, Game1.pinkTeamColor, 0f, Vector2.Zero, scale, SpriteEffects.None, 1f);

}

}

}

}

**Dice.cs**

using Microsoft.Xna.Framework;

using Microsoft.Xna.Framework.Graphics;

using System;

namespace BattleOfSquares

{

/// <summary>

/// CLass for drawing and manage dices

/// </summary>

public class Dice

{

public bool needToDraw;//нужно ли рисовать

static int timeForSide = 300;//сколько времени на одну сторону кости

static float scale = 0.7f;//масштаб

static Vector2 addTo = new Vector2(0, 5 \* scale);//вектор, на сколько изменять координаты

int currentTime = 0;//прошедшее время

public int[] randoms = new int[6];//массив рандомных чисел

//public для тестов

int currentSide = 1;//текущая сторона

Vector2 additionalVector = new Vector2(0, -100 \* scale);//вектор для изменения координат

/// <summary>

/// Request to generate a new roll

/// </summary>

/// <param name="diceNumber">Number of dice</param>

/// <param name="prCount">previous count for randomizer</param>

/// <returns>return random number</returns>

public int NewRoll(int diceNumber, int prCount)

{

randoms = new int[6];

currentSide = 1;

currentTime = 0;

if (diceNumber == 1) return doRandom(diceNumber, 0);

else return doRandom(diceNumber, prCount);

}

/// <summary>

/// Draws dices

/// </summary>

/// <param name="sb">SpriteBatch</param>

/// <param name="numberOfDices">How many dices</param>

/// <param name="team">team 0-blue, 1-pink</param>

public void Draw(SpriteBatch sb, int numberOfDices, int team)

{

Vector2 placeVector;

if (team == 0)

{

placeVector = new Vector2(1480, 30);

}

else if (team == 1) placeVector = new Vector2(1480, 560);

else return;

numberOfDices--;

Vector2 shiftPr = new Vector2(numberOfDices \* 110 \* scale, 100 \* scale);

Vector2 shift = new Vector2(numberOfDices \* 110 \* scale, 0);

currentTime += 16;

if (currentSide <= 6)

{

needToDraw = true;

if (currentTime >= timeForSide)//перелистывать, когда пришло время

{

currentTime = 0;

currentSide++;//перелистывание

additionalVector = Vector2.Zero;

}

if (currentSide != 7)

{

additionalVector += addTo;

Texture2D diceTexture = Game1.GetDiceTexture(randoms[currentSide - 1]);//получение текстуры стороны

if (currentSide != 1)

{

Texture2D dicePrevious = Game1.GetDiceTexture(randoms[currentSide - 2]);//получение текстуры предыдущей стороны

sb.Draw(dicePrevious, placeVector + shiftPr, null, Color.White, 0, Vector2.Zero, scale, SpriteEffects.None, 0.5f);//отрисовка предыдущей

sb.Draw(diceTexture, placeVector + additionalVector + shift, null, Color.White, 0, Vector2.Zero, scale, SpriteEffects.None, 1);//отрисовка этой

}

else

{

sb.Draw(diceTexture, placeVector + shiftPr, null, Color.White, 0, Vector2.Zero, scale, SpriteEffects.None, 1);//отрисовка этой

}

}

}

if (currentSide == 7)

{

needToDraw = false;

Texture2D diceTexture = Game1.GetDiceTexture(randoms[5]);//получение текстуры стороны

sb.Draw(diceTexture, placeVector + shiftPr, null, Color.White, 0, Vector2.Zero, scale, SpriteEffects.None, 1);//отрисовка

}

}

/// <summary>

/// Random a numbers for dices

/// </summary>

/// <param name="diceNumber">randomize first or second number</param>

/// <param name="c">count of randoms in previous dice</param>

/// <returns>count of randoms</returns>

public int doRandom(int diceNumber, int c)

{//public для тестов

Random rnd = new Random();

int count = 0;

if (diceNumber == 2)

{

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

{

rnd.Next(1, 7);

}

}

for (int i = 0; i < 6; i++)

{

bool write = true;

int r;

r = rnd.Next(1, 7);

count++;

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

{

if (randoms[j] == r)

{

write = false;

i--;

break;

}

}

if (write == true)

{

randoms[i] = r;

}

}

return count;

}

/// <summary>

/// returns a last element of randoms array - final number

/// </summary>

/// <returns>a last element of randoms array - final number</returns>

public int GetRandom()

{

if (randoms[5] == 0)

{

return 1;

}

else return randoms[5];

}

}

}

**GridSystem.cs**

using Microsoft.Xna.Framework;

using Microsoft.Xna.Framework.Graphics;

using System;

using System.Collections.Generic;

namespace BattleOfSquares

{

/// <summary>

/// A main class for control all things that connected with game field and with placing squares

/// </summary>

public class GridSystem

{

public int[,] gridArray = new int[20, 20];

//public is only for tests

List<Square.SquareInfo> squaresList = new List<Square.SquareInfo>();

int sumOfSquaresBlue;

int sumOfSquaresPink;

/// <summary>

/// Checking if square can be places in this position

/// </summary>

/// <param name="width">width of square</param>

/// <param name="height">height of square</param>

/// <param name="x">X of posiiton</param>

/// <param name="y">Y of posiiton</param>

/// <param name="rotate">is it rotated? 0 - no, 1 - yes</param>

/// <param name="team">team 0-blue, 1-pink, 2-error</param>

/// <returns>returns negative if it is impossible to fit,

/// positive if there is only enemies squares, and zero if it's all ok

/// </returns>

public int isItFit(int width, int height, int x, int y, int rotate, int team)

{//вмещается ли?

//если <0, то не вмещается вообще, если >0, то возле другой команды ставится, 0 если все впорядке

if (rotate == 1)

{

y += 1 - width;

int temp = width;

width = height;

height = temp;

}

if (x + width > 20 || y + height > 20 || y < 0) return -1;

for (int j = y; j < y + height; j++)

{

int sum = 0;

for (int i = x; i < x + width; i++)

{

sum += gridArray[j, i];

if (sum > 0 || gridArray[j, i] != 0)

return -1;

}

}

return isOnRightPlace(width, height, x, y, team);

}

/// <summary>

/// Another method for Checks is square in this position able to place, but it requares for squares name and position like a Point

/// </summary>

/// <param name="name">squares name in pattern "1-1"</param>

/// <param name="rotate">is it rotate? 0 - no, 1 - yes</param>

/// <param name="positionPoint">relative point where we place</param>

/// <param name="team">team 0-blue, 1-pink, 2-error</param>

/// <returns>returns negative if it is impossible to fit, positive if there is only enemies squares, and zero if its all ok</returns>

public int isItFit(string name, int rotate, Point positionPoint, int team)

{

int w = Convert.ToInt16(name.Substring(0, 1));

int h = Convert.ToInt16(name.Substring(2, 1));

return isItFit(w, h, positionPoint.X, positionPoint.Y, rotate, team);

}

/// <summary>

/// addition for isItFit that describes if you can place your square in this position according on its coords and another squares

/// </summary>

/// <param name="width">width of square</param>

/// <param name="height">height of square</param>

/// <param name="x">X of posiiton</param>

/// <param name="y">Y of posiiton</param>

/// <param name="team">team 0-blue, 1-pink, 2-error</param>

/// <returns>zero is for right place and 1 for it is not</returns>

public int isOnRightPlace(int width, int height, int x, int y, int team)

{//на правильное ли место ставим?

//если 0 - на верное место. Если 1 - нет

if (team == 0)//синие

{

int incJ = 0;//если на границе, надо учитывать на 1 строку меньше

int incI = 0;//если на границе, надо учитывать на 1 строку меньше

if (x == 0 && y == 0)//для первого

return 0;

if (x == 0)//касается левой границы

incI++;

if (y == 0)//касается верхней границы

incJ++;

if (x + width > 19)//касается правой границы

width--;

if (y + height > 19)//касается нижней границы

height--;

for (int j = y - 1 + incJ; j < y + height + 1; j++)//есть ли рядом что-то

{

for (int i = x - 1 + incI; i < x + width + 1; i++)

{

if (gridArray[j, i] % 2 == 1)

return 0;

}

}

}

else if (team == 1)//розовые

{

int incJ=0;//если на границе, надо учитывать на 1 строку меньше

int incI=0;//если на границе, надо учитывать на 1 строку меньше

if (x + width == 20 && y + height == 20)//для первого

return 0;

if (x == 0)//касается левой границы

incI++;

if (y == 0)//касается верхней границы

incJ++;

if (x + width > 19)//касается правой границы

width--;

if (y + height > 19)//касается нижней границы

height--;

for (int j = y - 1+incJ; j < y + height + 1; j++)//есть ли рядом что-то

{

for (int i = x - 1 + incI; i < x + width + 1; i++)

{

if (gridArray[j, i] != 0 && gridArray[j, i] % 2 == 0)

return 0;

}

}

}

return 1;

}

/// <summary>

/// describes if you can place your square in any position near your squares

/// </summary>

/// <param name="name">squares name with template "width-height"</param>

/// <param name="team">team 0-blue, 1-pink, 2-error</param>

/// <returns>true if it's the end, false if it is not</returns>

public bool isItTheEnd(string name, int team)

{

bool isIt = true;

int w = Convert.ToInt16(name.Substring(0, 1));

int h = Convert.ToInt16(name.Substring(2, 1));

for (int j = 0; j <= 19; j++)

{

for (int i = 0; i <= 19; i++)

{

if (gridArray[j, i] == 0)

{

if (isItFit(w, h, i, j, 0, team) == 0 || isItFit(w, h, i, j, 1, team) == 0)//если в точку помещается повернутый или не повернутый прямоугольник, то еще можно продолжать

{

if (isIt == true)

isIt = false;

}

}

}

}

return isIt;

}

/// <summary>

/// Main method for adding squares in system

/// </summary>

/// <param name="width">width of square</param>

/// <param name="height">height of square</param>

/// <param name="x">X of posiiton</param>

/// <param name="y">Y of posiiton</param>

/// <param name="team">team 0-blue, 1-pink, 2-error</param>

/// <param name="rotate">is it rotated? 0 - no, 1 - yes</param>

public void addSquare(int width, int height, int rotate, int team, int x, int y)

{

Point coords = new Point(x, y);

if (isItFit(width, height, x, y, rotate, team) == 0)

{

Square.SquareInfo el = new Square.SquareInfo(coords, height.ToString() + "-" + width.ToString(), rotate, team);

squaresList.Add(el);

el = null;

if (squaresList.Count % 2 == 1)

{

sumOfSquaresBlue += height \* width;

}

else sumOfSquaresPink += height \* width;

if (rotate == 1)

{

y += 1 - width;

int temp = width;

width = height;

height = temp;

}

for (int j = y; j < y + height; j++)

{

for (int i = x; i < x + width; i++)

{

gridArray[j, i] = squaresList.Count;

}

}

}

}

/// <summary>

/// Another method for adding squares in system

/// </summary>

/// <param name="name">squares name with template "width-height"</param>

/// <param name="rotate">is it rotate? 0 - no, 1 - yes</param>

/// <param name="team">team 0-blue, 1-pink, 2-error</param>

/// <param name="positionPoint">relative point at which we place</param>

public void addSquare(string name, int rotate, int team, Point positionPoint)

{

int w = Convert.ToInt16(name.Substring(0, 1));

int h = Convert.ToInt16(name.Substring(2, 1));

addSquare(w, h, rotate, team, positionPoint.X, positionPoint.Y);

}

/// <summary>

/// Draw both squares and score

/// </summary>

/// <param name="spriteBatch">Sprite Batch</param>

public void DrawAll(SpriteBatch spriteBatch)

{

DrawSquares(spriteBatch);

DrawScore(spriteBatch);

}

/// <summary>

/// Draw squares from squares list

/// </summary>

/// <param name="spriteBatch">Sprite Batch</param>

public void DrawSquares(SpriteBatch spriteBatch)

{

for (int n = 0; n < squaresList.Count; n++)

{

Square sq = new Square(spriteBatch);

Square.SquareInfo el = squaresList[n];

sq.Draw(el.name, el.rotate, el.team, el.position);

sq = null;

}

}

/// <summary>

/// Draw only score

/// </summary>

/// <param name="spriteBatch">Sprite Batch</param>

private void DrawScore(SpriteBatch spriteBatch)

{

Color blueTeamColor = new Color(102, 153, 255, 255);

Color pinkTeamColor = new Color(255, 51, 153, 255);

SpriteFont score = Game1.scoreDisplay;

spriteBatch.DrawString(score, "Score: " + sumOfSquaresBlue.ToString(), new Vector2(1480, 10), blueTeamColor);

spriteBatch.DrawString(score, "Score: " + sumOfSquaresPink.ToString(), new Vector2(1480, 540 + 10), pinkTeamColor);

}

/// <summary>

/// Clears all that Grid System includes

/// </summary>

public void Clear()

{

squaresList.Clear();

gridArray = new int[20, 20];

sumOfSquaresBlue = 0;

sumOfSquaresPink = 0;

}

}

}

**Program.cs**

using System;

namespace BattleOfSquares

{

#if WINDOWS || LINUX

/// <summary>

/// The main class.

/// </summary>

public static class Program

{

/// <summary>

/// The main entry point for the application.

/// </summary>

[STAThread]

static void Main()

{

using (var game = new Game1())

game.Run();

}

}

#endif

}

**Square.cs**

using Microsoft.Xna.Framework;

using Microsoft.Xna.Framework.Graphics;

using System;

namespace BattleOfSquares

{

//class for drawing squares

public class Square

{

public static Point sizeOfGrid = new Point(54, 54);

static Color wrongPlaceColor = new Color(255, 0, 0, 255);

SpriteBatch spriteBatch;

public Square(SpriteBatch spriteBatch)

{

this.spriteBatch = spriteBatch;

}

//Class for containing info about square

public class SquareInfo

{

public Point position;

public string name;

public int team;

public int rotate;

int teamWas = 3;

public bool wrong = false;

/// <summary>

/// Sets info

/// </summary>

/// <param name="position">Position point in relative points</param>

/// <param name="name">name with template "widht-height"</param>

/// <param name="rotate">if it rotate - 1, or 0 if its not</param>

/// <param name="team">team: 0-blue, 1 - pink, 2 - error</param>

public SquareInfo(Point position, string name, int rotate, int team)

{

this.position = position;

this.name = name;

this.team = team;

this.rotate = rotate;

}

/// <summary>

/// Sets info

/// </summary>

/// <param name="position">Position point in relative points</param>

/// <param name="width">width</param>

/// <param name="height">height</param>

/// <param name="rotate">if it rotate - 1, or 0 if its not</param>

/// <param name="team">team: 0-blue, 1 - pink, 2 - error</param>

public SquareInfo(Point position, int width, int height, int rotate, int team)

{

this.position = position;

name = height.ToString() + "-" + width.ToString();

this.team = team;

this.rotate = rotate;

}

/// <summary>

/// Changes rotate

/// </summary>

public void ChangeRotate()

{

rotate = (rotate == 0) ? 1 : 0;

}

/// <summary>

/// requares dices for set name in game

/// </summary>

/// <param name="dice1">number on fisrt dice</param>

/// <param name="dice2">number on second dice</param>

public void ChangeDices(int dice1, int dice2)

{

int x = dice1;

int y = dice2;

if (y < x)

{

int t = x;

x = y;

y = t;

}

name = x.ToString() + "-" + y.ToString();

}

/// <summary>

/// changes team

/// </summary>

public void ChangeTeam()

{

team = (team == 0) ? 1 : 0;

}

/// <summary>

/// requares isIt int from external class for remember team and set it in wrong (for draw a error square)

/// </summary>

/// <param name="isIt">if it is wrong - 1, else 0</param>

public void WrongPlace(int isIt)

{

if (isIt == 1)

{

teamWas = team;

team = 2;

wrong = true;

}

else

{

if (teamWas != 3)

{

team = teamWas;

teamWas = 3;

wrong = false;

}

}

}

}

/// <summary>

/// Draws a square

/// </summary>

/// <param name="width">width of square</param>

/// <param name="height">height of square</param>

/// <param name="rotate">is it rotate? 0 - no, 1 - yes</param>

/// <param name="team">team 0-blue, 1-pink, 2-error</param>

/// <param name="x">X of posiiton</param>

/// <param name="y">Y of posiiton</param>

public void Draw(int width, int height, int rotate, int team, int x, int y)

{

Vector2 position;

if (rotate == 1)

{

position = GetPoint(x, y + 1).ToVector2();

}

else position = GetPoint(x, y).ToVector2();

Color teamColor;

switch (team)

{

case 0: { teamColor = Game1.blueTeamColor; break; }

case 1: { teamColor = Game1.pinkTeamColor; break; }

case 2: { teamColor = wrongPlaceColor; break; }

default: { teamColor = Color.Black; break; }

}

string name = width.ToString() + "-" + height.ToString();

Texture2D sqTexture = Game1.GetSquareTexture(name);

spriteBatch.Draw(sqTexture, position, null, teamColor, 4.712f \* rotate, Vector2.Zero, 1f, SpriteEffects.None, 0.5f);

}

/// <summary>

/// Draws a square

/// </summary>

/// <param name="name">name with template "widht-height"</param>

/// <param name="rotate">is it rotate? 0 - no, 1 - yes</param>

/// <param name="team">team 0-blue, 1-pink, 2-error</param>

/// <param name="pos">X and Y in relative Point</param>

public void Draw(string name, int rotate, int team, Point pos)

{

int w = Convert.ToInt16(name.Substring(0, 1));

int h = Convert.ToInt16(name.Substring(2, 1));

Draw(w, h, rotate, team, pos.X, pos.Y);

}

/// <summary>

/// Draws square in pixel

/// </summary>

/// <param name="name">name with template "widht-height"</param>

/// <param name="rotate">is it rotate? 0 - no, 1 - yes</param>

/// <param name="team">team 0-blue, 1-pink, 2-error</param>

/// <param name="pos">X and Y in absoulte Point</param>

/// <param name="spriteBatch">Sprite Batch</param>

public void DrawInPixel(string name, int rotate, int team, Point pos, SpriteBatch spriteBatch)

{

int w = Convert.ToInt16(name.Substring(0, 1));

int h = Convert.ToInt16(name.Substring(2, 1));

Vector2 position;

if (rotate == 1) position = pos.ToVector2() + new Vector2(0, 54);

else position = pos.ToVector2();

Color teamColor;

switch (team)

{

case 0: { teamColor = Game1.blueTeamColor; break; }

case 1: { teamColor = Game1.pinkTeamColor; break; }

case 2: { teamColor = wrongPlaceColor; break; }

default: { teamColor = Color.Black; break; }

}

Texture2D sqTexture = Game1.GetSquareTexture(name);

spriteBatch.Draw(sqTexture, position, null, teamColor, 4.712f \* rotate, Vector2.Zero, 1f, SpriteEffects.None, 1f);

}

/// <summary>

/// Returns a absolute point by x and y of relative

/// </summary>

/// <param name="x">X</param>

/// <param name="y">Y</param>

/// <returns>Point in absolute presentation</returns>

public static Point GetPoint(int x, int y)

{//возвращает point по координатам сетки

return new Point(x \* 54 + (int)Game1.startPoint.X, y \* 54);

}

}

}

**Texture.cs**

using Microsoft.Xna.Framework.Graphics;

namespace BattleOfSquares

{

/// <summary>

/// an addition class that helps contain Texture2D and its name or number

/// </summary>

public class Texture

{//объект texture2d, но с именем отдельно

public Texture2D texture;//текстура

public string name;//имя, без имени папки

public int number;//номер, для перегрузки с интом

/// <summary>

/// Init for squares

/// </summary>

/// <param name="t">Texture2D</param>

/// <param name="n">name of square</param>

public Texture(Texture2D t, string n)//конструктор для квадратиков

{

texture = t;

name = n;

}

/// <summary>

/// Init for dices

/// </summary>

/// <param name="t">Texture2D</param>

/// <param name="n">number of dice</param>

public Texture(Texture2D t, int n)//конструктор для костей

{

texture = t;

number = n;

}

}

}

**DiceTest.cs**

using NUnit.Framework;

namespace BattleOfSquares.Tests

{

[TestFixture]

public class DiceTest

{

[Test]

public void doRandomTest()

{

Dice test1 = new Dice();

Dice test2 = new Dice();

int count1 = test1.doRandom(1, 0);

int count2 = test2.doRandom(2, count1);

Assert.AreNotEqual(test1.randoms, test2.randoms);

}

[Test]

public void GetRandomTest()

{

Dice test1 = new Dice();

test1.doRandom(1, 0);

Assert.AreNotEqual(test1.randoms[5], 0);

}

}

}

**GridSystemTest.cs**

using NUnit.Framework;

using System;

namespace BattleOfSquares.Tests

{

[TestFixture]

class GridSystemTest

{

[TestCase(5, 5, 7, 0, 0, 0,-1)]

[TestCase(1, 1, 0, 0, 0, 0, -1)]

[TestCase(2, 3, 6, 7, 1, 0, 0)]

public void isItFitTest(int width, int height, int x, int y, int rotate, int team, int expected)

{

int[,] array = new int[20, 20]{

{ 1,1,1,1,3,3,3,3,5,7,7,7,9,9,9,11,11,11,11,11},

{ 1,1,1,1,3,3,3,3,5,7,7,7,9,9,9,11,11,11,11,11},

{ 1,1,1,1,3,3,3,3,5,7,7,7,9,9,9,11,11,11,11,11},

{ 1,1,1,1,3,3,3,3,5,23,23,23,9,9,9,11,11,11,11,11},

{ 1,1,1,1,13,13,13,13,5,27,27,27,25,25,21,11,11,11,11,11},

{ 1,1,1,1,13,13,13,13,31,27,27,27,25,25,21,15,15,15,15,15},

{ 17,17,17,17,17,17,0,0,0,27,27,27,25,25,21,19,19,19,19,19},

{ 0,0,0,0,0,0,0,0,0,27,27,27,25,25,21,19,19,19,19,19},

{ 0,0,0,30,30,30,30,30,0,0,29,29,29,29,21,19,19,19,19,19},

{ 0,0,0,30,30,30,30,30,28,28,28,28,28,28,0,19,19,19,19,19},

{ 0,0,0,30,30,30,30,30,28,28,28,28,28,28,0,19,19,19,19,19},

{ 0,0,0,30,30,30,30,30,28,28,28,28,28,28,26,26,26,26,26,26},

{ 0,0,24,24,24,0,0,0,28,28,28,28,28,28,26,26,26,26,26,26},

{ 0,0,24,24,24,0,0,0,28,28,28,28,28,28,26,26,26,26,26,26},

{ 0,0,24,24,24,12,12,18,18,8,8,14,14,14,14,2,2,2,2,2},

{ 0,0,24,24,24,12,12,18,18,8,8,6,16,16,16,2,2,2,2,2},

{ 0,0,24,24,24,12,12,10,10,8,8,6,16,16,16,2,2,2,2,2},

{ 0,0,20,20,20,12,12,10,10,8,8,6,4,4,4,2,2,2,2,2},

{ 22,22,20,20,20,12,12,10,10,8,8,6,4,4,4,2,2,2,2,2},

{ 22,22,20,20,20,12,12,10,10,8,8,6,4,4,4,2,2,2,2,2} };

GridSystem gs = new GridSystem();

gs.gridArray = array;

int result = gs.isItFit(width, height, x, y, rotate, team);

Assert.AreEqual(result, expected);

}

[TestCase(1, 2, 7, 0, 0,0)]

[TestCase(1, 2, 7, 0, 1, 1)]

[TestCase(2, 3, 6, 7, 1,0)]

public void isItOnRightPlace(int width, int height, int x, int y, int team, int expected)

{

int[,] array = new int[20, 20]{

{ 1,1,1,1,3,3,3,3,5,7,7,7,9,9,9,11,11,11,11,11},

{ 1,1,1,1,3,3,3,3,5,7,7,7,9,9,9,11,11,11,11,11},

{ 1,1,1,1,3,3,3,3,5,7,7,7,9,9,9,11,11,11,11,11},

{ 1,1,1,1,3,3,3,3,5,23,23,23,9,9,9,11,11,11,11,11},

{ 1,1,1,1,13,13,13,13,5,27,27,27,25,25,21,11,11,11,11,11},

{ 1,1,1,1,13,13,13,13,31,27,27,27,25,25,21,15,15,15,15,15},

{ 17,17,17,17,17,17,0,0,0,27,27,27,25,25,21,19,19,19,19,19},

{ 0,0,0,0,0,0,0,0,0,27,27,27,25,25,21,19,19,19,19,19},

{ 0,0,0,30,30,30,30,30,0,0,29,29,29,29,21,19,19,19,19,19},

{ 0,0,0,30,30,30,30,30,28,28,28,28,28,28,0,19,19,19,19,19},

{ 0,0,0,30,30,30,30,30,28,28,28,28,28,28,0,19,19,19,19,19},

{ 0,0,0,30,30,30,30,30,28,28,28,28,28,28,26,26,26,26,26,26},

{ 0,0,24,24,24,0,0,0,28,28,28,28,28,28,26,26,26,26,26,26},

{ 0,0,24,24,24,0,0,0,28,28,28,28,28,28,26,26,26,26,26,26},

{ 0,0,24,24,24,12,12,18,18,8,8,14,14,14,14,2,2,2,2,2},

{ 0,0,24,24,24,12,12,18,18,8,8,6,16,16,16,2,2,2,2,2},

{ 0,0,24,24,24,12,12,10,10,8,8,6,16,16,16,2,2,2,2,2},

{ 0,0,20,20,20,12,12,10,10,8,8,6,4,4,4,2,2,2,2,2},

{ 22,22,20,20,20,12,12,10,10,8,8,6,4,4,4,2,2,2,2,2},

{ 22,22,20,20,20,12,12,10,10,8,8,6,4,4,4,2,2,2,2,2} };

GridSystem gs = new GridSystem();

gs.gridArray = array;

int result = gs.isOnRightPlace(width, height, x, y, team);

Assert.AreEqual(result, expected);

}

[TestCase("1-1", 0, false)]

[TestCase("1-2", 0, false)]

[TestCase("1-6", 0, false)]

[TestCase("4-6", 0, true)]

[TestCase("4-6", 1, true)]

public void isItTheEnd(string name, int team, bool expected)

{

int[,] array = new int[20, 20]{

{ 1,1,1,1,3,3,3,3,5,7,7,7,9,9,9,11,11,11,11,11},

{ 1,1,1,1,3,3,3,3,5,7,7,7,9,9,9,11,11,11,11,11},

{ 1,1,1,1,3,3,3,3,5,7,7,7,9,9,9,11,11,11,11,11},

{ 1,1,1,1,3,3,3,3,5,23,23,23,9,9,9,11,11,11,11,11},

{ 1,1,1,1,13,13,13,13,5,27,27,27,25,25,21,11,11,11,11,11},

{ 1,1,1,1,13,13,13,13,31,27,27,27,25,25,21,15,15,15,15,15},

{ 17,17,17,17,17,17,0,0,0,27,27,27,25,25,21,19,19,19,19,19},

{ 0,0,0,0,0,0,0,0,0,27,27,27,25,25,21,19,19,19,19,19},

{ 0,0,0,30,30,30,30,30,0,0,29,29,29,29,21,19,19,19,19,19},

{ 0,0,0,30,30,30,30,30,28,28,28,28,28,28,0,19,19,19,19,19},

{ 0,0,0,30,30,30,30,30,28,28,28,28,28,28,0,19,19,19,19,19},

{ 0,0,0,30,30,30,30,30,28,28,28,28,28,28,26,26,26,26,26,26},

{ 0,0,24,24,24,0,0,0,28,28,28,28,28,28,26,26,26,26,26,26},

{ 0,0,24,24,24,0,0,0,28,28,28,28,28,28,26,26,26,26,26,26},

{ 0,0,24,24,24,12,12,18,18,8,8,14,14,14,14,2,2,2,2,2},

{ 0,0,24,24,24,12,12,18,18,8,8,6,16,16,16,2,2,2,2,2},

{ 0,0,24,24,24,12,12,10,10,8,8,6,16,16,16,2,2,2,2,2},

{ 0,0,20,20,20,12,12,10,10,8,8,6,4,4,4,2,2,2,2,2},

{ 22,22,20,20,20,12,12,10,10,8,8,6,4,4,4,2,2,2,2,2},

{ 22,22,20,20,20,12,12,10,10,8,8,6,4,4,4,2,2,2,2,2} };

int w = Convert.ToInt16(name.Substring(0, 1));

int h = Convert.ToInt16(name.Substring(2, 1));

GridSystem gs = new GridSystem();

gs.gridArray = array;

bool result = gs.isItTheEnd(name, team);

Assert.AreEqual(result, expected);

}

}

}

**SquareTest.cs**

using Microsoft.Xna.Framework;

using NUnit.Framework;

namespace BattleOfSquares.Tests

{

[TestFixture]

public class SquareTest

{

[TestCase(0,0,370,0)]

[TestCase(0, 1, 370, 54)]

[TestCase(19,19,1396,1026)]

public void GetPointTest(int x, int y, int xExpected, int yExpected)

{

Point result = Square.GetPoint(x, y);

Point expected = new Point(xExpected, yExpected);

Assert.AreEqual(result,expected);

}

}

}