**Федеральное государственное образовательное бюджетное учреждение  
высшего образования**

**«Финансовый университет при Правительстве Российской Федерации»**

**КОЛЛЕДЖ ИНФОРМАТИКИ И ПРОГРАММИРОВАНИЯ**

**ПМ.03 Участие в интеграции программных модулей**

**Группа: 4ПКС-314**

**УТВЕРЖДАЮ**

**Председатель цикловой комиссии**

**программирования и баз данных**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ /Пестов А.И./**

**\_\_\_\_.\_\_\_\_. 2017**

**ПРОЕКТ КУРСОВОЙ**

**На тему:** Компьютерная игра «Домино» c графическим интерфейсом на языке С#

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**Руководитель курсового проекта**

**\_\_\_\_\_\_\_\_\_\_\_\_\_ /Морозова М.В./**

**Исполнитель курсового проекта**

**\_\_\_\_\_\_\_\_\_\_\_\_\_ /\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/**

**Оценка за проект: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**\_\_\_\_.\_\_\_\_.2017**

**2017**

Содержание

Введение 3

1. Общий раздел 4

1.1 Системные требования 4

1.2 Характеристика системы программирования 4

2. Технологический раздел 6

2.1 Предпроектное исследование предметной области 6

2.2 Анализ требований и определение спецификаций программного обеспечения 7

2.3 Проектирование программного обеспечения 10

2.4 Разработка пользовательских интерфейсов программного обеспечения 11

2.5 Тестирование и отладка программного обеспечения 14

3. Руководство по использованию программы 17

3.1 Руководство системного программиста 17

3.2 Руководство программиста 18

3.3 Руководство оператора 21

Заключение 23

Список использованных источников 24

Приложение А 25

Приложение Б 77

Введение

В данной курсовой работе требуется реализовать логическую игру “Домино”. Задачей пользователя в этой игре является выставление кости в тот или иной конец уже существующей цепочки костей.

Определять куда положить костяшку можно с помощью логического мышления, анализируя цифры, которые расположены на концах костяшек. Начинает игрок, у которого на руках находится "дубль шесть" (6-6). Следующие игроки выставляют камни с соответствующими очками (6-1; 6-2; 6-3…). Если подходящих камней нет, то приходится добирать из резерва. Если ни у кого из игроков нет на руках дубля 6-6, можно начать игру дублем 5-5. Если же на руках нет ни одного дубля, "на базар" за дублями не ходят, а начинают с камня, имеющего наибольшее количество (например, 6-5). Игра заканчивается, когда один из игроков выложит последний камень.

Цель данного курсового проекта – досуг пользователя, а также обучение и развития навыков игры “Домино”.

Данная программа будет реализована для операционной системы Windows с использованием графической библиотеки XNA и языка C#.

# Общий раздел

## Системные требования

## Рекомендованные системные требования:

* Выпуск Windows: Windows 7, Корпорация Майкрософт (Microsoft Corporation), 2008г. Либо более новая версия.
* Требования к системе:
* Процессор: Intel (R) Core (TM) i3-2130 CPU @ 3.40GHz 3.40 GHz.
* Установленная память (ОЗУ): 8,00 ГБ.
* Тип системы: 64-разрядная операционная система, процессор х64.
* Клавиатура.
* USB-мышь либо наличие сенсорной панели (touchpad).

## Характеристика среды программирования

Данная программа была написана на объектно-ориентированном языке программирования C# с помощью MS Visual Studio 17 Community с множеством различных встроенных библиотек и функций, удобным для пользователя интерфейсом и наличием отладчика кода.

Программирование выполнялось на языке C#, тип проекта – графическое окно.

C# — язык программирования, сочетающий объектно-ориентированные и контекстно-ориентированные концепции. Разработан в 1998—2001 годах группой инженеров под руководством Андерсa Хейлсбергa в компании Microsoft как основной язык разработки приложений для платформы Microsoft .NET. Компилятор с C# входит в стандартную установку самой .NET, поэтому программы на нём можно создавать и компилировать даже без инструментальных средств вроде Visual Studio.

C# относится к семье языков с C-подобным синтаксисом, из них его синтаксис наиболее близок к C++ и Java. Язык имеет строгую статическую типизацию, поддерживает полиморфизм, перегрузку операторов, указатели на функции-члены классов, атрибуты, события, свойства, исключения, комментарии в формате XML. Переняв многое от своих предшественников — языков C++, Delphi, Modula и Smalltalk — С#, опираясь на практику их использования, исключает некоторые модели, зарекомендовавшие себя как проблематичные при разработке программных систем: так, C# не поддерживает множественное наследование классов (в отличие от C++) или вывода типов (в отличие от Haskell).

1. Технологический раздел

2.1 Предпроектное исследование предметной области

Приложение должно представлять классическую логическую игру Домино. В данной игре, кости будут располагаться на двумерном поле. Программа должна иметь интуитивно понятный эргономичный интерфейс. Должны присутствовать меню, для удобного управления программой, а также для отображения информации об авторе. Настройки программы должны загружаться из файла, и при изменении пользователем, сохраняться.

Входная информация:

* Выбор сложности компьютера.   
  Выходная информация:
* Интерфейс игры.

Функциональные требования:

* Выбор сложности;
* Возможность игры с компьютером;
* Возможность просмотра правил;
* Возможность сохранения результатов.   
  Нефункциональные требования:
* Работа программы под операционной системой Microsoft Windows;
* Графический интерфейс пользователя;
* Отказоустойчивость;
* Простота в освоении для пользования.
  + 1. Обзор существующих аналогов

Самыми популярными графическими редакторами являются:

1. Domino Mail.ru
2. Domino Dominoplay.ru

Основными недостатками этих программ является, то что они почти все условно бесплатные, и напичканы кучей различных функцией, что значительно осложняет работу с ними. Сложные интерфейсы пугают не опытных пользователей.

К достоинству этих программ можно приписать: быстродействие, отсутствие багов, кроссплатформенность, большой функционал.

2.2 Анализ требований и определение спецификаций программного обеспечения

2.2.1 Диаграмма вариантов использования

Диаграмма вариантов использования представлена на рисунке 1, и состоит из действий, совершаемых пользователем в программе.

Пользователь играет в игру, которая включает в себя:

* Начало игры;
* Выбор настроек;
* Сохранение итогового результата.



Рисунок 1 – Диаграмма вариантов использования

2.2.2 Функциональная диаграмма

Функциональная диаграмма - диаграмма, отражающая взаимосвязи функций разрабатываемого программного обеспечения

Контекстная функциональная диаграмма представлена на рисунке 4.

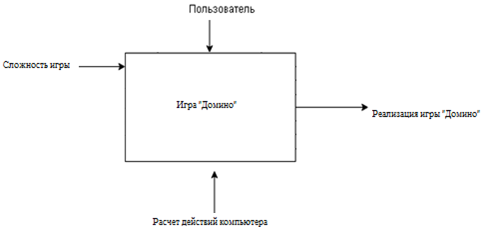


Рисунок 2 – Контекстная функциональная диаграмма

Детализированная функциональная диаграмма представлена на рисунке 5.

Данная диаграмма содержит основные функции программы:

* Вывод меню;
* Вывод выбранного пункта меню;
* Обработка выбранных данных;
* Вывод одного из списков;
* Расчет в зависимости от действий пользователя.

На вход программе подается:

* режим сложности.

На выходе пользователь получит:

* итоговый результат.

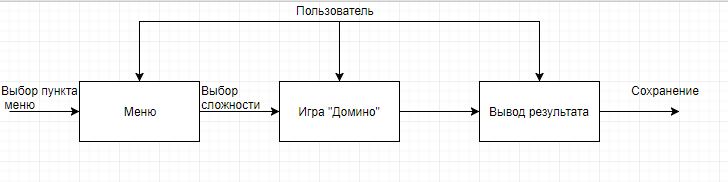


Рисунок 3 – Детализированная функциональная диаграмма

2.2.3 Диаграмма потоков данных

Контекстная диаграмма потоков данных представлена на рисунке 2, и представляет собой взаимодействие пользователя и приложения.

На диаграмме можно увидеть, что от пользователя к программе поступают такие действия, как:

* Выбор пункта меню;
* Выбор режима сложности.

Также можно увидеть, что от программы к пользователю поступают такие действия, как:

* Вывод меню;
* Вывод интерфейса игры;
* Вывод меню.

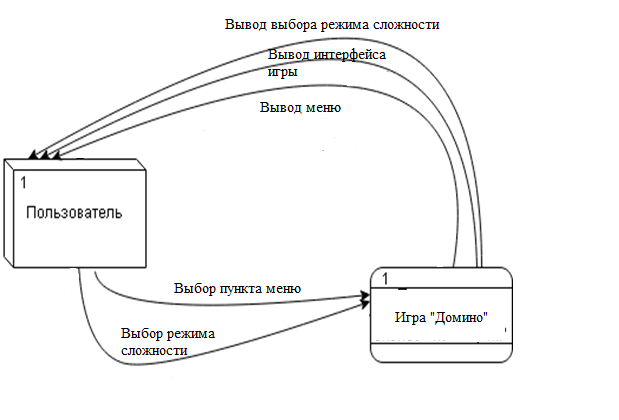


Рисунок 4 – Контекстная диаграмма потоков данных

Детализированная диаграмма потоков данных является более подробной и представлена на рисунке 3.

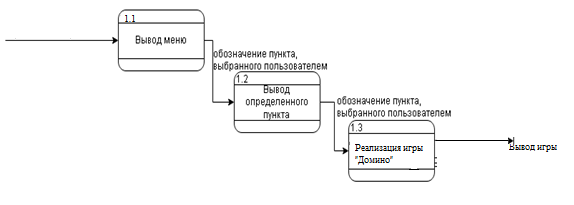


Рисунок 5 – Детализированная диаграмма потоков данных

2.3 Проектирование программного обеспечения

2.3.1 Структурная схема

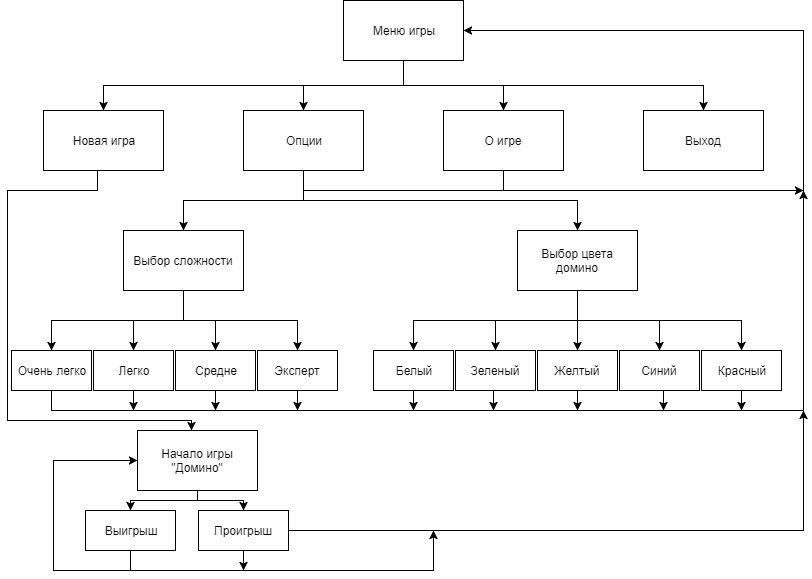


Рисунок 6 – Структурная схема

2.3.2 Функциональная схема

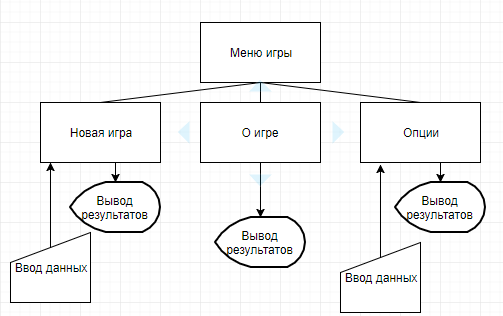


Рисунок 7 – Функциональная схема

2.4. Разработка пользовательских интерфейсов программного обеспечения.

2.4.1 Интерфейс главного меню

При загрузке игры перед пользователем появляется меню с несколькими пунктами, в виде анимированных пунктов меню, при наведении на которые проигрывается анимация. На рисунке 7 показано меню.

Список имеющихся пунктов меню:

- Новая игра;

- Опции;

- Об игре;

- Выйти.

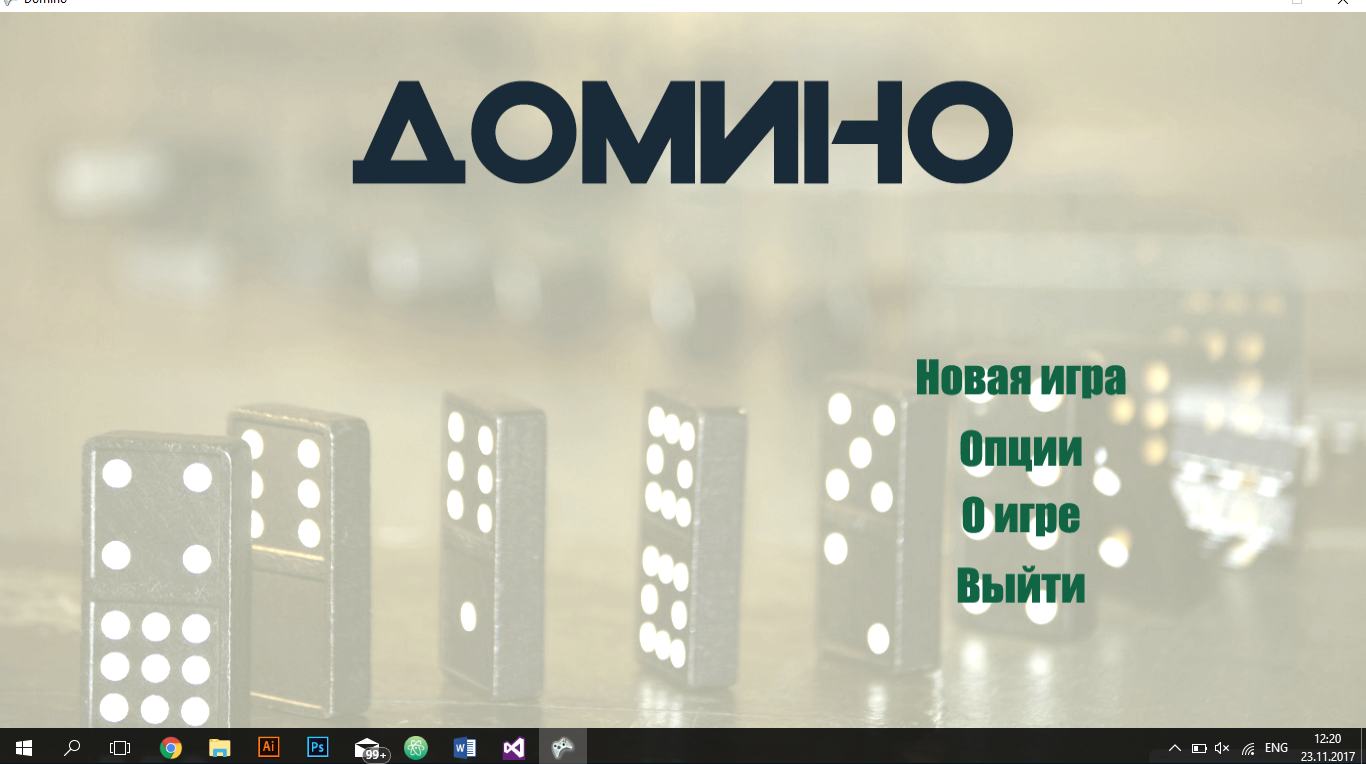


Рисунок 7 – Меню

2.4.2 Интерфейс пункта “Опции”

“Опции” включают в себя несколько подпунктов:

- Сложность;

- Цвет домино.

Дизайн меню данного пункта похож на дизайн главного меню и продемонстрирован на рисунке 8.



Рисунок 8 – Опции

Подпункт “Сложность”. При нажатии на данный подпункт перед пользователем выведется меню выбора сложности игры.

Подпункт “Цвет домино”. При нажатии на данный подпункт перед пользователем выведется меню выбора цвета домино.

2.4.3 Интерфейс пункта “Об игре”

В пункте “Об игре” содержится информация о программе и ее авторе (рисунок 11).

Дизайн меню данного пункта похож на дизайн главного меню и продемонстрирован на рисунке 9.



Рисунок 9 – Об игре

2.4.3 Интерфейс пункта “Новая игра”

Данный пункт запускает игру “Домино”, интерфейс, который показан на рисунке 10

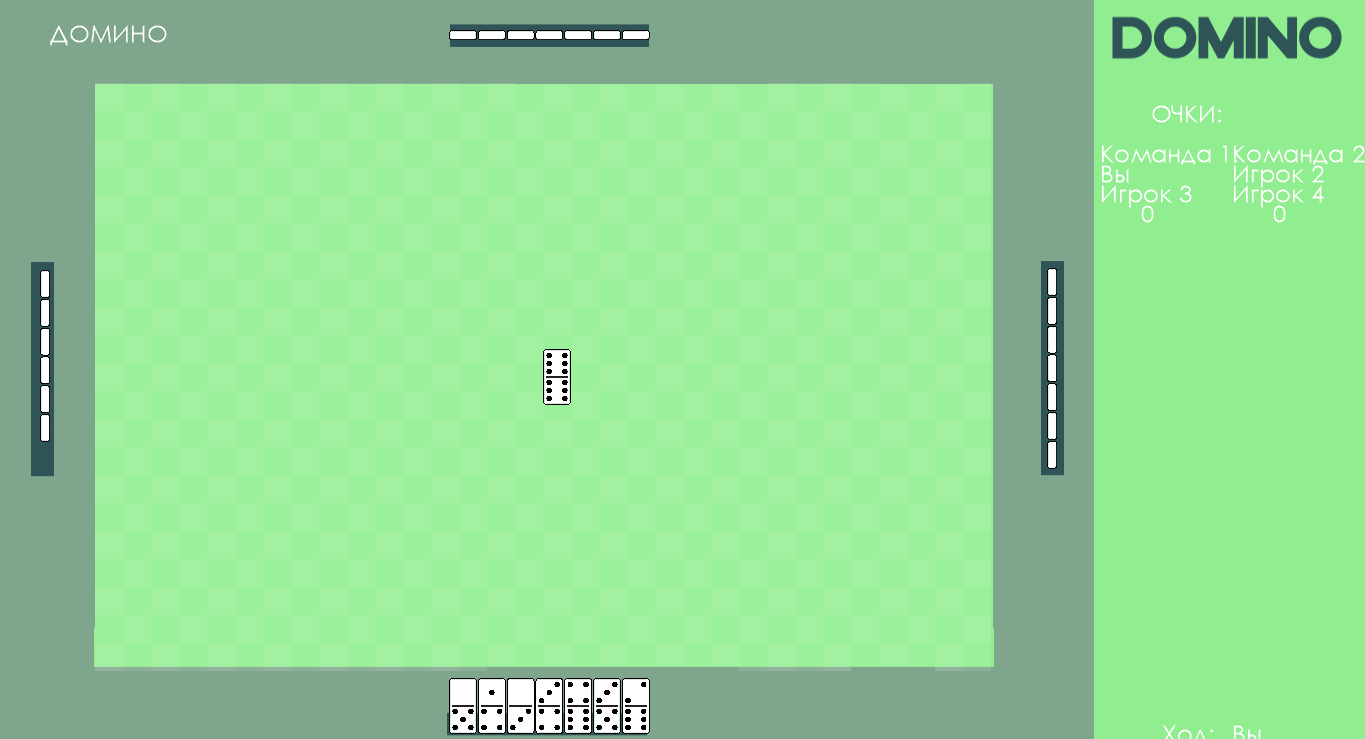


Рисунок 10 – Игра “Домино”

При поражении или победе появляется окно с советующей надписью, с возможностью начать следующий раунд или закончить игру нажав Esc.

2.5. Тестирование и отладка программного обеспечения.

В разделе “Опции” есть возможность выбора сложности и цвета домино.

Подпункт “Сложность”. При нажатии на данный подпункт перед пользователем выведется меню выбора сложности игры. (рисунок 11).



Рисунок 11 – Сложность

Подпункт “Цвет домино”. При нажатии на данный подпункт перед пользователем выведется меню выбора цвета домино. (рисунок 12).

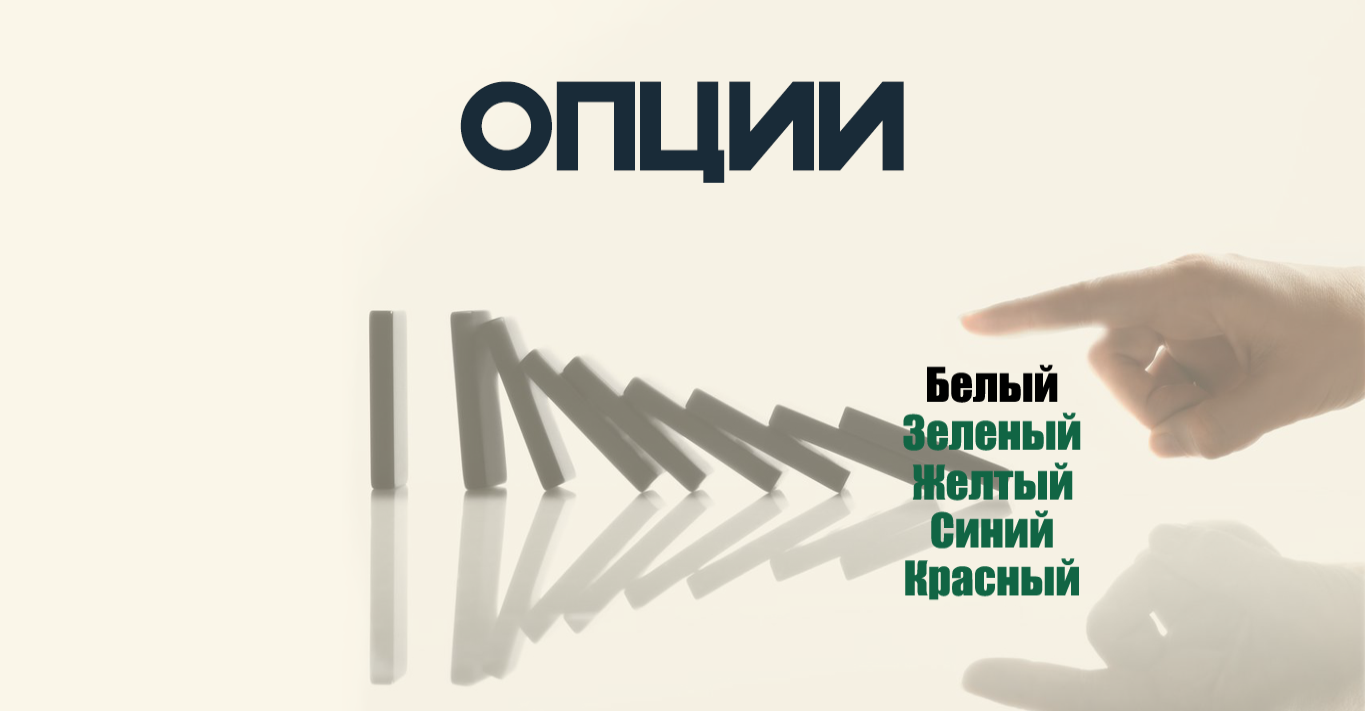


Рисунок 12 – Цвет домино

При поражении или победе появляется окно с советующей надписью, с возможностью начать следующий раунд или закончить игру нажав Esc. (рисунки 13, 14)

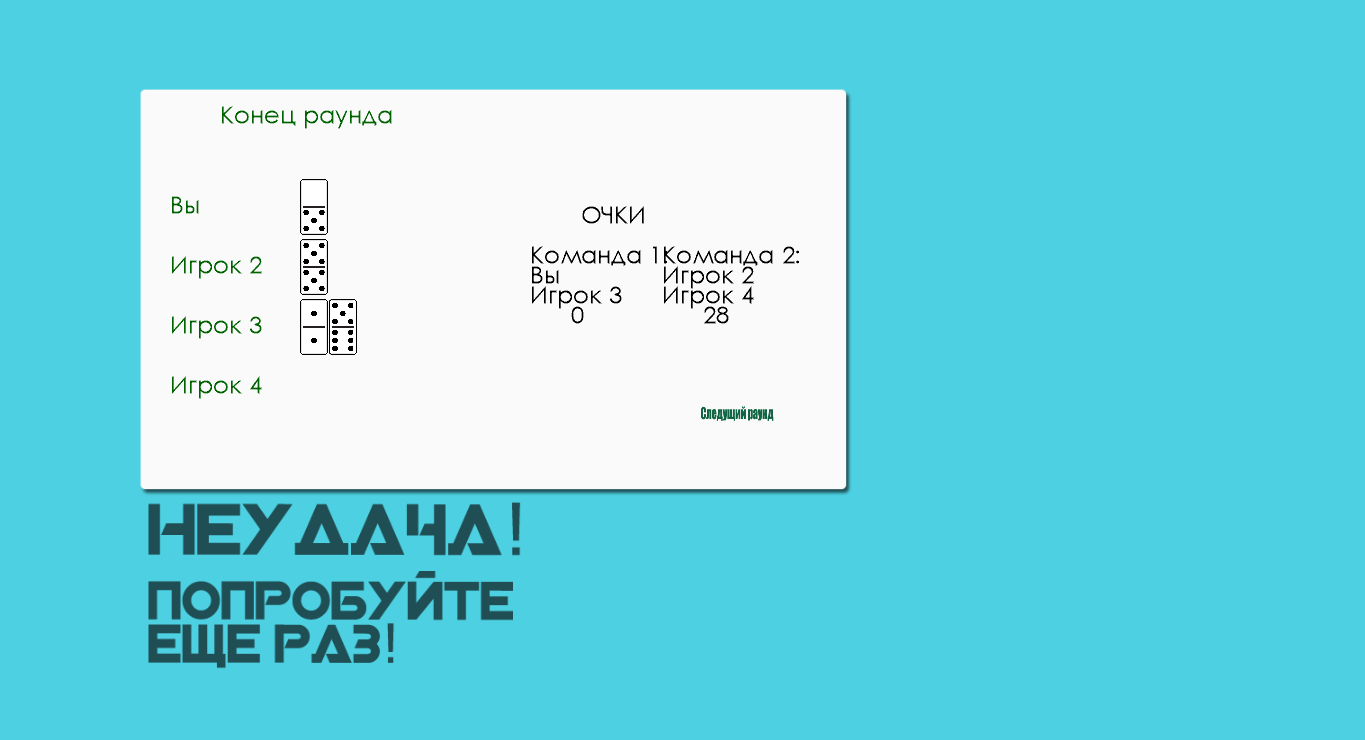


Рисунок 13 – Окно окончания раунда при поражении

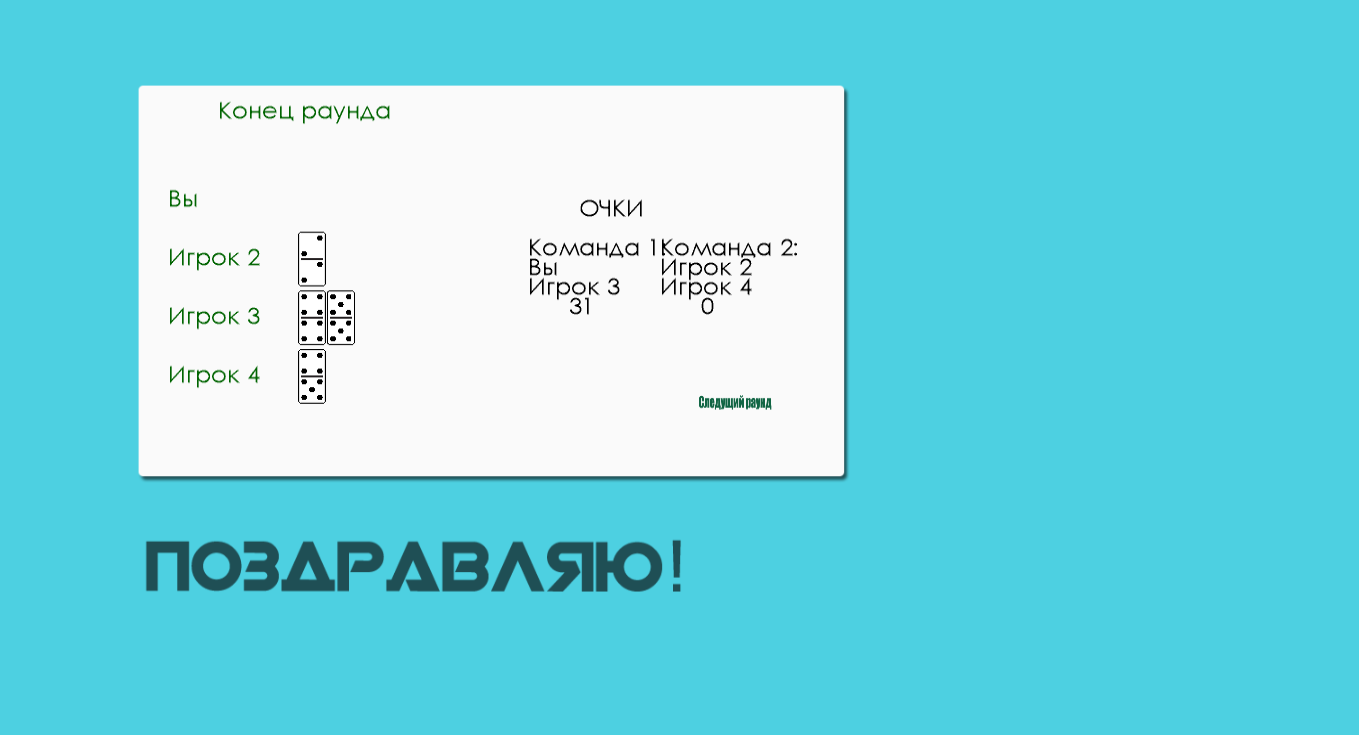


Рисунок 14 – Окно окончания раунда при победе

В ходе тестирования программы выявлено, что предъявляемые функциональные требования к программе выполняются. После нескольких тестовых запусков программы выяснилось, что предсказать победу игрока невозможно.

Также выполняются нефункциональные требования к программе. Программа имеет достаточно понятный интерфейс в виде меню, позволяет представлять информацию в графическом виде.

3. Руководство по использованию программы

3.1 Руководство системного программиста.

3.1.1 Общие сведения о программе

Логическая игра «Домино» является приложением, с помощью которого пользователь имеет возможность провести досуг и обучиться навыкам игре «Домино».

Функции, выполняемые программой:

* Игра «Домино»;
* Выбор режима сложности;
* Выбор цвета домино;
* Сохранение результата для дальнейшей игры.

3.1.2 Системные требования

* Системный блок;
* Монитор;
* Мышь либо наличие touchpad;
* Клавиатура;
* Процессор: Intel core i5;
* OЗУ: 6,00 ГБ;
* Тип системы: 64-разрядная операционная система.

3.1.3 Программные требования

* операционная система Windows 8/10.

3.1.4 Файлы программы

Приложение состоит из следующих файлов:

* файл «Domino.exe», являющийся главным файлом приложения;
* папка Content, содержит картинки и шрифты для правильного выполнения игры.

3.1.5 Проверка программы

Для проверки программы следует запустить исполняемый файл «Domino.exe» двойным щелчком левой клавишей мыши.

Далее на экране появится главное меню игры со следующими пунктами:

* Новая игра;
* Опции
* Об игре;
* Выйти.

3.1.6 Сообщения системному программисту

Сообщения системному программисту в данном курсовом проекте отсутствуют.

3.2 Руководство программиста

3.2.1 Назначение и условия применения программы

Логическая игра «Домино» является приложением, с помощью которого пользователь имеет возможность провести досуг и обучиться навыкам игре «Домино».

Функции, выполняемые программой:

* Игра «Домино»;
* Выбор режима сложности;
* Выбор цвета домино;

Сохранение результата для дальнейшей игры Необходимыми аппаратными средствами для работы данного приложения являются:

* Системный блок;
* Монитор;
* Мышь либо наличие touchpad;
* Клавиатура;
* Процессор: Intel core i5;
* OЗУ: 6,00 ГБ;
* Тип системы: 64-разрядная операционная система.

Необходимыми программными средствами являются:

* операционная система Windows 8/10.

3.2.2 Структура программы

Таблица 1 – Функции программы

|  |  |
| --- | --- |
| Функция | Действие |
| LoadContent() | Загрузка контента |
| Update() | Обновление экрана |
| Draw() | Отрисовка элементов на экране |
| DrawTilesOnBoard() | Отрисовка домино на игровом поле |
| DrawDraggableSquare() | Отрисовка перетаскивание домино |
| DrawBoard() | Отрисовка игрового поля |
| IsMouseCoordinateWithinBoard() | Возвращает координаты мыши |
| IsMouseInsideBoard() | Возвращает True если мышь над игровым полем |
| GetSquareFromMousePosition() | Берет позицию мыши на игровом поле |
| DealDominoTilesToPlayers() | Раздает домино всем игрокам |
| UpdateTable() | Обновляет игровое поле |
| TilePlacementLogic() | Расстановка домино на игровом поле |
| AddDominoTile() | Добавляет домино |
| CalculateTurn() | Рассчитывает чей сейчас ход |
| FindOutIfPlayerPassesHisTurn() | Рассчитывает если игрок пропускает ход |
| DrawTilesForNonHumanPlayersLevel1() | Рисует домино для неживых игроков |
| VerifyEndOfRoundCondition() | Проверяет конец раунда |
| InitiateSave() | Сохранение игры |

Таблица 2 – Классы программы

|  |  |
| --- | --- |
| Game1.cs | Исполнительный класс игры |
| eDifficultyLevel.cs | Перечисление уровней сложности |
| eTileColors.cs | Перечисление цветов домино |
| MenuButton.cs | Класс кнопок меню |
| Player.cs | Класс игрока |
| Sprite.cs | Класс для спрайтов |
| Table.cs | Класс игрового поля |
| Tile.cs | Класс домино |

3.2.3 Обращение к программе

Для запуска программы необходимо дважды кликнуть левой кнопкой мыши по файлу «Domino.exe».

3.2.4 Входные и выходные данные

Входными данными являются:

* Сложность игры;
* Цвет домино.

Выходными данными являются:

* Очки после победы или поражения.

3.2.5 Сообщения программисту

В данном курсовом проекте отсутствуют сообщения программисту.

3.3 Руководство оператора

3.3.1 Назначение и условия применения программы

Логическая игра «Домино» является приложением, с помощью которого пользователь имеет возможность провести досуг и обучиться навыкам игре «Домино».

Функции, выполняемые программой:

* Игра «Домино»;
* Выбор режима сложности;
* Выбор цвета домино.

Необходимыми аппаратными средствами для работы данного приложения являются:

* Системный блок;
* Монитор;
* Мышь либо наличие touchpad;
* Клавиатура;
* Процессор: Intel core i5;
* OЗУ: 6,00 ГБ;
* Тип системы: 64-разрядная операционная система.

Необходимыми программными средствами являются:

* Операционная система Windows 8/10.

3.3.2 Выполнение программы

Для проверки программы следует запустить исполняемый файл «Domino.exe» двойным щелчком левой клавишей мыши.

Далее на экране появится главное меню игры со следующими пунктами:

* Новая игра;
* Опции
* Об игре;
* Выйти.

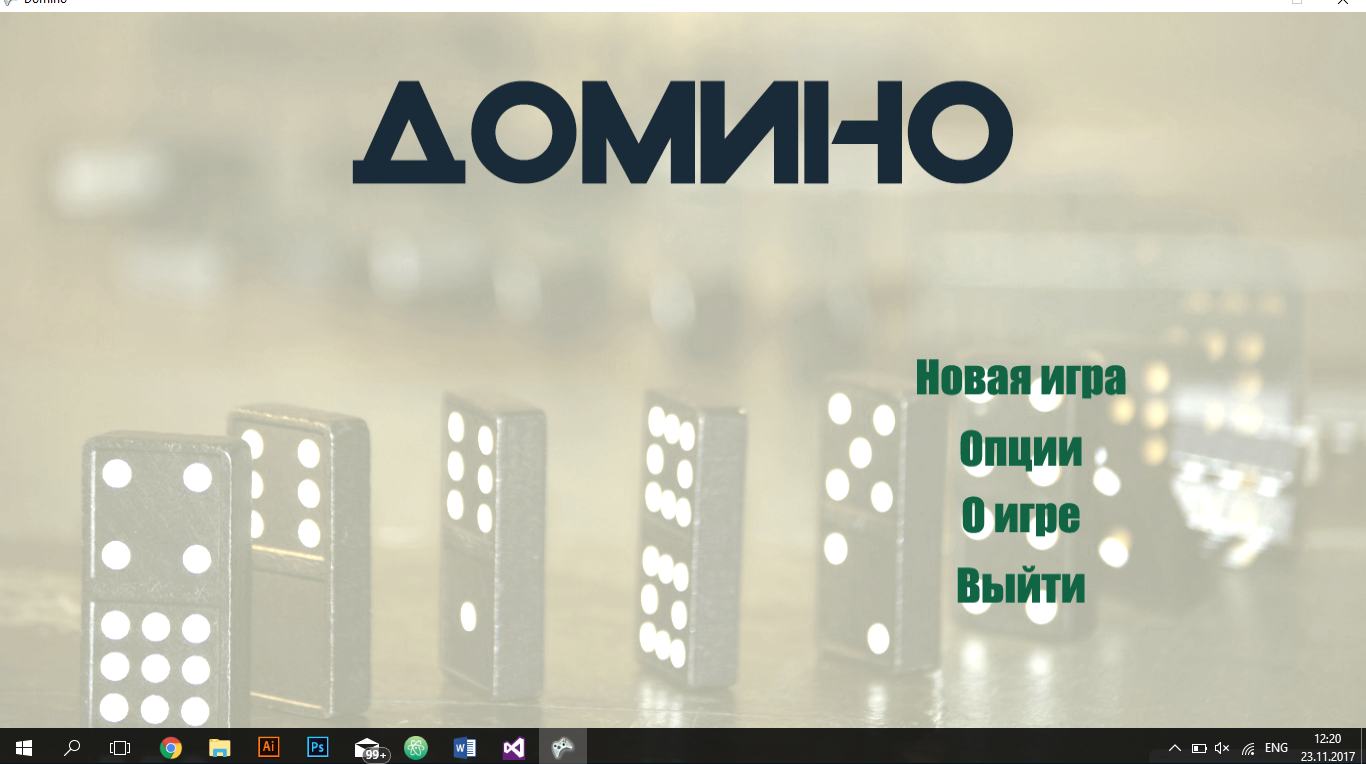


Рисунок 13 – Меню

В пункте меню «Опции» есть возможность выбора таких подпунктов как: «Сложность игры», «Цвет домино», которые позволяют изменить сложность игры и цвет домино.

В пункте меню «Об игре» представлена информация об игре и о ее разработчике.

Заключение

В данной работе была спроектирована и реализована игра, реализующая логическую игру “Домино”.

Было проведено тестирование игры "Домино". Анализ результатов тестирования показал, что система полностью соответствует функциональным требованиям.

В дальнейшем целесообразно добавить в программу возможности игры по сети, увеличение и уменьшения количества игроков, возможность отмены предыдущего действия, а также просмотр домино союзника.

Реализация системы проводилась с использованием инструментальных средств Visual Studio и графической библиотеки XNA. При написании программы было уделено внимание удобству работы пользователя и построению дружественного интерфейса.

# Список использованных источников

1. ГОСТ 7.1. – 2003. Библиографическая запись. Библиографическое описание. Общие требования и правила составления. – М.: ИПК Издательство стандартов, 2004. – 169 с.

2. ГОСТ 7.32 – 2001. Система стандартов по информацию, библиотечному и издательскому делу. Отчет о научно-исследовательской работе. Структура и правила оформления. – М.: ИПК Издательство стандартов, 2001. – 21 с.

3. ГОСТ 7.82 – 2001. Библиографическая запись. Библиографическое описание электронных ресурсов. Общие требования и правила составления. – М.: ИПК Издательство стандартов, 2001. –21 с.

# 5. Andrew Troelsen. Pro C# 5.0 and the .NET 4.5 Framework. – Apress, 2012. – 1534 с.

6. Библиотека XNA – [Электронный ресурс]. – Режим доступа: https://msdn.microsoft.com/en-us/library/bb203894.aspx

Приложение А

Game1.cs

using Domino.Entities;

using Microsoft.Xna.Framework;

using Microsoft.Xna.Framework.Audio;

using Microsoft.Xna.Framework.Design;

using Microsoft.Xna.Framework.GamerServices;

using Microsoft.Xna.Framework.Graphics;

using Microsoft.Xna.Framework.Input;

using Microsoft.Xna.Framework.Storage;

using System;

using System.Collections.Generic;

using System.IO;

using System.Threading;

using System.Xml.Serialization;

namespace Domino {

/// <summary>

/// This is the main type for your game

/// </summary>

public class Game1: Microsoft.Xna.Framework.Game {

GraphicsDeviceManager graphics;

SpriteBatch spriteBatch;

public delegate void DelegateObject(string s);

Color ColorToDraw;

bool PlayerHasWon;

bool PlayerHasLost;

int MaxScore = 2;

// Разрешение экрана

int screenWidth = 1366, screenHeight = 768;

// Пункты главного меню

enum GameState {

MainMenu,

Options,

About,

Playing,

Shuffling,

DifficultyLevel,

TileColor,

EndOfRound

}

GameState CurrentGameState = GameState.MainMenu;

MenuButton btnPlay;

MenuButton btnOptions;

MenuButton btnCredits;

MenuButton btnExit;

MenuButton btnKeepPlaying;

MenuButton btnDifficultyLevel;

MenuButton btnDominoTileColor;

MenuButton btnWhite;

MenuButton btnYellow;

MenuButton btnBlue;

MenuButton btnRed;

MenuButton btnGreen;

MenuButton btnExpert;

MenuButton btnEasy;

MenuButton btnVeryEasy;

MenuButton btnNormal;

MenuButton btnPassTurn;

MenuButton btnNextRound;

Texture2D Congratulations;

Texture2D SorryTryAgain;

bool GameStarted = false;

bool GameLocked;

bool StartOfNewGame = true;

bool StartOfRound = false;

Tile LastTileTaken = null;

Player PlayerWhoLastPlayed;

Vector2 TablePosition; // Позиция доски

Vector2 DraggableSquarePosition; //

Random rand = new Random(); // Рандом

Texture2D Background;

Texture2D EndOfRoundBackground;

// Фонт

SpriteFont FontCent;

// Игроки

Player player1 = new Player("Вы", false, true);

Player player2 = new Player("Игрок 2", false, false);

Player player3 = new Player("Игрок 3", false, false);

Player player4 = new Player("Игрок 4", false, false);

Player PlayerWhoWonLastRound;

bool IsPlayerPassing = false;

bool EndOfRound = false;

Color EndBack = new Color(77, 208, 225, 255);

int TeamOneTotalPoints = 0;

int TeamTwoTotalPoints = 0;

public Table Table1;

Tile LastDominoTilePlayed;

Tile TileToUpdate = null;

public List < Tile > AllTilesList = new List < Tile > ();

public List < Tile > AllTilesListForDealing = new List < Tile > ();

public List < Player > PlayersList = new List < Player > ();

DifficultyLevel CurrentDifficultyLevel = DifficultyLevel.VeryEasy;

TileColor CurrentTileColor = TileColor.White;

#

region Сохранение

StorageDevice device;

string containerName = "MyGamesStorage";

string filename = "mysave.sav";

[Serializable]

public struct SaveGame {

public DifficultyLevel DifficultyLevelToSave;

public TileColor TileColorToSave;

}

#

endregion

Texture2D WhiteSquare; // белый 64х64 квадратик

Vector2 MousePosition; // текущая позиция мыши

Rectangle DraggableSquareBorder;

bool[, ] Table = new bool[32, 21]; // хранит есть ли что то в квадратике

int SquareSize = 28; // размер костяшек

Texture2D OpponentTile;

Texture2D FichaDePareja;

//хранит предыдущее и текущую позицию мыши

MouseState PreviousMouseState, CurrentMouseState;

public Game1() {

graphics = new GraphicsDeviceManager(this);

Content.RootDirectory = "Content";

//позиция верхнего левого угла доски

TablePosition = new Vector2(95, 85);

//позиция квадратика для перемещения

DraggableSquarePosition = new Vector2((graphics.PreferredBackBufferWidth) - 120, 608);

}

protected override void Initialize() {

// TODO: Add your initialization logic here

this.IsMouseVisible = true;

base.Initialize();

}

protected override void LoadContent() {

if (CurrentTileColor == TileColor.White)

ColorToDraw = Color.White;

if (CurrentTileColor == TileColor.Yellow)

ColorToDraw = Color.Yellow;

if (CurrentTileColor == TileColor.Blue)

ColorToDraw = Color.Cyan;

if (CurrentTileColor == TileColor.Red)

ColorToDraw = Color.Red;

if (CurrentTileColor == TileColor.Green)

ColorToDraw = Color.Green;

// Create a new SpriteBatch, which can be used to draw textures.

spriteBatch = new SpriteBatch(GraphicsDevice);

// TODO: use this.Content to load your game content here

// Разрешение экрана

graphics.PreferredBackBufferWidth = screenWidth;

graphics.PreferredBackBufferHeight = screenHeight;

this.graphics.IsFullScreen = false;

graphics.ApplyChanges();

// Main Menu buttons

btnPlay = new MenuButton(Content.Load < Texture2D > (@ "Images\btnNewGame"), graphics.GraphicsDevice);

btnPlay.SetPosition(new Vector2(871, 350));

btnKeepPlaying = new MenuButton(Content.Load < Texture2D > (@ "Images\btnContinuePlaying"), graphics.GraphicsDevice);

btnKeepPlaying.SetPosition(new Vector2(871, 350));

btnOptions = new MenuButton(Content.Load < Texture2D > (@ "Images\btnOptions"), graphics.GraphicsDevice);

btnOptions.SetPosition(new Vector2(871, 420));

btnCredits = new MenuButton(Content.Load < Texture2D > (@ "Images\btnAboutDomino"), graphics.GraphicsDevice);

btnCredits.SetPosition(new Vector2(871, 490));

btnExit = new MenuButton(Content.Load < Texture2D > (@ "Images\btnQuitGame"), graphics.GraphicsDevice);

btnExit.SetPosition(new Vector2(871, 560));

btnDifficultyLevel = new MenuButton(Content.Load < Texture2D > (@ "Images\btnDifficultyLevel"), graphics.GraphicsDevice);

btnDifficultyLevel.SetPosition(new Vector2(842, 330));

btnDominoTileColor = new MenuButton(Content.Load < Texture2D > (@ "Images\btnDominoColor"), graphics.GraphicsDevice);

btnDominoTileColor.SetPosition(new Vector2(842, 640));

btnWhite = new MenuButton(Content.Load < Texture2D > (@ "Images\btnWhite"), graphics.GraphicsDevice);

btnWhite.SetPosition(new Vector2(842, 370));

btnYellow = new MenuButton(Content.Load < Texture2D > (@ "Images\btnYellow"), graphics.GraphicsDevice);

btnYellow.SetPosition(new Vector2(842, 470));

btnBlue = new MenuButton(Content.Load < Texture2D > (@ "Images\btnBlue"), graphics.GraphicsDevice);

btnBlue.SetPosition(new Vector2(842, 520));

btnRed = new MenuButton(Content.Load < Texture2D > (@ "Images\btnRed"), graphics.GraphicsDevice);

btnRed.SetPosition(new Vector2(842, 570));

btnGreen = new MenuButton(Content.Load < Texture2D > (@ "Images\btnGreen"), graphics.GraphicsDevice);

btnGreen.SetPosition(new Vector2(842, 420));

btnVeryEasy = new MenuButton(Content.Load < Texture2D > (@ "Images\btnVeryEasy"), graphics.GraphicsDevice);

btnVeryEasy.SetPosition(new Vector2(842, 410));

btnEasy = new MenuButton(Content.Load < Texture2D > (@ "Images\btnEasy"), graphics.GraphicsDevice);

btnEasy.SetPosition(new Vector2(842, 460));

btnNormal = new MenuButton(Content.Load < Texture2D > (@ "Images\btnNormal"), graphics.GraphicsDevice);

btnNormal.SetPosition(new Vector2(842, 510));

btnExpert = new MenuButton(Content.Load < Texture2D > (@ "Images\btnExpert"), graphics.GraphicsDevice);

btnExpert.SetPosition(new Vector2(842, 560));

btnPassTurn = new MenuButton(Content.Load < Texture2D > (@ "Images\btnPass"), graphics.GraphicsDevice, 1);

btnPassTurn.SetPosition(new Vector2(700, 710));

btnNextRound = new MenuButton(Content.Load < Texture2D > (@ "Images\btnNextRound"), graphics.GraphicsDevice, 1);

btnNextRound.SetPosition(new Vector2(700, 400));

// Квадратик

WhiteSquare = Content.Load < Texture2D > (@ "Images\white\_64x64");

// Грифт

FontCent = Content.Load < SpriteFont > (@ "Font\Cent");

DraggableSquareBorder = new Rectangle((int) DraggableSquarePosition.X, (int) DraggableSquarePosition.Y, SquareSize, SquareSize);

Background = Content.Load < Texture2D > (@ "Images\ExtendedBoard");

EndOfRoundBackground = Content.Load < Texture2D > (@ "Images\EndOfRoundBackground");

Congratulations = Content.Load < Texture2D > (@ "Images\Congratulations");

SorryTryAgain = Content.Load < Texture2D > (@ "Images\BadLuckTryAgain");

OpponentTile = Content.Load < Texture2D > (@ "Images\OpponentTile");

FichaDePareja = Content.Load < Texture2D > (@ "Images\TeammateTile");

#

region Все костяшки

AllTilesList.Add(new Tile(Content.Load < Texture2D > (@ "Images/00"),

10, Vector2.Zero, new Vector2(150, 150), 0, 0, true, true, false));

AllTilesList.Add(new Tile(Content.Load < Texture2D > (@ "Images/01"),

10, Vector2.Zero, new Vector2(150, 150), 0, 1, true, false, false));

AllTilesList.Add(new Tile(Content.Load < Texture2D > (@ "Images/02"),

10, Vector2.Zero, new Vector2(150, 150), 0, 2, true, false, false));

AllTilesList.Add(new Tile(Content.Load < Texture2D > (@ "Images/03"),

10, Vector2.Zero, new Vector2(150, 150), 0, 3, true, false, false));

AllTilesList.Add(new Tile(Content.Load < Texture2D > (@ "Images/04"),

10, Vector2.Zero, new Vector2(150, 150), 0, 4, true, false, false));

AllTilesList.Add(new Tile(Content.Load < Texture2D > (@ "Images/05"),

10, Vector2.Zero, new Vector2(150, 150), 0, 5, true, false, false));

AllTilesList.Add(new Tile(Content.Load < Texture2D > (@ "Images/06"),

10, Vector2.Zero, new Vector2(150, 150), 0, 6, true, false, false));

AllTilesList.Add(new Tile(Content.Load < Texture2D > (@ "Images/11"),

10, Vector2.Zero, new Vector2(150, 150), 1, 1, true, true, false));

AllTilesList.Add(new Tile(Content.Load < Texture2D > (@ "Images/12"),

10, Vector2.Zero, new Vector2(150, 150), 1, 2, true, false, false));

AllTilesList.Add(new Tile(Content.Load < Texture2D > (@ "Images/13"),

10, Vector2.Zero, new Vector2(150, 150), 1, 3, true, false, false));

AllTilesList.Add(new Tile(Content.Load < Texture2D > (@ "Images/14"),

10, Vector2.Zero, new Vector2(150, 150), 1, 4, true, false, false));

AllTilesList.Add(new Tile(Content.Load < Texture2D > (@ "Images/15"),

10, Vector2.Zero, new Vector2(150, 150), 1, 5, true, false, false));

AllTilesList.Add(new Tile(Content.Load < Texture2D > (@ "Images/16"),

10, Vector2.Zero, new Vector2(150, 150), 1, 6, true, false, false));

AllTilesList.Add(new Tile(Content.Load < Texture2D > (@ "Images/22"),

10, Vector2.Zero, new Vector2(150, 150), 2, 2, true, true, false));

AllTilesList.Add(new Tile(Content.Load < Texture2D > (@ "Images/23"),

10, Vector2.Zero, new Vector2(150, 150), 2, 3, true, false, false));

AllTilesList.Add(new Tile(Content.Load < Texture2D > (@ "Images/24"),

10, Vector2.Zero, new Vector2(150, 150), 2, 4, true, false, false));

AllTilesList.Add(new Tile(Content.Load < Texture2D > (@ "Images/25"),

10, Vector2.Zero, new Vector2(150, 150), 2, 5, true, false, false));

AllTilesList.Add(new Tile(Content.Load < Texture2D > (@ "Images/26"),

10, Vector2.Zero, new Vector2(150, 150), 2, 6, true, false, false));

AllTilesList.Add(new Tile(Content.Load < Texture2D > (@ "Images/33"),

10, Vector2.Zero, new Vector2(150, 150), 3, 3, true, true, false));

AllTilesList.Add(new Tile(Content.Load < Texture2D > (@ "Images/34"),

10, Vector2.Zero, new Vector2(150, 150), 3, 4, true, false, false));

AllTilesList.Add(new Tile(Content.Load < Texture2D > (@ "Images/35"),

10, Vector2.Zero, new Vector2(150, 150), 3, 5, true, false, false));

AllTilesList.Add(new Tile(Content.Load < Texture2D > (@ "Images/36"),

10, Vector2.Zero, new Vector2(150, 150), 3, 6, true, false, false));

AllTilesList.Add(new Tile(Content.Load < Texture2D > (@ "Images/44"),

10, Vector2.Zero, new Vector2(180, 150), 4, 4, true, true, false));

AllTilesList.Add(new Tile(Content.Load < Texture2D > (@ "Images/45"),

10, Vector2.Zero, new Vector2(150, 150), 4, 5, true, false, false));

AllTilesList.Add(new Tile(Content.Load < Texture2D > (@ "Images/46"),

10, Vector2.Zero, new Vector2(150, 150), 4, 6, true, false, false));

AllTilesList.Add(new Tile(Content.Load < Texture2D > (@ "Images/55"),

10, Vector2.Zero, new Vector2(150, 150), 5, 5, true, true, false));

AllTilesList.Add(new Tile(Content.Load < Texture2D > (@ "Images/56"),

10, Vector2.Zero, new Vector2(150, 150), 5, 6, true, false, false));

AllTilesList.Add(new Tile(Content.Load < Texture2D > (@ "Images/66"),

10, Vector2.Zero, new Vector2(150, 150), 6, 6, true, true, false));

#

endregion

PlayersList.Add(player1);

PlayersList.Add(player2);

PlayersList.Add(player3);

PlayersList.Add(player4);

Table1 = new Table(1);

base.LoadContent();

}

protected override void UnloadContent() {

// TODO: Unload any non ContentManager content here

}

protected override void Update(GameTime gameTime) {

// Allows the game to exit

if (GamePad.GetState(PlayerIndex.One).Buttons.Back == ButtonState.Pressed)

this.Exit();

// Мышка

CurrentMouseState = Mouse.GetState();

MousePosition = new Vector2(CurrentMouseState.X, CurrentMouseState.Y);

// Пункты меню

switch (CurrentGameState) {

case GameState.MainMenu:

if (btnPlay.isClicked == true)

CurrentGameState = GameState.Playing;

btnPlay.Update(CurrentMouseState);

if (btnKeepPlaying.isClicked == true)

CurrentGameState = GameState.Playing;

if (btnOptions.isClicked == true)

CurrentGameState = GameState.Options;

btnOptions.Update(CurrentMouseState);

if (btnCredits.isClicked == true)

CurrentGameState = GameState.About;

btnCredits.Update(CurrentMouseState);

if (btnExit.isClicked == true) {

this.Exit();

}

btnExit.Update(CurrentMouseState);

btnPlay.Update(CurrentMouseState);

btnOptions.Update(CurrentMouseState);

btnCredits.Update(CurrentMouseState);

btnExit.Update(CurrentMouseState);

break;

case GameState.Playing:

// TODO fix:

// Закончит игру на Enter

if (Keyboard.GetState().IsKeyDown(Keys.Enter))

CurrentGameState = GameState.EndOfRound;

if (StartOfNewGame || StartOfRound) {

foreach(Player j in PlayersList) {

j.PlayerTileList.Clear();

}

AllTilesListForDealing.AddRange(AllTilesList);

DealDominoTilesToPlayers(AllTilesListForDealing);

// У кого есть домино с 6 ходит первым

if (StartOfNewGame) {

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

foreach(Tile f in PlayersList[i].PlayerTileList) {

if (f.FirstTileValue == 6 && f.SecondTileValue == 6) {

PlayersList[i].MyTurn = true;

Table1.PlayerInTurn = PlayersList[i];

}

}

}

if (!Table1.PlayerInTurn.IsHuman) {

for (int i = 0; i < Table1.PlayerInTurn.PlayerTileList.Count; i++) {

if (StartOfNewGame) {

if (Table1.PlayerInTurn.PlayerTileList[i].FirstTileValue == 6 && Table1.PlayerInTurn.PlayerTileList[i].SecondTileValue == 6) {

Table1.PlayerInTurn.PlayerTileList[i].Position = new Vector2(TablePosition.X + 16 \* SquareSize, TablePosition.Y + 10 \* SquareSize);

StartOfNewGame = false;

TilePlacementLogic(Table1, Table1.PlayerInTurn.PlayerTileList[i], Table1.PlayerInTurn, i);

break;

}

}

}

CalculateTurn(Table1.PlayerInTurn);

}

} else if (StartOfRound) {

foreach(Player j in PlayersList) {

if (j.MyTurn && !j.IsHuman) {

Table1.PlayerInTurn = j;

j.PlayerTileList[0].Position = new Vector2(TablePosition.X + 16 \* SquareSize, TablePosition.Y + 10 \* SquareSize);

TilePlacementLogic(Table1, j.PlayerTileList[0], j, 0);

CalculateTurn(Table1.PlayerInTurn);

break;

}

}

StartOfRound = false;

}

}

#

region AI

// AI

switch (CurrentDifficultyLevel) {

case DifficultyLevel.VeryEasy:

if (!Table1.PlayerInTurn.IsHuman && (!EndOfRound) && !StartOfNewGame && !StartOfRound) {

DrawTilesForNonHumanPlayersLevel1(Table1.PlayerInTurn);

Thread.Sleep(2000);

CalculateTurn(Table1.PlayerInTurn);

}

break;

case DifficultyLevel.Easy:

if (!Table1.PlayerInTurn.IsHuman && (!EndOfRound) && !StartOfNewGame && !StartOfRound) {

DrawTilesForNonHumanPlayersLevel1(Table1.PlayerInTurn);

Thread.Sleep(2000);

CalculateTurn(Table1.PlayerInTurn);

}

break;

case DifficultyLevel.Normal:

if (!Table1.PlayerInTurn.IsHuman && (!EndOfRound) && !StartOfNewGame && !StartOfRound) {

DrawTilesForNonHumanPlayersLevel3(Table1.PlayerInTurn);

Thread.Sleep(2000);

CalculateTurn(Table1.PlayerInTurn);

}

break;

case DifficultyLevel.Expert:

if (!Table1.PlayerInTurn.IsHuman && (!EndOfRound) && !StartOfNewGame && !StartOfRound) {

DrawTilesForNonHumanPlayersLevel3(Table1.PlayerInTurn);

Thread.Sleep(2000);

CalculateTurn(Table1.PlayerInTurn);

}

break;

}

if (Table1.PlayerInTurn.IsHuman) {

FindOutIfPlayerPassesHisTurn(Table1.PlayerInTurn);

}

#

endregion

# region Update to drag tiles

foreach(Tile f in player1.PlayerTileList) {

if (PreviousMouseState.LeftButton == ButtonState.Released && CurrentMouseState.LeftButton == ButtonState.Pressed && f.TileEdge.Contains((int) MousePosition.X, (int) MousePosition.Y)) {

f.IsTileBeingDragged = true;

LastTileTaken = f;

}

// if the user just released the mousebutton - set IsTileBeingDragged to false, and check if we should add the square to the board

if (PreviousMouseState.LeftButton == ButtonState.Pressed && CurrentMouseState.LeftButton == ButtonState.Released) {

f.IsTileBeingDragged = false;

Vector2 Square = GetSquareFromMousePosition(MousePosition);

Rectangle rectFirstTileToPlay = new Rectangle((int)(TablePosition.X + 14 \* SquareSize), (int)(TablePosition.Y + 8 \* SquareSize), 5 \* SquareSize, 5 \* SquareSize);

if (Table1.TilesPlayedOnTableList.Count < 1) {

// if the mousebutton was released inside the board

if (IsMouseInsideBoard() && (rectFirstTileToPlay.Contains((int) MousePosition.X, (int) MousePosition.Y)) &&

Table1.PlayerInTurn.IsHuman) {

// find out which square the mouse is over

// and set that square to true (has a piece)

Table[(int) Square.X, (int) Square.Y] = true;

}

} else {

Rectangle RecFichaExtremoDerecho = new Rectangle((int) Table1.PositionOfRightHandSideEdge.X, (int) Table1.PositionOfRightHandSideEdge.Y, SquareSize, SquareSize);

Rectangle RecFichaExtremoIzquierdo = new Rectangle((int) Table1.PositionOfLeftHandSideEdge.X, (int) Table1.PositionOfLeftHandSideEdge.Y, SquareSize, SquareSize);

if (RecFichaExtremoDerecho.Contains((int) MousePosition.X, (int) MousePosition.Y) || RecFichaExtremoIzquierdo.Contains((int) MousePosition.X, (int) MousePosition.Y)) {

// find out which square the mouse is over

// and set that square to true (has a piece)

try {

Table[(int) Square.X, (int) Square.Y] = true;

} catch (Exception e) {

throw e;

}

}

}

}

}

#

endregion

btnPlay.Update(CurrentMouseState);

if (Keyboard.GetState().IsKeyDown(Keys.Escape)) {

GameStarted = true;

btnKeepPlaying.Update(CurrentMouseState);

CurrentGameState = GameState.MainMenu;

}

if (btnPassTurn.isClicked == true) {

CalculateTurn(Table1.PlayerInTurn);

}

btnPassTurn.Update(CurrentMouseState);

VerifyEndOfRoundCondition();

break;

case GameState.EndOfRound:

if (TeamOneTotalPoints > MaxScore) {

PlayerHasWon = true;

} else if (TeamTwoTotalPoints > MaxScore) {

PlayerHasLost = true;

}

if (btnNextRound.isClicked == true) {

StartOfRound = true;

Table1.TilesPlayedOnTableList.Clear();

foreach(Player j in PlayersList) {

j.PlayerTileList.Clear();

}

EndOfRound = false;

GameLocked = false;

CurrentGameState = GameState.Playing;

}

btnNextRound.Update(CurrentMouseState);

break;

case GameState.Options:

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

CurrentGameState = GameState.MainMenu;

if (btnDifficultyLevel.isClicked == true) {

CurrentGameState = GameState.DifficultyLevel;

}

if (btnDominoTileColor.isClicked == true) {

CurrentGameState = GameState.TileColor;

}

btnDominoTileColor.Update(CurrentMouseState);

btnDominoTileColor.Update(CurrentMouseState);

btnDifficultyLevel.Update(CurrentMouseState);

btnDifficultyLevel.Update(CurrentMouseState);

break;

case GameState.About:

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

CurrentGameState = GameState.MainMenu;

break;

case GameState.DifficultyLevel:

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

CurrentGameState = GameState.Options;

if (btnVeryEasy.isClicked == true) {

CurrentDifficultyLevel = DifficultyLevel.VeryEasy;

}

if (btnEasy.isClicked == true) {

CurrentDifficultyLevel = DifficultyLevel.Easy;

}

if (btnNormal.isClicked == true) {

CurrentDifficultyLevel = DifficultyLevel.Normal;

}

if (btnExpert.isClicked == true) {

CurrentDifficultyLevel = DifficultyLevel.Expert;

}

btnVeryEasy.Update(CurrentMouseState);

btnVeryEasy.Update(CurrentMouseState);

btnEasy.Update(CurrentMouseState);

btnEasy.Update(CurrentMouseState);

btnNormal.Update(CurrentMouseState);

btnNormal.Update(CurrentMouseState);

btnExpert.Update(CurrentMouseState);

btnExpert.Update(CurrentMouseState);

break;

case GameState.TileColor:

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

CurrentGameState = GameState.Options;

if (btnWhite.isClicked == true) {

CurrentTileColor = TileColor.White;

ColorToDraw = Color.White;

}

if (btnYellow.isClicked == true) {

CurrentTileColor = TileColor.Yellow;

ColorToDraw = Color.Yellow;

}

if (btnBlue.isClicked == true) {

CurrentTileColor = TileColor.Blue;

ColorToDraw = Color.Cyan;

}

if (btnRed.isClicked == true) {

CurrentTileColor = TileColor.Red;

ColorToDraw = Color.Red;

}

if (btnGreen.isClicked == true) {

CurrentTileColor = TileColor.Green;

ColorToDraw = Color.DarkGreen;

}

btnWhite.Update(CurrentMouseState);

btnWhite.Update(CurrentMouseState);

btnYellow.Update(CurrentMouseState);

btnYellow.Update(CurrentMouseState);

btnBlue.Update(CurrentMouseState);

btnBlue.Update(CurrentMouseState);

btnRed.Update(CurrentMouseState);

btnRed.Update(CurrentMouseState);

btnGreen.Update(CurrentMouseState);

btnGreen.Update(CurrentMouseState);

break;

}

PreviousMouseState = CurrentMouseState;

base.Update(gameTime);

}

protected override void Draw(GameTime gameTime) {

// Меню

switch (CurrentGameState) {

case GameState.MainMenu:

//белый фон

GraphicsDevice.Clear(Color.White);

spriteBatch.Begin();

spriteBatch.Draw(Content.Load < Texture2D > (@ "Images\MainMenu"),

new Rectangle(0, 0, screenWidth, screenHeight),

Color.White);

// Замена кнопки играть на продолжить игру

if (!GameStarted) {

btnPlay.Draw(spriteBatch);

} else btnKeepPlaying.Draw(spriteBatch);

btnOptions.Draw(spriteBatch);

btnCredits.Draw(spriteBatch);

btnExit.Draw(spriteBatch);

//

spriteBatch.End();

break;

case GameState.Playing:

// Фон

GraphicsDevice.Clear(Color.LightGreen);

// Рисовать всё

spriteBatch.Begin(SpriteSortMode.FrontToBack, BlendState.AlphaBlend);

DrawText(); // Рисовать текст

DrawBoard(); // Рисвоать доску

DrawTilesOnBoard(); //

DrawDraggableSquare(); //

foreach(Tile s in player1.PlayerTileList) // Draw player's tiles except the one he's dragging

if (!s.IsTileBeingDragged) {

spriteBatch.Draw(s.Image,

s.Position, null,

ColorToDraw, 0, Vector2.Zero,

.07 f, SpriteEffects.None, 1);

}

#

region Отрисовка домино других игроков

// Нарисовать домино 2 игрока (right hand side)

Vector2 Player2InitialPosition = new Vector2(1047, 270);

for (int i = 0; i < player2.PlayerTileList.Count; i++) {

spriteBatch.Draw(OpponentTile, Player2InitialPosition, null,

ColorToDraw, 0, Vector2.Zero,

.07 f, SpriteEffects.None, .5 f);

Player2InitialPosition = new Vector2(Player2InitialPosition.X, Player2InitialPosition.Y + 28.8 f);

}

// Нарисовать домино 3 игрока (teammate)

Vector2 Player3InitialPosition = new Vector2(449, 32);

for (int i = 0; i < player3.PlayerTileList.Count; i++) {

spriteBatch.Draw(FichaDePareja, Player3InitialPosition, null,

ColorToDraw, 0, Vector2.Zero,

.07 f, SpriteEffects.None, .5 f);

Player3InitialPosition = new Vector2(Player3InitialPosition.X + 28.8 f, Player3InitialPosition.Y);

}

// Нарисовать домино 4 игрока (left hand side)

Vector2 Player4InitialPosition = new Vector2(40, 272);

for (int i = 0; i < player4.PlayerTileList.Count; i++) {

spriteBatch.Draw(OpponentTile, Player4InitialPosition, null,

ColorToDraw, 0, Vector2.Zero,

.07 f, SpriteEffects.None, .5 f);

Player4InitialPosition = new Vector2(Player4InitialPosition.X, Player4InitialPosition.Y + 28.8 f);

}#

endregion

// Фон

spriteBatch.Draw(Background,

new Rectangle(0, 0, Window.ClientBounds.Width,

Window.ClientBounds.Height), null,

Color.White, 0, Vector2.Zero,

SpriteEffects.None, 0.0 f);

if (Table1.PlayerInTurn.IsHuman && IsPlayerPassing && Table1.PlayerInTurn.PlayerTileList.Count > 0) {

btnPassTurn.Draw(spriteBatch, 1 f);

}

// Нарисовать чей ход

spriteBatch.DrawString(FontCent, "Ход:", new Vector2(1162, 720), Color.White);

foreach(Player j in PlayersList) {

if (j.MyTurn)

spriteBatch.DrawString(FontCent, j.Name, new Vector2(1232, 720), Color.White);

}

// End drawing

spriteBatch.End();

break;

case GameState.EndOfRound:

String TeamOnePoints = TeamOneTotalPoints.ToString();

String TeamTwoPoints = TeamTwoTotalPoints.ToString();

GraphicsDevice.Clear(Color.FromNonPremultiplied(77, 208, 225, 255));

spriteBatch.Begin();

DelegateObject myDelegate = n => {

string s = n;spriteBatch.DrawString(FontCent, s, new Vector2(570, 300), Color.Black, 0 f, Vector2.Zero, 1 f, SpriteEffects.None, .99 f);

};

DelegateObject myDelegate2 = j => {

string s = j;spriteBatch.DrawString(FontCent, s, new Vector2(702, 300), Color.Black, 0 f, Vector2.Zero, 1 f, SpriteEffects.None, .99 f);

};

myDelegate(TeamOnePoints);

myDelegate2(TeamTwoPoints);

spriteBatch.Draw(EndOfRoundBackground, new Vector2(110, 60), null, Color.White, 0 f, Vector2.Zero, .9 f, SpriteEffects.None, .95 f);

if (PlayerHasWon) {

spriteBatch.Draw(Congratulations, new Vector2(144, 500), null, Color.White, 0 f, Vector2.Zero, .9 f, SpriteEffects.None, 1 f);

}

if (PlayerHasLost) {

spriteBatch.Draw(SorryTryAgain, new Vector2(144, 500), null, Color.White, 0 f, Vector2.Zero, .9 f, SpriteEffects.None, 1 f);

}

if (GameLocked) {

spriteBatch.DrawString(FontCent, "Победил: " + PlayerWhoLastPlayed.Name, new Vector2(250, 100), Color.DarkGreen, 0 f, Vector2.Zero, .7 f, SpriteEffects.None, .99 f);

} else {

spriteBatch.DrawString(FontCent, "Конец раунда", new Vector2(220, 100), Color.DarkGreen, 0 f, Vector2.Zero, 1 f, SpriteEffects.None, .99 f);

}

spriteBatch.DrawString(FontCent, player1.Name, new Vector2(170, 190), Color.DarkGreen, 0 f, Vector2.Zero, 1 f, SpriteEffects.None, .99 f);

spriteBatch.DrawString(FontCent, player2.Name, new Vector2(170, 250), Color.DarkGreen, 0 f, Vector2.Zero, 1 f, SpriteEffects.None, .99 f);

spriteBatch.DrawString(FontCent, player3.Name, new Vector2(170, 310), Color.DarkGreen, 0 f, Vector2.Zero, 1 f, SpriteEffects.None, .99 f);

spriteBatch.DrawString(FontCent, player4.Name, new Vector2(170, 370), Color.DarkGreen, 0 f, Vector2.Zero, 1 f, SpriteEffects.None, .99 f);

spriteBatch.DrawString(FontCent, "ОЧКИ", new Vector2(580, 200), Color.Black, 0 f, Vector2.Zero, 1 f, SpriteEffects.None, .99 f);

spriteBatch.DrawString(FontCent, "Команда 1: ", new Vector2(530, 240), Color.Black, 0 f, Vector2.Zero, 1 f, SpriteEffects.None, .99 f);

spriteBatch.DrawString(FontCent, player1.Name, new Vector2(530, 260), Color.Black, 0 f, Vector2.Zero, 1 f, SpriteEffects.None, .99 f);

spriteBatch.DrawString(FontCent, player3.Name, new Vector2(530, 280), Color.Black, 0 f, Vector2.Zero, 1 f, SpriteEffects.None, .99 f);

spriteBatch.DrawString(FontCent, TeamOnePoints, new Vector2(570, 300), Color.Black, 0 f, Vector2.Zero, 1 f, SpriteEffects.None, .99 f);

spriteBatch.DrawString(FontCent, "Команда 2: ", new Vector2(662, 240), Color.Black, 0 f, Vector2.Zero, 1 f, SpriteEffects.None, .99 f);

spriteBatch.DrawString(FontCent, player2.Name, new Vector2(662, 260), Color.Black, 0 f, Vector2.Zero, 1 f, SpriteEffects.None, .99 f);

spriteBatch.DrawString(FontCent, player4.Name, new Vector2(662, 280), Color.Black, 0 f, Vector2.Zero, 1 f, SpriteEffects.None, .99 f);

spriteBatch.DrawString(FontCent, TeamTwoPoints, new Vector2(702, 300), Color.Black, 0 f, Vector2.Zero, 1 f, SpriteEffects.None, .99 f);

btnNextRound.Draw(spriteBatch, 1 f);

Vector2 Player1EndOfRoundTilePosition = new Vector2(300, 180);

foreach(Tile s in player1.PlayerTileList) // Нарисовать домино

{

s.LastOrientation = 1;

spriteBatch.Draw(s.Image,

Player1EndOfRoundTilePosition, null,

ColorToDraw, 0, Vector2.Zero,

.07 f, SpriteEffects.None, 1);

Player1EndOfRoundTilePosition = new Vector2(Player1EndOfRoundTilePosition.X + 28.8 f, Player1EndOfRoundTilePosition.Y);

}

Vector2 Player2EndOfRoundTilePosition = new Vector2(300, 240);

foreach(Tile s in player2.PlayerTileList) // Нарисовать домино

{

s.LastOrientation = 1;

spriteBatch.Draw(s.Image,

Player2EndOfRoundTilePosition, null,

ColorToDraw, 0, Vector2.Zero,

.07 f, SpriteEffects.None, 1);

Player2EndOfRoundTilePosition = new Vector2(Player2EndOfRoundTilePosition.X + 28.8 f, Player2EndOfRoundTilePosition.Y);

}

Vector2 Player3EndOfRoundTilePosition = new Vector2(300, 300);

foreach(Tile s in player3.PlayerTileList) // Нарисовать домино

{

s.LastOrientation = 1;

spriteBatch.Draw(s.Image,

Player3EndOfRoundTilePosition, null,

ColorToDraw, 0, Vector2.Zero,

.07 f, SpriteEffects.None, 1);

Player3EndOfRoundTilePosition = new Vector2(Player3EndOfRoundTilePosition.X + 28.8 f, Player3EndOfRoundTilePosition.Y);

}

Vector2 Player4EndOfRoundTilePosition = new Vector2(300, 360);

foreach(Tile s in player4.PlayerTileList) // Нарисовать домино

{

s.LastOrientation = 1;

spriteBatch.Draw(s.Image,

Player4EndOfRoundTilePosition, null,

ColorToDraw, 0, Vector2.Zero,

.07 f, SpriteEffects.None, 1);

Player4EndOfRoundTilePosition = new Vector2(Player4EndOfRoundTilePosition.X + 28.8 f, Player4EndOfRoundTilePosition.Y);

}

spriteBatch.End();

break;

case GameState.Options:

GraphicsDevice.Clear(Color.Red);

spriteBatch.Begin();

spriteBatch.Draw(Content.Load < Texture2D > (@ "Images\optionsBackground"),

new Rectangle(0, 0, screenWidth, screenHeight),

Color.White);

btnDifficultyLevel.Draw(spriteBatch);

btnDominoTileColor.Draw(spriteBatch);

spriteBatch.End();

break;

case GameState.About:

spriteBatch.Begin();

spriteBatch.Draw(Content.Load < Texture2D > (@ "Images\aboutDominoBackground"),

new Rectangle(0, 0, screenWidth, screenHeight),

Color.White);

spriteBatch.DrawString(FontCent, "Игра", new Vector2(621, 500), Color.DarkGreen);

spriteBatch.DrawString(FontCent, "Домино", new Vector2(621, 540), Color.DarkGreen);

spriteBatch.DrawString(FontCent, "Разработал: Павел Семенов", new Vector2(621, 580), Color.DarkGreen);

spriteBatch.DrawString(FontCent, "Нажмите Escape чтобы продолжить...", new Vector2(945, 700), Color.DarkGreen, 0, Vector2.Zero, .7 f, SpriteEffects.None, 1);

spriteBatch.End();

break;

case GameState.DifficultyLevel:

spriteBatch.Begin();

spriteBatch.Draw(Content.Load < Texture2D > (@ "Images\optionsBackground"),

new Rectangle(0, 0, screenWidth, screenHeight),

Color.White);

if (CurrentDifficultyLevel == DifficultyLevel.VeryEasy) {

btnVeryEasy.Draw(spriteBatch, Color.Black);

btnEasy.Draw(spriteBatch);

btnNormal.Draw(spriteBatch);

btnExpert.Draw(spriteBatch);

}

if (CurrentDifficultyLevel == DifficultyLevel.Easy) {

btnEasy.Draw(spriteBatch, Color.Black);

btnVeryEasy.Draw(spriteBatch);

btnNormal.Draw(spriteBatch);

btnExpert.Draw(spriteBatch);

}

if (CurrentDifficultyLevel == DifficultyLevel.Normal) {

btnNormal.Draw(spriteBatch, Color.Black);

btnVeryEasy.Draw(spriteBatch);

btnEasy.Draw(spriteBatch);

btnExpert.Draw(spriteBatch);

}

if (CurrentDifficultyLevel == DifficultyLevel.Expert) {

btnExpert.Draw(spriteBatch, Color.Black);

btnVeryEasy.Draw(spriteBatch);

btnEasy.Draw(spriteBatch);

btnNormal.Draw(spriteBatch);

}

spriteBatch.End();

break;

case GameState.TileColor:

spriteBatch.Begin();

spriteBatch.Draw(Content.Load < Texture2D > (@ "Images\optionsBackground"),

new Rectangle(0, 0, screenWidth, screenHeight),

Color.White);

if (CurrentTileColor == TileColor.White) {

btnWhite.Draw(spriteBatch, Color.Black);

btnYellow.Draw(spriteBatch);

btnBlue.Draw(spriteBatch);

btnRed.Draw(spriteBatch);

btnGreen.Draw(spriteBatch);

}

if (CurrentTileColor == TileColor.Yellow) {

btnWhite.Draw(spriteBatch);

btnYellow.Draw(spriteBatch, Color.Black);

btnBlue.Draw(spriteBatch);

btnRed.Draw(spriteBatch);

btnGreen.Draw(spriteBatch);

}

if (CurrentTileColor == TileColor.Blue) {

btnWhite.Draw(spriteBatch);

btnYellow.Draw(spriteBatch);

btnBlue.Draw(spriteBatch, Color.Black);

btnRed.Draw(spriteBatch);

btnGreen.Draw(spriteBatch);

}

if (CurrentTileColor == TileColor.Red) {

btnWhite.Draw(spriteBatch);

btnYellow.Draw(spriteBatch);

btnBlue.Draw(spriteBatch);

btnRed.Draw(spriteBatch, Color.Black);

btnGreen.Draw(spriteBatch);

}

if (CurrentTileColor == TileColor.Green) {

btnWhite.Draw(spriteBatch);

btnYellow.Draw(spriteBatch);

btnBlue.Draw(spriteBatch);

btnRed.Draw(spriteBatch);

btnGreen.Draw(spriteBatch, Color.Black);

}

spriteBatch.End();

break;

}

base.Draw(gameTime);

}

#

region Методы рисования

private void DrawText() {

String PuntosDeEquipo1 = TeamOneTotalPoints.ToString();

String PuntosDeEquipo2 = TeamTwoTotalPoints.ToString();

spriteBatch.DrawString(FontCent, "ДОМИНО", new Vector2(50, 20), Color.White);

spriteBatch.DrawString(FontCent, "ОЧКИ:", new Vector2(1150, 100), Color.White);

spriteBatch.DrawString(FontCent, "Команда 1: ", new Vector2(1100, 140), Color.White);

spriteBatch.DrawString(FontCent, player1.Name, new Vector2(1100, 160), Color.White);

spriteBatch.DrawString(FontCent, player3.Name, new Vector2(1100, 180), Color.White);

spriteBatch.DrawString(FontCent, PuntosDeEquipo1, new Vector2(1140, 200), Color.White);

spriteBatch.DrawString(FontCent, "Команда 2: ", new Vector2(1232, 140), Color.White);

spriteBatch.DrawString(FontCent, player2.Name, new Vector2(1232, 160), Color.White);

spriteBatch.DrawString(FontCent, player4.Name, new Vector2(1232, 180), Color.White);

spriteBatch.DrawString(FontCent, PuntosDeEquipo2, new Vector2(1272, 200), Color.White);

}

private void DrawDraggableSquare() {

foreach(Tile h in player1.PlayerTileList) {

if (h.IsTileBeingDragged) {

spriteBatch.Draw(h.Image, new Rectangle((int)(MousePosition.X - WhiteSquare.Width / 4), (int)(MousePosition.Y - WhiteSquare.Height / 4), 30, 60), null, ColorToDraw, 0, Vector2.Zero, SpriteEffects.None, 1 f);

}

}

}

// Рисовать игровое прое

private void DrawBoard() {

float Opacity = 1;

Color ColorToUse = Color.White;

Rectangle PositionToDrawSquare = new Rectangle();

//

for (int x = 0; x < Table.GetLength(0); x++) {

//

for (int y = 0; y < Table.GetLength(1); y++) {

PositionToDrawSquare = new Rectangle((int)(x \* SquareSize + TablePosition.X), (int)(y \* SquareSize + TablePosition.Y), SquareSize, SquareSize);

if ((x + y) % 2 == 0) {

Opacity = .15 f;

} else {

Opacity = .1 f;

}

if (IsMouseInsideBoard() && IsMouseCoordinateWithinBoard(x, y)) {

ColorToUse = Color.Red;

Opacity = .5 f;

} else {

ColorToUse = Color.White;

}

spriteBatch.Draw(WhiteSquare, PositionToDrawSquare, null, ColorToUse \* Opacity, 0, Vector2.Zero, SpriteEffects.None, 0);

if (Table[x, y]) {

Rectangle recTileRightHandSide = new Rectangle((int) Table1.PositionOfRightHandSideEdge.X, (int) Table1.PositionOfRightHandSideEdge.Y, SquareSize, SquareSize);

Rectangle recTileLeftHandSide = new Rectangle((int) Table1.PositionOfLeftHandSideEdge.X, (int) Table1.PositionOfLeftHandSideEdge.Y, SquareSize, SquareSize);

if (((LastTileTaken.FirstTileValue == Table1.RightHandSide) || (LastTileTaken.SecondTileValue == Table1.RightHandSide) || (StartOfNewGame) || (StartOfRound)) && (recTileRightHandSide.Contains((int) MousePosition.X, (int) MousePosition.Y))) {

TileToUpdate = LastTileTaken;

TileToUpdate.Position = new Vector2((int)(x \* SquareSize + TablePosition.X), (int)(y \* SquareSize + TablePosition.Y));

UpdateTable(Table1, TileToUpdate);

Table[x, y] = false;

} else if (((LastTileTaken.SecondTileValue == Table1.LeftHandSide) || (LastTileTaken.FirstTileValue == Table1.LeftHandSide) || (StartOfNewGame) || (StartOfRound)) && (recTileLeftHandSide.Contains((int) MousePosition.X, (int) MousePosition.Y))) {

TileToUpdate = LastTileTaken;

TileToUpdate.Position = new Vector2((int)(x \* SquareSize + TablePosition.X), (int)(y \* SquareSize + TablePosition.Y));

UpdateTable(Table1, TileToUpdate);

Table[x, y] = false;

} else if (StartOfNewGame && LastTileTaken.FirstTileValue == 6 && LastTileTaken.SecondTileValue == 6) {

TileToUpdate = LastTileTaken;

TileToUpdate.Position = new Vector2((int)(x \* SquareSize + TablePosition.X), (int)(y \* SquareSize + TablePosition.Y));

UpdateTable(Table1, TileToUpdate);

Table[x, y] = false;

} else if (StartOfRound) {

TileToUpdate = LastTileTaken;

TileToUpdate.Position = new Vector2((int)(x \* SquareSize + TablePosition.X), (int)(y \* SquareSize + TablePosition.Y));

UpdateTable(Table1, TileToUpdate);

Table[x, y] = false;

} else {

Table[x, y] = false;

}

}

}

}

}

private void DrawTilesOnBoard() {

foreach(Tile h in Table1.TilesPlayedOnTableList) {

if (h.LastOrientation == 1)

spriteBatch.Draw(h.Image, h.Position, null, ColorToDraw, 0, Vector2.Zero, 0.07 f, SpriteEffects.None, .9 f);

if (h.LastOrientation == 2)

spriteBatch.Draw(h.Image, new Vector2(h.Position.X + 28, h.Position.Y), null, ColorToDraw, (float) Math.PI / 2, Vector2.Zero, 0.07 f, SpriteEffects.None, .9 f);

if (h.LastOrientation == 3)

spriteBatch.Draw(h.Image, new Vector2(h.Position.X + 28, h.Position.Y + 28), null, ColorToDraw, (float) Math.PI, Vector2.Zero, 0.07 f, SpriteEffects.None, .9 f);

if (h.LastOrientation == 4)

spriteBatch.Draw(h.Image, new Vector2(h.Position.X, h.Position.Y + 28), null, ColorToDraw, (float) Math.PI \* 3 / 2, Vector2.Zero, 0.07 f, SpriteEffects.None, .9 f);

}

}

#

endregion

# region Доска и мышка

private bool IsMouseCoordinateWithinBoard(int x, int y) {

return (int)(MousePosition.X - TablePosition.X) / SquareSize == x && (int)(MousePosition.Y - TablePosition.Y) / SquareSize == y;

}

bool IsMouseInsideBoard() {

if (MousePosition.X >= TablePosition.X && MousePosition.X <= TablePosition.X + Table.GetLength(0) \* SquareSize && MousePosition.Y >= TablePosition.Y && MousePosition.Y <= TablePosition.Y + Table.GetLength(1) \* SquareSize) {

return true;

} else {

return false;

}

}

Vector2 GetSquareFromMousePosition(Vector2 position) {

return new Vector2((int)(MousePosition.X - TablePosition.X) / SquareSize, (int)(MousePosition.Y - TablePosition.Y) / SquareSize);

}

#

endregion

# region Методы игры

private void DealDominoTilesToPlayers(List < Tile > ListOfDominoTilesToDeal) {

Tile dominoTileToRemove = null;

if (ListOfDominoTilesToDeal.Count > 0) {

for (int i = 0; i < 7; i++) {

foreach(Player player in PlayersList) {

int RandomTilePosition = rand.Next(ListOfDominoTilesToDeal.Count);

dominoTileToRemove = ListOfDominoTilesToDeal[RandomTilePosition];

ListOfDominoTilesToDeal.RemoveAt(RandomTilePosition);

player.PlayerTileList.Add(dominoTileToRemove);

}

}

}

Vector2 InitialPlayerPosition = new Vector2(449, 680);

for (int i = 0; i < player1.PlayerTileList.Count; i++) {

player1.PlayerTileList[i].Position = InitialPlayerPosition;

player1.PlayerTileList[i].TileEdge = new Rectangle((int) player1.PlayerTileList[i].Position.X, (int) player1.PlayerTileList[i].Position.Y, 28, 56);

InitialPlayerPosition = new Vector2(InitialPlayerPosition.X + 28.8 f, 680);

}

}

private void UpdateTable(Table Table1, Tile TileToUpdate) {

foreach(Player player in PlayersList) {

for (int i = 0; i < player.PlayerTileList.Count; i++) {

if ((player.PlayerTileList[i].FirstTileValue == TileToUpdate.FirstTileValue) && (player.PlayerTileList[i].SecondTileValue == TileToUpdate.SecondTileValue) && player.IsHuman && (Table1.PlayerInTurn == player)) {

if (StartOfNewGame) {

if (player.PlayerTileList[i].FirstTileValue == 6 && player.PlayerTileList[i].SecondTileValue == 6) {

TilePlacementLogic(Table1, TileToUpdate, player, i);

StartOfNewGame = false;

}

} else if (StartOfRound) {

TilePlacementLogic(Table1, TileToUpdate, player, i);

StartOfRound = false;

} else if ((player.PlayerTileList[i].FirstTileValue == Table1.RightHandSide) || (player.PlayerTileList[i].FirstTileValue == Table1.LeftHandSide) || (player.PlayerTileList[i].SecondTileValue == Table1.RightHandSide) || (player.PlayerTileList[i].SecondTileValue == Table1.LeftHandSide)) {

TilePlacementLogic(Table1, TileToUpdate, player, i);

}

}

}

}

}

/// <summary>

/// This is very messy and needs some emergency refactoring

/// </summary>

/// <param name="Table1"></param>

/// <param name="TileToUpdate"></param>

/// <param name="Player"></param>

/// <param name="PositionToRemove"></param>

private void TilePlacementLogic(Table Table1, Tile TileToUpdate, Player Player, int PositionToRemove) {

Vector2 ReferencePosition = TileToUpdate.Position;

if (Table1.TilesPlayedOnTableList.Count < 1) {

if (TileToUpdate.IsTileADouble) {

TileToUpdate.LastOrientation = 1;

TileToUpdate.Position = new Vector2(TileToUpdate.Position.X, TileToUpdate.Position.Y - 14);

Table1.PositionOfRightHandSideEdge = new Vector2((int)(ReferencePosition.X + SquareSize), (int)(ReferencePosition.Y));

Table1.PositionOfLeftHandSideEdge = new Vector2((int)(ReferencePosition.X - SquareSize), (int)(ReferencePosition.Y));

} else {

Table1.PositionOfRightHandSideEdge = new Vector2((int)(ReferencePosition.X + SquareSize), (int)(ReferencePosition.Y));

Table1.PositionOfLeftHandSideEdge = new Vector2((int)(ReferencePosition.X - 2 \* SquareSize), (int)(ReferencePosition.Y));

}

Table1.RightHandSide = TileToUpdate.FirstTileValue;

Table1.LeftHandSide = TileToUpdate.SecondTileValue;

Table1.TileOnRightHandSide = TileToUpdate;

Table1.TileOnLeftHandSide = TileToUpdate;

Table1.PositionOfTileOnRightHandSide = ReferencePosition;

Table1.PositionOfTileOnLeftHandSide = ReferencePosition;

AddDominoTile(Table1, TileToUpdate, Player, PositionToRemove);

if (Table1.PlayerInTurn == player1) {

CalculateTurn(Table1.PlayerInTurn);

}

} else {

if ((Table1.TileOnRightHandSide.LastOrientation == 1 || Table1.TileOnRightHandSide.LastOrientation == 3) && (TileToUpdate.IsTileADouble) && (ReferencePosition.Y > Table1.PositionOfTileOnRightHandSide.Y) && (Table1.RightHandSide == TileToUpdate.FirstTileValue)) {

TileToUpdate.LastOrientation = 2;

TileToUpdate.Position = new Vector2(TileToUpdate.Position.X + 14, TileToUpdate.Position.Y);

Table1.TileOnRightHandSide = TileToUpdate;

Table1.PositionOfRightHandSideEdge = new Vector2(ReferencePosition.X, ReferencePosition.Y + SquareSize);

Table1.PositionOfTileOnRightHandSide = ReferencePosition;

AddDominoTile(Table1, TileToUpdate, Player, PositionToRemove);

} else if ((Table1.TileOnLeftHandSide.LastOrientation == 1 || Table1.TileOnLeftHandSide.LastOrientation == 3) && (TileToUpdate.IsTileADouble) && (ReferencePosition.Y < Table1.PositionOfTileOnLeftHandSide.Y) && (Table1.LeftHandSide == TileToUpdate.FirstTileValue)) {

TileToUpdate.LastOrientation = 2;

TileToUpdate.Position = new Vector2(TileToUpdate.Position.X + 14, TileToUpdate.Position.Y);

Table1.TileOnLeftHandSide = TileToUpdate;

Table1.PositionOfLeftHandSideEdge = new Vector2(ReferencePosition.X, ReferencePosition.Y - SquareSize);

Table1.PositionOfTileOnLeftHandSide = ReferencePosition;

AddDominoTile(Table1, TileToUpdate, Player, PositionToRemove);

} else if ((Table1.TileOnLeftHandSide.LastOrientation == 2 || Table1.TileOnLeftHandSide.LastOrientation == 4) && (TileToUpdate.IsTileADouble) && ReferencePosition.X < Table1.PositionOfTileOnLeftHandSide.X && Table1.LeftHandSide == TileToUpdate.FirstTileValue) {

TileToUpdate.LastOrientation = 1;

TileToUpdate.Position = new Vector2(TileToUpdate.Position.X, TileToUpdate.Position.Y - 14);

Table1.TileOnLeftHandSide = TileToUpdate;

Table1.PositionOfLeftHandSideEdge = new Vector2(ReferencePosition.X - SquareSize, ReferencePosition.Y);

if (ReferencePosition.X < (TablePosition.X + (4 \* SquareSize))) {

TileToUpdate.Position = new Vector2(TileToUpdate.Position.X, TileToUpdate.Position.Y + 14);

TileToUpdate.LastOrientation = 2;

Table1.PositionOfLeftHandSideEdge = new Vector2(ReferencePosition.X - SquareSize, ReferencePosition.Y - SquareSize);

}

Table1.PositionOfTileOnLeftHandSide = ReferencePosition;

AddDominoTile(Table1, TileToUpdate, Player, PositionToRemove);

} else if ((Table1.TileOnRightHandSide.LastOrientation == 2 || Table1.TileOnRightHandSide.LastOrientation == 4) && (TileToUpdate.IsTileADouble) && ReferencePosition.X > Table1.PositionOfTileOnRightHandSide.X && Table1.RightHandSide == TileToUpdate.FirstTileValue) {

TileToUpdate.LastOrientation = 1;

TileToUpdate.Position = new Vector2(TileToUpdate.Position.X, TileToUpdate.Position.Y - 14);

Table1.TileOnRightHandSide = TileToUpdate;

Table1.PositionOfRightHandSideEdge = new Vector2(ReferencePosition.X + SquareSize, ReferencePosition.Y);

if (ReferencePosition.X > (TablePosition.X + (27 \* SquareSize))) {

TileToUpdate.Position = new Vector2(TileToUpdate.Position.X, TileToUpdate.Position.Y + 14);

TileToUpdate.LastOrientation = 4;

Table1.PositionOfRightHandSideEdge = new Vector2(ReferencePosition.X + SquareSize, ReferencePosition.Y + SquareSize);

}

Table1.PositionOfTileOnRightHandSide = ReferencePosition;

AddDominoTile(Table1, TileToUpdate, Player, PositionToRemove);

} else if ((Table1.TileOnRightHandSide.LastOrientation == 1 || Table1.TileOnRightHandSide.LastOrientation == 3) && (!TileToUpdate.IsTileADouble) && Table1.TileOnRightHandSide.IsTileADouble && ReferencePosition.X > Table1.TileOnRightHandSide.Position.X) {

if (Table1.RightHandSide == TileToUpdate.FirstTileValue) {

TileToUpdate.LastOrientation = 4;

Table1.TileOnRightHandSide = TileToUpdate;

Table1.RightHandSide = TileToUpdate.SecondTileValue;

} else if (Table1.RightHandSide == TileToUpdate.SecondTileValue) {

TileToUpdate.LastOrientation = 2;

TileToUpdate.Position = new Vector2(TileToUpdate.Position.X + 28, TileToUpdate.Position.Y);

Table1.TileOnRightHandSide = TileToUpdate;

Table1.RightHandSide = TileToUpdate.FirstTileValue;

}

Table1.PositionOfRightHandSideEdge = new Vector2(ReferencePosition.X + 2 \* SquareSize, ReferencePosition.Y);

if (ReferencePosition.X > (TablePosition.X + (27 \* SquareSize))) {

Table1.PositionOfRightHandSideEdge = new Vector2(ReferencePosition.X + SquareSize, ReferencePosition.Y + SquareSize);

}

Table1.PositionOfTileOnRightHandSide = ReferencePosition;

AddDominoTile(Table1, TileToUpdate, Player, PositionToRemove);

} else if ((Table1.TileOnLeftHandSide.LastOrientation == 1 || Table1.TileOnLeftHandSide.LastOrientation == 3) && (!TileToUpdate.IsTileADouble) && Table1.TileOnLeftHandSide.IsTileADouble && (ReferencePosition.X < Table1.TileOnLeftHandSide.Position.X)) {

if (Table1.LeftHandSide == TileToUpdate.FirstTileValue) {

TileToUpdate.LastOrientation = 2;

Table1.TileOnLeftHandSide = TileToUpdate;

Table1.LeftHandSide = TileToUpdate.SecondTileValue;

} else if (Table1.LeftHandSide == TileToUpdate.SecondTileValue) {

TileToUpdate.LastOrientation = 4;

TileToUpdate.Position = new Vector2(TileToUpdate.Position.X - SquareSize, ReferencePosition.Y);

Table1.TileOnLeftHandSide = TileToUpdate;

Table1.LeftHandSide = TileToUpdate.FirstTileValue;

}

Table1.PositionOfLeftHandSideEdge = new Vector2(ReferencePosition.X - 2 \* SquareSize, ReferencePosition.Y);

if (ReferencePosition.X < (TablePosition.X + (4 \* SquareSize))) {

Table1.PositionOfLeftHandSideEdge = new Vector2(ReferencePosition.X - SquareSize, ReferencePosition.Y - SquareSize);

}

Table1.PositionOfTileOnLeftHandSide = ReferencePosition;

AddDominoTile(Table1, TileToUpdate, Player, PositionToRemove);

} else if ((Table1.TileOnRightHandSide.LastOrientation == 2 || Table1.TileOnRightHandSide.LastOrientation == 4) && (!TileToUpdate.IsTileADouble) && Table1.TileOnRightHandSide.IsTileADouble && ReferencePosition.Y > Table1.TileOnRightHandSide.Position.Y) {

if (Table1.RightHandSide == TileToUpdate.FirstTileValue) {

TileToUpdate.LastOrientation = 1;

Table1.TileOnRightHandSide = TileToUpdate;

Table1.RightHandSide = TileToUpdate.SecondTileValue;

} else if (Table1.RightHandSide == TileToUpdate.SecondTileValue) {

TileToUpdate.LastOrientation = 3;

TileToUpdate.Position = new Vector2(TileToUpdate.Position.X, TileToUpdate.Position.Y + 28);

Table1.TileOnRightHandSide = TileToUpdate;

Table1.RightHandSide = TileToUpdate.FirstTileValue;

}

Table1.PositionOfRightHandSideEdge = new Vector2(ReferencePosition.X, ReferencePosition.Y + 2 \* SquareSize);

Table1.PositionOfTileOnRightHandSide = ReferencePosition;

AddDominoTile(Table1, TileToUpdate, Player, PositionToRemove);

} else if ((Table1.TileOnLeftHandSide.LastOrientation == 2 || Table1.TileOnLeftHandSide.LastOrientation == 4) && (!TileToUpdate.IsTileADouble) && Table1.TileOnLeftHandSide.IsTileADouble && ReferencePosition.Y < Table1.TileOnLeftHandSide.Position.Y) {

if (Table1.LeftHandSide == TileToUpdate.FirstTileValue) {

TileToUpdate.LastOrientation = 3;

Table1.LeftHandSide = TileToUpdate.SecondTileValue;

Table1.TileOnLeftHandSide = TileToUpdate;

} else if (Table1.LeftHandSide == TileToUpdate.SecondTileValue) {

TileToUpdate.LastOrientation = 1;

TileToUpdate.Position = new Vector2(TileToUpdate.Position.X, TileToUpdate.Position.Y - SquareSize);

Table1.TileOnLeftHandSide = TileToUpdate;

Table1.LeftHandSide = TileToUpdate.FirstTileValue;

}

Table1.PositionOfLeftHandSideEdge = new Vector2(ReferencePosition.X, ReferencePosition.Y - 2 \* SquareSize);

Table1.PositionOfTileOnLeftHandSide = ReferencePosition;

AddDominoTile(Table1, TileToUpdate, Player, PositionToRemove);

} else if ((Table1.TileOnRightHandSide.LastOrientation == 1 || Table1.TileOnRightHandSide.LastOrientation == 3) && (!TileToUpdate.IsTileADouble) && (!Table1.TileOnRightHandSide.IsTileADouble) && ReferencePosition.Y > Table1.TileOnRightHandSide.Position.Y) {

if (Table1.RightHandSide == TileToUpdate.FirstTileValue) {

TileToUpdate.LastOrientation = 1;

Table1.TileOnRightHandSide = TileToUpdate;

Table1.RightHandSide = TileToUpdate.SecondTileValue;

} else if (Table1.RightHandSide == TileToUpdate.SecondTileValue) {

TileToUpdate.LastOrientation = 3;

TileToUpdate.Position = new Vector2(TileToUpdate.Position.X, TileToUpdate.Position.Y + SquareSize);

Table1.TileOnRightHandSide = TileToUpdate;

Table1.RightHandSide = TileToUpdate.FirstTileValue;

}

Table1.PositionOfRightHandSideEdge = new Vector2(ReferencePosition.X, ReferencePosition.Y + 2 \* SquareSize);

Table1.PositionOfTileOnRightHandSide = ReferencePosition;

AddDominoTile(Table1, TileToUpdate, Player, PositionToRemove);

} else if ((Table1.TileOnLeftHandSide.LastOrientation == 1 || Table1.TileOnLeftHandSide.LastOrientation == 3) && (!TileToUpdate.IsTileADouble) && (!Table1.TileOnLeftHandSide.IsTileADouble) && ReferencePosition.Y < Table1.TileOnLeftHandSide.Position.Y) {

if (Table1.LeftHandSide == TileToUpdate.FirstTileValue) {

TileToUpdate.LastOrientation = 3;

Table1.TileOnLeftHandSide = TileToUpdate;

Table1.LeftHandSide = TileToUpdate.SecondTileValue;

} else if (Table1.LeftHandSide == TileToUpdate.SecondTileValue) {

TileToUpdate.LastOrientation = 1;

TileToUpdate.Position = new Vector2(TileToUpdate.Position.X, TileToUpdate.Position.Y - SquareSize);

Table1.TileOnLeftHandSide = TileToUpdate;

Table1.LeftHandSide = TileToUpdate.FirstTileValue;

}

Table1.PositionOfLeftHandSideEdge = new Vector2(ReferencePosition.X, ReferencePosition.Y - 2 \* SquareSize);

Table1.PositionOfTileOnLeftHandSide = ReferencePosition;

AddDominoTile(Table1, TileToUpdate, Player, PositionToRemove);

} else if ((Table1.TileOnLeftHandSide.LastOrientation == 2 || Table1.TileOnLeftHandSide.LastOrientation == 4) && (!TileToUpdate.IsTileADouble) && (!Table1.TileOnLeftHandSide.IsTileADouble) && ReferencePosition.X < Table1.TileOnLeftHandSide.Position.X) {

if (Table1.LeftHandSide == TileToUpdate.FirstTileValue) {

TileToUpdate.LastOrientation = 2;

Table1.TileOnLeftHandSide = TileToUpdate;

Table1.LeftHandSide = TileToUpdate.SecondTileValue;

} else if (Table1.LeftHandSide == TileToUpdate.SecondTileValue) {

TileToUpdate.LastOrientation = 4;

TileToUpdate.Position = new Vector2(TileToUpdate.Position.X - SquareSize, TileToUpdate.Position.Y);

Table1.TileOnLeftHandSide = TileToUpdate;

Table1.LeftHandSide = TileToUpdate.FirstTileValue;

}

Table1.PositionOfLeftHandSideEdge = new Vector2(ReferencePosition.X - 2 \* SquareSize, ReferencePosition.Y);

if (ReferencePosition.X < (TablePosition.X + (4 \* SquareSize))) {

Table1.PositionOfLeftHandSideEdge = new Vector2(ReferencePosition.X - SquareSize, ReferencePosition.Y - SquareSize);

}

Table1.PositionOfTileOnLeftHandSide = ReferencePosition;

AddDominoTile(Table1, TileToUpdate, Player, PositionToRemove);

} else if ((Table1.TileOnRightHandSide.LastOrientation == 2 || Table1.TileOnRightHandSide.LastOrientation == 4) && (!TileToUpdate.IsTileADouble) && (!Table1.TileOnRightHandSide.IsTileADouble) && ReferencePosition.X > Table1.TileOnRightHandSide.Position.X) {

if (Table1.RightHandSide == TileToUpdate.FirstTileValue) {

TileToUpdate.LastOrientation = 4;

Table1.TileOnRightHandSide = TileToUpdate;

Table1.RightHandSide = TileToUpdate.SecondTileValue;

} else if (Table1.RightHandSide == TileToUpdate.SecondTileValue) {

TileToUpdate.LastOrientation = 2;

TileToUpdate.Position = new Vector2(TileToUpdate.Position.X + SquareSize, TileToUpdate.Position.Y);

Table1.TileOnRightHandSide = TileToUpdate;

Table1.RightHandSide = TileToUpdate.FirstTileValue;

}

Table1.PositionOfRightHandSideEdge = new Vector2(ReferencePosition.X + 2 \* SquareSize, ReferencePosition.Y);

if (ReferencePosition.X > (TablePosition.X + (27 \* SquareSize))) {

Table1.PositionOfRightHandSideEdge = new Vector2(ReferencePosition.X + SquareSize, ReferencePosition.Y + SquareSize);

}

Table1.PositionOfTileOnRightHandSide = ReferencePosition;

AddDominoTile(Table1, TileToUpdate, Player, PositionToRemove);

} else if ((Table1.TileOnRightHandSide.LastOrientation == 2 || Table1.TileOnRightHandSide.LastOrientation == 4) && (!TileToUpdate.IsTileADouble) && (!Table1.TileOnRightHandSide.IsTileADouble) && ReferencePosition.Y > Table1.TileOnRightHandSide.Position.Y) {

if (Table1.RightHandSide == TileToUpdate.FirstTileValue) {

TileToUpdate.LastOrientation = 1;

Table1.TileOnRightHandSide = TileToUpdate;

Table1.RightHandSide = TileToUpdate.SecondTileValue;

} else if (Table1.RightHandSide == TileToUpdate.SecondTileValue) {

TileToUpdate.LastOrientation = 3;

TileToUpdate.Position = new Vector2(TileToUpdate.Position.X, TileToUpdate.Position.Y + SquareSize);

Table1.TileOnRightHandSide = TileToUpdate;

Table1.RightHandSide = TileToUpdate.FirstTileValue;

}

Table1.PositionOfRightHandSideEdge = new Vector2(ReferencePosition.X, ReferencePosition.Y + 2 \* SquareSize);

Table1.PositionOfTileOnRightHandSide = ReferencePosition;

AddDominoTile(Table1, TileToUpdate, Player, PositionToRemove);

} else if ((Table1.TileOnLeftHandSide.LastOrientation == 2 || Table1.TileOnLeftHandSide.LastOrientation == 4) && (!TileToUpdate.IsTileADouble) && (!Table1.TileOnLeftHandSide.IsTileADouble) && ReferencePosition.Y < Table1.TileOnLeftHandSide.Position.Y) {

if (Table1.LeftHandSide == TileToUpdate.FirstTileValue) {

TileToUpdate.LastOrientation = 3;

Table1.TileOnLeftHandSide = TileToUpdate;

Table1.LeftHandSide = TileToUpdate.SecondTileValue;

} else if (Table1.LeftHandSide == TileToUpdate.SecondTileValue) {

TileToUpdate.LastOrientation = 1;

TileToUpdate.Position = new Vector2(TileToUpdate.Position.X, TileToUpdate.Position.Y - SquareSize);

Table1.TileOnLeftHandSide = TileToUpdate;

Table1.LeftHandSide = TileToUpdate.FirstTileValue;

}

Table1.PositionOfLeftHandSideEdge = new Vector2(ReferencePosition.X, ReferencePosition.Y - 2 \* SquareSize);

Table1.PositionOfTileOnLeftHandSide = ReferencePosition;

AddDominoTile(Table1, TileToUpdate, Player, PositionToRemove);

}

if (Table1.PlayerInTurn == player1) {

CalculateTurn(Table1.PlayerInTurn);

}

}

}

private void AddDominoTile(Table Table1, Tile DominoTileToUpdate, Player Player, int PositionToRemove) {

Player.PlayerTileList.RemoveAt(PositionToRemove);

Table1.TilesPlayedOnTableList.Add(DominoTileToUpdate);

LastDominoTilePlayed = DominoTileToUpdate;

PlayerWhoLastPlayed = Table1.PlayerInTurn;

}

private void CalculateTurn(Player PlayerInTurn) {

if (PlayerInTurn == player4) {

Table1.PlayerInTurn = player1;

player4.MyTurn = false;

player1.MyTurn = true;

} else if (PlayerInTurn == player1) {

Table1.PlayerInTurn = player2;

player1.MyTurn = false;

player2.MyTurn = true;

} else if (PlayerInTurn == player2) {

Table1.PlayerInTurn = player3;

player2.MyTurn = false;

player3.MyTurn = true;

} else if (PlayerInTurn == player3) {

Table1.PlayerInTurn = player4;

player3.MyTurn = false;

player4.MyTurn = true;

}

}

private void FindOutIfPlayerPassesHisTurn(Player player) {

bool tempPlayerPassedTurn = true;

foreach(Tile f in player.PlayerTileList) {

if (f.FirstTileValue == Table1.RightHandSide || f.FirstTileValue == Table1.LeftHandSide || f.SecondTileValue == Table1.RightHandSide || f.SecondTileValue == Table1.LeftHandSide) {

tempPlayerPassedTurn = false;

}

}

IsPlayerPassing = tempPlayerPassedTurn;

}

private void ScoreTheGame() {

int pointsToAdd = 0;

if ((player1.PlayerTileList.Count == 0) || (player3.PlayerTileList.Count == 0)) {

for (int i = 0; i < player1.PlayerTileList.Count; i++) {

pointsToAdd += player1.PlayerTileList[i].TotalPointsValue;

}

for (int i = 0; i < player2.PlayerTileList.Count; i++) {

pointsToAdd += player2.PlayerTileList[i].TotalPointsValue;

}

for (int i = 0; i < player3.PlayerTileList.Count; i++) {

pointsToAdd += player3.PlayerTileList[i].TotalPointsValue;

}

for (int i = 0; i < player4.PlayerTileList.Count; i++) {

pointsToAdd += player4.PlayerTileList[i].TotalPointsValue;

}

TeamOneTotalPoints += pointsToAdd;

} else if ((player2.PlayerTileList.Count == 0) || (player4.PlayerTileList.Count == 0)) {

for (int i = 0; i < player1.PlayerTileList.Count; i++) {

pointsToAdd += player1.PlayerTileList[i].TotalPointsValue;

}

for (int i = 0; i < player2.PlayerTileList.Count; i++) {

pointsToAdd += player2.PlayerTileList[i].TotalPointsValue;

}

for (int i = 0; i < player3.PlayerTileList.Count; i++) {

pointsToAdd += player3.PlayerTileList[i].TotalPointsValue;

}

for (int i = 0; i < player4.PlayerTileList.Count; i++) {

pointsToAdd += player4.PlayerTileList[i].TotalPointsValue;

}

TeamTwoTotalPoints += pointsToAdd;

} else if (GameLocked) {

if (PlayerWhoLastPlayed.Name == player1.Name) {

int PointsFromPlayerWhoLockedGame = 0, PointsOfPlayerToTheRightOfPlayerWhoLockedGame = 0;

for (int i = 0; i < player1.PlayerTileList.Count; i++) {

PointsFromPlayerWhoLockedGame += player1.PlayerTileList[i].TotalPointsValue;

}

for (int i = 0; i < player2.PlayerTileList.Count; i++) {

PointsOfPlayerToTheRightOfPlayerWhoLockedGame += player2.PlayerTileList[i].TotalPointsValue;

}

if (PointsFromPlayerWhoLockedGame < PointsOfPlayerToTheRightOfPlayerWhoLockedGame || PointsFromPlayerWhoLockedGame == PointsOfPlayerToTheRightOfPlayerWhoLockedGame) {

for (int i = 0; i < player1.PlayerTileList.Count; i++) {

pointsToAdd += player1.PlayerTileList[i].TotalPointsValue;

}

for (int i = 0; i < player2.PlayerTileList.Count; i++) {

pointsToAdd += player2.PlayerTileList[i].TotalPointsValue;

}

for (int i = 0; i < player3.PlayerTileList.Count; i++) {

pointsToAdd += player3.PlayerTileList[i].TotalPointsValue;

}

for (int i = 0; i < player4.PlayerTileList.Count; i++) {

pointsToAdd += player4.PlayerTileList[i].TotalPointsValue;

}

TeamOneTotalPoints += pointsToAdd;

} else if (PointsFromPlayerWhoLockedGame > PointsOfPlayerToTheRightOfPlayerWhoLockedGame) {

for (int i = 0; i < player1.PlayerTileList.Count; i++) {

pointsToAdd += player1.PlayerTileList[i].TotalPointsValue;

}

for (int i = 0; i < player2.PlayerTileList.Count; i++) {

pointsToAdd += player2.PlayerTileList[i].TotalPointsValue;

}

for (int i = 0; i < player3.PlayerTileList.Count; i++) {

pointsToAdd += player3.PlayerTileList[i].TotalPointsValue;

}

for (int i = 0; i < player4.PlayerTileList.Count; i++) {

pointsToAdd += player4.PlayerTileList[i].TotalPointsValue;

}

TeamTwoTotalPoints += pointsToAdd;

}

}

}

}

private void DrawTilesForNonHumanPlayersLevel1(Player playerOnTurn) {

for (int i = 0; i < playerOnTurn.PlayerTileList.Count; i++) {

if ((Table1.LeftHandSide == playerOnTurn.PlayerTileList[i].FirstTileValue) || (Table1.LeftHandSide == playerOnTurn.PlayerTileList[i].SecondTileValue)) {

playerOnTurn.PlayerTileList[i].Position = Table1.PositionOfLeftHandSideEdge;

TilePlacementLogic(Table1, playerOnTurn.PlayerTileList[i], playerOnTurn, i);

break;

} else if ((Table1.RightHandSide == playerOnTurn.PlayerTileList[i].FirstTileValue) || (Table1.RightHandSide == playerOnTurn.PlayerTileList[i].SecondTileValue)) {

playerOnTurn.PlayerTileList[i].Position = Table1.PositionOfRightHandSideEdge;

TilePlacementLogic(Table1, playerOnTurn.PlayerTileList[i], playerOnTurn, i);

break;

}

}

}

private void DrawTilesForNonHumanPlayersLevel2(Player playerOnTurn) {

int tempRight = 0;

int tempLeft = 0;

bool RightHandSideEdge = true;

Tile tempTile = new Tile();

tempTile.TotalPointsValue = 0;

for (int i = 0; i < playerOnTurn.PlayerTileList.Count; i++) {

if ((Table1.LeftHandSide == playerOnTurn.PlayerTileList[i].FirstTileValue) || (Table1.LeftHandSide == playerOnTurn.PlayerTileList[i].SecondTileValue)) {

if (playerOnTurn.PlayerTileList[i].TotalPointsValue > tempTile.TotalPointsValue) {

tempLeft = i;

tempTile = playerOnTurn.PlayerTileList[i];

RightHandSideEdge = false;

}

} else if ((Table1.RightHandSide == playerOnTurn.PlayerTileList[i].FirstTileValue) || (Table1.RightHandSide == playerOnTurn.PlayerTileList[i].SecondTileValue)) {

if (playerOnTurn.PlayerTileList[i].TotalPointsValue > tempTile.TotalPointsValue) {

tempRight = i;

tempTile = playerOnTurn.PlayerTileList[i];

RightHandSideEdge = true;

}

}

if ((tempTile.TotalPointsValue != 0) && (RightHandSideEdge)) {

playerOnTurn.PlayerTileList[tempRight].Position = Table1.PositionOfRightHandSideEdge;

TilePlacementLogic(Table1, playerOnTurn.PlayerTileList[tempRight], playerOnTurn, tempRight);

} else if ((tempTile.TotalPointsValue != 0) && (!RightHandSideEdge)) {

playerOnTurn.PlayerTileList[tempLeft].Position = Table1.PositionOfLeftHandSideEdge;

TilePlacementLogic(Table1, playerOnTurn.PlayerTileList[tempLeft], playerOnTurn, tempLeft);

}

}

}

private void DrawTilesForNonHumanPlayersLevel3(Player playerOnTurn) {

List < int > TilesThatThePlayerIsAbleToPlayWith = new List < int > ();

int Doubles = 0;

for (int i = 0; i < playerOnTurn.PlayerTileList.Count; i++) {

if ((Table1.LeftHandSide == playerOnTurn.PlayerTileList[i].FirstTileValue) || (Table1.LeftHandSide == playerOnTurn.PlayerTileList[i].SecondTileValue) || (Table1.RightHandSide == playerOnTurn.PlayerTileList[i].FirstTileValue) || (Table1.RightHandSide == playerOnTurn.PlayerTileList[i].SecondTileValue)) {

TilesThatThePlayerIsAbleToPlayWith.Add(i);

if (playerOnTurn.PlayerTileList[i].IsTileADouble) {

Doubles++;

int temp = 0;

foreach(Tile g in Table1.TilesPlayedOnTableList) {

if (g.FirstTileValue == playerOnTurn.PlayerTileList[i].FirstTileValue || g.SecondTileValue == playerOnTurn.PlayerTileList[i].FirstTileValue) {

temp++;

}

}

if (temp >= 3 && temp < 5) {

playerOnTurn.PlayerTileList[i].Priority = 2;

} else if (temp >= 5) {

playerOnTurn.PlayerTileList[i].Priority = 3;

}

}

}

if (TilesThatThePlayerIsAbleToPlayWith.Count == 1) {

if ((Table1.LeftHandSide == playerOnTurn.PlayerTileList[TilesThatThePlayerIsAbleToPlayWith[0]].FirstTileValue) || (Table1.LeftHandSide == playerOnTurn.PlayerTileList[TilesThatThePlayerIsAbleToPlayWith[0]].SecondTileValue)) {

playerOnTurn.PlayerTileList[TilesThatThePlayerIsAbleToPlayWith[0]].Position = Table1.PositionOfLeftHandSideEdge;

TilePlacementLogic(Table1, playerOnTurn.PlayerTileList[TilesThatThePlayerIsAbleToPlayWith[0]], playerOnTurn, TilesThatThePlayerIsAbleToPlayWith[0]);

} else if ((Table1.RightHandSide == playerOnTurn.PlayerTileList[TilesThatThePlayerIsAbleToPlayWith[0]].FirstTileValue) || (Table1.RightHandSide == playerOnTurn.PlayerTileList[TilesThatThePlayerIsAbleToPlayWith[0]].SecondTileValue)) {

playerOnTurn.PlayerTileList[TilesThatThePlayerIsAbleToPlayWith[0]].Position = Table1.PositionOfRightHandSideEdge;

TilePlacementLogic(Table1, playerOnTurn.PlayerTileList[TilesThatThePlayerIsAbleToPlayWith[0]], playerOnTurn, TilesThatThePlayerIsAbleToPlayWith[0]);

}

} else if (Doubles > 0) {

int itmp = new int();

for (int h = 0; h < playerOnTurn.PlayerTileList.Count; h++) {

if (playerOnTurn.PlayerTileList[h].Priority > playerOnTurn.PlayerTileList[itmp].Priority) {

itmp = h;

}

}

if ((Table1.LeftHandSide == playerOnTurn.PlayerTileList[itmp].FirstTileValue) || (Table1.LeftHandSide == playerOnTurn.PlayerTileList[itmp].SecondTileValue)) {

playerOnTurn.PlayerTileList[itmp].Position = Table1.PositionOfLeftHandSideEdge;

TilePlacementLogic(Table1, playerOnTurn.PlayerTileList[itmp], playerOnTurn, itmp);

} else if ((Table1.RightHandSide == playerOnTurn.PlayerTileList[itmp].FirstTileValue) || (Table1.RightHandSide == playerOnTurn.PlayerTileList[itmp].SecondTileValue)) {

playerOnTurn.PlayerTileList[itmp].Position = Table1.PositionOfRightHandSideEdge;

TilePlacementLogic(Table1, playerOnTurn.PlayerTileList[itmp], playerOnTurn, itmp);

}

} else {

DrawTilesForNonHumanPlayersLevel1(playerOnTurn);

}

}

}

private void DrawTilesForNonHumanPlayersLevel4(Player PlayerOnTurn) {

if (PlayerOnTurn == player1) {

if ((player2.PositionOfTileLastPlayed != player3.PositionOfTileLastPlayed) && (player2.PositionOfTileLastPlayed != player4.PositionOfTileLastPlayed)) {

if (player2.PositionOfTileLastPlayed == 2) {

for (int i = 0; i < player1.PlayerTileList.Count; i++) {

if ((Table1.LeftHandSide == PlayerOnTurn.PlayerTileList[i].FirstTileValue) || (Table1.LeftHandSide == PlayerOnTurn.PlayerTileList[i].SecondTileValue)) {

PlayerOnTurn.PlayerTileList[i].Position = Table1.PositionOfLeftHandSideEdge;

TilePlacementLogic(Table1, PlayerOnTurn.PlayerTileList[i], PlayerOnTurn, i);

}

}

} else if (player2.PositionOfTileLastPlayed == 1) {

for (int i = 0; i < player1.PlayerTileList.Count; i++) {

if ((Table1.RightHandSide == PlayerOnTurn.PlayerTileList[i].FirstTileValue) || (Table1.RightHandSide == PlayerOnTurn.PlayerTileList[i].SecondTileValue)) {

PlayerOnTurn.PlayerTileList[i].Position = Table1.PositionOfRightHandSideEdge;

TilePlacementLogic(Table1, PlayerOnTurn.PlayerTileList[i], PlayerOnTurn, i);

}

}

} else {

DrawTilesForNonHumanPlayersLevel1(PlayerOnTurn);

}

}

} else if (PlayerOnTurn == player2) {

if ((player3.PositionOfTileLastPlayed != player4.PositionOfTileLastPlayed) && (player3.PositionOfTileLastPlayed != player1.PositionOfTileLastPlayed)) {

if (player3.PositionOfTileLastPlayed == 2) {

for (int i = 0; i < player2.PlayerTileList.Count; i++) {

if ((Table1.LeftHandSide == PlayerOnTurn.PlayerTileList[i].FirstTileValue) || (Table1.LeftHandSide == PlayerOnTurn.PlayerTileList[i].SecondTileValue)) {

PlayerOnTurn.PlayerTileList[i].Position = Table1.PositionOfLeftHandSideEdge;

TilePlacementLogic(Table1, PlayerOnTurn.PlayerTileList[i], PlayerOnTurn, i);

}

}

} else if (player3.PositionOfTileLastPlayed == 1) {

for (int i = 0; i < player2.PlayerTileList.Count; i++) {

if ((Table1.RightHandSide == PlayerOnTurn.PlayerTileList[i].FirstTileValue) || (Table1.RightHandSide == PlayerOnTurn.PlayerTileList[i].SecondTileValue)) {

PlayerOnTurn.PlayerTileList[i].Position = Table1.PositionOfRightHandSideEdge;

TilePlacementLogic(Table1, PlayerOnTurn.PlayerTileList[i], PlayerOnTurn, i);

}

}

} else {

DrawTilesForNonHumanPlayersLevel1(PlayerOnTurn);

}

}

} else if (PlayerOnTurn == player3) {

if ((player4.PositionOfTileLastPlayed != player1.PositionOfTileLastPlayed) && (player4.PositionOfTileLastPlayed != player2.PositionOfTileLastPlayed)) {

if (player4.PositionOfTileLastPlayed == 2) {

for (int i = 0; i < player3.PlayerTileList.Count; i++) {

if ((Table1.LeftHandSide == PlayerOnTurn.PlayerTileList[i].FirstTileValue) || (Table1.LeftHandSide == PlayerOnTurn.PlayerTileList[i].SecondTileValue)) {

PlayerOnTurn.PlayerTileList[i].Position = Table1.PositionOfLeftHandSideEdge;

TilePlacementLogic(Table1, PlayerOnTurn.PlayerTileList[i], PlayerOnTurn, i);

}

}

} else if (player4.PositionOfTileLastPlayed == 1) {

for (int i = 0; i < player3.PlayerTileList.Count; i++) {

if ((Table1.RightHandSide == PlayerOnTurn.PlayerTileList[i].FirstTileValue) || (Table1.RightHandSide == PlayerOnTurn.PlayerTileList[i].SecondTileValue)) {

PlayerOnTurn.PlayerTileList[i].Position = Table1.PositionOfRightHandSideEdge;

TilePlacementLogic(Table1, PlayerOnTurn.PlayerTileList[i], PlayerOnTurn, i);

}

}

} else {

DrawTilesForNonHumanPlayersLevel1(PlayerOnTurn);

}

}

} else if (PlayerOnTurn == player4) {

if ((player1.PositionOfTileLastPlayed != player2.PositionOfTileLastPlayed) && (player1.PositionOfTileLastPlayed != player3.PositionOfTileLastPlayed)) {

if (player1.PositionOfTileLastPlayed == 2) {

for (int i = 0; i < player4.PlayerTileList.Count; i++) {

if ((Table1.LeftHandSide == PlayerOnTurn.PlayerTileList[i].FirstTileValue) || (Table1.LeftHandSide == PlayerOnTurn.PlayerTileList[i].SecondTileValue)) {

PlayerOnTurn.PlayerTileList[i].Position = Table1.PositionOfLeftHandSideEdge;

TilePlacementLogic(Table1, PlayerOnTurn.PlayerTileList[i], PlayerOnTurn, i);

}

}

} else if (player1.PositionOfTileLastPlayed == 1) {

for (int i = 0; i < player4.PlayerTileList.Count; i++) {

if ((Table1.RightHandSide == PlayerOnTurn.PlayerTileList[i].FirstTileValue) || (Table1.RightHandSide == PlayerOnTurn.PlayerTileList[i].SecondTileValue)) {

PlayerOnTurn.PlayerTileList[i].Position = Table1.PositionOfRightHandSideEdge;

TilePlacementLogic(Table1, PlayerOnTurn.PlayerTileList[i], PlayerOnTurn, i);

}

}

} else {

DrawTilesForNonHumanPlayersLevel1(PlayerOnTurn);

}

}

}

for (int i = 0; i < PlayerOnTurn.PlayerTileList.Count; i++) {

if ((Table1.LeftHandSide == PlayerOnTurn.PlayerTileList[i].FirstTileValue) || (Table1.LeftHandSide == PlayerOnTurn.PlayerTileList[i].SecondTileValue)) {

PlayerOnTurn.PlayerTileList[i].Position = Table1.PositionOfLeftHandSideEdge;

TilePlacementLogic(Table1, PlayerOnTurn.PlayerTileList[i], PlayerOnTurn, i);

break;

} else if ((Table1.RightHandSide == PlayerOnTurn.PlayerTileList[i].FirstTileValue) || (Table1.RightHandSide == PlayerOnTurn.PlayerTileList[i].SecondTileValue)) {

PlayerOnTurn.PlayerTileList[i].Position = Table1.PositionOfRightHandSideEdge;

TilePlacementLogic(Table1, PlayerOnTurn.PlayerTileList[i], PlayerOnTurn, i);

break;

}

}

}

private void VerifyEndOfRoundCondition() {

foreach(Player j in PlayersList) {

if ((j.PlayerTileList.Count == 0) && (!StartOfRound) && (!StartOfNewGame)) {

EndOfRound = true;

ScoreTheGame();

PlayerWhoWonLastRound = j;

CurrentGameState = GameState.EndOfRound;

break;

}

}

bool tempGameLocked = true;

foreach(Player j in PlayersList) {

foreach(Tile f in j.PlayerTileList) {

if (Table1.LeftHandSide == f.FirstTileValue || Table1.RightHandSide == f.FirstTileValue || Table1.LeftHandSide == f.SecondTileValue || Table1.RightHandSide == f.SecondTileValue) {

tempGameLocked = false;

break;

}

}

if (!tempGameLocked) {

break;

}

}

GameLocked = tempGameLocked;

if (GameLocked) {

EndOfRound = true;

ScoreTheGame();

CurrentGameState = GameState.EndOfRound;

}

}

#

endregion

# region Save Game & Save Data Methods

public void InitiateSave() {

if (!Guide.IsVisible) {

try {

if (!Guide.IsVisible) {

device = null;

StorageDevice.BeginShowSelector(PlayerIndex.One, this.SaveToDevice, null);

}

} catch (InvalidOperationException invalidOperationException) {

//Logger.error(“InvalidOperationException”);

StorageDevice.BeginShowSelector(PlayerIndex.One, this.SaveToDevice, null);

}

}

}

void SaveToDevice(IAsyncResult result) {

device = StorageDevice.EndShowSelector(result);

if (device != null && device.IsConnected) {

SaveGame SaveData = new SaveGame() {

DifficultyLevelToSave = CurrentDifficultyLevel,

TileColorToSave = CurrentTileColor

};

IAsyncResult r = device.BeginOpenContainer(containerName, null, null);

result.AsyncWaitHandle.WaitOne();

StorageContainer container = device.EndOpenContainer(r);

if (container.FileExists(filename))

container.DeleteFile(filename);

Stream stream = container.CreateFile(filename);

XmlSerializer serializer = new XmlSerializer(typeof(SaveGame));

serializer.Serialize(stream, SaveData);

stream.Close();

container.Dispose();

result.AsyncWaitHandle.Close();

}

}

void LoadFromDevice(IAsyncResult result) {

device = StorageDevice.EndShowSelector(result);

IAsyncResult r = device.BeginOpenContainer(containerName, null, null);

result.AsyncWaitHandle.WaitOne();

StorageContainer container = device.EndOpenContainer(r);

result.AsyncWaitHandle.Close();

if (container.FileExists(filename)) {

Stream stream = container.OpenFile(filename, FileMode.Open);

XmlSerializer serializer = new XmlSerializer(typeof(SaveGame));

SaveGame SaveData = (SaveGame) serializer.Deserialize(stream);

stream.Close();

container.Dispose();

// Update the game based on the save game file

CurrentDifficultyLevel = SaveData.DifficultyLevelToSave;

CurrentTileColor = SaveData.TileColorToSave;

}

}

#

endregion

}

}

Приложение Б

На рисунке представлен титульный скриншот презентации данной курсовой работы.

