МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных Технологий

Кафедра Программной инженерии\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Специальность  1-40 01 01 Программное обеспечение информационных технологий

Специализация Программирование интернет-приложений

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**КУРСОВОГО ПРОЕКТА:**

по дисциплине       "Объектно-ориентированное программирование и стандарты проектирования"

тема                        Программное средство «Игровая платформа»

Исполнитель

Студент 2 курса группы 4 Ширитон Константин Витальевич

(Ф.И.О.)

Руководитель  асс. СеверинчикН*.*А.

(учен. степень, звание, должность, подпись, Ф.И.О.)

Курсовой проект защищен с оценкой

Председатель  Пацей Н.В.

(подпись)

Минск 2019

**Оглавление**

[**Введение** 3](#_Toc9526112)

[**1** **Постановка задачи** 4](#_Toc9526113)

[**1.1 Обзор аналогов** 4](#_Toc9526114)

[**1.2 Существующие прототипы** 4](#_Toc9526115)

[**1.3 Задачи приложения** 9](#_Toc9526116)

[**2 Разработка архитектуры проекта** 10](#_Toc9526117)

[**2.1 Выбор используемых технологий** 10](#_Toc9526118)

[**2.2 Структура проекта** 10](#_Toc9526119)

[**3 Тестирование** 20](#_Toc9526120)

[**4 Руководство пользователя** 23](#_Toc9526121)

[**Заключение** 27](#_Toc9526122)

[**Список использованных источников** 28](#_Toc9526123)

[**ПРИЛОЖЕНИЕ А** 29](#_Toc9526124)

[**ПРИЛОЖЕНИЕ Б** 83](#_Toc9526125)

# **Введение**

Задавались ли вы вопросом как обычно отдыхают люди? Это может быть спорт, прогулки, сон. Но я могу точно сказать, что с развитием информационных технологий добавился ещё один вид развлечений – компьютерные игры.

Игры являются неотъемлемой частью досуга многих людей независимо от их происхождения, возраста, пола. Но все эти люди всегда задаются одним и тем же вопросом во что бы поиграть?

На этот вопрос можно найти ответ в приложениях–магазинах, в которых нам предоставляют обширный каталог со всевозможными играми, и опираясь на такие приложения, разрабатывался мой проект.

Целью моего курсового проекта является разработка приложения, которое в полной мере могло бы составить конкуренцию многим крупным проектам и компаниям, таким как «Steam», «Epic Games» и т.д. :)

Мое приложение может стать местом, где люди могут общаться друг с другом, соревноваться. Также в будущем приложение станет платформой, где различные компании из области игропрома смогут публиковать свои игры.

# **Постановка задачи**

В рамках выполнения курсовой работы была поставлена задача – разработать приложение – «Игровая платформа».

Функционально ПС должно выполнять функции платформы, где пользователь может:

* Играть в определенный набор игр;
* Соревноваться с другими пользователями;
* Общаться с другими пользователями;

При выполнении курсового проекта необходимо использовать принципы и приемы ООП. Приложение разрабатывается под ОС Windows и представляет собой настольное приложение (desktop). Разработать программу необходимо в Microsoft Visual Studio. Язык разработки – С#. Управление программой должно быть интуитивно понятным и удобным.

## 1.1 Обзор аналогов

Актуальность и выбор темы состоит в том, что похожий функционал имеют набор приложений. Тема является актуальной потому, что такого вида приложений для настольных персональных компьютеров с открытым кодом и системой рейтинга почти нет, или их качество, принципы работы и модели взаимодействия с информацией и пользователями оставляют желать лучшего.

Игровая платформа как система, разработанная для людей, имеющих представление что такое программирование, и которые хотят играть и соревноваться, но только с определенными пользователями, а также вести анонимный чат, имеет целый ряд положительных характеристик, позволяющих:

* + - Играть только определенным пользователям, которые обладают уникальной версией (имеющей свои игры и интерфейс);
    - Вести анонимный чат, используя свои собственные сервера;
    - Самому менять как интерфейс, так и содержание приложения;

## 1.2 Существующие прототипы

«Steam»— онлайн-сервис цифрового распространения компьютерных игр и программ, разработанный и поддерживаемый компанией Valve. «Steam» выполняет роль средства технической защиты авторских прав, платформы для многопользовательских игр и потокового вещания, а также социальной сети для игроков.

Программный клиент «Steam» также обеспечивает установку и регулярное обновление игр, облачные сохранения игр, текстовую и голосовую связь между игроками.

Программа содержит информацию как для авторизированных пользователей, так и не для авторизированных:

* общая информация (рисунок 1.1);
* подробная информация о играх (рисунок 1.3);
* отзывы пользователей в приложении (рисунок 1.4).



Рисунок 1.1 – Общая информация приложения «Steam»

Приложение содержит функционал, предназначенный для лиц различного возраста (рисунок 1.2). Ниже будут представлены скриншоты приложения, предоставляющие пользователям, богатый выбор информацией.

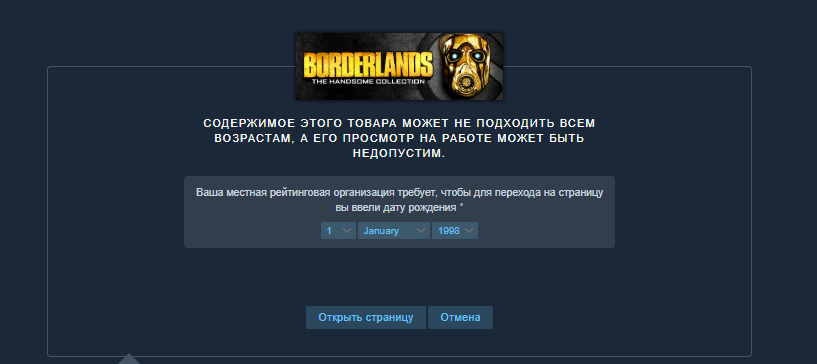


Рисунок 1.2 – Возрастные ограничения «Steam»



Рисунок 1.3 – Подробная информация о игре в приложения «Steam»

Авторизированный пользователь может написать отзыв о мероприятии или о приложении в целом, а также осуществлять бронь билетов.

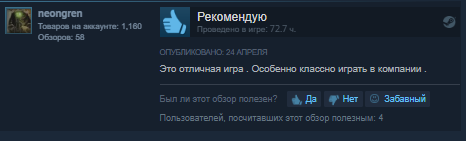


Рисунок 1.4 – Отзывы пользователей приложения «Steam»

Конечно, в приложении «Steam» присутствуют как положительные стороны, так и отрицательные. К положительным относятся:

* возможность играть на любом поддерживаемом устройстве;
* покупать контент по акциям со скидкой 50-75% и играть в платные игры бесплатно, причем такие акции проходят почти постоянно;
* устанавливать обновления игр в первоочередном порядке — платформа делает это автоматически;
* не думать о сохранности игровых данных — «Steam» также берет это на себя;
* создавать и сохранять резервные копии дистрибутивов игр;
* активировать поддерживаемые игры, купленные в версиях;
* покупать игры в подарок другим пользователям;
* создавать личные страницы и группы, вступать в сообщества;
* обмениваться сообщениями и общаться во встроенном голосовом чате;
* делиться достижениями;
* записывать и транслировать стримы без сторонних программ;
* получать свежие новости о событиях игрового мира;
* настраивать по своему вкусу дизайн личной страницы;
* пользоваться услугами технической поддержки разработчика.

К отрицательным сторонам относятся:

* отсутствие возможности запускать пиратский или свой (разработанный пользователем) контент.
* высокий риск потерять аккаунт, а вместе с ним — денежные средства, которые хранятся на балансе. Аккаунты пользователей «Steam» очень привлекательны для злоумышленников и часто подвергаются атакам. Для предотвращения нападений в платформу встроен защитник «Steam Guard», но и он не гарантирует полной безопасности.

Конечно «Steam» является не единственным приложением который распространяет подобного вида контент.

Среди таких приложений есть такие гиганты как Epic Games — компания из США, занимающаяся разработкой компьютерных игр. Штаб-квартира расположена в городе Кэри, штат Северная Каролина. Компания известна по игровым сериям Gears of War и Unreal Tournament, а также как создатели игрового движка Unreal Engine. (рисунок 1.5.1)

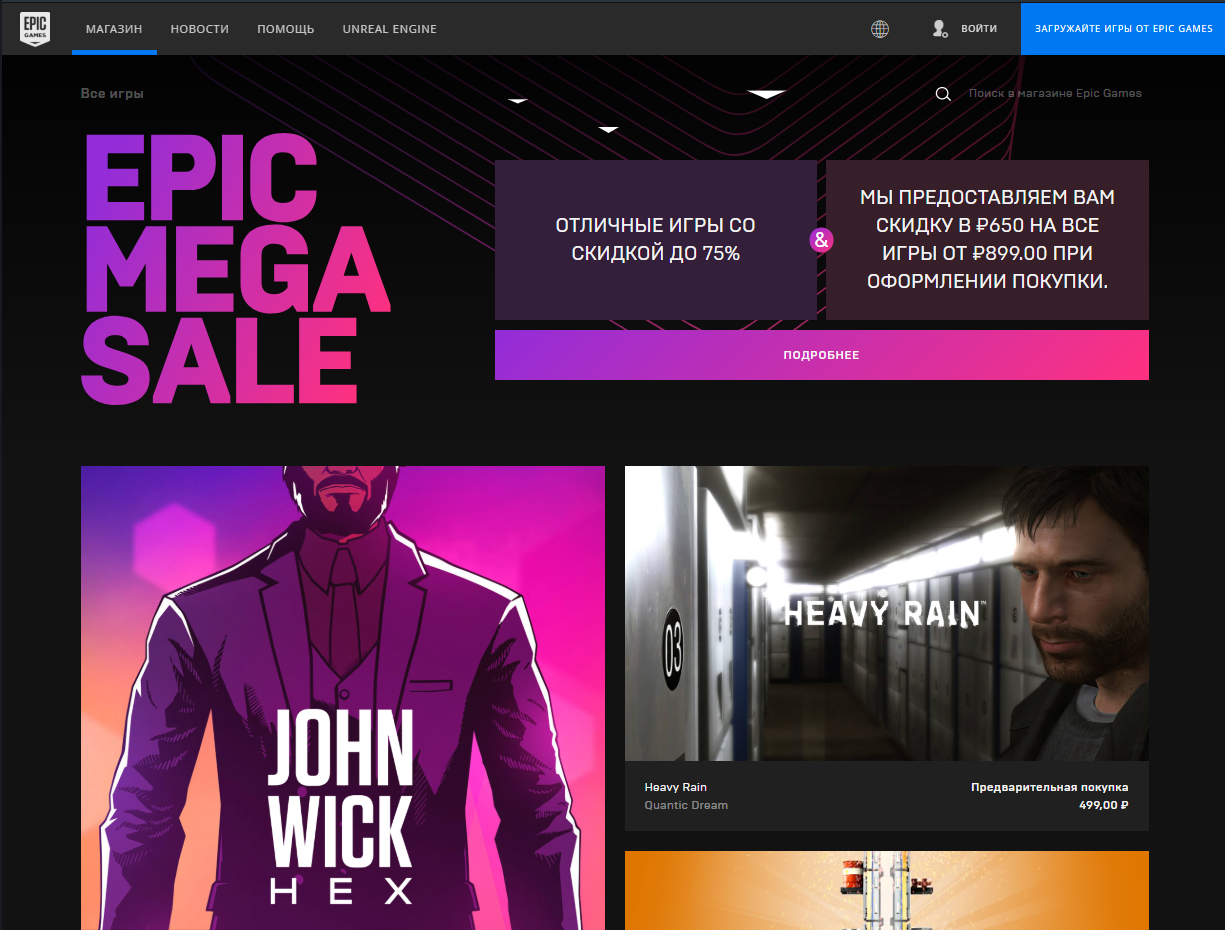


Рисунок 1.5.1– Общая информация приложения «Epic Games»

Battle.net — игровой онлайн-сервис, включающий функции цифровой дистрибуции и социальной платформы, разработанный Blizzard Entertainment. Battle.net был запущен 30 ноября 1996 года одновременно с выходом ролевой игры Diablo.

Battle.net стал первой игровой онлайн-платформой, встроенной прямо в игры, в которых он используется, что отличалось от подхода других компаний, использующих внешние интерфейсы. (рисунок 1.5.2)

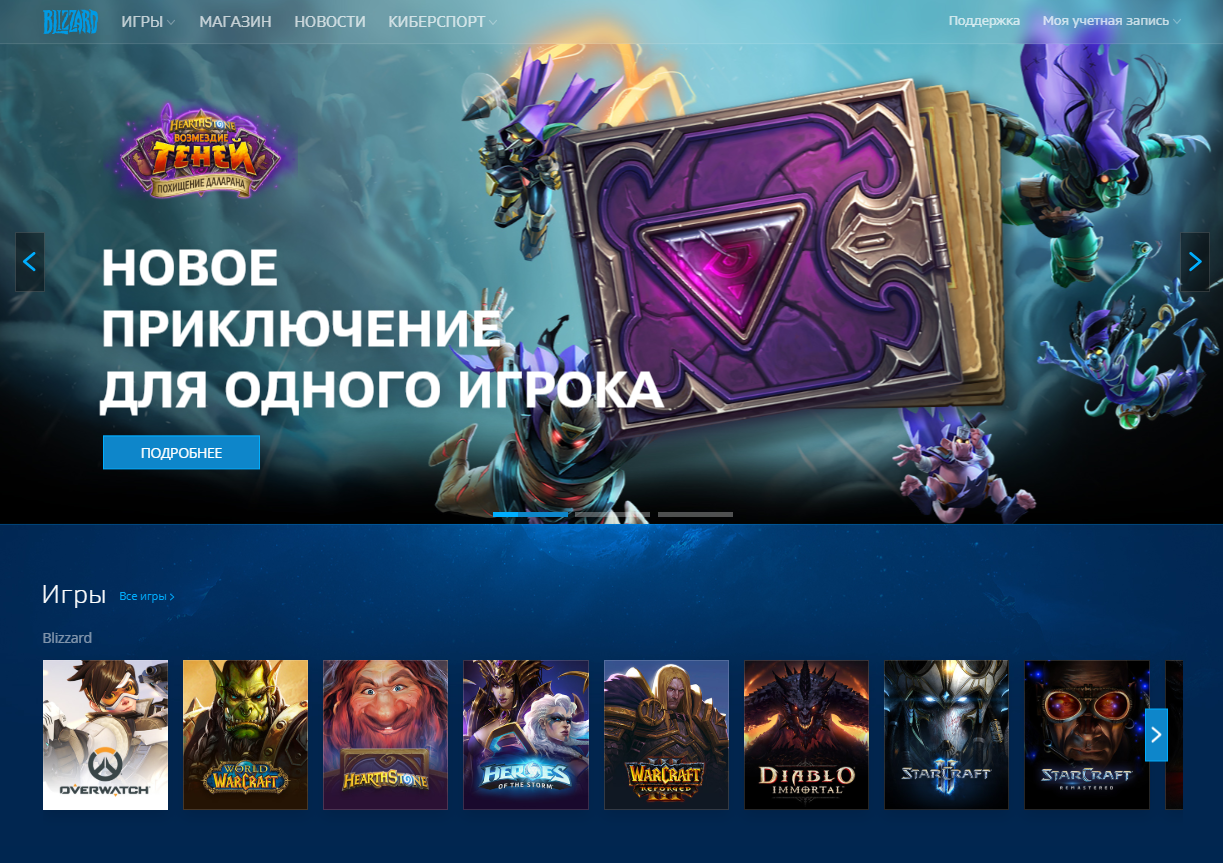


Рисунок 1.5.2– Общая информация приложения «Battle.net»

Origin — алгоритм цифровой дистрибуции компании Electronic Arts, которая даёт возможность пользователям приобретать компьютерные игры через Интернет и загружать их с помощью клиента Origin. (рисунок 1.5.3)

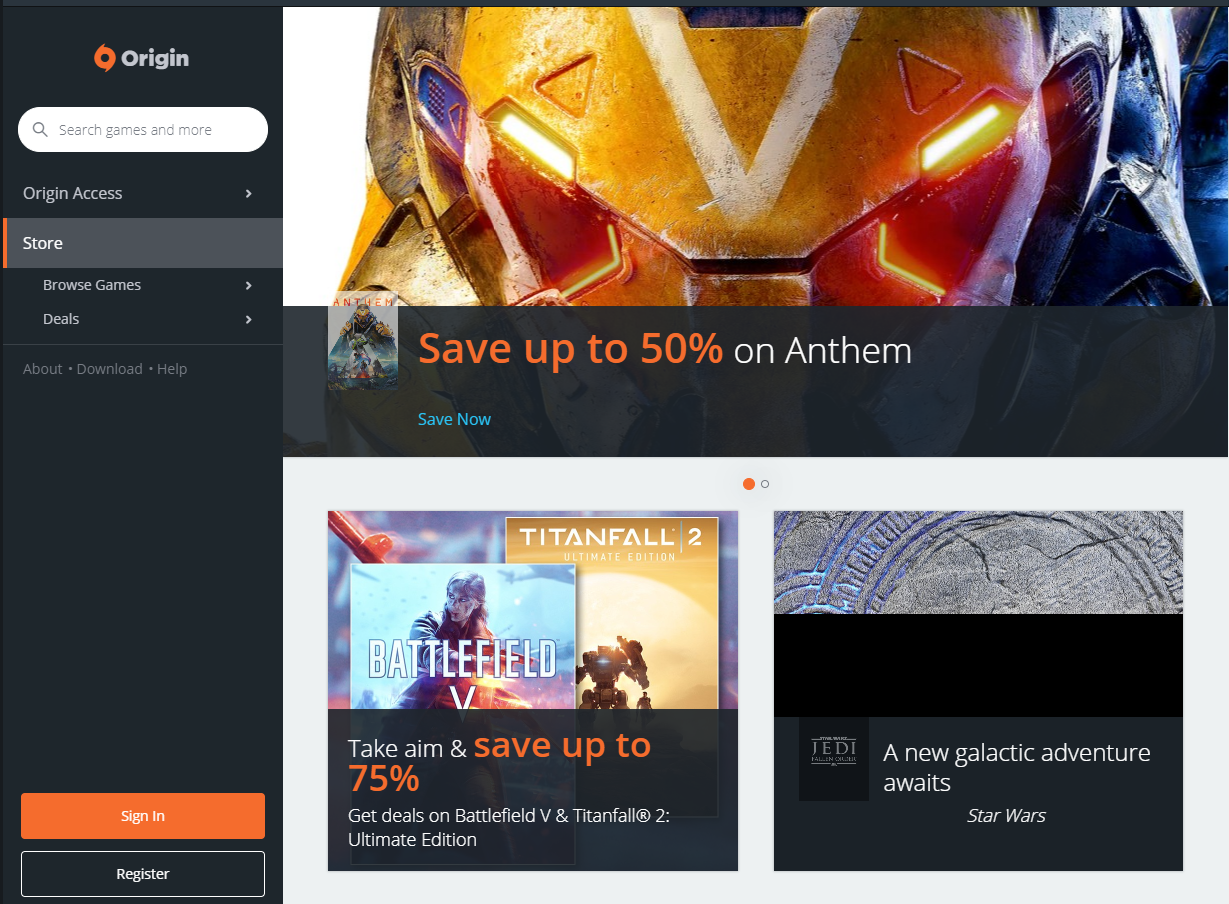


Рисунок 1.5.3– Общая информация приложения «Origin»

## 1.3 Задачи приложения

Как сказано выше, программа должна быть предназначена для различной аудитории пользователей. Это значит, что приложение должно быть доступным. В приложении должна соблюдаться минимальная конфиденциальность, т.е. пользователь может быть кем угодно.

Приложение должно отслеживать счет, который набирает пользователь, играя в определенные игры, предоставляемые платформой, а также отслеживать топ игроков.

**2 Разработка архитектуры проекта**

**2.1 Выбор используемых технологий**

Данное программное обеспечение разрабатывалось для работы в операционной системе Windows.

В качестве интерфейса прикладного программирования был выбран обширный API-интерфейс — **Windows Presentation Foundation** (WPF), предназначенный для создания настольных программ с графически насыщенным пользовательским интерфейсом. В основе графической технологии WPF лежит мощная инфраструктура, основанная на DirectX. Это является одним из основных отличий WPF от более ранней технологии создания пользовательских интерфейсов — Windows Forms.

Для работы с WPF использовался объектно-ориентированный язык программирования с С-подобным синтаксисом — С#, разработанный для создания приложений на платформе Microsoft .NET Framework.

Кроме того, используемая в WPF модель разделения кода и дизайна предполагала работу с декларативным языком описания интерфейса — eXtensible Application Markup Language (XAML).

Для хранения данных пользователей используется реляционная база данных типа SQL. В приложении доступны программные средства, позволяющие отслеживать статистику, а также статистику первых четырех игроков.

Информация для заполнения базы данных это определенный набор очков, который предоставляется базе после выигрыша или проигрыша в определенные игры. Заполнение происходит автоматически, пользователю достаточно просто сыграть в одну из игр.

Связь в чате приложения осуществляется с помощью асинхронных сокетов, предоставляющих низкоуровневый интерфейс для приема и отправки сообщений по сети (в двоичном формате).

**2.2 Структура проекта**

Проект состоит из различных элементов, таких как классы, окна, пользовательские элементы управления. Структура проекта представлена на рисунке 2.1.

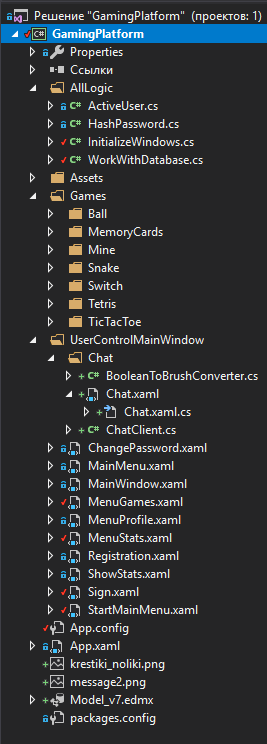


Рисунок 2.1 – Структура проекта

Основными являются окна MainMenu.xaml и MainWindow.xaml. В них хранится базовая визуализация элементов управления и методы, которые взаимодействуют с ними и сущностями.

MenuGames.xaml представляет собой окно, предназначенное для вывода набора игр. Оно должно открываться пользователем для того чтобы начать играть в одну из игр.

ChangePassword.xaml представляет собой окно, предназначенное для замены пароля.

Помимо двух основных окон есть ещё два окна: Registration.xaml, Sign.xaml. Они используются для регистрации и авторизации пользователей.

В папке Games находятся все разработанные игры, которые используются в приложении. Рассмотрим их подробнее. (рисунок 2.1.1)

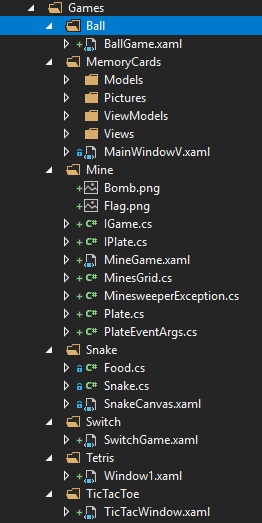


Рисунок 2.1.1 – Папка с играми

Игра Ball представляет из себя платформу и мяч, где мяч должен разбивать черные платформы. Целью игры является набор очков, которые начисляются по мере уничтожения платформ. (рисунок 2.1.2)

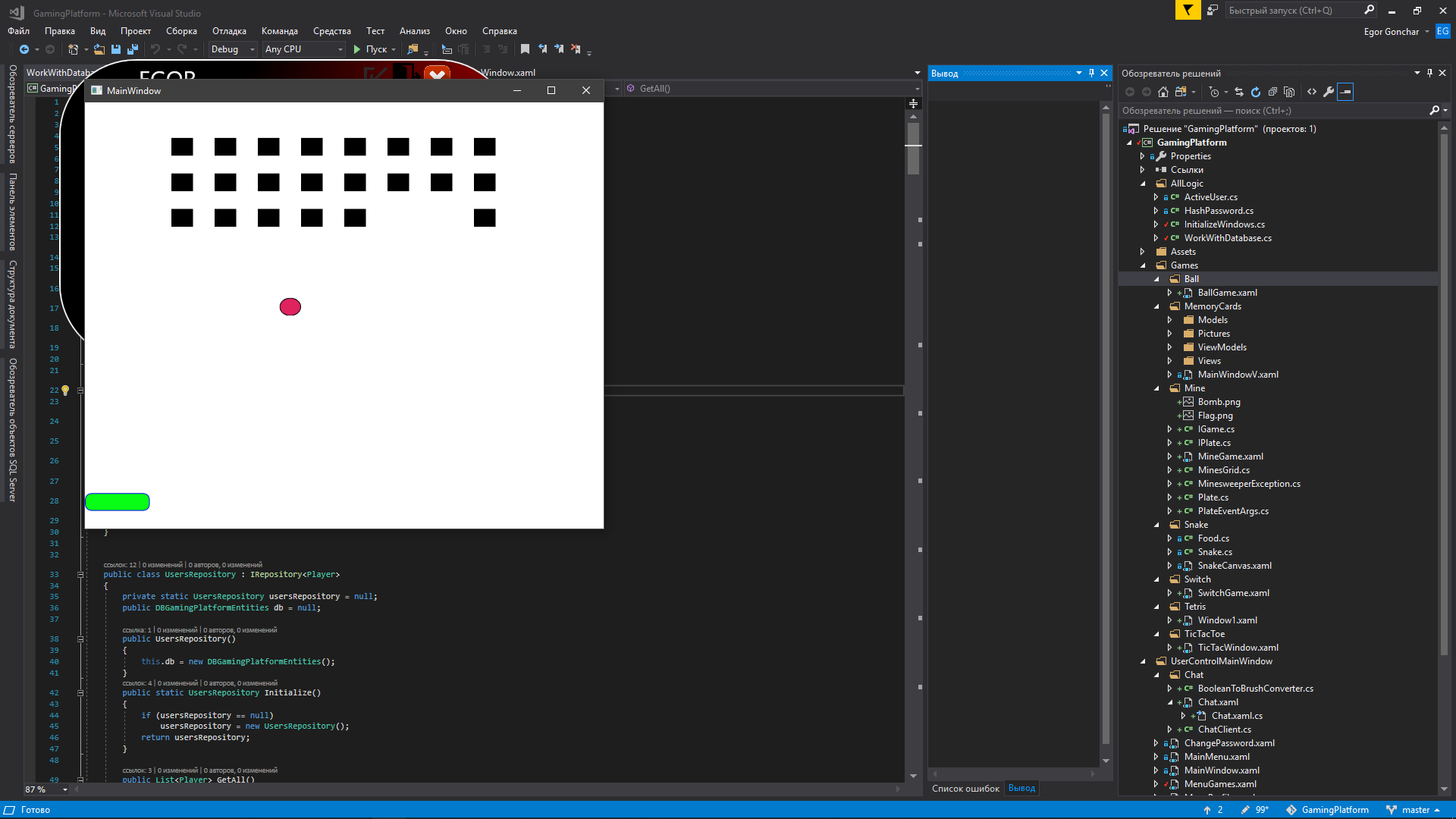


Рисунок 2.1.2 – Ball

MemoryCards. Целью игры является набор очков, которые начисляются при открытии верных совпадающих картинок. (рисунок 2.1.3)

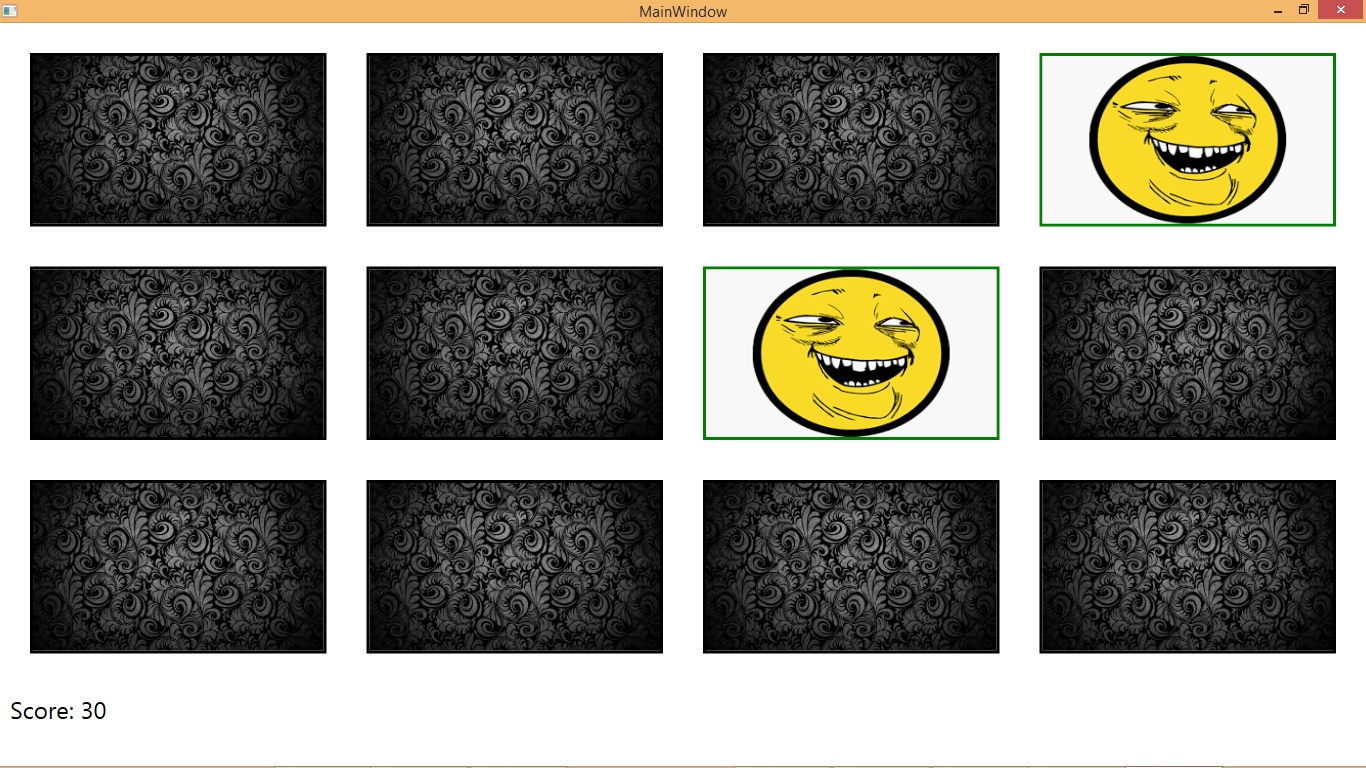


Рисунок 2.1.3 – MemoryCards

Snake. Целью игры является набор очков, которые начисляются, когда «змейка» съедает точку. (рисунок 2.1.4)

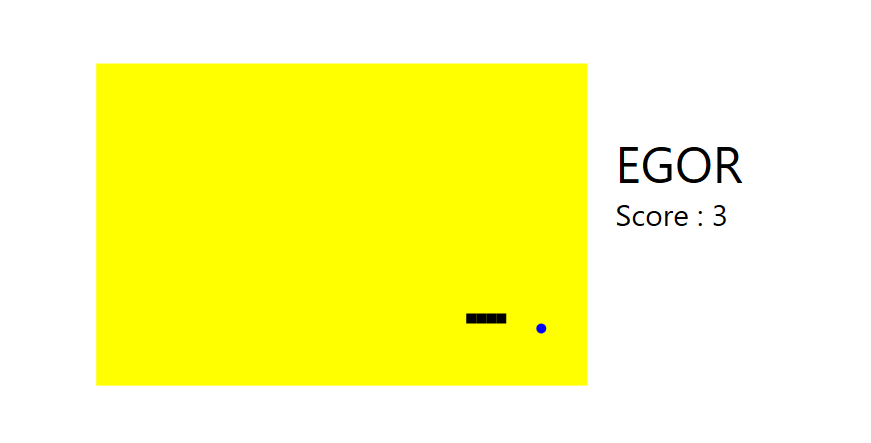


Рисунок 2.1.3 – Snake

Mine. Целью игры является «разминирование» области 10 на 10 где находится 15 мин. Игра считается законченной, если были найдены все 15 мин. В счет идет то время которое вы потратили на «разминирование». (рисунок 2.1.4)

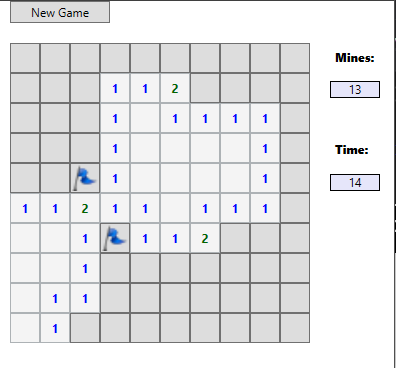


Рисунок 2.1.4 – Mine

Tetris. Игра, которая представляет из себя набор падающих геометрических фигур которые, складываясь в один ряд, уничтожаются. После уничтожения ряда происходит начисление очков. (рисунок 2.1.5)

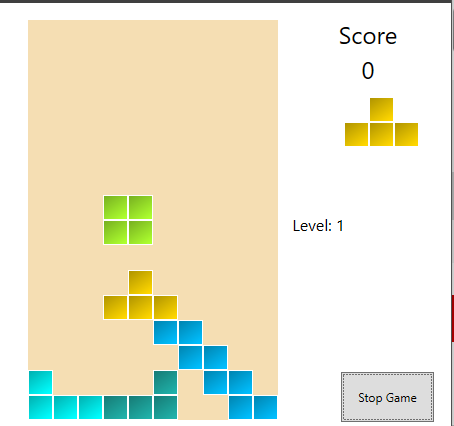


Рисунок 2.1.5 – Tetris

Switch. В этой игре надо переставлять ячейки местами так, чтобы они стали в правильном порядке. В счет засчитывается то количество перестановок, которые игрок совершил до победы. (рисунок 2.1.6)

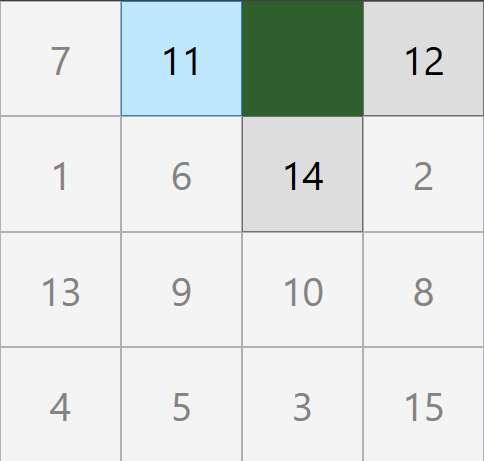


Рисунок 2.1.6 – Switch

TicTacToe. В этой игре играют два пользователя. Как только вы нажмете начать игру сразу же начнется поиск соперника. Игра закончится, как только один из соперников соберёт в ряд три крестика или три нолика. Игра работает через постоянный запрос к базе данных и делая ход вы изменяете значение в базе данных. (рисунок 2.1.7)

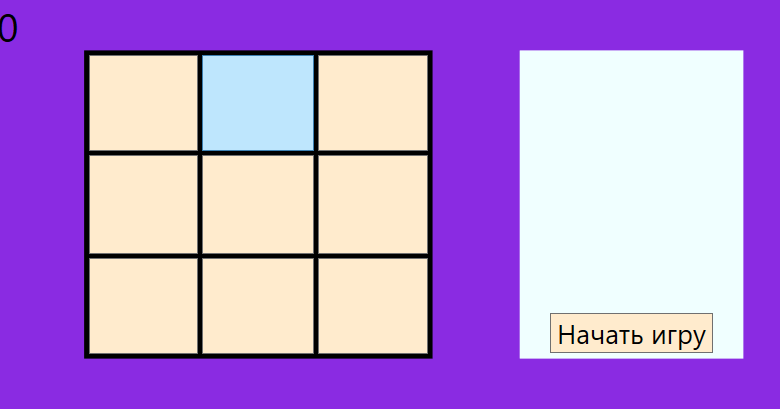


Рисунок 2.1.7 – TicTacToe

MenuProfile.xaml представляет собой окно, где отображается минимальная информация о пользователе.

MenuStats.xaml представляет собой окно, где можно выбрать одну из игр и ознакомится с рекордами, которые люди поставили в этой игре.

Chat представляет из себя клиент-серверный UserController, при помощи которого пользователи могут вести диалог друг с другом по средствам клиент серверной архитектуры. Отдельно запускается сервер, который получает все сообщения, отправленные пользователями, друг другу. Связь в чате приложения осуществляется с помощью асинхронных сокетов, предоставляющих низкоуровневый интерфейс для приема и отправки сообщений по сети (в двоичном формате). (рисунок 2.1.8)

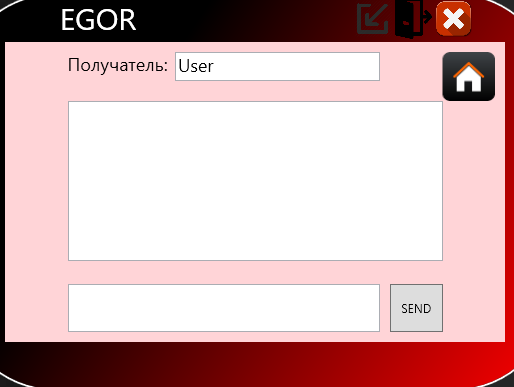


Рисунок 2.1.8 – Chat

Код основных классов приложения представлен в приложении А.

**2.3 Разработка классов**

На рисунке 2.3 представлена диаграмма классов взаимодействия игр с главным интерфейсом.

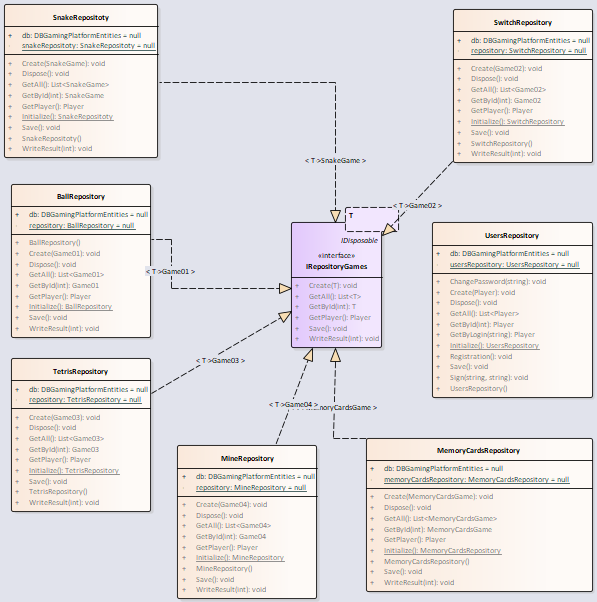


Рисунок 2.3 – Диаграмма классов

Как видно из рисунка, на диаграмме представлена только часть классов, взаимодействующих при записи игр и отправке их в базу данных. На диаграмме можно рассмотреть содержимое классов (поля, методы, свойства) и ассоциативные связи в проекте. Для работы с бд были применены паттерны Repository и Singleton.

**2.4 Проектирование базы данных**

Для реализации поставленной задачи была создана база данных GamingPlatformDB\_v4. Для её создания использовались средства MS SQL. База данных состоит из 8 таблиц, представленных на рисунке 2.4.

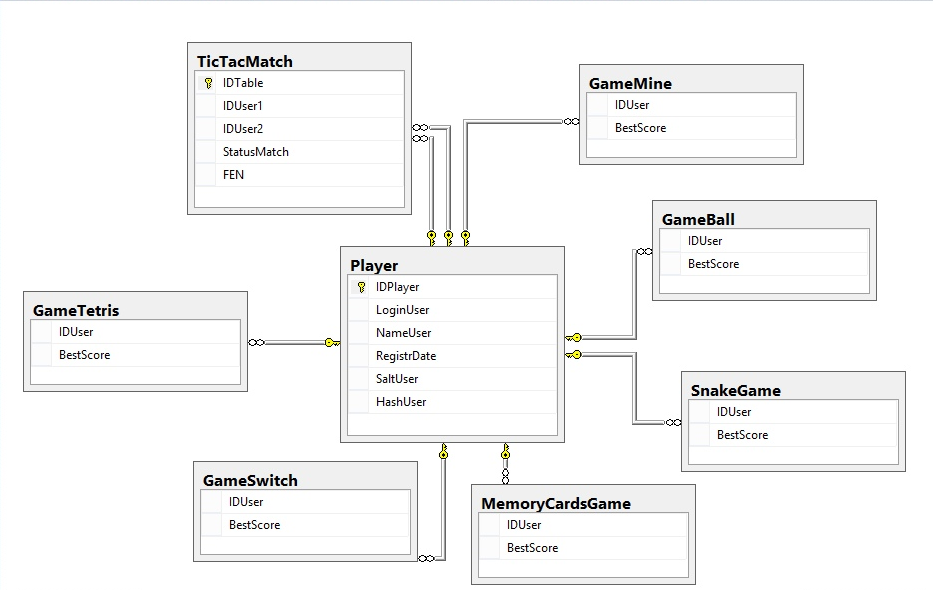


Рисунок 2.4 – Диаграмма базы данных

В таблице Player хранятся все пользователи, зарегистрированные в программе, в таблицах GameBall, SnakeGame, GameMine, MemoryCardsGame, GameSwitch, GameTetris, TicTacMatch, – все результаты.

Таблица Player состоит из шести столбцов:

* IDPlayer – хранит идентификатор пользователя;
* LoginUser – хранит логин пользователя;
* NameUser – хранит имя пользователя;
* RegistrDate – хранит дату регистрации пользователя;
* SaltUser – преобразует пароль в хэш;
* HashUser – хранит хэш пароля;.

Таблицы GameBall, SnakeGame, GameMine, MemoryCardsGame, GameSwitch, GameTetris состоят из 2 столбцов:

* IDUser – хранит идентификатор пользователя;
* BestScore – хранит лучший результат пользователя в игре GameBall;

Таблицы TicTacMatch состоят из 5 столбцов:

* IDTable – ID стола;
* IDUser1 – хранит идентификатор первого пользователя;
* IDUser2 – хранит идентификатор второго пользователя;
* FEN – Хранит позицию матча;
* StatusMatch – Хранит статус матча.

Все таблицы в базе данных зависят от главной таблицы Player.

Скрипты создания таблиц представлены в приложении Б.

**3 Тестирование**

Программа прошла тестирование на операционной системе Windows 8/10. Все функции программы были протестированы не один раз и в различных комбинациях.

Тестирование авторизации представлено в таблице 3.1.

Таблица 3.1 – Тестирование авторизации

|  |  |  |
| --- | --- | --- |
| **Название теста** | **Действие** | **Результат** |
| Авторизация. | Ввести имя пользователя и пароль, нажать кнопку «Войти». | Данные прошли про­верку. Загрузилось глав­ное окно. |

Тестирование регистрации представлено в таблице 3.2.

Таблица 3.2 – Тестирование регистрации

|  |  |  |
| --- | --- | --- |
| **Название теста** | **Действие** | **Результат** |
| Регистрация. | Ввести имя пользователя, логин, пароль и подтвер­ждение пароля, нажать кнопку «Зарегистриро­ватся». | Данные прошли проверку и были сохранены в базе данных. |

Тестирование отправки сообщений между пользователями представлено в таблице 3.3.

Таблица 3.3 – Тестирование отправки сообщений между пользователями

|  |  |  |
| --- | --- | --- |
| **Название теста** | **Действие** | **Результат** |
| Отправка сообщений между пользовате­лями. | Загрузить чат кликнув на иконку в главном меню у двух пользователей. От­править сообщение друг другу. | Оба пользователя полу­чили сообщения. |

Тестирование игр (кроме крестиков-ноликов) представлено в таблице 3.4.

Таблица 3.4 – Тестирование игр (кроме крестиков-ноликов)

|  |  |  |
| --- | --- | --- |
| **Название теста** | **Действие** | **Результат** |
| Игры. | Вход в меню игр. Загрузка одной из игр. Поиграть. | После выигрыша/проигрыша происходит проверка результата с лучшим и запись большего в бд. |

Тестирование крестиков-ноликов представлено в таблице 3.5.

Таблица 3.5 – Тестирование крестиков-ноликов

|  |  |  |
| --- | --- | --- |
| **Название теста** | **Действие** | **Результат** |
| Крестики-нолики. | Загрузка игры крестики-нолики у двух пользователей. Начать игру. | Запуск игры у обоих пользователей. |

Тестирование чата (без включения сервера) представлено в таблице 3.6.

Таблица 3.6 – Тестирование чата (без включения сервера)

|  |  |  |
| --- | --- | --- |
| **Название теста** | **Действие** | **Результат** |
| Чат (без включения сервера). | Загрузить чат кликнув на иконку в главном меню. | Выводит ошибку о подключении. |

Тестирование чата (с включением сервера) представлено в таблице 3.7.

Таблица 3.7 – Тестирование чата (с включением сервера)

|  |  |  |
| --- | --- | --- |
| **Название теста** | **Действие** | **Результат** |
| Чат (с включением сервера). | Загрузить чат кликнув на иконку в главном меню. | Выводит окно с полями ввода. Производит отправку сообщений между пользователями. |

Тестирование просмотра лучшего результата представлено в таблице 3.8.

Таблица 3.8 – Тестирование просмотра лучшего результата

|  |  |  |
| --- | --- | --- |
| **Название теста** | **Действие** | **Результат** |
| Просмотр лучшего результата. | Зайти в меню игр и выбрать статистику о одной из игр. | Извлечение всех лучших результатов пользователей. Сортировка их в зависимости от игры. Вывод четырех лучших игроков с наивысшим результатом |

Тестирование изменения пароля представлено в таблице 3.9.

Таблица 3.9 – Тестирование изменения пароля

|  |  |  |
| --- | --- | --- |
| **Название теста** | **Действие** | **Результат** |
| Изменение пароля. | Нажать на кнопку изменения пароля в информации об аккаунте. Ввести старый пароль, новый пароль и подтвердить его. | Замена старого пароля на новый. |

**4 Руководство пользователя**

После запуска приложения пользователь увидит окно регистрации, представленное на рисунке 4.1.

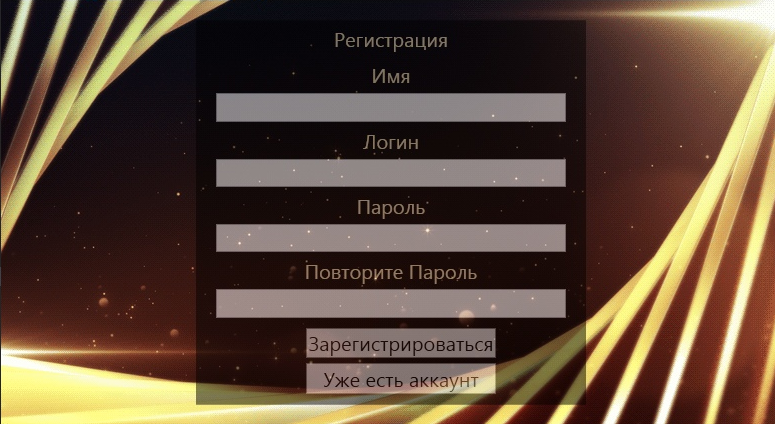


Рисунок 4.1 – Окно регистрации

В данном окне необходимо ввести имя и логин пользователя, а также пароль для регистрации в приложении. Если пользователь не нуждается в регистрации, то можно перейти в окно авторизации, нажав ЛКМ на кнопку с надписью «Уже есть аккаунт».

Окно авторизации представлено на рисунке 4.2.

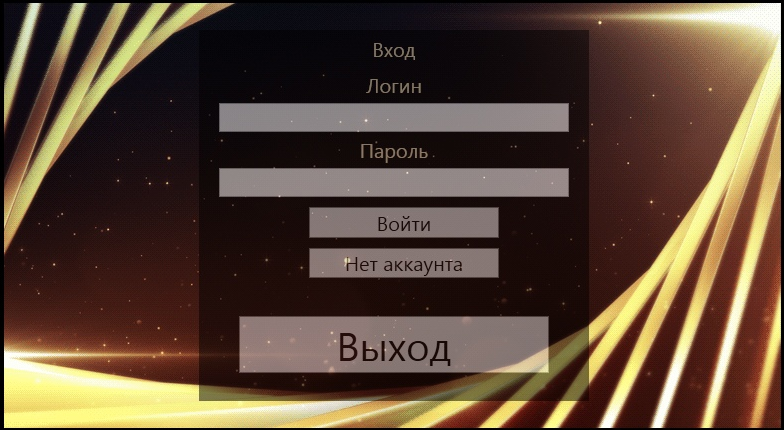


Рисунок 4.2 – Окно авторизации

В данном окне необходимо ввести логин пользователя. Если данные введены правильно, то загружается главное окно.

Главное окно представлено на рисунке 4.3.



Рисунок 4.3 – Главное окно

Если пользователь желает увидеть краткую информации о себе, ему необходимо нажать на кнопку профиля и откроется окно с информацией о пользователе.

Окно статистики профиля на рисунке 4.4.

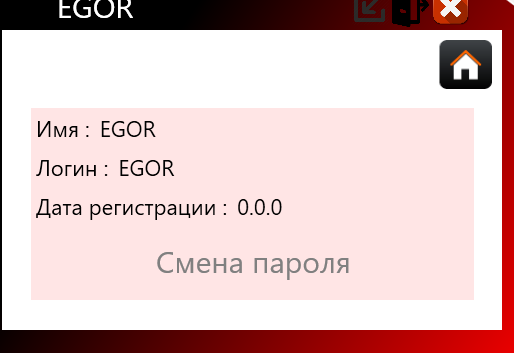


Рисунок 4.4 – профиль

Может возникнуть ситуация, при которой пользователю нужно будет изменить свой пароль в этом случае. Тогда пользователь в окне профиля нажимает на кнопку изменения пароля и вводит старый пароль, новый пароль и подтверждение нового пароля. После нажатия на кнопку «Сменить пароль» старый пароль изменяется на новый.

Смена пароля представлена на рисунке 4.5.

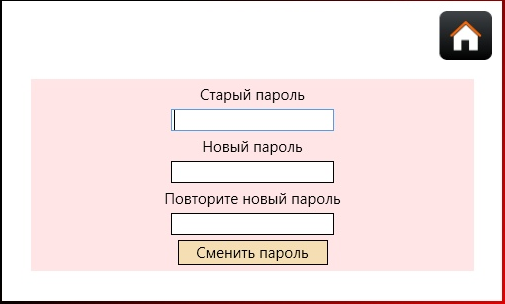


Рисунок 4.5 – Смена пароля

Для того, чтобы общаться в чате надо обеспечить подключение к серверу, которое производится автоматически при входе в чат. Также необходимо знать логин человека, которому вы собираетесь писать. Чат не сохраняет ваши диалоги. После выхода из чата происходит автоматическое отсоединение от сервера. Окно с чатом представлено на рисунке 4.6.

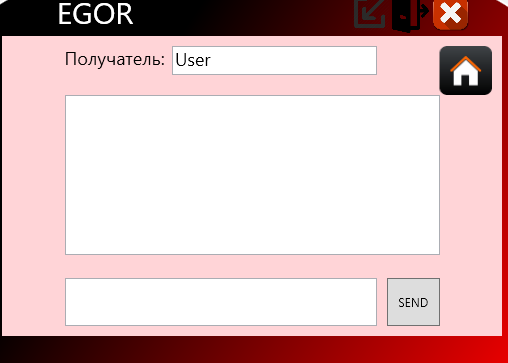


Рисунок 4.6 – чат

Для того чтобы играть в игры необходимо перейти из главного окна в окно с играми нажав на соответствующую кнопку. После нажатия на кнопку пользователя переходит на страницу с играми. Окно с играми представлено на рисунке 4.7.

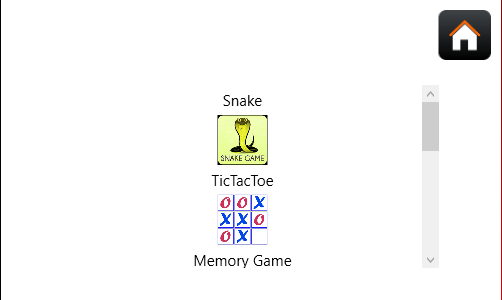


Рисунок 4.7 – Окно игр

Для того чтобы играть в игры необходимо нажать на один из значков в меню игр. Весь набор игр можно посмотреть в пункте 2.2.

Принцип записи результата игр был описан раньше.

На каждом окне приложения есть элемент, который перенаправляет на главное окно. Этот элемент представлено на рисунке 4.8.

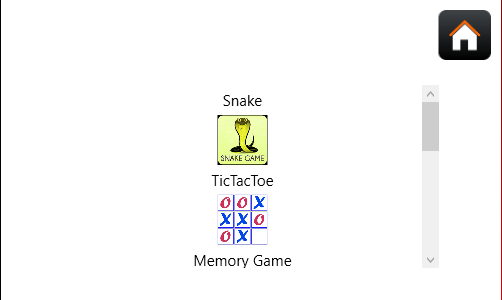


Рисунок 4.7 – Элемент перенаправления на главный экран

Так же в приложении есть набор функций управления окном. В эти функции входит закрытие и сворачивание окна.

**Заключение**

При выполнении курсового проекта было создано приложение «Gaming Platform».

Для разработки программы использовался язык программирования C# и фрейморк WPF. В качестве среды разработки были использованы: Visual Studio 2017.

Созданная программа соответствует основным требованиям:

* Играть в определенный набор игр;
* Соревноваться с другими пользователями;
* Общаться с другими пользователями;

Тестирование программы показало, что она работает корректно и выполняет все свои функции.

**Список использованных источников**

1. MSDN сеть разработчиков в Microsoft [Электронный ресурс]. – Электронные данные. – Режим доступа: http://msdn.microsoft.com/library/rus/.

2. Джеффри Рихтер. Программирование на платформе Microsoft .NET Framework СПб.: Русская редакция, – 2005. – 512 с.

3. METANIT.COM – Сайт о программировании [Электронный ресурс]. – Электронные данные. – Режим доступа: https://metanit.com/.

4. Stack Overflow Business Solutions [Электронный ресурс]. – Электронные данные. – Режим доступа: https://stackoverflow.com/.

# **ПРИЛОЖЕНИЕ А**

**AllLogic**

**ActiveUser.cs**

public class ActiveUser

{

public static int UserId { get; set; }

public static string UserName { get; set; }

public static string UserLogin { get; set; }

public static string UserPassword { get; set; }

public static string DateRegistr { get; set; }

}

**HashPassword.cs**

class HashPassword

{

public string Hash { get; private set; }

public string Salt { get; private set; }

public HashPassword(string password)

{

var saltBytes = new byte[32];

new Random().NextBytes(saltBytes);

Salt = Convert.ToBase64String(saltBytes);

var passwordAndSaltBytes = Concat(password, saltBytes);

Hash = ComputeHash(passwordAndSaltBytes);

}

static string ComputeHash(byte[] bytes)

{

using (var sha256 = SHA256.Create())

{

return

Convert.ToBase64String(sha256.ComputeHash(bytes));

}

}

static byte[] Concat(string password, byte[] saltBytes)

{

var passwordBytes = Encoding.UTF8.GetBytes(password);

return passwordBytes.Concat(saltBytes).ToArray();

}

public static bool Verify(string salt, string hash, string password)

{

var saltBytes = Convert.FromBase64String(salt);

var passwordAndSaltBytes = Concat(password, saltBytes);

var hashAttempt = ComputeHash(passwordAndSaltBytes);

return hash == hashAttempt;

}

}

**InitializeWindows.cs**

public class InitializeWindows

{

public static GamingPlatform.UserControlMainWindow.MainMenu mm;

public static GamingPlatform.Games.MemoryCards.MainWindowV mg;

public static GamingPlatform.UserControlMainWindow.MainWindow mw;

public static Games.Snake.SnakeCanvas sc;

public static Minesweeper.WPF.Mine ms;

public static Tetris.Tetris ts;

public static Puzzle.Switch sw;

public static Game.Game\_Ball bl;

public static Games.TicTacToe.TicTacWindow tt;

public static void InitWindows()

{

sc = new Games.Snake.SnakeCanvas();

}

public static void InitWindows\_1()

{

ts = new Tetris.Tetris();

}

public static void CloseAllWindows()

{

mm.Close();

mw.Close();

sc.Close();

mg.Close();

tt.Close();

ms.Close();

ts.Close();

sw.Close();

bl.Close();

}

}

**WorkWithDatabase.cs**

interface IRepository<T> : IDisposable

{

List<T> GetAll();

T GetById(int id);

void Create(T item);

void Save();

}

interface IRepositoryGames<T> : IDisposable

{

List<T> GetAll();

T GetById(int id);

void Create(T item);

void Save();

Player GetPlayer();

void WriteResult(int score);

}

public class UsersRepository : IRepository<Player>

{

private static UsersRepository usersRepository = null;

public DBGamingPlatformEntities db = null;

public UsersRepository()

{

this.db = new DBGamingPlatformEntities();

}

public static UsersRepository Initialize()

{

if (usersRepository == null)

usersRepository = new UsersRepository();

return usersRepository;

}

public List<Player> GetAll()

{

return db.Players.ToList();

}

public void Create(Player item)

{

db.Players.Add(item);

}

public void Dispose()

{

db.Dispose();

}

public Player GetById(int id)

{

return GetAll().Find(item => item.IDPlayer == id);

}

public void Save()

{

db.SaveChanges();

}

public Player GetByLogin(string login)

{

return GetAll().Find(item => item.LoginUser == login);

}

public void Sign(string login, string password)

{

Player result = GetByLogin(login);

if (result == null)

{

MessageBox.Show("Неверный логин или пароль");

}

else if (HashPassword.Verify(result.SaltUser, result.HashUser, password))

{

ActiveUser.UserId = result.IDPlayer;

ActiveUser.UserLogin = result.LoginUser;

ActiveUser.UserName = result.NameUser;

ActiveUser.DateRegistr = result.RegistrDate;

InitializeWindows.mm = new MainMenu();

InitializeWindows.mm.DataContext = new StartMainMenu();

InitializeWindows.mm.playerName.Content = ActiveUser.UserName;

InitializeWindows.mm.Show();

InitializeWindows.mw.Close();

InitializeWindows.mw = new MainWindow();

}

else

{

MessageBox.Show("Неверный логин или пароль");

}

}

public void Registration()

{

Player result = GetByLogin(ActiveUser.UserLogin);

if (result != null)

{

MessageBox.Show("Аккаунт с таким логином уже существует");

}

else

{

Player newUser = new Player();

newUser.LoginUser = ActiveUser.UserLogin;

newUser.NameUser = ActiveUser.UserName;

newUser.RegistrDate = ActiveUser.DateRegistr;

HashPassword hp = new HashPassword(ActiveUser.UserPassword);

newUser.HashUser = hp.Hash;

newUser.SaltUser = hp.Salt;

Create(newUser);

Save();

MessageBox.Show("Аккаунт успешно создан");

//catch (DbEntityValidationException ex)

//{

// string message = "";

// foreach (DbEntityValidationResult validationError in ex.EntityValidationErrors)

// {

// message = "Object: " + validationError.Entry.Entity.ToString();

// foreach (DbValidationError err in validationError.ValidationErrors)

// {

// message = message + err.ErrorMessage + "";

// }

// }

// MessageBox.Show(message);

//}

}

}

public void ChangePassword(string password)

{

Player result = GetByLogin(ActiveUser.UserLogin);

HashPassword hp = new HashPassword(password);

result.HashUser = hp.Hash;

result.SaltUser = hp.Salt;

ActiveUser.UserPassword = password;

Save();

MessageBox.Show("Вы поменяли пароль");

}

}

public class SnakeRepositoty : IRepositoryGames<SnakeGame>

{

private static SnakeRepositoty snakeRepositoty = null;

public DBGamingPlatformEntities db = null;

public SnakeRepositoty()

{

this.db = new DBGamingPlatformEntities();

}

public static SnakeRepositoty Initialize()

{

if (snakeRepositoty == null)

snakeRepositoty = new SnakeRepositoty();

return snakeRepositoty;

}

public void Create(SnakeGame item)

{

db.SnakeGames.Add(item);

}

public void Dispose()

{

db.Dispose();

}

public List<SnakeGame> GetAll()

{

return db.SnakeGames.ToList();

}

public SnakeGame GetById(int id)

{

return GetAll().Find(item => item.IDTable == id);

}

public Player GetPlayer()

{

return db.Players.ToList().Find(item => item.IDPlayer == ActiveUser.UserId);

}

public void Save()

{

db.SaveChanges();

}

public void WriteResult(int score)

{

Player player = GetPlayer();

SnakeGame game = GetAll().Find(item => item.Player == player);

if (game == null)

{

SnakeGame addUser = new SnakeGame();

addUser.Player = player;

addUser.TotalGames = 1;

addUser.BestScore = score;

Create(addUser);

}

else if (score > game.BestScore)

{

game.BestScore = score;

}

Save();

}

}

public class MemoryCardsRepository : IRepositoryGames<MemoryCardsGame>

{

private static MemoryCardsRepository memoryCardsRepository = null;

public DBGamingPlatformEntities db = null;

public MemoryCardsRepository()

{

this.db = new DBGamingPlatformEntities();

}

public static MemoryCardsRepository Initialize()

{

if (memoryCardsRepository == null)

memoryCardsRepository = new MemoryCardsRepository();

return memoryCardsRepository;

}

public void Create(MemoryCardsGame item)

{

db.MemoryCardsGames.Add(item);

}

public void Dispose()

{

db.Dispose();

}

public List<MemoryCardsGame> GetAll()

{

return db.MemoryCardsGames.ToList();

}

public MemoryCardsGame GetById(int id)

{

return GetAll().Find(item => item.IDTable == id);

}

public Player GetPlayer()

{

return db.Players.ToList().Find(item => item.IDPlayer == ActiveUser.UserId);

}

public void Save()

{

db.SaveChanges();

}

public void WriteResult(int score)

{

Player player = GetPlayer();

MemoryCardsGame game = GetAll().Find(item => item.Player == player);

if (game == null)

{

MemoryCardsGame addUser = new MemoryCardsGame();

addUser.Player = player;

addUser.TotalGames = 1;

addUser.BestScore = score;

Create(addUser);

}

else if (score > game.BestScore)

{

game.BestScore = score;

}

Save();

}

}

public class BallRepository : IRepositoryGames<Game01>

{

private static BallRepository repository = null;

public DBGamingPlatformEntities db = null;

public BallRepository()

{

this.db = new DBGamingPlatformEntities();

}

public static BallRepository Initialize()

{

if (repository == null)

repository = new BallRepository();

return repository;

}

public void Create(Game01 item)

{

db.Game01.Add(item);

}

public void Dispose()

{

db.Dispose();

}

public List<Game01> GetAll()

{

return db.Game01.ToList();

}

public Game01 GetById(int id)

{

return GetAll().Find(item => item.IDTable == id);

}

public Player GetPlayer()

{

return db.Players.ToList().Find(item => item.IDPlayer == ActiveUser.UserId);

}

public void Save()

{

db.SaveChanges();

}

public void WriteResult(int score)

{

Player player = GetPlayer();

Game01 game = GetAll().Find(item => item.Player == player);

if (game == null)

{

Game01 addUser = new Game01();

addUser.Player = player;

addUser.TotalGames = 1;

addUser.BestScore = score;

Create(addUser);

}

else if (score > game.BestScore)

{

game.BestScore = score;

}

Save();

}

}

public class SwitchRepository : IRepositoryGames<Game02>

{

private static SwitchRepository repository = null;

public DBGamingPlatformEntities db = null;

public SwitchRepository()

{

this.db = new DBGamingPlatformEntities();

}

public static SwitchRepository Initialize()

{

if (repository == null)

repository = new SwitchRepository();

return repository;

}

public void Create(Game02 item)

{

db.Game02.Add(item);

}

public void Dispose()

{

db.Dispose();

}

public List<Game02> GetAll()

{

return db.Game02.ToList();

}

public Game02 GetById(int id)

{

return GetAll().Find(item => item.IDTable == id);

}

public Player GetPlayer()

{

return db.Players.ToList().Find(item => item.IDPlayer == ActiveUser.UserId);

}

public void Save()

{

db.SaveChanges();

}

public void WriteResult(int score)

{

Player player = GetPlayer();

Game02 game = GetAll().Find(item => item.Player == player);

if (game == null)

{

Game02 addUser = new Game02();

addUser.Player = player;

addUser.TotalGames = 1;

addUser.BestScore = score;

Create(addUser);

}

else if (score < game.BestScore)

{

game.BestScore = score;

}

Save();

}

}

public class TetrisRepository : IRepositoryGames<Game03>

{

private static TetrisRepository repository = null;

public DBGamingPlatformEntities db = null;

public TetrisRepository()

{

this.db = new DBGamingPlatformEntities();

}

public static TetrisRepository Initialize()

{

if (repository == null)

repository = new TetrisRepository();

return repository;

}

public void Create(Game03 item)

{

db.Game03.Add(item);

}

public void Dispose()

{

db.Dispose();

}

public List<Game03> GetAll()

{

return db.Game03.ToList();

}

public Game03 GetById(int id)

{

return GetAll().Find(item => item.IDTable == id);

}

public Player GetPlayer()

{

return db.Players.ToList().Find(item => item.IDPlayer == ActiveUser.UserId);

}

public void Save()

{

db.SaveChanges();

}

public void WriteResult(int score)

{

Player player = GetPlayer();

Game03 game = GetAll().Find(item => item.Player == player);

if (game == null)

{

Game03 addUser = new Game03();

addUser.Player = player;

addUser.TotalGames = 1;

addUser.BestScore = score;

Create(addUser);

}

else if (score > game.BestScore)

{

game.BestScore = score;

}

Save();

}

}

public class MineRepository : IRepositoryGames<Game04>

{

private static MineRepository repository = null;

public DBGamingPlatformEntities db = null;

public MineRepository()

{

this.db = new DBGamingPlatformEntities();

}

public static MineRepository Initialize()

{

if (repository == null)

repository = new MineRepository();

return repository;

}

public void Create(Game04 item)

{

db.Game04.Add(item);

}

public void Dispose()

{

db.Dispose();

}

public List<Game04> GetAll()

{

return db.Game04.ToList();

}

public Game04 GetById(int id)

{

return GetAll().Find(item => item.IDTable == id);

}

public Player GetPlayer()

{

return db.Players.ToList().Find(item => item.IDPlayer == ActiveUser.UserId);

}

public void Save()

{

db.SaveChanges();

}

public void WriteResult(int score)

{

Player player = GetPlayer();

Game04 game = GetAll().Find(item => item.Player == player);

if (game == null)

{

Game04 addUser = new Game04();

addUser.Player = player;

addUser.TotalGames = 1;

addUser.BestScore = score;

Create(addUser);

}

else if (score < game.BestScore)

{

game.BestScore = score;

}

Save();

}

}

**ChatClient.cs**

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Linq;

using System.Net;

using System.Net.Sockets;

using System.Text;

using System.Threading;

using System.Threading.Tasks;

using System.Windows.Threading;

using GamingPlatform.AllLogic;

namespace GamingPlatform.UserControlMainWindow

{

public class ChatClient : INotifyPropertyChanged

{

private Dispatcher \_dispatcher;

private Thread \_thread;

private Socket \_socket;

private IPAddress \_ipAddress;

public string IpAddress

{

get

{

return \_ipAddress.ToString();

}

set

{

if (this.IsClientConnected)

throw new Exception("Can't change this property when server is active");

\_ipAddress = IPAddress.Parse(value);

}

}

private ushort \_port;

public ushort Port

{

get

{

return \_port;

}

set

{

if (this.IsClientConnected)

throw new Exception("Can't change this property when server is active");

this.\_port = value;

}

}

private IPEndPoint \_ipEndPoint => new IPEndPoint(\_ipAddress, \_port);

private bool \_isClientConnected;

public bool IsClientConnected

{

get

{

return \_isClientConnected;

}

private set

{

this.\_isClientConnected = value;

this.NotifyPropertyChanged("IsClientConnected");

this.NotifyPropertyChanged("IsClientDisconnected");

}

}

public bool IsClientDisconnected => !this.IsClientConnected;

private string \_username;

public string Username

{

get

{

return \_username;

}

set

{

this.\_username = value;

if (this.IsClientConnected)

{

this.SetUsername(value);

}

}

}

public BindingList<String> lstChat { get; set; }

#region INotifyPropertyChanged implementation

public event PropertyChangedEventHandler PropertyChanged;

private void NotifyPropertyChanged(string propName) => this.PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propName));

#endregion

public ChatClient()

{

this.\_dispatcher = Dispatcher.CurrentDispatcher;

this.lstChat = new BindingList<string>();

this.IpAddress = "127.0.0.1";

this.Port = 5960;

this.Username = ActiveUser.UserLogin;

}

public static bool IsSocketConnected(Socket s)

{

if (!s.Connected)

return false;

if (s.Available == 0)

if (s.Poll(1000, SelectMode.SelectRead))

return false;

return true;

}

public void SwitchClientState()

{

if (!this.IsClientConnected)

this.Connect();

else

this.Disconnect();

}

private void Connect()

{

if (this.IsClientConnected) return;

this.\_socket = new Socket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.Tcp);

this.\_socket.Connect(this.\_ipEndPoint);

SetUsername(this.Username);

this.\_thread = new Thread(() => this.ReceiveMessages());

this.\_thread.Start();

this.IsClientConnected = true;

}

private void Disconnect()

{

if (!this.IsClientConnected) return;

if (this.\_socket != null && this.\_thread != null)

{

this.\_socket.Shutdown(SocketShutdown.Both);

this.\_socket.Dispose();

this.\_socket = null;

this.\_thread = null;

}

this.lstChat.Clear();

this.IsClientConnected = false;

}

public void ReceiveMessages()

{

while (true)

{

byte[] inf = new byte[1024];

try

{

if (!IsSocketConnected(this.\_socket))

{

this.\_dispatcher.Invoke(new Action(() =>

{

this.Disconnect();

}));

return;

}

int x = this.\_socket.Receive(inf);

if (x > 0)

{

string message = Encoding.Unicode.GetString(inf).Trim('\0');

this.\_dispatcher.Invoke(new Action(() =>

{

this.lstChat.Add(message);

}));

}

}

catch (Exception)

{

this.\_dispatcher.Invoke(new Action(() =>

{

this.Disconnect();

}));

return;

}

}

}

private void SetUsername(string newUsername)

{

string cmd = string.Format("/setname {0}", newUsername);

this.\_socket.Send(Encoding.Unicode.GetBytes(cmd));

}

public void SendMessageTo(string targetUsername, string message)

{

string cmd = string.Format("/msgto {0}:{1}", targetUsername, message);

this.\_socket.Send(Encoding.Unicode.GetBytes(cmd));

}

}

}

**Games**

**BallGame.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Data;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Imaging;

using System.Windows.Navigation;

using System.Windows.Shapes;

using GamingPlatform.AllLogic;

namespace Game

{

/// <summary>

/// Interaction logic for MainWindow.xaml

/// </summary>

public partial class Game\_Ball : Window

{

private bool isGameStarted = false;

private bool isGameOver = false;

private int dx = -1;

private int dy = -1;

private System.Windows.Threading.DispatcherTimer dispatcherTimer;

private List<Rectangle> rectangles = new List<Rectangle>();

private int score = 0;

private int score\_l = 0; //score for KOSTY

private string Temp;

public Game\_Ball()

{

InitializeComponent();

for (int j = 2; j < 7; j = j + 2)

{

for (int i = 4; i < 20; i = i + 2)

{

score = score + 1;

Rectangle rectangle = new Rectangle();

rectangle.Fill = new SolidColorBrush(System.Windows.Media.Colors.Black);

Grid.SetColumn(rectangle, i);

Grid.SetColumnSpan(rectangle, 1);

Grid.SetRow(rectangle, j);

Grid.SetRowSpan(rectangle, 1);

MyGrid.Children.Add(rectangle);

rectangles.Add(rectangle);

}

}

}

private void UserBlock\_KeyDown(object sender, KeyEventArgs e)

{

Console.Out.WriteLine("Wojtek");

if (e.Key == Key.D)

{

int column = Grid.GetColumn(UserBlock);

if (column < 21)

{

Grid.SetColumn(UserBlock, column + 1);

if (!isGameStarted)

{

int ballColumn = Grid.GetColumn(Ball);

Grid.SetColumn(Ball, ballColumn + 1);

}

}

}

else if (e.Key == Key.A)

{

int column = Grid.GetColumn(UserBlock);

if (column > 0)

{

Grid.SetColumn(UserBlock, column - 1);

if (!isGameStarted)

{

int ballColumn = Grid.GetColumn(Ball);

Grid.SetColumn(Ball, ballColumn - 1);

}

}

}

else if (!isGameStarted && e.Key == Key.Space)

{

isGameStarted = true;

dispatcherTimer = new System.Windows.Threading.DispatcherTimer();

dispatcherTimer.Tick += new EventHandler(dispatcherTimer\_Tick);

dispatcherTimer.Interval = new TimeSpan(0, 0, 0, 0, 50);

dispatcherTimer.Start();

}

}

async private void dispatcherTimer\_Tick(object sender, EventArgs e)

{

if (isGameOver)

{

dispatcherTimer.Stop();

return;

}

else if (MyGrid.Children.Count == 2) //score for KOSTY

{

dispatcherTimer.Stop();

MessageBox.Show("You win !");

string scorestr = Convert.ToString(score\_l);

Temp = String.Concat("Your score = ", scorestr);

BallRepository br = BallRepository.Initialize();

await Task.Run(() => { br.WriteResult(score\_l); });

MessageBox.Show(Temp);

return;

}

checkCollisions();

int ballColumn = Grid.GetColumn(Ball);

Grid.SetColumn(Ball, ballColumn + dx);

int ballRow = Grid.GetRow(Ball);

Grid.SetRow(Ball, ballRow + dy);

}

async private void checkCollisions()

{

int userBlockColumn = Grid.GetColumn(UserBlock);

int ballColumn = Grid.GetColumn(Ball);

int ballRow = Grid.GetRow(Ball);

if (ballRow == 21 && dy > 0)

{

if (userBlockColumn >= (ballColumn - 2) && userBlockColumn <= ballColumn)

{

dy = -dy;

}

else if ((userBlockColumn == (ballColumn - 3) && dx < 0) || (userBlockColumn == (ballColumn + 1) && dx > 0))

{

dy = -dy;

dx = -dx;

}

else //score for KOSTY

{

isGameOver = true;

MessageBox.Show("Game over");

BallRepository br = BallRepository.Initialize();

await Task.Run(() => { br.WriteResult(score\_l); });

string scorestr = Convert.ToString(score\_l);

Temp = String.Concat("Your score = ", scorestr);

MessageBox.Show(Temp);

return;

}

}

if ((ballColumn == 0 && dx < 0) || (ballColumn == 23 && dx > 0))

{

dx = -dx;

}

if (ballRow == 0 && dy < 0)

{

dy = -dy;

}

UIElement upDownBlock = MyGrid.Children.Cast<Rectangle>().Where(i => (Grid.GetRow(i) == ballRow + dy) && (Grid.GetColumn(i) == ballColumn)).FirstOrDefault();

if (upDownBlock != null)

{

dy = -dy;

MyGrid.Children.Remove(upDownBlock);

score\_l = score\_l + 1;

}

UIElement leftRightBlock = MyGrid.Children.Cast<Rectangle>().Where(i => (Grid.GetColumn(i) == ballColumn + dx) && (Grid.GetRow(i) == ballRow)).FirstOrDefault();

if (leftRightBlock != null)

{

dx = -dx;

MyGrid.Children.Remove(leftRightBlock);

score\_l = score\_l + 1;

}

UIElement skos = MyGrid.Children.Cast<Rectangle>().Where(i => (Grid.GetColumn(i) == ballColumn + dx) && (Grid.GetRow(i) == ballRow + dy)).FirstOrDefault();

if (skos != null)

{

dx = -dx;

dy = -dy;

MyGrid.Children.Remove(skos);

score\_l = score\_l + 1;

}

}

}

}

**MemoryCards**

**GameInfoVM.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

using GamingPlatform.AllLogic;

namespace GamingPlatform.Games.MemoryCards.ViewModels

{

public class GameInfoVM : ObservableObject

{

private const int maxAttempts = 4;

private const int pointAward = 75;

private const int pointDeduction = 15;

private int matchAttempts;

private int score;

private bool gameLost;

private bool gameWon;

public int MatchAttempts

{

get

{

return matchAttempts;

}

private set

{

matchAttempts = value;

OnPropertyChanged("MatchAttempts");

}

}

public int Score

{

get

{

return score;

}

private set

{

score = value;

OnPropertyChanged("Score");

}

}

public Visibility LostMessage

{

get

{

if (gameLost)

return Visibility.Visible;

return Visibility.Hidden;

}

}

public Visibility WinMessage

{

get

{

if (gameWon)

return Visibility.Visible;

return Visibility.Hidden;

}

}

public void GameStatus(bool win)

{

MemoryCardsRepository mcr = MemoryCardsRepository.Initialize();

mcr.WriteResult(score);

if (!win)

{

gameLost = true;

OnPropertyChanged("LostMessage");

}

if (win)

{

gameWon = true;

OnPropertyChanged("WinMessage");

}

}

public void ClearInfo()

{

Score = 0;

MatchAttempts = maxAttempts;

gameLost = false;

gameWon = false;

OnPropertyChanged("LostMessage");

OnPropertyChanged("WinMessage");

}

public void Award()

{

Score += pointAward;

}

public void Penalize()

{

Score -= pointDeduction;

MatchAttempts--;

}

}

}

**GameVM.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using GamingPlatform.AllLogic;

namespace GamingPlatform.Games.MemoryCards.ViewModels

{

public class GameVM : ObservableObject

{

public SlideCollectionVM Slides { get; private set; }

public GameInfoVM GameInfo { get; private set; }

public GameVM()

{

SetupGame();

}

private void SetupGame()

{

Slides = new SlideCollectionVM();

GameInfo = new GameInfoVM();

GameInfo.ClearInfo();

Slides.CreateSlides("Pictures/Memes");

Slides.Memorize();

OnPropertyChanged("Slides");

OnPropertyChanged("GameInfo");

}

public void ClickedSlide(object slide)

{

if (Slides.canSelect)

{

var selected = slide as PictureVM;

Slides.SelectSlide(selected);

}

if (!Slides.areSlidesActive)

{

if (Slides.CheckIfMatched())

GameInfo.Award();

else

GameInfo.Penalize();

}

GameStatus();

}

private void GameStatus()

{

if (GameInfo.MatchAttempts < 0)

{

GameInfo.GameStatus(false);

Slides.RevealUnmatched();

}

if (Slides.AllSlidesMatched)

{

GameInfo.GameStatus(true);

}

}

public void Restart()

{

SetupGame();

}

}

}

**ObservableObject.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.ComponentModel;

namespace GamingPlatform.Games.MemoryCards.ViewModels

{

public class ObservableObject : INotifyPropertyChanged

{

public event PropertyChangedEventHandler PropertyChanged;

public ObservableObject ObservableObject1

{

get => default(ObservableObject);

set

{

}

}

public void OnPropertyChanged(string propertyName)

{

PropertyChangedEventHandler handler = PropertyChanged;

if (handler != null) handler(this, new PropertyChangedEventArgs(propertyName));

}

}

}

**PictureVM.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using GamingPlatform.Games.MemoryCards.Models;

using System.Windows.Media;

namespace GamingPlatform.Games.MemoryCards.ViewModels

{

public class PictureVM : ObservableObject

{

private PictureModel \_model;

public int Id { get; private set; }

private bool \_isViewed;

private bool \_isMatched;

private bool \_isFailed;

public bool isViewed

{

get

{

return \_isViewed;

}

private set

{

\_isViewed = value;

OnPropertyChanged("SlideImage");

OnPropertyChanged("BorderBrush");

}

}

public bool isMatched

{

get

{

return \_isMatched;

}

private set

{

\_isMatched = value;

OnPropertyChanged("SlideImage");

OnPropertyChanged("BorderBrush");

}

}

public bool isFailed

{

get

{

return \_isFailed;

}

private set

{

\_isFailed = value;

OnPropertyChanged("SlideImage");

OnPropertyChanged("BorderBrush");

}

}

public bool isSelectable

{

get

{

if (isMatched)

return false;

if (isViewed)

return false;

return true;

}

}

public string SlideImage

{

get

{

if (isMatched)

return \_model.ImageSource;

if (isViewed)

return \_model.ImageSource;

return "E:\\GamingPlatform\\GamingPlatform\\Games\\MemoryCards\\Pictures//background.jpg";

}

}

public Brush BorderBrush

{

get

{

if (isFailed)

return Brushes.Red;

if (isMatched)

return Brushes.Green;

if (isViewed)

return Brushes.Yellow;

return Brushes.Black;

}

}

public PictureVM(PictureModel model)

{

\_model = model;

Id = model.Id;

}

public void MarkMatched()

{

isMatched = true;

}

public void MarkFailed()

{

isFailed = true;

}

public void ClosePeek()

{

isViewed = false;

isFailed = false;

OnPropertyChanged("isSelectable");

OnPropertyChanged("SlideImage");

}

public void PeekAtImage()

{

isViewed = true;

OnPropertyChanged("SlideImage");

}

}

}

**SlideCollectionVM.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Collections.ObjectModel;

using System.Windows.Threading;

using GamingPlatform.Games.MemoryCards.Models;

using System.IO;

namespace GamingPlatform.Games.MemoryCards.ViewModels

{

public class SlideCollectionVM : ObservableObject

{

public ObservableCollection<PictureVM> MemorySlides { get; private set; }

private PictureVM SelectedSlide1;

private PictureVM SelectedSlide2;

private DispatcherTimer peekTimer;

private DispatcherTimer openingTimer;

private const int peekSeconds = 1;

private const int openSeconds = 3;

public bool areSlidesActive

{

get

{

if (SelectedSlide1 == null || SelectedSlide2 == null)

return true;

return false;

}

}

public bool AllSlidesMatched

{

get

{

foreach (var slide in MemorySlides)

{

if (!slide.isMatched)

return false;

}

return true;

}

}

public bool canSelect { get; private set; }

public SlideCollectionVM()

{

peekTimer = new DispatcherTimer();

peekTimer.Interval = new TimeSpan(0, 0, peekSeconds);

peekTimer.Tick += PeekTimer\_Tick;

openingTimer = new DispatcherTimer();

openingTimer.Interval = new TimeSpan(0, 0, openSeconds);

openingTimer.Tick += OpeningTimer\_Tick;

}

public void CreateSlides(string imagesPath)

{

MemorySlides = new ObservableCollection<PictureVM>();

var models = GetModelsFrom(imagesPath);

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

{

var newSlide = new PictureVM(models[i]);

var newSlideMatch = new PictureVM(models[i]);

MemorySlides.Add(newSlide);

MemorySlides.Add(newSlideMatch);

newSlide.PeekAtImage();

newSlideMatch.PeekAtImage();

}

ShuffleSlides();

OnPropertyChanged("MemorySlides");

}

public void SelectSlide(PictureVM slide)

{

slide.PeekAtImage();

if (SelectedSlide1 == null)

{

SelectedSlide1 = slide;

}

else if (SelectedSlide2 == null)

{

SelectedSlide2 = slide;

HideUnmatched();

}

OnPropertyChanged("areSlidesActive");

}

public bool CheckIfMatched()

{

if (SelectedSlide1.Id == SelectedSlide2.Id)

{

MatchCorrect();

return true;

}

else

{

MatchFailed();

return false;

}

}

private void MatchFailed()

{

SelectedSlide1.MarkFailed();

SelectedSlide2.MarkFailed();

ClearSelected();

}

private void MatchCorrect()

{

SelectedSlide1.MarkMatched();

SelectedSlide2.MarkMatched();

ClearSelected();

}

private void ClearSelected()

{

SelectedSlide1 = null;

SelectedSlide2 = null;

canSelect = false;

}

public void RevealUnmatched()

{

foreach (var slide in MemorySlides)

{

if (!slide.isMatched)

{

peekTimer.Stop();

slide.MarkFailed();

slide.PeekAtImage();

}

}

}

public void HideUnmatched()

{

peekTimer.Start();

}

public void Memorize()

{

openingTimer.Start();

}

private List<PictureModel> GetModelsFrom(string relativePath)

{

var models = new List<PictureModel>();

var images = Directory.GetFiles(relativePath, "\*.jpg", SearchOption.AllDirectories);

var id = 0;

foreach (string i in images)

{

models.Add(new PictureModel() { Id = id, ImageSource = "E:\\GamingPlatform\\GamingPlatform\\Games\\MemoryCards\\" + i });

id++;

}

return models;

}

private void ShuffleSlides()

{

var rnd = new Random();

for (int i = 0; i < 64; i++)

{

MemorySlides.Reverse();

int aa = rnd.Next(0, MemorySlides.Count);

int bb = rnd.Next(0, MemorySlides.Count);

MemorySlides.Move(aa, bb);

}

}

private void OpeningTimer\_Tick(object sender, EventArgs e)

{

foreach (var slide in MemorySlides)

{

slide.ClosePeek();

canSelect = true;

}

OnPropertyChanged("areSlidesActive");

openingTimer.Stop();

}

private void PeekTimer\_Tick(object sender, EventArgs e)

{

foreach (var slide in MemorySlides)

{

if (!slide.isMatched)

{

slide.ClosePeek();

canSelect = true;

}

}

OnPropertyChanged("areSlidesActive");

peekTimer.Stop();

}

}

}

**StartMenuVM.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace GamingPlatform.Games.MemoryCards.ViewModels

{

class StartMenuVM

{

private MainWindowV mainWindow;

public StartMenuVM(MainWindowV main)

{

mainWindow = main;

}

public void StartNewGame()

{

GameVM newGame = new GameVM();

mainWindow.DataContext = newGame;

}

}

}

**Mine**

**IGame.cs**

using System;

namespace Minesweeper.WPF

{

public interface IGame

{

event EventHandler CounterChanged;

event EventHandler TimerThresholdReached;

event EventHandler<PlateEventArgs> ClickPlate;

void Run();

}

}

**IPlate.cs**

using System;

namespace Minesweeper.WPF

{

interface IPlate

{

int RowPosition { get; }

int ColPosition { get; }

}

}

MinesGrid.cs

using System;

using System.Windows.Threading;

using System.Windows;

using GamingPlatform.AllLogic;

namespace Minesweeper.WPF

{

public class MinesGrid : IGame

{

public event EventHandler CounterChanged;

public event EventHandler TimerThresholdReached;

public event EventHandler<PlateEventArgs> ClickPlate;

public int Width { get; private set; }

public int Height { get; private set; }

public int Mines { get; private set; }

public static int TimeElapsed { get; private set; }

private Plate[,] plates;

private int correctFlags;

private int wrongFlags;

public int FlaggedMines { get { return (this.correctFlags + this.wrongFlags); } }

private DispatcherTimer gameTimer;

private int score = 0;

public double TotalMilliseconds { get; }

public MinesGrid(int width, int height, int mines)

{

this.Width = width;

this.Height = height;

this.Mines = mines;

}

public bool IsInGrid(int rowPosition, int colPosition)

{

return ((rowPosition >= 0) && (rowPosition < this.Width) && (colPosition >= 0) && (colPosition < this.Height));

}

public bool IsBomb(int rowPosition, int colPosition)

{

if (this.IsInGrid(rowPosition, colPosition))

{

return this.plates[rowPosition, colPosition].IsMined;

}

return false;

}

public bool IsFlagged(int rowPosition, int colPosition)

{

if (this.IsInGrid(rowPosition, colPosition))

{

return this.plates[rowPosition, colPosition].IsFlagged;

}

return false;

}

public int RevealPlate(int rowPosition, int colPosition)

{

if (this.IsInGrid(rowPosition, colPosition))

{

int result = this.plates[rowPosition, colPosition].Check();

CheckFinish();

return result;

}

throw new MinesweeperException("Invalid MinesGrid reference call [row, column] on reveal");

}

public void FlagMine(int rowPosition, int colPosition)

{

if (!this.IsInGrid(rowPosition, colPosition))

{

throw new MinesweeperException("Invalid MinesGrid reference call [row, column] on flag");

}

Plate currPlate = this.plates[rowPosition, colPosition];

if (!currPlate.IsFlagged)

{

if (currPlate.IsMined)

{

this.correctFlags++;

}

else

{

this.wrongFlags++;

}

}

else

{

if (currPlate.IsMined)

{

this.correctFlags--;

}

else

{

this.wrongFlags--;

}

}

currPlate.IsFlagged = !currPlate.IsFlagged;

CheckFinish();

this.OnCounterChanged(new EventArgs());

}

public void OpenPlate(int rowPosition, int colPosition)

{

if (this.IsInGrid(rowPosition, colPosition) && !this.plates[rowPosition, colPosition].IsRevealed)

{

this.OnClickPlate(new PlateEventArgs(rowPosition, colPosition));

}

}

async private void CheckFinish()

{

bool hasFinished = false;

if (this.wrongFlags == 0 && this.FlaggedMines == this.Mines)

{

hasFinished = true;

foreach (Plate item in this.plates)

{

if (!item.IsRevealed && !item.IsMined)

{

hasFinished = false;

break;

}

}

}

if (hasFinished)

{

score = TimeElapsed;

gameTimer.Stop();

MessageBox.Show(score.ToString()); //score for KOSTY

MineRepository mr = MineRepository.Initialize();

await System.Threading.Tasks.Task.Run(() => { mr.WriteResult(score); });

}

}

public void Run()

{

this.correctFlags = 0;

this.wrongFlags = 0;

TimeElapsed = 0;

this.plates = new Plate[Width, Height];

for (int row = 0; row < Width; row++)

{

for (int col = 0; col < Height; col++)

{

Plate cell = new Plate(this, row, col);

this.plates[row, col] = cell;

}

}

int minesCounter = 0;

Random minesPosition = new Random();

while (minesCounter < Mines)

{

int row = minesPosition.Next(Width);

int col = minesPosition.Next(Height);

Plate cell = this.plates[row, col];

if (!cell.IsMined)

{

cell.IsMined = true;

minesCounter++;

}

}

gameTimer = new DispatcherTimer();

gameTimer.Tick += new EventHandler(OnTimeElapsed);

gameTimer.Interval = new TimeSpan(0, 0, 1);

gameTimer.Start();

}

public void Stop()

{

gameTimer.Stop();

}

protected virtual void OnCounterChanged(EventArgs e)

{

EventHandler handler = CounterChanged;

if (handler != null)

{

handler(this, e);

}

}

protected virtual void OnTimeElapsed(object sender, EventArgs e)

{

TimeElapsed++;

EventHandler handler = TimerThresholdReached;

if (handler != null)

{

handler(this, e);

}

}

protected virtual void OnClickPlate(PlateEventArgs e)

{

EventHandler<PlateEventArgs> handler = ClickPlate;

if (handler != null)

{

handler(this, e);

}

}

}

}

**Plate.cs**

using System;

namespace Minesweeper.WPF

{

public class Plate : IPlate

{

public MinesGrid GameGrid { get; private set; }

public int RowPosition { get; private set; }

public int ColPosition { get; private set; }

public bool IsFlagged { get; set; }

public bool IsMined { get; set; }

public bool IsRevealed { get; private set; }

public Plate(MinesGrid grid, int rowPosition, int colPosition)

{

this.GameGrid = grid;

this.RowPosition = rowPosition;

this.ColPosition = colPosition;

}

public int Check()

{

int counter = 0;

if (!IsRevealed && !IsFlagged)

{

IsRevealed = true;

for (int i = 0; i < 9; i++)

{

if (i == 4) continue;

if (GameGrid.IsBomb(RowPosition + i / 3 - 1, ColPosition + i % 3 - 1)) counter++; // if there is a bomb, counts it

}

if (counter == 0)

{

for (int i = 0; i < 9; i++)

{

if (i == 4) continue;

GameGrid.OpenPlate(RowPosition + i / 3 - 1, ColPosition + i % 3 - 1); // reveal all neighbours

}

}

}

return counter;

}

}

}

**MineGame.cs**

using System;

using System.Linq;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Imaging;

namespace Minesweeper.WPF

{

/// <summary>

/// Interaction logic for MainWindow.xaml

/// </summary>

public partial class Mine : Window

{

private int nrMines;

public MinesGrid Mines { get; private set; }

private bool gameStarted;

private Color[] mineText;

public void Result()

{

string s = TimeIndicator.Text;

}

public Mine()

{

InitializeComponent();

gameStarted = false;

this.nrMines = 15;

mineText = new Color[] {Colors.White ,

Colors.Blue, Colors.DarkGreen, Colors.Red, Colors.DarkBlue,

Colors.DarkViolet, Colors.DarkCyan, Colors.Brown, Colors.Black };

GameSetup();

}

private void MenuItem\_Click\_New(object sender, RoutedEventArgs e)

{

GameSetup();

}

private void GameSetup()

{

Mines = new MinesGrid(10, 10, nrMines);

foreach (Button btn in ButtonsGrid.Children)

{

btn.Content = "";

btn.IsEnabled = true;

}

Mines.CounterChanged += OnCounterChanged;

MinesIndicator.Text = nrMines.ToString();

Mines.ClickPlate += OnClickPlate;

Mines.TimerThresholdReached += OnTimeChanged;

TimeIndicator.Text = "0";

Mines.Run();

gameStarted = true;

}

private void Button\_Click(object sender, RoutedEventArgs e)

{

Button btn = (Button)sender;

int row = ParseButtonRow(btn);

int col = ParseButtonColumn(btn);

if (!Mines.IsInGrid(row, col)) throw new MinesweeperException("Invalid Button to MinesGrid reference on reveal");

if (Mines.IsFlagged(row, col)) return;

btn.IsEnabled = false;

if (Mines.IsBomb(row, col))

{

StackPanel sp = new StackPanel();

sp.Orientation = Orientation.Horizontal;

Image bombImage = new Image();

BitmapImage bi = new BitmapImage();

bi.BeginInit();

bi.UriSource = new Uri(@"Bomb.png", UriKind.Relative);

bi.EndInit();

bombImage.Source = bi;

sp.Children.Add(bombImage);

btn.Content = sp;

Mines.Stop();

if (gameStarted)

{

gameStarted = false;

foreach (Button butn in ButtonsGrid.Children)

{

if (butn.IsEnabled) this.Button\_Click(butn, e);

}

}

}

else

{

int count = Mines.RevealPlate(row, col);

if (count > 0)

{

btn.Foreground = new SolidColorBrush(mineText[count]);

btn.FontWeight = FontWeights.Bold;

btn.Content = count.ToString();

}

}

}

private void Right\_Button\_Click(object sender, MouseButtonEventArgs e)

{

Button btn = (Button)sender;

int row = ParseButtonRow(btn);

int col = ParseButtonColumn(btn);

if (!Mines.IsInGrid(row, col)) throw new MinesweeperException("Invalid Button to MinesGrid reference on flag");

if (Mines.IsFlagged(row, col))

{

btn.Content = "";

}

else

{

StackPanel sp = new StackPanel();

sp.Orientation = Orientation.Horizontal;

Image flagImage = new Image();

BitmapImage bi = new BitmapImage();

bi.BeginInit();

bi.UriSource = new Uri(@"Flag.png", UriKind.Relative);

bi.EndInit();

flagImage.Source = bi;

sp.Children.Add(flagImage);

btn.Content = sp;

}

Mines.FlagMine(row, col);

}

private int ParseButtonRow(Button btn)

{

if (btn.Name.IndexOf("Button") != 0) throw new MinesweeperException("Wrong button name in UI module");

return int.Parse(btn.Name.Substring(6, (btn.Name.Length - 6) / 2));

}

private int ParseButtonColumn(Button btn)

{

if (btn.Name.IndexOf("Button") != 0) throw new MinesweeperException("Wrong button name in UI module");

return int.Parse(btn.Name.Substring(6 + (btn.Name.Length - 6) / 2, (btn.Name.Length - 6) / 2));

}

private void OnCounterChanged(object sender, EventArgs e)

{

MinesIndicator.Text = (this.nrMines - Mines.FlaggedMines).ToString();

}

private void OnTimeChanged(object sender, EventArgs e)

{

TimeIndicator.Text = MinesGrid.TimeElapsed.ToString();

}

private void OnClickPlate(object sender, PlateEventArgs e)

{

string btnName = "Button";

if (Mines.Width <= 10 && Mines.Height <= 10) btnName += String.Format("{0:D1}{1:D1}", e.PlateRow, e.PlateColumn);

else btnName += String.Format("{0:D2}{1:D2}", e.PlateRow, e.PlateColumn);

Button senderButton = (ButtonsGrid.FindName(btnName) as Button);

if (senderButton == null) throw new MinesweeperException("Invalid Button to MinesGrid reference on multiple reveal");

this.Button\_Click(senderButton, new RoutedEventArgs());

}

}

}

**Snake**

**Food.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Shapes;

using System.Windows.Media;

using System.Windows.Controls;

namespace GamingPlatform.Games.Snake

{

class Food

{

public double x, y;

public Ellipse ell = new Ellipse();

public Food(double x, double y)

{

this.x = x;

this.y = y;

}

public void setfoodposition()

{

ell.Width = ell.Height = 10;

ell.Fill = Brushes.Blue;

Canvas.SetLeft(ell, x);

Canvas.SetTop(ell, y);

}

}

}

**Snake.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Controls;

using System.Windows.Media;

using System.Windows.Shapes;

namespace GamingPlatform.Games.Snake

{

class Snake

{

public double x, y;

public Rectangle rec = new Rectangle();

public Snake(double x, double y)

{

this.x = x;

this.y = y;

}

public void setsnakeposition()

{

rec.Width = rec.Height = 10;

rec.Fill = Brushes.Black;

Canvas.SetLeft(rec, x);

Canvas.SetTop(rec, y);

}

}

}

**SnakeCanvas.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Data;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Imaging;

using System.Windows.Shapes;

using GamingPlatform.AllLogic;

using System.Windows.Threading;

namespace GamingPlatform.Games.Snake

{

public partial class SnakeCanvas : Window

{

DispatcherTimer time;

List<Snake> snakebody;

List<Food> food;

Random rd = new Random();

double x = 100;

double y = 100;

int direction = 0;

int left = 4;

int right = 6;

int up = 8;

int down = 2;

int score = 0;

int count = 0;

public SnakeCanvas()

{

InitializeComponent();

time = new DispatcherTimer();

snakebody = new List<Snake>();

food = new List<Food>();

snakebody.Add(new Snake(x, y));

food.Add(new Food(rd.Next(0, 48) \* 10, rd.Next(0, 31) \* 10));

time.Interval = new TimeSpan(0, 0, 0, 0, 100); //скорость змеи

time.Tick += time\_Tick;

}

void addfoodincanvas()

{

food[0].setfoodposition();

mycanvas.Children.Insert(0, food[0].ell);

}

void addsnakeincanvas()

{

foreach (Snake snake in snakebody)

{

snake.setsnakeposition();

mycanvas.Children.Add(snake.rec);

}

}

void time\_Tick(object sender, EventArgs e)

{

if (direction != 0)

{

for (int i = snakebody.Count - 1; i > 0; i--)

{

snakebody[i] = snakebody[i - 1];

}

}

if (direction == up)

y -= 10;

if (direction == down)

y += 10;

if (direction == left)

x -= 10;

if (direction == right)

x += 10;

if (snakebody[0].x == food[0].x && snakebody[0].y == food[0].y)

{

snakebody.Add(new Snake(food[0].x, food[0].y));

food[0] = new Food(rd.Next(0, 48) \* 10, rd.Next(0, 31) \* 10);

mycanvas.Children.RemoveAt(0);

addfoodincanvas();

score++;

txtbScore.Text = "Score : " + score.ToString();

}

snakebody[0] = new Snake(x, y);

if (snakebody[0].x > 480 || snakebody[0].y > 310 || snakebody[0].x < 0 || snakebody[0].y < 0)

{

SnakeRepositoty sr = SnakeRepositoty.Initialize();

sr.WriteResult(score);

score = 0;

this.Close();

}

for (int i = 1; i < snakebody.Count; i++)

{

if (snakebody[0].x == snakebody[i].x && snakebody[0].y == snakebody[i].y)

{

SnakeRepositoty sr = SnakeRepositoty.Initialize();

sr.WriteResult(score);

score = 0;

this.Close();

}

}

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

{

if (mycanvas.Children[i] is Rectangle)

count++;

}

mycanvas.Children.RemoveRange(1, count);

count = 0;

addsnakeincanvas();

}

private void Window\_KeyDown(object sender, KeyEventArgs e)

{

if (e.Key == Key.Up && direction != down)

direction = up;

if (e.Key == Key.Down && direction != up)

direction = down;

if (e.Key == Key.Left && direction != right)

direction = left;

if (e.Key == Key.Right && direction != left)

direction = right;

}

private void Window\_Loaded(object sender, RoutedEventArgs e)

{

addsnakeincanvas();

addfoodincanvas();

time.Start();

}

}

}

**SwitchGame.cs**

using System;

using System.Collections.ObjectModel;

using System.Diagnostics;

using System.Linq;

using System.Windows;

using System.Windows.Input;

using System.Threading.Tasks;

using GamingPlatform.AllLogic;

namespace Puzzle

{

public partial class Switch : Window

{

public int schet = 0;//Score for KOSTY

public string schet\_str;

const int gridSize = 4;

public ObservableCollection<int> Cells { get; private set; } = new ObservableCollection<int>();

public Switch()

{

InitializeComponent();

DataContext = Cells;

Fill();

}

async void CommandBinding\_Executed(object sender, ExecutedRoutedEventArgs e)

{

int val = (int)e.Parameter;

DoAction(val, (canMove, zero, cur) => {

Debug.WriteLine($"Diff = {zero - cur}; ({cur} + 1) % 4 = {(cur + 1) % 4}; curIndex - zero = {cur - zero}");

if (canMove)

{

Cells[zero] = val;

Cells[cur] = 0;

schet = schet + 1;

}

});

if (IsCorrect())

{

MessageBox.Show("Congratulation, you win!", "Puzzle", MessageBoxButton.OK, MessageBoxImage.Information);

SwitchRepository sr = SwitchRepository.Initialize();

await Task.Run(() => { sr.WriteResult(schet); });

schet\_str = Convert.ToString(schet); //Score for KOSTY

MessageBox.Show(schet\_str);

}

}

void CommandBinding\_CanExecute(object sender, CanExecuteRoutedEventArgs e)

{

DoAction((int)e.Parameter, (canMove, zero, cur) => e.CanExecute = canMove);

}

void Fill()

{

var random = new Random();

var size = gridSize \* gridSize;

var list = Enumerable.Range(0, size).ToList();

while (list.Count > 0)

{

int index = random.Next(0, list.Count - 1);

Cells.Add(list[index]);

list.RemoveAt(index);

}

}

public bool IsCorrect()

{

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

if (Cells[i] != i)

return false;

return true;

}

public void DoAction(int current, Action<bool, int, int> action)

{

var zeroIndex = 0;

var curIndex = 0;

var canMove = false;

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

{

if (Cells[i] == 0)

zeroIndex = i;

if (Cells[i] == current)

curIndex = i;

}

var diff = curIndex - zeroIndex;

var column = (curIndex + 1) % gridSize;

if (diff == gridSize) canMove = true;

else if (diff == -gridSize) canMove = true;

else if (diff == 1) canMove = column != 1;

else if (diff == -1) canMove = column != 0;

action(canMove, zeroIndex, curIndex);

}

}

}

**Tetris.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Data;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Imaging;

using System.Windows.Navigation;

using System.Windows.Shapes;

using System.Windows.Threading;

using GamingPlatform.AllLogic;

namespace Tetris

{

/// <summary>

/// Interaction logic for MainWindow.xaml

/// </summary>

///

public partial class Tetris : Window

{

private const int GAMESPEED = 700;

List<System.Media.SoundPlayer> soundList = new List<System.Media.SoundPlayer>();

DispatcherTimer timer;

Random shapeRandom;

private int score = 0;

private int rowCount = 0;

private int columnCount = 0;

private int leftPos = 0;

private int downPos = 0;

private int currentTetrominoWidth;

private int currentTetrominoHeigth;

private int currentShapeNumber;

private int nextShapeNumber;

private int tetrisGridColumn;

private int tetrisGridRow;

private int rotation = 0;

private bool gameActive = false;

private bool nextShapeDrawed = false;

private int[,] currentTetromino = null;

private bool isRotated = false;

private bool bottomCollided = false;

private bool leftCollided = false;

private bool rightCollided = false;

private bool isGameOver = false;

private int gameSpeed;

private int levelScale = 60;

private double gameSpeedCounter = 0;

private int gameLevel = 1;

private int gameScore = 0;

private static Color O\_TetrominoColor = Colors.GreenYellow;

private static Color I\_TetrominoColor = Colors.Red;

private static Color T\_TetrominoColor = Colors.Gold;

private static Color S\_TetrominoColor = Colors.Violet;

private static Color Z\_TetrominoColor = Colors.DeepSkyBlue;

private static Color J\_TetrominoColor = Colors.Cyan;

private static Color L\_TetrominoColor = Colors.LightSeaGreen;

List<int> currentTetrominoRow = null;

List<int> currentTetrominoColumn = null;

Color[] shapeColor = { O\_TetrominoColor,I\_TetrominoColor,

T\_TetrominoColor,S\_TetrominoColor,

Z\_TetrominoColor,J\_TetrominoColor,

L\_TetrominoColor

};

// ---------

string[] arrayTetrominos = { "","O\_Tetromino" , "I\_Tetromino\_0",

"T\_Tetromino\_0","S\_Tetromino\_0",

"Z\_Tetromino\_0","J\_Tetromino\_0",

"L\_Tetromino\_0"

};

#region Array of tetrominos shape

// arrays of tetromino shape

//---- O Tetromino------------

public int[,] O\_Tetromino = new int[2, 2] { { 1, 1 }, // \* \*

{ 1, 1 }}; // \* \*

//---- I Tetromino------------

public int[,] I\_Tetromino\_0 = new int[2, 4] { { 1, 1, 1, 1 }, { 0, 0, 0, 0 } };// \* \* \* \*

public int[,] I\_Tetromino\_90 = new int[4, 2] {{ 1,0 }, // \*

{ 1,0 }, // \*

{ 1,0 }, // \*

{ 1,0 }}; // \*

//---- T Tetromino------------

public int[,] T\_Tetromino\_0 = new int[2, 3] {{0,1,0}, // \*

{1,1,1}}; // \* \* \*

public int[,] T\_Tetromino\_90 = new int[3, 2] {{1,0}, // \*

{1,1}, // \* \*

{1,0}}; // \*

public int[,] T\_Tetromino\_180 = new int[2, 3] {{1,1,1}, // \* \* \*

{0,1,0}}; // \*

public int[,] T\_Tetromino\_270 = new int[3, 2] {{0,1}, // \*

{1,1}, // \* \*

{0,1}}; // \*

//---- S Tetromino------------

public int[,] S\_Tetromino\_0 = new int[2, 3] {{0,1,1}, // \* \*

{1,1,0}}; // \* \*

public int[,] S\_Tetromino\_90 = new int[3, 2] {{1,0}, // \*

{1,1}, // \* \*

{0,1}}; // \*

//---- Z Tetromino------------

public int[,] Z\_Tetromino\_0 = new int[2, 3] {{1,1,0}, // \* \*

{0,1,1}}; // \* \*

public int[,] Z\_Tetromino\_90 = new int[3, 2] {{0,1}, // \*

{1,1}, // \* \*

{1,0}}; // \*

//---- J Tetromino------------

public int[,] J\_Tetromino\_0 = new int[2, 3] {{1,0,0}, // \*

{1,1,1}}; // \* \* \*

public int[,] J\_Tetromino\_90 = new int[3, 2] {{1,1}, // \* \*

{1,0}, // \*

{1,0}}; // \*

public int[,] J\_Tetromino\_180 = new int[2, 3] {{1,1,1}, // \* \* \*

{0,0,1}}; // \*

public int[,] J\_Tetromino\_270 = new int[3, 2] {{0,1}, // \*

{0,1}, // \*

{1,1 }}; // \* \*

//---- L Tetromino------------

public int[,] L\_Tetromino\_0 = new int[2, 3] {{0,0,1}, // \*

{1,1,1}}; // \* \* \*

public int[,] L\_Tetromino\_90 = new int[3, 2] {{1,0}, // \*

{1,0}, // \*

{1,1}}; // \* \*

public int[,] L\_Tetromino\_180 = new int[2, 3] {{1,1,1}, // \* \* \*

{1,0,0}}; // \*

public int[,] L\_Tetromino\_270 = new int[3, 2] {{1,1}, // \* \*

{0,1}, // \*

{0,1 }}; // \*

public object Task { get; private set; }

#endregion

public Tetris()

{

InitializeComponent();

gameSpeed = GAMESPEED;

KeyDown += MainWindow\_KeyDown;

timer = new DispatcherTimer();

timer.Interval = new TimeSpan(0, 0, 0, 0, gameSpeed);

timer.Tick += Timer\_Tick;

tetrisGridColumn = tetrisGrid.ColumnDefinitions.Count;

tetrisGridRow = tetrisGrid.RowDefinitions.Count;

shapeRandom = new Random();

currentShapeNumber = shapeRandom.Next(1, 8);

nextShapeNumber = shapeRandom.Next(1, 8);

nextTxt.Visibility = levelTxt.Visibility = GameOverTxt.Visibility = Visibility.Collapsed;

}

private void MainWindow\_KeyDown(object sender, KeyEventArgs e)

{

if (!timer.IsEnabled) { return; }

switch (e.Key.ToString())

{

case "Up":

rotation += 90;

if (rotation > 270) { rotation = 0; }

shapeRotation(rotation);

break;

case "Down":

downPos++;

break;

case "Right":

TetroCollided();

if (!rightCollided) { leftPos++; }

rightCollided = false;

break;

case "Left":

TetroCollided();

if (!leftCollided) { leftPos--; }

leftCollided = false;

break;

}

moveShape();

}

private void shapeRotation(int \_rotation)

{

if (rotationCollided(rotation))

{

rotation -= 90;

return;

}

if (arrayTetrominos[currentShapeNumber].IndexOf("I\_") == 0)

{

if (\_rotation > 90) { \_rotation = rotation = 0; }

currentTetromino = getVariableByString("I\_Tetromino\_" + \_rotation);

}

else if (arrayTetrominos[currentShapeNumber].IndexOf("T\_") == 0)

{

currentTetromino = getVariableByString("T\_Tetromino\_" + \_rotation);

}

else if (arrayTetrominos[currentShapeNumber].IndexOf("S\_") == 0)

{

if (\_rotation > 90) { \_rotation = rotation = 0; }

currentTetromino = getVariableByString("S\_Tetromino\_" + \_rotation);

}

else if (arrayTetrominos[currentShapeNumber].IndexOf("Z\_") == 0)

{

if (\_rotation > 90) { \_rotation = rotation = 0; }

currentTetromino = getVariableByString("Z\_Tetromino\_" + \_rotation);

}

else if (arrayTetrominos[currentShapeNumber].IndexOf("J\_") == 0)

{

currentTetromino = getVariableByString("J\_Tetromino\_" + \_rotation);

}

else if (arrayTetrominos[currentShapeNumber].IndexOf("L\_") == 0)

{

currentTetromino = getVariableByString("L\_Tetromino\_" + \_rotation);

}

else if (arrayTetrominos[currentShapeNumber].IndexOf("O\_") == 0)

{

return;

}

isRotated = true;

addShape(currentShapeNumber, leftPos, downPos);

}

private void Timer\_Tick(object sender, EventArgs e)

{

downPos++;

moveShape();

if (gameSpeedCounter >= levelScale)

{

if (gameSpeed >= 50)

{

gameSpeed -= 50;

gameLevel++;

levelTxt.Text = "Level: " + gameLevel.ToString();

}

else { gameSpeed = 50; }

timer.Stop();

timer.Interval = new TimeSpan(0, 0, 0, 0, gameSpeed);

timer.Start();

gameSpeedCounter = 0;

}

gameSpeedCounter += (gameSpeed / 1000f);

}

private void Button\_Click\_1(object sender, RoutedEventArgs e)

{

if (isGameOver)

{

tetrisGrid.Children.Clear();

nextShapeCanvas.Children.Clear();

GameOverTxt.Visibility = Visibility.Collapsed;

isGameOver = false;

}

if (!timer.IsEnabled)

{

if (!gameActive) { scoreTxt.Text = "0"; leftPos = 3; addShape(currentShapeNumber, leftPos); }

nextTxt.Visibility = levelTxt.Visibility = Visibility.Visible;

levelTxt.Text = "Level: " + gameLevel.ToString();

timer.Start();

startStopBtn.Content = "Stop Game";

gameActive = true;

}

else

{

timer.Stop();

startStopBtn.Content = "Start Game";

}

}

private void addShape(int shapeNumber, int \_left = 0, int \_down = 0)

{

removeShape();

currentTetrominoRow = new List<int>();

currentTetrominoColumn = new List<int>();

Rectangle square = null;

if (!isRotated)

{

currentTetromino = null;

currentTetromino = getVariableByString(arrayTetrominos[shapeNumber].ToString());

}

int firstDim = currentTetromino.GetLength(0);

int secondDim = currentTetromino.GetLength(1);

currentTetrominoWidth = secondDim;

currentTetrominoHeigth = firstDim;

if (currentTetromino == I\_Tetromino\_90)

{

currentTetrominoWidth = 1;

}

else if (currentTetromino == I\_Tetromino\_0) { currentTetrominoHeigth = 1; }

for (int row = 0; row < firstDim; row++)

{

for (int column = 0; column < secondDim; column++)

{

int bit = currentTetromino[row, column];

if (bit == 1)

{

square = getBasicSquare(shapeColor[shapeNumber - 1]);

tetrisGrid.Children.Add(square);

square.Name = "moving\_" + Grid.GetRow(square) + "\_" + Grid.GetColumn(square);

if (\_down >= tetrisGrid.RowDefinitions.Count - currentTetrominoHeigth)

{

\_down = tetrisGrid.RowDefinitions.Count - currentTetrominoHeigth;

}

Grid.SetRow(square, rowCount + \_down);

Grid.SetColumn(square, columnCount + \_left);

currentTetrominoRow.Add(rowCount + \_down);

currentTetrominoColumn.Add(columnCount + \_left);

}

columnCount++;

}

columnCount = 0;

rowCount++;

}

columnCount = 0;

rowCount = 0;

if (!nextShapeDrawed)

{

drawNextShape(nextShapeNumber);

}

}

private void moveShape()

{

leftCollided = false;

rightCollided = false;

TetroCollided();

if (leftPos > (tetrisGridColumn - currentTetrominoWidth))

{

leftPos = (tetrisGridColumn - currentTetrominoWidth);

}

else if (leftPos < 0) { leftPos = 0; }

if (bottomCollided)

{

shapeStoped();

return;

}

addShape(currentShapeNumber, leftPos, downPos);

}

private bool rotationCollided(int \_rotation)

{

if (checkCollided(0, currentTetrominoWidth - 1)) { return true; }

else if (checkCollided(0, -(currentTetrominoWidth - 1))) { return true; }

else if (checkCollided(0, -1)) { return true; }

else if (checkCollided(-1, currentTetrominoWidth - 1)) { return true; }

else if (checkCollided(1, currentTetrominoWidth - 1)) { return true; }

return false;

}

private void TetroCollided()

{

bottomCollided = checkCollided(0, 1);

leftCollided = checkCollided(-1, 0);

rightCollided = checkCollided(1, 0);

}

private bool checkCollided(int \_leftRightOffset, int \_bottomOffset)

{

Rectangle movingSquare;

int squareRow = 0;

int squareColumn = 0;

for (int i = 0; i <= 3; i++)

{

squareRow = currentTetrominoRow[i];

squareColumn = currentTetrominoColumn[i];

try

{

movingSquare = (Rectangle)tetrisGrid.Children

.Cast<UIElement>()

.FirstOrDefault(e => Grid.GetRow(e) == squareRow + \_bottomOffset && Grid.GetColumn(e) == squareColumn + \_leftRightOffset);

if (movingSquare != null)

{

if (movingSquare.Name.IndexOf("arrived") == 0)

{

return true;

}

}

}

catch { }

}

if (downPos > (tetrisGridRow - currentTetrominoHeigth)) { return true; }

return false;

}

private void drawNextShape(int shapeNumber)

{

nextShapeCanvas.Children.Clear();

int[,] nextShapeTetromino = null;

nextShapeTetromino = getVariableByString(arrayTetrominos[shapeNumber]);

int firstDim = nextShapeTetromino.GetLength(0);

int secondDim = nextShapeTetromino.GetLength(1);

int x = 0;

int y = 0;

Rectangle square;

for (int row = 0; row < firstDim; row++)

{

for (int column = 0; column < secondDim; column++)

{

int bit = nextShapeTetromino[row, column];

if (bit == 1)

{

square = getBasicSquare(shapeColor[shapeNumber - 1]);

nextShapeCanvas.Children.Add(square);

Canvas.SetLeft(square, x);

Canvas.SetTop(square, y);

}

x += 25;

}

x = 0;

y += 25;

}

nextShapeDrawed = true;

}

private void shapeStoped()

{

timer.Stop();

if (downPos <= 2)

{

gameOver();

return;

}

int index = 0;

while (index < tetrisGrid.Children.Count)

{

UIElement element = tetrisGrid.Children[index];

if (element is Rectangle)

{

Rectangle square = (Rectangle)element;

if (square.Name.IndexOf("moving\_") == 0)

{

string newName = square.Name.Replace("moving\_", "arrived\_");

square.Name = newName;

}

}

index++;

}

checkComplete();

reset();

timer.Start();

}

private void checkComplete()

{

int gridRow = tetrisGrid.RowDefinitions.Count;

int gridColumn = tetrisGrid.ColumnDefinitions.Count;

int squareCount = 0;

for (int row = gridRow; row >= 0; row--)

{

squareCount = 0;

for (int column = gridColumn; column >= 0; column--)

{

Rectangle square;

square = (Rectangle)tetrisGrid.Children

.Cast<UIElement>()

.FirstOrDefault(e => Grid.GetRow(e) == row && Grid.GetColumn(e) == column);

if (square != null)

{

if (square.Name.IndexOf("arrived") == 0)

{

squareCount++;

}

}

}

if (squareCount == gridColumn)

{

deleteLine(row);

scoreTxt.Text = getScore().ToString();

checkComplete();

}

}

}

private void deleteLine(int row)

{

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

{

Rectangle square;

try

{

square = (Rectangle)tetrisGrid.Children

.Cast<UIElement>()

.FirstOrDefault(e => Grid.GetRow(e) == row && Grid.GetColumn(e) == i);

tetrisGrid.Children.Remove(square);

}

catch { }

}

foreach (UIElement element in tetrisGrid.Children)

{

Rectangle square = (Rectangle)element;

if (square.Name.IndexOf("arrived") == 0 && Grid.GetRow(square) <= row)

{

Grid.SetRow(square, Grid.GetRow(square) + 1);

}

}

}

private int getScore()

{

gameScore += 50 \* gameLevel;

return gameScore;

}

private void reset()

{

downPos = 0;

leftPos = 3;

isRotated = false;

rotation = 0;

currentShapeNumber = nextShapeNumber;

if (!isGameOver) { addShape(currentShapeNumber, leftPos); }

nextShapeDrawed = false;

shapeRandom = new Random();

nextShapeNumber = shapeRandom.Next(1, 8);

bottomCollided = false;

leftCollided = false;

rightCollided = false;

}

async private void gameOver()

{

score = gameScore; //Score for KOSTY

TetrisRepository tr = new TetrisRepository();

await System.Threading.Tasks.Task.Run(() => { tr.WriteResult(gameScore); });

isGameOver = true;

reset();

startStopBtn.Content = "Start Game";

GameOverTxt.Visibility = Visibility.Visible;

rowCount = 0;

columnCount = 0;

leftPos = 0;

gameSpeedCounter = 0;

gameSpeed = GAMESPEED;

gameLevel = 1;

gameActive = false;

gameScore = 0;

nextShapeDrawed = false;

currentTetromino = null;

currentShapeNumber = shapeRandom.Next(1, 8);

nextShapeNumber = shapeRandom.Next(1, 8);

timer.Interval = new TimeSpan(0, 0, 0, 0, gameSpeed);

}

private void removeShape()

{

int index = 0;

while (index < tetrisGrid.Children.Count)

{

UIElement element = tetrisGrid.Children[index];

if (element is Rectangle)

{

Rectangle square = (Rectangle)element;

if (square.Name.IndexOf("moving\_") == 0)

{

tetrisGrid.Children.Remove(element);

index = -1;

}

}

index++;

}

}

private Rectangle getBasicSquare(Color rectColor)

{

Rectangle rectangle = new Rectangle();

rectangle.Width = 25;

rectangle.Height = 25;

rectangle.StrokeThickness = 1;

rectangle.Stroke = Brushes.White;

rectangle.Fill = getGradientColor(rectColor);

return rectangle;

}

private LinearGradientBrush getGradientColor(Color clr)

{

LinearGradientBrush gradientColor = new LinearGradientBrush();

gradientColor.StartPoint = new Point(0, 0);

gradientColor.EndPoint = new Point(1, 1.5);

GradientStop black = new GradientStop();

black.Color = Colors.Black;

black.Offset = -1.5;

gradientColor.GradientStops.Add(black);

GradientStop other = new GradientStop();

other.Color = clr;

other.Offset = 0.70;

gradientColor.GradientStops.Add(other);

return gradientColor;

}

private int[,] getVariableByString(string variable)

{

return (int[,])this.GetType().GetField(variable).GetValue(this);

}

}

}

**TicTacWindow.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Data;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Imaging;

using System.Windows.Shapes;

using System.Threading;

using System.ComponentModel;

using System.Runtime.CompilerServices;

using GamingPlatform.AllLogic;

using System.Data.Entity.Validation;

namespace GamingPlatform.Games.TicTacToe

{

public class Position : INotifyPropertyChanged

{

public int CheckWinLose()

{

char lit = 'n';

for (int i = 1; i <= 3; i++)

{

if ((FEN[i] == FEN[i + 3]) && (FEN[i] == FEN[i + 6]))

{

lit = FEN[i];

}

}

for (int i = 1; i <= 9; i = i + 3)

{

if ((FEN[i] == FEN[i + 1]) && (FEN[i] == FEN[i + 2]))

{

lit = FEN[i];

}

}

if ((FEN[1] == FEN[5]) && (FEN[1] == FEN[9]))

{

lit = fen[1];

}

if ((FEN[3] == FEN[5]) && (FEN[5] == FEN[7]))

{

lit = FEN[3];

}

if (lit == '1')

{

return 1;

}

else if (lit == '2')

{

return 2;

}

return 0;

}

public void Draw()

{

if (fen[1] == '1') { P1 = "X"; }

else if (fen[1] == '2') { P1 = "0"; }

if (fen[2] == '1') { P2 = "X"; }

else if (fen[2] == '2') { P2 = "0"; }

if (fen[3] == '1') { P3 = "X"; }

else if (fen[3] == '2') { P3 = "0"; }

if (fen[4] == '1') { P4 = "X"; }

else if (fen[4] == '2') { P4 = "0"; }

if (fen[5] == '1') { P5 = "X"; }

else if (fen[5] == '2') { P5 = "0"; }

if (fen[6] == '1') { P6 = "X"; }

else if (fen[6] == '2') { P6 = "0"; }

if (fen[7] == '1') { P7 = "X"; }

else if (fen[7] == '2') { P7 = "0"; }

if (fen[8] == '1') { P8 = "X"; }

else if (fen[8] == '2') { P8 = "0"; }

if (fen[9] == '1') { P9 = "X"; }

else if (fen[9] == '2') { P9 = "0"; }

}

public bool yourMove;

public string fen;

public decimal price;

public int idTable;

public string user1;

public string user2;

public string p1 = "";

public string p2 = "";

public string p3 = "";

public string p4 = "";

public string p5 = "";

public string p6 = "";

public string p7 = "";

public string p8 = "";

public string p9 = "";

public string FEN

{

get { return fen; }

set

{

fen = value; OnPropertyChanged("FEN");

Draw();

}

}

public int IDTable { get { return idTable; } set { idTable = value; OnPropertyChanged("IDTable"); } }

public string User1 { get { return user1; } set { user1 = value; OnPropertyChanged("User1"); } }

public string User2 { get { return user2; } set { user2 = value; OnPropertyChanged("User2"); } }

public string P1 { get { return p1; } set { p1 = value; OnPropertyChanged("P1"); } }

public string P2 { get { return p2; } set { p2 = value; OnPropertyChanged("P2"); } }

public string P3 { get { return p3; } set { p3 = value; OnPropertyChanged("P3"); } }

public string P4 { get { return p4; } set { p4 = value; OnPropertyChanged("P4"); } }

public string P5 { get { return p5; } set { p5 = value; OnPropertyChanged("P5"); } }

public string P6 { get { return p6; } set { p6 = value; OnPropertyChanged("P6"); } }

public string P7 { get { return p7; } set { p7 = value; OnPropertyChanged("P7"); } }

public string P8 { get { return p8; } set { p8 = value; OnPropertyChanged("P8"); } }

public string P9 { get { return p9; } set { p9 = value; OnPropertyChanged("P9"); } }

public event PropertyChangedEventHandler PropertyChanged;

public void OnPropertyChanged([CallerMemberName]string prop = "")

{

if (PropertyChanged != null)

PropertyChanged(this, new PropertyChangedEventArgs(prop));

}

}

public partial class TicTacWindow : Window

{

public static TicTacWindow win { get; set; }

public Position position { get; set; }

public TicTacWindow()

{

win = this;

InitializeComponent();

position = new Position

{

IDTable = 0,

};

win.DataContext = position;

}

public void Play()

{

while (true)

{

using (DBGamingPlatformEntities d = new DBGamingPlatformEntities())

{

List<TicTacMatch> list = d.TicTacMatches.ToList();

List<Player> listP = d.Players.ToList();

TicTacMatch result = new TicTacMatch();

Player player = new Player();

player = listP.Find(item => item.IDPlayer == ActiveUser.UserId);

if (position.IDTable==0)

{

result = list.Find(item => item.StatusMatch == 0); //0-wait, 1-play 2-

position.IDTable = result.IDTable;

if (result.Player.IDPlayer == 15) //left user

{

position.yourMove = true;

result.Player = player;

position.User1 = result.Player.LoginUser;

}

else if(result.Player1.IDPlayer == 15)

{

position.yourMove = false;

position.User2 = result.Player1.LoginUser;

result.Player1 = player;

result.StatusMatch = 1;

}

}

else

{

result = list.Find(item => item.IDTable == position.IDTable);

position.FEN = result.FEN;

position.User1 = result.Player.LoginUser;

position.User2 = result.Player1.LoginUser;

if (position.CheckWinLose() == 1)

{

MessageBox.Show(position.User1 + " победил");

break;

}

else if (position.CheckWinLose() == 2)

{

MessageBox.Show(position.User2 + " победил");

break;

}

}

d.SaveChanges();

}

}

}

private async void Button\_Click\_1(object sender, RoutedEventArgs e)

{

await Task.Run(() => { Play(); });

}

public void WriteInDB(int numberButton)

{

using (DBGamingPlatformEntities writeMove = new DBGamingPlatformEntities())

{

List<TicTacMatch> list = writeMove.TicTacMatches.ToList();

TicTacMatch result = new TicTacMatch();

result = list.Find(item => item.IDTable == position.IDTable);

try

{

writeMove.SaveChanges();

}

catch (DbEntityValidationException d)

{

MessageBox.Show(d.Message);

}

char[] str = result.FEN.ToCharArray();

StringBuilder stringBuilder = new StringBuilder(result.FEN);

if (stringBuilder[0] == 'K')

{

stringBuilder[0] = 'N';

stringBuilder[numberButton] = '1';

}

else if (stringBuilder[0] == 'N')

{

stringBuilder[0] = 'K';

stringBuilder[numberButton] = '2';

}

result.FEN = stringBuilder.ToString();

writeMove.SaveChanges();

}

}

private void b0\_Click(object sender, RoutedEventArgs e)

{

if ((position.yourMove == true && position.FEN[0]=='K')||(position.yourMove == false && position.FEN[0] == 'N'))

{

WriteInDB(Convert.ToInt32(Convert.ToString((sender as Button).Name[1])));

}

}

}

# **ПРИЛОЖЕНИЕ Б**

Createdatabase GamingPlatformDB\_v6   
go   
  
create table Player(IDPlayer int not null identity primary key,   
LoginUser varchar(50) null,   
NameUser varchar(50) null,   
RegistrDate varchar(50) null,   
SaltUser varchar(50) null,   
HashUser varchar(50) null)   
  
go   
  
create table SnakeGame(   
ID int primary key identity not null,   
IDUser int foreign key references Player(IDPlayer),   
BestScore int)   
  
go   
  
create table MemoryCardsGame(   
ID int primary key identity not null,   
IDUser int foreign key references Player(IDPlayer),   
BestScore int)   
  
go   
  
create table GameTetris(   
ID int primary key identity not null,   
IDUser int foreign key references Player(IDPlayer),   
BestScore int)   
  
go   
  
create table GameSwitch(   
ID int primary key identity not null,   
IDUser int foreign key references Player(IDPlayer),   
BestScore int)   
  
go   
  
create table GameMine(   
ID int primary key identity not null,   
IDUser int foreign key references Player(IDPlayer),   
BestScore int)   
  
go   
  
create table GameBall(   
ID int primary key identity not null,   
IDUser int foreign key references Player(IDPlayer),   
BestScore int)   
  
go   
  
create table TicTacMatch(IDTable int primary key identity not null,   
IDUser1 int foreign key references Player(IDPlayer),   
IDUser2 int foreign key references Player(IDPlayer),   
StatusMatch int,   
FEN varchar(50))