Содержание

Введение

Актуальность исследования и разработки

Современность не перестаёт удивлять разнообразием компьютерных игр. Особенно популярны логические игры, где нужно подумать головой, чтобы решить что-то, ведь они не только увлекательны, но и полезны для мыслительных процессов. Каждый человек обладает мышлением, у кого-то оно хорошо развито, у других, наоборот, слабое. Одной из таких компьютерных игр является «Пятнашки». [#https://www.i-igrushki.ru/igrushkapedia/pyatnashki.html]

Игра в «Пятнашки» – это отличный способ развить логическое мышление, так как большинство задач, которые встают перед человеком в реальной жизни, требуют разрешения именно посредством логики, умением быстро принимать правильное решение и думать наперед. Кто-нибудь видел решающиеся сами по себе «Пятнашки»? Явно, нет. Это что-то новое и неизведанное, то, что не придётся решать самому, а всё за Вас сделает компьютер, если вы не сможете решить сами! Это как помощник в игре, который всегда протянет виртуальную руку помощи. Особенно это полезно для детей, которые только учатся логическому мышлению, так как они не всегда могут найти решения этой головоломки самостоятельно.

Цель представленной работы – разработка решателя для головоломки «Пятнашки».

Для достижения поставленной цели необходимо решить следующие *задачи*:

1. Спроектировать и разработать базу данных.
2. Разработать клиент-серверное приложение базы данных.
3. Разработать средства администрирования базы данных.
4. Разработать средства защиты базы данных.
5. Определить порядок работы с продуктом.

Предметом исследования являются существующие компьютерные игры Пятнашки и их влияние на развитие логического мышления с импользованием или без искусственного интелекта, позволяющего предложить готовое решение.

Объектом исследования является – технологии, позволяющие разработать головоломку с использованием средств ИИ, СУБД PostgreSQL и системы программирования C#/WPF на платформе .NET Framework.

1 Общая часть

1.1 Постановка задачи

Искусственный интеллект связан с процессом разработки неживого рационального агента. Рациональный агент – это сущность, способная воспринимать окружение и, исходя из полученных представлений, действовать рационально. Под рациональностью здесь понимается принятие агентом оптимальных решений (объект, действующий оптимальным для достижения наилучшего ожидаемого результата образом) для достижения наилучшего ожидаемого результата.

Люди – лучший пример живых рациональных агентов, постоянно получающих информацию из своего окружения и реагирующих на неё принятием решений, которые обычно идут на пользу.

Человеческий интеллект остаётся самым способным, сложным, совершенным в известной нам части Вселенной. А если это так и в ближайшем будущем вряд ли что-либо сравнится с ним по мощи, которой он сегодня обладает, то зачем создавать ИИ.

В некоторых специфических условиях реакция на поступающую из окружения информацию у ИИ лучше, чем у людей. ИИ способен за короткое время вычислять миллионы инструкций и выдавать осуществимое, а иногда оптимальное решение очень сложных задач.

Поэтому, хотя у людей в плане мышления интеллект более развит, в вычислениях они компьютерам уступают и именно в вычислительной среде создают ИИ, способный ускорить процесс получения максимального результата. Люди не способны мгновенно и правильно выполнять сложные вычисления, а калькулятор облегчает им жизнь.

Можно привести другой пример ИИ с использованием рационального агента программы на C# для решения головоломки «Пятнашки». Рациональность агента здесь обеспечивается алгоритмом поиска A* (поиск по первому наилучшему совпадению на графе), который связан с эвристикой, направляющей поиск к максимальному результату.

Цель головоломки — переставить клетки из начального состояния так, чтобы все они в итоге располагались в соответствии с конфигурацией целевого. Целевое состояние достигается за несколько ходов. Каждый ход — это замена пустой клетки на соседнюю. Чтобы достичь цели, необходима последовательность движений; один ход — это замена пустой плитки на другую плитку. [#https://habr.com/ru/companies/skillfactory/articles/655629/]

В рамках представленной курсовой работы требуется реализовать следующие функции:

1. Получить данные о последней игре
2. Получить статистику игр пользователя

1.2 Выбор используемой технологий решения задачи и его обоснование

«Игра в 15» или же «Пятнашки» изначально являлась развивающей игрой, она не относится к бесполезным играм по типу фэнтези, RPG, симуляторов и т.д. Именно поэтому игра пользуется спросом у людей разных возрастов.

Предложенные «Пятнашки» реализуют все функции, которые потребуются для максимально комфортной игры. Здесь есть и авторизация, регистрация пользователя в системе, и просмотр статистики последних игр, и самое главное – пятнашки, которые решаются сами по себе, или же Вами, в зависимости от Ваших способностей.

Современные аналоги пятнашек используют обычные алгоритмы решения, при которых пользователь должен только сам решать головоломку. Однако предлагаемая версия пятнашек отличается от других тем, что игрок может, как сам попробовать решить головоломку, так и воспользоваться помощью программы, при затруднении. Игра сама сможет найти самое краткое решение головоломки, на

основе искусственного интеллекта, используемого в проекте, так же предоставить пользователю возможность изучить предлагаемое системой решение, в котором игра выполняется без его участия. Такая возможность способствует развитию логического мышления для заинтересованных в его приобретении игроков.

Стократную пользу имеют те решения, до которых вы сами догадались (без подсказок). Но даже если нет, а просто самостоятельно потели над задачей полдня, а потом подсмотрели решение, то это тоже хорошо. Просмотр готового решения помогает сосредоточиться на недостатках, которые были незаметны при самостоятельном решении. Игрок может учиться у «наставника» пятнашек – искусственного интеллекта в приложении.

Предложенные «Пятнашки» реализуют все функции, которые потребуются для максимально комфортной игры. Здесь есть и авторизация, регистрация пользователя в системе, и просмотр статистики последних игр, и самое главное – пятнашки, которые решаются сами по себе, или же Вами, в зависимости от Ваших способностей.

Алгоритм пятнашек, которые решались бы сами по себе разрабатывался самостоятельно, черпая крупицы информации из Интернета. Полного, собранного алгоритма, который представлен в этой игре, со всем функционалом, нигде кроме этого приложения не найти. Именно это и выделяет эту игру от остальных. В реальности есть такая игра, но она не решается сама по себе, собственно это и является самой главной отличительной чертой этого проекта.

2 Специальная часть

2.1 Проектирование и разработка базы данных

Системный анализ предметной области

База данных и предметная область функционально связаны между собой.

Предметная область определяет назначение и функции базы данных.

Для определения функционала будущей базы данных необходимо выполнить системный анализ предметной области.

На основании анализа предметной области задачи были выделены сущности, описанные в таблице 1.

Таблица 1- Сведения о типах сущностей

Имя сущности	Псевдоним	Особенности использования
Пользователь	Users	В игре присутствует регистрация пользователя, чтобы прогресс игры закреплялся за конкретным человеком.
Игра	Game	В игре присутствует запись результата, который выводится в конце игры. Для показа статистики всех игр пользователя потребуется запись всех игр. Каждая игра будет записываться в эту сущность.
Информация	Information	В игре присутствует полезная информация о Пятнашках, которая выводится в главном окне игры по выбору пользователя.

Сущность описывается с помощью данных, именуемых свойствами или атрибутами (attributes) сущности.

Атрибуты являются определениями в высказывании о сущности и обозначаются именами существительными естественного языка.

Сущности вступают в связи друг с другом через свои атрибуты.

Каждая группа атрибутов, описывающих одно реальное проявление сущности, представляет собой экземпляр (instance) сущности.

Первичный ключ – это атрибут (или комбинация атрибутов), который уникально идентифицирует сущность. Первичные ключи облегчают работу с реляционными данными программным образом. Они дают возможность приложениям уникально идентифицировать и манипулировать каждой сущностью (строкой) в базе данных.

На основе анализа предметной области были выделены атрибуты и ключи сущностей (таблицы 2,3).

Таблица 2 – Атрибуты сущностей

Тип сущности	Атрибут
Users	UserID
	Username
	Login
	Password
	Date_registration
	Last_login_date
	Total_games
Game	GameID
	UserID
	Username
	Date_game
	Moves
	Researched_states
Information	InformationID
	Title
	Text_info

Таблица 3 – Сущности и их первичные ключи

Сущность	Первичный ключ
Users	UserID
Game	GameID
	UserID
Information	InformationID

Концептуальное проектирование

На основе анализа предметной области была построена ER – диаграмма, демонстрирующая концептуальную модель базы данных, которая представлена на рисунке 1.

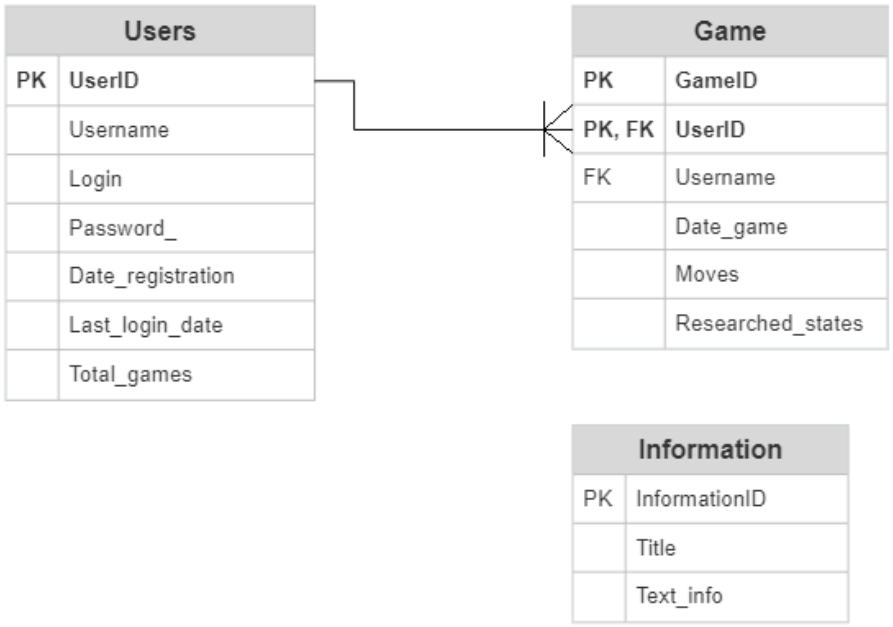


Рисунок 1 – Концептуальная модель базы данных

На схеме представлены следующие типы сущностей:

Родительские (справочные) сущности: Users, Game, Information

Дочерние сущности: Game

Описание средств используемой СУБД

PostgreSQL — система управления реляционными базами данных (СУБД), разработанная корпорацией Microsoft.

PostgreSQL обладает всеми качествами, необходимыми для реализации ключевых требований к СУБД, предъявленными заказчиком, а именно - производительностью, стабильность и возможность масштабирования.

Физическое проектирование

Для создания таблицы в PostgreSQL необходимо выполнить следующие действия:

– Подключиться к нужному нам серверу (рисунок 3).

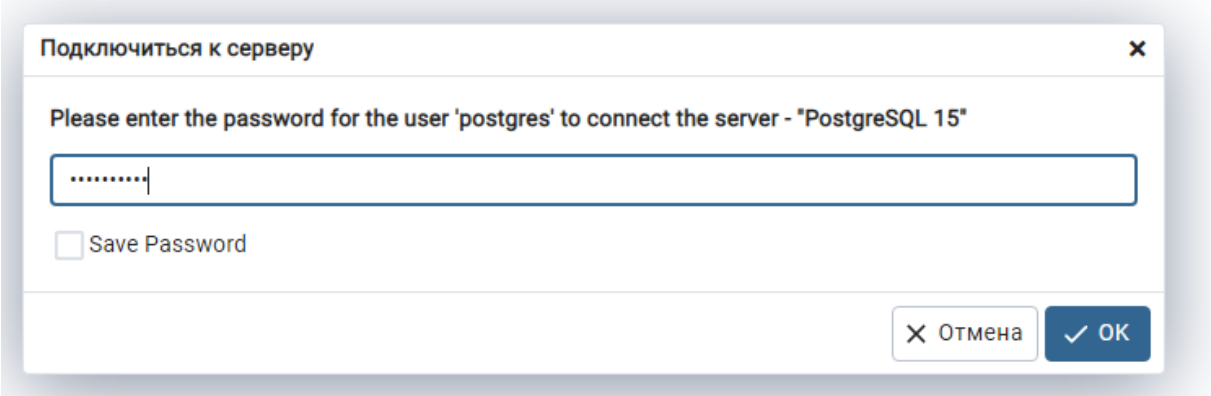


Рисунок 2 – Подключение к серверу PostgreSQL 15

– Правой кнопкой мыши нажимаем по пункту «Базы данных» и выбираем «Создать базу данных».

– В поле «Имя базы данных» вводим название базы и нажимаем ОК (рисунок 4). На этом этапе есть возможность указать место сохранения базы данных, но желательно использовать путь, который предлагает SQL.

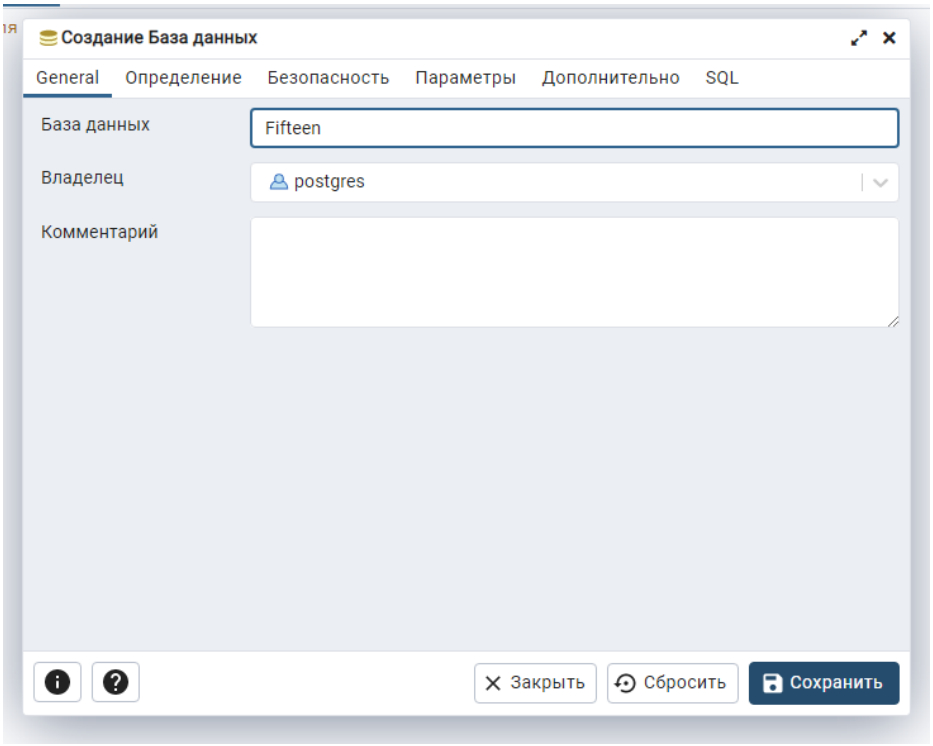


Рисунок 3 – Создание базы данных

– Щелкаем правой кнопкой мыши по Базе данных и выбираем пункт «Создать запрос».

– Далее появляется текстовый редактор запросов. В нем мы указываем таблицы, атрибуты, типы и создаем их связи (рисунок 5).

```
CREATE TABLE Users (
    UserID SERIAL PRIMARY KEY,
    Username VARCHAR NOT NULL,
    Login VARCHAR NOT NULL,
    Password_ VARCHAR NOT NULL,
    Date_registration TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    Last_login_date DATE DEFAULT CURRENT_TIMESTAMP,
    Total_games INT DEFAULT 0
);


CREATE TABLE Game (
    GameID SERIAL,
    UserID INT REFERENCES Users (UserID),
    Date_game TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    Moves INT DEFAULT 0,
    Researched_states INT DEFAULT 0,
    PRIMARY KEY (GameID, UserID)
);
CREATE TABLE Information(
InformationID SERIAL PRIMARY KEY,
Title VARCHAR UNIQUE,
Text_info TEXT
);
```

Рисунок 4 – Текст скрипта для создания таблиц базы данных

В таблице 4 представлена физическая структура таблиц базы данных.

Таблица 4 – Физическая структура таблиц базы данных Fifteen.

Ключ	Наименование поля	Тип данных	Обязательность заполнения	Заметки
Users				
PK	UserID	SERIAL	NOT NULL	Первичный ключ
	Username	VARCHAR	NOT NULL	UNIQUE
	Login	VARCHAR	NOT NULL	Логин
	Password_	VARCHAR	NOT NULL	Пароль
	Date_registration	DATE	NOT NULL	DEFAULT CURRENT_TIMESTAMP
	Last_login_date	DATE	NOT NULL	DEFAULT CURRENT_TIMESTAMP

Ключ	Наименование поля	Тип данных	Обязательность заполнения	Заметки
	Total_games	INT	NOT NULL	DEFAULT 0
Game				
PK	GameID	SERIAL	NOT NULL	Первичный ключ
FK	UserID	INT	NOT NULL	Внешний ключ
FK	Username	VARCHAR	NOT NULL	Внешний ключ
	Date_game	DATE	NOT NULL	DEFAULT CUR-RENT_TIMEST AMP
	Moves	INT	NOT NULL	DEFAULT 0
	Researched_states	INT	NOT NULL	DEFAULT 0
Information				
PK	InformationID	SERIAL	NOT NULL	Первичный ключ
	Title	VARCHAR	NOT NULL	UNIQUE
	Text_info	VARCHAR	NOT NULL	Текст информации

Описание используемых компонентов архитектуры сервера базы данных

Архитектура сервера PostgreSQL включает в себя несколько компонентов.

1 Сервер PostgreSQL - это основной компонент, который отвечает за хранение данных, управление сессиями и выполнение запросов. PostgreSQL может быть запущен на различных операционных системах, включая Linux, Windows, macOS.

2 Ядро базы данных - это основной компонент, который обеспечивает хранение и управление данными. Оно является объектно-ориентированным, что позволяет использовать различные типы данных, включая пользовательские типы.

3 Управление памятью - PostgreSQL имеет собственную систему управления памятью, которая обеспечивает эффективное использование системных ресурсов и уменьшает нагрузку на ОС.

4 Параллельное выполнение запросов - PostgreSQL поддерживает параллельное выполнение запросов, что позволяет процессору многозадачно обрабатывать несколько запросов одновременно.

Описанные компоненты архитектуры сервера PostgreSQL обеспечили эффективность, безопасность, высокую производительность и надежность работы с базой данных проекта.

База данных в PostgreSQL состоит из коллекции таблиц, в которой хранится особый набор структурированных данных. PostgreSQL реализован в архитектуре

клиент-сервер. Главный серверный процесс, управляющий файлами баз данных, принимающий подключения клиентских приложений и выполняющий различные запросы клиентов к базам данных. Эта программа сервера БД называется postgres.

Базы данных PostgreSQL хранятся в файловой системе в виде файлов. Файлы могут быть объединены в группы файлов. Экземпляр СУБД работает с несколькими базами данных. Эти базы данных называются кластером. Это не должно вводить в заблуждение — термин «кластер» имеет здесь не тот смысл, который в него обычно вкладывается: это не несколько экземпляров, работающих над одними данными, а именно несколько баз данных, обрабатываемых одним экземпляром. [#https://edu.postgrespro.ru/dbtech_part1-20181203.pdf]

2.2 Разработка клиентского приложения

Описание средств, используемых для разработки клиентского приложения

Для создания клиентского приложения «FifteenImage» использовались следующие программные средства:

– Visual Studio – интегрированной среды разработки приложений для платформы .NET и Windows:

- WPF – платформа пользовательского интерфейса для создания клиентских приложений;
- C# - объектно-ориентированный язык программирования.

– PostgreSQL – реляционная база данных с открытым кодом, декларативный язык программирования, применяемый для создания, модификаций и управления данными в реляционной базе данных.

Разработка программного решения

На рисунке 5 представлен обозреватель решений проекта .

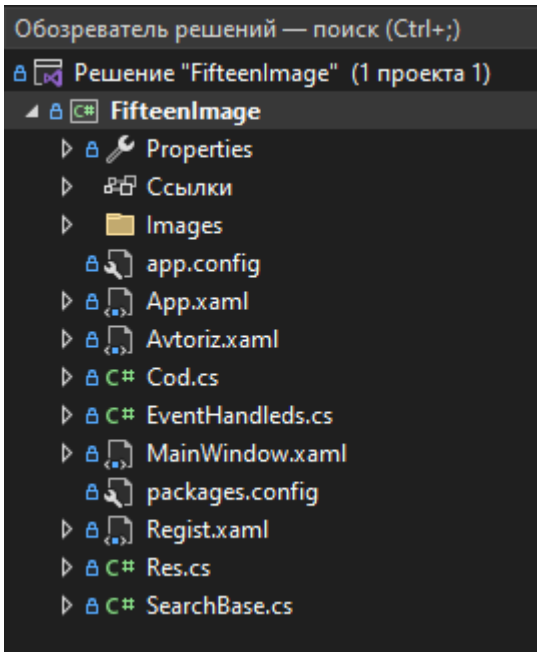


Рисунок 5 – Обозреватель решений проекта

Пояснения к рисунку:

- *FifteenImage* –вершина дерева решений, соответствующая названию проекта.
- *App.xaml* – модуль, содержащий все используемые стили в приложении.

– *Avtoriz.xaml* – модуль, содержащий описание формы авторизации на языке XAML, формирующийся автоматически.

На форме авторизации реализованы средства, позволяющие пользователю ввести свои логин и пароль для дальнейшей работы при авторизации или, если он заходит впервые, перейти на форму регистрации Regist.

– *Avtoriz.xaml.cs* – модуль формы авторизации на языке C#.

– *Cod.cs* – класс, управляющий алгоритмом IDA

– *EventHandleds* – класс, который реализует перемещение пятнашек по клеткам и другие методы

– *MainWindow.xaml* – модуль, содержащий реализацию игры «Пятнашки».

– *MainWindow.xaml.cs* – модуль, содержащий описание реализации игры на языке C#, разрабатываемый программистом.

– *Regist.xaml* – модуль, содержащий описание формы регистрации на языке XAML, формирующийся автоматически.

– *Regist.xaml.cs* – модуль, содержащий описание формы регистрации на языке C#, разрабатываемый программистом.

– *Res.cs* – класс, который инициализирует основные команды.

– *SearchBase* – класс, который описывает размерность матрицы поиска и другие методы

На форме реализации игры «Пятнашки» можно самому решить головоломку или воспользоваться помощью ИИ, нажав на кнопку в верхнем меню «Решение».

На форме авторизации пользователя реализована возможность войти в игру под своим профилем с сохранёнными результатами игр.

На форме регистрации пользователя реализована возможность зарегистрироваться в базе данных игры, чтобы в дальнейшем играть под своим профилем.

На форме статистики реализована возможность посмотреть статистику игр пользователя, которая принадлежит тому, кто вошёл в приложение под определённым логином и паролем.

Структура формы MainWindow.xaml.cs представлена ниже на рисунке 6.

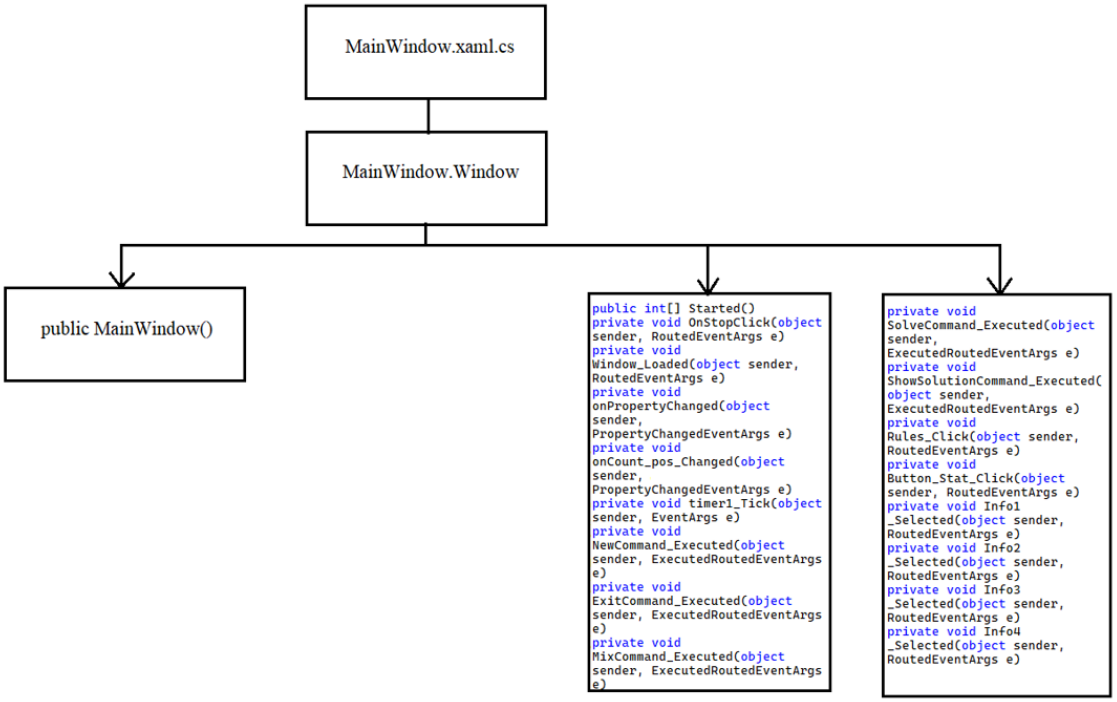


Рисунок 6 – Структура формы MainWindow

Фрагмент кода формы Avtoriz.xaml.cs обработчика события нажатия на кнопку «Вход» – Button_Vhod_Click.

```
        private void Button_Vhod_Click(object sender, RoutedEventArgs e)
        {
            try
            {
                bool ch = false;
                con.Open();
                //Выбор имени из бд в таблице пользователей
                //Проверка совпадения логина и пароля для входа
                com.CommandText = $"SELECT UserID FROM Users WHERE Login='{TextLogin.Text}' and Pass-
word_='{TextPassword.Password}'";
                Global.globalid = Convert.ToInt32(com.ExecuteScalar());
                if (TextLogin.Text == "")
                    {
                    TextLogin.Background = Brushes.IndianRed;
                    MessageBox.Show("Введите логин");
                    }
                else if (TextPassword.Password == "")
                    {
                    TextPassword.Background = Brushes.IndianRed;
                    MessageBox.Show("Введите пароль");
                    }
                reader = com.ExecuteReader();
                while (reader.Read()) { ch = true; }
                con.Close();
                if (ch == true)
                    {
                    MessageBox.Show("Добро пожаловать в игру Пятнашки !!");
                    MainWindow data = new MainWindow();
                    data.Show();
                    Close();
                    }
                else if (ch == false)
```

```
            {
            MessageBox.Show("Данные введены неверно");
            }
        con.Close();
    }
    catch (Exception ex)
    {
        MessageBox.Show("Возникло исключение: " + ex.Message);
        con.Close();
    }
}
```

Проектирование пользовательского интерфейса приложения

На рисунке 7 представлена схема пользовательского интерфейса окна MainWindow, реализованного с помощью языка разметки XAML;

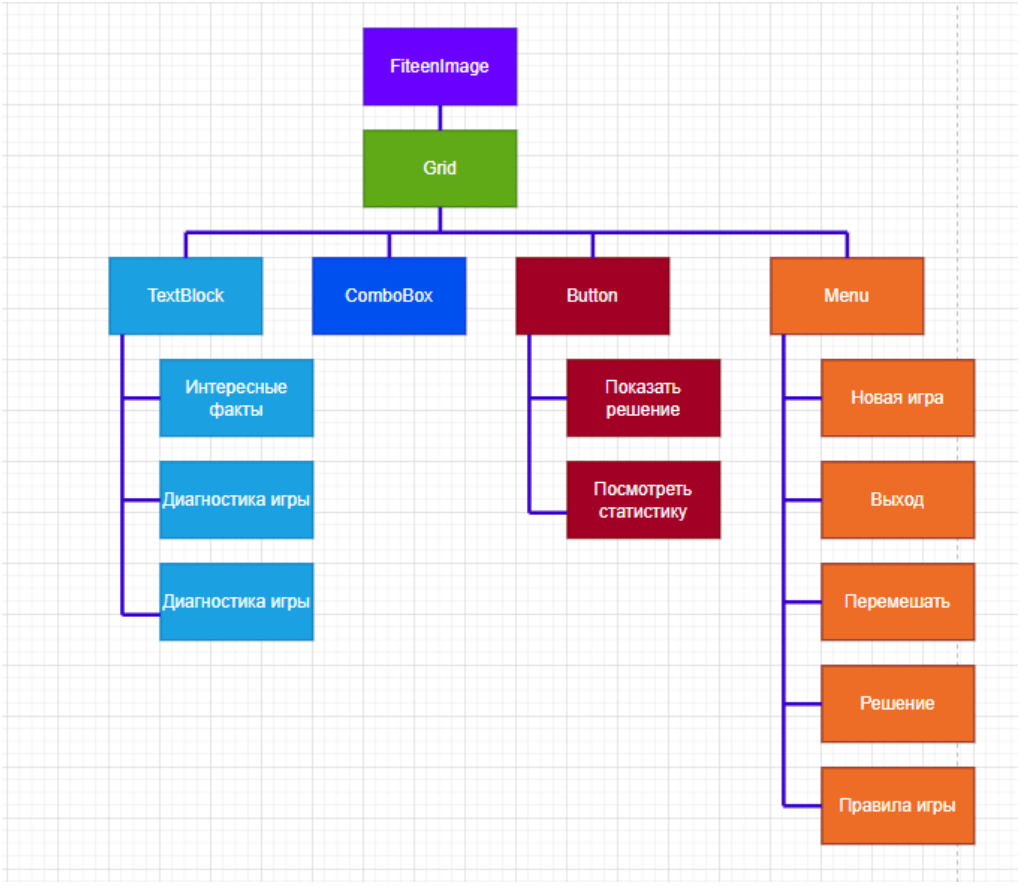


Рисунок 7 – Схема пользовательского интерфейса окна MainWindow

Пояснения к рисунку:

Главная окно

Таблица – Grid

Текстовое блок – TextBlock

Список – ComboBox

Кнопка – Button

Кнопка - меню – Menu

Описание средств доступа к базе данных

Для доступа к базе данных использовалась строка подключения к базе данных.

Программный код для подключения к базе данных:

```
static public NpgsqlConnection con = new NpgsqlConnection(@"Server=localhost; Port=5432; User Id=postgres; Password=Gfhjkm137!; Database = Fifteen;");
```

где:

Server = “Название сервера”

Port = “Порт к базе данных”

User ID = “Имя пользователя базы данных”

Password = “Пароль для подключения к базе данных”

Database = “Название базы данных”

Хранимая процедура, выполняемая над базой данных PostgreSQL Server (NpgsqlCommand):

```
static public NpgsqlCommand com = new NpgsqlCommand();
```

Чтение результатов запроса (NpgsqlDataReader):

```
NpgsqlDataReader reader = com.ExecuteReader();
```

Разработка PostgreSQL запросов

В клиентском приложении в соответствии с постановкой задачи разработаны следующие типы SQL – запросов:

1 Для запросов на добавление данных в таблицу при регистрации пользователя используется команда INSERT.

Фрагмент кода обработчика кнопки при регистрации пользователя.

```
private void Button_Regist_Click(object sender, RoutedEventArgs e)
```

```
            {
                try
                {
                    Avtoriz.con.Open();
                    com = new NpgsqlCommand($"INSERT INTO Users (username, login, password_) VALUES ('{Text-
Name.Text}', " +
                        $"'{TextLogin.Text}' '{TextPassword.Password}')", Avtoriz.con);
                    if (TextName.Text == "")
                    {
                        TextLogin.Background = Brushes.IndianRed;
                        MessageBox.Show("Введите имя пользователя");
                    }
                    else if (TextLogin.Text == "")
                    {
                        TextLogin.Background = Brushes.IndianRed;
                        MessageBox.Show("Введите логин");
                    }
                    else if (TextPassword.Password == "")
                    {
                        TextPassword.Background = Brushes.IndianRed;
                        MessageBox.Show("Введите пароль");
                    }
                    com.ExecuteNonQuery();
                    MessageBox.Show("Регистрация успешно завершилась!");
                    Avtoriz.con.Close();
                    Avtoriz avt = new Avtoriz();
                    avt.Show();
                    Close();
                }
                catch (Exception ex)
                {
                    MessageBox.Show("Возникло исключение: " + ex.Message);
                    Avtoriz.con.Close();
                }
                }
```

Запрос **Insert** добавляет данные в таблицу Users.

1. 2 Добавление данных в таблицу Game, при запуске начала игры.

```
com = new NpgsqlCommand($"INSERT INTO Game (userid, moves, researched_states) VALUES ('{Glob-
al.globalid}','{Global.globalmoves}','{Global.globalCountStates}')",Avtoriz.con);
```

Запрос **Insert** добавляет данные в таблицу Game.

Для запросов на обновление данных в таблицу при входе пользователя на окно игры используется команда UPDATE.

2 Обновление данных в таблице Users при запуске окна MainWindow

```
com = new NpgsqlCommand($"UPDATE Users SET total_games = 1 + total_games WHERE UserID = '{Global.globalid}'
", Avtoriz.con);
```

Запрос **Update** обновляет данные в столбце таблицы Users.

3 Для выбора определённых значений из таблицы базы данных используется команда SELECT.

Выбор имени пользователя при входе в приложение.

```
com.CommandText = $"SELECT UserID FROM Users WHERE Login='{TextLogin.Text}' and
Password_='{TextPassword.Password}'";
```

Запрос **Select** выбирет значение Username из таблицы Users.

3.1 Выбор интересного факта при выборке значения из ComboBox

```
com = new NpgsqlCommand($"SELECT text_info FROM Information WHERE Informationid=1", Avtoriz.con);
com = new NpgsqlCommand($"SELECT text_info FROM Information WHERE Informationid=2", Avtoriz.con);
com = new NpgsqlCommand($"SELECT text_info FROM Information WHERE Informationid=3", Avtoriz.con);
com = new NpgsqlCommand($"SELECT text_info FROM Information WHERE Informationid=4", Avtoriz.con);
```

Запрос **Select** выбирет значение text_info из таблицы Information.

3.2 Выбор информации о правилах, когда игрок нажимает кнопку в верхнем меню.

```
com = new NpgsqlCommand($"SELECT text_info FROM Information WHERE Informationid=1", Avtoriz.con);
```

Запрос **Select** выбирет значение text_info из таблицы Information.

3.3 Выбор даты регистрации пользователя при выводе статистики игр

```
com = new NpgsqlCommand($"SELECT date_registration FROM Users WHERE UserID = '{Global.globalid}'", Avtoriz.con);
```

Запрос **Select** выбирет значение date_registration из таблицы Users.

3.4 Выбор количества всех игр пользователя при выводе статистики игр

```
com = new NpgsqlCommand($"SELECT total_games FROM Users WHERE UserID = '{Global.globalid}'", Avtoriz.con);
```

Запрос **Select** выбирет значение total_games из таблицы Users.

3.5 Выбор даты последней игры пользователя при выводе статистики игр

```
com = new NpgsqlCommand($"SELECT date_game FROM Game WHERE UserID = '{Global.globalid}' ORDER BY date_game DESC LIMIT 1", Avtoriz.con);
```

Запрос **Select** выбирет значение date_game из таблицы Game.

Разработка средств аутентификации и регистрации пользователей

В информационной системе реализована парольная защита баз данных.

Парольная защита представляет простой и эффективный способ защиты БД от несанкционированного доступа. Пароли устанавливаются конечными пользователями БД. Пароль поможет входить игроку именно под свой, а не чужой профиль.

На рисунке 8 представлена форма авторизации Avtoriz.

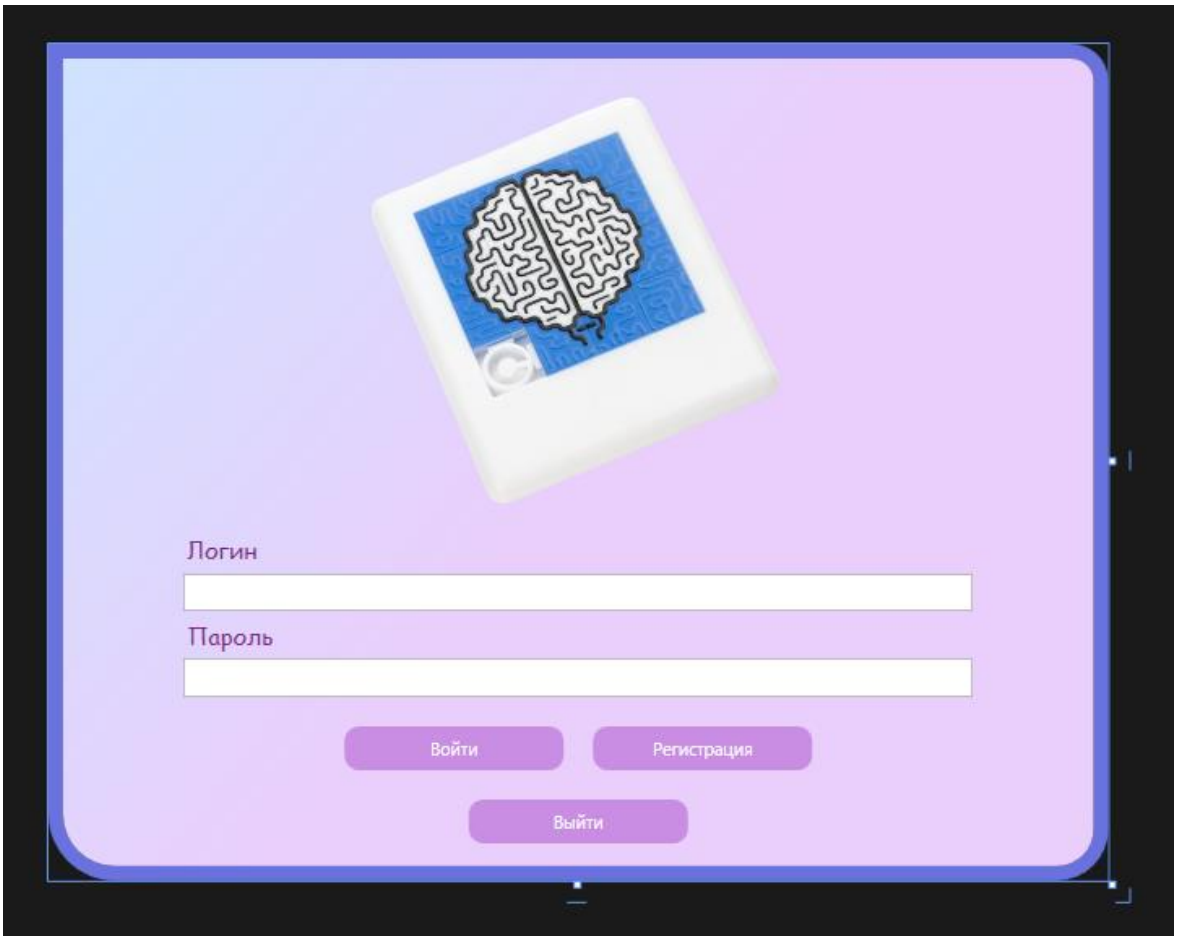


Рисунок 8 – Форма авторизации «Avtoriz»

Текст кода окна авторизации «Avtoriz» (XAML)

```
<Window x:Class="FifteenImage.Avtoriz"
      xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
      xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
      xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
      xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
      xmlns:local="clr-namespace:FifteenImage"
      mc:Ignorable="d"
      Title="Avtoriz" Height="572" Width="725" AllowsTransparency="True"
      WindowStyle="None" Background="Transparent" WindowStartupLocation="CenterScreen">
   <Grid>
      <Border Width="Auto" Height="Auto" Name="windowFrame"
    BorderBrush="#6972de" BorderThickness="10" CornerRadius="0,20,30,40" >
         <Border.Background>
            <LinearGradientBrush>
               <GradientBrush.GradientStops>
                  <GradientStopCollection>
                     <GradientStop Color="#CEE3FF" Offset="0.0"/>
                     <GradientStop Color="#eacffc" Offset="0.5"/>
                  </GradientStopCollection>
               </GradientBrush.GradientStops>
            </LinearGradientBrush>
         </Border.Background>
         <StackPanel Orientation="Vertical" VerticalAlignment="Center">
            <Image Source="Images/logo.png" Height="304" Width="508"></Image>
            <Label Style="{StaticResource Label_Menu}">Логин</Label>
            <TextBox x:Name="TextLogin" Style="{StaticResource TextBox_Menu}" ToolTip="Введите логин"/>
```

```
                <Label Style="{StaticResource Label_Menu}">Пароль</Label>
                <PasswordBox Name="TextPassword" Style="{StaticResource Pass_menu}" ToolTip="Введите пароль"></PasswordBox>
                <StackPanel Orientation="Horizontal" HorizontalAlignment="Center" Margin="20px">
                  <Button x:Name="Button_Vhod" Click="Button_Vhod_Click" Style="{StaticResource Button_Main}" Content="Войти"></Button>
                  <Button x:Name="Button_Regist" Click="Button_Regist_Click" Style="{StaticResource Button_Main}" Content="Регистрация" Margin="20 0 0 0"></Button>
                </StackPanel>
                <Button x:Name="Exit_Button" Click="Exit_Button_Click" Style="{StaticResource Button_Main}" Content="Выйти"></Button>
              </StackPanel>
            </Border>
          </Grid>
        </Window>
```

Текст кода окна авторизации «Avtoriz» (C#), с комментариями

```
using Microsoft.Win32;
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 Npgsql;

namespace FifteenImage
{
    /// <summary>
    /// Логика взаимодействия для Avtoriz.xaml
    /// </summary>
    public partial class Avtoriz : Window
    {

        //Создание переменных для подключения и выполнения команд в бд
        static public NpgsqlCommand com = new NpgsqlCommand();
        static public NpgsqlConnection con = new NpgsqlConnection(@"Server=localhost; Port=5432; User Id=postgres; Password=Gfhjkm137!; Database = Fifteen;");
        static public NpgsqlDataReader reader = null;

        //Глобальная переменная, значение которой возьмётся из этого окна
        public int GlobalId { get; set; }

        public Avtoriz()
        {
            InitializeComponent();
            com.Connection = con;

        }

        //Обработчик события нажатия на кнопку входа
        private void Button_Vhod_Click(object sender, RoutedEventArgs e)
        {
            try
            {
                bool ch = false;
                con.Open();
                //Выбор имени из бд в таблице пользователей
```

```
                //Проверка совпадения логина и пароля для входа
                com.CommandText = $"SELECT UserID FROM Users WHERE Login='{TextLogin.Text}' and Pass-
word_='{TextPassword.Password}'";
                Global.globalid = Convert.ToInt32(com.ExecuteScalar());
                if (TextLogin.Text == "")
                    {
                    TextLogin.Background = Brushes.IndianRed;
                    MessageBox.Show("Введите логин");
                    }
                else if (TextPassword.Password == "")
                    {
                    TextPassword.Background = Brushes.IndianRed;
                    MessageBox.Show("Введите пароль");
                    }
                reader = com.ExecuteReader();
                while (reader.Read()) { ch = true; }
                con.Close();
                if (ch == true)
                    {
                    MessageBox.Show("Добро пожаловать в игру Пятнашки !!");
                    MainWindow data = new MainWindow();
                    data.Show();
                    Close();
                    }
                else if (ch == false)
                    {
                    MessageBox.Show("Данные введены неверно");
                    }
                con.Close();
            }
            catch (Exception ex)
            {
                MessageBox.Show("Возникло исключение: " + ex.Message);
                con.Close();
            }
        }
        //Обработчик события нажатия на кнопку регистрации
        private void Button_Regist_Click(object sender, RoutedEventArgs e)
        {
            MessageBox.Show("Переходим на окно регистрации");
            Regist reg = new Regist();
            reg.Show();
            Close();
        }
        //Обработчик события нажатия на кнопку выхода
        private void Exit_Button_Click(object sender, RoutedEventArgs e)
        {
            Close();
        }
    }
}
```

Разработка искусственного интеллекта решателя для пятнашек

В приложении реализован автоматический решатель пятнашек. Теперь игрок может выбрать – решать ему самому или же воспользоваться помощью искусственного интеллекта приложения. Для создания решателя требуется создать несколько классов. Никаких дополнительных библиотек не подключаем, пользуемся стандартными функциями и методами C#.

Для того чтобы увидеть решение головоломки, игроку требуется нажать в верхнем меню на кнопку «Решение», которая в свою очередь запустит команду Solve. Код в XAML:

```
<MenuItem Header='Решение' Command="c:DataCommands.Solve" />

<CommandBinding Command="c:DataCommands.Solve"
    Executed="SolveCommand_Executed"></CommandBinding>
```

Обработчик события нажатия на кнопку «Решение» в верхнем меню (команды). Код C#:

```
private void SolveCommand_Executed(object sender, ExecutedRoutedEventArgs e)
      {
         _ = Dispatcher.BeginInvoke(
            (ThreadStart)delegate
            {
               _ida.DoStart();
            }
          , DispatcherPriority.Normal);
         try
         {
            Avtoriz.con.Open();
            com = new NpgsqlCommand($"UPDATE Users SET total_games = 1 + total_games WHERE Username =
'{Global.globalusername}' ", Avtoriz.con);
            com.ExecuteNonQuery();
            com = new NpgsqlCommand($"INSERT INTO Game (userid, username, moves, researched_states) VALUES
((SELECT userid FROM Users WHERE Username = '{Glob-
al.globalusername}'),'{Global.globalusername}','{Global.globalmoves}','{Global.globalCountStates}') ", Avtoriz.con);
            com.ExecuteNonQuery();
            dataAdapter = new NpgsqlDataAdapter(com);
            Avtoriz.con.Close();
         }
         catch (Exception ex)
         {
            MessageBox.Show("Возникло исключение: " + ex.Message);
            Avtoriz.con.Close();
         }
      }
```

Инициализация команды в классе Res:

```
solve = new RoutedUICommand(
          "Solve", "Solve", typeof(DataCommands), inputs4);
```

Метод в классе EventHandleds для видимости кнопки «Показать решение»:

```
private void OnFinSolve(object sender, SearchEventArgs e)
      {
         Dispatcher.BeginInvoke(
            (ThreadStart)delegate {
               ShowSolve.Visibility = Visibility.Visible;
            }
          , DispatcherPriority.Normal);
      }
```

Метод в классе EventHandleds для видимого решения пятнашек, без участия игрока. С помощью координат передвигаются клетки.

```
private void button_Click(object sender, EventArgs e)
      {
         Button but = sender as Button;
         _zeroPosition.X = (int)_buttons[15].GetValue(Grid.ColumnProperty) ;
         _zeroPosition.Y = (int)_buttons[15].GetValue(Grid.RowProperty) ;
```

```
            _butPosition.X = (int)but.GetValue(Grid.ColumnProperty);
            _butPosition.Y = (int)but.GetValue(Grid.RowProperty);
            if (Math.Abs(_butPosition.X - _zeroPosition.X) + Math.Abs(_butPosition.Y - _zeroPosition.Y) == 1)
            {
                _buttons[15].SetValue(Grid.RowProperty, (int)_butPosition.Y);
                _buttons[15].SetValue(Grid.ColumnProperty, (int)_butPosition.X);
                but.SetValue(Grid.RowProperty, (int)_zeroPosition.Y);
                but.SetValue(Grid.ColumnProperty, (int)_zeroPosition.X);
                int nullPos = (int)(_zeroPosition.Y * 4 + _zeroPosition.X);
                int clikPos = (int)(_butPosition.Y * 4 + _butPosition.X);
                _ida._start[nullPos] = _ida._start[clikPos];
                _ida._start[clikPos] = 0;
                progressBar1.Value = CalcProgrValue();
            }
            Count_pos++;
        }
```

Изменение значения StatusBar. Как только пользователь или искусственный интеллект приближается к завершению головоломки, полоска на главном окне игры заполняется. Метод, реализованный в классе EventHandleds

```
private int CalcProgrValue()
        {
            int val = 0;
            for (int i = 0; i < 16; i++)
            {
                if (_targetStat[i] == _ida._start[i]) val++;
            }
            return val;
        }
```

Изменение текста «Удачной игры» в главном окне игры на текст решения, который будет показываться при решении головоломки искусственным интеллектом. Решение будет генерироваться в реальном времени, то есть пользователь сможет видеть, как программа сама решает головоломку.

```
private void OnChangeText(object sender, SearchEventArgs e)
        {
            Dispatcher.BeginInvoke(
                (ThreadStart)delegate {
                Tblock.Text = e.Mes;
            }
             ,DispatcherPriority.Normal  );

        }
```

3 Порядок работы с программным продуктом

1 Для работы приложения необходима база данных «Fifteen» и приложение FifteenImage.

2 Для запуска приложения необходимо выполнить двойной щелчок левой кнопки мыши по пиктограмме с именем FifteenImage.exe - ярлык (рисунок 9).

Рисунок 9 – Запуск приложения

3 После запуска пользователю необходимо ввести логин и пароль на форме Avtoriz (рисунок 10), после успешной авторизации откроется форма самой игры в пятнашки (рисунок 11).

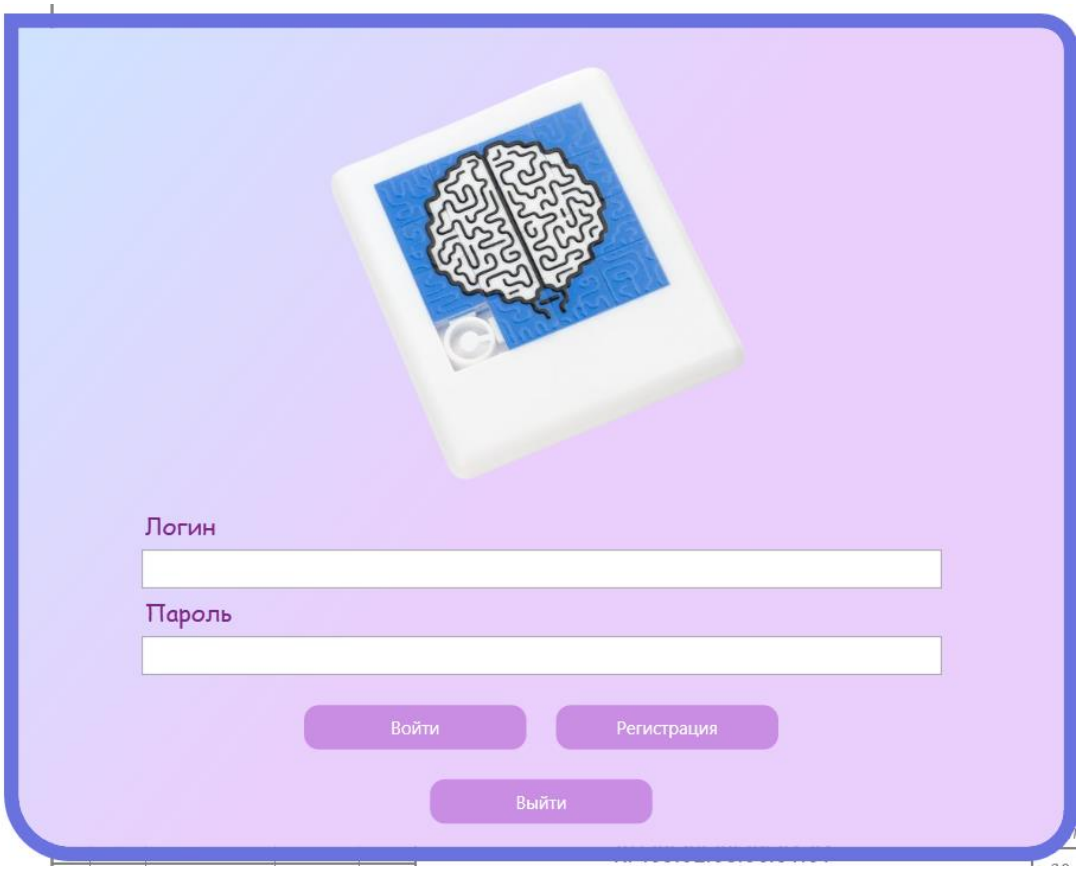


Рисунок 10 – Авторизация, форма Avtoriz

Рисунок 11 – Главное окно игры в пятнашки, форма MainWindow

4 На форме игры MainWindow (рисунок 11) можно:

– начать играть самостоятельно;

– перемешать игру;

– посмотреть решение головоломки;

– прочитать интересные факты про пятнашки и правила игры;

– перейти на окно с просмотром статистики.

5 На форме Statistics (рисунок 12) пользователь может посмотреть свою статистику игр, основанную на предыдущих его решениях, которые записывались в базу данных.

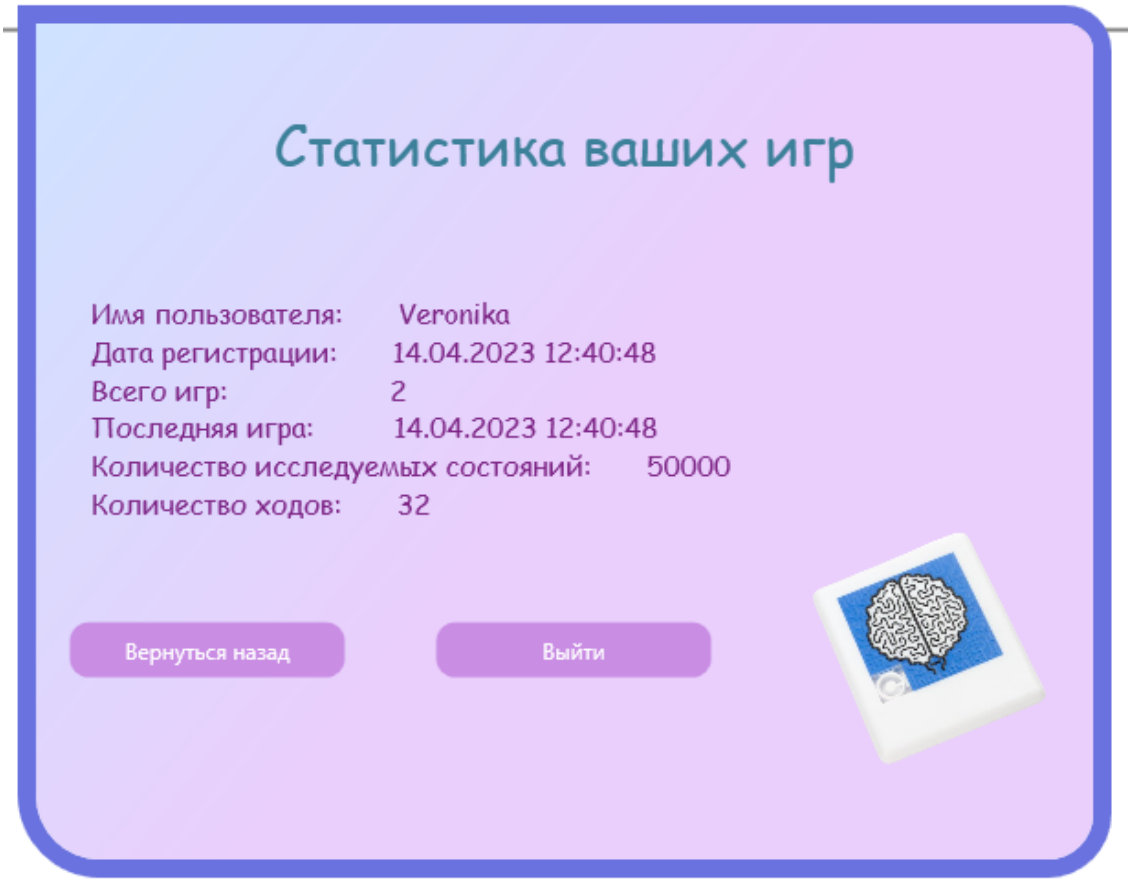


Рисунок 12 – Окно статистики игр, форма Statistics

6 Если же у пользователя нет профиля в игре, то он может зарегистрироваться на форме Regist (рисунок 13), его данные внесутся в базу данных, в дальнейшем он сможет входить в приложение под своим профилем.

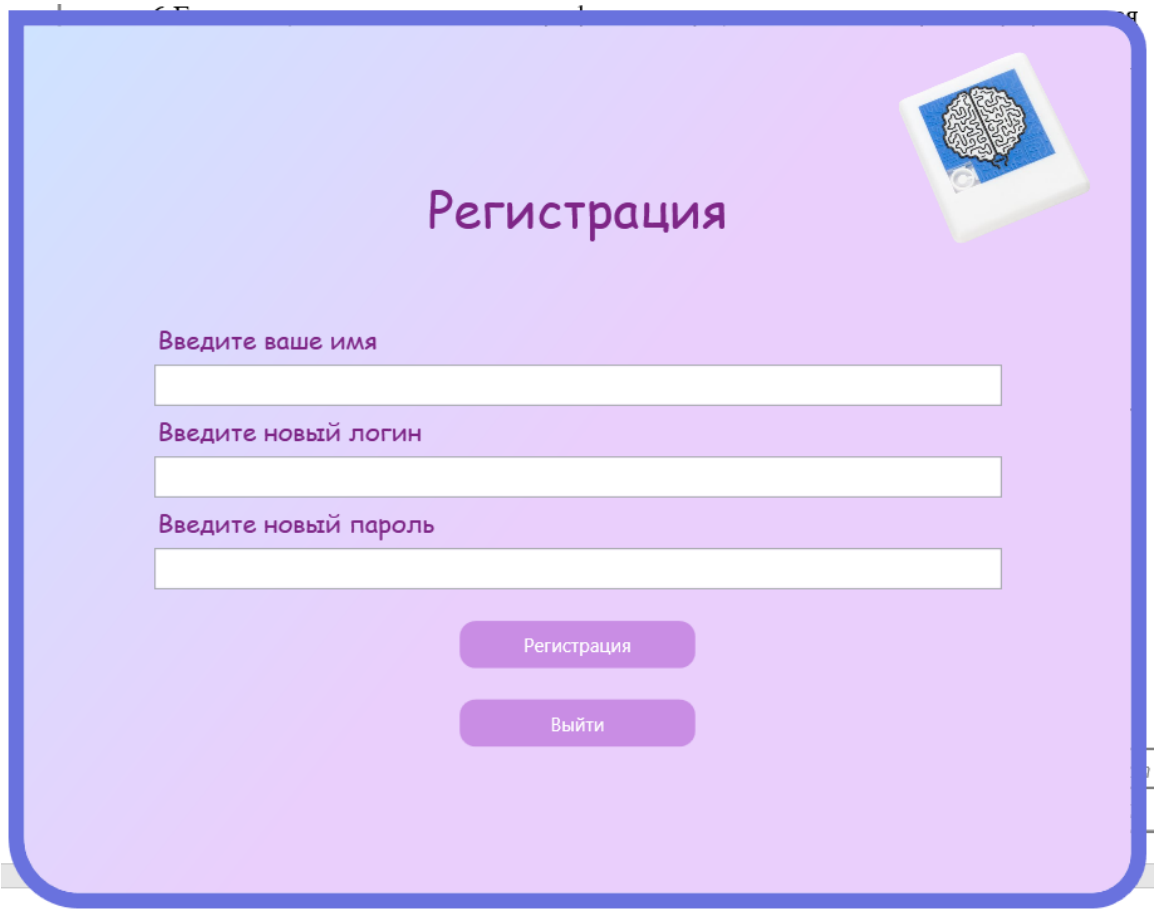


Рисунок 13 – Регистрация, форма Regist

Размещение проекта на GitHub

1 Войти в свой профиль в свои репозитории и нажать на кнопку создания нового репозитория (рисунок 14).

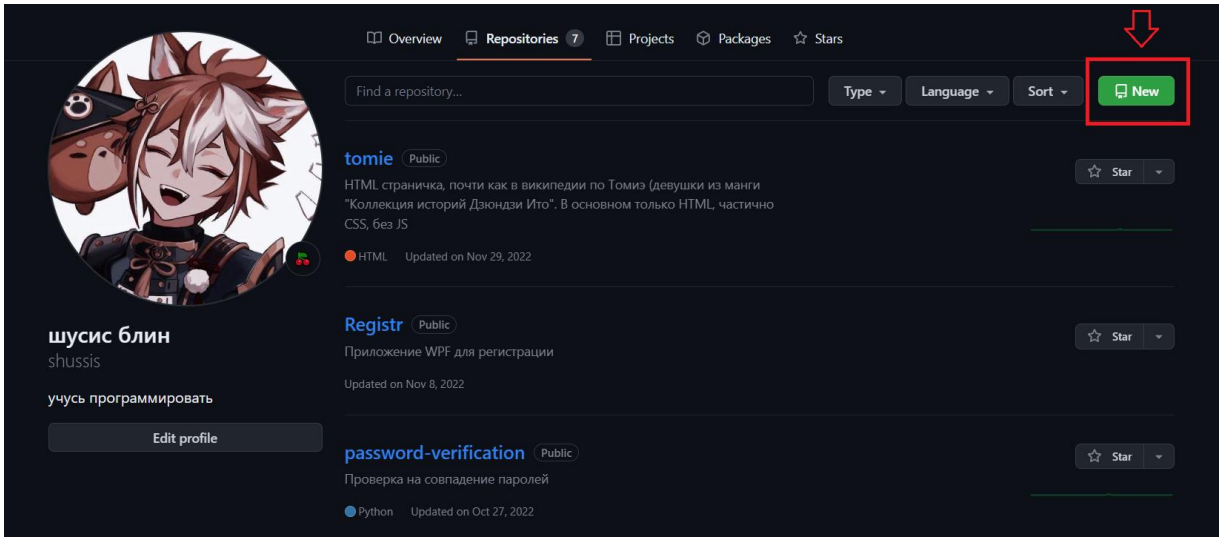
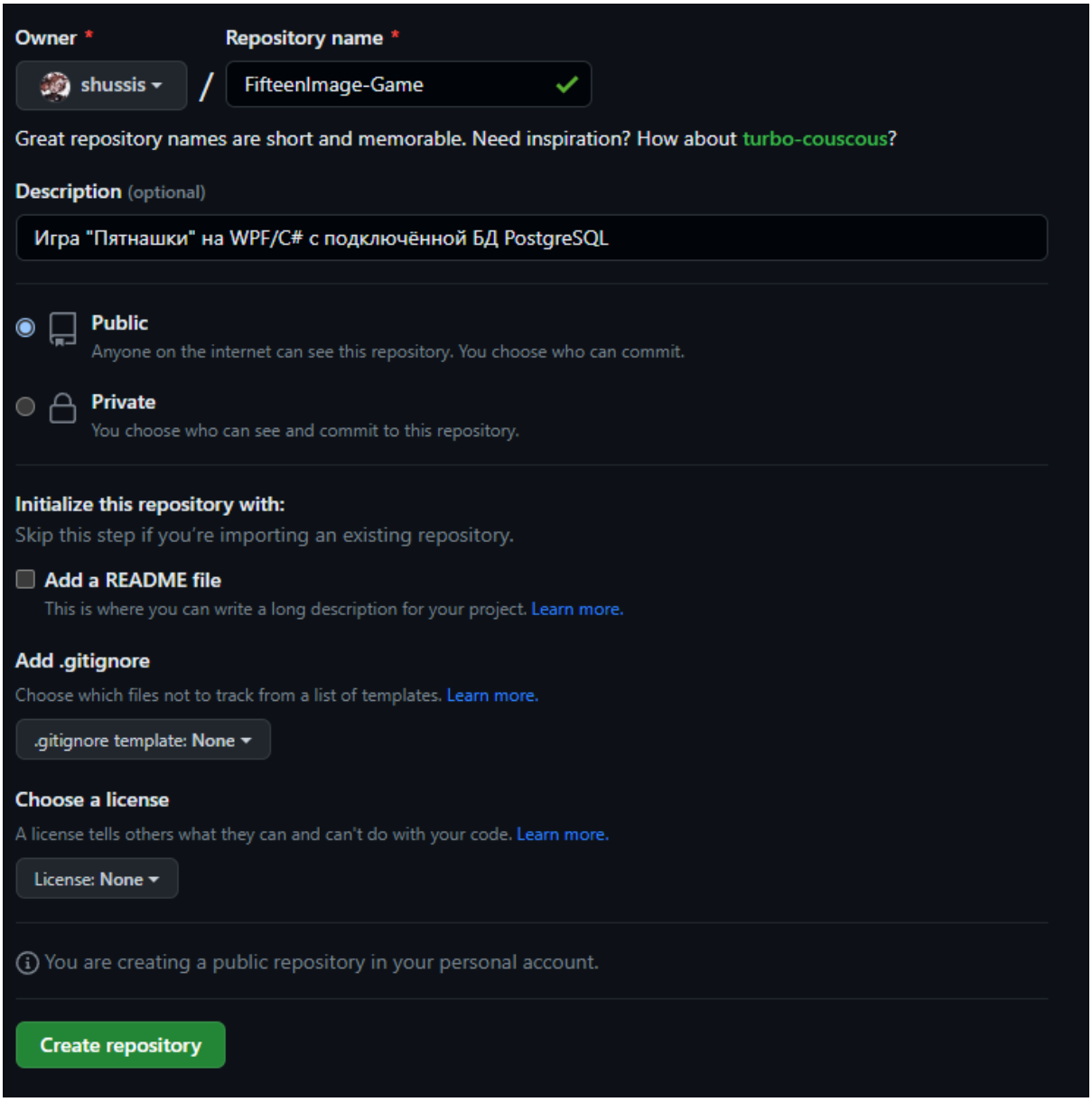


Рисунок 14 – Вкладка всех репозиториев на GitHub

Вводим название репозитория «FifteenImage Game», так же нужно ввести краткое описание. Репозиторий оставляем публичным. Создаём репозиторий, нажимая кнопку снизу «Create repository». После нажатия на кнопку сайт нам выдаёт ссылку (рисунок 15).



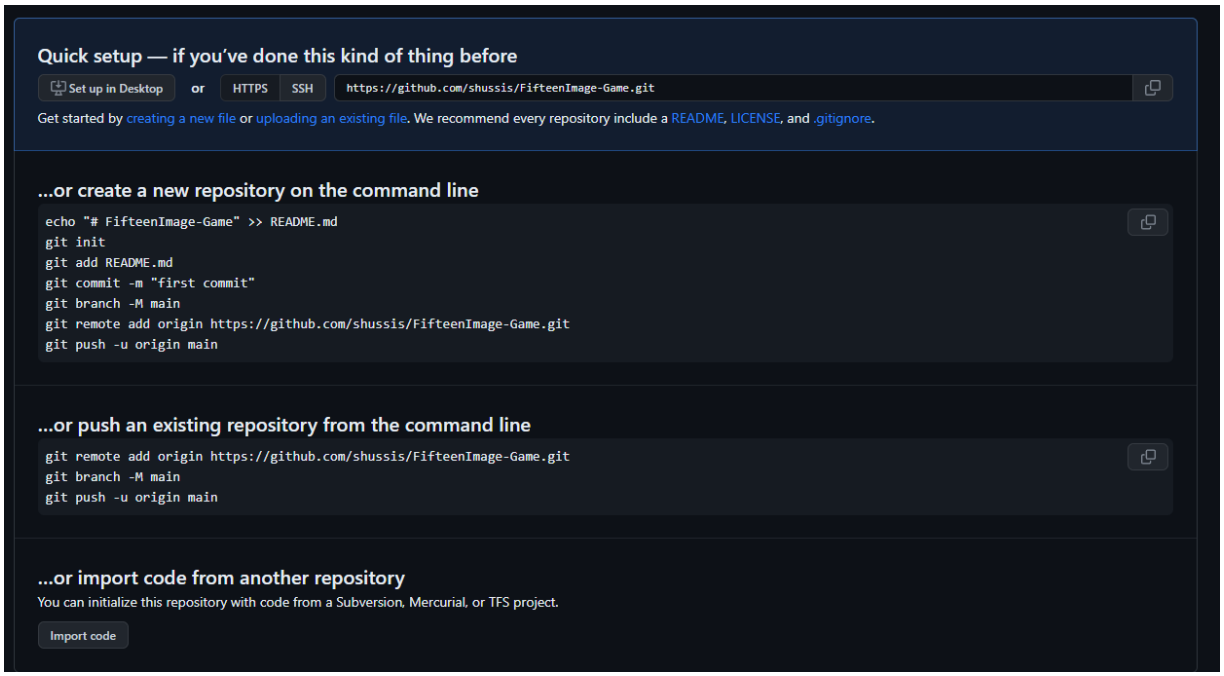
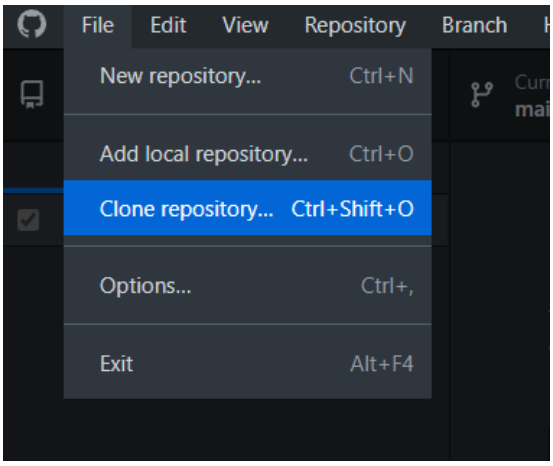


Рисунок 15 – Создание нового репозитория

Репозиторий создан, теперь остаётся поместить весь проект в него. Запускаем приложение GitHub Desktop, с помощью него запросто переместим все файлы проекта в созданный репозиторий. Далее в верхнем меню выбираем пункт «File» и выбираем клонирование репозитория «Clone repository». Из представленного списка всех репозиториев в профиле выбираем пустой FifteenImage-Game, собственно как мы его назвали ранее (рисунок 16)



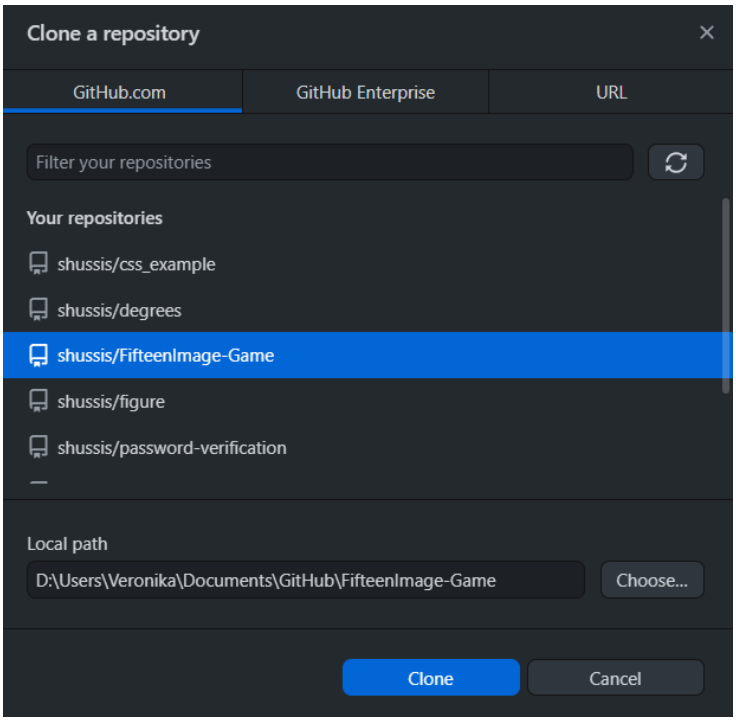


Рисунок 16 – Клонирование репозитория

Репозиторий клонировался. Далее нажимаем кнопку «Show in Exploer», с помощью чего сможем загрузить файлы проекта в новый созданный репозиторий. Перемещаем все файлы приложения в новую папку репозитория и нажимаем кнопку «Commit to main» (рисунок 17).

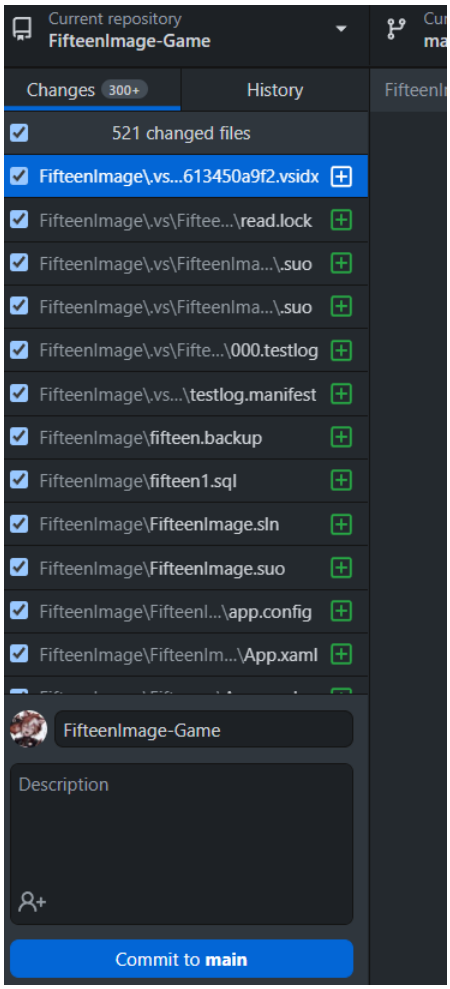


Рисунок 17 –Файлы проекта, перемещённые в новый репозиторий

Обновляем публичный репозиторий, нажимая на кнопку «Publish branch» (рисунок 18). После этих действий проект готов быть увиденным другими людьми!

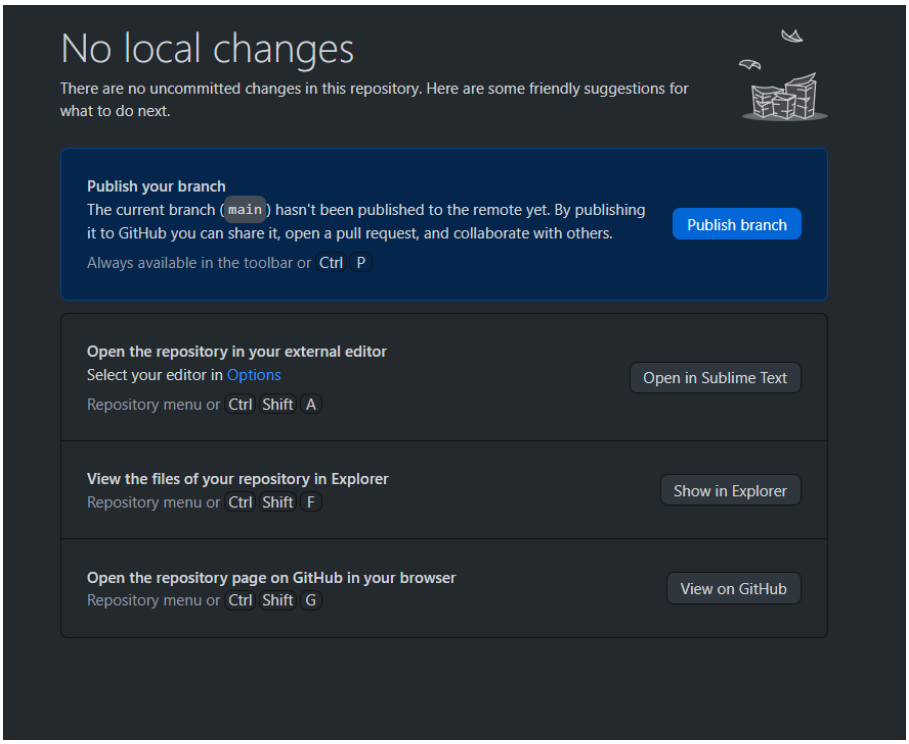


Рисунок 18 – Обновление публичного репозитория

Все файлы кода игры и базы данных доступны в публичном доступе на GitHub (рисунок 19). Теперь решатель для игры-головоломки «Пятнашки» может просмотреть абсолютно каждый пользователь Интернета по этой ссылке https://github.com/shussis/FifteenImage-Game

Рисунок 19 – Готовый проект «Пятнашки» на GitHub

Заключение

В результате разработки проекта выполнены все поставленные задачи. Реализованные алгоритмы сокращают долгое время решения головоломки, причиной которым ранее был человеческий фактор, так же позволяют увидеть решение после завершённой игры, что возможно натолкнёт игрока на новые мыслительные процессы в будущих играх.

Автоматическое решение пятнашек позволит:

- сократить время, затрачиваемое на решение головоломки;
- посмотреть решение для дальнейших выводов в будущих играх;
- возможно, найти какие-либо тактики решения;
- учиться на наглядных примерах решения;
- уменьшить число ошибок при игре;

В результате спроектирована база данных в PostgreSQL, составлены таблицы, составлена диаграмма базы данных, отображающая сущности с установленными связями.

Так же был построен интерфейс для связи пользователя с БД и внесения новых игроков в среде разработки WPF C#.

В дальнейшем развитии системы предполагается расширение функциональных возможностей информационной системы:

- Разработка ИИ, который сможет давать прогноз на будущие игры.
- Разработка средств анализа динамики влияния игры на развитие логического мышления.
- Разработка средства выявления лучших игроков, которые использовали автоматический решатель и без него за заданный период.

Литература

1 https://habr.com/ru/companies/skillfactory/articles/655629/ – статья, из которой брались фрагменты кода для автоматического решателя пятнашек.

2 https://www.youtube.com/watch?v=ujeVhBeSEvU – видео, из которого брались фрагменты кода для реализации самой игры «Пятнашки».

3 https://www.youtube.com/watch?v=ZMiCh_VnHtY – продолжение видео, которое написано выше.

4 http://aevolodin.blogspot.com/2012/11/wpf-net.html – статья, из которой брался фрагмент кода для перемешивания пятнашек.

5 https://www.cyberforum.ru/wpf-silverlight/thread2467575.html?nojs=1 – форум, с которого брался фрагмент кода логики координат пятнашек.

6 https://www.cyberforum.ru/wpf-silverlight/thread2461170.html#post13609239 – форум про игру 2048, с которого брались некоторые фрагменты кода для проекта.

7 https://cyberleninka.ru/article/n/sozdanie-igry-pyatnashki-na-yazyke-c/viewer – электронная книга, из которой брались некоторые фрагменты кода для пятнашек.

8 https://ru.stackoverflow.com/questions/914054/ – статья, из которой брался фрагмент для оптимального решения пятнашек.

9 https://www.i-igrushki.ru/igrushkapedia/pyatnashki.html – статья про историю и описание игры Пятнашки

10 https://edu.postgrespro.ru/dbtech_part1-20181203.pdf – учебник PostgreSQL