Министерство образования Республики Беларусь
Учреждение образования «Белорусский государственный университет информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра электронных вычислительных машин

Дисциплина: Программирование на языках высокого уровня

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
к курсовому проекту
на тему

ИГРА «КТО ХОЧЕТ СТАТЬ МИЛЛИОНЕРОМ»

БГУИР КП 1-40 02 01 229 ПЗ

Студент: гр. 250502 Цвирко Е. Д.

Руководитель: Богдан Е. В.

МИНСК 2023

СОДЕРЖАНИЕ

Учреждение образования
«Белорусский государственный университет информатики
и радиоэлектроники»

Факультет компьютерных систем и сетей

УТВЕРЖДАЮ
Заведующий кафедрой

_____
(подпись)
_____20 г.

ЗАДАНИЕ
по курсовому проектированию

Студенту _____*Цвирко Егору Дмитриевичу*_____

Тема проекта___*игра "Кто хочет стать миллионером"*_____

2. Срок сдачи студентом законченного проекта____*15 декабря 2023 г.*___

3. Исходные данные к проекту_____

Файлы q1.txt, q2.txt, q3.txt, q4.txt, q5.txt, q6.txt, q7.txt, q8.txt, q9.txt, q10.txt (информация о вопросах и ответах)

Файл Records.txt (информация о рекордах)

4. Содержание расчетно-пояснительной записки (перечень вопросов, которые подлежат разработке)_____

1. Лист задания.
2. Введение.
3. Обзор литературы.
  3.1. Обзор методов и алгоритмов решения поставленной задачи.
4. Функциональное проектирование.
  4.1. Структура входных и выходных данных.
  4.2. Разработка диаграммы классов.
  4.3. Описание классов.
5. Разработка программных модулей.
  5.1. Разработка схем алгоритмов (два наиболее важных метода).

5.2. Разработка алгоритмов (описание алгоритмов по шагам, для двух методов).
6. Результаты работы.
7. Заключение
8. Литература
9. Приложения

5. Перечень графического материала (с точным обозначением обязательных чертежей и графиков)

1. Диаграмма классов.

2. Схема алгоритма метода 1

3. Схема алгоритма метода 2

6. Консультант по проекту (с обозначением разделов проекта) А. М. Ковальчук

7. Дата выдачи задания *15.09.2023г.*

8. Календарный график работы над проектом на весь период проектирования (с обозначением сроков выполнения и трудоемкости отдельных этапов):

1. Выбор задания. Разработка содержания пояснительной записки. Перечень графического материала – 15 %;

разделы 2, 3 – 10 %;

разделы 4 к –20 %;

разделы 5 к – 35 %;

раздел 6,7,8 – 5 %;

раздел 9 к – 5%;

оформление пояснительной записки и графического материала к 15.12.22 – 10 %

Защита курсового проекта с 21.12 по 28.12.23г.

РУКОВОДИТЕЛЬ Е. В. Богдан

(подпись)

Задание принял к исполнению_____

(дата и подпись студента)

ВВЕДЕНИЕ

Язык C++ является невероятно мощным инструментом для опытных разработчиков. Он обладает уникальной гибкостью, предоставляя три различные парадигмы программирования, что позволяет строить структуры программ по-разному. C++ сочетает в себе характеристики как высокоуровневых, так и низкоуровневых языков программирования, делая его универсальным инструментом.

По сравнению с языком C, C++ был создан с целью расширения возможностей программирования. Он внедрил парадигму объектно-ориентированного программирования (ООП) и обобщенное программирование, что позволяет разработчикам более эффективно организовывать и переиспользовать код.

C++ широко применяется в различных областях, включая разработку программного обеспечения, создание игр, разработку операционных систем, написание драйверов и решение множества других задач. Его богатая стандартная библиотека включает в себя разнообразные алгоритмы, контейнеры, средства ввода-вывода, обработку строк, языковую поддержку, регулярные выражения, возможности для многозадачности и многое другое.

Еще одним важным аспектом C++ является его эффективность. Он предоставляет возможность непосредственного управления памятью, что позволяет оптимизировать производительность приложений, особенно в случае требовательных к ресурсам задач, таких как игры или системное программное обеспечение.

SFML (Simple and Fast Multimedia Library) является удобным инструментом для разработки мультимедийных приложений с графическим интерфейсом. Совмещая в себе модули для работы с аудио, графикой, окнами и сетью, SFML является универсальным средством для создания различных игр и приложений на C++ [1].

1 ПОСТАНОВКА ЗАДАЧИ

Для создания игры необходимо исследовать возможность применения автомата конечных состояний в качестве составной части игрового цикла. Также необходимо создать классы для работы с игровыми ресурсами, для обработки ввода от пользователя, взаимодействия с текстовыми файлами.

Необходимо ознакомиться с возможностями библиотеки SFML, которые можно использовать при создании приложения с графическим интерфейсом (графика, звук, сеть, система, события).

Следует реализовать алгоритмы перехода между состояниями, алгоритмы игровой логики: проверка правильного ответа, использование бонусов “Право на ошибку”, “Пропуск вопроса”, “Смена вопроса”, подсчет очков и сохранение результата в текстовый файл.

Необходимо создать текстовые файлы, которые будут содержать вопросы к игре, ответы на них и номер правильного ответа и реализовать их взаимодействие с классом состояния игрового уровня.

Возможно добавить в игру музыкальное сопровождение и звуковые эффекты для более интересного времяпрепровождения.

2 ОБЗОР ЛИТЕРАТУРЫ

2.1 Обзор методов и алгоритмов решения поставленной задачи

Для решения задачи был выбран язык программирования C++ и методология объектно-ориентированного программирования.

В процессе разработки программы были использованы различные возможности языка C++ и его фреймворка SFML, которые будут описаны ниже.

В качестве элементов графического интерфейса используются встроенные классы SFML:

`sf::Texture` – класс для работы с текстурами;

`sf::Sprite` – класс для вывода текстур на экран;

`sf::Text` – класс для вывода текста на экран;

`sf::SoundBuffer` – класс для хранения аудиофайлов;

`sf::Sound` – класс для проигрывания звуков;

`sf::Vector2i` и `sf::Vector2f` – экземпляры шаблонного класса `sf::Vector2<T>` для работы с двумерными векторами;

`sf::Mouse` – класс текущего состояния мыши;

`sf::Keyboard` – класс текущего состояния клавиатуры;

`sf::RenderWindow` – класс окна, способный взаимодействовать с другими графическими классами;

`sf::Font` – класс для работы со шрифтами;

`sf::Event` – класс для определения системных событий;

`sf::IntRect` и `sf::FloatRect` – экземпляры шаблонного класса `sf::Rect<T>` для работы с прямоугольниками;

`sf::Color` – класс для работы с цветами;

`sf::Clock` – класс для измерения времени;

`sf::VideoMode` – класс для запуска видеорежима.

В работе используется класс `StateMachine`, написанный на основе концепции конечного автомата – он предназначен для эффективной организации логики и управления состояниями игры. Класс предоставляет возможность перехода между состояниями, каждое состояние имеет методы для обработки ввода, отрисовки и обновления объектов.

2.2 Обзор аналогов приложения

2.2.1 Who Wants To Be A Millionaire от Ludia

Версия игры от Ludia была одной из самых известных адапатаций телешоу. Главные отличия от данного проекта: наличие 3D-анимаций, другие

темы вопросов, возможность создать своего персонажа из предустановленных пресетов, другие бонусы и небольшие отличия в правилах игры [2].

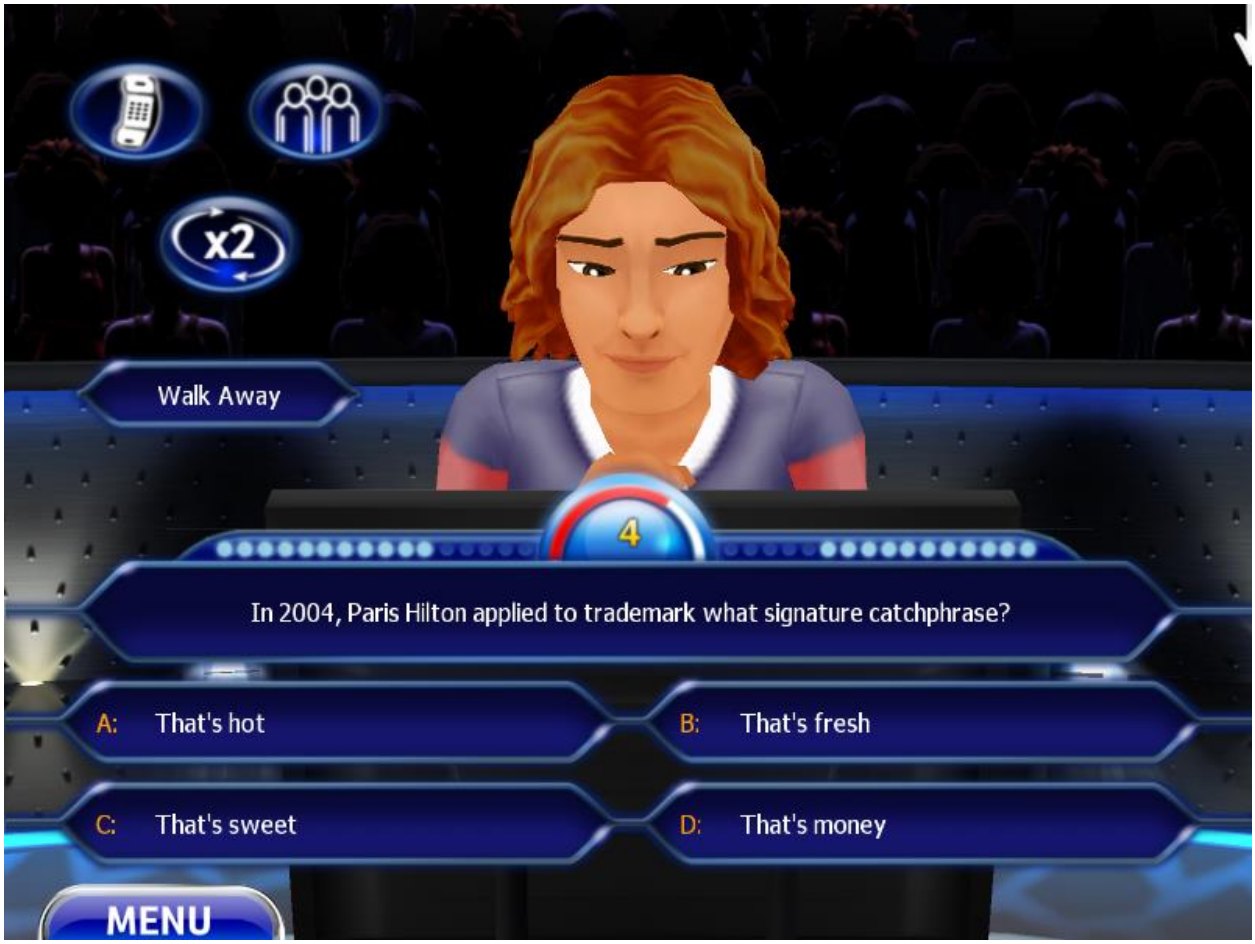

Рисунок 2.1 – Ludia’s Millionaire

2.2.2 Who Wants To Be A Millionaire от Appeal Studios

На данный момент, самая популярная версия игры. Особенности этой версии включают в себя: возможность отключить какие-либо темы либо добавить другие перед началом игры, “семейный режим” – задаются специальные вопросы для детей, онлайн-режим “королевская битва” – предоставляет возможность сыграть с 99 игроками на время [3].

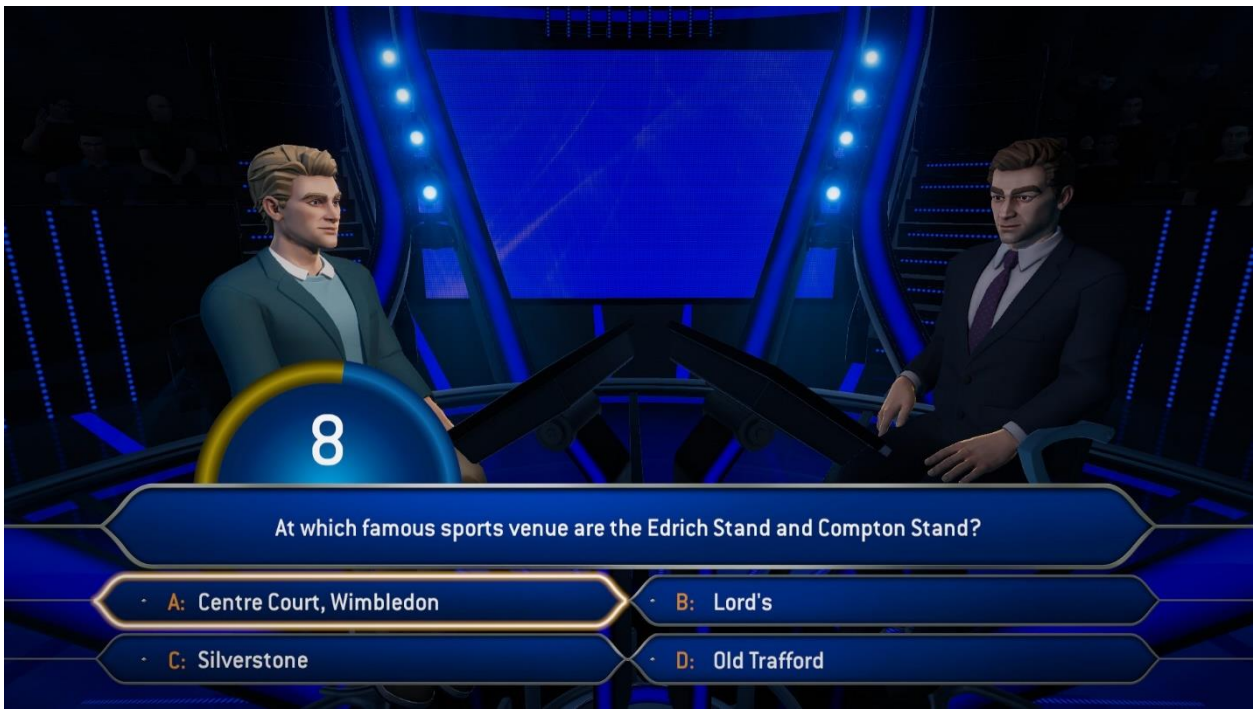

Рисунок 2.2 – Appeal Studios’ Millionaire

3 ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ

В данном разделе описываются входные и выходные данные программы, диаграмма классов, а также приводится описание используемых классов и их методов.

3.1 Структура входных и выходных данных

Таблица 3.1 – файлы с вопросами для игры q1.txt, q2.txt, q3.txt, q4.txt, q5.txt, q6.txt, q7.txt, q8.txt, q9.txt, q10.txt

Вопрос	Уровень	Ответ A	Ответ B	Ответ C	Ответ D	Правильный ответ
Назовите столицу Франции	1	Париж	Москва	Лондон	Берлин	A

Таблица 3.2 – файл с информацией о рекордах Records.txt

Имя игрока	Итоговый счёт
Pudge	420690

3.2 Разработка диаграммы классов

Диаграмма классов для данного курсового проекта представлена в приложении А.

3.3 Описание классов

3.3.1 Классы, управляющие игровым циклом

Класс `InputManager` обрабатывает событие ввода от пользователя.

Методы класса:

- `bool IsSpriteClicked(sf::Sprite object, sf::Mouse::Button button, sf::RenderWindow& window)` – метод, который обрабатывает нажатия на графические объекты;
- `bool IsTextClicked(const sf::Text& text, sf::Mouse::Button button, sf::RenderWindow& window)` – метод, который обрабатывает нажатия на текст;
- `sf::Vector2i GetMousePosition(sf::RenderWindow& window)` – метод, который возвращает позицию курсора мыши в виде двумерного вектора с целочисленными компонентами.

Класс `AssetManager` управляет ресурсами, такими как изображения, звуки, шрифты и другие файлы, используемые в игре.

Поля класса:

- `std::map<std::string, sf::Texture> _textures` – словарь, который хранит графические объекты, используемые в игре;
- `std::map<std::string, sf::Font> _fonts` – словарь, который хранит шрифты, используемые в игре.

Методы класса:

- `void LoadTexture(std::string name, std::string fileName)` – метод, загружающий изображение в словарь и присваивающий ему ключ;
- `sf::Texture& GetTexture(std::string name)` – метод, возвращающий графический объект из словаря по его ключу;
- `void LoadFont(std::string name, std::string fileName)` - метод, загружающий шрифт в словарь и присваивающий ему ключ;
- `sf::Font& GetFont(std::string name)` - метод, возвращающий шрифт из словаря по его ключу.

Класс `StateMachine` управляет состояниями экрана и игры. Класс основан на концепции автомата конечных состояний Мура.

Поля класса:

- `std::stack<StateRef> _states` – стек, который хранит указатели на состояния;
- `StateRef _newState` – указатель на новое состояние;
- `bool _isAdding` – флаг, указывающий на добавление состояния в стек;
- `bool _isRemoving` – флаг, указывающий на удаление состояния из стека;
- `bool _isReplacing` – флаг, указывающий на замену последнего состояния в стеке на текущее.

Методы класса:

- `void AddState(StateRef newState, bool isReplacing = true)` – метод, который вызывает операцию добавления;
- `void RemoveState()` – метод, который вызывает операцию удаления;
- `void ProcessStateChanges()` – метод выполнения либо операции добавления в стек, либо операции удаления из стека;
- `StateRef& StateMachine::GetActiveState()` – метод, который возвращает текущее состояние из стека.

Класс `Game` управляет основными аспектами игры: циклическим обновлением состояний, хранением ресурсов, выполнением операций.

Поля класса:

- `const float dt 1.0f / 90.f` – константа, которая обозначает временной период между обновлениями;
- `sf::Clock _clock` – счетчик времени, который служит для обеспечения равномерной частоты обновлений;
- `GameDataRef _data = std::make_shared<GameData>()` – умный указатель на контейнер, содержащий объекты других классов, которые управляют игровым циклом.

Методы класса:

- `Game(int width, int height, std::string title)` – конструктор, предназначенный для инициализации начальных параметров игры;
- `void Run()` – метод для запуска основного цикла выполнения игры.

3.3.2 Класс вопроса игры

Класс `Question` предназначен для хранения и загрузки вопроса и ответов на него из текстового файла.

Поля класса:

- `std::string task` – строка, которая хранит текст вопроса;
- `std::string answerA` – строка, которая хранит текст ответа A;
- `std::string answerB` – строка, которая хранит текст ответа B;
- `std::string answerC` – строка, которая хранит текст ответа C;
- `std::string answerD` – строка, которая хранит текст ответа D;
- `int rightAnswer` – переменная, которая хранит число, обозначающее правильный ответ на вопрос;
- `int difficulty = 0` – переменная, которая хранит уровень сложности вопроса.

Методы класса:

- `void getQuestion(int level)` – метод, который получает вопроса из файла в зависимости от уровня сложности.

3.3.3 Классы состояний игры

Класс `State` является абстрактный классом. Он предназначен для представления состояний в конечном автомате.

Методы класса:

- `virtual void Init() = 0` – чисто виртуальная функция, которая предназначена для инициализации параметров состояния;
- `virtual void HandleInput() = 0` – чисто виртуальная функция, которая предназначена для обработки ввода от пользователя;
- `virtual void Draw(float dt) = 0` – чисто виртуальная функция, которая предназначена для визуализации текущего состояния игры.
- `virtual void Update(float dt) = 0` – чисто виртуальная функция, которая предназначена для обновления текущего состояния игры.

От класса `State` наследуются классы, каждый из которых будет отвечать за свое конкретное состояние игры.

Класс `MainMenuState` отвечает за состояние главного меню. Является производным от класса `State`.

Поля класса:

- `GameDataRef _data` – умный указатель на контейнер, содержащий объекты других классов, которые управляют игровым циклом;

- `sf::Sprite _background` – графический объект, который хранит фоновое изображение окна;
- `sf::Sprite _gameLogo` – графический объект, который хранит логотип игры;
- `sf::Text _startText` – текст, при нажатии на который производится переход в состояние обучающего окна;
- `sf::Text _settingText` – текст, при нажатии на который производится переход в состояние настроек;
- `sf::Text _recordTableText` – текст, при нажатии на который производится переход в состояние таблицы рекордов;
- `sf::Text _exitText` – текст, при нажатии на который производится выход из программы;
- `sf::Music _mainMenuMusic` – объект, который предназначен для воспроизведения фоновой звуковой дорожки в состоянии главного меню;
- `int volume` – переменная, которая хранит число, обозначающее громкость фоновой звуковой дорожки.

Методы класса:

- `MainMenuState(GameDataRef data, int volume)` – конструктор, предназначенный для инициализации состояния главного меню;
- `void Init() override` – виртуальная функция, которая является переопределением метода из базового класса. Предназначена для инициализации начальных параметров состояния главного меню;
- `void HandleInput() override` – виртуальная функция, которая является переопределением метода из базового класса. Предназначена для обработки ввода от пользователя в состоянии главного меню;
- `void Draw(float dt) override` – виртуальная функция, которая является переопределением метода из базового класса. Предназначена для визуализации состояния главного меню;
- `void Update(float dt) override` – виртуальная функция, которая является переопределением метода из базового класса. Предназначена для обновления параметров состояния главного меню;
- `void SetMainMenuText(sf::Text& text, std::string name, int divSize)` – метод, который предназначен для установки свойств кликабельного текста в состоянии главного меню.

Класс `SettingState` отвечает за состояние меню настроек. Является производным от класса `State`.

Поля класса:

- `GameDataRef _data` – умный указатель на контейнер, содержащий объекты других классов, которые управляют игровым циклом;
- `sf::Sprite _background` – графический объект, который хранит фоновое изображение окна;
- `sf::Text _settingsText` – текст, показывающий что игра находится в состоянии меню настроек;

- `sf::Text _musicText` – текст, показывающий на строку изменения громкости звука игры;
- `sf::Text _backText` – текст, при нажатии на который происходит возврат в состояние главного меню;
- `sf::Text _plus` – текст, при нажатии на который происходит увеличение громкости звука игры;
- `sf::Text _minus` – текст, при нажатии на который происходит уменьшение громкости звука игры;
- `sf::Text _volume` – текст, показывающий текущую громкость звука игры;
- `sf::Music _settingsMusic` – объект, который предназначен для воспроизведения фоновой звуковой дорожки в состоянии меню настроек;
- `int volume` – переменная, которая хранит число, обозначающее громкость фоновой звуковой дорожки.

Методы класса:

- `SettingState(GameDataRef data, int volume)` – конструктор, предназначенный для инициализации состояния меню настроек;
- `void Init() override` – виртуальная функция, которая является переопределением метода из базового класса. Предназначена для инициализации начальных параметров состояния меню настроек;
- `void HandleInput() override` – виртуальная функция, которая является переопределением метода из базового класса. Предназначена для обработки ввода от пользователя в состоянии меню настроек;
- `void Draw(float dt) override` – виртуальная функция, которая является переопределением метода из базового класса. Предназначена для визуализации состояния меню настроек;
- `void Update(float dt) override` – виртуальная функция, которая является переопределением метода из базового класса. Предназначена для обновления параметров состояния меню настроек;
- `void SetSettingsText(sf::Text& text, std::string name, int divSize)` – метод, который предназначен для установки свойств кликабельного текста в состоянии меню настроек.

Класс `RecordTableState` отвечает за состояние таблицы рекордов. Является производным от класса `State`.

Поля класса:

- `GameDataRef _data` – умный указатель на контейнер, содержащий объекты других классов, которые управляют игровым циклом;
- `sf::Sprite _background` – графический объект, который хранит фоновое изображение окна;
- `sf::Text _titleText` – текст, показывающий что игра находится в состоянии таблицы рекордов;
- `sf::Text _musicText` – текст, показывающий на строку изменения громкости звука игры;

- `sf::Text _backText` – текст, при нажатии на который происходит возврат в состояние главного меню;
- `std::vector<sf::Text> _names` – вектор, содержащий объекты текста, который выводится на экран как имя игрока;
- `std::vector<sf::Text> _score` – вектор, содержащий объекты текста, который выводится на экран как счет игрока;
- `std::vector<RecordUnit> _allRecords` – вектор, содержащий контейнеры с именем и счетом игроков, которые загружаются из файла;
- `sf::Music _rtMusic` – объект, который предназначен для воспроизведения фоновой звуковой дорожки в состоянии таблицы рекордов;
- `int volume` – переменная, которая хранит число, обозначающее громкость фоновой звуковой дорожки.

Методы класса:

- `RecordTableState(GameDataRef data, int volume)` – конструктор, предназначенный для инициализации состояния таблицы рекордов;
- `void Init() override` – виртуальная функция, которая является переопределением метода из базового класса. Предназначена для инициализации начальных параметров состояния таблицы рекордов;
- `void HandleInput() override` – виртуальная функция, которая является переопределением метода из базового класса. Предназначена для обработки ввода от пользователя в состоянии таблицы рекордов;
- `void Draw(float dt) override` – виртуальная функция, которая является переопределением метода из базового класса. Предназначена для визуализации состояния таблицы рекордов;
- `void Update(float dt) override` – виртуальная функция, которая является переопределением метода из базового класса. Предназначена для обновления параметров состояния таблицы рекордов;
- `void SetRecordTableText(sf::Text& text, std::string name, int size, std::string font, int thickness)` – метод, который предназначен для установки свойств кликабельного текста в состоянии таблицы рекордов.
- `void LoadRecords(std::string filename)` – метод, который загружает информацию о рекордах из файла в контейнер.

Класс `TutorialState` отвечает за состояние обучения правилам игры. Является производным от класса `State`.

Поля класса:

- `GameDataRef _data` – умный указатель на контейнер, содержащий объекты других классов, которые управляют игровым циклом;
- `sf::Sprite _background` – графический объект, который хранит фоновое изображение окна;
- `sf::Text _infoText` – текст, показывающий информацию о правилах игры;

- `sf::RectangleShape _info` – объект прямоугольника, в котором выводится информация о правилах игры;
- `sf::Music _tutorialMusic` – объект, который предназначен для воспроизведения фоновой звуковой дорожки в состоянии обучения правилам игры;
- `int volume` – переменная, которая хранит число, обозначающее громкость фоновой звуковой дорожки.

Методы класса:

- `TutorialState(GameDataRef data, int volume)` – конструктор, предназначенный для инициализации состояния обучения правилам игры;
- `void Init() override` – виртуальная функция, которая является переопределением метода из базового класса. Предназначена для инициализации начальных параметров состояния обучения правилам игры;
- `void HandleInput() override` – виртуальная функция, которая является переопределением метода из базового класса. Предназначена для обработки ввода от пользователя в состоянии обучения правилам игры;
- `void Draw(float dt) override` – виртуальная функция, которая является переопределением метода из базового класса. Предназначена для визуализации состояния обучения правилам игры;
- `void Update(float dt) override` – виртуальная функция, которая является переопределением метода из базового класса. Предназначена для обновления параметров состояния обучения правилам игры.

Класс `QuestionState` отвечает за состояние игрового уровня. Является производным от класса `State`.

Поля класса:

- `GameDataRef _data` – умный указатель на контейнер, содержащий объекты других классов, которые управляют игровым циклом;
- `sf::Sprite _background` – графический объект, который хранит фоновое изображение окна;
- `sf::Text _infoText` – текст, показывающий информацию о правилах игры;
- `sf::RectangleShape _info` – объект прямоугольника, в котором выводится информация о правилах игры;
- `sf::Music _tutorialMusic` – объект, который предназначен для воспроизведения фоновой звуковой дорожки в состоянии обучения правилам игры;
- `int volume` – переменная, которая хранит число, обозначающее громкость фоновой звуковой дорожки;
- `sf::Sprite _buttonA` – графический объект, при нажатии на который происходит проверка правильности ответа A;

- `sf::Sprite _buttonB` – графический объект, при нажатии на который происходит проверка правильности ответа B;
- `sf::Sprite _buttonC` – графический объект, при нажатии на который происходит проверка правильности ответа C;
- `sf::Sprite _buttonD` – графический объект, при нажатии на который происходит проверка правильности ответа D;
- `sf::Sprite _skip` – графический объект, при нажатии на который активируется бонус “Пропуск вопроса”;
- `sf::Sprite _mistake` – графический объект, при нажатии на который активируется бонус “Право на ошибку”;
- `sf::Sprite _change` – графический объект, при нажатии на который активируется бонус “Замена вопроса”;
- `bool _twoChances = false` – флаг активации бонуса “Право на ошибку”;
- `Lifeline availableBonuses` – контейнер, показывающий наличие бонусов у игрока;
- `sf::Time _timer` – переменная, хранящая оставшееся время на ответ в секундах;
- `sf::CircleShape _timerFrame` – объект круговой формы, показывающий оставшееся время на ответ;
- `sf::Clock _timerClock` – счетчик времени, предназначенный для ежесекундного обновления оставшегося времени на ответ;
- `sf::Text _timeRemaining` – текст, показывающий количество оставшегося времени на ответ игрока;
- `sf::Text _task` – текстовый объект, показывающий текст вопроса;
- `sf::Text _answerA` – текстовый объект, показывающий текст ответа A;
- `sf::Text _answerB` – текстовый объект, показывающий текст ответа B;
- `sf::Text _answerC` – текстовый объект, показывающий текст ответа C;
- `sf::Text _answerD` – текстовый объект, показывающий текст ответа D;
- `sf::Text _progress` – текстовый объект, показывающий номер текущего уровня;
- `sf::Text _score` – текстовый объект, показывающий текущий счет игрока;
- `Question currentQuestion` – контейнер, который хранит текст текущего вопроса, тексты ответов и номер правильного ответа;
- `int currentLevel` – переменная, которая хранит номер текущего уровня;
- `int currentScore = 0` – переменная, которая хранит текущий счет игрока.

Методы класса:

- `QuestionState(GameDataRef data, int volume, int level, int score, sf::Time timer, Lifeline hints)` – конструктор, предназначенный для инициализации состояния игрового уровня;
- `void Init() override` – виртуальная функция, которая является переопределением метода из базового класса. Предназначена для инициализации начальных параметров состояния игрового уровня;
- `void HandleInput() override` – виртуальная функция, которая является переопределением метода из базового класса. Предназначена для обработки ввода от пользователя в состоянии игрового уровня;
- `void Draw(float dt) override` – виртуальная функция, которая является переопределением метода из базового класса. Предназначена для визуализации состояния игрового уровня;
- `void Update(float dt) override` – виртуальная функция, которая является переопределением метода из базового класса. Предназначена для обновления параметров состояния игрового уровня;
- `void SetQuestionText(sf::Text& text, std::wstring name)` – метод, который предназначен для установки свойств текста в состоянии игрового уровня.

Класс `GameOverState` отвечает за состояние поражения. Является производным от класса `State`.

Поля класса:

- `GameDataRef _data` – умный указатель на контейнер, содержащий объекты других классов, которые управляют игровым циклом;
- `sf::Sprite _background` – графический объект, который хранит фоновое изображение окна;
- `sf::SoundBuffer _gameOverSoundBuffer` – объект, который предназначен для хранения звукового эффекта поражения;
- `sf::Sound _gameOverSound` – объект, который предназначен для воспроизведения звукового эффекта поражения;;
- `sf::Text _loss` – текстовый объект, который выводит на экран сообщение о поражении;
- `int volume` – переменная, которая хранит число, обозначающее громкость фоновой звуковой дорожки.

Методы класса:

- `GameOverState(GameDataRef data, int volume)` – конструктор, предназначенный для инициализации состояния поражения;
- `void Init() override` – виртуальная функция, которая является переопределением метода из базового класса. Предназначена для инициализации начальных параметров состояния поражения;
- `void HandleInput() override` – виртуальная функция, которая является переопределением метода из базового класса. Предназначена для обработки ввода от пользователя в состоянии поражения;

- `void Draw(float dt) override` – виртуальная функция, которая является переопределением метода из базового класса. Предназначена для визуализации состояния поражения;
- `void Update(float dt) override` – виртуальная функция, которая является переопределением метода из базового класса. Предназначена для обновления параметров состояния поражения.

Класс `VictoryState` отвечает за состояние победы. Является производным от класса `State`.

Поля класса:

- `GameDataRef _data` – умный указатель на контейнер, содержащий объекты других классов, которые управляют игровым циклом;
- `sf::Sprite _background` – графический объект, который хранит фоновое изображение окна;
- `sf::SoundBuffer _victorySoundBuffer` – объект, который предназначен для хранения звукового эффекта победы;
- `sf::Sound _victorySound` – объект, который предназначен для воспроизведения звукового эффекта победы;
- `sf::Text _scoreText` – текстовый объект, который выводит на экран итоговый счет игрока;
- `sf::Text _inputText` – текстовое окно, в которое вводится имя игрока;
- `std::string playerName` – строка, которая хранит имя игрока;
- `int _score` – переменная, которая хранит итоговый счет игрока;
- `std::vector<RecordUnit> _allRecords` – вектор, содержащий контейнеры с именем и счетом игроков, которые загружаются из файла;
- `int volume` – переменная, которая хранит число, обозначающее громкость фоновой звуковой дорожки.

Методы класса:

- `VictoryState(GameDataRef data, int volume, int score)` – конструктор, предназначенный для инициализации состояния победы;
- `void Init() override` – виртуальная функция, которая является переопределением метода из базового класса. Предназначена для инициализации начальных параметров состояния победы;
- `void HandleInput() override` – виртуальная функция, которая является переопределением метода из базового класса. Предназначена для обработки ввода от пользователя в состоянии победы;
- `void Draw(float dt) override` – виртуальная функция, которая является переопределением метода из базового класса. Предназначена для визуализации состояния победы;
- `void Update(float dt) override` – виртуальная функция, которая является переопределением метода из базового класса. Предназначена для обновления параметров состояния победы;

- `void getRecords(std::string filename)` — метод, который загружает информацию о рекордах из файла в контейнер;
- `void updateRecords(std::string playerName, int playerScore, std::string filename)` — метод, который добавляет новый результат в контейнер и сохраняет его в файл.

4 РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ

4.1 Разработка алгоритмов

`void Game::Run()` – метод для запуска основного цикла выполнения игры.

Алгоритм по шагам:

1. Начало.
2. Получить текущее время в секундах.
3. Если окно приложения закрыто, перейти к шагу X.
4. Обработать изменения текущего состояния игры.
5. Получить новое текущее время в секундах.
6. Найти время, прошедшее с предыдущего кадра.
7. Если время кадра меньше либо равно 0.25 секунды, перейти к шагу 9.
8. Установить время кадра, равное 0.25 секунды.
9. Прошлому значению текущего времени присвоить новое значение текущего времени.
10. Прибавить к накопителю время кадра.
11. Если накопитель меньше времени обновления, перейти к шагу 16.
12. Запустить процесс обработки ввода пользователя в текущем состоянии.
13. Обновить параметры текущего состояния.
14. Вычесть из накопителя время обновления.
15. Если накопитель больше или равен времени обновления, перейти к шагу 12.
16. Визуализировать текущее состояние игры.
17. Если окно приложения открыто, перейти к шагу 4.
18. Конец.

`void StateMachine::ProcessStateChanges()` – метод выполнения либо операции добавления в стек, либо операции удаления из стека.

Алгоритм по шагам:

1. Начало.
2. Если не установлен флаг удаления либо стек пуст, перейти к шагу 5.
3. Удалить верхнее состояние из стека.
4. Сбросить флаг удаления.
5. Если не установлен флаг добавления, перейти к шагу 11.
6. Если не установлен флаг замены либо стек пуст, перейти к шагу 8.
7. Удалить верхнее состояние из стека.
8. Добавить новое состояние в стек.
9. Инициализировать верхнее состояние из стека.
10. Сбросить флаг добавления.
11. Конец.

4.2 Разработка схем алгоритмов

Схема алгоритма `void Game::Run()` представлена в приложении Б. Данный метод вызывается при запуске программы. Он представляет собой цикл игры, в котором обрабатывается ввод пользователя, обновляется состояние игры и визуализируется текущее состояние. Данный алгоритм выполняется без остановки с момента запуска приложения до его закрытия. Он вычисляет временной интервал между кадрами, обрабатывает изменения в состояниях каждый такой временной интервал и предотвращает слишком сильное повышение временного интервала между кадрами, и, соответственно, скачки в игровой логике, стабилизируя обновление состояний игры.

Схема алгоритма `void StateMachine::ProcessStateChanges()` представлена в приложении В. Данный метод выполняет либо добавление состояния в стек, либо удаление состояния из стека – в зависимости от того, какой флаг установлен. Также он обрабатывает тип добавления состояния в стек: с заменой текущего состояния либо без замены. Данный алгоритм выполняется во время работы программы постоянно, с временным интервалом равным разности интерполяции между кадрами и интервалом обновления экрана.

5 РЕЗУЛЬТАТЫ РАБОТЫ

При запуске программы пользователя приветствует окно с главным меню, состоящим из кнопок: “Start Game” (начать игру), “Settings” (настройки), “Record Table” (таблица рекордов) и “Exit” (выйти). Окно главного меню показано на рисунке 5.1.

Рисунок 5.1 – Окно главного меню

При нажатии на кнопку “Settings”, пользователь переходит в меню настроек, где предоставляется возможность отрегулировать громкость звука игры. Меню настроек показано на рисунке 5.2.

При нажатии на кнопку “Record Table”, пользователь переходит в таблицу рекордов, где показаны 7 самых высоких результатов за все прошедшие игровые сессии. Таблица рекордов показана на рисунке 5.3.

При нажатии на кнопку “Start Game”, пользователь начинает игровую сессию и на экран выводятся краткие правила игры. Правила игры показаны на рисунке 5.4.

При нажатии на прямоугольный фон правил игры на экран выводится игровой интерфейс. Он содержит: текст вопроса, текст вариантов ответа A, B, C и D, номер текущего уровня, кнопки подсказок: (слева направо) “Право на ошибку”, “Сменить вопрос”, “Пропустить вопрос”, также на экран выводится текущий счет пользователя и таймер с обратным отсчетом оставшегося времени на ответ пользователя. Игровой интерфейс показан на рисунке 5.5.

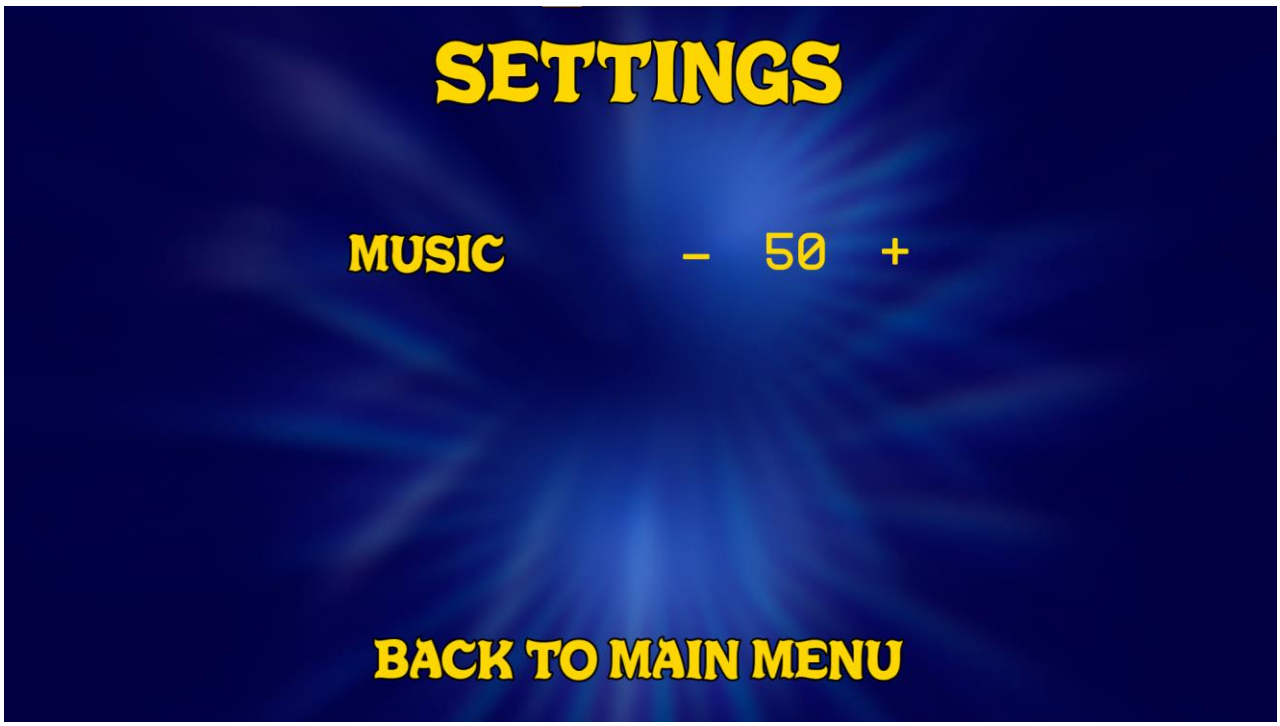

Рисунок 5.2 – Меню настроек

Рисунок 5.3 – Таблица рекордов

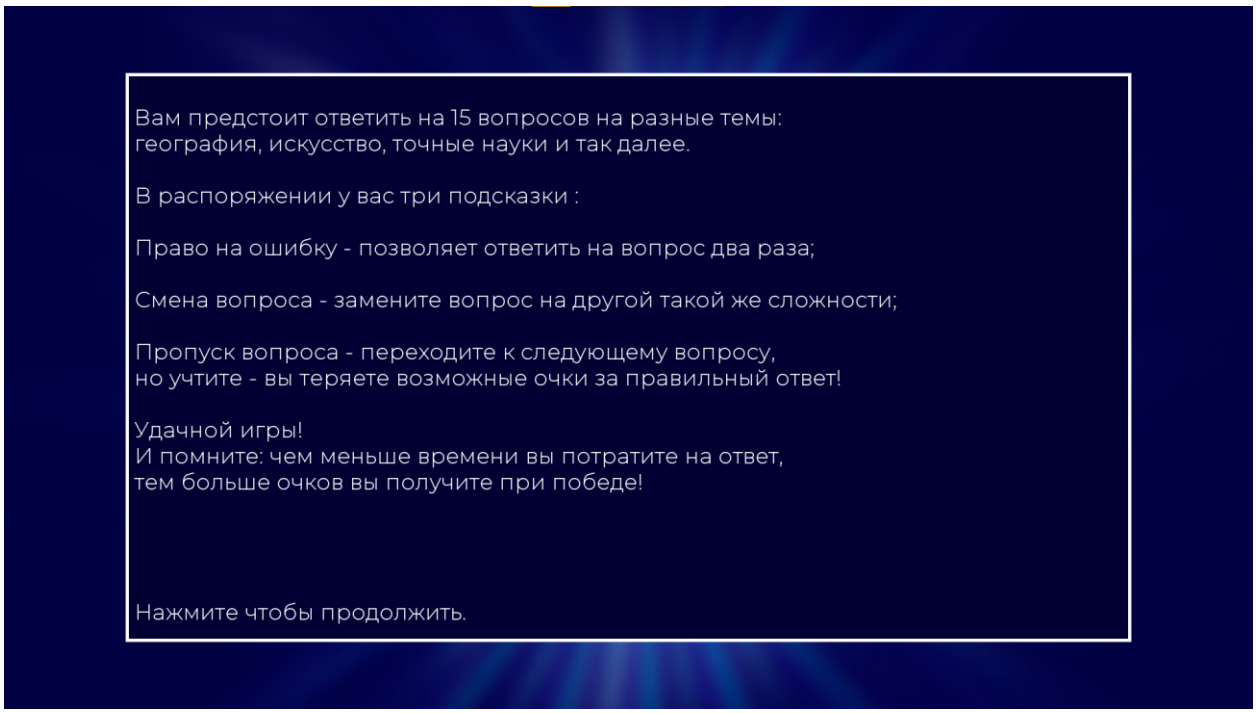

Рисунок 5.4 – Правила игры

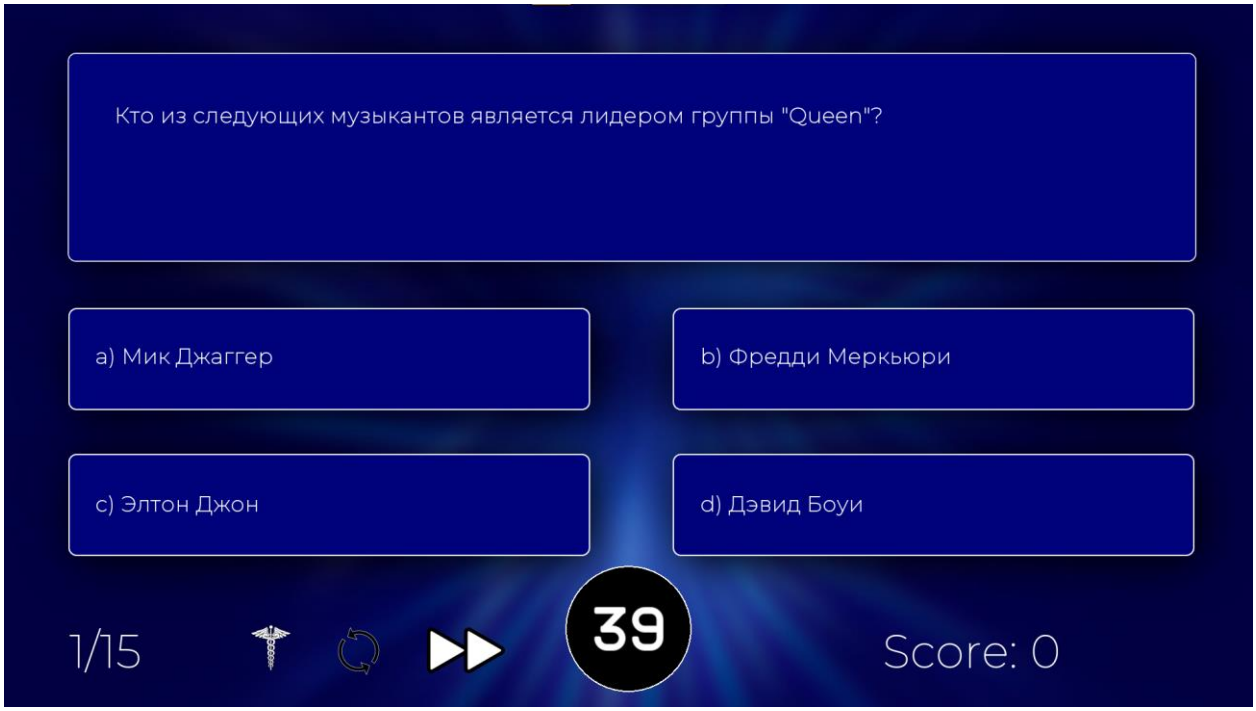

Рисунок 5.5 – Игровой интерфейс

В случае если пользователь даст неправильный ответ на вопрос, либо истечет таймер ответа, игра переходит в состояние поражения. На экран выводится короткое сообщение “Your answer was wrong.” (с англ. “Твой ответ был неправильным.”). Состояние поражения показано на рисунке 5.6.

В случае если пользователь даст верный ответ на вопрос, игра переходит в состояние следующего вопроса. Если пользователь даст верный ответ на пятнадцатый вопрос, игра переходит в состояние победы. На экран выводится сообщение “Victory!” (с англ. “Победа!”), выводится итоговый счет пользователя и появляется окно ввода текста, куда пользователь должен ввести свое имя или псевдоним. После ввода имени и нажатия клавиши Enter игра сравнивает результаты пользователя с таблицей рекордов, и если его результат выше чем какой-либо из таблицы, то он попадет в таблицу и будет записан в файл для хранения. Состояние победы показано на рисунке 5.7.

При нажатии на кнопку подсказки “Право на ошибку” (обозначается значком “кадуцей”) активируется флаг этого бонуса и даже если пользователь даст неправильный ответ, игра не перейдет в состояние поражения. Если со второй попытки пользователь дает неправильный ответ, игра заканчивается. Если он дает правильный ответ, игра переходит в состояние следующего вопроса.

При нажатии на кнопку подсказки “Сменить вопрос” (обозначается стрелками, образующими круг) игра заново выбирает случайном образом файл темы и заново запускает текущий уровень.

При нажатии на кнопку подсказки “Пропустить вопрос” (обозначается значком перемотки) игра сразу же переходит в состояние следующего вопроса, без необходимости ответа на текущий вопрос.

Каждую подсказку можно использовать один раз за игровую сессию, и после использования их значки становятся полупрозрачными.

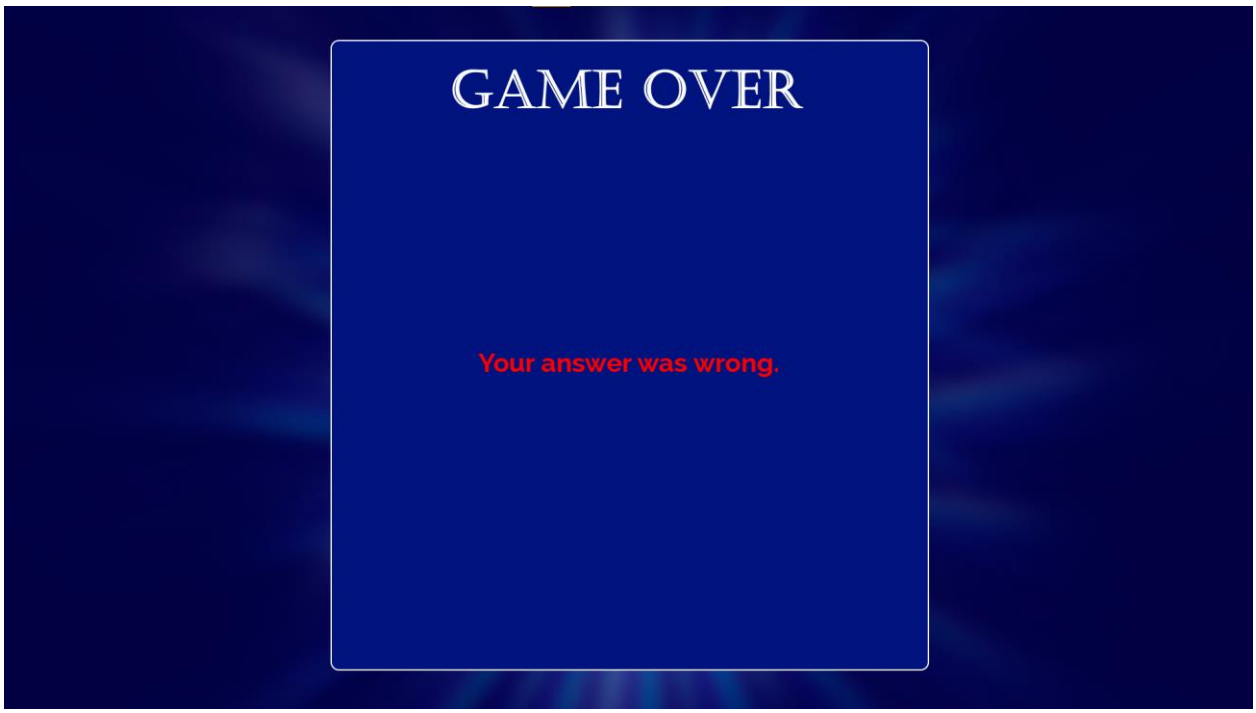

Рисунок 5.6 – Состояние поражения

Рисунок 5.7 – Состояние победы

ЗАКЛЮЧЕНИЕ

В ходе выполнения данной курсовой работы было создана игра “Кто хочет стать миллионером”. Графический интерфейс игры был создан с помощью библиотеки SFML. Игра была создана на ОС Windows 10 в интегрированной среде разработки Visual Studio 2022 – классический инструмент для создания приложений различного типа на языке программирования C++.

Было изучено: принцип работы игровых циклов, возможности применения автомата конечных состояний в разработке игры. Эти концепции стали ключевыми в создании структурированного и эффективного кода, управляющего логикой игры.

Также были изучены классы и контейнеры SFML, позволяющие работать с различными типами файлов: изображениями, текстовыми, аудио- и видеофайлами.

SFML является отличным инструментом для разработки небольших игр, так как имеет различные собственные классы, такие `sf::Color` для работы с цветовой палитрой, `sf::Sound` и `sf::Music` для работы со звуком, `sf::Clock` для работы с временем.

Код программы представлен в приложении Г.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] SFML Tutorials [Электронный ресурс]. -Электронные данные. -Режим доступа: https://www.sfml-dev.org/tutorials/2.6/ - Дата доступа: 12.12.2023

[2] Internet Archive: Who Wants to be a Millionaire by Ludia [Электронный ресурс]. – Электронные данные. – Режим доступа: https://archive.org/details/millionaire-2010 – Дата доступа: 12.12.2023

[3] Steam: Who Wants to be a Millionaire [Электронный ресурс]. – Электронные данные. – Режим доступа: https://store.steampowered.com/app/1356240/ – Дата доступа: 12.12.2023

ПРИЛОЖЕНИЕ А
(Обязательно)
Диаграмма классов

ПРИЛОЖЕНИЕ Б

(Обязательное)

Схема метода void Game::Run()

ПРИЛОЖЕНИЕ В

(Обязательное)

Схема метода void StateMachine::ProcessStateChanges()

ПРИЛОЖЕНИЕ Г
(Обязательное)
Полный код программы

Файл AssetManager.cpp:
```
#include "AssetManager.h"
namespace TheGame
{
	void AssetManager::LoadTexture(std::string name, std::string fileName)
	{
		sf::Texture texture;
		if (texture.loadFromFile(fileName)) {
			this->_textures[name] = texture;
		}
	}
	sf::Texture& AssetManager::GetTexture(std::string name)
	{
		return this->_textures.at(name);
	}
	void AssetManager::LoadFont(std::string name, std::string fileName)
	{
		sf::Font font;
		if (font.loadFromFile(fileName)) {
			this->_fonts[name] = font;
		}
	}
	sf::Font& AssetManager::GetFont(std::string name)
	{
		return this->_fonts.at(name);
	}
}
```

Файл AssetManager.h:
```
#pragma once
#include <map>
#include "SFML/Graphics.hpp"
namespace TheGame
{
	class AssetManager {
	public:
		void LoadTexture(std::string name, std::string fileName);
		sf::Texture& GetTexture(std::string name);
		void LoadFont(std::string name, std::string fileName);
		sf::Font& GetFont(std::string name);
	private:
		std::map<std::string, sf::Texture> _textures;
		std::map<std::string, sf::Font> _fonts;
	};
}
```

Файл DEFINITIONS.h:
```
#pragma once
#define SCREEN_WIDTH 1920
#define SCREEN_HEIGHT 1080
#define SPLASH_STATE_SHOW_TIME 1.5
#define GAMESTART_SHOW_TIME 0.3
#define QUESTION_TIMER 45
#define NAME_LENGTH_LIMIT 10
#define RECORD_ENTRIES_LIMIT 7
#define FONT_FILEPATH "Font/thefont.ttf"
```

```
#define INGAME_FONT_FILEPATH "Font/muhammadsumbul.ttf"
#define BOLD_RECORD_FILEPATH "Font/ralewaybold.ttf"
#define TIMER_FONT_FILEPATH "Font/telegrama_render.otf"
#define INTRO_MUSIC_FILEPATH "Audio/intro.mp3"
#define MAIN_MENU_MUSIC_FILEPATH "Audio/mainmenu2.ogg"
#define RECORDTABLE_MUSIC_FILEPATH "Audio/rt.ogg"
#define TUTORIAL_MUSIC_FILEPATH "Audio/explain.ogg"
#define QUESTION_MUSIC_FILEPATH "Audio/quiz.ogg"
#define WRONG_ANSWER_SFX "Audio/wrong.mp3"
#define VICTORY_SFX "Audio/victory.mp3"
#define SESSION_INTRO_SFX "Audio/tboiintro.mp3"
#define MAIN_MENU_BACKGROUND_FILEPATH "Img/bg.png"
#define QUESTION_BACKGROUND_FILEPATH "Img/bg2.png"
#define GAME_OVER_BACKGROUND_FILEPATH "Img/bgover.png"
#define VICTORY_BACKGROUND_FILEPATH "Img/bgvictory.png"
#define RT_BACKGROUND_FILEPATH "Img/rtwide2.png"
#define IF_RIGHT_A "Img/answerbg/if_right_A.png"
#define IF_RIGHT_B "Img/answerbg/if_right_B.png"
#define IF_RIGHT_C "Img/answerbg/if_right_C.png"
#define IF_RIGHT_D "Img/answerbg/if_right_D.png"
#define IF_WRONG_A "Img/answerbg/if_wrong_A.png"
#define IF_WRONG_B "Img/answerbg/if_wrong_B.png"
#define IF_WRONG_C "Img/answerbg/if_wrong_C.png"
#define IF_WRONG_D "Img/answerbg/if_wrong_D.png"
#define CHANGE_QUESTION_ICON_FILEPATH "Img/change70.png"
#define SKIP_QUESTION_ICON_FILEPATH "Img/skip70.png"
#define SECOND_CHANCE_ICON_FILEPATH "Img/caduceus70.png"
#define GAME_TITLE_FILEPATH "Img/logo2.png"
#define PLAY_BUTTON_FILEPATH "Img/start.png"
#define EXIT_BUTTON_FILEPATH "Img/exit.png"
#define SETTING_BUTTON_FILEPATH "Img/setting.png"
#define RECORDS_FILE "Misc/Records.txt"
#define TOPIC_FILENAME "Questions/q"
#define TOPIC_EXTENSION ".txt"
#define FULL_PROGRESS "/15"
#define _SCORE_ "Score: "
```

Файл Game.cpp:

```
#include "Game.h"
#include "MainMenuState.h"

namespace TheGame
{
	Game::Game(int width, int height, std::string title)
	{
		_data->window.create(sf::VideoMode(width, height), title,
sf::Style::Close | sf::Style::Titlebar | sf::Style::Fullscreen);
		_data->machine.AddState(StateRef(new MainMenuState(_data, 50)),
true);
		this->Run();
	}

	void Game::Run()
	{
		float newTime, frameTime;
		float currentTime = this->_clock.getElapsedTime().asSeconds();
		float accumulator = 0.0f;
		while (this->_data->window.isOpen()) {
			this->_data->machine.ProcessStateChanges();
			newTime = this->_clock.getElapsedTime().asSeconds();
			frameTime = newTime - currentTime;
			if (frameTime > 0.25f)
				frameTime = 0.25f;
```

```
                currentTime = newTime;
                accumulator += frameTime;
                while (accumulator >= dt) {
                        this->_data->machine.GetActiveState()->HandleInput();
                        this->_data->machine.GetActiveState()->Update(dt);
                        accumulator -= dt;
                }
                this->_data->machine.GetActiveState()->Draw(dt);
        }
    }
}
```

Файл Game.h:

```
#pragma once
#include <memory>
#include <string>
#include <SFML/Graphics.hpp>
#include "StateMachine.h"
#include "AssetManager.h"
#include "InputManager.h"
namespace TheGame
{
    struct GameData {
        StateMachine machine;
        sf::RenderWindow window;
        AssetManager assets;
        InputManager input;
    };
    typedef std::shared_ptr<GameData> GameDataRef;
    class Game {
    public:
        Game(int width, int height, std::string title);
    private:
        const float dt = 1.0f / 90.0f;
        sf::Clock _clock;
        GameDataRef _data = std::make_shared<GameData>();
        void Run();
    };
}
```

Файл GameOverState.cpp:

```
#include "GameOverState.h"
#include "DEFINITIONS.h"
#include "MainMenuState.h"
#include <sstream>
#include <iostream>
namespace TheGame
{
    GameOverState::GameOverState(GameDataRef data, int volume) :
_data(data)
    {
        this->volume = volume;
    }

    void GameOverState::Init()
    {
        this->_data->assets.LoadTexture("Game Over Background",
GAME_OVER_BACKGROUND_FILEPATH);
        this->_data->assets.LoadFont("Font", BOLD_RECORD_FILEPATH);
        _background.setTexture(this->_data->assets.GetTexture("Game Over
Background"));
        if (!_gameOverSoundBuffer.loadFromFile(WRONG_ANSWER_SFX))
```

```
                  std::cout << "error\n";
            _loss.setString("Your answer was wrong.");
            _loss.setFont(this->_data->assets.GetFont("Font"));
            _loss.setCharacterSize(40);
            _loss.setFillColor(sf::Color::Red);
            _loss.setPosition((SCREEN_WIDTH / 2) -
(_loss.getGlobalBounds().width / 2),
                  (SCREEN_HEIGHT / 2) - _loss.getGlobalBounds().height / 2);
            _gameOverSound.setBuffer(_gameOverSoundBuffer);
            _gameOverSound.setVolume(this->volume);
            _gameOverSound.play();
      }
      void GameOverState::HandleInput()
      {
            sf::Event event;
            while (this->_data->window.pollEvent(event)) {
                  if (sf::Event::Closed == event.type)
                        this->_data->window.close();
                  if (sf::Keyboard::isKeyPressed(sf::Keyboard::Enter) ||
_data->input.IsSpriteClicked(_background, sf::Mouse::Left, _data->window)) {
                        //_tutorialSound.stop();
                        _data->machine.AddState(StateRef(new
MainMenuState(_data, this->volume)), true);
                  }
            }
      }

      void GameOverState::Update(float dt)
      {
      }

      void GameOverState::Draw(float dt)
      {
            this->_data->window.clear(sf::Color::Red);
            this->_data->window.draw(this->_background);
            this->_data->window.draw(_loss);
            this->_data->window.display();
      }
}
```

Файл GameOverState.h:

```
#pragma once
#include <SFML/Graphics.hpp>
#include <SFML/Audio.hpp>
#include "State.h"
#include "Game.h"
namespace TheGame
{
      class GameOverState : public State {
      public:
            GameOverState(GameDataRef data, int volume);
            void Init() override;
            void HandleInput() override;
            void Update(float dt) override;
            void Draw(float dt) override;
      private:
            GameDataRef _data;
            sf::Sprite _background;
            sf::SoundBuffer _gameOverSoundBuffer;
            sf::Sound _gameOverSound;
            sf::Text _loss;
            int volume;
      };
```

```
}
```

Файл InputManager.cpp:

```
#include "InputManager.h"
namespace TheGame
{
	sf::Vector2i InputManager::GetMousePosition(sf::RenderWindow& window)
	{
		return sf::Mouse::getPosition(window);
	}
	bool InputManager::IsSpriteClicked(sf::Sprite object, sf::Mouse::Button
button, sf::RenderWindow& window)
	{
		if (sf::Mouse::isButtonPressed(button)) {
			sf::IntRect playButtonRect(object.getPosition().x,
object.getPosition().y, object.getGlobalBounds().width,
object.getGlobalBounds().height);
			if
(playButtonRect.contains(sf::Mouse::getPosition(window)))
				return true;
		}
		return false;
	}
	bool InputManager::IsTextClicked(const sf::Text& text,
sf::Mouse::Button button, sf::RenderWindow& window)
	{
		if (sf::Mouse::isButtonPressed(button)) {
			sf::FloatRect textBounds = text.getGlobalBounds();
			sf::Vector2f textPosition = text.getPosition();
			sf::Vector2i mousePosition =
sf::Mouse::getPosition(window);
			sf::Vector2f translatedMousePos =
window.mapPixelToCoords(mousePosition);
			if (textBounds.contains(translatedMousePos))
				return true;
		}
		return false;
	}
}
```

Файл InputManager.h:

```
#pragma once
#include "SFML/Graphics.hpp"
namespace TheGame
{
	class InputManager {
	public:
		bool IsSpriteClicked(sf::Sprite object, sf::Mouse::Button button,
sf::RenderWindow& window);
		bool IsTextClicked(const sf::Text& text, sf::Mouse::Button
button, sf::RenderWindow& window);
		sf::Vector2i GetMousePosition(sf::RenderWindow& window);
	};
}
```

Файл main.cpp:

```
#include <SFML/Graphics.hpp>
#include "Game.h"
#include "DEFINITIONS.h"
#include <iostream>
int main()
{
```

```
    TheGame::Game session(SCREEN_WIDTH, SCREEN_HEIGHT, "Who Wants to be a
Millionaire") ;
    return 0;
}
```

Файл MainMenuState.cpp:

```
#include <sstream>
#include "MainMenuState.h"
#include "TutorialState.h"
#include "SettingState.h"
#include "RecordTableState.h"
#include "DEFINITIONS.h"

#include <iostream>

namespace TheGame
{
      void MainMenuState::SetMainMenuText(sf::Text& text, std::string name,
int divSize)//div size == 10 to text, 6 to name
      {
            text.setFont(this->_data->assets.GetFont("The Font"));
            text.setString(name);
            text.setCharacterSize(SCREEN_HEIGHT / divSize);
            sf::Color textColor(0xFF, 0xD7, 0x00);
            text.setFillColor(textColor);
      }

      MainMenuState::MainMenuState(GameDataRef data, int volume) :
_data(data)
      {
            this->volume = volume;
      }

      void MainMenuState::Init()
      {
            this->_data->assets.LoadFont("The Font", FONT_FILEPATH);
            this->_data->assets.LoadTexture("Main Menu Background",
MAIN_MENU_BACKGROUND_FILEPATH);
            this->_data->assets.LoadTexture("Game Title",
GAME_TITLE_FILEPATH);


            _background.setTexture(this->_data->assets.GetTexture("Main Menu
Background"));
            _gameLogo.setTexture(this->_data->assets.GetTexture("Game
Title"));

            _background.setPosition(0, 0);
            _gameLogo.setPosition(735, 30);

            SetMainMenuText(_startText, "START", 12);
            _startText.setOutlineThickness(3);
            SetMainMenuText(_settingText, "SETTINGS", 12);
            _settingText.setOutlineThickness(3);
            SetMainMenuText(_recordTableText, "RECORD TABLE", 12);
            _recordTableText.setOutlineThickness(3);
            SetMainMenuText(_exitText, "EXIT", 12);
            _exitText.setOutlineThickness(3);

            _startText.setPosition((SCREEN_WIDTH / 2) -
(_startText.getGlobalBounds().width / 2),
            (SCREEN_HEIGHT / 2) - _startText.getGlobalBounds().height / 2);
```

```
            _settingText.setPosition((SCREEN_WIDTH / 2) -
(_settingText.getGlobalBounds().width / 2),
                  (SCREEN_HEIGHT / 2) + (SCREEN_HEIGHT / 10) -
_settingText.getGlobalBounds().height / 2);

            _recordTableText.setPosition((SCREEN_WIDTH / 2) -
(_recordTableText.getGlobalBounds().width / 2),
                  (SCREEN_HEIGHT / 2) + (SCREEN_HEIGHT / 10 * 2) -
_recordTableText.getGlobalBounds().height / 2);

            _exitText.setPosition((SCREEN_WIDTH / 2) -
(_exitText.getGlobalBounds().width / 2),
            (SCREEN_HEIGHT / 2) + (SCREEN_HEIGHT / 10 * 3) -
_exitText.getGlobalBounds().height / 2);

            if (!_mainMenuMusic.openFromFile(MAIN_MENU_MUSIC_FILEPATH))
            {
                  std::cout << "error\n";
            }
            _mainMenuMusic.play();
            _mainMenuMusic.setVolume(this->volume);
            _mainMenuMusic.setLoop(true);
      }

      void MainMenuState::HandleInput()
      {
            sf::Event event;

            while (this->_data->window.pollEvent(event))
            {
                  if (sf::Event::Closed == event.type)
                        this->_data->window.close();

                  if (_data->input.IsTextClicked(_startText, sf::Mouse::Left,
_data->window)) {
                        _mainMenuMusic.stop();
                        _data->machine.AddState(StateRef(new
TutorialState(_data, this->volume)), true);
                  }
                  if (_data->input.IsTextClicked(_settingText,
sf::Mouse::Left, _data->window)) {
                        _mainMenuMusic.stop();
                        _data->machine.AddState(StateRef(new
SettingState(_data, this->volume)), true);
                  }
                  if (_data->input.IsTextClicked(_recordTableText,
sf::Mouse::Left, _data->window)) {
                        _mainMenuMusic.stop();
                        _data->machine.AddState(StateRef(new
RecordTableState(_data, this->volume)), true);
                  }
                  if (_data->input.IsTextClicked(_exitText, sf::Mouse::Left,
_data->window))
                        this->_data->window.close();
            }
      }

      void MainMenuState::Update(float dt)
      {
      }

      void MainMenuState::Draw(float dt)
      {
```

```
            this->_data->window.clear(sf::Color::Red);
            this->_data->window.draw(this->_background);
            this->_data->window.draw(this->_gameLogo);
            this->_data->window.draw(this->_startText);
            this->_data->window.draw(this->_settingText);
            this->_data->window.draw(this->_recordTableText);
            this->_data->window.draw(this->_exitText);
            this->_data->window.display();
      }
}
```

Файл: MainMenuState.h:

```
#pragma once

#include "SFML/Graphics.hpp"
#include "SFML/Audio.hpp"
#include "State.h"
#include "Game.h"

namespace TheGame
{
      class MainMenuState:public State
      {
      public:
            MainMenuState(GameDataRef data, int volume);
            void Init();
            void HandleInput() override;
            void Update(float dt) override;
            void Draw(float dt) override;
            void SetMainMenuText(sf::Text&, std::string, int);

      private:
            GameDataRef _data;
            sf::Sprite _background;
            sf::Sprite _gameLogo;
            sf::Text _startText;
            sf::Text _settingText;
            sf::Text _recordTableText;
            sf::Text _exitText;
            sf::Music _mainMenuMusic;
            int volume;
      };
}
```

Файл Question.cpp:

```
#include "Question.h"
#include "DEFINITIONS.h"
#include <string>
#include <fstream>

namespace TheGame
{
      void Question::getQuestion(int level)
      {
            srand(time(nullptr));
            int topic = std::rand() % 10 + 1;
            std::string filename = TOPIC_FILENAME + std::to_string(topic) +
TOPIC_EXTENSION;
            std::cout << filename << std::endl;
            std::ifstream inputFile(filename);
            std::string right;
            for (int i = 0; i < 2 + 8*(level - 1); i++)
```

```
                std::getline(inputFile, this->task);


            std::getline(inputFile, this->answerA);
            std::getline(inputFile, this->answerB);
            std::getline(inputFile, this->answerC);
            std::getline(inputFile, this->answerD);
            std::getline(inputFile, right);
            this->rightAnswer = std::stoi(right);
      }
}
```

Файл Question.h:

```
#pragma once
#include <iostream>

namespace TheGame
{
      class Question {
      public:
            std::string task;
            std::string answerA;
            std::string answerB;
            std::string answerC;
            std::string answerD;

            int rightAnswer;
            int difficulty = 0;
      public:
            void getQuestion(int level);
      };
}
```

Файл RecordTableState.h:

```
#pragma once
#include "SFML/Graphics.hpp"
#include "SFML/Audio.hpp"
#include "State.h"
#include "Game.h"

namespace TheGame
{
      struct RecordUnit {
            std::string playerName;
            int playerScore;
      };


      class RecordTableState : public State {
      public:
            RecordTableState(GameDataRef data, int volume);
            void Init() override;
            void HandleInput() override;
            void Update(float dt) override;
            void Draw(float dt) override;
            void SetRecordTableText(sf::Text&, std::string, int, std::string,
int);
            void LoadRecords(std::string filename);
      private:
            GameDataRef _data;
            sf::Sprite _background;
            sf::Text _titleText;
```

```
            sf::Text _backText;
            sf::Music _rtMusic;
            int volume;
            std::vector<sf::Text> _names;
            std::vector<sf::Text> _scores;
            std::vector<RecordUnit> _allRecords;
      };
}
```

Файл RecordTableState.cpp:

```
#include "RecordTableState.h"
#include <iostream>
#include <sstream>
#include <fstream>
#include <vector>
#include "MainMenuState.h"
#include "DEFINITIONS.h"
namespace TheGame
{
      void RecordTableState::SetRecordTableText(sf::Text& text, std::string
name, int size, std::string font, int thickness)//div size == 10 to text, 6
to name
      {
            text.setFont(this->_data->assets.GetFont(font));
            text.setString(name);
            text.setCharacterSize(size);
            sf::Color textColor(0xFF, 0xD7, 0x00);
            text.setFillColor(textColor);
            text.setOutlineThickness(thickness);
      }

      RecordTableState::RecordTableState(GameDataRef data, int volume) :
_data(data)
      {
            this->volume = volume;
      }

      void RecordTableState::Init()
      {
            this->_data->assets.LoadFont("The Font", FONT_FILEPATH);
            this->_data->assets.LoadFont("Ingame Font",
INGAME_FONT_FILEPATH);
            this->_data->assets.LoadFont("Bold Font", BOLD_RECORD_FILEPATH);

            this->LoadRecords(RECORDS_FILE);

            for (int i = 0; i < _allRecords.size(); i++) {
                  sf::Text _nameBuf, _scoreBuf;
                  SetRecordTableText(_nameBuf, this-
>_allRecords[i].playerName, 65, "Bold Font", 2);
                  SetRecordTableText(_scoreBuf, std::to_string(this-
>_allRecords[i].playerScore), 65, "Bold Font", 2);
                  _nameBuf.setPosition((SCREEN_WIDTH / 2) -
(_nameBuf.getGlobalBounds().width / 2) - 365, 216 + (90 * i));
                  _nameBuf.setStyle(sf::Text::Bold);
                  this->_names.push_back(_nameBuf);

                  _scoreBuf.setPosition((SCREEN_WIDTH / 2) -
(_scoreBuf.getGlobalBounds().width / 2) + 365, 216 + (90 * i));
                  _scoreBuf.setStyle(sf::Text::Bold);
                  this->_scores.push_back(_scoreBuf);
            }
```

```
            this->_data->assets.LoadTexture("Record Table Background",
RT_BACKGROUND_FILEPATH);

            _background.setTexture(this->_data->assets.GetTexture("Record
Table Background"));

            SetRecordTableText(_titleText, "RECORD TABLE", 120, "The Font",
3);
            SetRecordTableText(_backText, "BACK TO MAIN MENU", 70, "The
Font", 3);

            _titleText.setPosition((SCREEN_WIDTH / 2) -
(_titleText.getGlobalBounds().width / 2), 20);

            _backText.setPosition((SCREEN_WIDTH / 2) -
(_backText.getGlobalBounds().width / 2),
                  (SCREEN_HEIGHT / 2) + (SCREEN_HEIGHT / 10 * 4) -
_backText.getGlobalBounds().height / 2);
            if (!_rtMusic.openFromFile(RECORDTABLE_MUSIC_FILEPATH))
            {
                  std::cout << "error\n";
            }
            _rtMusic.play();
            _rtMusic.setVolume(this->volume);
            _rtMusic.setLoop(true);
      }

      void RecordTableState::HandleInput()
      {
            sf::Event event;

            while (this->_data->window.pollEvent(event))
            {
                  if (sf::Event::Closed == event.type)
                  {
                        this->_data->window.close();
                  }

                  if (sf::Keyboard::isKeyPressed(sf::Keyboard::Escape))
                  {
                        _rtMusic.stop();
                        _data->machine.AddState(StateRef(new
MainMenuState(_data, this->volume)), true);
                  }

                  if (_data->input.IsTextClicked(_backText, sf::Mouse::Left,
_data->window))
                  {
                        _rtMusic.stop();
                        _data->machine.AddState(StateRef(new
MainMenuState(_data, this->volume)), true);
                  }
            }
      }

      void RecordTableState::Update(float dt)
      {
      }

      void RecordTableState::Draw(float dt)
      {
            this->_data->window.clear(sf::Color::Red);
```

```
		this->_data->window.draw(this->_background);

		this->_data->window.draw(this->_titleText);
		this->_data->window.draw(this->_backText);

		for (const auto& unit : this->_names) {
			this->_data->window.draw(unit);
		}
		for (const auto& unit : this->_scores) {
			this->_data->window.draw(unit);
		}

		this->_data->window.display();
	}

	void RecordTableState::LoadRecords(std::string filename)
	{
		std::fstream recordFile(filename, std::ios::in | std::ios::out);

		if (!recordFile.is_open()) {
			std::cout << "Unable to get records." << std::endl;
			return;
		}

		std::string recordEntry;
		std::streampos filePos;

		while (std::getline(recordFile, recordEntry)) {
			std::istringstream inputStream(recordEntry);
			RecordUnit _entry;

			if (std::getline(inputStream, _entry.playerName, ':') &&
(inputStream >> _entry.playerScore)) {
				this->_allRecords.push_back(_entry);
			}
		}

		recordFile.close();
	}
}
```

Файл SettingState.cpp:

```
#include "SettingState.h"
#include <sstream>
#include "MainMenuState.h"
#include "DEFINITIONS.h"
#include <iostream>
namespace TheGame
{
	void SettingState::SetSettingsText(sf::Text& text, std::string name,
int divSize)//div size == 10 to text, 6 to name
	{
		text.setFont(this->_data->assets.GetFont("The Font"));
		text.setString(name);
		text.setCharacterSize(SCREEN_HEIGHT / divSize);
		sf::Color textColor(0xFF, 0xD7, 0x00);
		text.setFillColor(textColor);
	}

	SettingState::SettingState(GameDataRef data, int volume) : _data(data)
	{
		this->volume = volume;
	}
```

```
	void SettingState::Init()
	{
		this->_data->assets.LoadFont("The Font", FONT_FILEPATH);
		this->_data->assets.LoadFont("Volume Font", TIMER_FONT_FILEPATH);
		this->_data->assets.LoadTexture("Main Menu Background",
MAIN_MENU_BACKGROUND_FILEPATH);
		_background.setTexture(this->_data->assets.GetTexture("Main Menu
Background"));
		SetSettingsText(_settingsText, "SETTINGS", 9);
		_settingsText.setOutlineThickness(3);
		SetSettingsText(_musicText, "MUSIC", 15);
		_musicText.setOutlineThickness(3);
		SetSettingsText(_backText, "BACK TO MAIN MENU", 14);
		_backText.setOutlineThickness(3);
		SetSettingsText(_minus, "-", 15);
		_minus.setFont(this->_data->assets.GetFont("Volume Font"));
		_minus.setPosition(((SCREEN_WIDTH / 3) -
(_minus.getGlobalBounds().width / 2) + 400),
			(SCREEN_HEIGHT / 3) - _musicText.getGlobalBounds().height /
2);
		SetSettingsText(_plus, "+", 15);
		_plus.setFont(this->_data->assets.GetFont("Volume Font"));
		_plus.setPosition(((SCREEN_WIDTH / 3) -
(_plus.getGlobalBounds().width / 2) + 700),
			(SCREEN_HEIGHT / 3) - _musicText.getGlobalBounds().height /
2);
		SetSettingsText(_volume, std::to_string(this->volume), 15);
		_volume.setFont(this->_data->assets.GetFont("Volume Font"));
		_volume.setPosition(((SCREEN_WIDTH / 3) -
(_volume.getGlobalBounds().width / 2) + 550),
			(SCREEN_HEIGHT / 3) - _musicText.getGlobalBounds().height /
2);
		_settingsText.setPosition((SCREEN_WIDTH / 2) -
(_settingsText.getGlobalBounds().width / 2), 20);
		_musicText.setPosition((SCREEN_WIDTH / 3) -
(_musicText.getGlobalBounds().width / 2),
			(SCREEN_HEIGHT / 3) - _musicText.getGlobalBounds().height /
2);
		_backText.setPosition((SCREEN_WIDTH / 2) -
(_backText.getGlobalBounds().width / 2),
			(SCREEN_HEIGHT / 2) + (SCREEN_HEIGHT / 10 * 4) -
_backText.getGlobalBounds().height / 2);
		if (!_settingsMusic.openFromFile(MAIN_MENU_MUSIC_FILEPATH))
		{
			std::cout << "error\n";
		}
		_settingsMusic.setVolume(this->volume);
		_settingsMusic.play();
	}

	void SettingState::HandleInput()
	{
		sf::Event event;
		while (this->_data->window.pollEvent(event))
		{
			if (sf::Event::Closed == event.type)
				this->_data->window.close();

			if (_data->input.IsTextClicked(_backText, sf::Mouse::Left,
_data->window)) {
				_settingsMusic.stop();
```

```
                        _data->machine.AddState(StateRef(new
MainMenuState(_data, this->volume)), true);
                  }

                  if (_data->input.IsTextClicked(_plus, sf::Mouse::Left,
_data->window) && this->volume < 100) {
                        this->volume += 10;
                        _settingsMusic.setVolume(this->volume + 10);
                  }

                  if (_data->input.IsTextClicked(_minus, sf::Mouse::Left,
_data->window) && this->volume > 0) {
                        this->volume -= 10;
                        _settingsMusic.setVolume(this->volume - 10);
                  }
            }
      }

      void SettingState::Update(float dt)
      {
            _volume.setString(std::to_string(this->volume));
            this->_data->window.draw(this->_volume);
      }

      void SettingState::Draw(float dt)
      {
            this->_data->window.clear(sf::Color::Red);
            this->_data->window.draw(this->_background);
            this->_data->window.draw(this->_settingsText);
            this->_data->window.draw(this->_musicText);
            this->_data->window.draw(this->_minus);
            this->_data->window.draw(this->_plus);
            this->_data->window.draw(this->_volume);
            this->_data->window.draw(this->_backText);


            this->_data->window.display();
      }
}
```

Файл SettingState.h:

```
#pragma once
#include "SFML/Graphics.hpp"
#include "SFML/Audio.hpp"
#include "State.h"
#include "Game.h"
namespace TheGame
{
      class SettingState : public State {
      public:
            SettingState(GameDataRef data, int volume);
            void Init() override;
            void HandleInput() override;
            void Update(float dt) override;
            void Draw(float dt) override;
            void SetSettingsText(sf::Text&, std::string, int);
      private:
            GameDataRef _data;
            sf::Sprite _background;
            sf::Text _settingsText;
            sf::Text _musicText;
            sf::Text _backText;
            sf::Text _plus;
```

```
            sf::Text _minus;
            sf::Text _volume;
            sf::Music _settingsMusic;
            int volume;
      };
}
```

Файл State.h:

```
#pragma once
namespace TheGame {
      class State {
      public:
            virtual void Init() = 0;
            virtual void HandleInput() = 0;
            virtual void Update(float dt) = 0;
            virtual void Draw(float dt) = 0;
      };
}
```

Файл StateMachine.cpp:

```
#include "StateMachine.h"

namespace TheGame
{
      void StateMachine::AddState(StateRef newState, bool isReplacing)
      {
            this->_isAdding = true;
            this->_isReplacing = isReplacing;
            this->_newState = std::move(newState);
      }
      void StateMachine::RemoveState()
      {
            this->_isRemoving = true;
      }
      void StateMachine::ProcessStateChanges()
      {
            if (this->_isRemoving && !this->_states.empty()) {
                  this->_states.pop();
                  this->_isRemoving = false;
            }

            if (this->_isAdding) {
                  if (!this->_states.empty() && this->_isReplacing)
                              this->_states.pop();
                  this->_states.push(std::move(this->_newState));
                  this->_states.top()->Init();
                  this->_isAdding = false;
            }
      }
      StateRef& StateMachine::GetActiveState()
      {
            return this->_states.top();
      }
}
```

Файл StateMachine.h:

```
#pragma once
#include <memory>
#include <stack>
#include "State.h"
namespace TheGame
{
```

```
	typedef std::unique_ptr<State> StateRef;
	class StateMachine {
	public:
		void AddState(StateRef newState, bool isReplacing = true);
		void RemoveState();
		void ProcessStateChanges();
		StateRef& GetActiveState();
	private:
		std::stack<StateRef> _states;
		StateRef _newState;
		bool _isAdding;
		bool _isRemoving;
		bool _isReplacing;
	};
}
```

Файл TutorialState.cpp:

```
#include "TutorialState.h"
#include "QuestionState.hpp"
#include "GameOverState.h"
#include "DEFINITIONS.h"
#include "MainMenuState.h"
#include <sstream>
#include <iostream>
namespace TheGame
{
	TutorialState::TutorialState(GameDataRef data, int volume) :
_data(data)
	{
		this->volume = volume;
	}

	void TutorialState::Init()
	{
		std::setlocale(LC_ALL, "ru_RU.UTF-8");
		this->_data->assets.LoadFont("Ingame Font",
INGAME_FONT_FILEPATH);
		this->_data->assets.LoadTexture("Game Background",
MAIN_MENU_BACKGROUND_FILEPATH);
		_info.setFillColor(sf::Color(0x00, 0x00, 0x33));
		_info.setPosition(192, 108);
		_info.setSize(sf::Vector2f(1536, 864));
		_info.setOutlineColor(sf::Color::White);
		_info.setOutlineThickness(5);
		_infotext.setString(L"Вам предстоит ответить на 15 вопросов на
разные темы:\nгеография, искусство, точные науки и так далее. \n\n"
			"В распоряжении у вас три подсказки : \n\n"
			"Право на ошибку - позволяет ответить на вопрос два
раза;\n\n"
			"Смена вопроса - замените вопрос на другой такой же
сложности;\n\n"
			"Пропуск вопроса - переходите к следующему вопросу,\nно
учтите - вы теряете возможные очки за правильный ответ!\n\n"
			"Удачной игры!\n"
			"И помните: чем меньше времени вы потратите на ответ,\nтем
больше очков вы получите при победе!\n"
			"\n\n\n\nНажмите чтобы продолжить.");
		_infotext.setFont(this->_data->assets.GetFont("Ingame Font"));
		_infotext.setCharacterSize(33);
		_infotext.setFillColor(sf::Color::White);
		_infotext.setPosition(200, 150);
		_infotext.setStyle(sf::Text::Bold);
```

```
            _background.setTexture(this->_data->assets.GetTexture("Game
Background"));
            if (!_tutorialMusic.openFromFile(TUTORIAL_MUSIC_FILEPATH))
                  std::cout << "error\n";
            _tutorialMusic.setVolume(this->volume);
            _tutorialMusic.play();
            _tutorialMusic.setLoop(true);
      }

      void TutorialState::HandleInput()
      {
            sf::Event event;
            while (this->_data->window.pollEvent(event)) {
                  if (sf::Event::Closed == event.type)
                        this->_data->window.close();
                  if (sf::Keyboard::isKeyPressed(sf::Keyboard::Escape)) {
                        _tutorialMusic.stop();
                        _data->machine.AddState(StateRef(new
MainMenuState(_data, this->volume)), true);
                  }
                  if (sf::Keyboard::isKeyPressed(sf::Keyboard::Enter) ||
_data->input.IsTextClicked(_infotext, sf::Mouse::Left, _data->window)) {
                        _tutorialMusic.stop();
                        Lifeline hints;
                        _data->machine.AddState(StateRef(new
QuestionState(_data, this->volume, 1, 0, sf::seconds(45.f), hints)), true);
                  }
            }
      }

      void TutorialState::Update(float dt)
      {
      }

      void TutorialState::Draw(float dt)
      {
                  this->_data->window.clear(sf::Color::Red);
                  this->_data->window.draw(this->_background);
                  this->_data->window.draw(_info);
                  this->_data->window.draw(_infotext);
                  this->_data->window.display();
      }
}
```

Файл TutorialState.h:

```
#pragma once
#include "SFML/Graphics.hpp"
#include "SFML/Audio.hpp"
#include "State.h"
#include "Game.h"
namespace TheGame
{
      class TutorialState : public State {
      public:
            TutorialState(GameDataRef data, int volume);
            void Init() override;
            void HandleInput() override;
            void Update(float dt) override;
            void Draw(float dt) override;
      private:
            GameDataRef _data;
            sf::Sprite _background;
            sf::RectangleShape _info;
```

```
            sf::Text _infotext;
            sf::Music _tutorialMusic;
            int volume;
      };
}
```

Файл VictoryState.cpp:

```
#include "VictoryState.h"
#include "DEFINITIONS.h"
#include "MainMenuState.h"
#include <sstream>
#include <fstream>
#include <vector>
#include <iostream>
namespace TheGame
{
      VictoryState::VictoryState(GameDataRef data, int volume, int score) :
_data(data)
      {
            this->volume = volume;
            this->_score = score;
      }
      void VictoryState::Init()
      {
            this->_data->assets.LoadFont("Bold Font", BOLD_RECORD_FILEPATH);
            std::cout << "Victory!" << std::endl;
            if (!_victorySoundBuffer.loadFromFile(VICTORY_SFX))
            {
                  std::cout << "error\n";
            }
            _victorySound.setBuffer(_victorySoundBuffer);
            _victorySound.setVolume(this->volume);
            _victorySound.play();
            _inputText.setFont(this->_data->assets.GetFont("Bold Font"));
            _inputText.setCharacterSize(40);
            _inputText.setFillColor(sf::Color(0xFF, 0xD7, 0x00));
            _inputText.setPosition(615, 330);
            _scoreText.setFont(this->_data->assets.GetFont("Bold Font"));
            _scoreText.setCharacterSize(40);
            _scoreText.setFillColor(sf::Color(0xFF, 0xD7, 0x00));
            _scoreText.setPosition(615, 550);
            _scoreText.setString("Score: " + std::to_string(this->_score));
            this->_data->assets.LoadTexture("Victory Background",
VICTORY_BACKGROUND_FILEPATH);
            _background.setTexture(this->_data->assets.GetTexture("Victory
Background"));
            this->getRecords(RECORDS_FILE);
      }
      void VictoryState::HandleInput()
      {
            sf::Event event;
            while (this->_data->window.pollEvent(event))
            {
                  if (sf::Event::Closed == event.type)
                  {
                        this->_data->window.close();
                  }
                  if (event.type == sf::Event::TextEntered) {
                        if (sf::Keyboard::isKeyPressed(sf::Keyboard::Enter))
                        {
                              this->updateRecords(this->playerName, this-
>_score, RECORDS_FILE);
```

```
                              _data->machine.AddState(StateRef(new
MainMenuState(_data, this->volume)), true);
                        }
                        else if (event.text.unicode < 128 &&
event.text.unicode != 8 && this->playerName.length() < NAME_LENGTH_LIMIT) {
                              this->playerName +=
static_cast<char>(event.text.unicode);
                              _inputText.setString(this->playerName);
                        }
                        else if (event.text.unicode == 8 && !this-
>playerName.empty()) {
                              this->playerName.pop_back();
                              _inputText.setString(this->playerName);
                        }
                  }
            }
      }
      void VictoryState::Update(float dt)
      {
      }
      void VictoryState::Draw(float dt)
      {
            this->_data->window.clear(sf::Color::Red);
            this->_data->window.draw(this->_background);
            this->_data->window.draw(this->_inputText);
            this->_data->window.draw(this->_scoreText);
            this->_data->window.display();
      }
      void VictoryState::getRecords(std::string filename)
      {
            std::fstream recordFile(filename, std::ios::in | std::ios::out);
            if (!recordFile.is_open()) {
                  std::cout << "Unable to get records." << std::endl;
                  return;
            }
            std::string recordEntry;
            std::streampos filePos;
            while (std::getline(recordFile, recordEntry)) {
                  std::istringstream inputStream(recordEntry);
                  RecordUnit _entry;
                  if (std::getline(inputStream, _entry.playerName, ':') &&
(inputStream >> _entry.playerScore)) {
                        this->_allRecords.push_back(_entry);
                  }
            }
            recordFile.close();
      }
      void VictoryState::updateRecords(std::string playerName, int
playerScore, std::string filename)
      {
            RecordUnit entry;
            entry.playerName = playerName;
            entry.playerScore = playerScore;
            this->_allRecords.push_back(entry);
            std::sort(this->_allRecords.begin(), this->_allRecords.end(),
[](const RecordUnit& a, const RecordUnit& b) {return a.playerScore >
b.playerScore; });
            if (this->_allRecords.size() > RECORD_ENTRIES_LIMIT)
                  this->_allRecords.pop_back();
            std::ofstream recordFile(filename);
            if (recordFile.is_open()) {
                  for (const auto& unit : this->_allRecords) {
```

```
                        recordFile << unit.playerName << ": " <<
unit.playerScore << std::endl;
                  }
                  recordFile.close();
            }
            else {
                  std::cout << "Unable to update records." << std::endl;
                  return;
            }
      }
}
```

Файл VictoryState.h:

```
#pragma once
#include "SFML/Graphics.hpp"
#include "SFML/Audio.hpp"
#include "State.h"
#include "Game.h"
namespace TheGame
{
      struct RecordUnit {
            std::string playerName;
            int playerScore;
      };
      class VictoryState : public State {
      public:
            VictoryState(GameDataRef data, int volume, int score);
            void Init() override;
            void HandleInput() override;
            void Update(float dt) override;
            void Draw(float dt) override;
            void getRecords(std::string filename);
            void updateRecords(std::string playerName, int playerScore,
std::string filename);
      private:
            GameDataRef _data;
            sf::Sprite _background;
            sf::Text _inputText;
            sf::Text _scoreText;
            std::string playerName;
            sf::SoundBuffer _victorySoundBuffer;
            sf::Sound _victorySound;
            int volume;
            int _score;
            std::vector<RecordUnit> _allRecords;
      };
}
```

Файл QuestionState.cpp:

```
#include "QuestionState.h" // 100.470 + width 800 + height 150
#include "DEFINITIONS.h"      //100.690 -> 1030.470 -> 1030.690
#include "MainMenuState.h"
#include "Question.h"
#include "GameOverState.h"
#include "VictoryState.h"
#include <sstream>
#include <locale>
#include <codecvt>
#include <iostream>
namespace TheGame
{
```

```
	QuestionState::QuestionState(GameDataRef data, int volume, int level,
int score, sf::Time timer, Lifeline hints) : _data(data)
	{
		this->volume = volume;
		this->currentLevel = level;
		this->currentScore = score;
		this->_timer = timer;
		this->availableBonuses = hints;
	}
	void QuestionState::SetQuestionText(sf::Text& text, std::wstring
name)//div size == 10 to text, 6 to name
	{
		text.setFont(this->_data->assets.GetFont("Ingame Font"));
		text.setString(name);
		text.setCharacterSize(33);
		text.setFillColor(sf::Color::White);
		text.setStyle(sf::Text::Bold);
	}
	void QuestionState::Init()
	{
		std::setlocale(LC_ALL, "ru_RU.UTF-8");
		currentQuestion.getQuestion(currentLevel);
		std::cout << currentQuestion.task << std::endl;
		//current.task.insert(5, "\n");
		std::cout << currentQuestion.rightAnswer << std::endl;
		std::string progress = std::to_string(currentLevel) +
FULL_PROGRESS;
		std::string cScore = _SCORE_ + std::to_string(currentScore);
		std::cout << progress << std::endl;
		std::cout << "Score: " << currentScore << std::endl;
		std::setlocale(LC_ALL, "ru_RU.UTF-8");
		this->_data->assets.LoadFont("Ingame Font",
INGAME_FONT_FILEPATH);
		this->_data->assets.LoadFont("Timer Font", TIMER_FONT_FILEPATH);
		this->_data->assets.LoadTexture("Task Background",
QUESTION_BACKGROUND_FILEPATH);
		this->_data->assets.LoadTexture("Win A", IF_RIGHT_A);
		this->_data->assets.LoadTexture("Win B", IF_RIGHT_B);
		this->_data->assets.LoadTexture("Win C", IF_RIGHT_C);
		this->_data->assets.LoadTexture("Win D", IF_RIGHT_D);
		this->_data->assets.LoadTexture("Lose A", IF_WRONG_A);
		this->_data->assets.LoadTexture("Lose B", IF_WRONG_B);
		this->_data->assets.LoadTexture("Lose C", IF_WRONG_C);
		this->_data->assets.LoadTexture("Lose D", IF_WRONG_D);
		this->_data->assets.LoadTexture("Skip",
SKIP_QUESTION_ICON_FILEPATH);
		_skip.setTexture(this->_data->assets.GetTexture("Skip"));
		_skip.setPosition(650, 933);
		this->_data->assets.LoadTexture("Change",
CHANGE_QUESTION_ICON_FILEPATH);
		_change.setTexture(this->_data->assets.GetTexture("Change"));
		_change.setPosition(500, 955);
		this->_data->assets.LoadTexture("Mistake",
SECOND_CHANCE_ICON_FILEPATH);
		_mistake.setTexture(this->_data->assets.GetTexture("Mistake"));
		_mistake.setPosition(380, 954);
		_timerFrame.setRadius(95);
		_timerFrame.setPointCount(45);
		_timerFrame.setPosition(865, 865);
		_timerFrame.setOutlineColor(sf::Color::White);
		_timerFrame.setOutlineThickness(2);
		_timerFrame.setFillColor(sf::Color::Black);
```

```
            SetQuestionText(_timeRemaining,
std::to_wstring(static_cast<int>(_timer.asSeconds())));
            _timeRemaining.setPosition(902.5, 908.5);
            _timeRemaining.setCharacterSize(85);
            _timeRemaining.setFont(this->_data->assets.GetFont("Timer
Font"));
            _background.setTexture(this->_data->assets.GetTexture("Task
Background"));
            std::wstring_convert<std::codecvt_utf8<wchar_t>, wchar_t>
converter;
            std::wstring wTask = converter.from_bytes(currentQuestion.task);
            SetQuestionText(_task, wTask);
            _task.setPosition(170, 150);
            std::wstring wA = converter.from_bytes(currentQuestion.answerA);
            SetQuestionText(_answerA, wA);
            _answerA.setPosition(140, 520);
            _buttonA.setTextureRect(sf::IntRect(100, 470, 800, 150));
            _buttonA.setColor(sf::Color::Transparent);
            std::wstring wB = converter.from_bytes(currentQuestion.answerB);
            SetQuestionText(_answerB, wB);
            _answerB.setPosition(1070, 520);
            _buttonB.setTextureRect(sf::IntRect(1030, 470, 800, 150));
            _buttonB.setColor(sf::Color::Transparent);
            std::wstring wC = converter.from_bytes(currentQuestion.answerC);
            SetQuestionText(_answerC, wC);
            _answerC.setPosition(140, 747);
            _buttonC.setTextureRect(sf::IntRect(100, 690, 800, 150));
            _buttonC.setColor(sf::Color::Transparent);
            std::wstring wD = converter.from_bytes(currentQuestion.answerD);
            SetQuestionText(_answerD, wD);
            _answerD.setPosition(1070, 747);
            _buttonD.setTextureRect(sf::IntRect(1030, 690, 800, 150));
            _buttonD.setColor(sf::Color::Transparent);
            std::wstring wProgress = converter.from_bytes(progress);
            SetQuestionText(_progress, wProgress);
            _progress.setPosition(100, 950);
            _progress.setCharacterSize(70);
            std::wstring wScore = converter.from_bytes(cScore);
            SetQuestionText(_score, wScore);
            _score.setPosition(1350, 950);
            _score.setCharacterSize(70);
            if (!_questionMusic.openFromFile(QUESTION_MUSIC_FILEPATH))
            {
                  std::cout << "error\n";
            }
            _questionMusic.setVolume(this->volume);
            _questionMusic.play();
      }
      void QuestionState::HandleInput()
      {
            sf::Event event;
            while (this->_data->window.pollEvent(event))
            {
                  if (sf::Event::Closed == event.type)
                  {
                        this->_data->window.close();
                  }
                  if (sf::Keyboard::isKeyPressed(sf::Keyboard::Escape))
                  {
                        _questionMusic.stop();
                        _data->machine.AddState(StateRef(new
MainMenuState(_data, this->volume)), true);
                  }
```

```
                  if (event.type == sf::Event::MouseButtonPressed) {
                        if (event.mouseButton.button == sf::Mouse::Left) {
                              sf::FloatRect boundsA(100, 470, 800, 150);
                              sf::FloatRect boundsB(1030, 470, 800, 150);
                              sf::FloatRect boundsC(100, 690, 800, 150);
                              sf::FloatRect boundsD(1030, 690, 800, 150);
                              sf::FloatRect bounds50(400, 950, 100, 70);
                              sf::Vector2f mousePos = this->_data-
>window.mapPixelToCoords(sf::Mouse::getPosition(this->_data->window));
                              if (boundsA.contains(mousePos)) {
                                    if (currentQuestion.rightAnswer == 1) {
                                          currentLevel++;
                                          currentScore += 100 * (currentLevel
- 1) + (currentLevel - 1) * (static_cast<int>(_timer.asSeconds()));
                                          _questionMusic.stop();
                                          _background.setTexture(this->_data-
>assets.GetTexture("Win A"));

      _answerA.setFillColor(sf::Color::Black);
                                          if (currentLevel == 16) { //16
                                                _data-
>machine.AddState(StateRef(new VictoryState(_data, this->volume,
currentScore)));
                                                break;
                                          }
                                          _data-
>machine.AddState(StateRef(new QuestionState(_data, this->volume,
currentLevel, currentScore, sf::seconds(45.f), this->availableBonuses)));
                                    }
                                    else {
                                          _background.setTexture(this->_data-
>assets.GetTexture("Lose A"));
                                          if (!_twoChances) {
                                                _questionMusic.stop();
                                                _data-
>machine.AddState(StateRef(new GameOverState(_data, this->volume)), true);
                                          }
                                          _twoChances = false;
                                    }
                              }
                              if (boundsB.contains(mousePos)) {
                                    if (currentQuestion.rightAnswer == 2) {
                                          currentLevel++;
                                          currentScore += 100 * (currentLevel
- 1) + (currentLevel - 1) * (static_cast<int>(_timer.asSeconds()));
                                          _questionMusic.stop();
                                          _background.setTexture(this->_data-
>assets.GetTexture("Win B"));

      _answerB.setFillColor(sf::Color::Black);
                                          if (currentLevel == 16) { //16
                                                _data-
>machine.AddState(StateRef(new VictoryState(_data, this->volume,
currentScore)));
                                                break;
                                          }
                                          _data-
>machine.AddState(StateRef(new QuestionState(_data, this->volume,
currentLevel, currentScore, sf::seconds(45.f), this->availableBonuses)));
                                    }
                                    else {
                                          _background.setTexture(this->_data-
>assets.GetTexture("Lose B"));
```

```
                                        if (!_twoChances) {
                                              _questionMusic.stop();
                                              _data-
>machine.AddState(StateRef(new GameOverState(_data, this->volume)), true);
                                        }
                                        _twoChances = false;
                                  }
                                  //Button B functionality
                            }
                            if (boundsC.contains(mousePos)) {
                                  if (currentQuestion.rightAnswer == 3) {
                                        currentLevel++;
                                        currentScore += 100 * (currentLevel
- 1) + (currentLevel - 1) * (static_cast<int>(_timer.asSeconds()));
                                        _questionMusic.stop();
                                        _background.setTexture(this->_data-
>assets.GetTexture("Win C"));

      _answerC.setFillColor(sf::Color::Black);
                                        if (currentLevel == 16) { //16
                                              _data-
>machine.AddState(StateRef(new VictoryState(_data, this->volume,
currentScore)));
                                                    break;
                                        }
                                        _data-
>machine.AddState(StateRef(new QuestionState(_data, this->volume,
currentLevel, currentScore, sf::seconds(45.f), this->availableBonuses)));
                                  }
                                  else {
                                        _background.setTexture(this->_data-
>assets.GetTexture("Lose C"));
                                        if (!_twoChances) {
                                              _questionMusic.stop();
                                              _data-
>machine.AddState(StateRef(new GameOverState(_data, this->volume)), true);
                                        }
                                        _twoChances = false;
                                  }
                                  //Button C functionality
                            }
                            if (boundsD.contains(mousePos)) {
                                  if (currentQuestion.rightAnswer == 4) {
                                        currentLevel++;
                                        currentScore += 100 * (currentLevel
- 1) + (currentLevel - 1) * (static_cast<int>(_timer.asSeconds()));
                                        _questionMusic.stop();
                                        _background.setTexture(this->_data-
>assets.GetTexture("Win D"));

      _answerD.setFillColor(sf::Color::Black);
                                        if (currentLevel == 16) { //16
                                              _data-
>machine.AddState(StateRef(new VictoryState(_data, this->volume,
currentScore)));
                                                    break;
                                        }
                                        _data-
>machine.AddState(StateRef(new QuestionState(_data, this->volume,
currentLevel, currentScore, sf::seconds(45.f), this->availableBonuses)));
                                  }
                                  else {
```

```
                                                _background.setTexture(this->_data-
>assets.GetTexture("Lose D"));
                                                if (!_twoChances) {
                                                      _questionMusic.stop();
                                                      _data-
>machine.AddState(StateRef(new GameOverState(_data, this->volume)), true);
                                                }
                                                _twoChances = false;
                                          }
                                          //Button D functionality
                                    }
                              }
                        }
                        if (_data->input.IsSpriteClicked(_skip, sf::Mouse::Left,
_data->window) && this->availableBonuses.skipQuestion)
                        {
                              this->availableBonuses.skipQuestion = false;
                              _questionMusic.stop();
                              if (currentLevel == 15)
                                    _data->machine.AddState(StateRef(new
VictoryState(_data, this->volume, currentScore)));
                              else
                                    _data->machine.AddState(StateRef(new
QuestionState(_data, this->volume, currentLevel + 1, currentScore,
sf::seconds(45.f), this->availableBonuses)));
                        }
                        if (_data->input.IsSpriteClicked(_change, sf::Mouse::Left,
_data->window) && this->availableBonuses.changeQuestion)
                        {
                              this->availableBonuses.changeQuestion = false;
                              _questionMusic.stop();
                              _data->machine.AddState(StateRef(new
QuestionState(_data, this->volume, currentLevel, currentScore,
sf::seconds(45.f), this->availableBonuses)));
                        }
                        if (_data->input.IsSpriteClicked(_mistake, sf::Mouse::Left,
_data->window) && this->availableBonuses.twoChances)
                        {
                              this->availableBonuses.twoChances = false;
                              _twoChances = true;
                        }
                  }
            }
            void QuestionState::Update(float dt)
            {
                  if (_timerClock.getElapsedTime().asSeconds() >= 1) {
                        _timer -= sf::seconds(1);
                        if (static_cast<int>(_timer.asSeconds()) >= 10)

            _timeRemaining.setString(std::to_wstring(static_cast<int>(_timer.asSeco
nds())));
                        else {
                              _timeRemaining.setString(L"0" +
std::to_wstring(static_cast<int>(_timer.asSeconds())));
                        }
                        _timerClock.restart();
                  }
                  if (static_cast<int>(_timer.asSeconds()) <= 0) {
                        _questionMusic.stop();
                        _data->machine.AddState(StateRef(new GameOverState(_data,
this->volume)), true);
                  }
                  this->_data->window.draw(this->_background);
```

```
	this->_data->window.draw(this->_task);
	this->_data->window.draw(this->_answerA);
	this->_data->window.draw(this->_answerB);
	this->_data->window.draw(this->_answerC);
	this->_data->window.draw(this->_answerD);
	this->_data->window.draw(this->_buttonA);
	this->_data->window.draw(this->_buttonB);
	this->_data->window.draw(this->_buttonC);
	this->_data->window.draw(this->_buttonD);
	this->_data->window.draw(this->_progress);
	this->_data->window.draw(this->_score);
	if (!this->availableBonuses.skipQuestion) {
		sf::Color x = _skip.getColor();
		x.a = 60;
		_skip.setColor(x);
	}
	if (!this->availableBonuses.twoChances) {
		sf::Color x = _mistake.getColor();
		x.a = 60;
		_mistake.setColor(x);
	}
	if (!this->availableBonuses.changeQuestion) {
		sf::Color x = _change.getColor();
		x.a = 60;
		_change.setColor(x);
	}
	this->_data->window.draw(this->_skip);
	this->_data->window.draw(this->_mistake);
	this->_data->window.draw(this->_change);
	this->_data->window.draw(this->_timerFrame);
	this->_data->window.draw(this->_timeRemaining);
}
void QuestionState::Draw(float dt)
{
	this->_data->window.clear(sf::Color::Red);
	this->_data->window.draw(this->_background);
	this->_data->window.draw(this->_task);
	this->_data->window.draw(this->_answerA);
	this->_data->window.draw(this->_answerB);
	this->_data->window.draw(this->_answerC);
	this->_data->window.draw(this->_answerD);
	this->_data->window.draw(this->_buttonA);
	this->_data->window.draw(this->_buttonB);
	this->_data->window.draw(this->_buttonC);
	this->_data->window.draw(this->_buttonD);
	this->_data->window.draw(this->_progress);
	this->_data->window.draw(this->_score);
	if (!this->availableBonuses.skipQuestion) {
		sf::Color x = _skip.getColor();
		x.a = 60;
		_skip.setColor(x);
	}
	if (!this->availableBonuses.twoChances) {
		sf::Color x = _mistake.getColor();
		x.a = 60;
		_mistake.setColor(x);
	}
	if (!this->availableBonuses.changeQuestion) {
		sf::Color x = _change.getColor();
		x.a = 60;
		_change.setColor(x);
	}
	this->_data->window.draw(this->_skip);
```

```
            this->_data->window.draw(this->_mistake);
            this->_data->window.draw(this->_change);
            this->_data->window.draw(this->_timerFrame);
            this->_data->window.draw(this->_timeRemaining);
            this->_data->window.display();
      }
}
```

Файл QuestionState.h:

```
#pragma once
#include "SFML/Graphics.hpp"
#include "SFML/Audio.hpp"
#include "State.h"
#include "Game.h"
#include "Question.h"
namespace TheGame
{
      struct Lifeline {
            bool skipQuestion = true;
            bool changeQuestion = true;
            bool twoChances = true;
      };
      class QuestionState : public State
      {
      public:
            QuestionState(GameDataRef data, int volume, int level, int score,
sf::Time timer, Lifeline hints);
            void Init() override;
            void HandleInput() override;
            void Update(float dt) override;
            void Draw(float dt) override;
            void SetQuestionText(sf::Text&, std::wstring);
      private:
            GameDataRef _data;
            sf::Sprite _background;
            sf::Sprite _buttonA;
            sf::Sprite _buttonB;
            sf::Sprite _buttonC;
            sf::Sprite _buttonD;
            sf::Sprite _skip;
            sf::Sprite _mistake;
            sf::Sprite _change;
            sf::Time _timer;
            sf::CircleShape _timerFrame;
            sf::Clock _timerClock;
            sf::Text _timeRemaining;
            Lifeline availableBonuses;
            bool _twoChances = false;
            sf::Music _questionMusic;
            int volume;
            sf::Text _task;
            sf::Text _answerA;
            sf::Text _answerB;
            sf::Text _answerC;
            sf::Text _answerD;
            sf::Text _progress;
            sf::Text _score;

            Question currentQuestion;
            int currentLevel;
            int currentScore = 0;
      };
}
```

ПРИЛОЖЕНИЕ Д
(Обязательное)
Ведомость документов