МИНОБРНАУКИ РОССИИ
федеральное государственное бюджетное образовательное
учреждение высшего образования
«Балтийский государственный технический университет «ВОЕНМЕХ» им. Д.Ф. Устинова» (БГТУ «ВОЕНМЕХ» им. Д.Ф. Устинова»)

БГТУ.СМК-Ф-4.2-К5-02

Факультет О Естественнонаучный
шифр наименование

Кафедра О7 Информационные системы и программная инженерия
шифр наименование

Дисциплина Информационные технологии и программирование

КУРСОВАЯ РАБОТА
на тему

Объектно-ориентированная разработка программ с графическим пользовательским интерфейсом «снизу-вверх»
Вариант: Лабиринт

Выполнил студент группы О722Б

Вяткин Н. А.
Фамилия И.О.

РУКОВОДИТЕЛЬ

Васюков В. М. _____
Фамилия И.О. Подпись

Оценка _____

«_____» _____ 2023 г.

САНКТ-ПЕТЕРБУРГ,
2023 г.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В современном мире игры являются неотъемлемой частью развлечений и отдыха. Игры создаются на различных платформах и языках программирования, одним из которых является C++ [1]. Для разработки игр на C++ существует множество библиотек, одной из наиболее популярных является SFML (англ. Simple and Fast Multimedia Library) [2]. Она предоставляет удобный интерфейс для работы с графикой, звуком и сетью. Будут рассмотрены основные принципы работы с графикой, звуком и управлением игровым процессом.

Целью курсовой работы является разработка интерактивного графического приложения – игры с использованием мультимедийной библиотеки SFML.

В данной курсовой работе будет рассмотрен процесс разработки игры «Лабиринт», захватывающей головоломки с элементами управления игровым персонажем.

Для достижения поставленной цели необходимо решить следующие задачи:

- описать основные требования к разрабатываемой игре;
- составить иерархию классов;
- разработать игру;
- продемонстрировать работоспособность игры.

1 Постановка задачи

Игра лабиринт предполагает, что есть пять лабиринтов разных уровней сложности, пользователь сам может ввести имя игрока и выбрать уровень для прохождения, после прохождения уровня на экран выводится имя, номер уровня и время прохождения. Сложность уровней зависит от его порядкового номера. Информация для построения лабиринта хранится в файле. Пользователь управляет мышью, которая находится в начале лабиринта, чтобы пройти уровень нужно найти выход. Во время прохождения есть пауза, в которой возможен выход обратно в меню. Игру можно считать пройденной, когда пройдены все пять уровней. Управление происходит при помощи клавиатуры.

В приложении требуется реализовать пункт меню «Новая игра», при выборе которого пользователь попадает в подменю, где может выбрать уровень и ввести имя игрока. При попадании в режим игры необходимо реализовать игровой персонаж, его анимацию и движение, реализовать создание карты из файла, карта должна превосходить размер экрана, поэтому камера должна двигаться непосредственно за персонажем, он не должен выходить за пределы карты и проходить сквозь стены.

После прохождении уровня нужно дать пользователю возможность перейти на следующий уровень или выйти, если уровень последний, только выйти.

Также требуется реализовать в главном меню пункт «Справка и управление», где пользователь может ознакомиться с принципом игры и с управлением.

Также должен быть пункт «Результаты», в котором необходимо создать подменю, где пользователь может выбрать по какому уровню посмотреть результаты, после выбора уровня, данные о прохождении уровня должны представляться в табличном виде, и в отсортированном порядке, от лучшего времени к худшему, также у пользователя должна быть возможность возврата обратно в меню, очищения результатов, и подготовки файла для печати.

2 Описание иерархии классов

В процессе разработки приложения были созданы следующие классы:

– State – абстрактный класс, описывающий основной интерфейс состояния игры;

– Entity – абстрактный класс, описывающий представление игрового персонажа;

– Game – класс, описывающий всю игру, в котором происходит переключение всех состояний;

– BaseStateData – класс, хранящий основные данные для состояний;

– MenuState – класс, описывающий интерфейс главного меню игры;

– LevelState – класс, описывающий интерфейс меню выбора уровня и ввода имени игрока;

– Button – класс, описывающий поведение кнопок;

– TextBox – класс, описывающий поле для ввода имени игрока;

– GameState – класс, описывающий игровой процесс;

– PauseMenu – класс, описывающий меню паузы;

– Tile – класс, описывающий один элемент из игровой карты;

– TileMap – класс, описывающий игровую карту;

– WinState – класс, описывающий состояние игры после прохождения уровня;

– RulesState – класс, описывающий состояние игры, где показаны правила игры и управление;

– ResultState – класс, описывающий интерфейс меню выбора уровня, для показа результатов;

– TableState – класс, описывающий табличный показ результатов по конкретному уровню;

– Player – класс игрока, используется для игры;

– Animation – класс, реализующий компоненты для анимации игрока;

– AnimationPlayer – класс, описывающий анимацию игрока;

– Movement – класс, описывающий движение игрока;

– CollisionComponent – класс, описывающий столкновения игрока.

2.1 Класс BaseStateData

Данный класс содержит следующие публичные поля и методы:

– float gridSize – переменная для хранения размера сетки игрового окна;

– RenderWindow *window – игровое окно, используется для прорисовки игры и получения позиций мыши;

– map<string, int> *supportKeys – контейнер для хранения клавиш игры;

– stack<State*> *situations – стек для переключения между состояниями игры.

– BaseStateData() – конструктор без параметров, заполняется в другом классе.

2.2 Класс State

Данный класс содержит следующие частные поля и методы:

– BaseStateData *BaseState – данное поле используется для хранения и передачи, данных о вспомогательных клавишах, стека вызова состояний, размера сетки окна;

– RenderWindow *window – игровое окно, используется для прорисовки игры и получения позиций мыши;

– stack<State*> *situations – стек для переключения между состояниями игры;

– map<string, int> *supportKeys – контейнер для хранения клавиш игры;

– map<string, int> keyBinds – контейнер для хранения клавиш игрового состояния;

– map<string, Texture> textures – контейнер для хранения текстур и

быстрого доступа к ним в состоянии;

– bool quit – переменная для хранения состояния кнопки выход;

– bool paused – переменная для хранения состояния паузы;

– bool pressed – переменная для хранения состояния поля для ввода имени, выбрано или нет;

– float gridSize – переменная для хранения размера сетки игрового окна;

– Vector2i mousePosScreen – позиция мыши на экране;

– Vector2i mousePosWindow – позиция мыши в окне;

– Vector2f mousePosView – позиция мыши в виде;

– Void InitKeyBinds() – виртуальный метод, используется для добавления клавиш, в наследуемых состояниях.

Также данный класс описывает следующие публичные методы:

– State(BaseStateData *BaseState) – метод для создания состояния, на вход получает указатель на данные BaseStateData для состояния;

– const bool &getQuit() const – метод возвращающий значение переменной выход;

– void Paused() – метод, ставящий состояние на паузу, используется только в состоянии игры;

– void UnPaused() – метод, отменяющий паузу, используется только в состоянии игры;

– void endState() – метод, который заканчивает текущее состояние;

– void Selected() – метод, который показывает выбирает поле для ввода, используется только в LevelState;

– void UnSelected() – метод, который отменяет выбор поля для ввода, используется только в LevelState;

– void updateMousePosition(View *view = NULL) – виртуальный метод обновления позиции мыши, в качестве входного параметра выступает view, если его нет, то NULL, для определения позиции мыши в View;

– void updateInput(const float &deltaTime) – чистый виртуальный метод обновления действий пользователя относительно времени, входной параметр вещественное число, показывающее изменение времени;

– void update(const float &deltaTime) – чистый виртуальный метод обновления экрана относительно времени, входной параметр вещественное число, показывающее изменение времени;

– void render(RenderTarget *target=NULL) – чистый виртуальный метод показа состояния на экран, входной параметр RenderTarget*, то где необходимо показать состояние.

2.3 Класс MenuState

Данный класс содержит следующие частные поля и методы:

– Texture BackGroundTexture – переменная используется для хранения текстуры для заднего фона;

– RectangleShape BackGround – прямоугольная область для заднего фона;

– Font font – шрифт для текста;

– RectangleShape container – прямоугольная область для текста;

– Text text – переменная для отображения текста на экране;

– map<string, Button *> buttons – контейнер для хранения кнопок в данном состояние с ключом string и указателем на кнопку;

– void InitBackGround() – метод загружающий текстуру для заднего фона;

– void InitFonts() – метод загружающий шрифт для текста;

– void InitKeyBinds() – метод как в базовом классе для загрузки клавиш, в данном случае пустой;

– void InitButtons() – метод загрузки кнопок для меню.

Данный класс содержит следующие публичны методы:

– MenuState(BaseStateData *BaseState) – метод создания состояния главного меню, на вход получает указатель на данные для состояния

BaseStateData;

– void updateInput(const float &deltaTime) – метод обновления действий пользователя относительно времени, такой же как и в базовом классе State, входной параметр вещественное число, показывающее изменение времени;

– void updateButtons() – уникальный метод класс, обновления состояний кнопок;

– void update(const float &deltaTime) – данный метод реализует соответствующий метод базового класса State, особенностью реализации является использования дополнительных методов обновления позиции мыши, обновление ввода пользователя и обновления кнопок;

– void renderButtons(RenderTarget &target) – данный метод прорисовывает кнопки, на вход получает RenderTarget, то есть область где рисовать;

– void render(RenderTarget *target=NULL) – данный метод реализует соответствующий метод базового класса State, отличается тем что прорисовывает несколько прямоугольников и текст, а также кнопки, входной параметр RenderTarget*, то где необходимо показать состояние.

2.4 Класс LevelState

Данный класс содержит следующие частные поля и методы:

– Texture BackGroundTexture – переменная используется для хранения текстуры для заднего фона;

– RectangleShape BackGround – прямоугольная область для заднего фона;

– TextBox *text – указатель на объект текстового поля, необходимо для ввода имени;

– string NamePlayer – переменная, которая хранит имя игрока;

– int level – переменная для выбора уровня;

– Font font – шрифт для текста;

– RectangleShape container – прямоугольная область для текста

заголовка;

– Text menuText – переменная для отображения текста на экране;

– map<string, Button *> buttons – контейнер для хранения кнопок в данном состояние с ключом string и указателем на кнопку;

– void InitBackGround() – метод загружающий текстуру для заднего фона;

– void InitFonts() – метод загружающий шрифт для текста;

– void InitKeyBinds() – метод как в базовом классе для загрузки клавиш, в данном случае пустой;

– void InitButtons() – метод загрузки кнопок для меню.

Данный класс содержит следующие публичные методы:

– LevelState(BaseStateData *BaseState) – метод создания состояния меню выбора уровня и ввода имени, на вход получает указатель на данные для состояния BaseStateData;

– void updateInput(const float &deltaTime) – метод обновления действий пользователя относительно времени, такой же как и в базовом классе State, входной параметр вещественное число, показывающее изменение времени;

– void updateButtons() – уникальный метод класса, обновление состояний кнопок и обновление поля ввода имени;

– void update(const float &deltaTime) – данный метод реализует соответствующий метод базового класса State, особенностью реализации является использования дополнительных методов обновления позиции мыши, обновление ввода пользователя и обновления кнопок;

– void renderButtons(RenderTarget &target) – данный метод прорисовывает кнопки, на вход получает RenderTarget, то есть область где рисовать;

– void render(RenderTarget *target=NULL) – данный метод реализует соответствующий метод базового класса State, отличается тем что прорисовывает несколько прямоугольников и текст, также рисуется текстовое

поле и имя игрока, а также кнопки, входной параметр RenderTarget*, то где необходимо показать состояние.

2.5 Класс RulesState

Данный класс содержит следующие частные поля и методы:

– Font font – переменная для шрифта;

– Text RulesText – переменная для текста, в котором прописано управление;

– Text HistoryText – переменная для текста, в котором прописана суть игры;

– Text HeadingText – переменная для текста заголовка состояния;

– Texture BackGrounTexture – текстура для заднего фона;

– RectangleShape BackGround – прямоугольная область для фона;

– RectangleShape heading – прямоугольная область для заголовка;

– RectangleShape rule – прямоугольная область для управления;

– RectangleShape history – прямоугольная область для справки;

– Button *button – кнопка для выхода;

– void InitBackGround() – метод загружающий текстуру для заднего фона;

– void InitFonts() – метод загружающий шрифт для текста;

– void InitKeyBinds() – метод как в базовом классе для загрузки клавиш, в данном случае пустой;

– void InitButtons() – метод загрузки кнопки для выхода.

Данный класс содержит следующие публичны методы:

– RulesState(BaseStateData *BaseState) – метод создания состояния меню выбора уровня и ввода имени, на вход получает указатель на данные для состояния BaseStateData;

– void updateInput(const float &deltaTime) – метод обновления действий пользователя относительно времени, такой же как и в базовом классе State, входной параметр вещественное число, показывающее изменение времени;

– void updateButtons() – уникальный метод класса, обновление состояний кнопок и обновление поля ввода имени;

– void update(const float &deltaTime) – данный метод реализует соответствующий метод базового класса State, особенностью реализации является использования дополнительных методов обновления позиции мыши, обновление ввода пользователя и обновления кнопок;

– void renderButtons(RenderTarget &target) – данный метод прорисовывает кнопку, на вход получает RenderTarget, то есть область где рисовать;

– void render(RenderTarget *target=NULL) – данный метод реализует соответствующий метод базового класса State, отличается тем что прорисовывает несколько прямоугольников и текст, а также кнопки, входной параметр RenderTarget*, то где необходимо показать состояние.

2.6 Класс ResultState

Данный класс содержит следующие частные поля и методы:

– Texture BackGroundTexture – переменная используется для хранения текстуры для заднего фона;

– RectangleShape BackGround – прямоугольная область для заднего фона;

– int level – переменная для выбора уровня;

– Font font – шрифт для текста;

– RectangleShape container – прямоугольная область для текста заголовка;

– Text menuText – переменная для отображения текста на экране;

– map<string, Button *> buttons – контейнер для хранения кнопок в данном состояние с ключом string и указателем на кнопку;

– void InitBackGround() – метод загружающий текстуру для заднего фона;

– void InitFonts() – метод загружающий шрифт для текста;

– void InitKeyBinds() – метод как в базовом классе для загрузки клавиш, в данном случае пустой;

– void InitButtons() – метод загрузки кнопок для меню.

Данный класс содержит следующие публичные методы:

– ResultState(BaseStateData *BaseState) – метод создания состояния меню выбора уровня, на вход получает указатель на данные для состояния BaseStateData;

– void updateInput(const float &deltaTime) – метод обновления действий пользователя относительно времени, такой же как и в базовом классе State, входной параметр вещественное число, показывающее изменение времени;

– void updateButtons() – уникальный метод класса, обновление состояний кнопок;

– void update(const float &deltaTime) – данный метод реализует соответствующий метод базового класса State, особенностью реализации является использования дополнительных методов обновления позиции мыши, обновления кнопок;

– void renderButtons(RenderTarget &target) – данный метод прорисовывает кнопки, на вход получает RenderTarget, то есть область где рисовать;

– void render(RenderTarget *target=NULL) – данный метод реализует соответствующий метод базового класса State, отличается тем что прорисовывает несколько прямоугольников и текст, а также кнопки, входной параметр RenderTarget*, то где необходимо показать состояние.

2.7 Класс TableState

Данный класс содержит следующие публичные поля и методы:

– Font font – шрифт для текста;

– Text menuText – текст для заголовка состояния;

– Text head – текст для заголовка таблицы;

– Texture BackGrounTexture – текстура для заднего фона;

– RectangleShape BackGround – прямоугольная область для заднего фона;

– int level – переменная, хранящая уровень, по которому показывать результаты;

– int count – переменная хранит количество записей в файле результатов;

– RectangleShape container – прямоугольная область для текста заголовка;

– RectangleShape heading – прямоугольная область для текста заголовка таблицы;

– vector<node> fromFile – массив структурных записей, состоящих из имени игрока, уровня и времени прохождения, используется для создания таблицы;

– vector<RectangleShape> results – массив прямоугольных областей для записи туда текстов результатов;

– vector<Text> res – массив текстов для записи в табличном виде;

– map<string, Button *> buttons – контейнер для хранения кнопок в данном состояние с ключом string и указателем на кнопку;

– void InitBackGround() – метод загружающий текстуру для заднего фона;

– void InitFonts() – метод загружающий шрифт для текста;

– void InitKeyBinds() – метод как в базовом классе для загрузки клавиш, в данном случае пустой;

– void InitButtons() – метод загрузки кнопок для действий пользователя.

Данный класс содержит следующие публичные методы:

– TableState(BaseStateData *BaseState, cons tint &level) – метод создания состояния меню выбора уровня, на вход получает указатель на данные для состояния BaseStateData и переменная типа int, которая показывает какой уровень пользователь выбрал для просмотра результатов;

– void updateInput(const float &deltaTime) – метод обновления действий пользователя относительно времени, такой же как и базовый класс State, входной параметр вещественное число, показывающее изменение времени;

– void updateButtons() – уникальный метод класса, обновление состояний кнопок;

– void update(const float &deltaTime) – данный метод реализует соответствующий метод базового класса State, особенностью реализации является использования дополнительных методов обновления позиции мыши, обновление ввода пользователя и обновления кнопок;

– void renderButtons(RenderTarget &target) – данный метод прорисовывает кнопки, на вход получает RenderTarget, то есть область где рисовать;

– void clearTable() – данный метод позволяет очистить результаты на экране, и очистить содержимое файла;

– void createTxt() – метод для создания txt файла для печати;

– void render(RenderTarget *target=NULL) – данный метод реализует соответствующий метод базового класса State, отличается тем что прорисовывает несколько прямоугольников и текст, а также кнопки, входной параметр RenderTarget*, то где необходимо показать состояние.

2.8 Класс GameState

Данный класс содержит следующие частные поля и методы:

– Font font – шрифт для паузы;

– PauseMenu *pause_menu – меню для паузы;

– Player *player – объект класса игрок, используется как основной персонаж для игры;

– Texture texture – текстура для анимации игрока;

– TileMap *map – объект карты для игры;

– string NamePlayer – переменная для имени игрока;

– int level – переменная для загрузки карты по уровню, хранит

порядковый номер уровня;

– Clock time1 – таймер для подсчета времени прохождения;

– float timeGame – переменная, в которую записывается время прохождения игры;

– View view –вид, для передвижения камеры за игроком;

– void InitView() – метод создания вида;

– void InitKeyBinds() – метод добавления клавиш для игры;

– void InitTextures() – метод для загрузки текстуры мыши;

– void InitPauseMenu() – метод для создания меню паузы;

– void InitPlayers() – создание игрока – мыши, для игры;

– void InitFonts() – метод загрузки шрифта, для текста в меню паузы;

– void InitTileMap() – метод для загрузки карты из файла.

Данный класс содержит следующие публичные методы:

– GameState(BaseStateData *BaseState, string name, const int level) – метод создания игрового состояния, на вход получает указатель на данные для состояния BaseStateData и переменная типа int, которая показывает какой уровень пользователь выбрал для прохождения, и также передается имя игрока, для записи конечных результатов после прохождения;

– void updateTime() – уникальный метод для подсчета времени прохождения уровня;

– void updateView(const float &deltaTime) – уникальный метод обновление вида в зависимости от мыши, получает на вход изменение времени;

– void updateInput(const float &deltaTime) – метод отличается от базового класса State, обновляет состояние игры в зависимости от времени, изменяет состояние на паузу при нажатии клавиши;

– void updatePauseButtons() – уникальный метод, обновляет состояние кнопок в меню паузы;

– void updatePlayer(const float &deltaTime) – уникальный метод, который позволяет, управлять игроком, входной параметр вещественное

число изменение времени, в зависимости от него происходит движение игрока;

– void update(const float &deltaTime) – данный метод реализует соответствующий метод базового класса State, отличающиеся тем, что происходит обновление в зависимости от изменения времени, также обновление происходит в зависимости от состояния игрового процесса, также происходит обновление столкновение с картой и стенами, и постоянно происходит проверка окончания игры, при прохождение уровня результаты записываются в файл;

– void render(RenderTarget *target) – данный метод реализует соответствующий метод базового класса State, отличается тем ,что прорисовывает карту, игрока, при нажатии паузы показывает меню паузы, устанавливает вид в зависимости от положения игрока, входной параметр RenderTarget*, для прорисовки на экране.

2.9 Класс WinState

Данный класс содержит следующие частные поля и методы:

– Texture BackGroundTexture – переменная используется для хранения текстуры для заднего фона;

– RectangleShape BackGround – прямоугольная область для заднего фона;

– Font font – шрифт для текста;

– RectangleShape container – прямоугольная область для текста;

– RectangleShape results – прямоугольная область для записи результатов;

– string NamePlayer – переменная, хранящая имя игрока;

– int level – переменная, хранящая уровень, который пользователь проходил;

– Text text – переменная для отображения текста на экране;

– map<string, Button *> buttons – контейнер для хранения кнопок в данном состояние с ключом string и указателем на кнопку;

– void InitBackGround() – метод загружающий текстуру для заднего фона;

– void InitFonts() – метод загружающий шрифт для текста;

– void InitKeyBinds() – метод как в базовом классе для загрузки клавиш, в данном случае пустой;

– void InitButtons() – метод загрузки кнопок для дальнейших действий.

Данный класс содержит следующие публичные методы:

– WinState(BaseStateData *BaseState, const int &level, const float time, const string name) – метод создания состояния прохождения уровня, на вход получает указатель на данные для состояния BaseStateData, также получает номер уровня типа int,вещественное число время прохождения уровня, и строка с именем игрока;

– void updateInput(const float &deltaTime) – метод обновления действий пользователя относительно времени, такой же как и в базовом классе State, входной параметр вещественное число, показывающее изменение времени;

– void updateButtons() – уникальный метод класса, обновление состояний кнопок;

– void update(const float &deltaTime) – данный метод реализует соответствующий метод базового класса State, особенностью реализации является использования дополнительных методов обновления позиции мыши, обновление кнопок;

– void renderButtons(RenderTarget &target) – данный метод прорисовывает кнопки, на вход получает RenderTarget, то есть область где рисовать;

– void render(RenderTarget *target=NULL) – данный метод реализует соответствующий метод базового класса State, отличается тем что прорисовывает несколько прямоугольников и текст, а также кнопки, входной параметр RenderTarget*,указатель на то где необходимо показать состояние.

2.10 Класс PauseMenu

Данный класс содержит следующие частные поля:

– Font & font – ссылка на шрифт, для текста в мню паузы;

– Text menuText – текст для заголовка в меню паузы;

– RectangleShape background – прямоугольная область для заднего фона;

– RectangleShape container – прямоугольная область для записи текста;

– map<string, Button *> buttons – контейнер для хранения кнопок в данном состояние с ключом string и указателем на кнопку.

Данный класс содержит следующие публичные методы:

– PauseMenu(RenderWindow &window, Font &font) – метод создания меню паузы, на вход получает ссылку на окно, и ссылку на шрифт для текста;

– const bool isButtonPressed(const string key) – метод, который возвращает нажата ли кнопка или нет, на вход получает строковый ключ кнопки;

– void AddButton(const string key, float y, const string Text) – метод добавления кнопки в меню паузы, на вход получает ключ для кнопки, координату по y, и текст для надписи на кнопке;

– void update(const Vector2f &mousePos) – метод обновления меню паузы, относительно позиции мыши;

– void render(RenderTarget &target) – функция прорисовки меню паузы, на вход получает ссылку на область для прорисовки.

2.11 Класс TextBox

Данный класс содержит следующие частные поля и методы:

– bool pressed – переменная для состояния нажатия поля для ввода;

– RectangleShape shape – прямоугольная область для текстового поля;

– Font *font – указатель на шрифт, для букв в текстовом полу;

– Text text – текст для кнопки;

– map<int, string> keys – контейнер, хранящий клавиши для ввода имени, ключ – порядковый номер в перечисление Keyboard, и значение string, символ для соответствующего ключа;

– vector<bool> flags – булевый массив, для хранения флагов нажатия кнопок, для контроля одного нажатия клавиши;

– int max – переменная, которая хранит максимальное число символов в текстовом поле;

– int cur – переменная, которая хранит в себе текущее состояние символов в текстовом поле;

– string name – строка, которая хранит имя игрока, которое вводится в текстовом поле;

– Color idleColor – цвет контейнера для текстового поля;

– Color textIdCol – цвет для вводимого текста;

– void IntitKeys() – метод загрузки клавиш, для ввода имени;

– void InitVector() – метод обнуляющий массив флагов.

Данный класс содержит следующие публичные методы:

– TextBox(float x, float y, float width, float height, Font *font, unsigned sizeText, Color BoxColor, Color TextColor, bool pressed) – метод создания текстового поля, получает на вход координаты x и y, ширину и высоту текстового поля типа float, указатель на шрифт для текста, дальше размер символов текста, получает еще два цвета, один для текста, второй для цвета, и состояние текстового поля;

– void addChar(string key) – метод добавления символа к вводимому тексту, на вход получает строку из одного символа, который необходимо добавить;

– void deleteChar() – метод удаляющий символ из вводимого текста;

– void KeyPressed(int i, const bool &x) – метод который проверяет одно нажатие на клавишу, исключая лишние действия, на вход получает

числовой ключ символа, для получений значения в контейнере клавиш, и получает также значение нажата ли какая – либо клавиша или нет;

– string getName() – метод возвращающий строку, из текстового поля;

– void update(Vector2f mousePos) – метод обновляющий, состояние нажатия на текстовое поле, обрабатывает вводимые пользователем символы, получает на вход позиции мыши, для обработки выбора пользователем текстового поля;

– void render(RenderTarget &target) – метод прорисовки текстового поля, и текста в нем, на вход получает ссылку на область для прорисовки.

2.12 Класс Button

Данный класс содержит следующие частные поля:

– short unsigned buttonState – переменная хранит состояние кнопки;

– RectangleShape shape – прямоугольная основа для кнопки;

– Font *font – указатель на шрифт, для хранения шрифта для текста внутри кнопок;

– Text text – текст для кнопки;

– Color idleColor – цвет прямоугольной области без наведения мыши;

– Color hoverColor – цвет прямоугольной области при наведении мыши;

– Color activeColor – цвет прямоугольной области при нажатой кнопке;

– Color textIdCol – цвет текста без наведения мыши;

– Color textHovCol – цвет текста при наведении мыши;

– Color textActCol – цвет текста при нажатой кнопке;

– enum button_states – перечисление используется для хранения состояний кнопки, без наведения мыши, при наведении мыши, при нажатии на кнопку.

Перечисление содержит следующие состояния: BTN_IDLE = 0, BTN_HOVER, BTN_ACTIVE.

Данный класс содержит следующие публичные методы:

– Button(float x, float y, float width, float height, Font *font, const string Text, unsigned sizeText, Color textIdCol, Color textHovCol, Color textActCol, Color idleColor, Color hoverColor, Color activeColor) – метод для создания кнопки, получает на вход координаты x и y типа float, ширину и высоту кнопки типа float, указатель на шрифт для текста, получает строку для текста внутри кнопки, размер символов для отображения типа unsigned, три цвета для текста по цвету для каждого состояния, и также три цвета для прямоугольника кнопки;

– const bool isPressed() const – метод который возвращает состояние кнопки, нажата она или нет;

– void update(Vector2f mousePos) – метод обновления кнопки, на вход получает позицию мыши;

– void render(RenderTarget & target) – метод прорисовки кнопки, и текста в нем, на вход получает ссылку на область для прорисовки.

2.13 Класс Tile

Данный класс содержит следующие частные поля:

– Sprite tile – переменная спрайта для элемента карты;

– short type – переменная, которая хранит тип элемента карты;

– bool collision – переменная, которая хранит наличие коллизии у элемента карты.

Данный класс содержит следующие публичные методы:

– Tile() – метод создания элемента карты;

– Tile(float x, float y, float gridSizeF, Texture &texture, int inx, int iny, int type, bool collision) – метод создания элемента карты, получает на вход координаты типа float, чтобы расположить элементы карты на карте, размер одного элемента типа float, текстура для заполнения элемента карты текстурой, начальная координаты x и y на текстуре типа int, тип элемента

карты типа int, и наличие коллизии типа bool;

– const short &getType() const – метод возвращающий тип элемента карты;

– const bool &getCollision() const – метод возвращающий значение поля коллизии;

– const Vector2f &getPosition() const – метод возвращающий, ссылку на координаты элемента карты;

– const FloatRect getGloabalBounds() const – метод возвращает прямоугольную область элемента карты;

– const bool intersects(const FloatRect bounds) const – метод возвращает пересечение элемента карты с другим объектом, получает на вход прямоугольную область типа float;

– void render(RenderTarget &target) – метод прорисовки элемента карты, на вход получает ссылку на область для показа карты.

2.14 Класс TileMap

Данный класс содержит следующие частные поля и методы:

– void clearTileMap() – метод очистки игровой карты;

– Texture tliesheet – переменная, для хранения текстур, для создания карты;

– float gridSizeF – размер сетки для одного элемента карты типа float;

– unsigned gridSize – размер сетки для одного элемента карты типа unsigned int;

– Vector2f maxWorldSize – переменная которая хранит максимальный размер карты;

– Vector2u maxSize – переменная, которая хранит количество элементов карты, размерность игровой сетки;

– vector<vector<vector<Tile*>>> map – трехмерный массив элементов карты, для записи, хранения, и использования, игровой карты;

– unsigned layers – переменная, которая хранит количество слоев в

карте;

– string tilename – строка, которая хранит имя файла с текстурами для элементов карты.

Данный класс содержит следующие публичные методы:

– TileMap() – метод создания карты без параметров;

– TileMap(const string filename) – метод загрузки игровой карты из файла, на вход получает строку, содержащую имя файла, откуда надо загрузить;

– constVector2f &getMaxSizeF() – метод возвращает максимальные размеры игровой карты в формате Vector2f для удобного использования;

– void updateWorldCollision(Entity *player) – метод, который обрабатывает столкновения игрока и границ игровой карты, на вход поступает указатель на объект игрока, для доступа к его перемещению;

– void updateWorldCollision(Entity *player, const float& deltaTime) – метод, который обрабатывает столкновения игрока и стен лабиринта, на вход поступает указатель на объект игрока, для доступа к его перемещению, и вещественное число изменения времени, для получений возможной следующей позиции игрока;

– void render(RenderTarget &target) – метод прорисовки карты, на вход получает ссылку на область для показа карты.

2.15 Класс Entity

Данный класс содержит следующий частный метод void InitVariables() – метод для обнуления компонентов движения объекта.

Данный метод имеет следующие защищенные поля:

– Sprite sprite – переменная для движущегося объекта сущности;

– Movement *movement – переменная для движения объекта;

– AnimationPlayer *animation – переменная для анимации объекта;

– CollisionComponent *hitbox – переменная для столкновения объекта.

Данный метод имеет следующие публичные методы:

– Entity() – метод создания движущегося объекта;

– void SetTexture(Texture &texture) – метод установки текстуры для движущегося объекта, на вход получает ссылку на текстуру;

– void CreateMovement(const float MaxVelocity, const float acceleration, const float deceleration) – метод создания движения для объекта, на вход поступает три вещественных числа, максимальная скорость, ускорение и замедление;

– void CreateAnimation(Texture &texturesheet) – метод создания анимации для объекта, на вход поступает ссылка на текстуру с объектом;

– void CreateHitBox(Spritte &sprite, float sx, float sy, float width, float height) – метод создания контейнера вокруг объекта для обработки столкновений, на вход получает ссылку на спрайт, начальные координаты для контейнера вокруг спрайта типа float, и размеры прямоугольной области типа float;

– const Vector2f &getPosition() const – виртуальный метод получений координат объекта;

– const Vector2f &getPositionSprite() const – виртуальный метод получения позиции спрайта объекта;

– const FloatRect getGlobalBounds() const – виртуальный метод получений прямоугольника для обработки перечечений, возвращает значение FloatRect;

– const Vector2u getGridPosition(const unsigned gridSize) const – виртуальный метод получения позции объекта в сетке карты;

– const FloatRect getNextBounds(const float &deltaTime) const – виртуальный метод получения следующей прямоугольной области объекта в зависимости от времени, на вход получ——ает вещественное число изменение времени;

– void SetPosition(const float x, const float y) – виртуальный метод установки позиции на экране для движущегося объекта, на вход получает два вещественных числа, координаты для позиции;

– void move(const float dx, const float dy, const float &deltaTime) – виртуальный метод движения объекта, на вход получает два вещественных числа, изменения по координатам x и y, и ссылку на значение изменения времени для движения;

– void stopY() – виртуальный метод остановки движения объекта по y;

– void stopX() – виртуальный метод остановки движения объекта по x;

– void update(const float &deltaTime) – виртуальный метод обновления движения объекта, на вход получает ссылку на значение изменения времени;

– void render(RenderTarget &target) – виртуальный метод прорисовки движущегося объекта.

2.16 Класс Player

Данный класс содержит следующие частные методы:

– Player(float x, float y, Texture &textureSheet) – метод создания, игрока, на вход получает два вещественных числа – координаты для расположения игрока, и текстуру для анимации;

– void update(const float &deltatime) – данный метод реализует соответствующий метод базового класса Entity, отличается тем, что обновляет движение игрока и анимацию движения.

2.17 Класс Animation

Данный класс имеет следующие публичные поля и методы:

– Texture &texturesheet – переменная, которая хранит ссылку на текстуру с изображениями для анимации;

– Sprite &sprite – переменная, которая хранит ссылку на спрайт, в который помещаются изображения;

– float timer – переменная для хранения времени и изменения времени при переключение изображений анимации;

– float speedTimer – переменная, которая хранит скорость

переключения анимации;

– IntRect BeginRect – прямоугольная область для первого изображения анимации;

– IntRect CurRect – прямоугольная область для текущего изображения анимации;

– IntRect EndRect – прямоугольная область для конечного изображения анимции;

– Animation(Sprite &sprite, Texture &textureSheet, float speed, int BeginFrameX, int BeginFrameY, int FrameX, int FrameY, int width, int height) – метод для создания анимации, получает ссылку на спрайт, и ссылку на текстуру с изображениями, скорость для переключения анимаций типа float, номера для первой прямоугольной области и, и количество изображений для анимации на текстуре типа int по x и y, размеры этой области типа int;

– void play(const float &deltaTime) – метод переключения изображений в анимации, получает на вход ссылку на изменение времени, чтобы изменять в зависимости от времени;

– void reset() – метод для того чтобы начать анимацию сначала.

2.18 Класс AnimationPlayer

Данный класс содержит следующие частные поля:

– Sprite &sprite – переменная, которая хранит ссылку на спрайт который будет анимироваться;

– Texture &textureSheet – переменная, которая хранит ссылку на текстуру с изображениями для анимации;

– map<string, Animation *> animations – контейнер для хранения анимации по строковому ключу;

– Animation *lastAnimation – указатель на анимацию для использования анимации из map.

Данный класс содержит следующие публичные методы:

– AnimationPlayer(Sprite & sprite, Texture &texturesheet) – метод создания анимации игрока, получает на вход ссылку на спрайт и ссылку на

текстуру;

– void AddAnimation(string key, , float speed, int BeginFrameX, int BeginFrameY, int FrameX, int FrameY, int width, int height) – метод добавления анимации, получает на вход строковый ключ, скорость переключения анимаций типа float, номера первой текстуры типа int по x и y, количество изображений для анимации по x и y, размеры прямоугольника с изображением для анимации;

– void play(const string key, const float &deltaTime) – метод переключения анимации, получает на вход строковый ключ для анимации, и ссылку на значение изменения времени для переключения.

2.19 Класс Movement

Данный класс содержит следующие частные поля:

– Sprite &sprite – переменная, которая хранит ссылку на движущейся объект;

– Float MaxVelocity – переменная, которая хранит значение максимальной скорости движения игрока;

– Vector2f Velocity – переменная, которая хранит скорость по двум координатам;

– float acceleration – переменная, которая хранит значение ускорения при движение игрока;

– float deceleration – переменная, которая хранит значение замедления при движение игрока.

Данный класс содержит следующие публичные методы:

– Movement(Sprite &sprite, float maxVelocity, float acceleration, float deceleration) – метод создания элементов для движения, на вход получает ссылку на движущейся объект, максимальную скорость движения типа float, значение ускорения и замедления типа float;

– void move(const float dx, const float dy, const float &deltaTime) – метод движения объекта, на вход получает два вещественных числа, изменения по координатам x и y, и ссылку на значение изменения времени для

движения;

– const bool getState(const short unsigned state) const – данный метод возвращает состояние движение, получает на вход значение типа движения short unsigned;

– void stop() – метод остановки движения объекта;

– void stopY() – метод остановки движения объекта по y;

– void stopX() – метод остановки движения объекта по x;

– void update(const float &deltaTime) – метод обновления движения объекта, на вход получает ссылку на значение изменения времени.

2.20 Класс CollisionComponent

Данный класс содержит следующие частные методы:

– Sprite &sprite – переменная, которая хранит ссылку на движущейся объект;

– RectangleShape shape – переменная, которая хранит прямоугольник для проверки столкновений с объектами;

– float SX – координата x для контейнера на области спрайта;

– float SY – координата y для контейнера на области спрайта;

– FloatRect nextPos – переменная, которая хранит следующую прямоугольную область для обработки столкновений.

Данный класс содержит следующие публичные методы:

– CollisionComponent(Sprite &sprite, float sx, float sy, float width, float height) – метод создания компонентов для столкновений, получает на вход ссылку на спрайт, координаты для контейнера на спрайте типа float, и размеры для этого контейнера;

– const Vector2f &getPosition() const – данный метод возвращает координаты контейнера для столкновений;

– const FloatRect getGlobalBounds() const – данный метод возвращает;

– const FloatRect &getNextPosition(const sf::Vector2f &velocity) –

возвращает прямоугольную область при следующей позиции контейнера для столкновений, получает на вход получает скорость передвижения игрока типа Vector2f;

– void setPosition(const sf::Vector2f &position) – метод установки значения для контейнера по входному параметру типа Vector2f;

– void setPosition(const float x, const float y) – метод установки позиции для прямоугольной области, по двум координатам типа float;

– bool checkIntersect(const FloatRect &rect) – метод который проверяет столкновение данной прямоугольного контейнера для столкновений с другой прямоугольной областью, получает на вход ссылку на другую прямоугольную область типа FloatRect;

– void update() – метод обновления позиции контейнера для столкновений;

– void render(RenderTarget &target) – метод прорисовки данного контейнера для столкновений, получает на вход ссылку на область для прорисовки, типа RenderTarget &.

2.21 Класс Game

Данный класс содержит следующие частные поля и методы:

– Texture BackGrounTexture – текстура для фона заставки;

– RectangleShape BackGround – прямоугольная область для заднего фона;

– Font font – шрифт для текста;

– RectangleShape container – прямоугольный контейнер для текста;

– Text text – переменная для текста;

– Music music – переменная для музыки;

– RenderWindow *window – указатель на окно, для игры;

– Event event – переменная для событий;

– BaseStateData BaseState – переменная для записи основных данных для реализации состояний;

– bool fullscreen – переменная, для состояния окна;

– Clock timer – переменная для подсчета времени показа заставки;

– float time – переменная для хранения времени для заставки;

– Clock deltaClock – переменная, для измерения изменения времени;

– float deltaTime – переменная, для получения изменения времени;

– float gridSize – переменная с размером сетки;

– stack<State *> situations – стек для хранения состояний игры;

– map<string, int> SupportKeys – контейнер из клавиш для игры, с ключом string и значением типа int;

– void InitWindow() – метод создания окна;

– void InitFont() – метод для загрузки шрифта для заставки;

– void InitPreview() – метод для создания окна;

– void InitBaseState() – метод создание базового состояния;

– void InitKeys() – метод загрузки клавиш для игры;

– void InitStates() – метод для создания первого состояния;

– void InitVariables() – метод изначального обнуления окна игры и времени;

– void InitMusic() – метод загрузки музыки и подключения.

Данный класс содержит следующие публичные методы:

– Game() – конструктор игры, с загрузкой всего необходимого для игры;

– void UpdateDeltaTime() – метод обновления изменения времени, для обновления кадров игры;

– void update() – метод обновления состояний игры;

– void render() – метод прорисовки состояний для игры;

– void run() – метод игры, в котором происходит обновление и прорисовка состояний;

– void updateSFMLEvents() – метод обновления событий.

2.22 Демонстрация иерархии классов

Разработанные иерархии классов наглядно продемонстрированы на диаграммах классов, данные диаграммы представлены на рисунках 1 и 2.

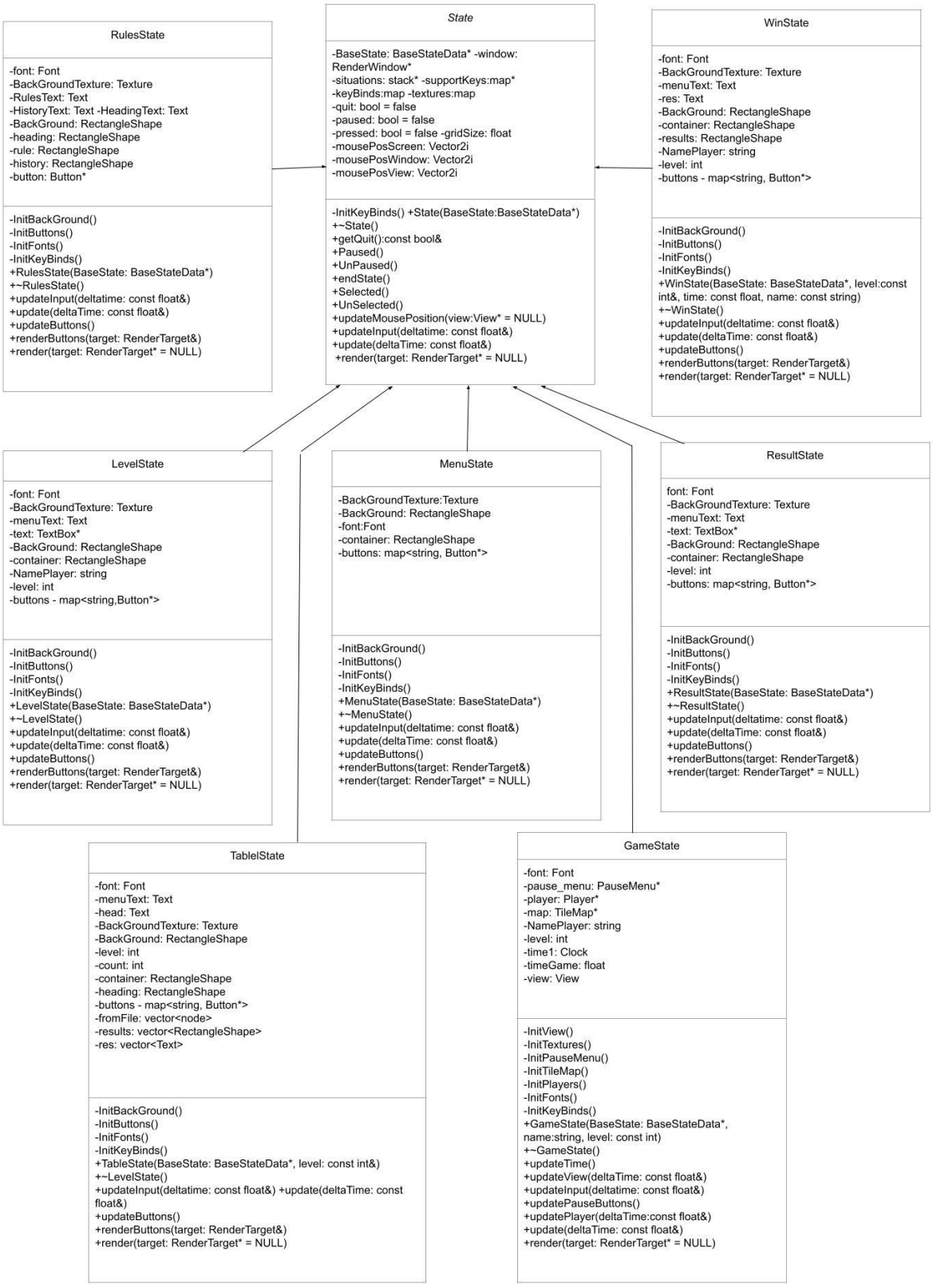

Рисунок 1 – Иерархия классов состояний игры

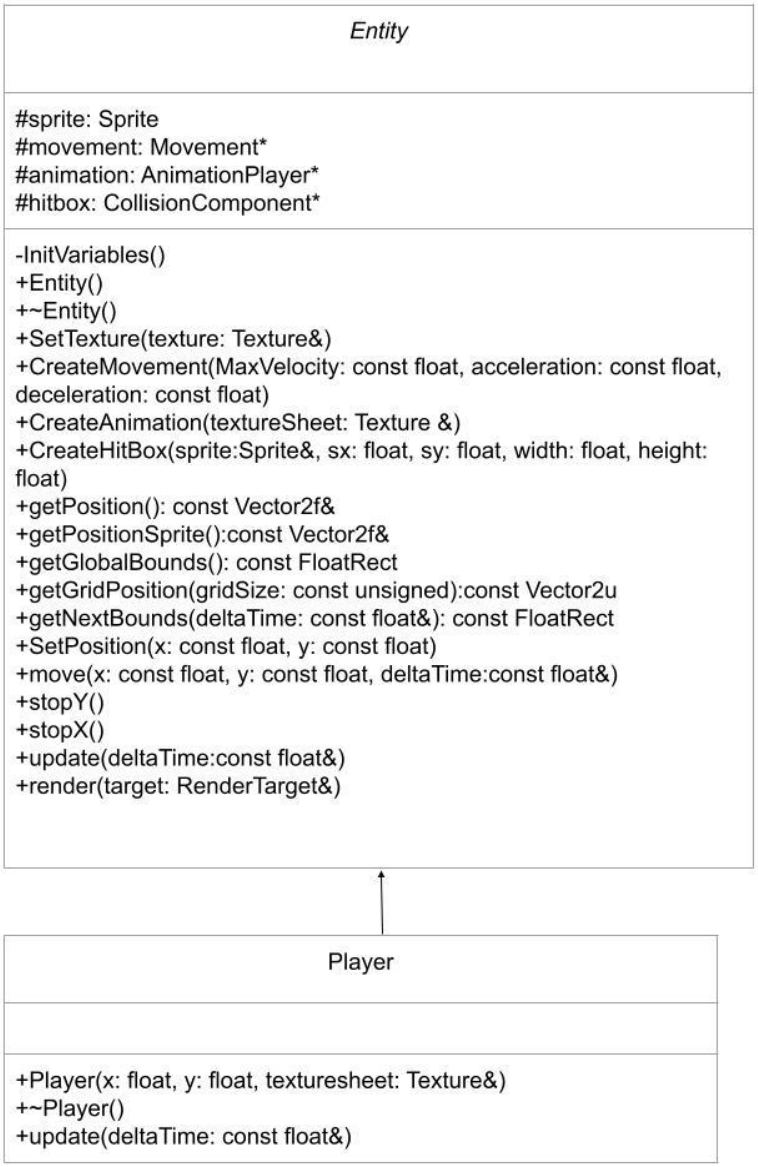

Рисунок 2 – Иерархия классов движущегося объекта

3 Использованные мультимедийные ресурсы и сторонние библиотеки

В данной игре используются следующие мультимедийные ресурсы:

– шрифт для текста в игре в папке Font с именем 1font.ttf [3];

– музыка для игры хранится в папке Resourses с именем music.ogg [4];

– изображение для анимации движения мыши хранится в папке Resourses с именем spritesheet.png [5];

– изображения с текстурами для элементов карты, хранятся в папке Resourses с названиями firstlevel.png и texturelevel.png;

– изображения для задних фонов различных состояний хранятся в папке Resourses с именами 2.png, level.png, win.png.

Для создания игровых карт лабиринтов, использовалось приложение Tiled [6].

4 Демонстрация работы

В разработанной игре, во время загрузки игры, показывается заставка, что показано на рисунке 3.

Рисунок 3 – Заставка перед игрой

После загрузки меню, на экране отображается игровое меню, что показано на рисунке 4.

Рисунок 4 – Главное меню

При выборе пункта «Новая игра», пользователь попадает в подменю с

выбором уровня и ввода имени для игры, что показано на рисунках 5 и 6.

Рисунок 5 – Подменю с выбором уровня игры и не активным вводом имени игрока

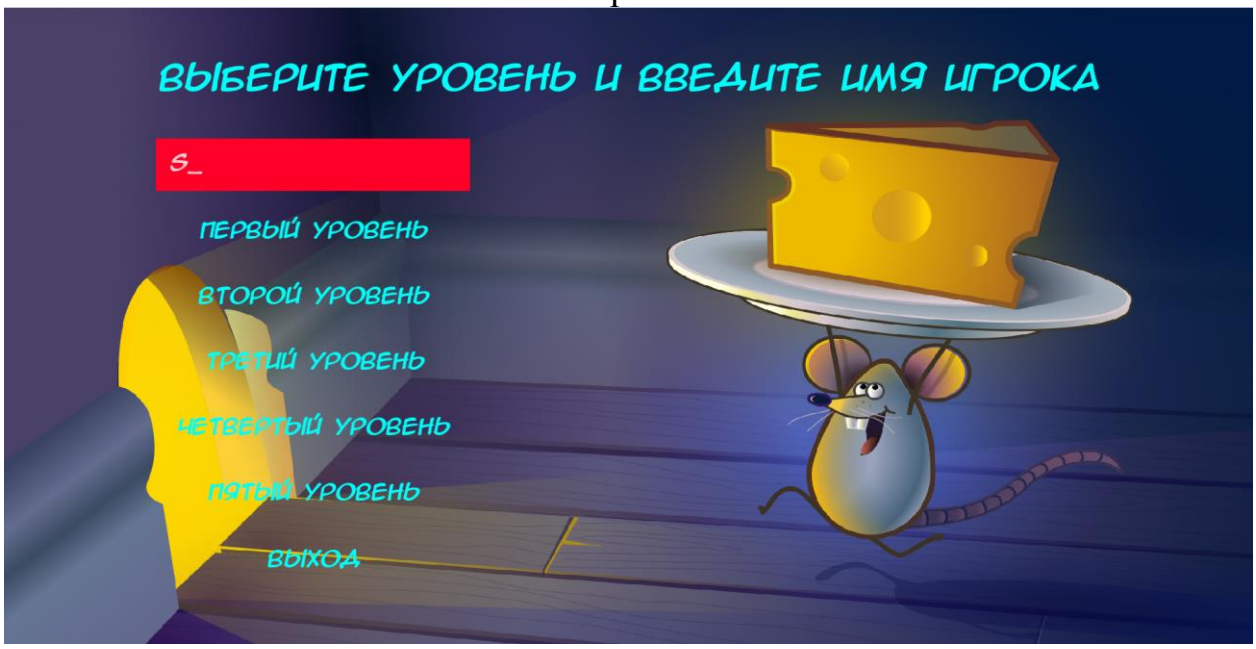

Рисунок 6 – Подменю выбора уровня игры и активным вводом имени игрока

При выборе пункта «Правила игры и справка», на экран выводятся правила игры и справка, что показано на рисунке 7.

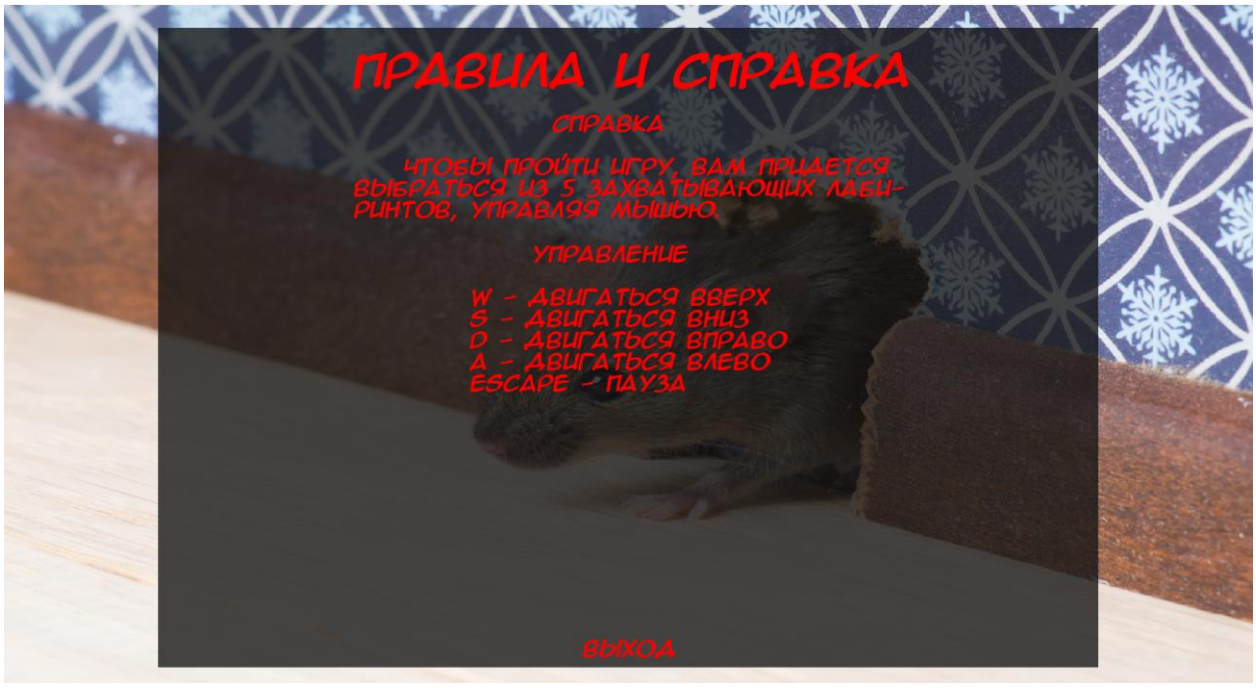

Рисунок 7 – Подменю правила и справка

При выборе пункта «Результаты», на экран выводится подменю с выбором уровня для просмотра результатов, что показано на рисунке 8.

Рисунок 8 – Подменю выбора уровня для просмотра результатов

После выбора уровня в подменю «Результаты», пользователю на экран выводятся результаты прохождения выбранного уровня, отсортированные от лучшего к худшему, если результатов по уровню нет, то выводится сообщение об этом, что показано на рисунках 9 и 10.

Рисунок 9 – Результаты в табличном виде

Рисунок 10 – Если результатов по уровню еще нет

После ввода имени игрока и выбора уровня для прохождения, начинается игровой процесс, в котором отображается игровая карта, и мышь, что показано на рисунке 11, 12.

Рисунок 11 – Мышь и часть игровой карты

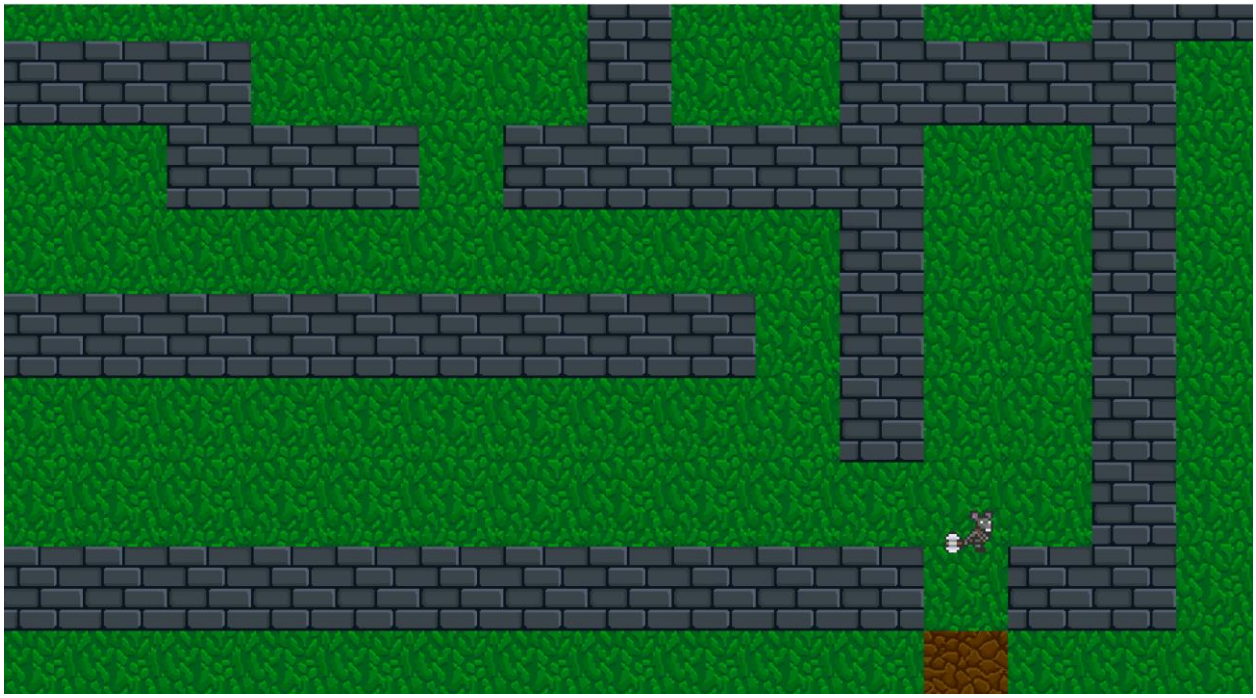

Рисунок 12 – Мышь перед выходом из лабиринта

Также во время игрового процесса, есть пауза, при нажатии кнопки паузы, появляется меню с возможностью выхода и продолжением игры, что показано на рисунке 13.

Рисунок 13 – Меню паузы

После прохождения уровня, на экран выводится имя игрока, номер уровня и время прохождения, что показано на рисунке 14.

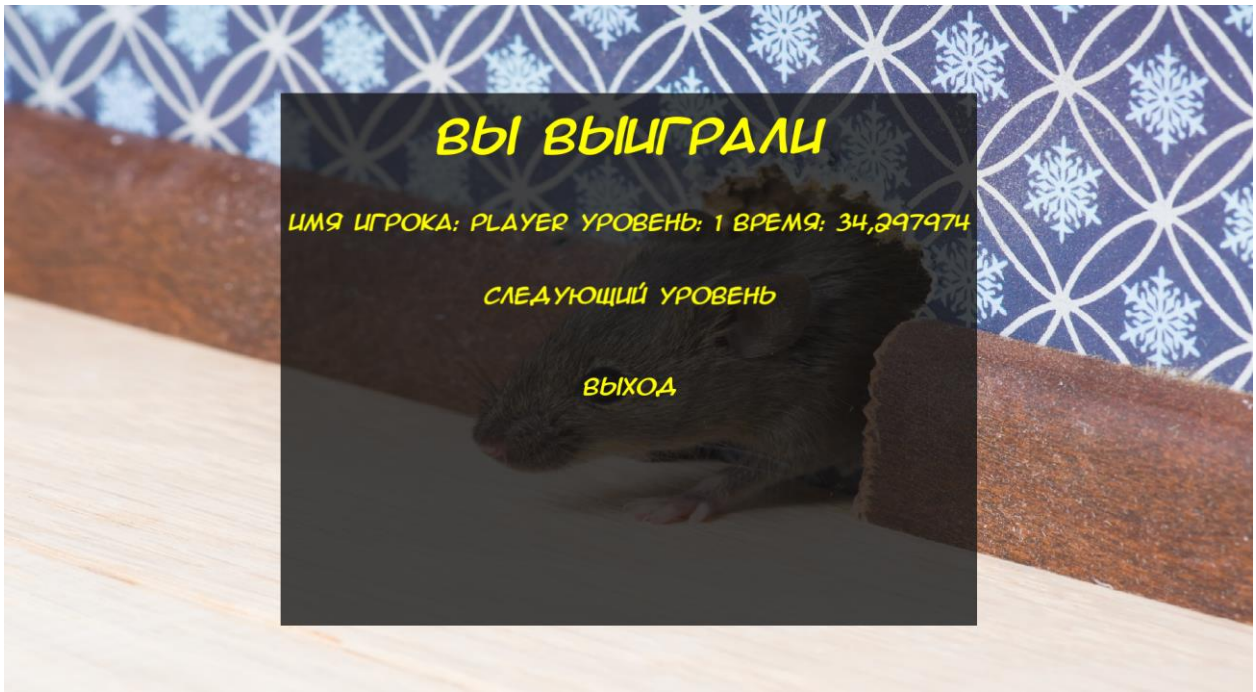

Рисунок 14 – Показ результатов после прохождения уровня

ЗАКЛЮЧЕНИЕ

В ходе работы были выполнены следующие действия по каждой из поставленных задач.

Были определены основные требования к игре «Лабиринт»: реализовано главное меню с подпунктами, создан управляемый анимированный персонаж, который не должен выходить за границы карты и проходить сквозь стены. Сам лабиринт должен превосходить размеры окна.

Была создана иерархия классов, включающая в себя классы для главного меню, различных состояний игры и движения персонажа.

Было разработано приложение с использованием библиотеки SFML, приложение включает в себя главное меню, игровой процесс, звуковое сопровождение, с самим приложением можно ознакомится в Приложении А.

Была продемонстрирована работоспособность приложения на примере игрового процесса.

Задачи выполнены, цель курсовой работы достигнута.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Документация по C++. Начало работы, руководства, справочные материалы. – URL: https://learn.microsoft.com/ru-RU/cpp/cpp/?view=msvc-170 (дата обращения 10.05.2023).

2. SFML. – URL: https://www.sfml-dev.org/ (дата обращения 15.05.2023).

3. Anime Ace v02 Italic Bold. – URL: https://old.fonts-online.ru/font/Anime-Ace-v02--Italic-Bold (дата обращения 28.05.2023).

4. Спокойная музыка для фона без слов. – URL: https://zvukipro.com/music/773-spokojna-muzyka-dlja-fona-bez-slov-chill.html (дата обращения 10.06.2023).

5. 2D Pixel Art Ratfolk Axe Sprites by Elthen's Pixel Art Shop. – URL: https://elthen.itch.io/2d-pixel-art-ratfolk-axe-sprites (дата обращения 01.06.2023).

6. Tiled | Flexible level editor. – URL: https://www.mapeditor.org/ (дата обращения 05.06.2023).

ПРИЛОЖЕНИЕ А

Текст программы

Исходные тексты программы располагаются на прилагаемом электронном носителе.