МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего образования
«Южно-Уральский государственный университет
(национальный исследовательский университет)»
Высшая школа электроники и компьютерных наук
Кафедра системного программирования

ДОПУСТИТЬ К ЗАЩИТЕ

Заведующий кафедрой, д.ф.-м.н.,
профессор

_____ Л.Б. Соколинский

«___»_____ 2024 г.

Разработка компьютерной игры в жанре «Roguelite» для платформы Windows

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
ЮУрГУ – 09.03.04.2024.308-561.ВКР

Научный руководитель,
доцент кафедры СП, к.т.н.
_____ Н.Ю. Долганина

Автор работы,
студент группы КЭ-403
_____ В.Р. Гумаров

Ученый секретарь
(нормоконтролер)
_____ И.Д. Володченко
«___»_____ 2024 г.

Челябинск, 2024 г.

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего образования
«Южно-Уральский государственный университет
(национальный исследовательский университет)»
Высшая школа электроники и компьютерных наук
Кафедра системного программирования

УТВЕРЖДАЮ
Зав. кафедрой СП
_____ Л.Б. Соколинский
29.01.2024 г.

ЗАДАНИЕ
на выполнение выпускной квалификационной работы бакалавра
студенту группы КЭ-403
Гумарову Вадиму Рифатовичу,
обучающемуся по направлению
09.03.04 «Программная инженерия»

1. **Тема работы** (утверждена приказом ректора от 22.04.2024 г. № 764-13/12) Разработка компьютерной игры в жанре «Roguelite» для платформы Windows.
2. **Срок сдачи студентом законченной работы:** 03.06.2024 г.
3. **Исходные данные к работе**

3.1. Роллингз Э., Моррис Д. Проектирование и архитектура игр. // Разработка архитектуры игр, СПб:Вильямс, 2006. – С. 485–818.

3.2. Страуструп Б. Язык программирования C++. // Механизмы абстракции, М: Бином, 2022. – С. 461–759.

4. **Перечень подлежащих разработке вопросов**

4.1. Провести анализ предметной области.

4.2. Спроектировать игровое приложение.

4.3. Реализовать игровое приложение.

4.4. Провести тестирование приложения.

5. **Дата выдачи задания:** 29.01.2024 г.

Научный руководитель,
доцент кафедры СП, к.т.н. Н.Ю. Долганина

Задание принял к исполнению В.Р. Гумаров

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Актуальность

На сегодняшний день видеоигры являются одним из главных направлений индустрии развлечений. Несмотря на спад прибыли за два года [1], данное направление все равно имеет большой потенциал роста.

Постановка задачи

Целью выпускной квалификационной работы является разработка компьютерной игры в жанре «Roguelite». Для достижения поставленной цели необходимо решить следующие задачи:

1) провести анализ предметной области;
2) спроектировать игровое приложение;
3) реализовать игровое приложение;
4) провести тестирование.

Структура и содержание работы

Работа состоит из введения, четырех глав, заключения и списка литературы. Объем работы составляет 51 страницу, объем списка литературы – 25 источников.

В первой главе, «Анализ предметной области», приводится краткое описание жанра и обзор наиболее популярных игр, похожих на создаваемую.

Во второй главе, «Проектирование», описаны требования в форме эскизного проекта, приведены макеты интерфейса приложения, составлены варианты использования и диаграмма компонентов системы.

В третьей главе, «Реализация» содержатся средства реализации игрового приложения, а также описание компонентов игрового приложения и их реализация.

Четвертая глава «Тестирование» посвящена тестированию созданного приложения.

В заключении сделаны выводы о проделанной работе.

В приложении содержатся листинги исходных кодов.

1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

1.1. Предметная область проекта

В ходе данной работы необходимо разработать двухмерную компьютерную игру в жанре roguelite. Жанр roguelite или «упрощенный рогалик» является поджанром жанра roguelike [2], вдохновленный Берлинской интерпретацией [3] roguelike, но с различными изменениями в механиках.

Берлинская интерпретация roguelike представляет собой поход к жанру видеоигр, вдохновленный классическими roguelike-играми, такими как Rogue [4] и Nethack [5].

Этот подход был определен игроками и разработчиками на конференции в Берлине в 2008 году.

Основные механики Берлинской интерпретации roguelike.

1. Процедурно-генерируемый мир: игровой мир создается случайным образом, увеличивая вариативность и повышая возможность повторного прохождения.

2. Перманентная смерть: если персонаж игрока умирает, игра начинается заново, и прошлый прогресс утрачивается.

3. Пошаговость: каждая команда соответствует одному действию/движению, что позволяет игроку принимать решения в своем темпе;

4. Сетка: мир представлен равномерной сеткой плиток, на каждую из которых помещается монстры и игрок.

5. Анти-модальность: действия, такие как движение и битва, происходят в одном режиме, обеспечивая доступность всех действий в любой момент.

6. Сложность: игра обладает достаточной сложностью, давая возможность найти несколько общих решений.

7. Управление ресурсами: игрок должен эффективно управлять ограниченными ресурсами, такими как еда и зелья, а также использовать их наилучшим образом.

8. Hack and slash: убийство множества монстров является важной частью геймплея. игра представляет собой концепцию игрок против мира: не существует отношений монстр/монстр (такие как дипломатия, вражда и т.д.) [6].

9. Исследование и открытия: требуется исследование уровней подземелий и обнаружение функционала нераспознанных предметов при каждом новом запуске игры.

Жанр roguelite сохраняет некоторые черты roguelike, но вносит дополнительные элементы и усовершенствования.

Он позволяет экспериментировать с идеями и механиками roguelike, предлагая более гибкий и менее жесткий опыт для игроков, сохраняя при этом некоторую степень вызова и непредсказуемости, характерных для классических roguelike-игр.

Например, жанр roguelite может предоставить короткие игровые периоды, в которые можно играть бесконечно. Краткость одного игрового процесса в roguelite может мотивировать игроков постоянно переигрывать игру.

Основные отличия roguelite от roguelike:

1) более динамичный и аркадный геймплей;

2) упрощенные или измененные механики по сравнению с традиционными roguelike;

3) присутствуют элементы постоянного прогресса, такие как улучшения или предметы, которые сохраняются между игровыми сессиями.

В данный момент этот жанр является очень распространенным и успешным, особенно на персональных компьютерах. Многие разработчики стараются удивить игрока смесью механик и игрового процесса. В соответствии с этим становится актуальной задача разработки компьютерной игры в жанре roguelite.

1.2. Анализ аналогичных проектов

Перед началом работы над проектом необходимо изучить существующие аналоги. С помощью данного анализа можно выделить особенности проектов, их преимущества и недостатки. В дальнейшем это поможет реализовать создаваемый проект.

В настоящее время существует множество успешных игровых проектов, разработанных в жанре roguelite. Поэтому среди аналогов были выделены следующие игры: Spelunky, Vampire Survivors, Hades.

Spelunky [7]

Spelunky – видеоигра в жанре roguelite, разработанная и выпущенная в 2008 году компанией Mossmouth [8] для системы Windows. Скриншот из игры представлен на рисунке 1.

Рисунок 1 – Игра Spelunky

Данный проект дал толчок развития данного жанра, так как был выпущен вскоре после формирования Берлинской интерпретации. В игре пе-

реплетаются механики платформера [9] и roguelike. Игра получила признание многих критиков и изданий, которые считали ее одной из величайших видеоигр всех времен [10].

Игрок перемещается по двумерному миру, разделенному на клетки. Каждый уровень генерируется случайно. Уровни сгруппированы в четыре зоны с возрастающей сложностью. Каждая зона имеет свой набор предметов, врагов, типов местности и особых элементов. В более поздних зонах можно найти более ценные сокровища, секретные локации и предметы. Если герой теряет все жизни или попадает в ловушку мгновенного убийства, ему придется начать заново.

Цель игры – исследовать туннели, собирая как можно больше сокровищ, избегая при этом ловушек и врагов чтобы в конце попасть в золотой город. Протагонист может атаковать врагов ударами или прыжками, подбирать предметы для метания или создания ловушек, а также использовать ограниченное количество бомб и веревок для перемещения по пещерам.

Игра выполнена в стилизованном графическом стиле, который очень подходит игре. Особенные игровые элементы легко заметны, как и неигровые персонажи, с которыми можно взаимодействовать.

Саундтрек игры расслабляющий и за долгое время не надоедает. На каждой зоне играет своя мелодия.

Достоинства игры:

1) затягивающий геймплей;
2) секреты и разблокируемый контент;
3) простое управление.

Недостатки игры:

1) сложность может отпугнуть некоторых игроков;
2) несправедливая смерть.

Vampire Survivors [11]

Vampire Survivors – видеоигра в жанре roguelite, разработанная и изданная Галанте Л. [12] в конце 2021 года. Скриншот из игры представлен на рисунке 2.

Рисунок 2 – Игра Vampire Survivals

Данная видеоигра была вдохновлена другой видеоигрой под названием Magic Survivor [13]. Совместив в себе механики shoot’em up [14] и roguelike игра дает необычный игровой опыт и за короткое время стала очень популярной, но точных причин популярности игры не знает сам разработчик [15]. Изначально проект вышел на платформе itch.io [16] в качестве пробного игрового проекта для улучшения навыков разработчика.

В начале каждой попытки игрок выбирает одного из нескольких персонажей с разным стартовым оружием и бонусами. Он сражается на бесконечной арене с автоматически генерируемой планировкой. Оружие атакует автоматически.

Главная цель – выжить как можно дольше против бесконечных волн монстров, которые наносят урон при соприкосновении. Победив монстров

и исследовав арену, игрок собирает самоцветы опыта, еду чтобы восстановить здоровье и другие полезные предметы.

Каждый новый уровень игрока дает выбор из трех-четырех видов оружия и пассивных улучшений. Другой способ получения улучшений – сундуки, выпадающие с сильных монстров. Большинство оружия имеет финальную форму, открывающуюся при определенных условиях и при максимальном улучшении.

Игра заканчивается через 15, 20 или 30 минут (в зависимости от арены). Появляется «жнец» – сильный враг, а за ним каждую 1 минуту новый. Сессия, дотянувшая до конца, считается успешной и награждается золотом. Полученное золото тратится на новых персонажей и постоянные улучшения. Задания открывают новые арены, оружие, персонажей, модификаторы.

Игрок может свободно перемещаться по миру. Персонаж игрока атакует автоматически вне зависимости от того есть ли рядом противники или нет.

Игра выполнена в пиксельном стиле. Противники получились очень колоритными, а бонусы хорошо выделяются на заднем фоне. При этом все выглядит очень просто и не напрягает глаза.

Музыкальное сопровождение простое и отлично подходит под геймплей.

Достоинства игры:

1) простой и затягивающий геймплей;
2) простое управление.

Недостатки игры:

1) небольшое количество контента;
2) не всегда отзывчивое управление.

Hades [17]

Hades – видеоигра в жанре roguelite разработанная компанией Supergiant Games [18] в 2018 году. Данная игра принесла Supergiant Games еще

большую популярность, благодаря чему компания приняла решение сделать продолжение игры [19]. Скриншот игры представлен на рисунке 3.

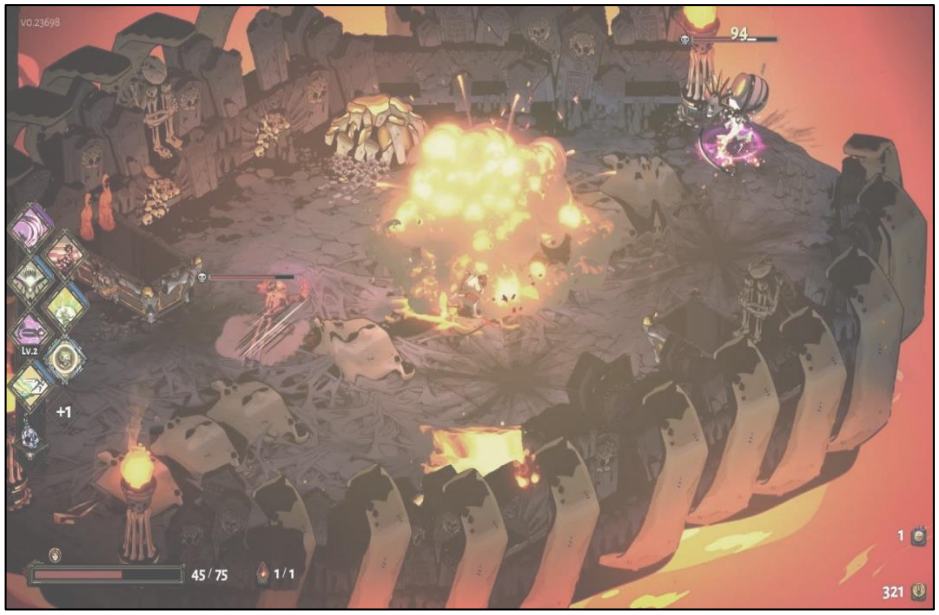

Рисунок 3 – Игра Hades

Игра представлена в изометрической проекции, где игрок контролирует Загреуса. Каждый побег начинается с прохождения через серию комнат, планировка и наполнение которых случайны. Боевая система представляет собой классический hack-and-slash с основными и специальными атаками, рывком для уклонения и нанесения урона, а также призывом силы одного из богов. На протяжении побега олимпийцы предоставляют Загреусу благословения, каждое из которых предлагает три варианта усилений и эффектов на текущий побег. Благословения соответствуют тематике богов – например, дары Зевса наносят урон молниями, а дары Посейдона отталкивают волнами.

Прогресс осуществляется по мере зачистки комнат, за которые игрок получает награды, позволяющие становиться сильнее или восстанавливать здоровье. Однако если у Загреуса заканчиваются очки здоровья и нет жетонов возрождения, его побеждают, и он возвращается в начальную комнату. Hades сочетает элементы roguelike и dungeon-crawler [20].

Мир игры разделен на клетки, игрок и противники перемещаются по ним беспрепятственно. Сражения происходят в реальном времени.

Игра имеет великолепный графический стиль. Локации хорошо детализированы и разнообразны. Каждый персонаж в игре имеет свой цвет, благодаря которому очень просто понять от кого получен то или иное улучшение или с кем конкретно ведется диалог в данный момент.

Достоинства игры:

1) интересный сюжет;
2) хорошо прописанные диалоги между персонажами;
3) графический стиль;
4) оружия кардинально меняют стиль игры.

Недостатки игры:

1) низкая реиграбельность между сессиями;
2) высокая сложность для новых игроков.

Вывод по первой главе

Благодаря анализу существующих решений можно сделать следующий вывод. Главными задачами в большинстве roguelite игр является изучение случайно сгенерированных структур для дальнейшего продвижения и улучшение персонажа по мере их прохождения. Игра должна иметь возможность сохранять улучшения персонажа между игровыми сессиями, Обычные противники должны быть в меру сбалансированными по сравнению с боссами. Игровой мир может состоять из одинаково разделенных плиток. Игрок и противники не должны быть привязаны к одной из плиток, то есть свободно перемещаться. Сражения с противниками должны происходить в реальном времени. Управление персонажем и настройки камеры должны делать игру динамичной и в то же время удобной, чтобы она не ощущалась недоработанной.

2. ПРОЕКТИРОВАНИЕ

2.1. Эскизный проект

Для написания эскизного проекта применяется определенный набор пунктов, при помощи которых редактируется будущая игра. Данный набор пунктов будет являться ориентиром для создания и представления текущего разрабатываемого проекта. Составим описание для каждого из пунктов.

Общие сведения

Игрок перемещается по случайно сгенерированному многоуровневому подземелью, побеждая противников, с целью добраться до самого нижнего этажа. Игроку придется развивать характеристики персонажа, а именно количество здоровья и наносимый урон. Развитие персонажа позволит игроку проходить этажи подземелья быстрее.

Концепция

Проектируемая работа является игрой в жанре roguelite, в которой основной целью является спуск до последнего этажа подземелья.

При победе над противниками, игрок может получать очки опыта, которые можно распределить между основными навыками – наносимый урон и здоровье. Для продвижения по подземелью, игроку необходимо сразить босса этажа. Босс этажа отличается от обычного противника только характеристиками.

Основной игровой процесс состоит из элементов hack and slash и roguelike.

Игровая атмосфера

Игровой мир представлен эпохой близкой к средневековью и населен разными вымышленными расами и существами. Основное игровое окружение – подземелье.

Игровой процесс

Каждое игровое действие происходит в реальном времени. Игрок свободно перемещает персонажа по экрану имея возможность войти в сраже-

ние с противником. Для улучшения персонажа, игроку необходимо получить новый уровень, набрав достаточное количество очков опыта, и распределить полученные очки умений.

Игровой прогресс игрока сохраняется между сессиями и в процессе прохождения подземелья.

Сражение

Сражения в игровом проекте происходят с использованиями механик жанра hack and slash. Игрок может свободно атаковать одного или нескольких противников. Противники так же, как и игрок, не ограничены в возможности атаковать. Сражение длится до тех пор, пока игрок не сразит всех атакующих его противников или сам не будет сражен.

Игроки

Игра будет являться однопользовательской, с возможностью играть только на одном устройстве.

2.2. Интерфейс пользователя

Было создано визуальное представление планируемого интерфейса игры при помощи схематической прорисовки макетов.

На рисунке 4 представлено схематическое изображение главного меню.

Рисунок 4 – Макет экрана главного меню

В главном меню планируется реализовать следующие элементы:

1) кнопка «Играть»;
2) кнопка «Настройки»;
3) кнопка «Выход».

Нажатие кнопки «выход» закрывает игру. Нажатие кнопки «Настройки» инициирует переход на экран настроек. На рисунке 5 представлено схематические изображение экрана настроек.

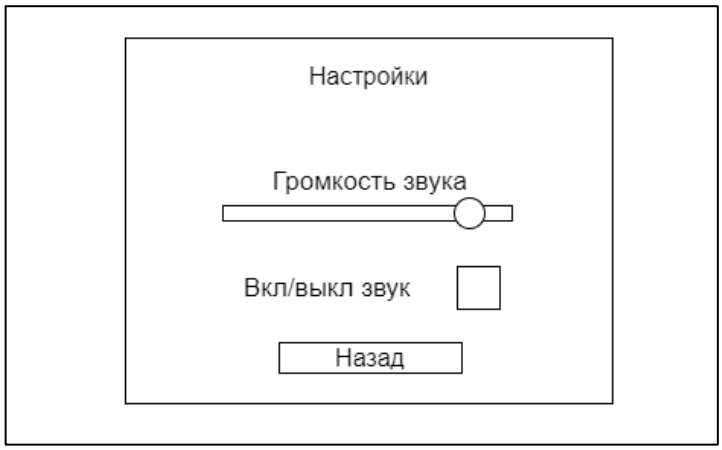

Рисунок 5 – Макет экрана настроек

На экране настроек планируется добавить следующие элементы:

1) слайдер «Громкость звука» регулирует громкость звука;
2) кнопка «Вкл/Вкл звук» включает или выключает звук в игре;
3) кнопка «Назад» инициирует переход на предыдущий экран.

Нажатие кнопки «Играть» в главном меню инициирует переход на экран выбора игры. Экран выбора игры представлен на рисунке 6.

Рисунок 6 – Макет экрана выбора игры

На экране выбора игры планируется добавить следующие элементы:

1) кнопка «Продолжить игру»;
2) кнопка «Начать новую игру»;
3) кнопка «Назад».

Нажатие кнопки «Назад» инициирует переход на экран главного меню. На рисунке 7 изображен макет игрового интерфейса.

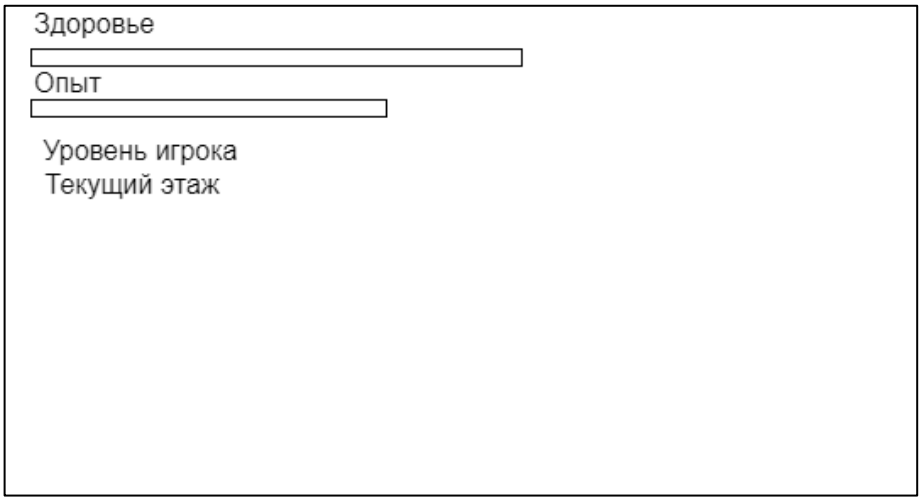

Рисунок 7 – Макет игрового интерфейса

Игровой интерфейс состоит из полосы здоровья игрока и полосы опыта игрока для отображения текущей позиции игрока на этаже.

Игрок может остановить игру, открыв меню паузы. На рисунке 8 изображен макет экрана паузы.

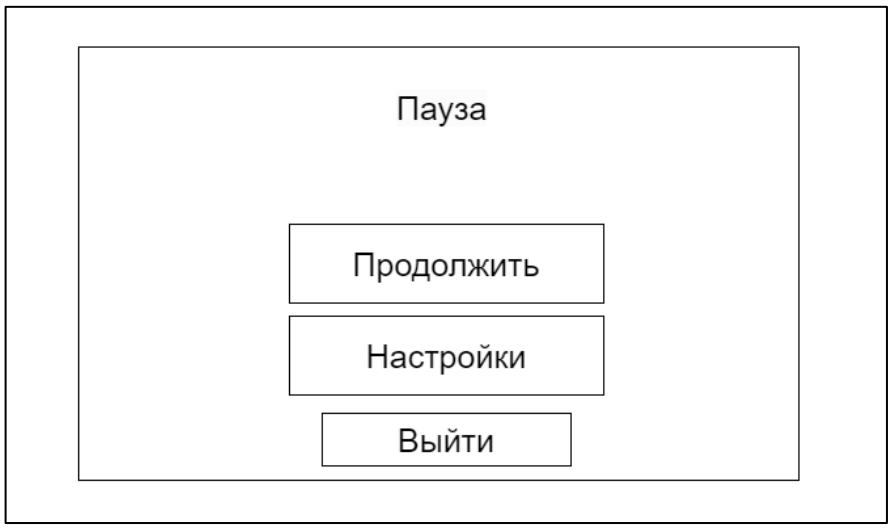

Рисунок 8 – Макет экрана паузы

На экран паузы планируется добавить следующе элементы:

1) кнопка «Продолжить игру»;
2) кнопка «Настройки»;
3) кнопка «Выйти в главное меню».

Во время игры игрок может открыть экран улучшений. Макет экрана улучшений представлен на рисунке 9.

Рисунок 9 – Макет экрана улучшений

На экран улучшений планируется добавить следующе элементы:

1) кнопка «+Здоровье»;
2) кнопка «+Урон»;
3) кнопка «Выход».

При поражении игрока противником инициируется переход на экран поражения. На рисунке 10 представлен экран поражения.

Рисунок 10 – Макет экрана поражения

На экран поражения планируется добавить следующе элементы:

1) кнопка «Продолжить»;
2) кнопка «Выход»;

При нажатии кнопки «Выход» инициируется переход на экран главного меню.

2.3. Диаграмма прецедентов использования

Диаграмма прецедентов – это диаграмма, с помощью которой, имеется возможность отразить отношение между актерами и прецедентами, а также является составной частью модели прецедентов, позволяющих описать систему на концептуальном уровне.

Сформируем модель прецедентов использования разрабатываемой системы. Основным актером, взаимодействующим с системой, является «Игрок», который взаимодействует с игровым приложением. Данный актер, используя главное игровое меню может начать игру, открыть окно настроек, выйти из игры. Модель представлена на рисунке 11.

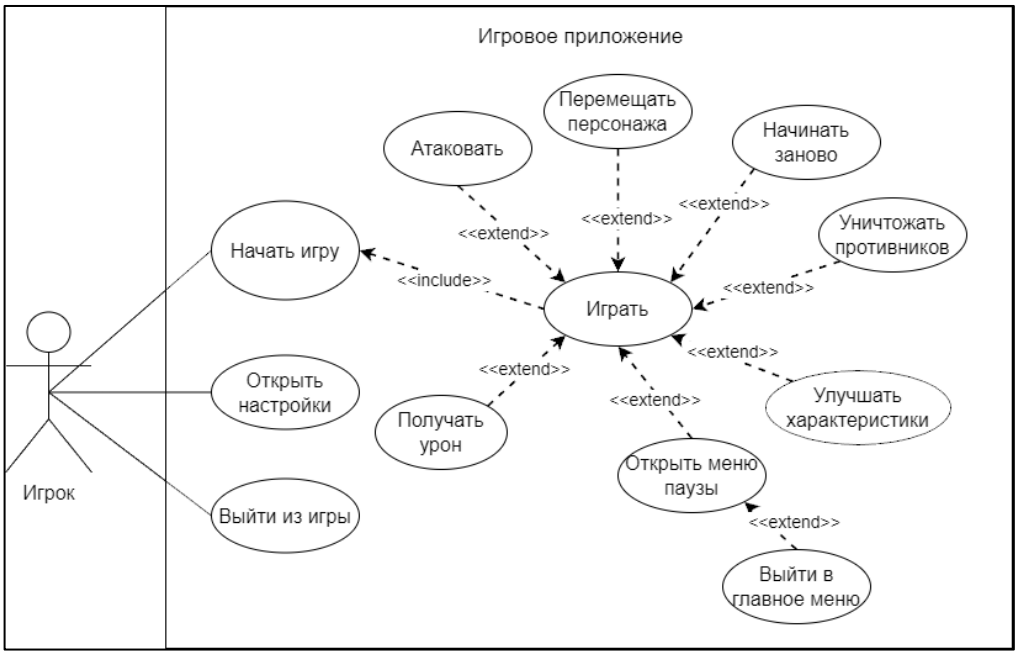

Рисунок 11 – Диаграмма прецедентов использования

2.4. Архитектура разрабатываемой системы

На рисунке 12 изображена диаграмма компонентов игровой системы.

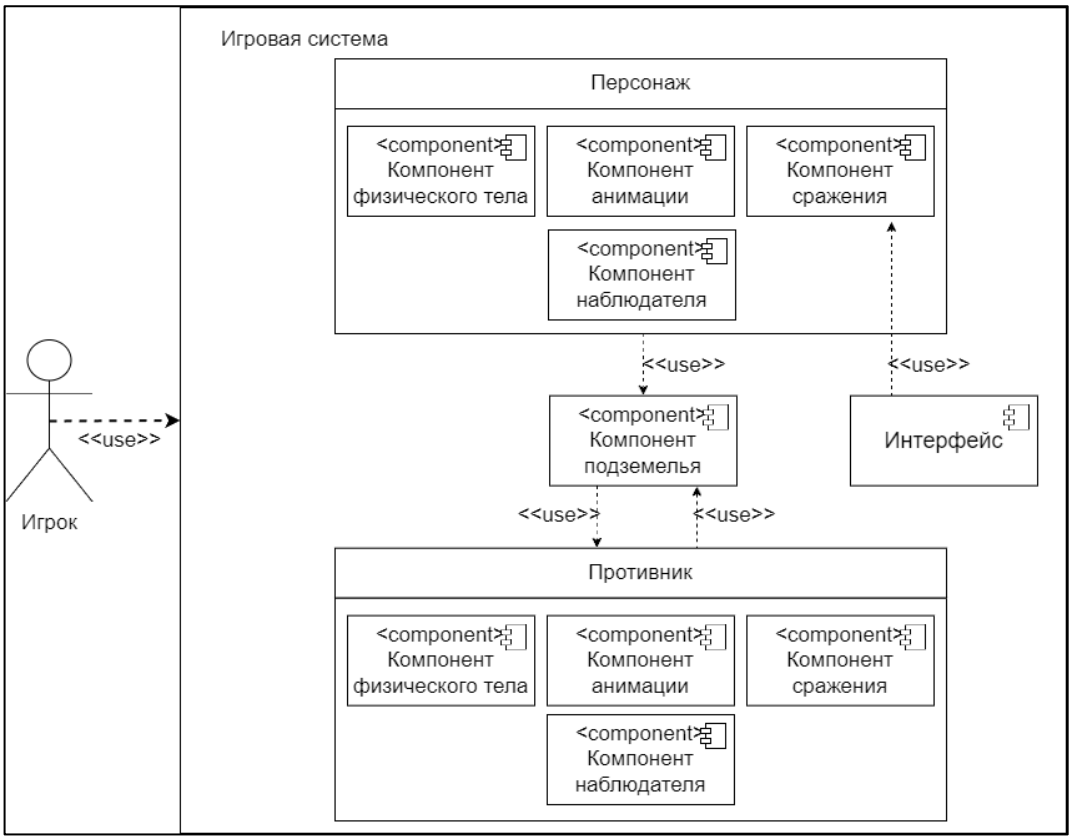

Рисунок 12 – Диаграмма компонентов игры

На данной диаграмме изображены основные объекты и компоненты системы. Каждый объект взаимодействует и управляет с своими компонентами. Ниже представлено описание каждого компонента.

Компонент сражения хранит в себе количество очков опыта и здоровья, текущие характеристики существа и его оружие, а также отвечает за логику характеристик и сражений.

Компонент физического тела содержит в себе логику взаимодействия с закрепленным телом в игровом мире.

Компонент подземелья хранит в себе информацию о текущем этаже, количестве противников и контролирует их характеристики.

Компонент анимации хранит в себе информацию о состоянии анимации.

Компонент наблюдателя указывает направление сущности.

Интерфейс связан с компонентом сражения игрока чтобы получать текущие характеристики.

3. РЕАЛИЗАЦИЯ

3.1. Программные средства реализации

Система будет разрабатываться на языке C++ (стандарт C++17). Для реализации системы были выбраны следующие библиотеки.

1. SFML 2.6.1 [21] – одна из самых популярных графических библиотек для C++. SFML обеспечивает простой интерфейс для различных компонентов ПК, таких как операционная система, окна, графика, аудио и сеть. Она упрощает разработку игр и мультимедийных приложений и является кроссплатформенной;
2. Box2D 2.4.1 [22] – высокопроизводительная библиотека для физического моделирования, созданная для двухмерных игр и приложений. она предоставляет инструменты для симуляции реалистичной физики в 2D-пространстве, таких как столкновения, силы и движения;
3. Nlohmann JSON 3.11.3 [23] – популярная библиотека для работы с JSON [25]. Она предоставляет удобный интерфейс для сериализации и десериализации JSON-данных, а также для работы с JSON-объектами.

Также для реализации были использованы: редактор кода Visual Studio Code. Для создания текстур персонажей, оружия и окружения использовалась программа создания текстур Aseprite [24].

3.2. Реализация компонентов системы

Класс сущности

Каждый игровой объект является контейнером, который хранит в себе определенное количество компонентов. При вызове метода обновления сущности, по очереди будут обновлены все закрепленные за ним компоненты. Листинг класса сущности представлен в листинге 1 приложения.

Каждая сущность инициализируется с помощью JSON объекта, хранящего в себе описание параметров данной сущности и списка компонентов с параметрами. В листинге 1 представлен пример JSON объекта сущности.

Листинг 1 – JSON объект сущности

```
"entity": {
    "combatComponent": {
      "maxHp": 20,
      "damageCooldown": 500
    },
    "observerComponent": {
      "observeState": 1
    },
    "spriteComponent": {
      "texturePath": "textures/little_hero.png"
    },
    "bodyComponent": {
      "body": {
        "enabled": true,
        "position": {
          "x": 0.0,
          "y": 0.0
        },
        "allowSleep": true,
        "physics": "dynamic",
        "bullet": true,
        "linearDamping": 100.0,
        "bodyType": "player"
      },
    }
}
```

С помощью фабричного класса, мы можем передавать любой JSON объект для создания сущности на его основе. В листинге 2 представлен фабричный класс сущностей.

Листинг 2 – Фабричный класс сущностей

```
class EntityFactory
{

public:
  EntityFactory();
  EntityStrongPtr CreateEntityFromFile(const char* path_resource);
  EntityStrongPtr CreateEntityFromStringJson(const char* string_json);
  EntityStrongPtr CreateEntityFromJson(json& root);
  EntityStrongPtr CreateEntity();

protected:
  EntityStrongPtr CreateEntity(json& root);
  ComponentStrongPtr CreateComponent(json::iterator& componentData);

private:

  size_t GetNextEntityID() { ++entityID; return entityID; }

  size_t entityID;

  ComponentCreators entityComponentCreators;
};
```

Базовый класс компонентов

Каждый компонент основан на базовом классе. Каждый компонент знает о том, кто им владеет и имеет возможность обратиться к другим компонентам в рамках одной сущности. Базовый класс компонентов представлен в листинге 3.

Листинг 3 – Базовый класс компонентов

```
class Component
{
public:
  virtual ~Component() {}
  // This functions shoud be overriden by implementation
  virtual void Update(float delta) {};
  virtual bool Init(json::iterator& component_data) = 0;
  virtual void PostInit() {};
  virtual void Destroy() {};
  // Should be overriden by the interface class
  virtual ComponentID GetComponentID() const = 0;

protected:

  EntityStrongPtr owner;

private:

  friend class EntityFactory;
  friend class EntityConstructor;

  void SetOwner(EntityStrongPtr new_owner) { owner = new_owner; }

};
```

Компонент физического тела

Компонент физического тела или `BodyComponent` содержит в себе тело сущности и его параметры. Так же этот компонент служит интерфейсом для работы с экземпляром тела Box2D. Листинг компонента физического тела представлен в листинге 2 приложения.

Компонент анимации

Компонент анимации или `AnimationComponent` отвечает за выбор текущего кадра анимации из текстуры сущности. Используя описание состояний из переданного JSON объекта, данный компонент отсчитывает необходимое время на кадр и рассчитывает координаты следующего кадра анимации. Методы обновления кадра анимации представлен в листинге 4.

Листинг 4 – Методы обновления кадра анимации

```
void AnimationComponent::Update(float delta)
{
  if (is_animated)
  {
    AnimationState& state = animation_states[current_state];
    if (state.tickSizeMs > 0)
    {
      if (!animation_timer.IsReady())
      {
        animation_timer.Update(delta);
      }
      else
      {
        current_tick++;
        animation_timer.Start();
      }
      if (current_tick > state.ticks)
        current_tick = 1;
    }
  }
  UpdateSpriteRect();
}

void AnimationComponent::UpdateSpriteRect()
{
  AnimationState& state = animation_states[current_state];

  sprite_rect.left = state.col * static_cast<int>(state.width) * cur-
rent_tick;
  sprite_rect.width = static_cast<int>(state.width);
  sprite_rect.top = state.row * state.height;
  sprite_rect.height = state.height;
}
```

Для отображения спрайта из текстуры используется `SpriteComponent`. Он просто запрашивает размеры кадра из `AnimationComponent`. Листинг запроса кадра представлен в листинге 5

Листинг 5 – Обновление кадра анимации в SpriteComponent

```
void SpriteComponent::UpdateSpriteRect()
{
  if (auto& currentAnimation = animation.lock())
  {
    sf::IntRect sprite_rect = currentAnimation->GetSpriteRect();
    if (spriteDirection.x < -0.1f)
    {
      sprite_rect.left += sprite_rect.width;
      sprite_rect.width *= -1;
      sprite.setTextureRect(sprite_rect);
    }
    else if (spriteDirection.x > 0.1f)
      sprite.setTextureRect(sprite_rect);
    sf::Vector2f sprite_rect_f(sprite_rect.width, sprite_rect.height);
    sprite.setOrigin(sprite_rect_f / 2.f);
  }
}
```

Такое разделение на два компонента обусловлено тем, что не всему необходима анимация. Листинги `AnimationComponent` и `SpriteComponent` представлены в листингах 3 и 4 приложения. Описание в формате JSON компонентов `AnimationComponent` и `SpriteComponent` представлены в листинге 6.

Листинг 6 – Описание AnimationComponent и SpriteComponent в формате JSON

```
"spriteComponent": {
      "texturePath": "textures/little_hero.png" },
"animationComponent": {
      "spriteRect": [8, 8],
      "animationStates": [ {
          "type": "IDLE",
          "width": 8,
          "height": 8,
          "row": 0,
          "col": 0,
          "ticks": 0,
          "tickSizeMs": 0
        },
        {
          "type": "MOVE",
          "width": 8,
          "height": 8,
          "row": 0,
          "col": 1,
          "ticks": 2,
          "tickSizeMs": 150
        }
      ]
    }
```

Компонент наблюдателя

Компонент наблюдателя используется для того, чтобы понять куда направлен взгляд сущности. У данного компонента есть 3 состояния: `Unlocked`, `OnMovement` и `OnTarget`. `Unlocked` – разблокированное состояние, результирующий вектор будет равен нулю. `OnMovement` – заблокирован на движении, результирующий вектор будет определяться по движениям `BodyComponent`, если он присутствует. `OnTarget` – заблокирован на цели, а именно на другой сущности. Направление взгляда рассчитывается при каждом обновлении компонента. Метод обновления компонента наблюдателя представлен в листинге 5 приложения.

Данный компонент задействован как в логике поведения сущности противника, так и в логике поведения сущности игрока. Противник, при обнаружении игрока будет направлен в его сторону. Игрок в свою очередь будет направлен в сторону курсора. Листинг компонента наблюдателя представлен в листинге 6 приложения.

Компонент сражения

Компонент сражения или `CombatComponent` хранит в себе очки здоровья, опыта, значение уровня и количество заработанного опыта. Так же он отвечает за использование оружия сущности, логику обработки полученного урона и логику обработки улучшений. Листинг компонента сражения представлен в листинге 7 приложения.

Каждому компоненту сражения назначается используемое оружие, которое так же является сущностью с компонентом оружия, компонентом тела и компонентом анимации. Компонент оружия отвечает за готовность к атаке и наносимый урон. Компонент оружия представлен в листинге 7.

Листинг 7 – Компонент оружия

```
class WeaponComponent : public Component
{
public:

  inline static const ComponentID COMPONENT_ID =
    Components::WeaponComponentID;

  WeaponComponent() = default;
  virtual ~WeaponComponent() = default;

  void Update(float delta);
  bool Init(json::iterator& component_data);

  void SetWeaponProperties(Weapon& new_properties);
  Weapon GetWeaponProperties() { return weapon_properties_; }

  ComponentID GetComponentID() const override { return COMPONENT_ID; };
  bool IsReady() { return attack_cooldown_timer_.IsReady(); }
  bool Use();
  int GetWeaponDamage();

private:

  friend class CombatComponent;
  Weapon weapon_properties_;
  b2DistanceJointDef weapon_joint_def_;
  GameTimer attack_cooldown_timer_;

};
```

Класс персонажа

Класс персонажа используется для управления сущностью игрока. Он хранит в себе параметры игрока, а также обрабатывает нажатия клавиш управления, управляет состояниями анимации в соответствии с выполненным действием. Листинг класса персонажа представлен в приложении. Листинг метода обработки нажатия клавиш представлен в листинге 8 приложения.

Класс противника

Класс противника используется для управления сущностью противника. Так же, как и класс персонажа он хранит параметры противника, но помимо сущности противника используется сущность зоны видимости, которая проверяет есть ли игрок в зоне досягаемости или нет. На основе зоны видимости основан простейший искусственный интеллект, где если игрок попадает в зону видимости, противник подходит на расстояние удара и пытается сразить игрока. Листинг класса противника представлен в листинге 9 приложения.

3.3. Компонент подземелья

Компонент подземелья создает игровой мир с помощью процедурной генерации. Общий алгоритм генерации этажа представлен на рисунке 13.

Рисунок 13 – Общий алгоритм генерации этажа

Компонент начинает работу с инициализации карты заданного размера. Карта является структурой типа `Map`, которая хранит в себе вектор из клеток типа `Cell`. Объявление структуры клетки представлено в листинге 8.

Листинг 8 – Объявление структуры клетки

```
struct Cell
{
  enum Direction
  {
    NORTH, EAST, WEST, SOUTH
  };

  Cell(int x, int y;

  int x, y;
  bool is_visited;
  bool is_room;
  bool is_path;

  std::array<std::pair<Direction, bool>, 4> available_direction;


bool close_to_room;
  bool is_door;
  bool is_filled;

  Room* binded_room;
  Cell::Direction nearest_door_direction;
  bool has_door_direction;

  void RemoveSideConnection(Cell* other, Direction side);
  Direction GetFacingDirection(Cell* other);
  Direction GetOppositeDirection(Direction dir);
  bool IsOpenWithCell(Cell* other);
  Cell* GetCellByDirection(Map& cell_map, Direction direction);
}
```

Объявление структуры карты представлено в листинге 9.

Листинг 9 – Объявление структуры карты

```
struct Map
{
  Map() : is_initialized(false) {}
  ~Map();

  void Initialize(int w, int h);
  void Reset();

  std::vector<Cell*> map_vector;

  int width;
  int height;

  bool is_initialized;

  Cell* GetCell(int x, int y);
};
```

Далее, необходимое количество комнат компонуется в размеры массива и помечаются соответствующим флагом в структуре `Cell`. Как только комнаты созданы, начинается создание лабиринта. Метод создания лабиринта представлен в листинге 10 приложения.

Изначально создается вектор не посещенных клеток. Далее, проходя по созданному вектору убираются стенки между клетками. Посещенные клетки стираются из созданного вектора и помечаются посещенными.

После подготовки лабиринта, случайным образом в созданных комнатах создаются дверные проходы.

Затем созданные двери соединяются кратчайшими возможными путями с помощью алгоритма поиска с возвратом [25]. Метод создания путей представлен в листинге 10.

Листинг 10 – Метод создания путей

```
void Floor::CreatePaths(Cell* current, Cell* prev, Cell* start)
{
  current->is_path = true;

  if(current->is_door || current->is_room)
  {
    return;
  }
  bool meet_visited = false;
  std::vector<std::pair<Cell*, Cell::Direction>> unvisited_neighbors;
  for (Cell::Direction next_direction : {Cell::Direction::NORTH, Cell::Di-
rection::EAST, Cell::Direction::SOUTH, Cell::Direction::WEST})
  {
    if (Cell* cell = current->GetCellByDirection(cell_map_, next_direc-
tion))
    {
      Cell::Direction to_prev = current->GetFacingDirection(prev);
      if (to_prev != next_direction && current->IsOpenWithCell(cell))
      {
        unvisited_neighbors.push_back({cell, next_direction});
      }
    }
  }
  int path_number = unvisited_neighbors.size();

  for (auto& neighbor_pair : unvisited_neighbors)
  {
    Cell* neighbor = neighbor_pair.first;
    if (neighbor->is_path)
    {
      meet_visited = true;
      break;
    }
    else if (!neighbor->is_door && !meet_visited)
      CreatePaths(neighbor, current, start);
  }
```

```
  if (current->is_visited ||
    (current->binded_room && current->binded_room != start->binded_room)
  )
    meet_visited = true;

  bool path_found = meet_visited;

  if (!path_found)
  {
    current->is_path = false;
  }
  else
    prev->is_visited = true;
}
```

Приведенный в листинге метод вызывается для каждой созданной двери. Все не помеченные клетки полностью блокируются стенами и становятся недоступными. Пример генерации этажа представлен на рисунке 14.

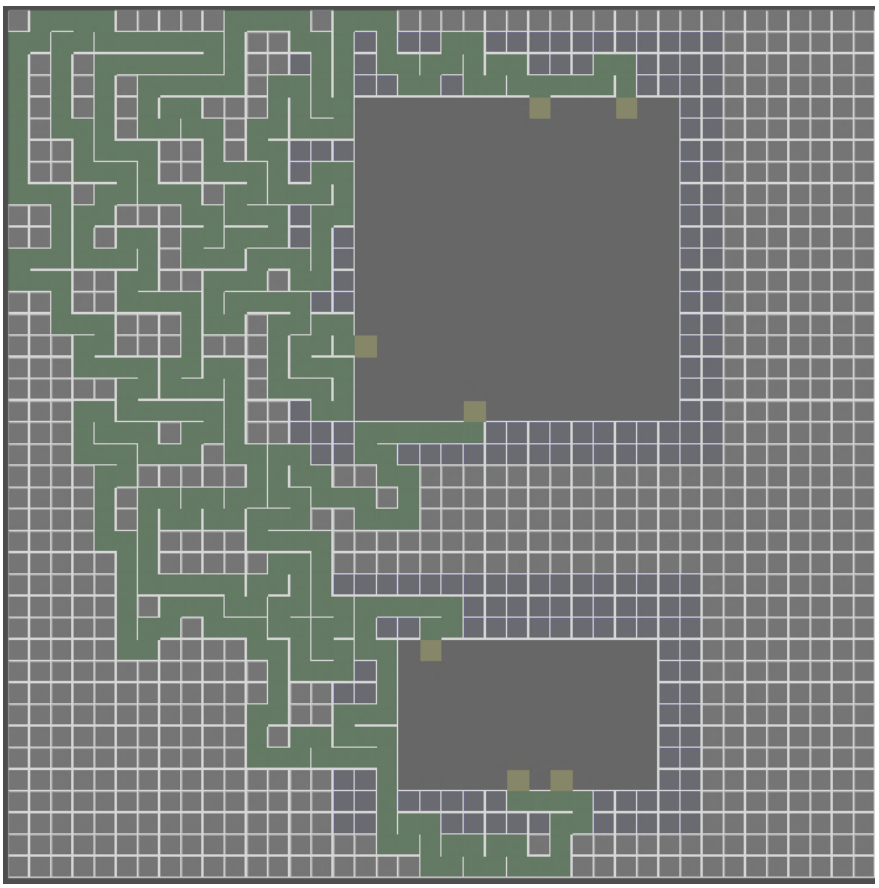

Рисунок 14 – Пример генерации этажа

Этап расширения подразумевает собой увеличение прохода лабиринта на 1 клетку во все возможные направления. Этап расширения представлен на рисунке 15.

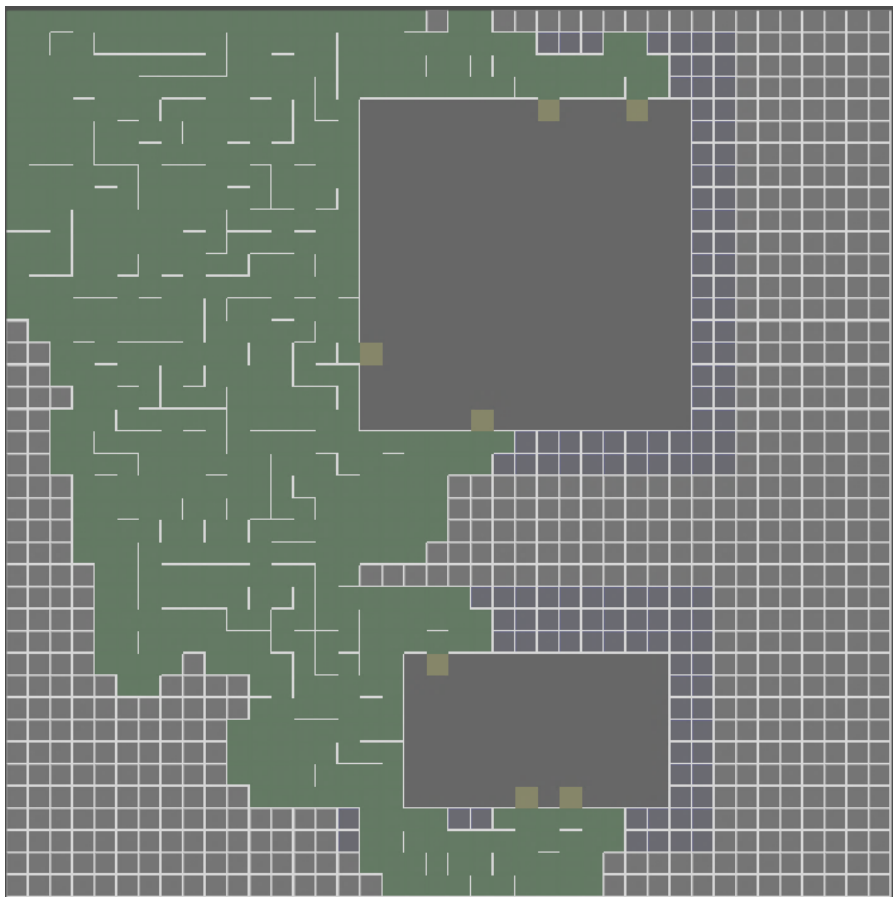

Рисунок 15 – Этап расширения путей

Приведенный выше алгоритм используется в компоненте подземелья, который генерирует карту, подготавливает описание объектов Box2D и инициализирует их в мире Box2D. Во время подготовки описаний объектов, сама карта прорисовывается на ранее подготовленной текстуре согласно масштабу мира Box2D, который задается в компоненте физического тела как константа. Прорисованная текстура используется для дальнейшего отображения в игровом цикле. Листинг компонента подземелья представлен в листинге 11 приложения.

3.4. Реализация интерфейса

Реализация экрана главного меню представлена на рисунке 16.

Рисунок 16 – Экран главного меню

Реализация экрана настроек представлена на рисунке 17.

Рисунок 17 – Экран настроек

Реализация экрана выбора игры представлена на рисунке 18.

Рисунок 18 – Экран выбора игры

Реализация игрового интерфейса представлена на рисунке 19.

Рисунок 19 – Реализация игрового интерфейса

Реализация экрана улучшений представлена на рисунке 20.

Рисунок 20 – Экран улучшений

Реализация экрана паузы представлена на рисунке 21.

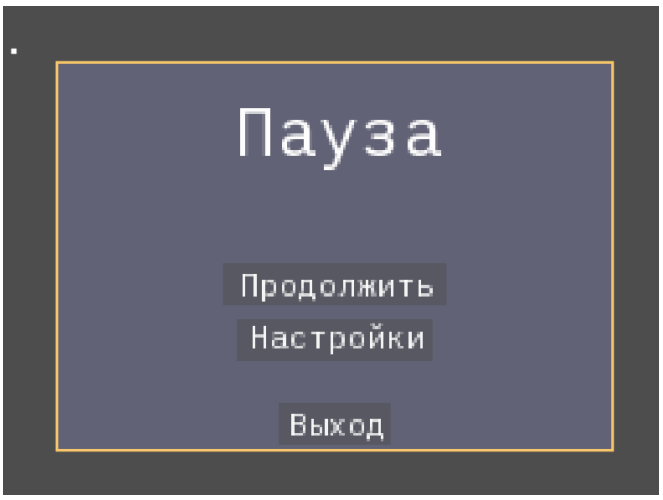

Рисунок 21 – Экран паузы

Реализация экрана поражения представлена на рисунке 22.

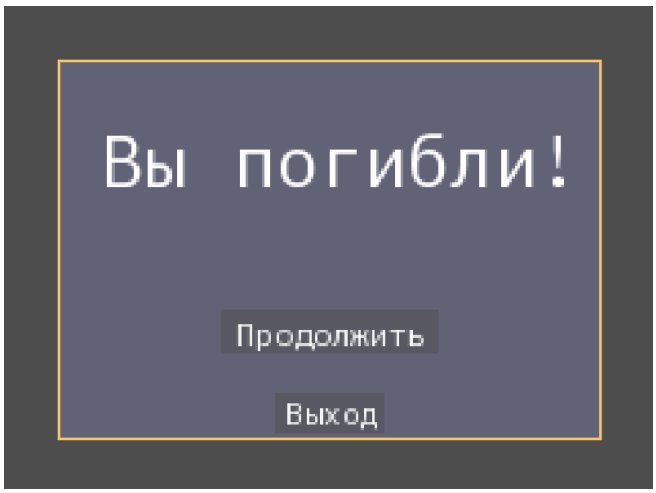

Рисунок 22 – Экран поражения

3.5. Реализация спрайтов

Для персонажа игрока был создан спрайт с анимацией и его оружием. Спрайт персонажа игрока представлен на рисунке 23.

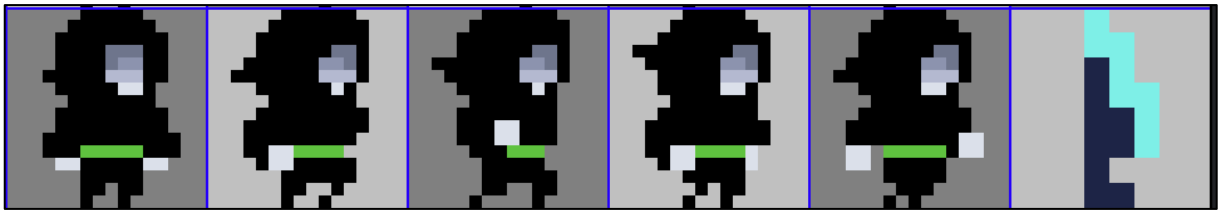

Рисунок 23 – Спрайт персонажа игрока

На рисунке 24 представлен спрайт противника Огр с небольшой анимацией и его оружия. Аналогично представлены и другие противники, такие как Гоблин, Вампир и Дух. Они представлены на рисунках 25, 26 и 27 соответственно.

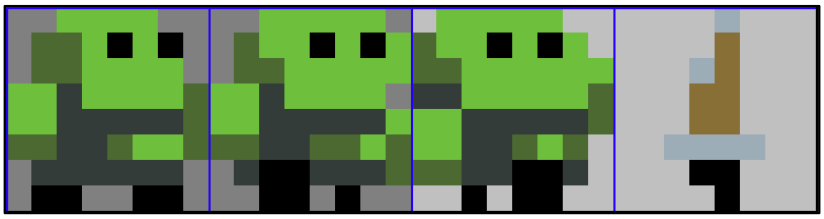

Рисунок 24 – Спрайт противника Огр

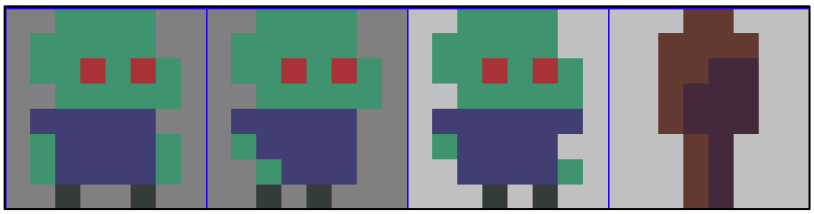

Рисунок 25 – Спрайт противника Гоблин

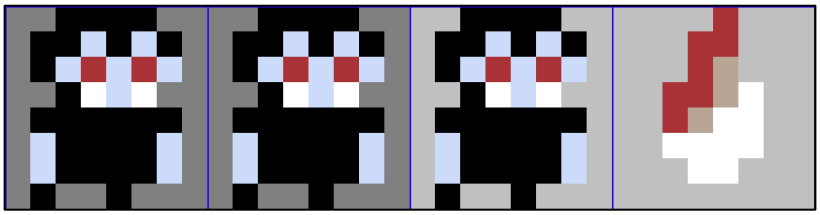

Рисунок 26 – Спрайт противника Вампир

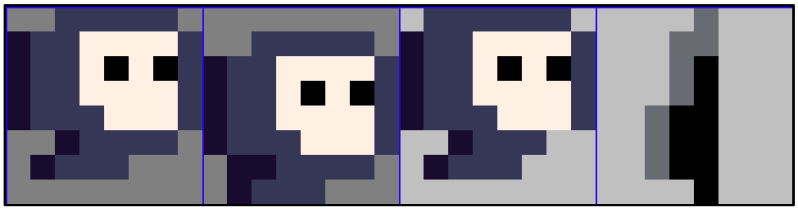

Рисунок 27 – Спрайт противника Дух

Вывод по третьей главе

В соответствии с заданием и спроектированной архитектурой было создано игровое приложение. Были описаны основные компоненты игры, а также были приведены листинги исходного кода, скриншоты и спрайты.

4. ТЕСТИРОВАНИЕ

4.1. Функциональное тестирование

Для тестирования системы применялось функциональное тестирование, т.е. тестирование программного обеспечения в целях проверки реализуемости функциональных требований.

Тестирование игры проводилось вручную. В таблице 1 приведен протокол ручного тестирования некоторых аспектов работы системы.

Таблица 1 – Протокол функционального тестирования

№	Название теста	Действия	Результат	Тест пройден?
1	Запуск игры	Запустить исполняемый файл.	Запуск игры и вывод на экран главного меню.	Да
2	Передвижение	1. Переместить персонажа с помощью WASD. 2. Ускорить персонажа нажатием Shift. 3. Выполнить рывок нажатием Пробел	Персонаж выполняет указанную команду	Да
3	Нанесение урона и получение опыта	Довести здоровье противника до нуля	Противник получает урон от ударов и погибает при нуле здоровья, игрок получает очки опыта.	Да
4	Получение урона	Попасть под удары противника	Персонаж получает урон, при нуле здоровья, появляется экран поражения	Да
5	Повтор игры с сохранением прогресса	1. Попасть под удары противника. 2. Довести здоровье до нуля. 3. На экране поражения нажать «Продолжить»	Прогресс игрока остался таким же, этаж сбросился до 1	Да
6	Переход на следующий этаж	1. Довести здоровье босса до нуля	Игрок переходит на новый этаж, значение этажа увеличилось на 1	Да
7	Улучшение здоровья	1. Получить достаточно опыта чтобы появилось уведомление об улучшении. 2. Нажать клавишу L. 3. Нажать на кнопку «+Здоровье»	Значение улучшения здоровья увеличилось на 1, здоровье восполнилось	Да

Окончание таблицы 1

8	Улучшение урона	1. Получить достаточно опыта чтобы появилось уведомление об улучшении. 2. Нажать клавишу L. 3. Нажать на кнопку «+Урон»	Значение улучшения урона увеличилось на 1	Да

4.2. Юзабилити тестирование

Было проведено юзабилити тестирование разработанного приложения. В тестировании принимала участие группа из пяти человек состоящая из друзей и одногруппников со значительно различающимся уровнем игры. В рамках тестирования были выявлены и исправлены следующий ряд недочетов.

1. После выхода из игрового процесса в меню и начала новой игры, игра продолжалась в месте выхода.
2. Была вероятность погибнуть до начала игры, так как игровой процесс начинался во время нахождения в меню.
3. После перехода на следующий этаж некоторые противники были в состоянии преследования игрока.

Вывод по четвертой главе

В четвертой главе было проведено функциональное тестирование и юзабилити-тестирование разработанного приложения. Все функциональные тесты были пройдены успешно. В ходе юзабилити-тестирования был исправлен ряд обнаруженных недочетов.

ЗАКЛЮЧЕНИЕ

В рамках данной работы была разработана компьютерная игра в жанре «Roguelite» для платформы Windows. При этом были решены следующие задачи:

1. Проведен анализ предметной области, выявлены особенности и аспекты жанра roguelite.
2. Спроектировано игровое приложение, описан эскизный проект, визуализированы макеты интерфейса, описаны основные компоненты игрового приложения.
3. Реализовано игровое приложение, реализованы все компоненты игрового приложения, а также все спроектированные макеты интерфейса.
4. Проведено тестирование приложения. Все тесты в ходе функционального тестирования были успешно пройдены. Найденные недочёты в ходе юзабилити тестирования были исправлены.

К планам о дальнейшем развитии игры можно отнести:

1. Добавление различных активностей в игровом мире, таких как неигровые персонажи, новые комнаты.
2. Добавление расходуемых предметов.
3. Улучшение пользовательского интерфейса.

ЛИТЕРАТУРА

1. Итоги и тенденции игрового рынка в 2023 году. [Электронный ресурс] URL: https://games.logrusit.com/ru/news/game-industry-trends-year/ (дата обращения: 10.03.2024 г.).

2. Roguelike. [Электронный ресурс] URL: https://www.cloudfallstudios.com/blog/2020/7/13/what-even-is-a-roguelike (дата обращения: 10.02.2024 г.).

3. Berlin interpretation. [Электронный ресурс] URL: https://www.roguebasin.com/index.php?title=Berlin_Interpretation (дата обращения: 10.02.2024 г.).

4. Rogue. [Электронный ресурс] URL: https://crpgaddict.blogspot.com/2010/02/rogue-story-and-gameplay.html (дата обращения: 10.02.2024 г.).

5. Nethack. [Электронный ресурс] URL: https://www.nethack.org/ (дата обращения: 10.02.2024 г.).

6. What Are Hack and Slash Video Games? [Электронный ресурс] URL: https://www.makeuseof.com/what-are-hack-and-slash-video-games/ (дата обращения: 15.02.2024 г.).

7. Spelunky. [Электронный ресурс] URL: https://www.spelunkyworld.com/ (дата обращения: 10.02.2024 г.).

8. Mossmouth. [Электронный ресурс] URL: https://www.mossmouth.com (дата обращения: 10.02.2024 г.).

9. What is Platform Game? [Электронный ресурс] URL: https://www.lifewire.com/what-is-a-platform-game-812371 (дата обращения: 10.02.2024 г.).

10. List of video games considered the best. [Электронный ресурс] URL: https://www.gq-magazine.co.uk/article/best-video-games-all-time (дата обращения: 10.02.2024 г.).

11. Vampire Survivors. [Электронный ресурс] URL: https://poncle.itch.io/vampire-survivors (дата обращения: 10.02.2024 г.).

12. From Big Macs to Baftas: the incredible story behind the hit video game Vampire Survivors. [Электронный ресурс] URL: https://www.theguardian.com/games/2023/aug/04/baftas-video-game-vampire-survivors-luca-galante (дата обращения: 10.02.2024 г.).

13. Magic Survivor Wiki. [Электронный ресурс] URL: https://magic-survival-rpg.fandom.com/wiki/Magic_Survival_Wiki (дата обращения: 11.02.2024 г.).

14. All-Time Favorites: Shoot-em-up Games. [Электронный ресурс] URL: https://www.gamedesigning.org/gaming/shoot-em-up/ (дата обращения: 19.02.2024 г.).

15. Not even Vampire Survivors' creator knows why it's so good. [Электронный ресурс] URL: https://videogames.si.com/news/vampire-survivors-success-reasons-unknown (дата обращения: 11.02.2024 г.).

16. Vampire Survivors by poncle. [Электронный ресурс] URL: https://poncle.itch.io/vampire-survivors (дата обращения: 11.02.2024 г.).

17. Supergiant Games | Hades. [Электронный ресурс]: URL: https://www.supergiantgames.com/games/hades/ (дата обращения: 11.02.2024 г.).

18. Supergiant Games. [Электронный ресурс] URL: https://www.supergiantgames.com (дата обращения: 11.02.2024 г.).

19. Greek mythology. [Электронный ресурс] URL: https://www.britannica.com/topic/Greek-mythology (дата обращения: 11.02.2024 г.).

20. Dungeon crawl. [Электронный ресурс] URL: https://plarium.com/en/glossary/dungeon-crawl/ (дата обращения: 11.02.2024 г.).

21. SFML. [Электронный ресурс] URL: https://www.sfml-dev.org/ (дата обращения: 14.02.2024 г.).

22. Box2D. [Электронный ресурс] URL: https://box2d.org/ (дата обращения: 14.02.2024 г.).

23. Nlohmann JSON. [Электронный ресурс] URL: https://json.nlohmann.me/ (дата обращения: 14.02.2024 г.).

24. JSON. [Электронный ресурс] URL: https://www.json.org/json-en.html/ (дата обращения: 15.02.2024 г.).

25. Алгоритм backtracking. [Электронный ресурс] URL: https://habr.com/ru/companies/otus/articles/746408/ (дата обращения: 04.06.2024 г.).

ПРИЛОЖЕНИЕ. Листинги исходных кодов

Листинг 1 – Класс сущности

```
class Entity
{

public:

  explicit Entity(size_t id)
  {
    ID = id;
  }

  virtual ~Entity() {}

  virtual void Update(float delta);
  virtual bool Init(json& entityResource) { return true; }
  virtual void PostInit();
  virtual void Destroy()
  {
    for (auto& component : components)
    {
      component.second->Destroy();
    }
  }


  size_t GetId() { return ID; }
  template<class ComponentType>
  std::weak_ptr<ComponentType> GetComponent(size_t componentID)
  {
    EntityComponents::iterator lookIt = components.find(componentID);
    if (lookIt != components.end())
    {

      ComponentStrongPtr baseComp(lookIt->second);
      // cast to subclass of a pointer
      std::shared_ptr<ComponentType> subComp(
        std::static_pointer_cast<ComponentType>(baseComp)
      );
      std::weak_ptr<ComponentType> weakComp(subComp);
      return weakComp;
    }
    else
    {
      return std::weak_ptr<ComponentType>();
    }
  }
private:
  friend class EntityFactory;
  friend class EntityConstructor;
  void AddComponent(ComponentStrongPtr newComponent);

  size_t ID;
  EntityComponents components;
};
```

Листинг 2 – Компонент физического тела

```
class BodyComponent : public Component
{

public:

  inline static const float PIXELS_IN_METER = 16;
  inline static const float PIXEL_TO_METER = 1.f / PIXELS_IN_METER;

  enum BodyCathegory
  {
    BOUNDARY =              1,
    ENEMY =                 2,
    ENEMY_PROJECTILE =      4,
    ENEMY_FOV_SENSOR =      8,
    PLAYER =                16,
    PLAYER_PICKUP_SENSOR =  32,
    WEAPON =                64,
    ITEM =                  128
  };
  inline static const ComponentID COMPONENT_ID = Components::BodyComponen-
tID;

  BodyComponent() = default;

  virtual ~BodyComponent() = default;

  void Update(float delta) override;
  bool Init(json::iterator& data) override;
  void PostInit() override;
  void Destroy() override;
  ComponentID GetComponentID() const override { return COMPONENT_ID; }
  void InitializeBody(b2World& world);
  void ApplyLinearImpulse(sf::Vector2f& velocity);
  void ApplyForceToCenter(sf::Vector2f& force, bool wake = true);
  void SetLinearDamping(float damping);
  void SetPosition(sf::Vector2f& position);
  void SetRotation(float angle);
  bool IsEnabled() { return body_->IsEnabled(); }
  void SetEnabled(bool enabled) { body_->SetEnabled(enabled); }
  sf::Vector2f GetLinearVelocity();
  sf::Vector2f GetPosition();
  b2Body* GetBody() { return body_; }
  BodyCathegory GetBodyCathegory() { return body_type_; }
  static void BeginContact(BodyComponent* first, BodyComponent* second);
  static void PreSolve(BodyComponent* first, BodyComponent* second, const
b2Manifold* old_manifold);
  static void PostSolve(BodyComponent* first, BodyComponent* second, const
b2ContactImpulse* impulse);
private:
  friend class CombatComponent;

  uint16 GetBodyCathegoryFromString(std::string cathegory);
  b2BodyDef body_def_;
  b2Body* body_;
  b2World* body_world_;
  std::vector<b2FixtureDef> body_fixtures_;
  std::vector<b2PolygonShape> fixture_polygons_;

  BodyCathegory body_type_;
};
```

Листинг 3 – AnimationComponent

```
class AnimationComponent : public Component
{
  // iterates from left to right
  struct AnimationState
  {
    int state;
    long long width;
    size_t height;
    size_t row;
    size_t col;
    size_t ticks;
    size_t tickSizeMs;
  };

public:

  const static ComponentID COMPONENT_ID;

  AnimationComponent();
  ~AnimationComponent();

  void Update(float delta) override;
  bool Init(json::iterator& component_data) override;
  void PostInit() override;

  virtual ComponentID GetComponentID() const { return COMPONENT_ID; }

  void StartAnimation();

  void StopAnimation() { is_animated = false; }

  bool IsAnimated() { return is_animated; }

  void ChangeState(size_t state);
  void ResetAnimation();
  std::map<std::string, size_t>& GetAnimationStates(){ return state_types;
}

  sf::IntRect GetSpriteRect();

private:

  void UpdateSpriteRect();

  bool can_be_animated;

  sf::IntRect sprite_rect;

  bool is_animated;


  std::map<std::string, size_t> state_types;

  std::map<size_t, AnimationState> animation_states;

  size_t current_state;
  size_t current_tick;
  GameTimer animation_timer;
};
```

Листинг 4 – SpriteComponent

```
class SpriteComponent : public Component, public sf::Drawable
{
public:

  static const ComponentID COMPONENT_ID;

  SpriteComponent();
  ~SpriteComponent() = default;

  void Update(float delta) override;
  bool Init(json::iterator& componentData) override;
  void PostInit() override;

  ComponentID GetComponentID() const { return COMPONENT_ID; }

  bool SetTexture(const char* path);
  void SetPosition(sf::Vector2f position);
  void SetRotationDeg(float angle_deg);
  void SetRotationRad(float angle_rad);
  void Rotate(float angle);
  sf::Vector2i GetSpriteSize();
  void SetColorMod(sf::Color color) { sprite.setColor(color); color_mod =
color; }

  bool IsAnimated() { return isAnimated; }
  bool IsControlled() { return isControlled; }

  void SetEnabled(bool flag) { is_enabled = flag; }
  bool IsEnabled() { return is_enabled; }
private:
  void draw(sf::RenderTarget& target, sf::RenderStates states) const
  {
    if (is_enabled)
    {
      if (auto& entity_body = body.lock())
      {
        if (entity_body->IsEnabled())
          target.draw(sprite, states);
      }
    }
  }
  void UpdatePosition();
  void UpdateSpriteDirection();
  void UpdateSpriteRect();

  std::weak_ptr<AnimationComponent> animation;
  std::weak_ptr<BodyComponent> body;
  std::weak_ptr<ObserverComponent> observer;
  bool isAnimated;
  bool isControlled;
  bool isObserving;
  bool is_enabled;
  sf::Vector2f spriteDirection;
  sf::Color color_mod;
  std::string texturePath;
  sf::Texture texture;
  sf::Sprite sprite;
  float angle;
};
```

Листинг 5 – Метод обновления компонента наблюдателя

```
void ObserverComponent::Update(float delta)
{
  switch (currentState)
  {
    case ObserveState::OnMovement:
    {
      std::weak_ptr<BodyComponent> body_comp = owner->GetComponent<Body-
Component>(Components::BodyComponentID);
      if (auto& body = body_comp.lock())
      {
        sf::Vector2f move_vector = body->GetLinearVelocity();
        float move_x_abs = std::abs(move_vector.x);
        float move_y_abs = std::abs(move_vector.y);
        if (move_x_abs > 0.01f && move_y_abs > 0.01f)
        {
          float normalization = 1.f / std::sqrt(move_vector.x * move_vec-
tor.x + move_vector.y * move_vector.y);
          move_vector *= normalization;
          if (move_vector.x != 0 || move_vector.y != 0)
            lookDirection = move_vector;
        }
      }
      break;
    }
    case ObserveState::OnTarget:
    {
      if (auto& target_entity = target)// .lock())
      {

        if (auto& this_controller = body)//.lock())
        {
          sf::Vector2f result_vector = target_entity->GetPosition() -
this_controller->GetPosition();
          float x_abs = std::abs(result_vector.x);
          float y_abs = std::abs(result_vector.y);

          if (x_abs > 0.01f && y_abs > 0.01f)
          {
            float normalization = 1.f / std::sqrt(result_vector.x * re-
sult_vector.x +
              result_vector.y * result_vector.y);

            result_vector *= normalization;
            if (result_vector.x != 0 || result_vector.y != 0)
              lookDirection = result_vector;
          }
        }
      }
      break;
    }
    case ObserveState::Unlocked:
    {
      lookDirection.x = 0;
      lookDirection.y = 0;
      break;
    }
  }
}
```

Листинг 6 – Компонент наблюдателя

```
class ObserverComponent : public Component
{

public:

  enum class ObserveState
  {
    Unlocked,
    OnMovement,
    OnTarget
  };

  static const ComponentID COMPONENT_ID;

  ObserverComponent();
  ~ObserverComponent() = default;

  void Update(float delta) override;
  bool Init(json::iterator& componentData) override;
  void PostInit() override;

  sf::Vector2f GetTargetDirection();

  bool LockOnTarget(EntityStrongPtr& e);
  bool LockOnTarget(Entity* e);
  bool LockOnMovement();
  ObserveState GetCurrentState() { return currentState; }
  void Unlock();

  ComponentID GetComponentID() const { return COMPONENT_ID; }

private:

  ObserveState currentState;

  sf::Vector2f lookDirection;
  std::shared_ptr<BodyComponent> body;
  std::shared_ptr<BodyComponent> target;

};
```

Листинг 7 – Метод создания лабиринта

```
void Floor::CreateMaze()
{
  std::vector<Cell*> unfixed_cells_;

  for (int y = 0; y < cell_map_.height; y++)
  {

for (int x = 0; x < cell_map_.width; x++)
    {
      Cell* cell = cell_map_.GetCell(x, y);
      if (!cell->is_room)
        unfixed_cells_.push_back(cell);
    }
  }

  std::default_random_engine generator(seed);
```

Окончание листинга 7 приложения

```
  while (unfixed_cells_.size())
  {
    Cell* current_cell = unfixed_cells_.back();
    current_cell->is_visited = true;

    std::vector<std::pair<Cell*, Cell::Direction>> unvisited_neighbors;

    for (Cell::Direction next_direction : {Cell::Direction::NORTH,
Cell::Direction::EAST, Cell::Direction::SOUTH, Cell::Direction::WEST})
    {
      if (Cell* cell = current_cell->GetCellByDirection(cell_map_, next_di-
rection))
      {
        if (!cell->is_visited && !cell->is_room)
        {
          unvisited_neighbors.push_back({cell, next_direction});
        }
      }
    }

    size_t neighbor_count = unvisited_neighbors.size();

    if (neighbor_count > 0)
    {
      std::uniform_int_distribution<int> distribution(0, neighbor_count -
1);
      int picked_neighbor_index = distribution(generator);


      std::pair<Cell*, Cell::Direction> picked_neighbor = unvisited_neigh-
bors[picked_neighbor_index];
      current_cell->RemoveSideConnection(picked_neighbor.first,
picked_neighbor.second);
      unfixed_cells_.push_back(picked_neighbor.first);
    }
    else
      unfixed_cells_.pop_back();
  }
}
```

Листинг 8 – Класс персонажа

```
class PlayerView : public ViewBase, public sf::Drawable
{
public:

  PlayerView(EntityStrongPtr p) { player = p; };
  // Initialize methods
  bool InitializeView();
  bool InitializeParameters(json& parameters_json);
  void Update(float delta) override;
  void SetEnabled(bool flag) override;
  void SetPosition(sf::Vector2f position) override;
void SetVelocity(float velocity) override { current_velocity = velocity; }
  void LockOnTarget(EntityStrongPtr entity) override;
  void LockOnMovement() override;
  bool IsAlive() override;
  sf::Vector2f GetPosition();
  void SetMouseEntity(EntityStrongPtr mouse) { mouse_entity = mouse; }
  bool SetWeapon(EntityStrongPtr weapon_entity);
```

```
  void SetHP(int hp);
  void IncreaseDamage(int damage);
  void IncreaseHP(int addition);
  int GetLevel();
  int GetLevelsToUp();
  float GetNextLevelExperience();
  int GetExperience();
  int GetDamageLevel();
  float GetWeaponDamage();
  int GetHPLevel();
  float GetHP();
  void SetHP(float hp);
  float GetMaxHP();
  void IncreaseExperience(int experience);
  void IncreaseDamageLevel();
  void IncreaseHPLevel();
  EntityWeakPtr GetPlayerEntity() { return EntityWeakPtr(player); }
private:
  void HandleInput(float delta);
  virtual void draw(sf::RenderTarget& target, sf::RenderStates states)
const;
  EntityStrongPtr player;
  std::weak_ptr<BodyComponent> body_ptr;
  std::weak_ptr<AnimationComponent> animation_ptr;
  std::weak_ptr<ObserverComponent> observer_ptr;
  std::weak_ptr<SpriteComponent> sprite_ptr;
  std::weak_ptr<CombatComponent> combat_ptr;
  PlayerInputHandler input_handler;
  EntityWeakPtr mouse_entity;
  bool is_running;
  bool is_dashing;
  bool is_crouching;
  float damage_addition;
  float hp_addition;
  float current_velocity;
  float walk_velocity;
  float run_velocity;
  float dash_velocity;
  GameTimer dash_timer;
  GameTimer dash_cooldown_timer;  };
```

Листинг 9 – Метод обработки нажатий клавиш

```
void PlayerView::HandleInput(float delta)
{
  if (!animation_ptr.expired() && !body_ptr.expired())
  {
     auto& animation = animation_ptr.lock();
    auto& body = body_ptr.lock();
    input_handler.Update();
    // get current input state and animation states
    std::vector<int>& input_state = input_handler.GetCurrentState();
    std::map<std::string, size_t>& animation_states = animation->GetAnima-
tionStates();
    float result_velocity = 0.f;
     if (dash_timer.IsReady())
    {
      if   (input_state[PlayerActions::MOVE]   &&   input_state[PlayerAc-
tions::TOGGLE_RUN])
      {
        result_velocity = run_velocity;
```

```
        animation->ChangeState(animation_states["RUN"]);
      }
      else if (input_state[PlayerActions::MOVE])
      {
        result_velocity = walk_velocity;
        animation->ChangeState(animation_states["MOVE"]);
      }
      if (dash_cooldown_timer.IsReady())
      {
        if (input_state[PlayerActions::TOGGLE_DASH])
        {
          result_velocity = dash_velocity;
          dash_timer.Start();
          animation->ChangeState(animation_states["DASH"]);
        }
      }
      else
      {
        dash_cooldown_timer.Update(delta);
      }
    }
    else
    {
      result_velocity = dash_velocity;
      dash_timer.Update(delta);

      animation->ChangeState(animation_states["DASH"]);
      if (dash_timer.IsReady())
      {
        dash_cooldown_timer.Start();
      }
    }
 if (result_velocity == 0)
       animation->ChangeState(animation_states["IDLE"]);

     //change weapon direction
     if (!combat_ptr.expired() && !observer_ptr.expired() && !body_ptr.ex-
 pired())
    {
      auto& combat = combat_ptr.lock();
      auto& observer = observer_ptr.lock();
      auto& body = body_ptr.lock();

      sf::Vector2f target_direction = observer->GetTargetDirection();
      sf::Vector2f target_position =
body->GetPosition() + target_direction;

      if (combat->IsReadyToAttack())
      {
        combat->PointTo(target_position);
      }
      if (input_state[PlayerActions::ATTACK])
      {
        combat->Attack();
      }
    }

    sf::Vector2f result_vector = input_handler.GetResultVector();
    result_vector *= result_velocity;
    body->ApplyLinearImpulse(result_vector);
  }
}
```

Листинг 9 – Класс противника

```
class EnemyView : public ViewBase, public sf::Drawable
{
  enum ActionState
  {
    PATROL,
    CHASE,
    ATTACK };
public:
  EnemyView(EntityStrongPtr enemy_entity) { enemy = enemy_entity; }
  ~EnemyView() = default;
  bool InitializeView();
  bool InitializeParameters(json& parameters_json);
  void Update(float delta) override;
  void SetEnabled(bool flag) override;
  void SetPosition(sf::Vector2f position) override;
  void SetVelocity(float velocity) override { current_velocity_ = velocity;
}
  void LockOnTarget(EntityStrongPtr entity) override;
  void LockOnMovement() override;
  bool IsAlive() override;
  bool SetEntityFov(EntityStrongPtr fov_cone_entity);
  bool SetWeapon(EntityStrongPtr weapon_entity);
  void SetHP(int hp);
  void IncreaseDamage(int damage);
void IncreaseHP(int addition);
  EntityWeakPtr GetEnemyEntity() { return EntityWeakPtr(enemy); }

private:
  void UpdateFov(float delta);
void draw(sf::RenderTarget& target, sf::RenderStates states) const;
void UpdateAnimationState();
EntityStrongPtr enemy;
EntityStrongPtr fov_cone;
  float attack_trigger_distance;
  std::weak_ptr<BodyComponent> body_ptr;
  std::weak_ptr<ObserverComponent> observer_ptr;
  std::weak_ptr<CombatComponent> combat_ptr;
  std::weak_ptr<AnimationComponent> animation_ptr;
  std::weak_ptr<SpriteComponent> sprite_ptr;
  bool is_animated;
  bool is_moving;
  std::weak_ptr<BodyComponent> fov_body_ptr;
  std::weak_ptr<ObserverComponent> fov_observer_ptr;
  std::function<void(Entity*, float)> fov_update_function;

  float current_velocity_;
  float walk_velocity_;
  float chase_velocity_;

  ActionState ai_state_;
  b2Vec2 patrol_direction_;
  b2Vec2 player_last_position_;
  b2Vec2 player_last_direction_vector_;

  GameTimer chase_timeout_;
  GameTimer patrol_wait_timer_;
};
```

Листинг 10 – Компонент подземелья

```
class DungeonComponent : public Component, public sf::Drawable
{
  inline static const size_t ENEMY_POOL_SIZE = 40;
public:
  struct DungeonFloor { sf::RenderTexture floor_texture;
    std::vector<b2BodyDef> blocks;
    std::vector<b2FixtureDef> fixtures;
    std::vector<b2PolygonShape> polygons;
    std::vector<b2Body*> bodies;
    sf::Vector2f player_pos;
    sf::Vector2f boss_pos;
    Map cell_map_;
    size_t seed; };
  inline static ComponentID COMPONENT_ID = Components::DungeonComponentID;

  DungeonComponent() = default;
  virtual ~DungeonComponent() = default;
  void Update(float delta) override;
  bool Init(json::iterator& component_data) override;
  void PostInit() override;
  // Should be overriden by the interface class
  ComponentID GetComponentID() const override { return COMPONENT_ID; };

bool InitializeDungeon(b2World* world);
  void CreateCurrentFloor();
  sf::Vector2f GetPlayerSpawnPosition();
  void IncreaseFloor(); sf::RenderTexture& GetCurrentFloorTexture() { re-
turn floors_[current_floor_]->floor_texture; };
  void SetEnemyPool(std::vector<std::shared_ptr<EnemyView>>& pool);
  int GetPoolMaxSize() { return enemy_pool_max_; }
  int GetPoolSize() { return enemy_pool_->size(); }
  void SetAsBoss(std::shared_ptr<EnemyView> enemy_view);
  void IncreaseDungeonLevel();
   void SetDungeonLevel(int new_level) { dungeon_level_ = new_level; };
  int GetDungeonLevel() { return dungeon_level_; };
  int GetCurrentFloorNumber() { return current_global_floor_; }
  void ResetDungeon(); bool IsFloorCleared();
private:
  int cell_size;
  int draw_offset_x;
  int draw_offset_y;
  void SpawnEnemies();
  void DestroyFloor();
  void draw(sf::RenderTarget& target, sf::RenderStates states) const;
  int enemy_pool_max_;
  std::vector<std::shared_ptr<EnemyView>>* enemy_pool_;
  std::shared_ptr<EnemyView> floor_boss_;
  int floor_enemy_count_;
  float enemy_health_addition_;
  float enemy_damage_addition_;
  float boss_health_addition_;
  float boss_damage_addition_;
  int floor_count_;
  int dungeon_level_;
  int floors_per_level_;
  int current_floor_;
  int current_global_floor_;
  sf::Vector2i floor_size_;
  sf::Vector2i min_room_size_;
  sf::Vector2i max_room_size_;
```

```
  std::vector<DungeonFloor*> floors_;
  b2World* world;
  sf::Texture south_wall_texture;
  sf::Texture north_wall_texture;
  sf::Texture north_wall_thin_texture;
  sf::Texture wall_floor_connection_texture;
  sf::Texture corner_north_east_texture;
  sf::Texture room_floor_texture;
  sf::Texture dungeon_floor_texture;
  sf::Sprite south_wall_;
  sf::Sprite south_thin_wall_;
  sf::Sprite east_wall_;
  sf::Sprite east_thin_wall_;
  sf::Sprite west_wall_;
  sf::Sprite west_thin_wall_;
  sf::Sprite north_wall_;
  sf::Sprite north_thin_wall_;
  sf::Sprite corner_north_east;
  sf::Sprite room_floor_;
  sf::Sprite dungeon_floor_;
  sf::Sprite wall_floor_connection_;
};
```