Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра программного обеспечения информационных технологий

Дисциплина: Конструирование программного обеспечения (КПО)

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту на тему

Компьютерная игра «Соул Найт»

БГУИР КП I–40 01 01 020 ПЗ

Выполнил

студент гр.251003 Панкратьев Е.С.

Проверил: Шостак Е.В.

Минск 2023

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

УТВЕРЖДАЮ

Заведующий кафедрой ПОИТ

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись)

Лапицкая Н.В. 2023г.

ЗАДАНИЕ

по курсовому проектированию

Студенту Панкратьеву Егору Сергеевичу \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. Тема работы Компьютерная игра «Соул Найт»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

2. Срок сдачи законченной работы *25.12.2023г.*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

3. Исходные данные к работе *Среда программирования: C++. Игра в реальном времени с управлением персонажем. Создание разнообразных врагов с уникальными характеристиками и стратегиями поведения. Игровое взаимодействие между игроком и врагами. Реализация динамичного игрового поля для сражений.*

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

*Введение*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*1 Анализ предметной области \_*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*2 Проектирование программного средства* \_\_\_\_\_\_*\_\_\_\_\_\_\_\_\_\_\_*

*3 Разработка программного средства средства*

*4 Тестирование и проверка работоспособности программного средства \_\_\_*

*5 Руководство по использованию программного средства\_\_\_\_\_*

*Заключение \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Список использованных источников\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Приложения* \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

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

*Схема алгоритма в формате А1*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

6. Консультант по курсовой работе *Шостак Е.В.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

7.Дата выдачи задания *07.09.2023г.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

Календарный график работы над проектом на весь период проектирования (с обозначением сроков выполнения и процентом от общего объёма работы):

раздел 1 к 15.09.2023 – 15 % готовности работы;

раздел 2 к 15.10.2023 – 30 % готовности работы;

разделы 3,4 к 15.11.2023 – 60 % готовности работы;

раздел 5 к 15.12.2023 – 90 % готовности работы;

оформление пояснительной записки и графического материала к 17.12.2023 – 100 % готовности работы.

Защита курсового проекта с 13 по 25 декабря 2023 г.––––––––––––––––

РУКОВОДИТЕЛЬ *\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Шостак Е.В.*

*(подпись)*

Задание принял к исполнению *\_\_\_\_\_\_\_\_Панкратьев Е.С. 07.09.2023г.*

*(дата и подпись студента)*

Содержание

[Введение 5](#_Toc154137965)

[1 Анализ предметной области 6](#_Toc154137966)

[1.1 Анализ существующих аналогов 6](#_Toc154137967)

[1.1.1 Игровое приложение «Bomberman» 6](#_Toc154137968)

[1.1.2 Игровое приложение «Dig Dug» 7](#_Toc154137969)

[1.1.3 Игровое приложение «Maze Craze: A Game of Cops 'n Robbers» 8](#_Toc154137970)

[1.2 Постановка задачи 9](#_Toc154137971)

[2 Проектирование и разработка программного средства 10](#_Toc154137972)

[2.1 Структура программы 10](#_Toc154137973)

[2.2 Проектирование интерфейса программного средства 10](#_Toc154137974)

[2.2.1 Главное меню 11](#_Toc154137975)

[2.2.2 Окно приложения для одиночной игры 11](#_Toc154137976)

[2.2.3 Окно приложения для многопользовательской игры 12](#_Toc154137977)

[2.3 Проектирование функционала программного средства 13](#_Toc154137978)

[2.3.1 Инициализация игрового окна 13](#_Toc154137979)

[2.3.2 Обновление игрового состояния 16](#_Toc154137980)

[2.3.3 Обновление состояния главного героя 19](#_Toc154137981)

[3 Разработка программного средства 23](#_Toc154137982)

[3.1 Обновление анимации врагов 23](#_Toc154137983)

[3.2 Обработчик нажатий клавиш 23](#_Toc154137984)

[3.3 Обновление состояния главного героя 24](#_Toc154137985)

[3.4 Инициализация игрового окна 25](#_Toc154137986)

[4 Тестирование программного средства 27](#_Toc154137987)

[4.1 Результаты игры 27](#_Toc154137988)

[4.2 Увеличение количества очков при собирании точек 27](#_Toc154137989)

[4.3 Добавление очков за поедание врага 29](#_Toc154137990)

[5 Руководство пользователя 31](#_Toc154137991)

[5.1 Игровой процесс одиночного режима 31](#_Toc154137992)

[5.2 Игровой процесс многопользовательского режима 32](#_Toc154137993)

[Заключение 34](#_Toc154137994)

[Список использованной литературы 35](#_Toc154137995)

[Приложение A 36](#_Toc154137996)

[Приложение Б 42](#_Toc154137997)

[Приложение В 50](#_Toc154137998)

[Приложение Г 59](#_Toc154137999)

Введение

«Соул Найт» – захватывающая компьютерная игра, воплощающая современные тенденции в мире развлечений. В этом захватывающем виртуальном мире игроки погружаются в уникальное приключение, где умение управлять персонажем становится ключевым элементом выживания.

В игре «Соул Найт» разнообразие врагов поражает своей необычной характеристикой и стратегиями поведения, создавая динамичные сражения на каждом уровне. Игровое взаимодействие между игроком и врагами требует тактического мышления и мастерства владения разнообразным арсеналом оружия, предназначенного для сражений в реальном времени.

Уникальность «Соул Найт» также заключается в создании захватывающего игрового поля, наполненного тайнами, ловушками и неожиданными сюжетными поворотами. Каждый уровень становится увлекательным испытанием для навыков игрока, где необходимо проявить смекалку и реакцию для преодоления вызовов.

«Соул Найт» предоставляет уникальный игровой опыт, объединяя в себе интенсивные сражения, стратегическое мышление и взаимодействие с захватывающим виртуальным миром.

Цель проекта – разработка захватывающей компьютерной игры, объединяющей технологии и креативность. Разработка включает в себя использование разнообразных аспектов программирования: графический интерфейс, управление игровой логикой, алгоритмы и структуры данных. Проект предоставляет возможность создать увлекательный виртуальный мир, где технологии и развлечения переплетаются, обеспечивая интересный опыт для игроков и разработчиков.

# Анализ предметной области

## Анализ существующих аналогов

### Игровое приложение «KnightNight»

«KnightNight» – это захватывающая аркадная игра, предлагающая уникальный игровой опыт. Разработанная с любовью к деталям, она объединяет классические элементы жанра с современной графикой и инновационными механиками.

Основные характеристики игры:

* загадочные лабиринты: игроки исследуют захватывающие лабиринты, полные загадок и опасностей;
* многопользовательский режим: возможность соревноваться с друзьями или другими игроками, добавляя элемент соревновательности и веселья;
* разнообразие врагов: уникальные вражеские персонажи, каждый с собственными характеристиками и стратегиями, создают динамичные бои;
* адаптация на различные платформы: играйте на любом устройстве, сохраняя при этом качество и веселье на всех поддерживаемых платформах.

Интерфейс приложения представлен на рисунке 1.1.



Рисунок . – Интерфейс игрового приложения «KnightNight»

«KnightNight» сохраняет дух классических аркадных игр, предоставляя игрокам увлекательное приключение в мире рыцарей и тайн.

### Игровое приложение «Dig Dug»

«Survivor.io» – захватывающая многопользовательская онлайн-игра, погружающая игроков в интенсивные сражения за выживание. Созданная с учетом последних тенденций в жанре Battle Royale, эта игра предлагает уникальный опыт в выживании, внедряя виртуальных героев в динамичные и опасные бои.

Основные особенности игры:

* битва за выживание: игроки соревнуются в огромном виртуальном мире, где только самый ловкий и умелый сможет остаться в живых;
* многопользовательский формат: возможность сражаться с реальными игроками со всего мира, делая каждый матч уникальным и захватывающим;
* разнообразные локации: игровая карта предлагает разнообразные местности – от городов до лесов, создавая разнообразные условия для выживания;
* стратегическое взаимодействие: создайте союзы с другими игроками или идите на все, чтобы выжить. Выбор за вами.

Интерфейс приложения представлен на рисунке 1.2.

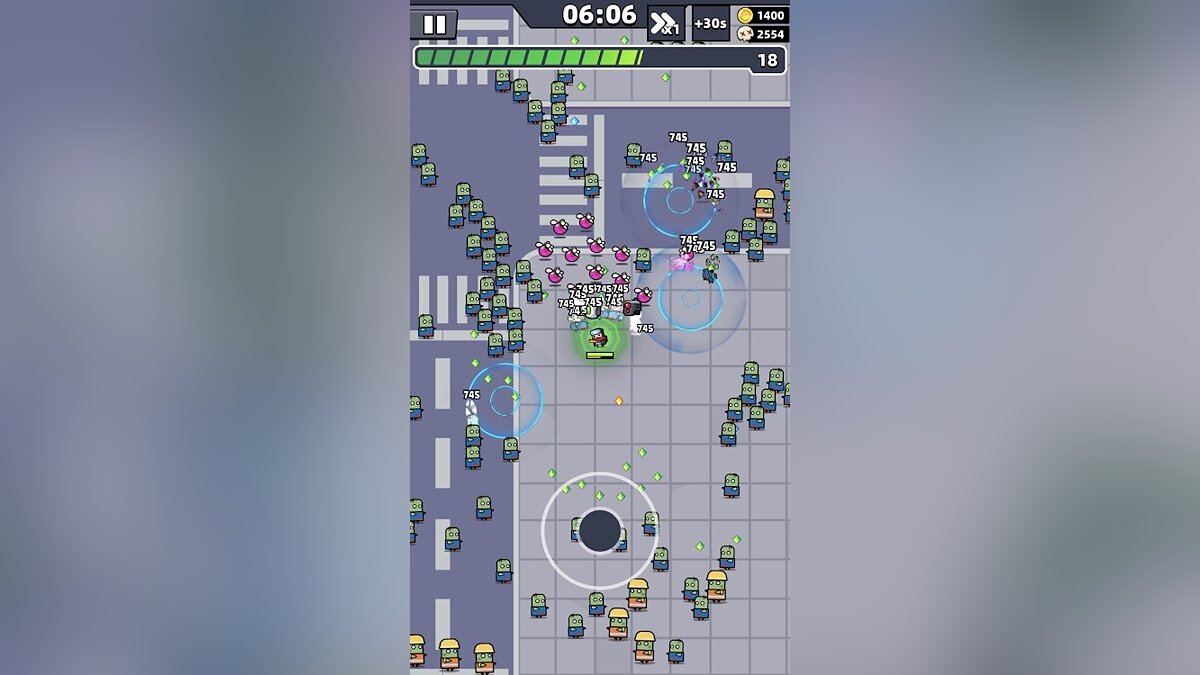


Рисунок . – Интерфейс игрового приложения «Survivor.io»

«Survivor.io» предоставляет игровой опыт, где стратегии игроков и их навыки будут определять вашу судьбу в захватывающем мире выживания.

### Игровое приложение «Pocket Rogues: Ultimate»

«Pocket Rogues: Ultimate» – захватывающая ролевая игра, погружающая игроков в захватывающие подземелья и приключения. Эта ультимативная версия предлагает улучшенный геймплей и новые возможности, делая каждое приключение уникальным и захватывающим.

Основные особенности игры:

* бескрайние подземелья: исследуйте бескрайние лабиринты, наполненные тайнами, сокровищами и монстрами, создавая непредсказуемый опыт игры;
* уникальные классы персонажей: выберите из разнообразных классов, каждый со своими уникальными способностями и стилем игры;
* глубокая прогрессия: улучшайте своего персонажа, собирайте снаряжение и развивайте навыки, чтобы стать настоящим мастером подземелий;
* рогалик-элементы: случайно генерируемые уровни, перманентная смерть и случайные события создают непредсказуемый и захватывающий игровой опыт.

Интерфейс приложения представлен на рисунке 1.3.



Рисунок .3 – Интерфейс игрового приложения «Pocket Rogues: Ultimate»

«Pocket Rogues: Ultimate» приглашает вас в захватывающий мир приключений, где каждый шаг может означать новую угрозу или возможность.

## Постановка задачи

В рамках данной курсовой работы планируется разработать компьютерную игру для платформы Windows. В процессе разработки должны быть реализованы базовые элементы игры:

* игровое поле: Реализовано динамичное и разнообразное игровое поле, включающее в себя различные локации, лабиринты и арены. Окружение было организовано таким образом, чтобы создавать увлекательные сражения и тактические вызовы для игрока;
* разнообразие врагов: Созданы уникальные вражеские персонажи, адаптированные к различным уровням. Враги обладают разнообразными характеристиками и стратегиями поведения, что добавляет тактические элементы в игру;
* иерархия уровней: Внедрена система иерархии уровней, где каждый последующий уровень становится более сложным и насыщенным врагами. Разнообразие врагов и их сила увеличиваются с уровнем, что обеспечивает постепенное увеличение сложности и поддерживает интерес игрока на протяжении всего процесса;
* арены: Арены на уровнях были разнообразными и хаотичными, обеспечивая интересные бои и увлекательные моменты для игрока.

Эти элементы, в совокупности, создают атмосферу и геймплей, соответствующие ключевым характеристикам.

Для разработки программного средства будет использоваться язык программирования C++ вместе с библиотекой SFML (Simple and Fast Multimedia Library), а также среда разработки JetBrains Rider.

# Проектирование и разработка программного средства

## Структура программы

При разработке приложения будут использованы следующие классы:

1. Launcher – это главный класс, управляющий запуском и основными параметрами игры, включая окно, переключение полноэкранного режима, работу с файлами, учет прогресса игрока, игровое управление и меню.
2. MenuManager – это класс, управляющий отображением и взаимодействием с главным меню игры, обрабатывающий события клавиатуры, включая создание кнопок, управление полноэкранным режимом и отображение текста.
3. SimpleSprite – это базовый класс для спрайтов в игре, содержащий информацию о z-индексе, прямоугольнике текстуры, и текстуре, а также предоставляющий методы для установки текстуры, получения размеров и сравнения по z-индексу.
4. AnimatedSprite – это класс, наследующий от SimpleSprite, предназначенный для анимированных спрайтов в игре. Он содержит информацию о кадрах анимации, ширине кадра, прошедшем времени, а также предоставляет методы для изменения состояния анимации и проверки перезапуска анимации.
5. Quadtree – это шаблонный класс, представляющий собой квадродерево для эффективного поиска и управления элементами с использованием полигональных данных. Класс содержит методы для вставки, удаления и поиска коллизий элементов, используя квадратные области.
6. QuadtreeNode – это шаблонный класс, представляющий узел в квадродереве для управления коллизиями и поиском элементов типа T. Класс содержит методы для вставки, удаления и поиска коллизий в пределах своей границы с использованием полигональных данных. Он также может подразделяться на четыре дочерних узла, обеспечивая оптимизацию поиска.
7. Polygon – это базовый класс, представляющий многоугольник в двумерном пространстве. Он содержит методы для перемещения, вращения и получения различных характеристик формы, таких как центр, угол поворота и размеры ограничивающего прямоугольника.
8. Rectangle – это класс, представляющий собой прямоугольник, являющийся производным от базового класса "Polygon". Он может быть создан на основе данных о вращенном или выровненном прямоугольнике, а также по двум точкам – фокусу и половине размера.
9. Line – это класс, представляющий собой линию, производный от базового класса "Polygon". Он создается на основе двух точек - начальной и конечной. Класс реализует методы для вычисления центра, угла поворота и размеров ограничивающего прямоугольника для данной линии.
10. Axis – это класс, представляющий собой ось в двумерном пространстве. Он создается на основе двух точек - начальной и конечной.
11. CollisionManager – это класс, ответственный за управление и обработку коллизий в игровом мире. Он использует таблицу коллизий и методы для фильтрации, обнаружения и обработки столкновений между элементами. Класс также предоставляет функциональность для проверки видимости линии от одного игрового элемента к другому.
12. GameMaster – это класс, управляющий игровым процессом. Он создает и управляет различными компонентами, такими как система игры, карты сущностей, обработчики ввода и обновления игры. Класс также отвечает за создание и управление игровым циклом.
13. SpriteDrawer – это класс, отвечающий за отрисовку спрайтов. Он используется для отображения элементов, хранящихся в наборе коллизий (которые были получены с коллизией прямоугольника окна), с учетом их приоритета.
14. GameFieldCreator – это класс, отвечающий за создание игрового поля в игре. Его функциональность включает в себя генерацию и распределение комнат, определение типов комнат, управление процессом создания игрового мира и его элементов. Этот класс играет ключевую роль в формировании уровней игры, обеспечивая разнообразие и интересные взаимодействия на игровом поле.
15. Identifiable – это базовый класс, предоставляющий идентификационный номер для объектов в игровом мире. Каждый объект, производный от этого класса, получает уникальный идентификационный номер при создании.
16. Element – это базовый класс, наследуемый от класса Identifiable, представляющий элемент в игровом мире. Этот класс содержит информацию о форме и визуальном представлении элемента, а также параметры трансформации для его отображения.
17. ReplaceableSprites – это класс, представляющий собой элемент в игровом мире, который может иметь несколько вариантов отображения, между которыми можно переключаться во время выполнения игры. Этот класс является производным от класса Element и расширяет его возможности, добавляя поддержку нескольких спрайтов.
18. FileManager – это класс, отвечающий за управление файлами в игровой системе. Он предоставляет функциональность для сохранения и загрузки данных игрока (прогресса) и управления настройками управления.
19. Entity – это базовый класс, представляющий сущность в игровой системе. Он наследуется от класса Identifiable и содержит информацию о сущности, такую как здоровье, броня и идентификационные данные.
20. FightingEntity – это класс, представляющий боевую единицу в игровой системе. Он наследуется от класса Entity и содержит информацию о персонаже (Character), его оружии (Gun) и параметры боя, такие как отсутствие урона в течение определенного времени и возможность регенерации здоровья.
21. EntityComponent – это базовый класс, представляющий компонент сущности в игровой системе. Он наследуется от класса Identifiable и содержит информацию о номере компонента и его уникальном идентификаторе.
22. Gun – это класс, представляющий оружие в игровой системе. Он наследуется от класса EntityComponent и содержит информацию об элементе, представляющем оружие, а также о параметрах оружия.
23. Character – это класс, представляющий персонажа в игре. Он наследуется от класса EntityComponent и содержит информацию об элементе, представляющем персонажа, а также характеристиках персонажа.
24. BulletCasing – это класс, представляющий оболочку пули в игре. Он наследуется от класса EntityComponent и содержит информацию об элементе, представляющем оболочку пули, а также статистику пули.
25. Bullet – это класс, представляющий сущность пули в игровой системе. Он наследуется от класса Entity и расширяется добавлением вектора скорости и объекта BulletCasing, представляющего внешний вид пули.
26. Enemy – это класс, представляющий враждебного персонажа в игровой системе. Он наследуется от класса FightingEntity и включает в себя объект EnemyManagement, управляющий врагом.
27. Player – это класс, представляющий игрока в игре. Он наследуется от класса FightingEntity и содержит информацию о персонаже, его оружии, управлении и времени последней смены оружия.
28. Executor – это базовый абстрактный класс, предоставляющий интерфейс для выполнения действий в игровом контексте. Он содержит виртуальную функцию handle, которую нужно реализовать в производных классах для обработки временных изменений в игре.
29. AnimationExecutor – это класс, который является реализацией Executor. Он отвечает за обновление анимаций в игре.
30. BulletExecutor – это класс, который является реализацией Executor. Он отвечает за обновление полета пуль в игре.
31. PlayerExecutor – это класс, который является реализацией Executor. Он отвечает за обработку действий игрока в игре.
32. EnemyExecutor – это класс, который является реализацией Executor. Он отвечает за обработку действий врагов в игре.

## Проектирование интерфейса программного средства

### Главное меню

Главное окно приложения представляет собой окно с элементами выбора начала игры, настроеек, информации об авторе и выход. Макет главного меню приложения представлен на рисунке 2.1.

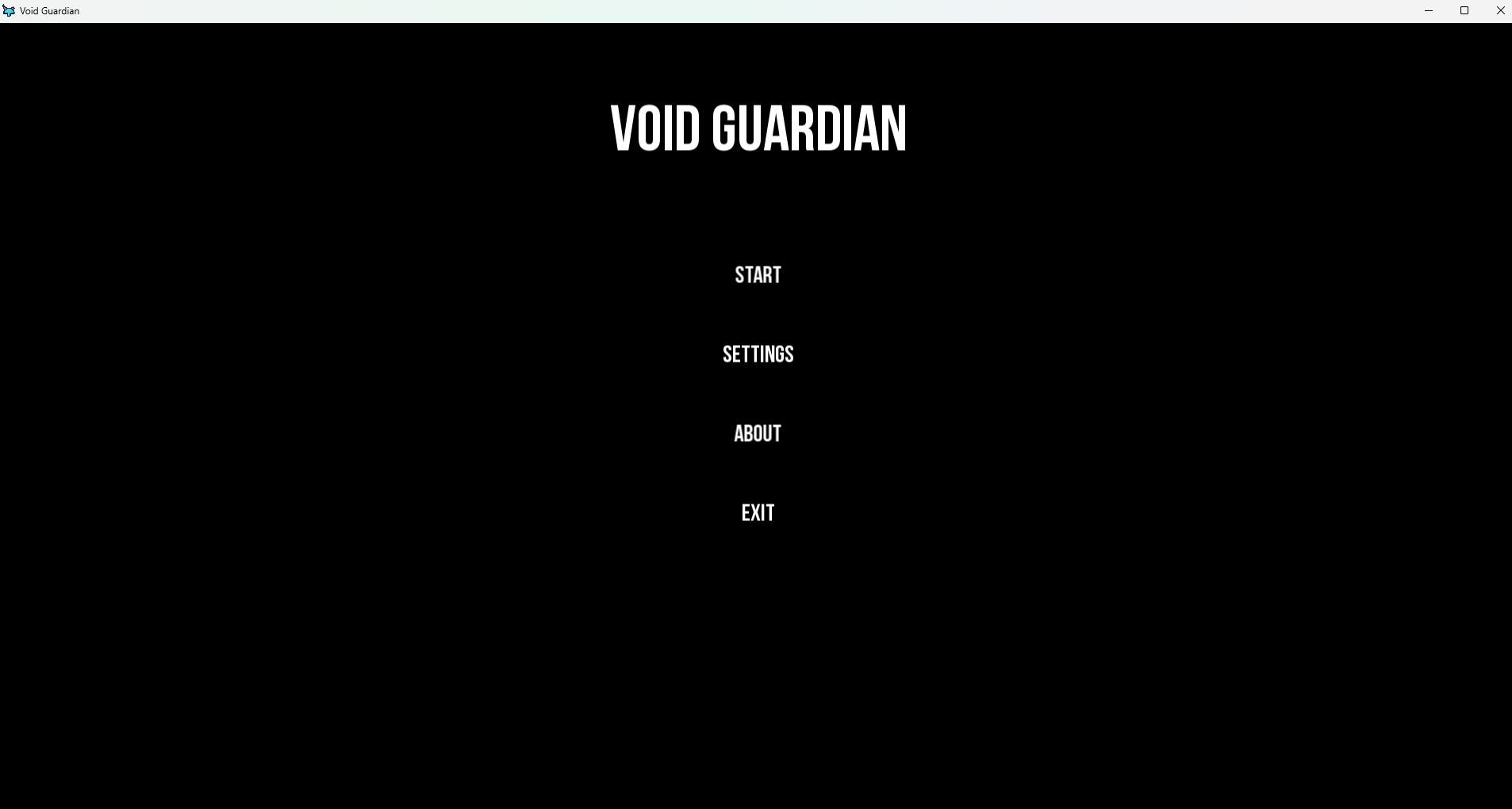


Рисунок . – Главное меню приложения

### Окно приложения игры

Окно приложения для игры представляет собой игровое поле, на котором расположен лабиринт, также главный герой.

Макет игровой карты представлен на рисунке 2.2.



Рисунок ..1 – Окно приложения игры

При заходе на арену будут появляться враги, контролируемые классом EnemyExecutor, который отвечает за управление их поведением, создание, а также обработку взаимодействий с игровым полем и другими элементами.



Рисунок ..2 – Окно приложения игры с врагами

## Проектирование функционала программного средства

Для нормального функционирования предстоящей компьютерной игры необходимо внедрить следующие ключевые функции:

1. Инициализация игрового окна:

* Создание и настройка игрового окна для отображения графики и взаимодействия с пользователем.
* Установка параметров разрешения, режима отображения и других параметров окна.
* Загрузка необходимых ресурсов, таких как текстур, звуков и других игровых элементов.

1. Обновление игрового состояния:

* Обновление игровой логики и механики в соответствии с текущим состоянием игры.
* Обработка ввода от пользователя, такого как клавиатура, мышь или контроллеры.
* Детекция и обработка столкновений между игровыми объектами.
* Управление анимациями и другими визуальными эффектами.
* Обновление и отображение пользовательского интерфейса (UI).

1. Обновление состояния главного героя:

* Обработка пользовательского ввода для управления главным героем.
* Реализация логики перемещения, взаимодействия с объектами и других действий героя.
* Управление здоровьем, боеприпасами, энергией и другими ресурсами главного персонажа.
* Обработка событий, связанных с действиями героя, такими как стрельба, использование предметов и т.д.

Эти функции являются важными компонентами для обеспечения стабильной и интересной игровой динамики. Их успешная реализация содействует созданию увлекательного и полноценного игрового опыта.

### Инициализация игрового окна

В игровых приложениях функция инициализации игрового окна играет ключевую роль, поскольку она предоставляет пользователю визуальный доступ к игровому полю и взаимодействие с ним.

Блок схема метода представлена на рисунке 2.7.



Рисунок . – Блок-схема метода Launcher::startWindow (часть 1)



Рисунок . – Блок-схема метода Launcher::startWindow (часть 2)

### Создание игрового меню

Построение игрового меню необходимо для создания удобного и интуитивно понятного интерфейса, который позволит игроку взаимодействовать с игрой. Этот процесс включает в себя отображение различных опций, таких как начало новой игры, настройки, выход из игры и другие.

Блок-схема данного алгоритма представлена на рисунке 2.9.



Рисунок 2.9 – Блок-схема метода MenuManager::createMenu

### Обновление состояния главного героя

Данный метод выполняет обновление внутреннего состояния персонажа в игре, а именно главного героя. Он реагирует на события, происходящие в игре, такие как движение мыши, и осуществляет необходимые изменения в состоянии героя.

Блок схема данного метода представлена на рисунке 2.10.



Рисунок 2.10 – Блок-схема метода PlayerExecutor::handle

### Обновление состояния врагов

Данный метод отвечает за обновление внутреннего состояния врагов в игре. Он реагирует на различные события, происходящие в игровом мире, и вносит соответствующие изменения в состояние каждого врага. В основном, метод обрабатывает логику перемещения, атаки и другие действия врагов в зависимости от текущей ситуации в игре.

Блок схема данного метода представлена на рисунке 2.11.



Рисунок 2.11 – Блок-схема метода EnemyExecutor::handle

### Обновление состояния анимаций

Данный метод занимается обновлением анимаций в игре. Он перебирает все текущие анимации в коллекции и применяет изменения к каждой из них в соответствии с прошедшим временем. Главная цель метода – управление состоянием анимированных спрайтов и их анимаций.

Блок схема данного метода представлена на рисунке 2.12.



Рисунок 2.12 – Блок-схема метода AnimationExecutor::handle

### Игровой цикл

Данный метод представляет собой основной игровой цикл, который обрабатывает каждую итерацию игры. Он выполняет следующие основные шаги для обеспечения корректной работы игры

Блок схема данного метода представлена на рисунке 2.13.



Рисунок 2.13 – Блок-схема метода GameLoop::nextIteration

# Разработка программного средства

## Экспоненциального распределения

Данная функция представляет собой генератор случайных чисел с использованием экспоненциального распределения. Она используется для управления вероятностью генерации больших чисел в заданном диапазоне. Чем больше значение параметра \_exponent, тем меньше вероятность генерации больших чисел. Это полезно, например, при определении количества ботов или сложности игрового уровня, где нужно управлять распределением для достижения желаемой балансировки в игровом процессе.

int ExponentGenerator::generate() const {

const auto random\_value =

RandomGenerator::getRandom(\_distribution);

const float raised\_value = std::pow(random\_value,

\_exponent);

const auto range = static\_cast<float>(\_max - \_min);

return \_min + static\_cast<int>(std::round(raised\_value \*

range));

}

## Квадридерево

Квадридерево представляет собой структуру данных для эффективного хранения и поиска элементов в двумерном пространстве. Каждый узел квадридерева делит пространство на четыре подобласти, что позволяет организовать быстрый доступ к элементам в определенной области.

Эти методы обеспечивают эффективную вставку, удаление и поиск элементов в пространстве, используя структуру квадридерева.

template <typename T, typename Enabler>

bool QuadtreeNode<T, Enabler>::insert(T \*element, const Axes &axes) {

if (CollisionDetection::hasCollision(\_boundary,

element->getPolygon(),

\_boundary.getAxes(), axes)) {

if (isSubdivide()) {

\_total\_elements = 0;

bool result = false;

for (size\_t i = 0; i < CHILD\_COUNT; ++i) {

result |= \_children[i].insert(element,

axes);

\_total\_elements +=

\_children[i].\_total\_elements;

}

return result;

}

else {

++\_total\_elements;

\_elements->insert(element);

if (\_elements->size() >= \_capacity) {

subdivide();

}

return true;

}

}

return false;

}

template <typename T, typename Enabler>

bool QuadtreeNode<T, Enabler>::remove(T \*element, const

Axes &axes) {

if (CollisionDetection::hasCollision(\_boundary,

element->getPolygon(),

\_boundary.getAxes(), axes)) {

if (isSubdivide()) {

\_total\_elements = 0;

bool result = false;

for (size\_t i = 0; i < CHILD\_COUNT; ++i) {

result |= \_children[i].remove(element,

axes);

\_total\_elements +=

\_children[i].\_total\_elements;

}

if (\_total\_elements <= \_capacity / 2) {

mergeWithChildren();

}

return result;

}

else {

if (const auto it = \_elements-

>find(element); it != \_elements->end()) {

\_elements->erase(it);

--\_total\_elements;

return true;

}

else

return false;

}

}

return false;

}

template <typename T, typename Enabler>

void QuadtreeNode<T, Enabler>::fillCollisionSet(const

Polygon &polygon, const Axes &axes, CollisionSet

&collisions\_info) const {

if (isSubdivide()) {

for (size\_t i = 0; i < CHILD\_COUNT; ++i) {

if

(CollisionDetection::hasCollision(\_children[i].\_boundary, polygon,

\_children[i].\_boundary.getAxes(), axes)) {

\_children[i].fillCollisionSet(polygon, axes, collisions\_info);

}

}

}

else {

for (auto \*other\_element : \*\_elements) {

if (collisions\_info.find(other\_element) == collisions\_info.end()) {

const Polygon &other\_polygon = other\_element->getPolygon();

Axes other\_axes;

CollisionDetection::fillAxes(other\_polygon, other\_axes);

if (CollisionDetection::hasCollision(polygon, other\_polygon,

axes, other\_axes))

collisions\_info.insert(other\_element);

}

}

}

}

## Тригонометрия

Тригонометрия представляет собой пространство имен, предоставляющее набор функций для работы с углами и тригонометрическими вычислениями. Эти функции часто используются в игровом программировании и графике для работы с углами поворота, направлениями и другими геометрическими аспектами.

float Trigonometry::degreesToRadians(const float degrees) {

return degrees \* DEGREES\_TO\_RADIANS;

}

float Trigonometry::radiansToDegrees(const float radians) {

return radians \* RADIANS\_TO\_DEGREES;

}

float Trigonometry::sinDegrees(const float degrees) {

return std::sin(degreesToRadians(degrees));

}

float Trigonometry::cosDegrees(const float degrees) {

return std::cos(degreesToRadians(degrees));

}

float Trigonometry::sinRadians(const float radians) {

return std::sin(radians);

}

float Trigonometry::cosRadians(const float radians) {

return std::cos(radians);

}

float Trigonometry::clampAngle(float angle, const float min\_range, const float max\_range) {

while (angle < min\_range)

angle += max\_range - min\_range;

while (angle >= max\_range)

angle -= max\_range - min\_range;

return angle;

}

bool Trigonometry::isAngleInQuadrant2Or3(float angle) {

angle = clampAngle(angle);

return angle > M\_PI\_2\_ && angle < M\_3\_MULT\_PI\_2\_;

}

bool Trigonometry::isAngleInQuadrant2Or3(const sf::Vector2f &vector) {

return isAngleInQuadrant2Or3(GeomAuxiliaryFunc::calcAngle(vector));

}

## Обработчик коллизий

Данные функции представляет обработку коллизий между элементами в игровом мире. Эти методы решают проблему пересечения объектов, предотвращая их наложение друг на друга и обеспечивая корректное взаимодействие.

sf::Vector2f calcCollisionNormal(const sf::Vector2f& collision\_point, const sf::Vector2f& center) {

sf::Vector2f result(collision\_point - center);

GeomAuxiliaryFunc::normalize(result);

return result;

}

void CollisionResolution::separateElement(const Element& element, const CollisionResult &collision\_result) {

const sf::Vector2f collision\_normal(calcCollisionNormal(collision\_result.collision\_point,

element.getPolygon().calcCenter()));

separateElement(element, collision\_normal, collision\_result.overlap);

}

void CollisionResolution::separateElement(const Element& element, const sf::Vector2f &collision\_normal,

const float overlap) {

element.move(collision\_normal \* -overlap);

}

## Кнопка

Данный класс предназначен для отображения и взаимодействия с пользовательским интерфейсом. Он представляет собой кнопку с текстовой меткой, которая реагирует на действия пользователя.

bool Button::contains(const sf::Vector2f &p) const {

return \_text.getGlobalBounds().contains(p);

}

Button::Button(const std::string& label, const sf::Font &font, std::function<void()> &&on\_click\_callback,

const ButtonColors& button\_colors): \_text(label, font, TEXT\_SIZE), \_button\_colors(button\_colors),

\_on\_click\_callback(std::move(on\_click\_callback)) {

\_text.setFillColor(button\_colors.text\_color);

\_text.setOrigin(\_text.getGlobalBounds().width / 2.0f, \_text.getGlobalBounds().height / 2.0f);

}

void Button::draw(sf::RenderWindow& render\_window) const {

render\_window.draw(\_text);

}

void Button::checkClick(const sf::Vector2f& mouse) const {

if (contains(mouse))

\_on\_click\_callback();

}

void Button::setColor(const sf::Vector2f& position) {

if (contains(position)) { \_text.setFillColor(\_button\_colors.active\_color); }

else { \_text.setFillColor(\_button\_colors.text\_color); }

}

void Button::setPos(const sf::Vector2f& p) {

\_text.setPosition(p);

}

## Отрисовка игры

Данный класс ответственный за визуализацию игровых элементов в графическом интерфейсе. Он обеспечивает удобный механизм отрисовки различных объектов на главном игровом окне, создавая при этом визуально привлекательный игровой опыт.

SpriteDrawer::SpriteDrawer(sf::RenderWindow &window, FightingInfoDrawer &&fighting\_info\_drawer, const sf::Color &color):

\_window(&window), \_fighting\_info\_drawer(std::move(fighting\_info\_drawer)), \_color(color) { }

void SpriteDrawer::drawFromPq(SpritesPQ &pq) const {

sf::Sprite sprite;

while (!pq.empty()) {

pq.top()->fillSprite(sprite);

pq.pop();

\_window->draw(sprite);

}

}

void SpriteDrawer::drawAll(const ElementCollisionSet &elements) const {

SpritesPQ pq(elements.begin(), elements.end());

\_window->clear(\_color);

drawFromPq(pq);

\_fighting\_info\_drawer.drawHealthBars(elements);

\_window->display();

}

# Тестирование программного средства

## Запуск игры

Предполагаемый результат: запуск игры.



Рисунок . – Результат при запуске игры

Полученный результат: запуск игры.

## Вход в боевую зону

Предполагаемый результат: закрытие боевой зоны.



Рисунок .2 – Количество очков до собирания точки

Полученный результат: закрытие боевой зоны.

## Убийство врага

Предполагаемый результат: убийство врага.



Рисунок 3.3 – Количество очков до собирания точки

Полученный результат: убийство врага.

## Прохождение локации

Предполагаемый результат: открытие боевой зоны.



Рисунок 3.4 – Количество очков до собирания точки

Полученный результат: открытие боевой зоны.

# Руководство пользователя

## Игровой процесс

Игровой процесс представляет захватывающее приключение в мире динамичных сражений, оружия, и магии. Вот обзор ключевых аспектов игрового процесса:

* открытие и Настройка Игры. Игровой процесс начинается с открытия главного меню. Игрок настроить управление, выбрать персонажа и продолжить игру с того уровня, где он остановился;
* исследование Уровней. Уровни представляют из себя разнообразные местности. Игроки исследуют каждый уровень, сражаясь с волнами врагов;
* бой и Сражения. Центральным элементом игрового процесса являются бои с врагами. Игрок сражается с разнообразными врагами, используя широкий арсенал оружия;
* сбор оружия. Новое оружие, найденное в бою, может помочь с прохождением игры;
* уникальные Персонажи. Игрок имеет возможность выбирать между различными персонажами, каждый из которых обладает своими характеристиками;
* необходимость Выживания. Цель игры — выжить как можно дольше. С увеличением сложности уровней и появлением более сильных врагов, игроки должны постоянно совершенствовать свои навыки и стратегии, чтобы выжить во все более сложных условиях.

Заключение

В ходе данной курсовой работы было разработано игровое приложение с использованием языка программирования C++ и библиотеки SFML. Проект успешно достиг поставленных целей, демонстрируя высокую степень функциональности и производительности.

Основные задачи, такие как создание игрового окна, обновление игрового состояния и главного персонажа, были решены эффективно. Программное средство прошло успешное тестирование, что подтверждает его корректную работу даже в условиях возможных исключительных ситуаций.

Для достижения поставленных целей потребовалось углубленное изучение основ объектно-ориентированного программирования на языке C++, а также активное применение полученных знаний при использовании библиотеки SFML. Этот опыт не только позволил успешно реализовать проект, но и расширил понимание принципов разработки игровых приложений.

В перспективе данное программное средство может быть усовершенствовано путем оптимизации кода и добавления новых функций, что позволит расширить возможности игры и улучшить пользовательский опыт. Создание данного проекта является лишь начальным этапом, и его дальнейшее развитие может внести дополнительный вклад в область разработки игровых приложений.

Список использованной литературы

1. Страуструп Б. Язык программирования C++. – М.: Издательство "Бином-Пресс", 2015. – 352 с.
2. Лафоре Р. Объектно-ориентированное программирование в С++. – СПб.: Питер, 2018. – 816 с.
3. Мэйерс С. Эффективное использование C++. 55 верных способов улучшить структуру и код ваших программ. – М.: Издательский дом "Вильямс", 2017. – 320 с.
4. Чернов С. Программирование на С++ с использованием библиотеки SFML. – Минск: БГУИР, 2019. – 120 с.

Приложение A

**Исходный код модуля QuadtreeNode**

#pragma once

#include "../../include/quadtree/QuadtreeNode.hpp"

template <typename T, typename Enabler>

QuadtreeNode<T, Enabler>::QuadtreeNode(const AlignedRectangleData &data, const size\_t capacity):

\_capacity(capacity), \_boundary(data) {

\_elements->reserve(\_capacity);

}

template <typename T, typename Enabler>

bool QuadtreeNode<T, Enabler>::isSubdivide() const { return \_children; }

template <typename T, typename Enabler>

void QuadtreeNode<T, Enabler>::subdivide() {

const auto &points = \_boundary.getPoints();

const float x\_start = points[0].x;

const float y\_start = points[0].y;

const float x\_last = points[2].x;

const float y\_last = points[2].y;

const float width = (x\_last - x\_start) / 2;

const float height = (y\_last - y\_start) / 2;

\_children = new QuadtreeNode[CHILD\_COUNT]{

QuadtreeNode{{x\_start, y\_start, x\_start + width, y\_start + height}, \_capacity},

QuadtreeNode{{x\_start + width, y\_start, x\_last, y\_start + height}, \_capacity},

QuadtreeNode{{x\_start, y\_start + height, x\_start + width, y\_last}, \_capacity},

QuadtreeNode{{x\_start + width, y\_start + height, x\_last, y\_last}, \_capacity}

};

redistribute();

}

template <typename T, typename Enabler>

void QuadtreeNode<T, Enabler>::redistribute() {

\_total\_elements = 0;

for (auto \*element : \*\_elements) {

const Polygon &polygon = element->getPolygon();

Axes axes;

CollisionDetection::fillAxes(polygon, axes);

for (size\_t i = 0; i < CHILD\_COUNT; ++i) {

\_children[i].insert(element, axes);

}

}

delete \_elements;

for (size\_t i = 0; i < CHILD\_COUNT; ++i) {

\_total\_elements += \_children[i].\_total\_elements;

}

}

template <typename T, typename Enabler>

void QuadtreeNode<T, Enabler>::mergeWithChildren() {

\_elements = new CollisionSet; \_elements->reserve(\_capacity);

for (size\_t i = 0; i < CHILD\_COUNT; ++i) {

if (\_children[i].isSubdivide())

\_children[i].mergeWithChildren();

const auto& children\_elements = \*\_children[i].\_elements;

\_elements->insert(children\_elements.begin(), children\_elements.end());

}

delete []\_children;

\_children = nullptr;

\_total\_elements = \_elements->size();

}

template <typename T, typename Enabler>

bool QuadtreeNode<T, Enabler>::insert(T \*element, const Axes &axes) {

if (CollisionDetection::hasCollision(\_boundary, element->getPolygon(),

\_boundary.getAxes(), axes)) {

if (isSubdivide()) {

\_total\_elements = 0;

bool result = false;

for (size\_t i = 0; i < CHILD\_COUNT; ++i) {

result |= \_children[i].insert(element, axes);

\_total\_elements += \_children[i].\_total\_elements;

}

return result;

}

else {

++\_total\_elements;

\_elements->insert(element);

if (\_elements->size() >= \_capacity) {

subdivide();

}

return true;

}

}

return false;

}

template <typename T, typename Enabler>

bool QuadtreeNode<T, Enabler>::remove(T \*element, const Axes &axes) {

if (CollisionDetection::hasCollision(\_boundary, element->getPolygon(),

\_boundary.getAxes(), axes)) {

if (isSubdivide()) {

\_total\_elements = 0;

bool result = false;

for (size\_t i = 0; i < CHILD\_COUNT; ++i) {

result |= \_children[i].remove(element, axes);

\_total\_elements += \_children[i].\_total\_elements;

}

if (\_total\_elements <= \_capacity / 2) {

mergeWithChildren();

}

return result;

}

else {

if (const auto it = \_elements->find(element); it != \_elements->end()) {

\_elements->erase(it);

--\_total\_elements;

return true;

}

else

return false;

}

}

return false;

}

template <typename T, typename Enabler>

void QuadtreeNode<T, Enabler>::fillCollisionSet(const Polygon &polygon, const Axes &axes, CollisionSet &collisions\_info) const {

if (isSubdivide()) {

for (size\_t i = 0; i < CHILD\_COUNT; ++i) {

if (CollisionDetection::hasCollision(\_children[i].\_boundary, polygon,

\_children[i].\_boundary.getAxes(), axes)) {

\_children[i].fillCollisionSet(polygon, axes, collisions\_info);

}

}

}

else {

for (auto \*other\_element : \*\_elements) {

if (collisions\_info.find(other\_element) == collisions\_info.end()) {

const Polygon &other\_polygon = other\_element->getPolygon();

Axes other\_axes;

CollisionDetection::fillAxes(other\_polygon, other\_axes);

if (CollisionDetection::hasCollision(polygon, other\_polygon,

axes, other\_axes))

collisions\_info.insert(other\_element);

}

}

}

}

template <typename T, typename Enabler>

typename QuadtreeNode<T, Enabler>::Collision QuadtreeNode<T, Enabler>::getCollision(const Polygon& polygon,

const Axes& axes, CollisionResult &collision\_result) const {

if (isSubdivide()) {

for (size\_t i = 0; i < CHILD\_COUNT; ++i) {

if (CollisionDetection::hasCollision(\_children[i].\_boundary, polygon,

\_children[i].\_boundary.getAxes(), axes)) {

return \_children[i].getCollision(polygon, axes, collision\_result);

}

}

}

else {

for (auto \*other\_element : \*\_elements) {

const Polygon &other\_polygon = other\_element->getPolygon();

Axes other\_axes;

CollisionDetection::fillAxes(other\_polygon, other\_axes);

if (CollisionDetection::getCollisionResult(polygon, other\_polygon,

axes, other\_axes, collision\_result))

return other\_element;

}

}

return nullptr;

}

template <typename T, typename Enabler>

void QuadtreeNode<T, Enabler>::destroyElements() {

// Check for moved state by verifying

if (!\_elements && !\_children)

return;

if (isSubdivide())

mergeWithChildren();

for (const auto \*element : \*\_elements)

delete element;

}

template <typename T, typename Enabler>

QuadtreeNode<T, Enabler>::~QuadtreeNode() noexcept {

if (isSubdivide())

delete []\_children;

else

delete \_elements;

}

template <typename T, typename Enabler>

QuadtreeNode<T, Enabler>::QuadtreeNode(QuadtreeNode&& quadtree\_node) noexcept :

\_capacity(quadtree\_node.\_capacity),

\_total\_elements(quadtree\_node.\_total\_elements),

\_elements(quadtree\_node.\_elements),

\_children(quadtree\_node.\_children),

\_boundary(std::move(quadtree\_node.\_boundary)){

quadtree\_node.\_elements = nullptr;

quadtree\_node.\_children = nullptr;

}

Приложение Б

**Исходный код модуля GameField**

#include "../../../../include/game/construction/game field creation/GameFieldCreator.hpp"

#include "../../../../include/additionally/RandomGenerator.hpp"

#include "../../../../include/additionally/SimpleCreators.hpp"

#include "../../../../include/game/construction/game field creation/LocationPlaceholder.hpp"

#include "../../../../include/game/construction/game field creation/RoomSizeManager.hpp"

RoomType GameFieldCreator::generateType() {

const auto &item\_sequence = \_location\_info\_map.getItemSequence();

return \_room\_type\_generator.generate(item\_sequence.size() > \_min\_quantity);

}

LocationInfo\* GameFieldCreator::createLocationInfo(const sf::Vector2i& next\_pos, const RoomType room\_type) {

auto\* next\_location = new LocationInfo(next\_pos, \_room\_size\_manager.getSize(room\_type), room\_type);

\_location\_info\_map.set(next\_location, next\_location->getPosition());

return next\_location;

}

void GameFieldCreator::createTransitions(LocationInfo &location\_info, const DoorOpeningMask mask) {

LocationInfo\* connected\_locations[TOTAL\_DIRECTIONS];

size\_t count = 0;

for (const DoorOpening door\_opening : DOOR\_OPENINGS) {

if (hasDoor(mask, door\_opening)) {

const auto new\_pos(movePosition(door\_opening, location\_info.getPosition()));

if (auto \*neighbour = \_location\_info\_map.getArray2D().get(new\_pos); neighbour)

location\_info.addOutgoingDoor(neighbour, door\_opening);

else {

const auto type = generateType();

auto\* next\_location = createLocationInfo(new\_pos, type);

location\_info.addOutgoingDoor(next\_location, door\_opening);

if (type != RoomType::PORTAL)

connected\_locations[count++] = next\_location;

}

}

}

for (size\_t i = 0; i < count; ++i)

createNeighbors(connected\_locations[i]);

}

sf::Vector2i GameFieldCreator::getLatestMapIndex() const {

const size\_t even = \_lvl & ~1u;

const size\_t is\_odd = \_lvl & 1u;

return {static\_cast<int>(START\_INDEX + even), static\_cast<int>(START\_INDEX + even + is\_odd)};

}

size\_t GameFieldCreator::checkCoordinate(DoorOpeningMask &mask, const sf::Vector2i &pos,

const int coord, const int critical\_index, const DoorOpening door\_opening) const {

if (coord == critical\_index) {

mask &= ~doorToMask(door\_opening);

return 1;

}

const auto \*neighbor = \_location\_info\_map.getArray2D().get(movePosition(door\_opening, pos));

if (!neighbor)

return 0;

if (const auto type = neighbor->getRoomType(); type == RoomType::SPAWN || type == RoomType::PORTAL)

mask &= ~doorToMask(door\_opening);

return 1;

}

size\_t GameFieldCreator::getTotalObstacles(DoorOpeningMask &mask, const sf::Vector2i &pos) const {

return checkCoordinate(mask, pos, pos.x, 0, DoorOpening::LEFT)

+ checkCoordinate(mask, pos, pos.x, \_location\_info\_map.getArray2D().getLastIndex().x, DoorOpening::RIGHT)

+ checkCoordinate(mask, pos, pos.y, 0, DoorOpening::TOP)

+ checkCoordinate(mask, pos, pos.y, \_location\_info\_map.getArray2D().getLastIndex().y, DoorOpening::BOTTOM);

}

DoorOpeningMask GameFieldCreator::getDirections(const sf::Vector2i &pos) const {

DoorOpeningMask opening\_mask = ALL\_DIRECTIONS\_MASK;

const size\_t total\_obstacles = getTotalObstacles(opening\_mask, pos);

const size\_t strikeout\_count = total\_obstacles == TOTAL\_DIRECTIONS

? TOTAL\_DIRECTIONS / 2 // There is a chance to open an additional door (between the current and existing one)

: TOTAL\_DIRECTIONS - total\_obstacles + (\_room\_type\_generator.isCreatedPortal() ? 1 : -1);

for (size\_t i = strikeout\_count; i > 0; --i)

opening\_mask &= ~doorToMask(\_direction\_generator.generateDirection());

return opening\_mask;

}

void GameFieldCreator::createNeighbors(LocationInfo\* location\_info) {

const DoorOpeningMask door\_opening\_mask = getDirections(location\_info->getPosition());

if (door\_opening\_mask != 0) { createTransitions(\*location\_info, door\_opening\_mask); }

}

void GameFieldCreator::initSpawnRoomPos(sf::Vector2i& pos, sf::Vector2i& next\_pos,

DoorOpening& neighbors\_direction) const {

std::uniform\_int\_distribution<int> zero\_or\_one{0, 1};

if (RandomGenerator::getRandom(zero\_or\_one) == 0) {

if (RandomGenerator::getRandom(zero\_or\_one) == 0) {

pos.x = 0;

next\_pos.x = 1;

neighbors\_direction = DoorOpening::RIGHT;

}

else {

pos.x = \_location\_info\_map.getArray2D().getLastIndex().x;

next\_pos.x = \_location\_info\_map.getArray2D().getLastIndex().x - 1;

neighbors\_direction = DoorOpening::LEFT;

}

std::uniform\_int\_distribution<int> range(0, \_location\_info\_map.getArray2D().getLastIndex().y);

pos.y = RandomGenerator::getRandom(range);

next\_pos.y = pos.y;

}

else {

if (RandomGenerator::getRandom(zero\_or\_one) == 0) {

pos.y = 0;

next\_pos.y = 1;

neighbors\_direction = DoorOpening::BOTTOM;

}

else {

pos.y = \_location\_info\_map.getArray2D().getLastIndex().y;

next\_pos.y = \_location\_info\_map.getArray2D().getLastIndex().y - 1;

neighbors\_direction = DoorOpening::TOP;

}

std::uniform\_int\_distribution<int> range(0, \_location\_info\_map.getArray2D().getLastIndex().x);

pos.x = RandomGenerator::getRandom(range);

next\_pos.x = pos.x;

}

}

void GameFieldCreator::createRooms() {

sf::Vector2i pos;

sf::Vector2i next\_pos;

DoorOpening neighbors\_direction;

initSpawnRoomPos(pos, next\_pos, neighbors\_direction);

auto\* spawn = createLocationInfo(pos, RoomType::SPAWN);

auto\* next\_location = createLocationInfo(next\_pos, RoomType::ARENA);

spawn->addOutgoingDoor(next\_location, neighbors\_direction);

createNeighbors(next\_location);

if (!\_room\_type\_generator.isCreatedPortal()) { addExtraRooms(); }

}

void GameFieldCreator::addExtraRooms() {

auto& item\_sequence = \_location\_info\_map.getItemSequence();

do {

for (size\_t i = item\_sequence.size(); i > 0; --i) {

LocationInfo &location\_info = \*item\_sequence[i - 1];

DoorOpeningMask opening\_mask = ALL\_DIRECTIONS\_MASK;

const size\_t total\_obstacles = getTotalObstacles(opening\_mask, location\_info.getPosition());

if (total\_obstacles != TOTAL\_DIRECTIONS) {

createTransitions(location\_info, opening\_mask);

break;

}

}

} while(!\_room\_type\_generator.isCreatedPortal());

}

GameFieldCreator::GameFieldCreator(const size\_t lvl) noexcept :

\_lvl(lvl),

\_room\_size\_manager(lvl),

\_location\_info\_map(getLatestMapIndex(), nullptr),

\_min\_quantity(\_location\_info\_map.getArray2D().getTotalCount() \* 2 / 3) { createRooms(); }

[[nodiscard]] sf::Vector2f GameFieldCreator::getStartPoint(const sf::Vector2i &block\_delta) const {

const auto [p0, p1] = \_location\_info\_map.getItemSequence()[0]->getRangeRect(block\_delta,

\_room\_size\_manager.getMaxSize());

return {static\_cast<float>((p0.x + p1.x)) / 2.0f, static\_cast<float>((p0.y + p1.y)) / 2.0f};

}

GameField GameFieldCreator::initialize(const BoundaryInfo &boundary\_info) const {

return {

LocationTransformation::getMinMaxPoint(\_location\_info\_map.getItemSequence(),

\_room\_size\_manager.getMaxSize(), boundary\_info.boundary\_data.delta),

getStartPoint(boundary\_info.boundary\_data.delta)

};

}

void GameFieldCreator::create(GameField& game\_field, const BuildingInfo &building\_info, GunManager &gun\_manager,

SimpleCreators &simple\_creators, const InOutPortals& portals\_data) const {

const RoomCreator room\_creator(game\_field.quadtree\_el, building\_info, simple\_creators);

LocationMap location\_map(\_location\_info\_map.getArray2D().getLastIndex(), nullptr);

LocationTransformation::buildLocation(\_location\_info\_map.getItemSequence(), location\_map,

\_room\_size\_manager.getMaxSize(), room\_creator, game\_field.quadtree\_loc);

LocationPlaceholder::fillRooms(location\_map, gun\_manager,

{simple\_creators.element\_creator, game\_field, portals\_data},

{room\_creator.getBoundaryCreator(), {game\_field.spawn\_map}}, \_lvl);

}

GameFieldCreator::~GameFieldCreator() noexcept {

for (const auto \*el : \_location\_info\_map.getItemSequence())

delete el;

}

Приложение В

**Исходный код модуля LocationTransformation**

// ReSharper disable CppClangTidyClangDiagnosticCoveredSwitchDefault CppClangTidyClangDiagnosticImplicitFallthrough CppInconsistentNaming

#include <array>

#include <stdexcept>

#include "../../../../include/game/construction/game field creation/LocationTransformation.hpp"

MinMaxPoint LocationTransformation::getMinMaxPoint(const LocationInfos &location\_infos,

const sf::Vector2i &max\_size, const sf::Vector2i &block\_delta) {

sf::Vector2i min(INT\_MAX, INT\_MAX);

sf::Vector2i max(INT\_MIN, INT\_MIN);

for (const auto \*loc\_info : location\_infos) {

const auto [p0, p1](loc\_info->getRangeRect(block\_delta, max\_size));

if (p0.x < min.x)

min.x = p0.x;

if (p0.y < min.y)

min.y = p0.y;

if (p1.x > max.x)

max.x = p1.x;

if (p1.y > max.y)

max.y = p1.y;

}

return std::make\_pair(min, max);

}

namespace LocationTransformation::BuildLocation {

typedef LocationMap::Sequence Locations;

void createHorTransition(const RoomCreator &room\_creator, const Polygon &left\_polygon, Polygon const &right\_polygon) {

room\_creator.createHorTransition(

static\_cast<sf::Vector2i>(left\_polygon.getPoints()[1]),

static\_cast<int>(right\_polygon.getPoints()[0].x)

);

}

void createVertTransition(const RoomCreator &room\_creator, const Polygon &top\_polygon, const Polygon &bottom\_polygon) {

room\_creator.createVertTransition(

static\_cast<sf::Vector2i>(top\_polygon.getPoints()[3]),

static\_cast<int>(bottom\_polygon.getPoints()[0].y)

);

}

void createTransition(const RoomCreator &room\_creator, const DoorOpening door\_opening,

const Polygon &polygon, const Polygon &neighbor) {

switch (door\_opening) {

case DoorOpening::LEFT:

createHorTransition(room\_creator, neighbor, polygon);

break;

case DoorOpening::RIGHT:

createHorTransition(room\_creator, polygon, neighbor);

break;

case DoorOpening::TOP:

createVertTransition(room\_creator, neighbor, polygon);

break;

case DoorOpening::BOTTOM:

createVertTransition(room\_creator, polygon, neighbor);

break;

case DoorOpening::NONE:

default:

throw std::invalid\_argument("Invalid DoorOpening type");

}

}

int getDoorSizeCount(const RoomCreator &room\_creator, const DoorOpening door\_opening) {

const auto& door\_size\_count = room\_creator.getBoundaryCreator().getDoorSizeCount();

return isVert(door\_opening) ? door\_size\_count.y : door\_size\_count.x;

}

size\_t getDoorIndex(const DoorOpeningMask mask, const DoorOpening door\_opening) noexcept(false) {

size\_t result = 0;

switch (door\_opening) {

case DoorOpening::BOTTOM : result += hasTopDoor(mask);

case DoorOpening::TOP : result += hasRightDoor(mask);

case DoorOpening::RIGHT : result += hasLeftDoor(mask);

case DoorOpening::LEFT : return result;

case DoorOpening::NONE :

default : throw std::invalid\_argument("Invalid DoorOpening type");

}

}

size\_t getMissedBlockIndex(const LocationInfo \*location\_info, const DoorOpening door\_opening, const size\_t count) {

const DoorOpeningMask door\_outgoing\_mask = location\_info->getOutgoingDoorsMask();

const DoorOpeningMask door\_incoming\_mask = location\_info->getIncomingDoorsMask();

return getDoorIndex(door\_outgoing\_mask | door\_incoming\_mask, door\_opening) \* count;

}

void handleDoor(const RoomCreator &room\_creator, const LocationMap &location\_map, const LocationInfo \*location\_info) {

for (const DoorOpening door\_opening : DOOR\_OPENINGS) {

if (hasDoor(location\_info->getOutgoingDoorsMask(), door\_opening)) {

const auto \*neighbor\_location\_info = location\_info->getOutgoingDoor(door\_opening);

const size\_t door\_size\_count = getDoorSizeCount(room\_creator, door\_opening);

const size\_t missed\_index = getMissedBlockIndex(location\_info, door\_opening, door\_size\_count);

const size\_t neighbor\_missed\_index = getMissedBlockIndex(neighbor\_location\_info,

getOppositeDoor(door\_opening), door\_size\_count);

auto \*location = location\_map.getArray2D().get(location\_info->getPosition());

auto \*neighbor\_location = location\_map.getArray2D().get(neighbor\_location\_info->getPosition());

createTransition(room\_creator, door\_opening,

location->getMissedBlocks()[missed\_index]->getPolygon(),

neighbor\_location->getMissedBlocks()[neighbor\_missed\_index]->getPolygon());

}

}

}

void buildTransitions(const LocationInfos &location\_infos, const RoomCreator &room\_creator,

const LocationMap &location\_map) {

for (const auto \*loc\_info : location\_infos)

handleDoor(room\_creator, location\_map, loc\_info);

}

void buildLocations(const LocationInfos &location\_infos, const sf::Vector2i &max\_size,

const RoomCreator &room\_creator, LocationMap &location\_map) {

for (const auto \*loc\_info : location\_infos) {

const auto [p0, p1](loc\_info->getRangeRect(

room\_creator.getBoundaryCreator().getDelta(), max\_size));

const size\_t door\_opening = loc\_info->getIncomingDoorsMask() | loc\_info->getOutgoingDoorsMask();

location\_map.set(room\_creator.create(p0, p1, door\_opening, loc\_info->getRoomType()), loc\_info->getPosition());

}

}

void addToQuadtreeLocs(const Locations &locations, QuadtreeLoc &quadtree\_locs) {

for (auto \*location : locations)

quadtree\_locs.insert(location);

}

}

void LocationTransformation::buildLocation(const LocationInfos &location\_infos, LocationMap &location\_map,

const sf::Vector2i &max\_size, const RoomCreator &room\_creator, QuadtreeLoc &quadtree\_locs) {

BuildLocation::buildLocations(location\_infos, max\_size, room\_creator, location\_map);

BuildLocation::buildTransitions(location\_infos, room\_creator, location\_map);

BuildLocation::addToQuadtreeLocs(location\_map.getItemSequence(), quadtree\_locs);

}

Приложение Г

**Исходный код модуля CollisionDetection**

#include "../../../include/geometry/collision/CollisionDetection.hpp"

#include "../../../include/geometry/GeomAuxiliaryFunc.hpp"

namespace InnerLogic {

struct CollisionResultHelper final {

float smallest\_overlap;

const Axis \*collision\_axis;

bool is\_polygon1\_axis;

CollisionResultHelper(): smallest\_overlap(FLT\_MAX), collision\_axis(nullptr), is\_polygon1\_axis(false) { }

};

bool areProjectionsOverlapping(const Axes &axes, const Polygon &polygon1, const Polygon &polygon2) {

for (const auto &axis : axes) {

Projection projection1;

CollisionDetection::getProjection(polygon1, axis, projection1);

Projection projection2;

CollisionDetection::getProjection(polygon2, axis, projection2);

if (std::min(projection1.max - projection2.min, projection2.max - projection1.min) <= 0) {

return false;

}

}

return true;

}

bool isSmallestOverlapAxisFound(const Axes &axes, const bool is\_axes1,

const Polygon &polygon1, const Polygon &polygon2, CollisionResultHelper &collision\_result\_help) {

for (const auto &axis : axes) {

Projection projection1;

CollisionDetection::getProjection(polygon1, axis, projection1);

Projection projection2;

CollisionDetection::getProjection(polygon2, axis, projection2);

const float overlap = std::min(projection1.max - projection2.min, projection2.max - projection1.min);

if (overlap <= 0.0f)

return false;

if (overlap < collision\_result\_help.smallest\_overlap) {

collision\_result\_help.smallest\_overlap = overlap;

collision\_result\_help.collision\_axis = &axis;

collision\_result\_help.is\_polygon1\_axis = is\_axes1;

}

}

return true;

}

void findClosestVertex(const Polygon &polygon1, const Polygon &polygon2, const Axis &axis,

const bool is\_polygon1\_axis, sf::Vector2f &result) {

Projection projection;

ExtendedProjection extended\_projection;

if (is\_polygon1\_axis && !GeomAuxiliaryFunc::areOrthogonalOrCollinear(polygon1.getRotation(), polygon2.getRotation())) {

CollisionDetection::getProjection(polygon1, axis, projection);

CollisionDetection::getExtendedProjection(polygon2, axis, false, extended\_projection);

}

else {

CollisionDetection::getProjection(polygon2, axis, projection);

CollisionDetection::getExtendedProjection(polygon1, axis, true, extended\_projection);

}

result = extended\_projection.max - projection.min < projection.max - extended\_projection.min

? extended\_projection.max\_point

: extended\_projection.min\_point;

}

}

constexpr float EPSILON = 1.0f;

void CollisionDetection::getProjection(const Polygon &polygon, const sf::Vector2f &axis, Projection &result) {

const auto &points = polygon.getPoints();

float min = GeomAuxiliaryFunc::dotProduct(axis, points[0]);

float max = min;

for (auto &point : points) {

if (const float dot\_product\_result = GeomAuxiliaryFunc::dotProduct(axis, point); dot\_product\_result < min) {

min = dot\_product\_result;

}

else if (dot\_product\_result > max) {

max = dot\_product\_result;

}

}

result = { min, max };

}

void CollisionDetection::getExtendedProjection(const Polygon &polygon, const sf::Vector2f &axis,

const bool calculate\_mid\_point, ExtendedProjection &result) {

const auto &points = polygon.getPoints();

float min = GeomAuxiliaryFunc::dotProduct(axis, points[0]);

float max = min;

sf::Vector2f min\_point(points[0]);

sf::Vector2f max\_point(points[0]);

for (auto &point : points) {

if (const float dot\_product\_result = GeomAuxiliaryFunc::dotProduct(axis, point); dot\_product\_result < min) {

min = dot\_product\_result;

min\_point = point;

}

else if (dot\_product\_result > max) {

max = dot\_product\_result;

max\_point = point;

}

else if (calculate\_mid\_point && std::abs(dot\_product\_result - min) < EPSILON) {

min\_point = GeomAuxiliaryFunc::calcMidpoint(min\_point, point);

}

else if (calculate\_mid\_point && std::abs(dot\_product\_result - max) < EPSILON) {

max\_point = GeomAuxiliaryFunc::calcMidpoint(max\_point, point);

}

}

result = {min, max, min\_point, max\_point };

}

void CollisionDetection::fillAxes(const Polygon &polygon, Axes &axes) {

const auto &points = polygon.getPoints();

axes.reserve(points.size());

const size\_t last\_index = points.size() - 1;

for (size\_t i = 0; i < last\_index; ++i) {

axes.emplace\_back(points[i], points[i + 1]);

}

axes.emplace\_back(points[last\_index], points[0]);

}

bool CollisionDetection::getCollisionResult(const Polygon &polygon1, const Polygon &polygon2, CollisionResult &result) {

Axes axes1; fillAxes(polygon1, axes1);

Axes axes2; fillAxes(polygon2, axes2);

return getCollisionResult(polygon1, polygon2, axes1, axes2, result);

}

bool CollisionDetection::getCollisionResult(const Polygon& polygon1, const Polygon& polygon2, const Axes& axes1,

const Axes& axes2, CollisionResult& result) {

InnerLogic::CollisionResultHelper collision\_result\_help;

if (!isSmallestOverlapAxisFound(axes1, true, polygon1, polygon2, collision\_result\_help) ||

!isSmallestOverlapAxisFound(axes2, false, polygon1, polygon2, collision\_result\_help))

return false;

InnerLogic::findClosestVertex(polygon1, polygon2, \*collision\_result\_help.collision\_axis,

collision\_result\_help.is\_polygon1\_axis, result.collision\_point);

result.overlap = collision\_result\_help.smallest\_overlap;

return true;

}

bool CollisionDetection::hasCollision(const Polygon &polygon1, const Polygon &polygon2,

const Axes &axes1, const Axes &axes2) {

return InnerLogic::areProjectionsOverlapping(axes1, polygon1, polygon2) &&

InnerLogic::areProjectionsOverlapping(axes2, polygon1, polygon2);

}

Приложение Д

**Исходный код модуля BoundaryCreator**

#include "../../../include/game/construction/BoundaryCreator.hpp"

#include "../../../include/additionally/RandomGenerator.hpp"

#include "../../../include/game/construction/LocationCreator.hpp"

#include "../../../include/geometry/GeomAuxiliaryFunc.hpp"

void BoundaryCreator::createBlock(const sf::Vector2f &pos) const {

\_quadtree->insert(\_simple\_creators->element\_creator.create({

pos, 0, ElementType::BLOCK, \_building\_info.num, \_building\_info.boundary\_data.scale}));

}

void BoundaryCreator::createMissedBlock(const int x, const int y, Location\* location) const {

location->addMissedBlock(\_simple\_creators->element\_creator.create({

{static\_cast<float>(x), static\_cast<float>(y)},

0, ElementType::BLOCK, RandomGenerator::getRandom(\_missed\_blocks\_num),

\_building\_info.boundary\_data.scale}));

}

void BoundaryCreator::createHorBoundary(int start\_x, const int last\_x, const int y) const {

while (start\_x < last\_x) {

createBlock({static\_cast<float>(start\_x), static\_cast<float>(y)});

start\_x += \_building\_info.boundary\_data.delta.x;

}

}

void BoundaryCreator::createHorMissedBlocks(int start\_x, const int last\_x, const int y, Location \*location) const {

while (start\_x < last\_x) {

createMissedBlock(start\_x, y, location);

start\_x += \_building\_info.boundary\_data.delta.x;

}

}

void BoundaryCreator::createVertBoundary(int start\_y, const int last\_y, const int x) const {

while (start\_y < last\_y) {

createBlock({static\_cast<float>(x), static\_cast<float>(start\_y)});

start\_y += \_building\_info.boundary\_data.delta.y;

}

}

void BoundaryCreator::createVertMissedBlocks(int start\_y, const int last\_y, const int x, Location \*location) const {

while (start\_y < last\_y) {

createMissedBlock(x, start\_y, location);

start\_y += \_building\_info.boundary\_data.delta.y;

}

}

void BoundaryCreator::createHorBoundaryWithDoor(const sf::Vector2i &p, const int last, Location \*location, const int door\_size) const {

const int last\_offset = (p.x + last - door\_size) / 2;

createHorBoundary(p.x, last\_offset, p.y);

createHorMissedBlocks(last\_offset, last\_offset + door\_size, p.y, location);

createHorBoundary(last\_offset + door\_size, last, p.y);

}

void BoundaryCreator::createVertBoundaryWithDoor(const sf::Vector2i &p, const int last, Location \*location, const int door\_size) const {

const int last\_offset = (p.y + last - door\_size) / 2;

createVertBoundary(p.y, last\_offset, p.x);

createVertMissedBlocks(last\_offset, last\_offset + door\_size, p.x, location);

createVertBoundary(last\_offset + door\_size, last, p.x);

}

BoundaryCreator::BoundaryCreator(const BoundaryInfo &building\_info, QuadtreeEl &quadtree, SimpleCreators &simple\_creators) :

\_building\_info(building\_info), \_quadtree(&quadtree), \_simple\_creators(&simple\_creators),

\_door\_size(GeomAuxiliaryFunc::multiplyVectors(building\_info.boundary\_data.door\_size\_count,

building\_info.boundary\_data.delta)){ }

Location \*BoundaryCreator::createLocation(const sf::Vector2i &p0, const sf::Vector2i &p1,

const DoorOpeningMask door\_opening, const RoomType room\_type) const {

const int start\_x = p0.x + \_building\_info.boundary\_data.delta.x;

const int start\_y = p0.y + \_building\_info.boundary\_data.delta.y;

const int last\_x = p1.x - \_building\_info.boundary\_data.delta.x;

const int last\_y = p1.y - \_building\_info.boundary\_data.delta.y;

const auto location = \_simple\_creators->location\_creator.create({

static\_cast<float>(start\_x),

static\_cast<float>(start\_y),

static\_cast<float>(last\_x),

static\_cast<float>(last\_y)

}, room\_type);

if (hasLeftDoor(door\_opening)) createVertBoundaryWithDoor({p0.x, start\_y}, last\_y, location, \_door\_size.y);

else createVertBoundary(start\_y, last\_y, p0.x);

if (hasRightDoor(door\_opening)) createVertBoundaryWithDoor({last\_x, start\_y}, last\_y, location, \_door\_size.y);

else createVertBoundary(start\_y, last\_y, last\_x);

if (hasTopDoor(door\_opening)) createHorBoundaryWithDoor(p0, p1.x, location, \_door\_size.x);

else createHorBoundary(p0.x, p1.x, p0.y);

if (hasBottomDoor(door\_opening)) createHorBoundaryWithDoor({p0.x, last\_y}, p1.x, location, \_door\_size.x);

else createHorBoundary(p0.x, p1.x, last\_y);

return location;

}

Location \*BoundaryCreator::createLocation(const sf::Vector2i &p0, const int count\_x, const int count\_y,

const DoorOpeningMask door\_opening, const RoomType room\_type) const {

const sf::Vector2i p1(p0.x + \_building\_info.boundary\_data.delta.x \* count\_x,

p0.y + \_building\_info.boundary\_data.delta.y \* count\_y);

return createLocation(p0, p1, door\_opening, room\_type);

}

int BoundaryCreator::createVertTransition(const sf::Vector2i &p0, const int last\_y) const {

const int last\_x = p0.x + \_door\_size.x + \_building\_info.boundary\_data.delta.x;

createVertBoundary(p0.y, last\_y, p0.x);

createVertBoundary(p0.y, last\_y, last\_x);

return last\_x + \_building\_info.boundary\_data.delta.x;

}

int BoundaryCreator::createHorTransition(const sf::Vector2i &p0, const int last\_x) const {

const int last\_y = p0.y + \_door\_size.y + \_building\_info.boundary\_data.delta.y;

createHorBoundary(p0.x, last\_x, p0.y);

createHorBoundary(p0.x, last\_x, last\_y);

return last\_y + \_building\_info.boundary\_data.delta.y;

}

const sf::Vector2i &BoundaryCreator::getDelta() const { return \_building\_info.boundary\_data.delta; }

const sf::Vector2i &BoundaryCreator::getDoorSizeCount() const { return \_building\_info.boundary\_data.door\_size\_count; }

Приложение E

**Исходный код модуля LevelParameters**

#include "../../include/additionally/LevelParameters.hpp"

float getExponent(const size\_t lvl) {

if (lvl < LevelParameters::FIRST\_THRESHOLD)

return static\_cast<float>(LevelParameters::FIRST\_THRESHOLD - lvl);

if (lvl < LevelParameters::SECOND\_THRESHOLD)

return 1 / static\_cast<float>(2 + lvl - LevelParameters::FIRST\_THRESHOLD);

return 1 / static\_cast<float>(2 + LevelParameters::SECOND\_THRESHOLD - LevelParameters::FIRST\_THRESHOLD);

}

ExponentGenerator LevelParameters::createGunNumGenerator(const size\_t lvl) {

static constexpr int START\_GUN\_NUM = 1;

static constexpr int END\_GUN\_NUM = 15;

return {START\_GUN\_NUM, END\_GUN\_NUM, getExponent(lvl)};

}

ExponentGenerator LevelParameters::createEnemyCharacterNumGenerator(const size\_t lvl) {

static constexpr int START\_ENEMY\_CHARACTER\_NUM = 3;

static constexpr int END\_ENEMY\_CHARACTER\_NUM = 5;

return {START\_ENEMY\_CHARACTER\_NUM, END\_ENEMY\_CHARACTER\_NUM, getExponent(lvl)};

}

size\_t LevelParameters::getBackgroundNum(const size\_t lvl) {

static constexpr size\_t START\_BACKGROUND\_NUM = 0;

if (lvl < FIRST\_THRESHOLD)

return START\_BACKGROUND\_NUM;

if (lvl < SECOND\_THRESHOLD)

return START\_BACKGROUND\_NUM + 1;

return START\_BACKGROUND\_NUM + 2;

}

size\_t LevelParameters::getBoundaryNum(const size\_t lvl) {

static constexpr size\_t START\_BOUNDARY\_NUM = 0;

if (lvl < FIRST\_THRESHOLD)

return lvl < FIRST\_THRESHOLD / 2 ? START\_BOUNDARY\_NUM : START\_BOUNDARY\_NUM + 1;

if (lvl < SECOND\_THRESHOLD)

return START\_BOUNDARY\_NUM + 2;

return START\_BOUNDARY\_NUM + 3;

}

EnemySpawnConfiguration LevelParameters::getEnemySpawnConfiguration(const size\_t lvl) {

static constexpr size\_t BASE\_ENEMIES = 3;

static constexpr int BASE\_SPAWN\_INTERVAL = 5000;

if (lvl < FIRST\_THRESHOLD)

return {BASE\_ENEMIES + lvl, BASE\_SPAWN\_INTERVAL / (static\_cast<int>(lvl) + 1)};

if (lvl < SECOND\_THRESHOLD)

return {BASE\_ENEMIES + lvl \* 2, BASE\_SPAWN\_INTERVAL / static\_cast<int>(lvl)};

return {BASE\_ENEMIES + SECOND\_THRESHOLD \* 2, BASE\_SPAWN\_INTERVAL / SECOND\_THRESHOLD};

}

RandomSize LevelParameters::getArenaRandomSize(const size\_t lvl) {

static constexpr int BASE\_START\_SIZE\_X = 16;

static constexpr int BASE\_START\_SIZE\_Y = 12;

static constexpr int BASE\_SIZE\_OFFSET = 4;

int start\_x;

int start\_y;

int offset;

if (lvl < FIRST\_THRESHOLD) {

start\_x = BASE\_START\_SIZE\_X + static\_cast<int>(lvl);

start\_y = BASE\_START\_SIZE\_Y + static\_cast<int>(lvl);

offset = BASE\_SIZE\_OFFSET;

}

else if (lvl < SECOND\_THRESHOLD) {

start\_x = BASE\_START\_SIZE\_X + static\_cast<int>(lvl) \* 2;

start\_y = BASE\_START\_SIZE\_Y + static\_cast<int>(lvl) \* 2;

offset = BASE\_SIZE\_OFFSET \* 2;

}

else {

start\_x = BASE\_START\_SIZE\_X + SECOND\_THRESHOLD \* 2;

start\_y = BASE\_START\_SIZE\_Y + SECOND\_THRESHOLD \* 2;

offset = BASE\_SIZE\_OFFSET \* 3;

}

return {std::uniform\_int\_distribution<int>{start\_x, start\_x + offset},

std::uniform\_int\_distribution<int>{start\_y, start\_y + offset}};

}

std::uniform\_real\_distribution<float> LevelParameters::getBulletSpread(const size\_t lvl) {

static constexpr float BASE\_SPREAD = 0.6f;

float spread;

if (lvl < FIRST\_THRESHOLD)

spread = BASE\_SPREAD - static\_cast<float>(lvl) / 35.0f;

else if (lvl < SECOND\_THRESHOLD)

spread = BASE\_SPREAD / 1.5f - static\_cast<float>(lvl) / 34.0f;

else

spread = BASE\_SPREAD / 4;

return std::uniform\_real\_distribution<float>{-spread, spread};

}

Приложение E

**Исходный код модуля ArenaCreator**

// ReSharper disable CppIncompleteSwitchStatement CppDefaultCaseNotHandledInSwitchStatement

#include "../../../../../include/game/construction/game field creation/arena/ArenaCreator.hpp"

ArenaCreator::ArenaCreator(const BoundaryCreator& boundary\_creator, SpawnMapCreator &&spawn\_map\_creator):

\_boundary\_creator(&boundary\_creator), \_spawn\_map\_creator(spawn\_map\_creator) { }

void ArenaCreator::createSpawns(Field& field, const int spawn\_count) const {

const auto& last\_index = field.getLastIndex();

const int attempt\_count = std::max(last\_index.x, last\_index.y);

std::uniform\_int\_distribution<int> random\_x(0, last\_index.x);

std::uniform\_int\_distribution<int> random\_y(0, last\_index.y);

for (int i = 0; i < spawn\_count; ++i) {

sf::Vector2i pos;

int attempt = 0;

do {

pos = { RandomGenerator::getRandom(random\_x), RandomGenerator::getRandom(random\_y) };

++attempt;

} while (field.get(pos) != CellType::EMPTY && attempt <= attempt\_count);

if (field.get(pos) == CellType::EMPTY)

field.set(CellType::SPAWN, pos);

}

}

void ArenaCreator::verifyEndCoord(sf::Vector2i& end\_coord, const sf::Vector2i& last\_index) {

end\_coord.x = std::max(0, std::min(end\_coord.x, last\_index.x));

end\_coord.y = std::max(0, std::min(end\_coord.y, last\_index.y));

}

void ArenaCreator::createWalls(Field& field, const int wall\_count) const {

const auto& last\_index = field.getLastIndex();

std::uniform\_int\_distribution<int> random\_x(0, last\_index.x);

std::uniform\_int\_distribution<int> random\_y(0, last\_index.y);

std::uniform\_int\_distribution<int> random\_width(-last\_index.x / 3, last\_index.x / 3);

std::uniform\_int\_distribution<int> random\_height(-last\_index.y / 3, last\_index.y / 3);

for (int i = 0; i < wall\_count;) {

const sf::Vector2i start\_coord { RandomGenerator::getRandom(random\_x), RandomGenerator::getRandom(random\_y) };

sf::Vector2i end\_coord { start\_coord.x + RandomGenerator::getRandom(random\_width),

start\_coord.y + RandomGenerator::getRandom(random\_height) };

verifyEndCoord(end\_coord, last\_index);

createWallSegment(field, start\_coord, end\_coord);

i += (std::abs(end\_coord.x - start\_coord.x) + 1) \* (std::abs(end\_coord.y - start\_coord.y) + 1);

}

}

void ArenaCreator::createWallSegment(Field &field, const sf::Vector2i& start, const sf::Vector2i& end) const {

const auto [min\_x, max\_x] = std::minmax(start.x, end.x);

const auto [min\_y, max\_y] = std::minmax(start.y, end.y);

for (int x = min\_x; x <= max\_x; ++x)

for (int y = min\_y; y <= max\_y; ++y)

field.set(CellType::WALL, {x, y});

}

sf::Vector2i ArenaCreator::getLastIndex(const Location& location) const {

const auto& points = location.getPolygon().getPoints();

const auto& size = \_boundary\_creator->getDelta();

const int count\_x = static\_cast<int>((points[1].x - points[0].x)) / size.x - 2 \* INDENT;

const int count\_y = static\_cast<int>((points[3].y - points[0].y)) / size.y - 2 \* INDENT;

return {count\_x, count\_y};

}

sf::Vector2f ArenaCreator::transferPos(const sf::Vector2f &start\_pos, const sf::Vector2i &size, const sf::Vector2i &pos) {

return {

start\_pos.x + static\_cast<float>(size.x \* (pos.x + INDENT)),

start\_pos.y + static\_cast<float>(size.y \* (pos.y + INDENT))

};

}

void ArenaCreator::createArena(const Field &field, SpawnPoints& spawn\_points,

const sf::Vector2f &start\_pos, const sf::Vector2i &size) const {

const sf::Vector2i& last\_index = field.getLastIndex();

for (sf::Vector2i index = {0, 0}; index.x <= last\_index.x; ++index.x) {

for (; index.y <= last\_index.y; ++index.y) {

switch (field.get(index)) {

case CellType::WALL:

\_boundary\_creator->createBlock(transferPos(start\_pos, size, index));

break;

case CellType::SPAWN:

spawn\_points.push\_back(transferPos(start\_pos, size, index));

break;

}

}

index.y = 0;

}

}

void ArenaCreator::create(const Location& location) const {

Field field(getLastIndex(location), CellType::EMPTY);

const int square = static\_cast<int>(field.getTotalCount());

const int wall\_count = square / RandomGenerator::getRandom(\_random\_wall\_count\_denominator);

const int spawn\_count = square / RandomGenerator::getRandom(\_random\_spawn\_count\_denominator);

createWalls(field, wall\_count);

createSpawns(field, spawn\_count);

createArena(field, \_spawn\_map\_creator.installLocation(location),

location.getPolygon().getPoints()[0], \_boundary\_creator->getDelta());

}

Приложение Ё

**Исходный код модуля CollisionManager**

#include <algorithm>

#include "../../../include/geometry/collision/CollisionManager.hpp"

#include "../../../include/game/identifiable/ElementIdTracker.hpp"

#include "../../../include/geometry/collision/CollisionResolution.hpp"

CollisionManager::CollisionManager(CollisionTable type\_collision): \_type\_collision(std::move(type\_collision)) { }

bool CollisionManager::processCollisionSet(const Element &element, const CollisionTable::AvailableCollisions& available\_collisions,

const ElementCollisionSet &element\_collision\_set) {

bool result = false;

CollisionResult collision\_result;

const auto& polygon = element.getPolygon();

for (auto \*collision : element\_collision\_set) {

const ElementType collision\_type = ElementIdTracker::extractType(collision->getId());

if (available\_collisions.find(collision\_type) != available\_collisions.end() &&

CollisionDetection::getCollisionResult(polygon, collision->getPolygon(), collision\_result)) {

result = true;

CollisionResolution::separateElement(element, collision\_result);

}

}

return result;

}

void CollisionManager::filterCollisions(const CollisionTable::AvailableCollisions& available\_collisions, ElementCollisionSet &result) {

for (auto it = result.begin(); it != result.end();) {

const ElementType collision\_type = ElementIdTracker::extractType((\*it)->getId());

if (available\_collisions.find(collision\_type) == available\_collisions.end())

it = result.erase(it);

else

++it;

}

}

void CollisionManager::fillCollisionSet(const Element &element, const QuadtreeEl &quadtree, ElementCollisionSet &result) const {

if (const auto\* available\_collisions = \_type\_collision[ElementIdTracker::extractType(element.getId())]) {

quadtree.fillCollisionSet(element.getPolygon(), result);

filterCollisions(\*available\_collisions, result);

}

}

bool CollisionManager::hasCollisions(const CollisionTable::AvailableCollisions& available\_collisions,

const ElementCollisionSet& element\_collision\_set) {

return std::any\_of(element\_collision\_set.begin(), element\_collision\_set.end(), [&](const auto \*collision) {

return available\_collisions.find(ElementIdTracker::extractType(collision->getId())) != available\_collisions.end();

});

}

bool CollisionManager::hasCollisions(const Element& element, const QuadtreeEl& quadtree) const {

if (const auto\* available\_collisions = \_type\_collision[ElementIdTracker::extractType(element.getId())]) {

ElementCollisionSet element\_collision\_set;

quadtree.fillCollisionSet(element.getPolygon(), element\_collision\_set);

return hasCollisions(\*available\_collisions, element\_collision\_set);

}

return false;

}

bool CollisionManager::processCollisions(const Element &element, const ElementCollisionSet &element\_collision\_set) const {

if (const auto\* available\_collisions = \_type\_collision[ElementIdTracker::extractType(element.getId())])

return processCollisionSet(element, \*available\_collisions, element\_collision\_set);

return false;

}

bool CollisionManager::processCollisions(const Element& element, const QuadtreeEl &quadtree) const {

if (const auto\* available\_collisions = \_type\_collision[ElementIdTracker::extractType(element.getId())]) {

ElementCollisionSet element\_collision\_set;

quadtree.fillCollisionSet(element.getPolygon(), element\_collision\_set);

return processCollisionSet(element, \*available\_collisions, element\_collision\_set);

}

return false;

}

bool CollisionManager::isVisible(const Line& line, const FightingEntity& watcher, const FightingEntity& target,

const QuadtreeEl &quadtree) const {

ElementCollisionSet element\_collision\_set;

quadtree.fillCollisionSet(line, element\_collision\_set);

element\_collision\_set.erase(&watcher.getCharacter().getElement());

element\_collision\_set.erase(&target.getCharacter().getElement());

return !hasCollisions(\_type\_collision.getRayCollisions(), element\_collision\_set);

}

ВЕДОМОСТЬ ДОКУМЕНТОВ

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Обозначение | | | | | Наименование | | | | Дополнительные сведения | |
|  | | | | | Текстовые документы | | | |  | |
|  | | | | |  | | | |  | |
| БГУИР КП 1–40 01 01 020 ПЗ | | | | | Пояснительная записка | | | | 72 с. | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | | Графические документы | | | |  | |
|  | | | | |  | | | |  | |
| ГУИР.251003-20 СА | | | | | Метод buildLocation. Схема алгоритма | | | | Формат А1 | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  |  |  |  |  | БГУИР КП I- 40 01 01 020 ПЗ | | | | | |
|  |  |  |  |  |
|  |  |  |  |  |
| Изм | Лист | № докум. | Подп. | Дата | Компьютерная игра «Соул Найт».  Ведомость курсового  проекта | Литера | | | Лист | Листов |
| Разраб. | | Панкратьев Е.С. |  |  | Т |  |  | 73 | 73 |
| Провер. | | Шостак Е.В. |  |  | Кафедра ПОИТ  гр. 251003 | | | | |