МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«МИРЭА – Российский технологический университет»
РТУ МИРЭА

Институт искусственного интеллекта
Кафедра программного обеспечения систем радиоэлектронной аппаратуры

КУРСОВАЯ РАБОТА

по дисциплине «Методы и стандарты программирования»

на тему: «Создание компьютерной игры Fantom»

Обучающийся _____ Димитриев Егор Александрович
Подпись *Фамилия Имя Отчество*

Шифр 23К0021

Группа КМБО-02-23

Руководитель работы _____ Черноусов Игорь Дмитриевич
Подпись *Фамилия Имя Отчество*

Москва 2024

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Данный документ представляет собой пояснительную записку к курсовой работе по курсу «Методы и стандарты программирования». Темой курсовой работы является разработка игры "Fantom: Dark Entity", представляющей собой 2D-игру-платформер, в которой игроки исследуют уникальные комнаты, полные опасностей. Игрокам предстоит собирать кристаллы хаоса, избегая охранников в каждой комнате, что непременно добавит элемент стратегии и напряжения в игровой процесс.

1 ТЕХНИЧЕСКОЕ ЗАДАНИЕ

1.1 Требования к функциональным характеристикам программы

Игра "Fantom" должна включать следующие функциональные элементы:

1. Игровой процесс: игрок может исследовать комнаты, пытаясь найти выход.
2. Уникальные комнаты: Игра состоит из 9 уникальных комнат, каждая из которых имеет свои особенности, комнаты ловушки из которых можно только отходить назад.
3. Охрана: Каждая комната охраняется стражами, которые обладают механикой стрельбы по игроку.
4. Сбор кристаллов: Игроки могут собирать кристаллы хаоса, которые дают дополнительную энергию для полета.
5. Интерфейс пользователя: Полоска энергии в левом верхнем углы окна, которая демонстрирует кол-во энергии позволяющее летать и общий таймер игры.

1.2 Критерии работоспособности

Список критериев работоспособности программы:

1. Все 9 комнат реализованы с охраной и звуковым сопровождением.
2. Реализованы комнаты ловушки, из которых нужно отходить назад.
3. Игрок может собирать кристаллы хаоса, которые увеличивают энергию для полета.
4. Игрок может убивать охрану взрывом если она находиться в радиусе достигаемости.
5. Реализовано стартовое меню, начала игры, и меню победы в игре.
6. При попадании пули охраны игра завершается с выводом меню.

2 ОТЧЕТ О РАЗРАБОТКЕ ПРОГРАММЫ

2.1 Архитектура программы

Для представления общей архитектуры программы выделены ключевые методы для представления взаимодействий классов друг с другом.

Далее термин "тайл" в контексте программы, обозначает квадратное изображение, которое используется в качестве строительного блока для создания игровой карты.

Класс Game является центральным элементом архитектуры игры и отвечает за управление основным игровым процессом. Он инициализирует все необходимые компоненты, обрабатывает пользовательский ввод, обновляет состояние игры и отвечает за отрисовку графики на экране.

Основные Члены Класса:

- Game::Game() - инициализирует окно игры, загружает текстуры для персонажа и карт, а также настраивает элементы интерфейса, такие как полоска энергии и таймер.
- void Game::run() - основной цикл игры, который выполняется до тех пор, пока окно открыто.
- void Game::draw() - отвечает за от рисовку всех элементов на экране.
- void Game::processEvents() - проверяет нажатия клавиш для управления движением игрока (вверх, вниз, влево, вправо).Обрабатывает нажатие пробела для активации взрыва.
- void Game::update() - обновляет состояние игры, включая движение игрока и взаимодействие с окружением.
- void Game::triggerExplosion(sf::Vector2f position) - вызывает взрыв в заданной позиции в зависимости от направления движения игрока.

Класс Map отвечает за представление и управление картами в игре. Он загружает текстуры, создает и обновляет состояние карты, а также задает врагов на карте и взаимодействие игрока с окружением. Класс обеспечивает от рисовку всех элементов карты и обработку логики выхода из уровней.

Основные Члены Класса:

- Map::Map(const std::unordered_map<int, std::string>& textureFiles, const std::vector<std::vector<int>>& mapData, int currentMap) - конструктор инициализирует карту, загружая текстуры из переданного словаря и устанавливает противников для текущей карты.
- void Map::draw(sf::RenderWindow& window) – обрисовывает общий вид карты. Проходит по двумерному вектору map, заполняет тайлы в зависимости от их типа.
- bool Map::isExitTile(const sf::Vector2f& position) const - проходит по всем тайлам карты и проверяет, содержит ли позиция игрока тайл выхода.
- Метод bool Map::isExitTileEnd(const sf::Vector2f& position) const - проверяет, находится ли игрок на тайле выхода в тупик для завершения уровня.
- void Map::updateEnemies(float deltaTime, Entity &player, const std::vector<Explosion>& explosions) - вызывает метод обновления для каждого врага, передавая время с последнего кадра и информацию о состоянии игрока.
- void Map::update(float deltaTime) - проверяет состояние дверей на карте и обновляет их анимацию в зависимости от времени.

Класс Entity представляет собой класс игрока. Он отвечает за управление состоянием игрока, и его анимацией, и взаимодействием с окружающей средой.

Основные Члены Класса:

- Entity::Entity(const std::vector<std::string>& textureFiles, const std::vector<std::string>& dieTexturesFile, float posX, float posY, float width, float

height)- конструктор инициализирует объект игрок, загружая текстуры для анимации и устанавливая начальные параметры.

- void Entity::move(float x, float y, Map &map) - сохраняет текущее положение спрайта. Перемещает спрайт на заданные значения по осям X и Y. Проверяет столкновения с картой через метод checkCollision(). Если столкновение обнаружено, возвращает спрайт на старую позицию.
- bool Entity::checkCollision(Map& map) - получает границы сущности и проходит по всем тайлам карты. Если обнаруживается пересечение с тайлом определенного типа (например, препятствия или кристаллы), выполняются соответствующие действия (например, сбор кристаллов).
- void Entity::update(float deltaTime) - если сущность мертва, обновляет анимацию смерти; если жива — обновляет анимацию обычного состояния.

Класс Enemy представляет собой охрану. Класс реализует логику поведения врагов, включая анимацию атаки и анимацию смерти.

Основные Члены Класса

- Enemy::Enemy(const std::string& textureFile, const std::string& attackTextureFile, const std::string& deathTextureFile, float posX, float posY, float speedEnemy, float rechargeTime, int yShootEnemy) - конструктор инициализирует объект врага, загружая текстуры для анимации движения, атаки и смерти.
- void Enemy::animate(float deltaTime); void Enemy::attackAnimate(float deltaTime); void Enemy::deathAnimation(float deltaTime) – увеличивают время анимации и переключает текущий кадр в зависимости от заданной скорости анимации.
- void Enemy::move(float deltaX, float deltaY, const Map& map) - рассчитывает новые координаты и проверяет столкновения с тайлами карты. Если столкновение обнаружено, меняет направление движения.

- void Enemy::update(float deltaTime, const Map& map, Entity &player, const std::vector<Explosion>& explosions) - проверяет столкновения с взрывами. Если игрок виден, запускает атаку; если нет — продолжает движение по карте.
- bool Enemy::checkCollisionWithPlayer(const Bullet& bullet, const Entity& player) const - возвращает true, если пуля пересекает границы игрока.
- void Enemy::draw(sf::RenderWindow& window) - вызывает метод от рисовки для спрайта и всех его пуль.
- void Enemy::shoot(const Entity& player) - добавляет пулю в вектор пуль с учетом текущей позиции врага.
- bool Enemy::canSeePlayer(const Entity& player, const Map& map) - проверяет расстояние до игрока и наличие препятствий между врагом и игроком.

Класс Explosion отвечает за визуализацию и анимацию взрывов. Он управляет состоянием взрыва, включая его масштаб, время анимации и от рисовки на экране.

Класс Bullet представляет пули, которыми стреляют враги. Он управляет созданием, движением и от рисовкой пуль, а также их взаимодействием с другими объектами, такими как игрок и стены.

Так же в программе присутствуют два утилитарных класса:

1. Класс Menu отвечает за управление пользовательским интерфейсом игры. Он предоставляет игроку возможность взаимодействовать с меню, включая начало новой игры, доступ к настройкам и отображение результатов после завершения уровня. Класс реализует логику для отображения элементов интерфейса и обработки событий ввода.
2. Класс ResourceLoader объединяет все функции, связанные с загрузкой ресурсов. Методы класса, такие как loadTexturesFromDirectory, loadTextures и loadMapsFromFile, позволяют разработчику быстро загружать

ресурсы из файловой системы или текстовых файлов. Что упрощает процесс интеграции новых ресурсов в игру и минимизирует вероятность ошибок.

2.2 Алгоритмическая часть

В игре реализован алгоритм видимости охраны. Сначала вычисляется направление к игроку и его длина. В цикле проверяется наличие непрозрачных тайлов (стен) между врагом и игроком, если они есть значит игрок скрыт от охраны, и она продолжает идти до того, как не встретит не проходимый тайлов, и не повернет обратно.

3 РУКОВОДСТВО ПО СБОРКЕ И ЗАПУСКУ

3.1 Требования к составу и параметрам технических средств

Чтобы установить и запустить игру нужные следующие средства:

1. Компилятор: Необходим компилятор C++, совместимый с SFML. Рекомендуется использовать:

- Windows: MinGW или Visual Studio (2017 и выше);
- macOS: Xcode или g++ через Homebrew;
- Linux: g++ или clang.

2. Библиотека SFML: убедитесь, что у вас установлена последняя версия SFML (рекомендуется версия 2.5.1 или выше). Библиотеку можно скачать с официального сайта SFML.

3. Система контроля версий Git: убедитесь, что у вас установлен Git для работы с репозиториями. Git необходим для клонирования репозитория проекта. Вы можете скачать его с официального сайта.

3.2 Сборка и запуск проекта

Для корректного запуска проекта выполните следующие шаги:

1. Перейдите в директорию проекта: “cd fantom/src”
2. Настройка пути к SFML в Makefile

В вашем Makefile необходимо указать пути к заголовочным файлам и библиотека SFML. Пример секции Makefile:

SFML_DIR = /path/your/SFML

В /path/your/SFML ваш фактический путь к установленной библиотеке SFML

3. После настройки Makefile выполните команду для сборки проекта: “make”
4. Запустить игру выполнив команду: “./prog”

4 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

4.1 Игровое управление

Управление персонажем осуществляется через клавиши WASD. Вызывание взрывов осуществляется при нажатии TAB клавиатуре.

4.2 Графический интерфейс

Стартовое меню пользователя представляет из себя:

- кнопка "Начать игру";
- кнопка "Настройки" (см. Рисунок 1).

Рисунок 1 – Стартовое меню

На протяжении всей игры пользователь может видеть окно энергии и таймер, отображающий начало сессии (см. Рисунок 2).

Рисунок 2 – Полоса энергии и таймер

Чтобы передвигаться между комнатами пользователь, должен оказаться рядом с терминалом (см. Рисунок 3), что откроет люки для прохода дальше (см. Рисунок 4).

Рисунок 3 – Терминал

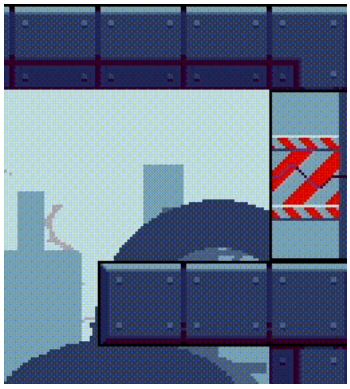

Рисунок 4 – Открытие двери

Игровая карта игрока представлена набором комнат, разделенных на четыре уровня. 1 уровень это первые 4 комнаты, которые пройдет игрок, в них содержится один охранник (см Рисунок 5).

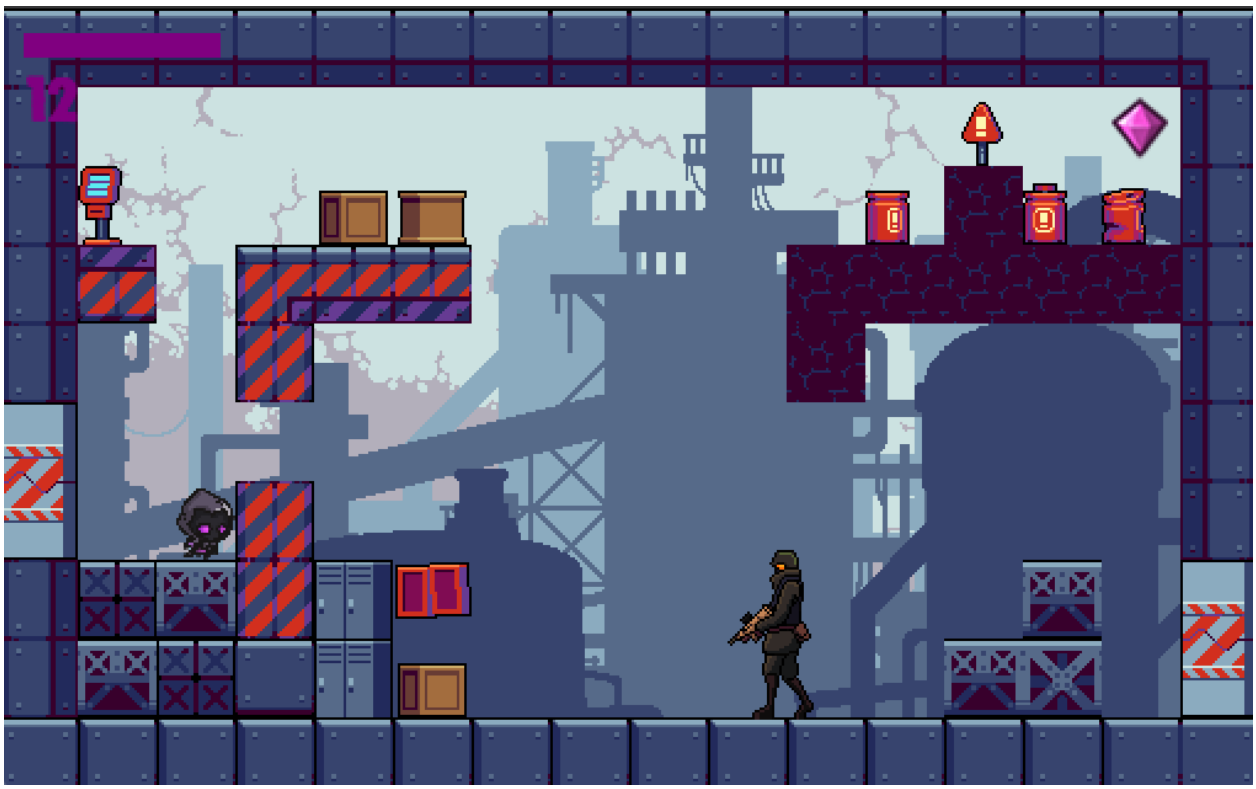

Рисунок 5 – 3 комната (1 уровень)

2 уровень комнаты в которых уже два стража (см Рисунок 6).

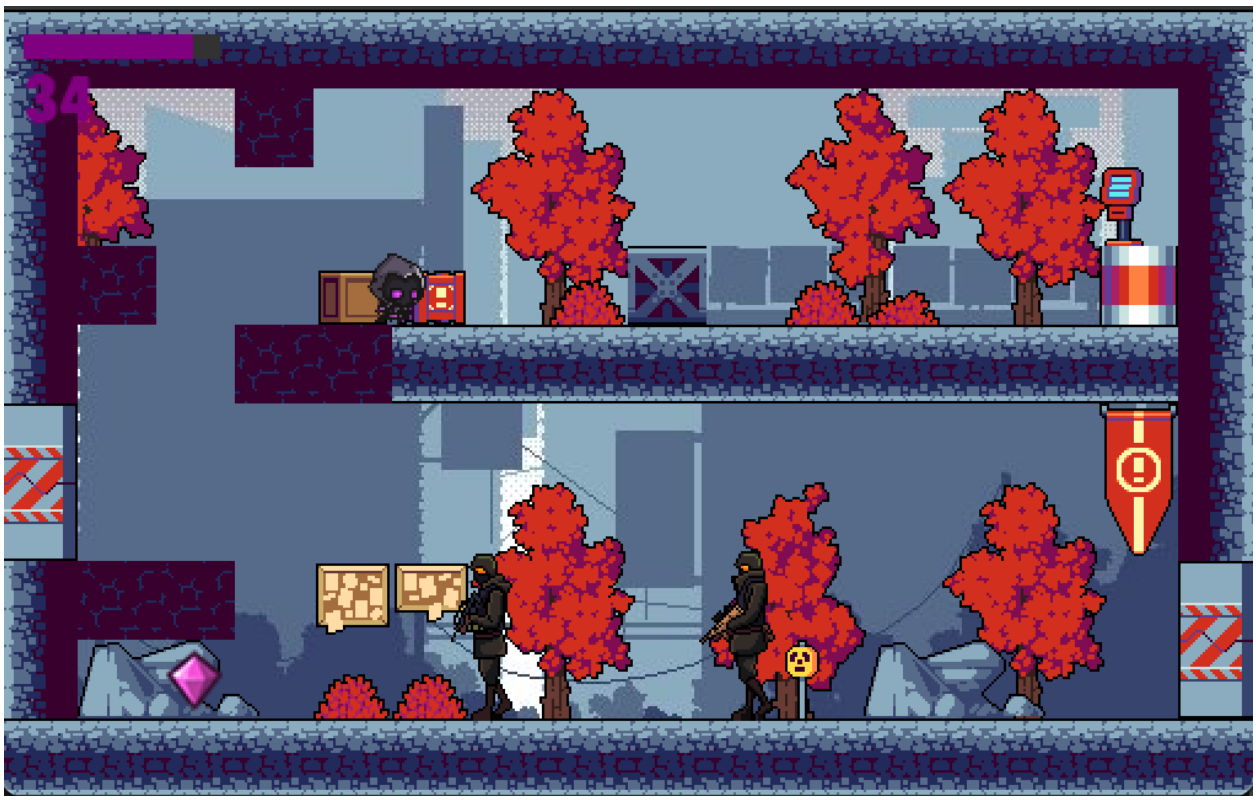

Рисунок 6 – 6 комната (2 уровень)

Следующий тип, 7 комната, которая дает выбор куда пойти (см. Рисунок 7) она может привести как к обычной комнате, так и к комнате ловушке (см. Рисунок 8), из которой можно вернуться только обратно.

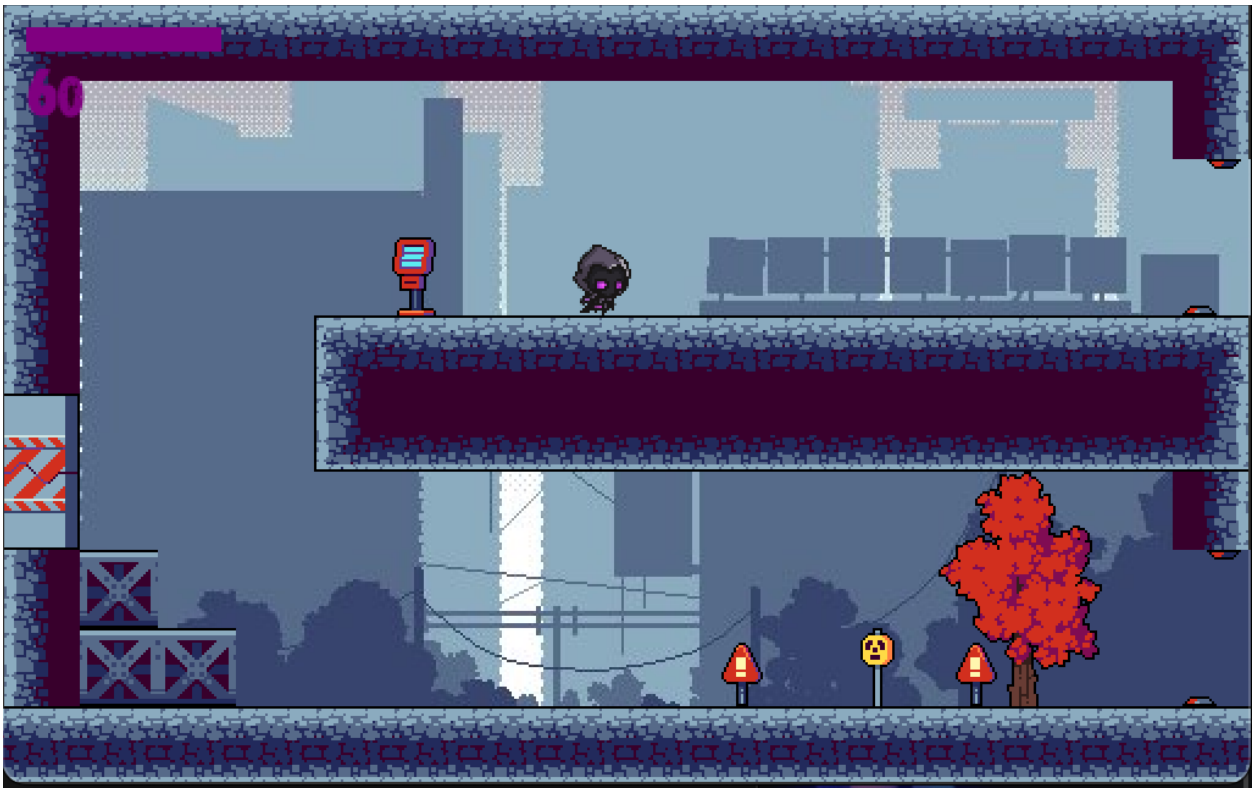

Рисунок 7 – 7 комната (выбор пути)

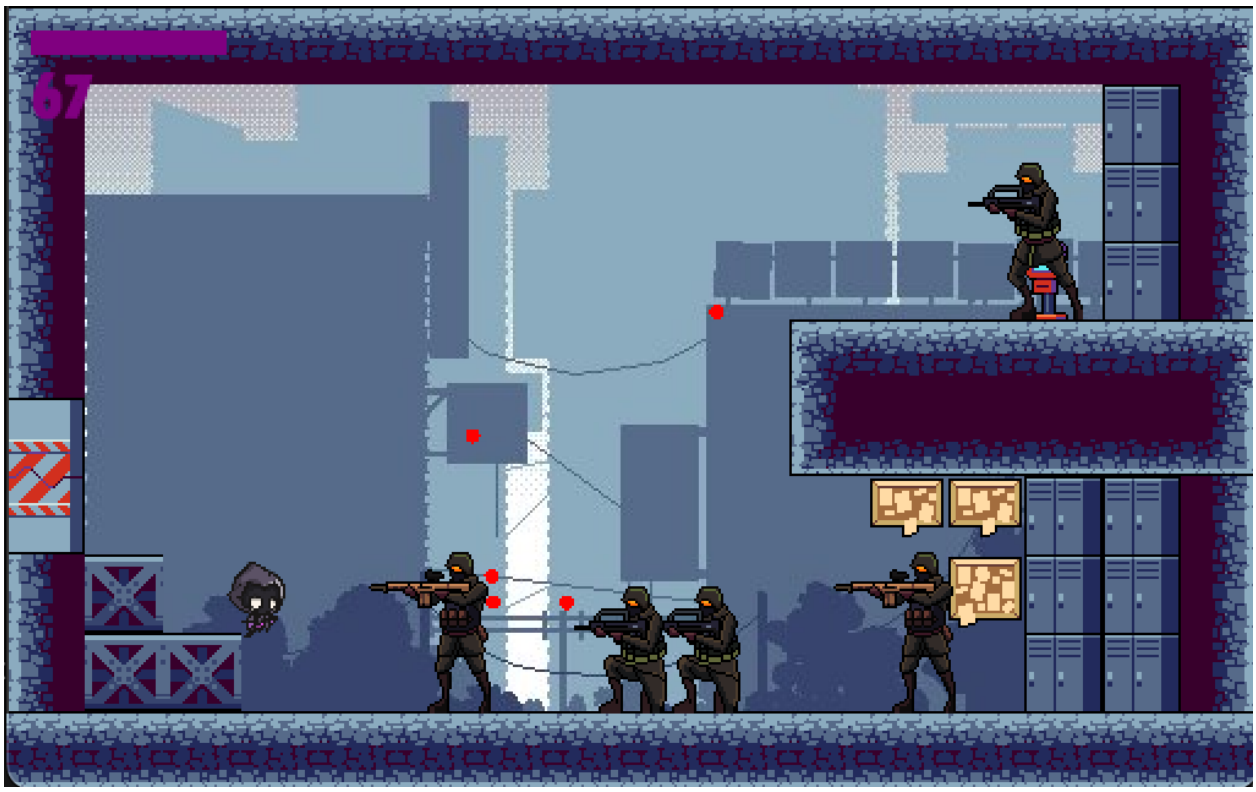

Рисунок 8 – 8 комната (ловушка)

При проигрыше игроку предлагается выбрать дальнейшие действия: “Выход” или “Новая игра” (см. Рисунок 9).

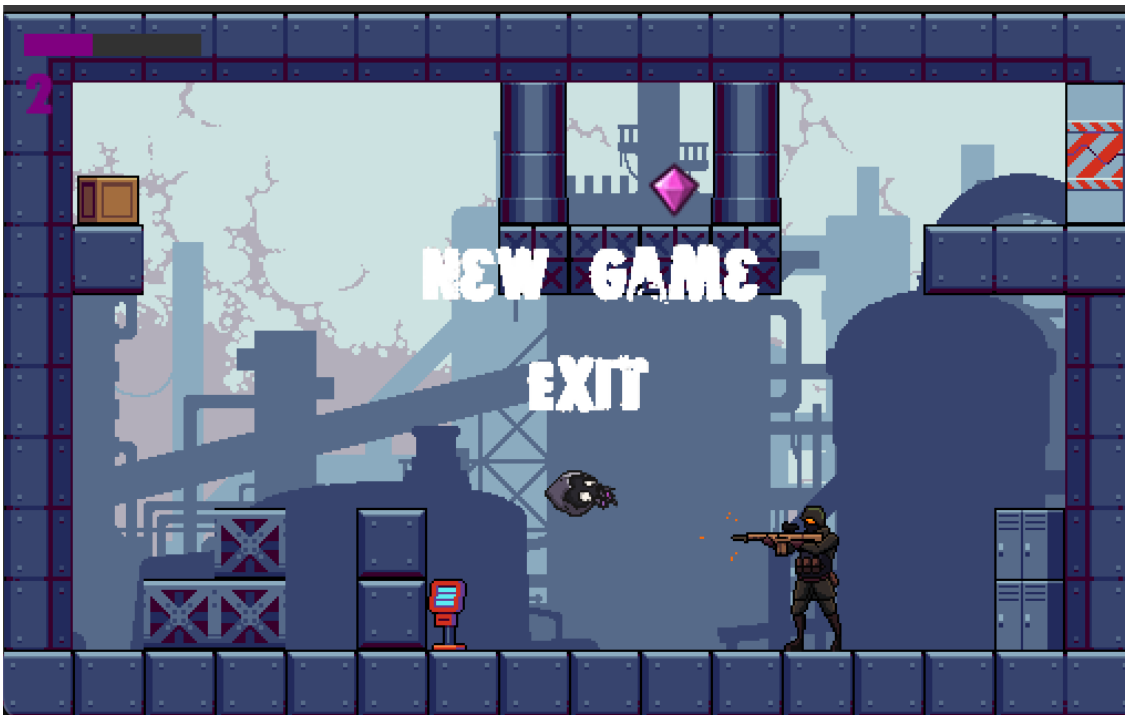

Рисунок 9 – Меню проигрыша

При победе перед игроком выходит окно статистики игры и возможность начать новую (см. Рисунок 10).

Рисунок 10 – Меню победы игрока

ЗАКЛЮЧЕНИЕ

В ходе выполнения курсовой работы были достигнуты поставленные цели по разработке игры "Fantom: Dark Entity". Проект позволил углубить знания в области программирования на C++ и познакомиться с библиотекой SFML.

Изучение технологии SFML стало ключевым элементом в разработке, так как эта библиотека предоставляет мощные инструменты для работы с графикой, звуком и пользовательским вводом. В процессе работы над проектом я освоил основные классы и функции SFML, такие как создание окон, обработка событий, работа с текстурами и спрайтами. Разработка игры также включала в себя реализацию игрового меню и механики взаимодействия игрока с окружением.

Таким образом, работа над проектом не только позволила применить теоретические знания на практике, но и получить новые.

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1. SFML Documentation. - URL: https://www.sfml-dev.org/documentation (дата обращения 07.12.2024).
2. Git Documentation. - URL: https://git-scm.com/doc (дата обращения 07.12.2024).
3. C++ Documentation. - URL: https://devdocs.io/cpp (дата обращения 20.12.24).