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, полных опасностей. Игроку предстоит собирать кристаллы хаоса, избегая охранников в каждой комнате, что добавляет элемент стратегии и напряжения в игровой процесс.

1 Техническое Задание

1.1 Требования к функциональным характеристикам программы

Игра "Fantom" должна включать следующие функциональные элементы:

Игровой процесс: игрок может исследовать комнаты, пытаясь найти выход.

Уникальные комнаты: Игра состоит из 9 уникальных комнат, каждая из которых имеет свои особенности, комнаты ловушки из которых можно только отходить назад.

Охрана: Каждая комната охраняется стражами, которые обладают механикой стрельбы по игроку.

Сбор кристаллов: Игроки могут собирать кристаллы хаоса, которые дают дополнительную энергию для полета.

Интерфейс пользователя: Полоска энергии в левом верхнем углы окна, которая демонстрирует кол-во энергии позволяющее летать и общий таймер игры.

1.2 Список критериев, по которым определяется работоспособность программы

Все 9 комнат реализованы с охраной.

Реализованы комнаты ловушки, из которых нужно отходить назад.

Игрок может собирать кристаллы хаоса, которые увеличивают энергию для полета.

Игрок может убивать охрану взрывом если она находиться в радиусе достигаемости кнопкой Tab.

В игре присутствует звуковое сопровождение.

Реализовано стартовое меню, начала игры, и меню победы в игре.

При попадании пули охраны игра завершается с выводом меню.

2 Отчёт о разработке программы

2.1 Архитектура программы

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 от рисовку 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 управляет врагами и взаимодействием 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 устанавливая данные для текущей карты.
- void Map::draw(sf::RenderWindow& window) – Обрисовывает фоновую текстуру. 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 сущности, 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.
- void Entity::updateSprite(bool turn) - Если параметр turn равен true, переворачивает спрайт по оси X.
- 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::draw(sf::RenderWindow& window) - Отрисовывает спрайт сущности на переданном окне.
- void Entity::update(float deltaTime) - Если сущность мертва, обновляет анимацию смерти; если жива — обновляет анимацию обычного состояния, если видит противиника обновляет анимацию акаки.
- bool Entity::isOnGround(Map &map): Проверяет, находится ли сущность на земле.

Класс Enemy представляет собой охрану. Класс реализует логику поведения врагов, включая 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 снаряды, которые стреляют враги. Он управляет созданием, движением и от рисовки пуль, а также их взаимодействием с другими объектами, такими как игрок и стены.

Так же в программе присутствуют два утилитарных класса:

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.

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

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.

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

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

1. Перейдите в директорию проекта: “cd fantom”
2. Настройка пути к SFML в Makefile
   В вашем Makefile необходимо указать пути к заголовочным файлам и библиотекам 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)

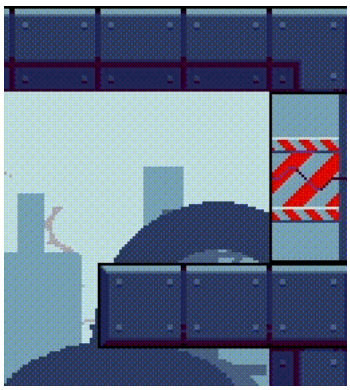
Рисунок 1 – Стартовое меню

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

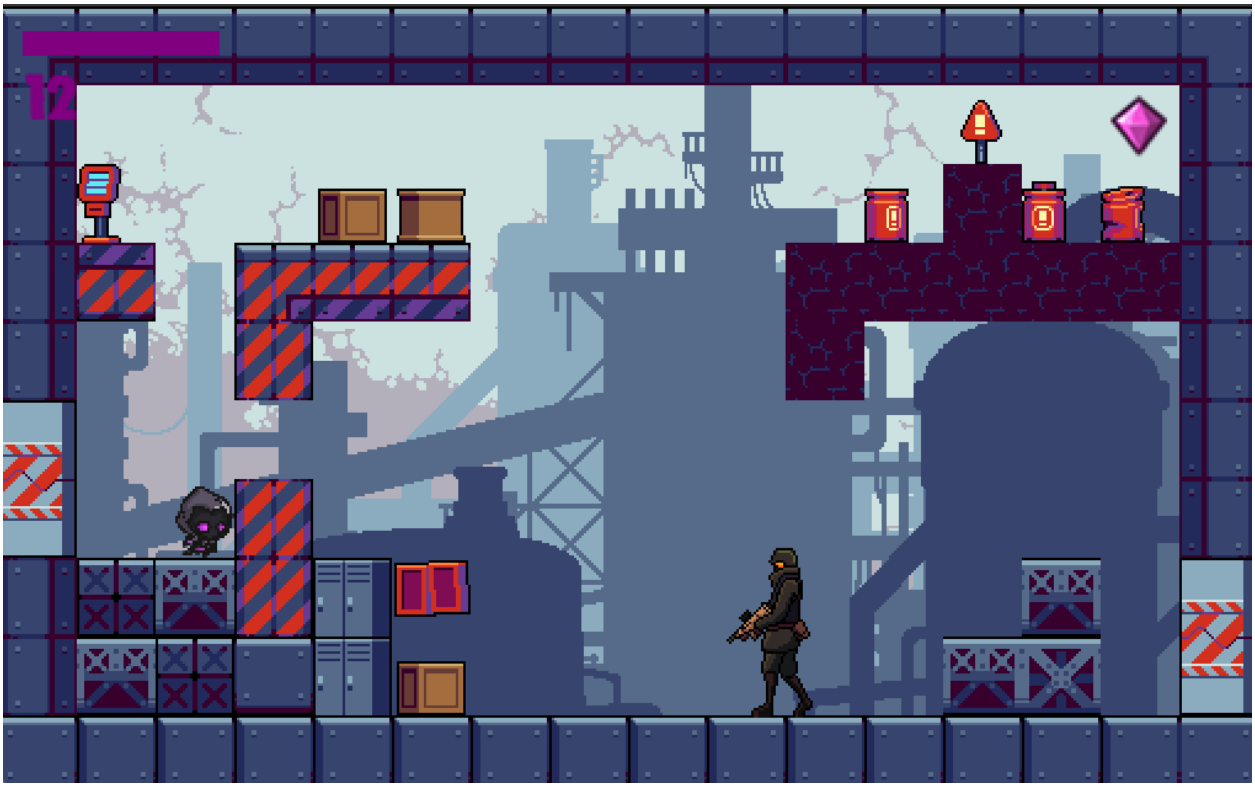


Рисунок 5 – 3 комната (1 уровень)

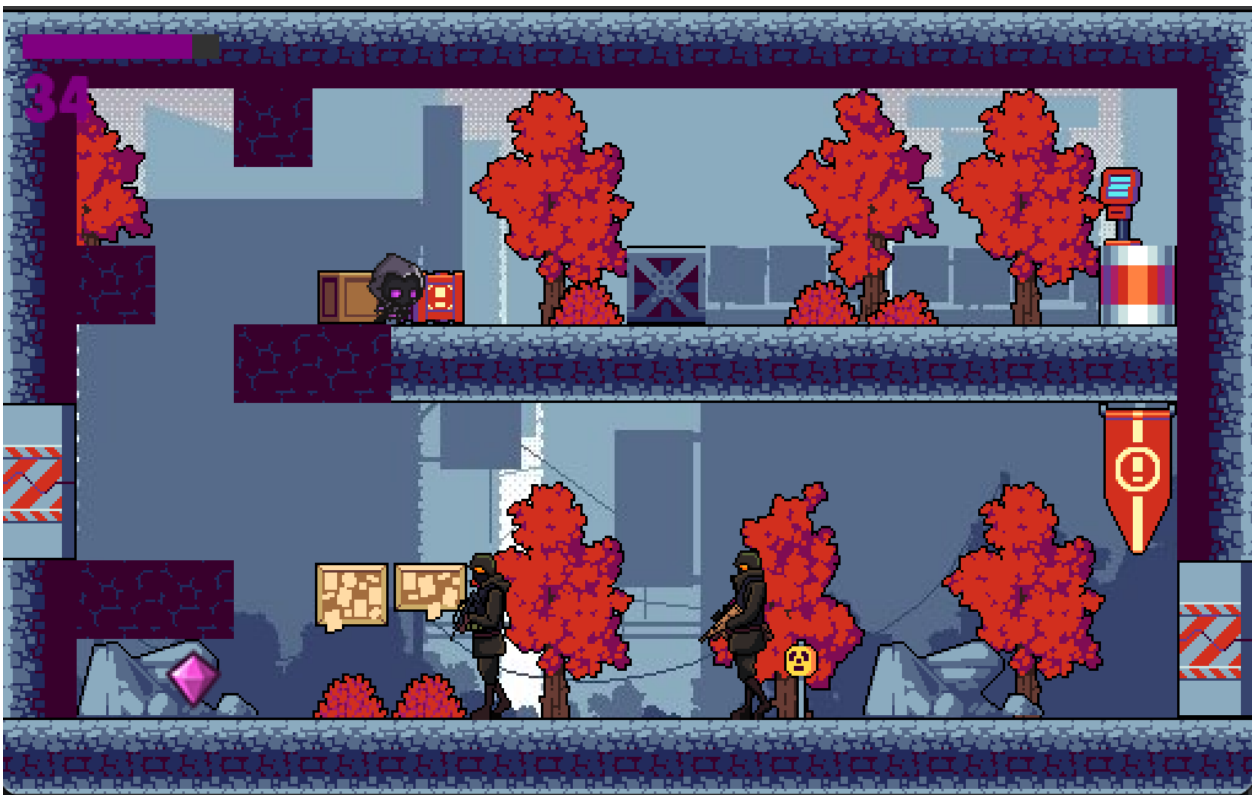


Рисунок 6 – 6 комната (2 уровень)

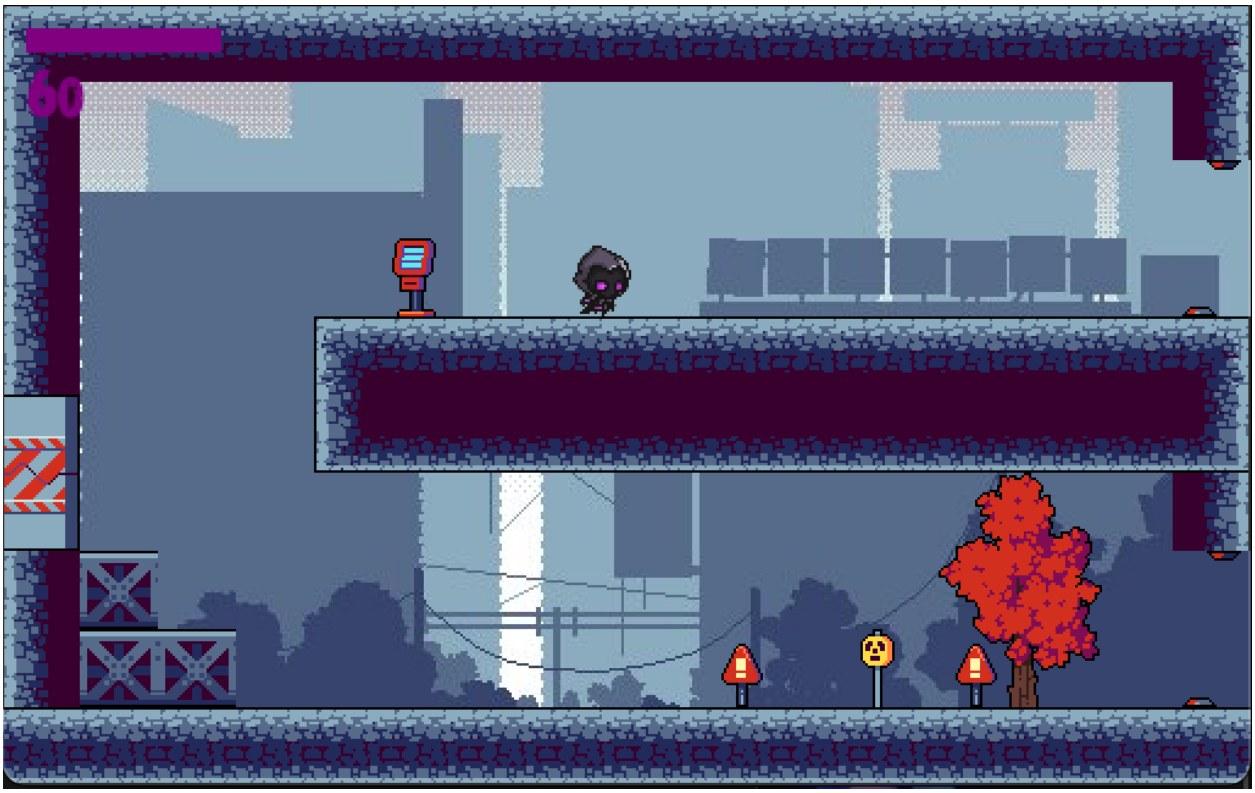


Рисунок 7 – 7 комната (выбор пути)

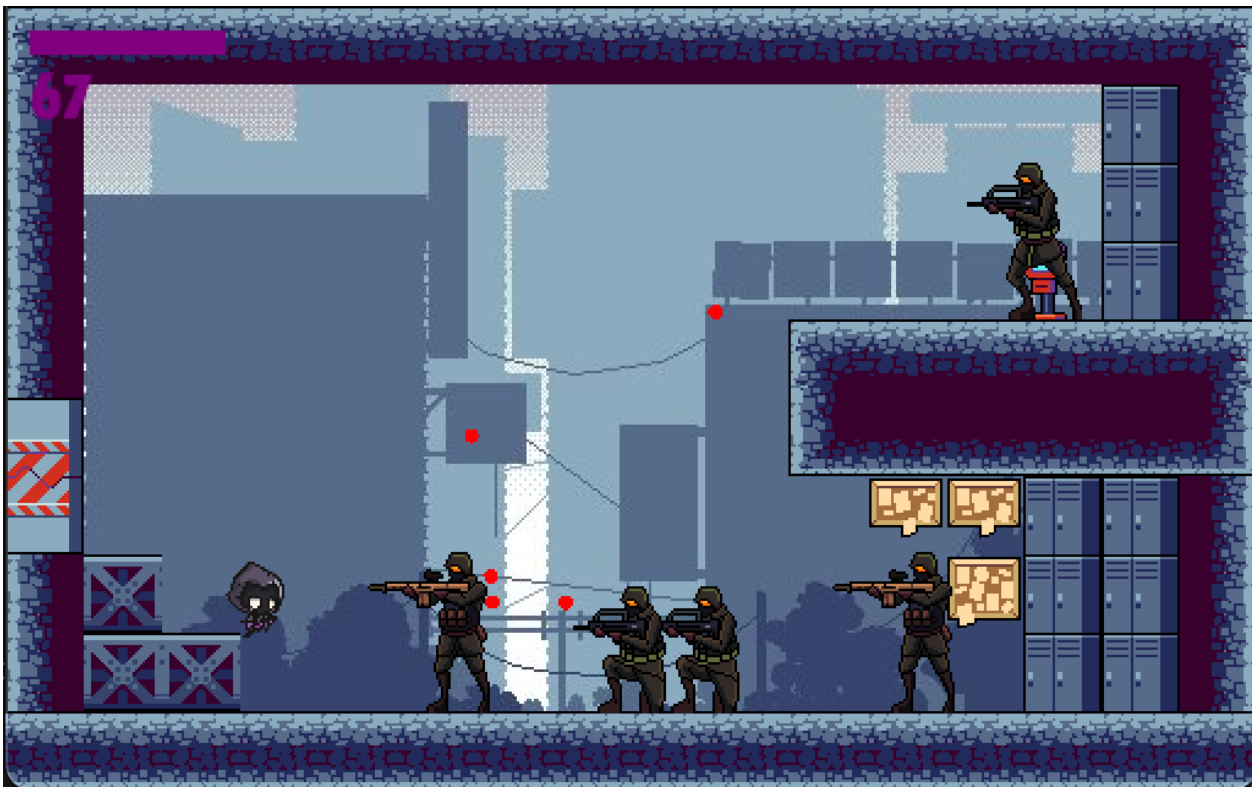
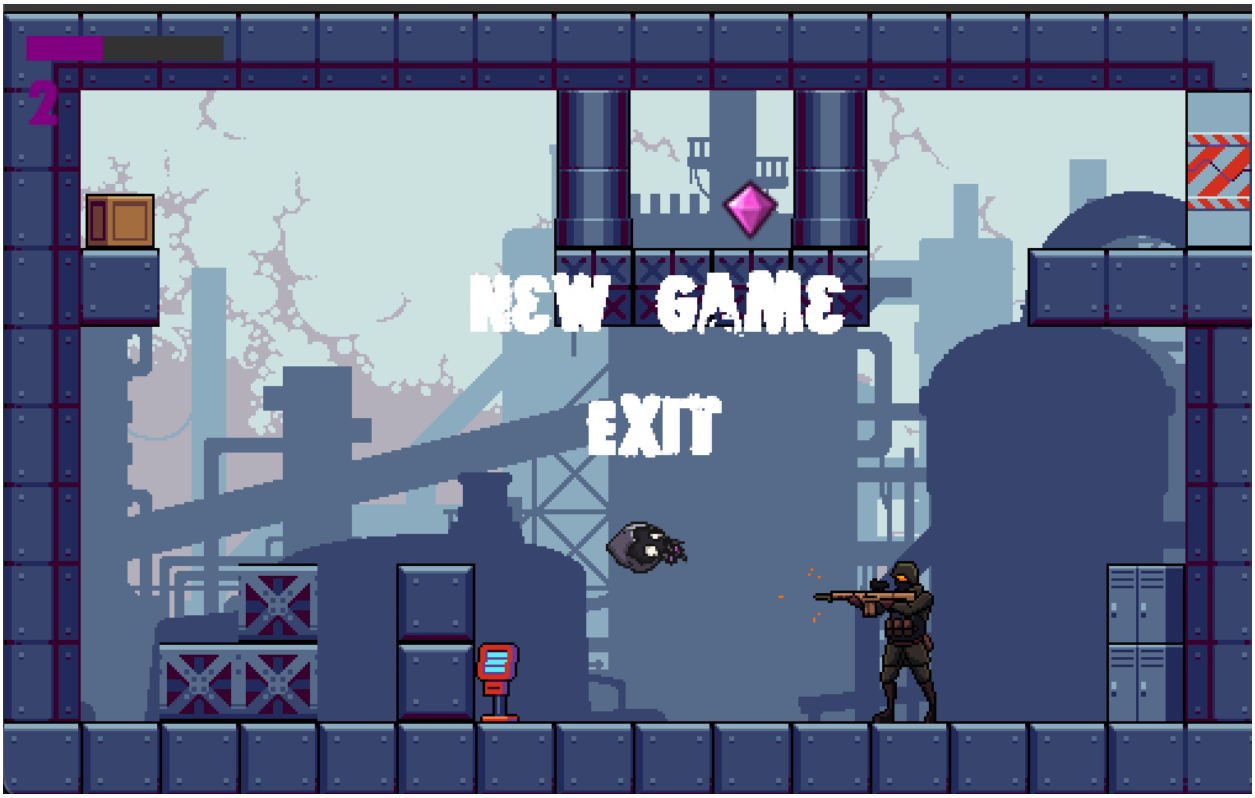


Рисунок 8 – 8 комната (ловушка)

При проигрыше игроку предлагается выбрать дальнейшие действия: “Выход” или “Новая игра” (см. Рисунок 9). При победе перед игроком выходит окно статистики игры и возможность начать новую (см. Рисунок 10)



(Рисунок 9 – Меню смерти)

(Рисунок 10 – Меню победы игрока)

5 Архив с исходным кодом приложения

https://github.com/Demos-gloryofRome44/Fantom

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. Наиболее интересными этапом работы было продумывание основной логики игры и ее графическое оформление.

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1. SFML Documentation. (n.d.). Retrieved from https://www.sfml-dev.org/documentation/
2. YouTube Tutorials and Online Courses.
3. Git Documentation. (n.d.). Retrieved from https://git-scm.com/doc