Пояснительная Записка к Курсовой Работе

Оглавление

1. Введение ***1***

2. Техническое Задание ***2***

2.1. Требования к функциональным характеристикам программы ***2***

2.2. Требования к составу и параметрам технических средств ***2***

2.3. Порядок контроля и приемки ***2***

2.3.1. Описание подготовки к проведению испытаний ***2***

2.3.2. Список критериев, по которым определяется работоспособность программы ***3***

3. Отчёт о разработке программы ***3***

3.1. Архитектура программы ***3***

3.2. Алгоритмическая часть ***7***

3.3. Графический интерфейс программы ***7***

3.4. Инструкция по сборке программы ***8***

4. Заключение ***10***

5. Архив с исходным кодом приложения. ***10***

1. 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.

2. Техническое Задание

2.1. Требования к функциональным характеристикам программы

Игра "Fantom" должна включать следующие функциональные элементы:

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.

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

Интерфейс: Игра должна иметь интуитивно понятный интерфейс с кнопками для начала игры, выбора уровня сложности и выхода из игры.

2.2. Требования к составу и параметрам технических средств

Язык разработки: C++.

Используемые сторонние библиотеки: SFML (Simple and Fast Multimedia Library) для работы с графикой и звуком.

2.3. Порядок контроля и приемки

2.3.1. Описание подготовки к проведению испытаний

Для проведения испытаний необходимо установить среду разработки (например, Visual Studio или Visual Studio Code) и библиотеку SFML.

2.3.2. Список критериев, по которым определяется работоспособность программы

Все 9 комнат реализованы с охраной.

Реализованы комнаты ловушки, из которых нужно выбираться.

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

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

При попадании пули охраны игра завершается с выводом кнопок «Выход» и «Новая игра».

Интерфейс игры интуитивно понятен и позволяет легко управлять игровым процессом.

3. Отчёт о разработке программы

3.1. 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 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 отрисовку 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 отрисовку 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) - Отрисовывает фоновую текстуру. Проходит по двумерному вектору map, отрисовывает 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 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 Explosion отвечает за визуализацию и анимацию взрывов. Он управляет состоянием взрыва, включая его масштаб, время анимации и отрисовку на экране.

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 отрисовки 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 Bullet представляет снаряды, которые стреляют враги. Он управляет созданием, движением и отрисовкой пуль, а также их взаимодействием с другими объектами, такими как игрок и стены.

3.2. Алгоритмическая часть

В игре реализован алгоритм поиска пути для охраны (например, A* или Dijkstra), который позволяет стражам находить игрока в пределах комнаты.

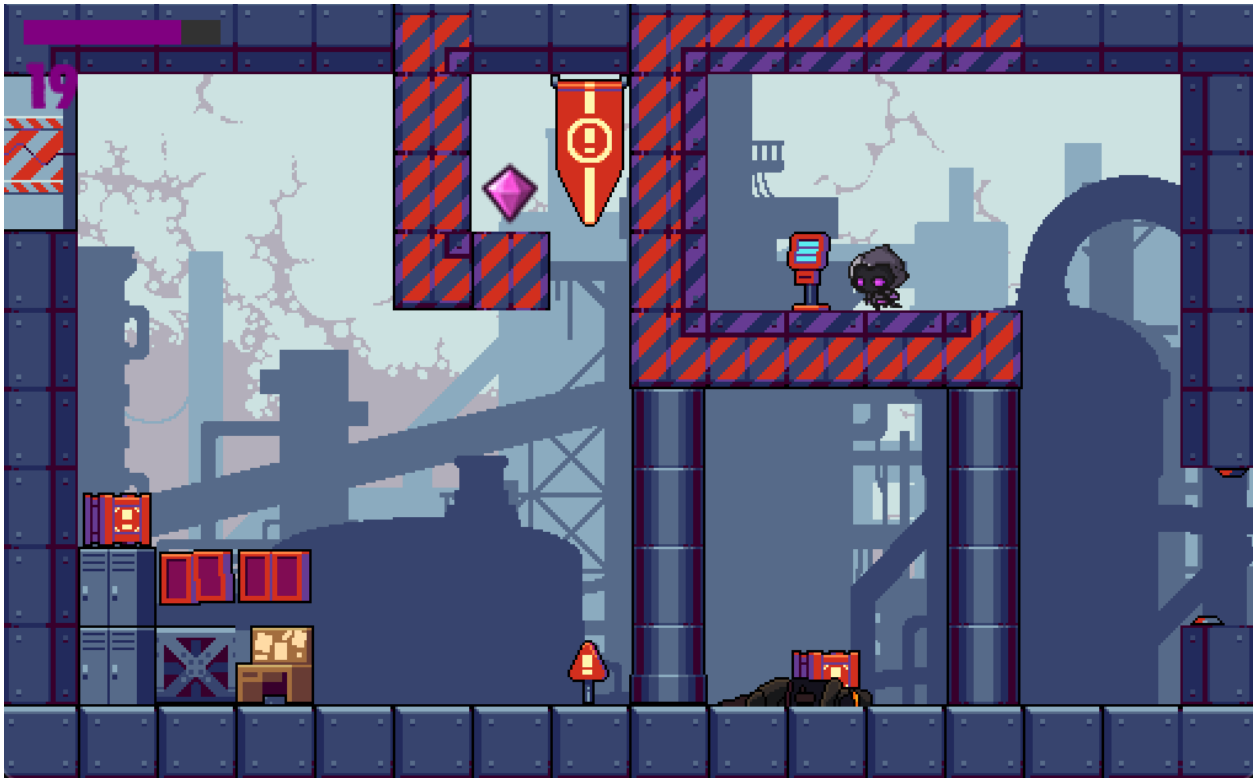
3.3. Графический интерфейс программы

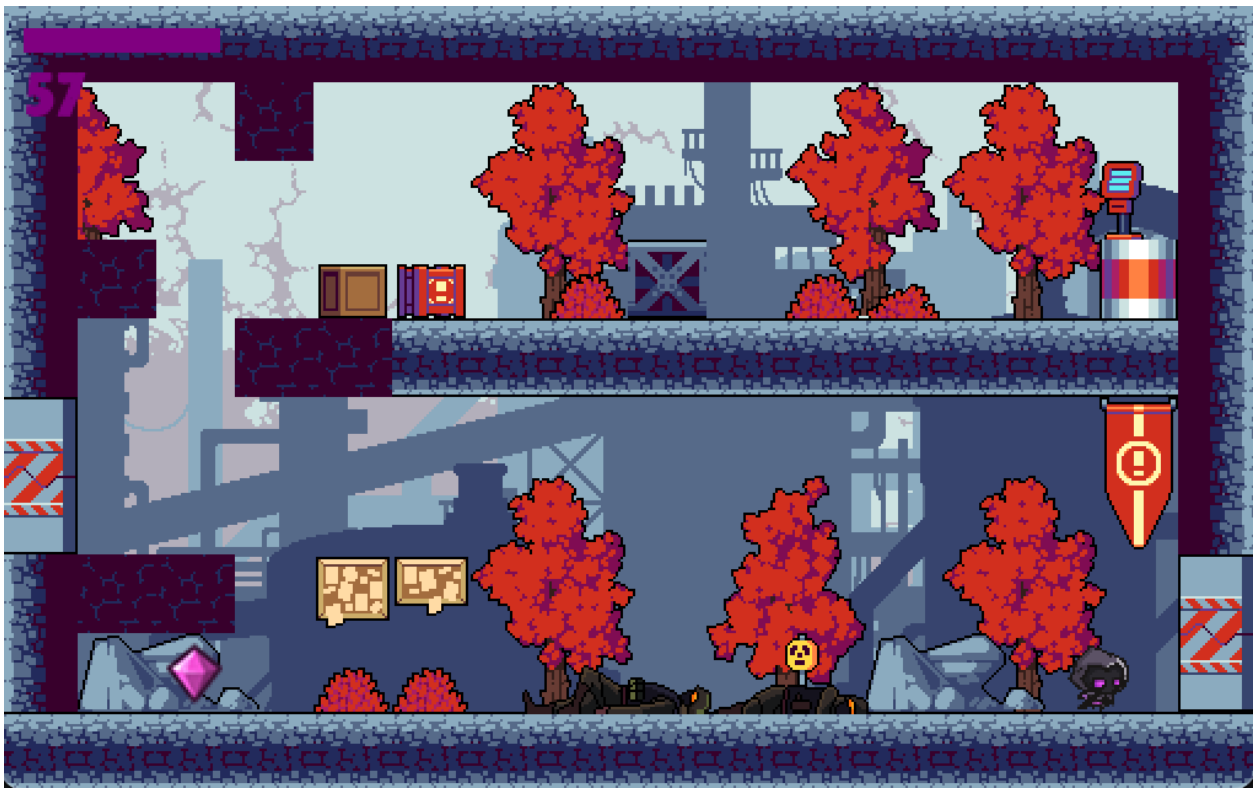
В графическом интерфейсе представлены основные элементы управления:

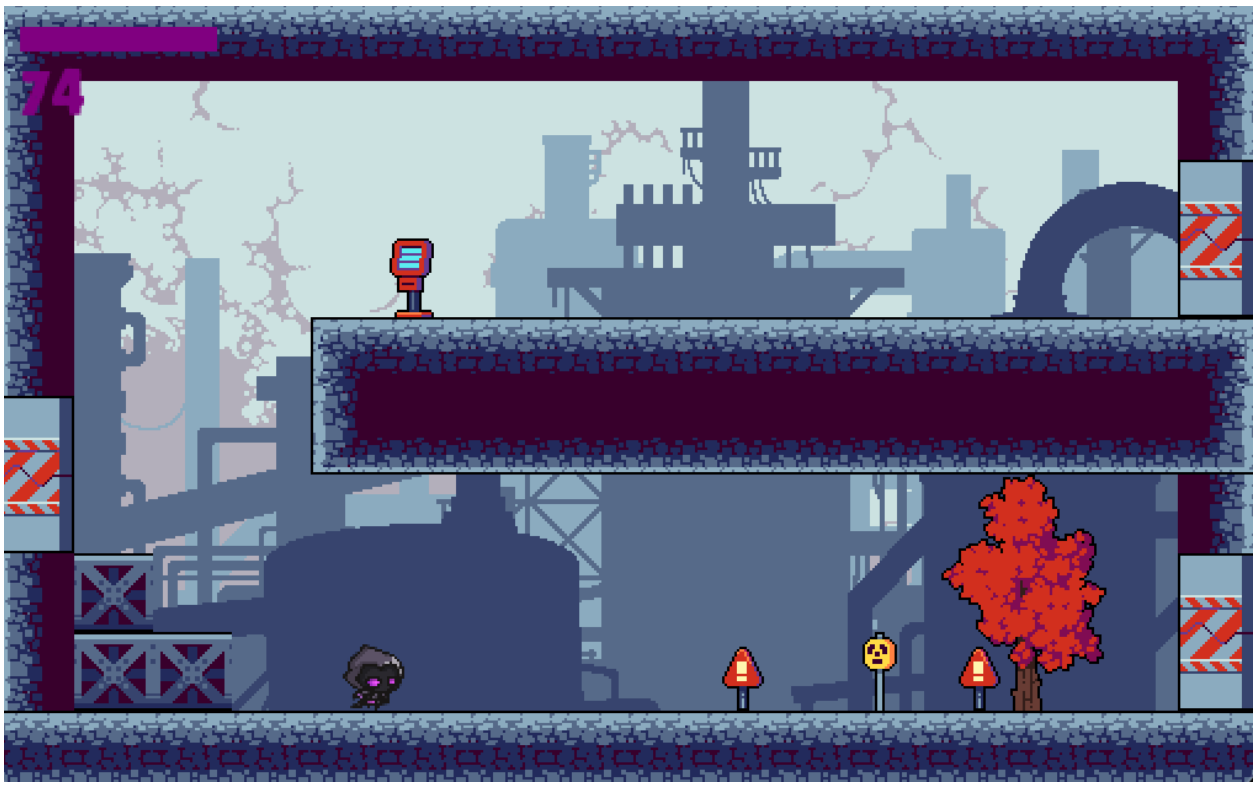
Кнопка "Начать игру"

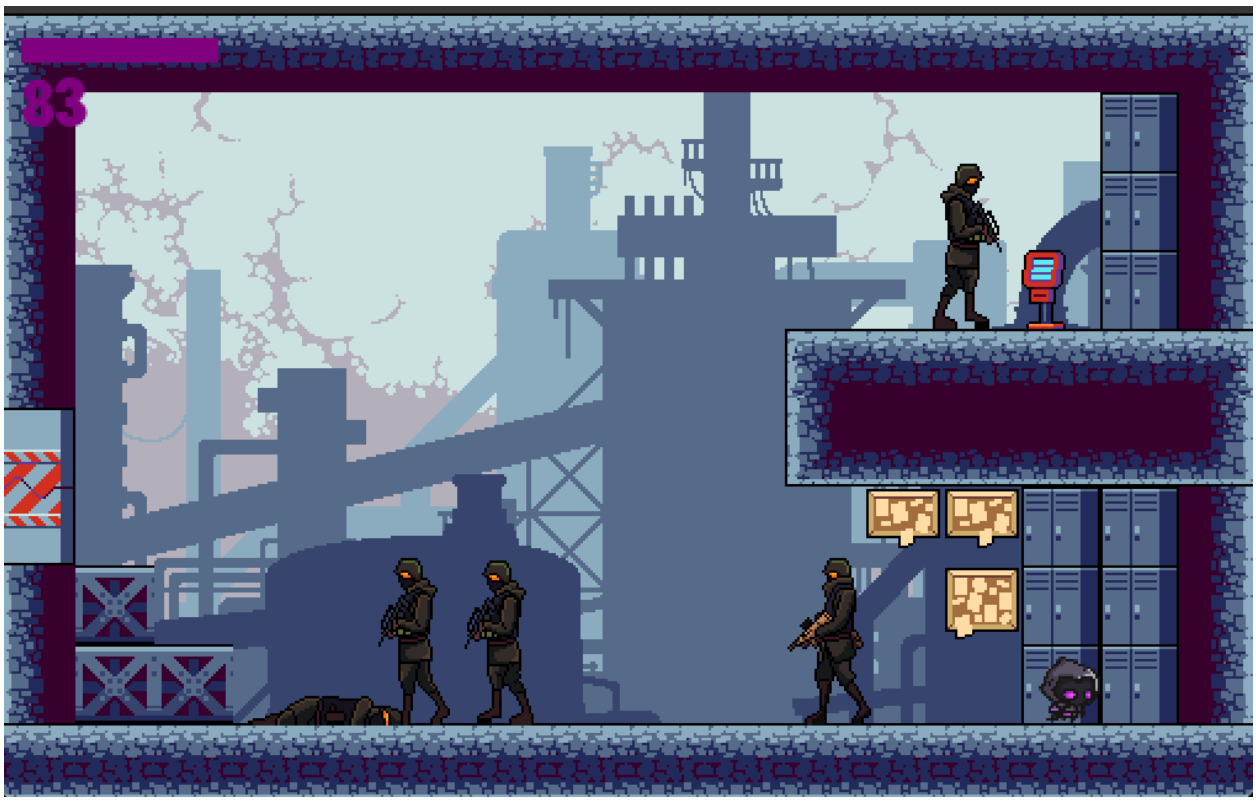
Кнопка "Настройки"

Примеры игровых карт:




57


74



3.4. Инструкция по сборке программы

Перед сборкой программы необходимо установить среду разработки (например, Visual Studio) и подключить последние стабильные версии библиотек SFML в makefile для автоматизации процесса сборки вашего проекта.

4. 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 этапами работы стали реализация механики охраны алгоритма поиска противника.

5. Архив с исходным кодом приложения

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