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 Endless Survival»

Обучающийся _____ *Мяктинов Никита Алексеевич*
Подпись

Шифр 23К0049

Группа КМБО-02-23

Руководитель
работы _____ *Черноусов Игорь Дмитриевич*
Подпись

Москва 2024

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Данный документ представляет собой пояснительную записку к курсовой работе по предмету “Методы и стандарты программирования”. Темой курсовой работы является разработка игры “Endless Survivors”, представляющей собой rogue-like – игру, в которой игроку предстоит сразить множество врагов, улучшить оружие и найти звезду Эдема для победы.

1 ПОСТАНОВКА ЗАДАЧИ

1.1 Требования к функциональным характеристикам программы

Игра “Endless Survivors” должна включать следующие функциональные элементы:

Игровой процесс: игрок должен иметь возможность исследовать открытый мир, взаимодействовать с различными объектами и сражаться с врагами.

Система здоровья и опыта: игрок должен иметь полоску здоровья и опыта, отображаемую на экране. Полоска здоровья должна уменьшаться при получении урона и восстанавливаться со временем

Враги должны обладать уникальными способностями, такими как бег, прыжок, огромное значение здоровья.

Снаряды и взрывы: игрок должен иметь возможность стрелять. Снаряды могут быть особенными. Взрывы должны наносить урон врагам в радиусе, критический урон должен быть повышенным.

Подбираемые предметы: в игровом мире должны генерироваться подбираемые предметы, такие как зелья и Звезда Эдема. Игрок должен получать бонусы при сборе этих предметов.

Анимации и текстуры: все игровые объекты должны иметь соответствующие текстуры и анимации для различных состояний.

Интерфейс пользователя: интерфейс должен отображать важную информацию, такую как здоровье, опыт, текущие задания и цели. Он должен быть интуитивно понятным и легко читаемым, чтобы игрок мог быстро ориентироваться и принимать решения.

Система волн врагов: враги должны появляться волнами, с увеличением сложности по мере прогресса игрока. Каждая волна должна включать различных врагов и увеличивать их количество, чтобы поддерживать интерес и вызов для игрока.

2 ОТЧЁТ О РАЗРАБОТКЕ ПРОГРАММЫ

2.1 Архитектура программы

Класс Player – представляет игрока, управляет его состоянием, статистикой, движением и взаимодействием с окружающим миром.

Прототипы функций:

- Player(float x, float y, float size, const sf::Texture& texture) – конструктор, инициализирует игрока с заданной позицией, размером и текстурой;
- void handleInput(); - Обрабатывает ввод пользователя для управления движением игрока;
- void update(float dt, World& world) – обновляет состояние игрока с учетом времени и взаимодействия с миром;
- void draw(sf::RenderWindow& window) – отрисовывает игрока на окне;
- sf::Vector2f getPosition() const – возвращает текущую позицию игрока;
- sf::FloatRect getBounds() const – возвращает границы игрока для проверки столкновений;
- bool isDead() const – проверяет, мертв ли игрок;
- bool hasWon() const – проверяет, выиграл ли игрок;
- bool restartRequested() const – проверяет, запрошен ли перезапуск игры;
- float getHealth() const – возвращает текущее здоровье игрока;
- float getMaxHealth() const – возвращает максимальное здоровье игрока;

- float getCurrentExperience() const – возвращает текущий опыт игрока;
- float getExperienceForNextLevel() const – возвращает необходимый опыт для следующего уровня;
- int getLevel() const – возвращает текущий уровень игрока;
- float getSpeed() const – возвращает скорость игрока;
- float getDamage() const – возвращает урон игрока;
- void resetRestartRequest() – сбрасывает запрос на перезапуск игры;
- void winGame() – обрабатывает логику победы в игре;
- void showDeathMenu(sf::RenderWindow& window) – отображает меню смерти;
- void showWinMenu(sf::RenderWindow& window) – отображает меню победы;
- void takeDamage(float amount) – уменьшает здоровье игрока на заданную величину при получении урона;
- void gainExperience(float amount); - увеличивает опыт игрока на указанную величину;
- void applyUpgrade(const std::string& upgradeName) – применяет улучшение к игроку по названию улучшения;
- void updateAnimation(float dt) – обновляет анимацию игрока в зависимости от состояния;
- void levelUp() – повышает уровень игрока и улучшает его характеристики.

Класс Enemy (абстрактный класс) – базовый класс для всех типов врагов, определяет общий интерфейс и поведение.

Прототипы функций:

- virtual void update(float dt, const sf::Vector2f& playerPosition, const std::vector<std::unique_ptr<Enemy>>& enemies, Player& player,

sf::RenderWindow& window) = 0 – чисто виртуальная функция для обновления состояния врага;

- virtual void draw(sf::RenderWindow& window) = 0 – чисто виртуальная функция для отрисовки врага;
- virtual void takeDamage(float amount, sf::RenderWindow& window) = 0 – чисто виртуальная функция для применения урона к врагу;
- virtual bool isDead() const = 0 – чисто виртуальная функция для проверки, мертв ли враг;
- virtual sf::FloatRect getBounds() const = 0 – чисто виртуальная функция для получения границ врага;
- virtual sf::Vector2f getPosition() const = 0 – чисто виртуальная функция для получения позиции врага.

Класс Runner (наследует Enemy) – быстрый враг, который преследует игрока.

Прототипы функций:

- Runner(float x, float y, const sf::Texture& walkTexture, const sf::Texture& attackTexture, const sf::Texture& deathTexture, Player& player) – конструктор, инициализирует бегуна с заданными текстурами и ссылкой на игрока;
- void update(float dt, const sf::Vector2f& playerPosition, const std::vector<std::unique_ptr<Enemy>>& enemies, Player& player, sf::RenderWindow& window) override – обновляет состояние бегуна, включая движение и анимацию;
- void draw(sf::RenderWindow& window) override – отрисовывает бегуна на окне;
- void takeDamage(float amount, sf::RenderWindow& window) override – применяет урон к бегуну и управляет его состоянием смерти;
- bool isDead() const override – проверяет, мертв ли бегун.

- sf::FloatRect getBounds() const override – возвращает границы бегуна для проверки столкновений;
- sf::Vector2f getPosition() const override – возвращает текущую позицию бегуна;
- void updateAnimation(float dt) – изменяет анимацию бегуна со временем.

Класс Jumper (наследует Enemy) – враг, способный прыгать к игроку.

Прототипы функций:

- Jumper(float x, float y, const sf::Texture& walkTexture, const sf::Texture& attackTexture, const sf::Texture& deathTexture, const sf::Texture& jumpTexture, Player& player) – конструктор, инициализирует джампера с заданными текстурами и ссылкой на игрока;
- void update(float dt, const sf::Vector2f& playerPosition, const std::vector<std::unique_ptr<Enemy>>& enemies, Player& player, sf::RenderWindow& window) override – обновляет состояние джампера, включая логику прыжка;
- void draw(sf::RenderWindow& window) override – отрисовывает джампера на окне;
- void takeDamage(float amount, sf::RenderWindow& window) override – применяет урон к джамперу;
- bool isDead() const override – проверяет, мертв ли джампер.
- sf::FloatRect getBounds() const override – возвращает границы джампера для проверки столкновений;
- sf::Vector2f getPosition() const override – возвращает текущую позицию джампера;
- void updateAnimation(float dt) – изменяет анимацию прыгуна со временем;

- sf::Vector2f calculateJumpPosition(float t) – расчёт позиции в прыжке.

Класс Boss (наследует Enemy) – мощный враг с большим здоровьем и уроном.

Прототипы функций:

- Boss(float x, float y, const sf::Texture& walkTexture, const sf::Texture& attackTexture, const sf::Texture& deathTexture, Player& player) – конструктор, инициализирует босса с заданными текстурами и ссылкой на игрока;
- void update(float dt, const sf::Vector2f& playerPosition, const std::vector<std::unique_ptr<Enemy>>& enemies, Player& player, sf::RenderWindow& window) override – обновляет состояние босса, включая движение и атаки;
- void draw(sf::RenderWindow& window) override – отрисовывает босса на окне;
- void takeDamage(float amount, sf::RenderWindow& window) override – применяет урон к боссу;
- bool isDead() const override – проверяет, мертв ли босс;
- sf::FloatRect getBounds() const override – возвращает границы босса для проверки столкновений;
- sf::Vector2f getPosition() const override – возвращает текущую позицию босса;
- void updateAnimation(float dt) – изменяет анимацию босса со временем.

Класс World – создает и управляет игровым миром, включая генерацию чанков, биомов и объектов.

Прототипы функций:

- World(int chunkSize, int blockSize, int viewRadius, const sf::Texture& grass_texture, const sf::Texture& tree_texture, const sf::Texture& pickable_texture, const sf::Texture& gregStarTexture, const sf::Texture& cactus_texture) – конструктор, инициализирует мир с заданными параметрами и текстурами;
- void update(const sf::Vector2i& playerChunkPos, Player& player, sf::RenderWindow& window) – обновляет мир в зависимости от позиции игрока;
- void draw(sf::RenderWindow& window, const sf::Vector2f& playerPos) – отрисовывает окружающий мир вокруг позиции игрока;
- bool checkCollision(const sf::FloatRect& bounds) const – проверяет столкновения с объектами мира;
- std::vector<Pickable>& getPickables() – возвращает список подбираемых предметов в мире;
- Biome getBiomeAt(float x, float y) const – определяет биом в заданной позиции;
- bool shouldSpawnObject(Biome biome, float detailValue) const – проверяет, следует ли спавнить объект в данном биоме;
- sf::Color getGroundColor(float biomeValue) const – возвращает цвет земли в зависимости от значения биома;
- float getBiomeValue(float x, float y) const – возвращает значение биома в заданной позиции;
- void generateChunk(const sf::Vector2i& chunkPos) – генерирует новый чанк в заданной позиции;
- void generatePickablesInChunk(const sf::Vector2i& chunkPos) – распределяет подбираемые предметы внутри чанка;
- void generateGregStarInChunk(const sf::Vector2i& chunkPos) – генерирует особые объекты (например, «GregStar») в чанке;

- void updatePickables(Player& player, sf::RenderWindow& window) – обновляет состояние подбираемых предметов и проверяет их сбор игроком.

Класс Chunk – представляет участок мира, состоящий из блоков.

Прототипы функций:

- Chunk(int chunkX, int 11hunky, int blockSize, int chunkSize, const sf::Texture& grass_texture, const sf::Texture& tree_texture) – конструктор, создает чанк с заданными параметрами и текстурами;
- void draw(sf::RenderWindow& window) – отрисовывает все блоки внутри чанка;
- const std::vector<Block>& getBlocks() const – возвращает список блоков в чанке;
- void setBlocks(std::vector<Block>&& newBlocks) – устанавливает новые блоки для чанка.

Класс Block – представляет отдельный блок в мире.

Прототипы функций:

- Block(float x, float y, float size, const sf::Texture& texture, bool isWalkable) – конструктор, создает блок с заданными параметрами и текстурой;
- Block(const Block& other) – конструктор копирования;
- Block& operator=(const Block& other) – оператор присваивания;
- void draw(sf::RenderTarget& target) – отрисовывает блок на заданной поверхности;
- bool isWalkable() const – возвращает, можно ли пройти через блок;
- sf::FloatRect getBounds() const – возвращает границы блока для проверки столкновений;
- void setColor(const sf::Color& color) – устанавливает цвет блока.

Класс Projectile – представляет снаряд, выпущенный игроком.

Прототипы функций:

- Projectile(sf::Vector2f position, sf::Vector2f direction, float speed, float damage, float size, float lifetime, int pierces, float critChance, float critMult, float explChance, float explRadius) – конструктор, создает снаряд с заданными параметрами;
- void update(float dt) – обновляет позицию снаряда с учетом времени;
- void draw(sf::RenderWindow& window) – отрисовывает снаряд на окне;
- bool checkCollision(const std::unique_ptr<Enemy>& enemy, sf::RenderWindow& window) – проверяет столкновение снаряда с врагом;
- void applyExplosionDamage(const std::vector<std::unique_ptr<Enemy>>& enemies, sf::RenderWindow& window) – применяет урон от взрыва к врагам в радиусе;
- bool isExpired() const – проверяет, истекло ли время жизни снаряда;
- sf::FloatRect getBounds() const – возвращает границы снаряда для проверки столкновений;
- float getDamage() const – возвращает урон снаряда;
- float getExplosionRadius() const – возвращает радиус взрыва снаряда;
- sf::Vector2f getPosition() const – возвращает текущую позицию снаряда;
- int getRemainingPierces() const – возвращает количество оставшихся пробиваний снаряда;
- bool hasExploded() const – проверяет, взорвался ли снаряд.

Класс ProjectileManager – управляет всеми активными снарядами в игре.

Прототипы функций:

- ProjectileManager(sf::RenderWindow& window, Player& player) – конструктор, инициализирует менеджер снарядов;
- void update(float dt, const std::vector<std::unique_ptr<Enemy>>& enemies) – обновляет все снаряды и обрабатывает их столкновения с врагами;
- void draw() – отрисовывает все активные снаряды;
- void shoot(const sf::Vector2f& startPosition, const sf::Vector2f& targetPosition) – создает новый снаряд по направлению к цели;
- void handleMouseInput(const sf::Vector2f& playerPosition, const sf::View& view) – обрабатывает ввод мыши для стрельбы;
- void checkCollisions(const std::vector<std::unique_ptr<Enemy>>& enemies) – проверяет и обрабатывает столкновения снарядов с врагами.

Класс UIManager – управляет отображением пользовательского интерфейса, включая здоровье, опыт и различные меню.

Прототипы функций:

- UIManager(sf::RenderWindow& window, Player& player) – конструктор, инициализирует менеджер интерфейса с окном и игроком;
- void update() – обновляет элементы интерфейса на основе текущего состояния игрока;
- void draw() – отрисовывает элементы интерфейса на окне;
- void updateHealthBar() – обновляет отображение полоски здоровья игрока;
- void updateExperienceBar() – обновляет отображение полоски опыта игрока;
- void updateStatTexts() – обновляет текстовые элементы статистики игрока;
- void showUpgradeMenu() – отображает меню улучшений при повышении уровня игрока;

- void showDeathMenu() – отображает меню смерти при гибели игрока;
- void showWinMenu() – отображает меню победы при завершении игры.

Класс TextureManager – управляет загрузкой и хранением текстур для использования в игре.

Прототипы функций:

- TextureManager() – конструктор, инициализирует менеджер текстур и загружает необходимые ресурсы;
- const sf::Texture& getTexture(const std::string& textureName) const – возвращает ссылку на текстуру по имени textureName;
- const sf::Font& getFont(const std::string& fontName) const – возвращает ссылку на шрифт по имени fontName;
- sf::Music& getMusic(const std::string& musicName) – возвращает ссылку на музыку по имени musicName;
- void loadTexture(const std::string& textureName, const std::string& filePath) – загружает текстуру из файла filePath и сохраняет её под именем textureName;
- void loadFont(const std::string& fontName, const std::string& filePath) – загружает шрифт из файла filePath и сохраняет его под именем fontName;
- void loadMusic(const std::string& musicName, const std::string& filePath) – загружает музыку из файла filePath и сохраняет её под именем musicName.

Класс WaveManager (или EnemyManager) – управляет спавном врагов и прогрессией волн.

Прототипы функций:

- EnemyManager(TextureManager& textureManager, Player& player, sf::RenderWindow& window) – конструктор, инициализирует менеджер врагов с текстурным менеджером, игроком и окном;
- void update(float dt, const sf::Vector2f& playerPos) – обновляет состояние врагов и текущей волны с учетом времени и позиции игрока;
- void draw() – отрисовывает всех активных врагов на окне;
- const std::vector<std::unique_ptr<Enemy>>& getEnemies() const – возвращает список всех активных врагов;
- void initializeWaves() – инициализирует параметры и настройки для различных волн врагов;
- void updateWave(float dt) – обновляет состояние текущей волны с учетом времени;
- void spawnEnemies(const sf::Vector2f& playerPos) – спавнит новых врагов в зависимости от позиции игрока;
- std::unique_ptr<Enemy> createEnemy(const std::string& type, const sf::Vector2f& position) – создает нового врага заданного типа в указанной позиции;
- sf::Vector2f getSpawnPosition(const sf::Vector2f& playerPos) – вычисляет позицию для спавна врагов относительно позиции игрока.

Класс Pickable – представляет подбираемые предметы в игровом мире, такие как опыт или здоровье.

Прототипы функций:

- Pickable(const sf::Vector2f& position, const sf::Texture& texture, const std::string& type) – конструктор, инициализирует подбираемый предмет с заданной позицией, текстурой и типом;
- void update(float dt) – обновляет состояние подбираемого предмета с учетом времени;

- void draw(sf::RenderWindow& window) – отрисовывает подбираемый предмет на окне;
- sf::FloatRect getBounds() const – возвращает границы подбираемого предмета для проверки столкновений;
- std::string getType() const – возвращает тип подбираемого предмета;
- sf::Vector2f getPosition() const – возвращает текущую позицию подбираемого предмета.

Файл Main.cpp – точка входа в приложение, инициализирует игру и содержит главный игровой цикл.

Функция main, запускает игру, обрабатывает события, обновляет и отрисовывает игровые объекты.

Файл Const.h – содержит глобальные константы, используемые по всему проекту.

Содержимое:

- константы окна (размеры, заголовок);
- пути к ресурсам (текстуры, шрифты, звуки);
- параметры игрока (скорость, здоровье, урон);
- параметры врагов (типы, здоровье, урон);
- параметры игровых объектов (размеры, шансы спавна);
- константы анимации (длительность кадров, количество кадров).

2.2 Алгоритмическая часть

Процедурная генерация мира в игре осуществляется с использованием перлин шума для создания биомов и деталей. Мир состоит из чанков, каждый из которых содержит множество блоков. Для каждого блока определяется биом

(лес или пустыня) и цвет земли. Затем, на основе значений шума, в блоках спавнятся объекты, такие как деревья или кактусы, а также подбираемые предметы, такие как зелья опыта.

3 РУКОВОДСТВО ПО СБОРКЕ И ЗАПУСКУ

3.1 Требования

- CMake (версия 3.28 или выше)
- MinGW (для сборки. Крайне желательно использовать MSYS2 установщик)
- Git (для клонирования репозитория)

3.2 Сборка и запуск проекта

- перейдите в директорию проекта: “cd EndlessSurvivors”;
- создайте директорию сборки и перейдите в неё: “mkdir build && cd build”;
- запустите CMake для генерации файлов сборки: “cmake -G "MinGW Makefiles" ..”;
- соберите проект с помощью Make: “mingw32-make”;
- запустите игру: “cd bin && main.exe”.

Дополнительно: настоятельно рекомендуется ознакомиться с инструкций по полной установке в файле “readme.md”.

4 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

4.1 Управление

Управление персонажем осуществляется через клавиши WASD. Стрельба по нажатию левой кнопкой мыши в точку, где находится курсор.

4.2 Графический интерфейс программы

Графический интерфейс представляет пользователю возможность отслеживать изменение характеристик (см. Рисунок 1)

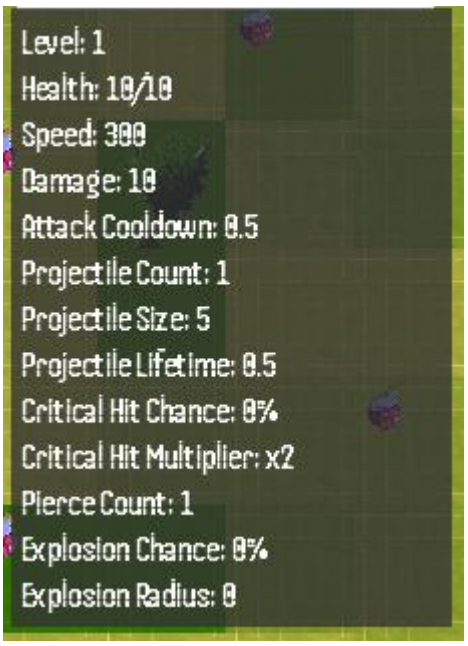

Рисунок 1 – характеристики в UI

Также пользователь видит полоску, отображающую его текущее здоровье над окном характеристик (см. Рисунок 2)

Рисунок 2 - полоса здоровья

Внизу экрана располагается полоска, показывая уровень опыт и сколько осталось до следующего уровня (см. Рисунок 3)

Рисунок 3 - Полоса опыта

При проигрыше игроку предлагается выбрать дальнейшие действия: выход или “restart” (см. Рисунок 4). При победе выбор аналогичен (см. Рисунок 5)

Рисунок 4 - Меню смерти игрока

Рисунок 5 - Меню победы игрока

При наборе достаточного количества очков опыта игрок может выбрать одну из двух обновлений его характеристик (см. Рисунок 6)

Рисунок 6 - Меню обновления характеристик

ЗАКЛЮЧЕНИЕ

В ходе выполнения курсовой работы были достигнуты поставленные цели по разработке игры “Endless Survivors”. Проект позволил углубить знания в области программирования на современном C++17, а также познакомиться с библиотекой SFML.

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1. Официальный сайт языка C++. – URL: https://en.cppreference.com/w/https://en.cppreference.com/w/ (дата обращения 20.12.24).

2. Официальная документация библиотеки SFML. – URL: https://www.sfml-dev.org/documentation/2.6.2/ (дата обращения 20.12.24).

3. Дополнительная библиотека, реализующая перлин шум. – URL: https://github.com/Reputeless/PerlinNoise/tree/master (дата обращения 20.12.24).