МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего образования
**«Южно-Уральский государственный университет
(национальный исследовательский университет)»**
Высшая школа электроники и компьютерных наук
Кафедра системного программирования

ДОПУСТИТЬ К ЗАЩИТЕ

Заведующий кафедрой, д.ф.-м.н.,
профессор

_____ Л.Б. Соколинский

«___»_____ 2024 г.

Разработка редактора локаций для компьютерной 2D-игры в жанре Survival RPG на платформе Godot Engine

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
ЮУрГУ – 02.03.02.2024.308-491.ВКР

Научный руководитель,
ст. преподаватель кафедры СП
_____ Я.А. Краева

Автор работы,
студент группы КЭ-401
_____ К.И. Слитиков

Ученый секретарь
(нормоконтролер)
_____ И.Д. Володченко
«___»_____ 2024 г.

Челябинск, 2024 г.

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего образования
«Южно-Уральский государственный университет
(национальный исследовательский университет)»
Высшая школа электроники и компьютерных наук
Кафедра системного программирования

УТВЕРЖДАЮ
Зав. кафедрой СП
_____ Л.Б. Соколинский
29.01.2024 г.

ЗАДАНИЕ
на выполнение выпускной квалификационной работы бакалавра
студенту группы КЭ-401
Слитикову Кириллу Игоревичу,
обучающемуся по направлению
02.03.02 «Фундаментальная информатика и информационные технологии»

1. **Тема работы** (утверждена приказом ректора от 22.04.2024 г. № 764-13/12)
Разработка редактора локаций для компьютерной 2D-игры в жанре Survival RPG на платформе Godot Engine.
2. **Срок сдачи студентом законченной работы:** 03.06.2024 г.
3. **Исходные данные к работе**
3.1. Godot Docs – 3.4 branch – Godot Engine (stable) documentation in English. [Электронный ресурс] URL: https://docs.godotengine.org/en/stable/ (дата обращения: 20.03.2023 г.).
3.2. Лассер А., Лассер М., Райтер У. BurnTime – Manual. // Max Design, 1993. – 1-е изд. – 134 с.
3.3. Реализованный симулятор ходьбы по миру игры «BurnTime» с наработками модулей персонажа, инвентаря и локаций.
4. **Перечень подлежащих разработке вопросов**
4.1. Завершить реализацию модулей персонажа и инвентаря.
4.2. Завершить реализацию модулей локаций и сооружений.

4.3. Реализовать модуль событий.

4.4. Реализовать модуль редактора локаций, с использованием всех разработанных ранее модулей.

5. Дата выдачи задания: 29.01.2024 г.

Научный руководитель,
ст. преподаватель кафедры СП Я.А. Краева

Задание принял к исполнению К.И. Слитиков

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Актуальность

На 2024 год игровая промышленность является одним из главных элементов индустрии развлечений. На текущий момент она занимает второе место на рынке, опережая по бюджетам таких крупных конкурентов как киноиндустрия, музыкальная и книжная индустрии.

Со временем разработка игровых проектов и процессы усложнения компонентов становятся проще, что позволяет большему количеству людей войти в эту сферу развлечений как разработчикам, соответственно, усложняя задачи, которые ставятся перед ними, что и является аспектом развития индустрии.

Необходимо также отметить, что разработка игрового продукта зачастую сопровождается созданием внутренних редакторов для упрощения и ускорения разработки поставленных задач. Благодаря этому и появились редакторы локаций, или уровневые редакторы, которые предоставляют необходимые инструменты для создания, изменения и настройки игрового контента не только разработчикам, но и игрокам.

В качестве основы актуальности создания различных редакторов для игровых продуктов можно указать следующие аспекты.

1. Возможность существенно ускорить процесс создания игровых миров, реализации прототипов уровней и их тестирования.
2. Продление жизненного цикла продукта за счет того, что игры становятся более гибкими и адаптивными благодаря возможности создания собственного контента и модификации существующего.
3. Привлечение сообщества возможностью создавать и делиться своими наработками, что может быть отличной основой для идеи проекта и непосредственного направления в разработке.
4. Возможность обучения основам геймдизайна благодаря экспериментированию с созданием уровней и параллельной наработке навыков игрового дизайна.

Постановка задачи

Целью выпускной квалификационной работы является разработка редактора локаций для компьютерной 2D-игры в жанре Survival RPG на платформе Godot Engine. Для достижения поставленной цели необходимо решить следующие задачи:

1) провести обзор аналогов в данной области;
2) спроектировать приложение и работу модулей;
3) реализовать и протестировать итоговое приложение и модули.

Структура и содержание работы

Работа состоит из введения, четырех глав, заключения и списка литературы. Объем работы составляет 34 страницы, объем списка литературы – 15 источников.

В первой главе производится анализ предметной области и обзор аналогов приложения. В данной главе описываются особенности работы редакторов локаций аналогов и их смысл для основного игрового приложения, обосновывается выбор средств реализации и определяется эффективность различных аспектов редакторов.

Вторая глава посвящена проектированию системы и логики работы модулей и приложения между друг другом. В ней представлены схемы и диаграммы, реализованные при разработке.

В третьей главе описан процесс реализации программы.

В четвертой главе проведено комплексное тестирование приложения.

В приложениях А и Б представлен коды реализации инвентаря и скриптов из реализации самого модуля редактора.

1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

1.1. Предметная область

Для разработки выбрана реализация редактора локаций для 2D-игры в жанре Survival RPG.

Survival RPG – жанр игр сочетающий элементы двух жанров: Survival и RPG. В основе игр жанра лежит все та же ролевая составляющая, но с элементами выживания, степень которых отличается в зависимости от проекта. Аспектами выживания могут быть как простые показатели голода и жажды, так и целые системы состояний и эффектов, усложняющих процесс самого выживания. Ролевые игры – жанр, который берет основные аспекты из классических ролевых игр, такие как: различные числовые характеристики, способности, умения и игровые механики.

Обращаясь к истокам жанра, можно заметить довольно значимую роль такого участника настольных ролевых игр как «Мастер». «Мастер» представляет из себя человека, который создает основной сюжет и контролирует развитие игровых событий в зависимости от действий игроков. Соответственно, главной задачей этого человека является создать некое явление игрового мира, достаточное, для погружения в выбранные игроками роли. Именно для исполнения роли «Мастера» и создаются редакторы, которые позволяют создавать сценарии, сюжеты и места в условиях возможностей движка и фантазии пользователя.

1.2. Обзор аналогов

К сожалению, в современных ролевых играх мало внимания уделяют такой возможности, как редактирование локаций и создание новых, но существуют примеры редакторов для различных игр с различным набором функциональных составляющих.

Серия игр «Heroes Of Might And Magic»

Рассмотрим редактор карт для третьей части серии [4]. Интерфейс делится на несколько частей.

1. Панель инструментов для переключения между списком инструментов, таких как: удаление, сохранение, создание новой карты, вырезания, копирования, просмотра свойств, изменения масштаба, переключения на подземный уровень, включение и отключение сетки и коллизий, ландшафт, реки дороги, ластик, леса, замки, монстры, герои, артефакты, ресурсы и наборы объектов для каждого типа ландшафта (рисунок 1).

Рисунок 1 – Панель инструментов редактора карт
«Heroes of Might and Magic 3»

2. Панель объектов, которая содержит список объектов или настроек для выбранной вкладки в панели инструментов и карту над ним (рисунок 2).

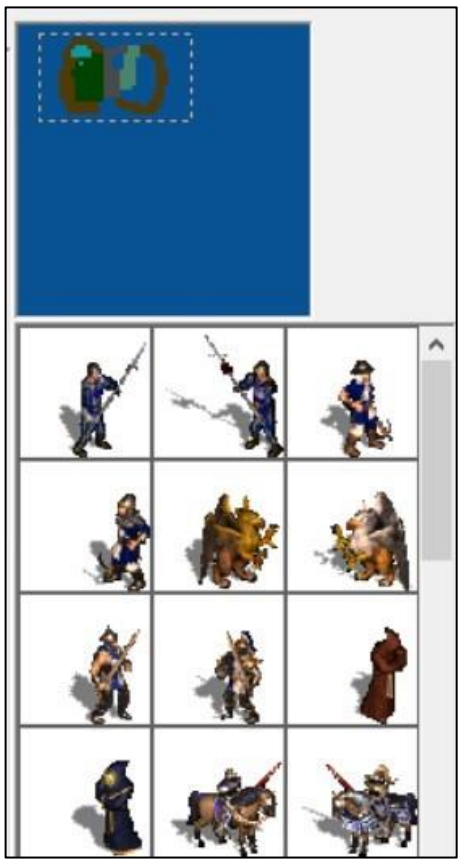

Рисунок 2 – Панель объектов и мини-карта редактора карт
«Heroes of Might and Magic 3»

3. Панель карты, на которой пользователь создает саму карту, используя две предыдущих панели (рисунок 3).

Рисунок 3 – Панель карты редактора карт «Heroes of Might and Magic 3»

Серия игр «Titan Quest»

Еще один пример редакторов, но уже более сложных. Titan Quest [5] имеет в своем арсенале также несколько редакторов: «Editor.exe», «QuestEditor.exe» и «ArtManager.exe». Последняя утилита в основном используется для перевода файлов карт, создаваемых в редакторе, в нужный формат, но также позволяет управлять модификациями. Редактор квестов является программой для создания и написания логических квестов для игры. Основной же редактор предназначен для создания новых уровней, при этом он легче других в освоении за счет проработанного графического интерфейса.

Рассмотрим основной редактор – «Editor.exe». В нем существует два основных режима просмотра: режим «Макета» и режим «Редактирования». Режим «Макета» представляет из себя окно перспективы, где отображается сам уровень. В данном окне программы не так много функционала, но здесь

предлагаются такие функции как: переключение перспективы, создание новых территорий и регионов, их перемещение и другие различные манипуляции между ними (рисунок 4).

Рисунок 4 – Режим «Макета» программы «Editor.exe»

В режиме «Редактирования» окно можно разделить на несколько следующих панелей:

1) верхняя панель, на которой представлены различные инструменты редактирования ландшафта и управления камерой, инструменты работы с документами, а также вкладка «», содержащая объекты (рисунок 5);

2) боковая панель – панель, которая содержит свойства, модификаторы, переменные, списки объектов.

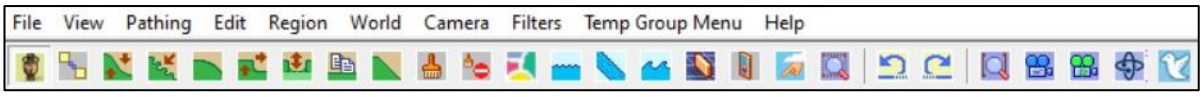

Рисунок 5 – Верхняя панель режима «Редактирования»

Боковая панель предназначена для детальной настройки работы тех или иных инструментов программы, выбора объектов и их предпросмотра.

Функциональная составляющая панели меняется в зависимости от выбранного инструмента. На рисунке 6 представлена боковая панель при выборе инструмента «Entity» из верхней панели и продемонстрировано окно созданной территории.

Рисунок 6 – Окно территории и боковая панель режима «Редактирования» программы «Editor.exe»

1.3. Обзор средства разработки

Для реализации будет использована среда разработки Godot Engine и встроенный в нее язык программирования GDScript. Godot Engine обладает визуальной средой для разработки и является бесплатным движком и открытым исходным кодом с возможностью создавать кроссплатформенные приложения.

Данная среда разработки имеет обширный список доступных ассетов, большую библиотеку обучающих материалов и курсов, что делает изучение программы еще проще. Присутствует возможность писать код не только на GDScript, но и на C#. В движке реализован интуитивно понятный дизайн, который помогает понять иерархию создания объектов и логику их взаимодействия друг с другом.

Среда разработки содержит большое количество объектов различных типов с их компонентами и определенной направленностью использования для разрабатываемого приложения, при этом создание происходит модульно – вся программа разбита на множество отдельных модулей, которые можно использовать друг с другом, создавая определенную иерархию между ними, что делает намного легче процесс моделирования и структуризации программы.

По данным «The Global Game Jam 2024» – Godot Engine занимает третье место по количеству разработанных игр в этом году, что в общем составляет примерно 7% от всего количества реализованных проектов. На данный момент, количество участников данного события насчитывает около 35 тысяч человек на 796 электронных ресурсах в 102 странах. Благодаря приведенным данным, можно сделать вывод, что Godot Engine является достаточно популярным движком для разработки в условиях временного ограничения в связи с простотой в освоении и скоростью разработки программного продукта.

Вывод по первой главе

В текущей главе был проведен анализ предметной области и обзор существующих аналогов, в ходе которого были выделены основные необходимые аспекты разрабатываемого приложения. Редактор должен иметь интуитивно понятный интерфейс, возможность создавать локации, которые можно будет использовать, загружать, сохранять и редактировать, возможность создания событий для реализации большей интерактивности с игроками.

2. ПРОЕКТИРОВАНИЕ

2.1. Требования

После анализа предметной области и обзора существующих решений были сформированы функциональные и нефункциональные требования к разрабатываемому приложению.

Функциональные требования

Редактор локаций должен удовлетворять следующим фунциональным требованиям.

1. Возможность создать, сохранить и загрузить созданные локации.
2. Генерация локаций согласно указанным размерам.
3. Выбор и размещение предметов на локации.
4. Создание событий.
5. Запуск созданной локации.
6. Возможность создать, сохранить и загрузить созданную карту локаций.
7. Генерация карты, согласно заданному количеству локаций.
8. Запуск созданной карты.

Нефункциональные требования

Для разрабатываемого приложения также были выделены следующие нефункциональные требования.

1. Приложение должно быть разработано с использованием языка GDScript на платформе Godot Engine.
2. Приложение должно корректно работать на операционной системе Windows 10.

2.2. Диаграмма вариантов использования режима создания локаций

Сформирована диаграмма вариантов использования режима создания локаций для разрабатываемого редактора локаций.

Игрок – главный актер, у которого есть возможность создать новую или загрузить уже созданную ранее карту.

После загрузки или создания локации появляется возможность создавать события и размещать объекты. Выделив объект или событие, можно его настроить через контекстное меню.

Также присутствуют такие действия как сохранение созданной или отредактированной локации и тестирование – загрузка уровня как игрового. Диаграмма вариантов использования режима создания локаций для редактора локаций представлена на рисунке 7.

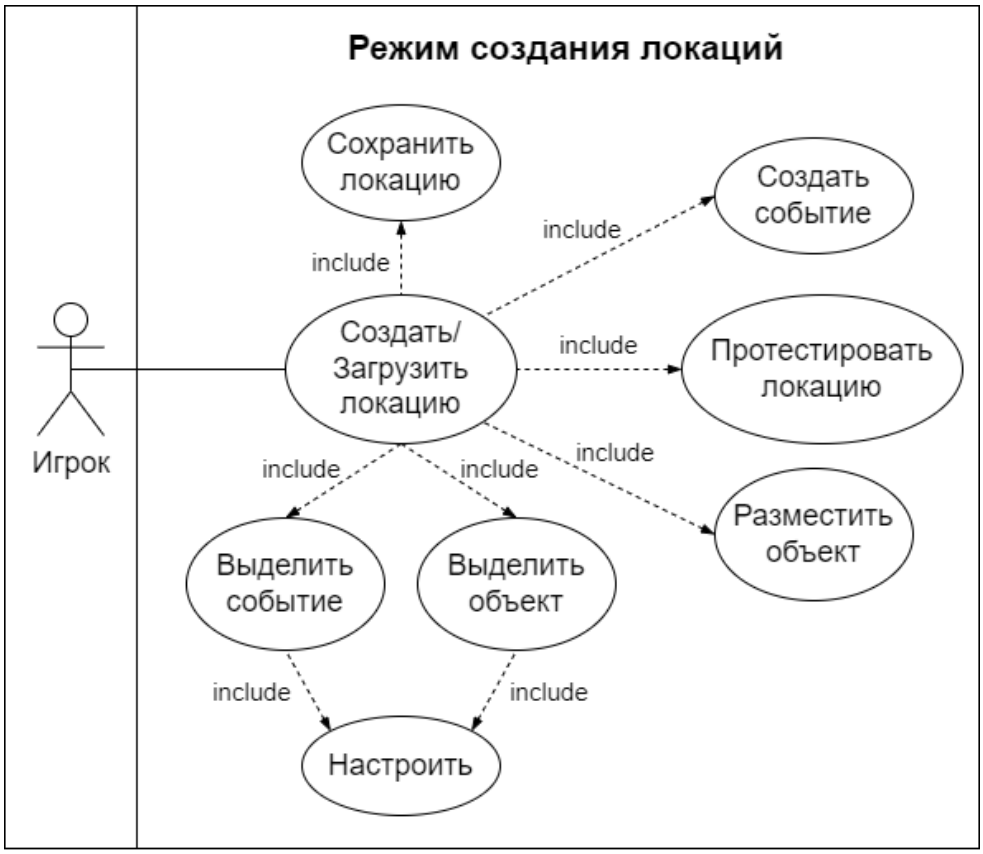

Рисунок 7 – Диаграмма вариантов использования режима создания локаций

2.3. Интерфейс режима создания локаций

Интерфейс режима создания локаций должен содержать основной набор стандартных фукнций с файлами и списки объектов. Приложение будет разделено на две группы панелей и основное окно редактора. Панели делятся на панель инструментов и панель настроек инструментов.

Верхняя панель содержит функции работы с файлами: создание, загрузка, сохранение; функции вида, такие как отображение коллизий; функции редактирования, и помощь, где будут описаны инструменты редактора и как ими пользоваться. Также ниже находится панель инструментов, на которой будут расположены главные инструменты программы. Макет верхней панели представлен на рисунке 8.

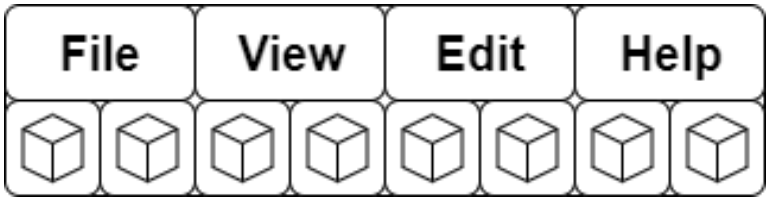

Рисунок 8 – Макет верхней панели режима создания локаций

Боковая панель инструментов (рисунок 9) содержит вкладки объектов, их настройки, настройки логики, кнопку помощи. По центру располагается панель с информацией. В самом низу расположена панель вида, которая задает настройки для предпросмотра объекта.

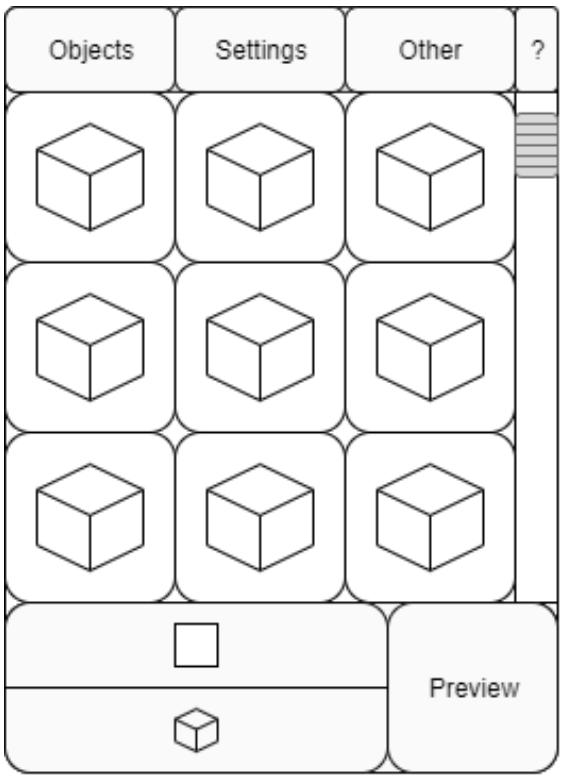

Рисунок 9 – Боковая панель инструментов режима создания локаций

2.4. Диаграмма вариантов использования режима создания карт

После режима создания локаций была сформирована диаграмма вариантов использования режима создания карт (рисунок 10).

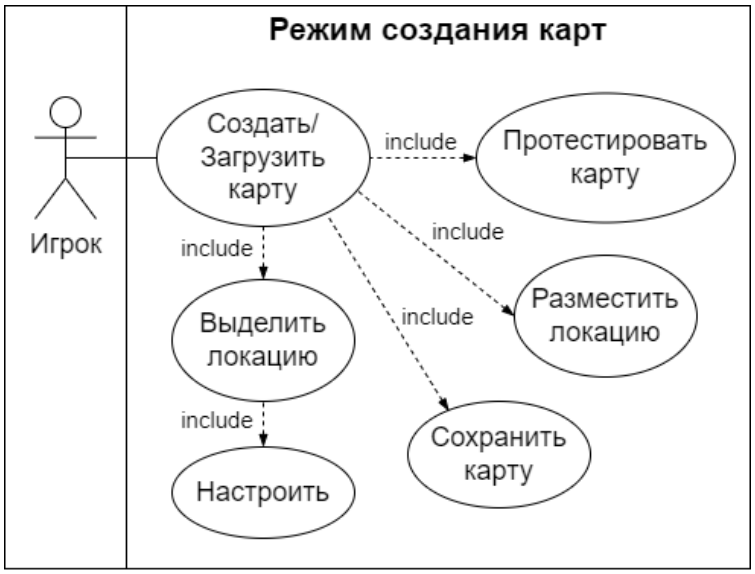

Рисунок 10 – Диаграмма вариантов использования режима создания карт

Игрок в режиме создания карт имеет тот же набор функций, что и в режиме создания локаций, но все они направлены именно на создание уже не совокупности объектов и событий, а совокупности локаций и событий.

Также данный режим имеет отличную от предыдущего режима боковую панель (рисунок 11).

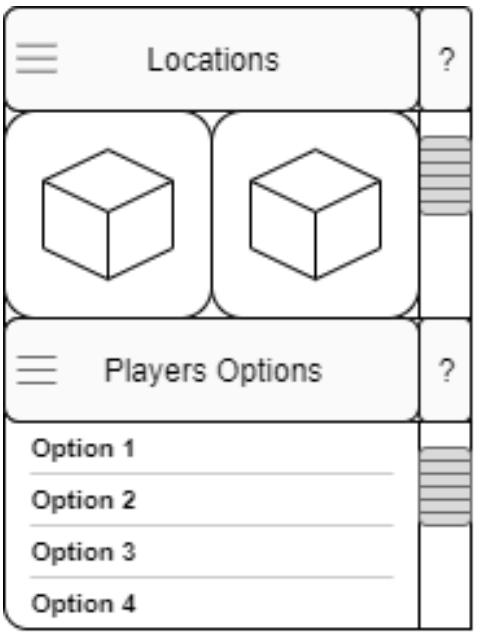

Рисунок 11 – Боковая панель инструментов режима создания карт

Данная боковая панель состоит из двух отдельных панелей, которые можно свободно перемещать по экрану редактора, чтобы они не мешали при создании карты. Первая панель содержит информацию о локациях, созданных игроком и тех, что были в игре изначально. Вторая панель – настройки игроков, она нужна для того, чтобы выставить точки появления игроков, отредактировать стартовые предметы и параметры персонажей игроков, а также установить процентное соотношение этих параметров, для точного контроля сложности игрового процесса на разрабатываемой карте.

Ниже представлено окно настройки соединения локаций (рисунок 12). Левая часть окна нужна для проверки и соединения локаций: сверху показывается весь список мест на карте с краткой информацией о них, ниже информация о соединениях, последний раздел нужен для настройки условий. Правая часть предназначена для детальной настройки наполненности выбранной местности.

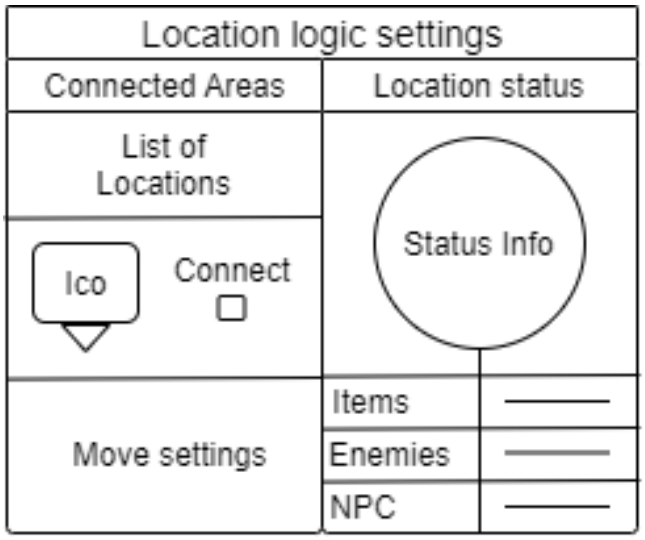

Рисунок 12 – Окно настройки логики соединений выбранной локации

Вывод по второй главе

В данной главе были построены диаграммы вариантов использования модулей редактора локаций и редактора карт, сформированы макеты визуальных компонентов интерфейса и выявлены фунциональные и нефункциональные требования для разработки описанных модулей.

3. РЕАЛИЗАЦИЯ

3.1. Создание персонажа игрока

Для полноценной работоспособности главного персонажа необходимо было реализовать следующие функции: система инвентарей, движения с коллизией и интерфейс.

Изначально был создан скрипт персонажа игрока, для движений и коллизий, также внутри были выставлены значения лимитов камеры в зависимости от размеров локации, на которой находится игрок (листинг 1).

Листинг 1 – Скрипт игрока

```
extends KinematicBody2D
onready var ui = get_viewport().get_node("Game/UI/Control") #ссылка на
интерфейс
export(int) var speed = 180.0
var items = 0
var inventory = {}
func _physics_process(delta):
    var velocity = Vector2.ZERO
    if Input.is_action_pressed("right_walk"):
       velocity.x += 1.0
    if Input.is_action_pressed("left_walk"):
        velocity.x -= 1.0
    if Input.is_action_pressed("down_walk"):
        velocity.y += 1.0
    if Input.is_action_pressed("up_walk"):
        velocity.y -= 1.0
    velocity = velocity.normalized()
    if velocity == Vector2.ZERO:
        $AnimationTree.get("parameters/playback").travel("Idle")
    else:
        $AnimationTree.get("parameters/playback").travel("Walk")
        $AnimationTree.set("parameters/Idle/blend_position", velocity)
        $AnimationTree.set("parameters/Walk/blend_position", velocity)
        move_and_slide(velocity * speed)
position.x = clamp(position.x, 10, $Camera2D.limit_right-10)
position.y = clamp(position.y, 10, $Camera2D.limit_bottom-10)
```

Как видно из листинга, здесь подается пользовательский интерфейс на экран, задается параметр скорости игрока и инициализируется инвентарь, затем анализируются нажатия для использования правильной анимации и изменения скорости. В конце задаются границы камеры [3].

Анимация игрока состоит из дерева анимации и непосредственного воспроизводителя анимации. Дерево анимации содержит возможные состояния игрока, которые в данном примере делятся на состояния «Idle» и «Walk». Воспроизводитель представляет из себя временную ленту, где для

каждого возможного направления перемещения игрока задается отдельное состояние внутри воспроизводителя, которое будет менять спрайты в зависимости от соблюдения условий смены этих состояний внутри скрипта перемещения игрока. Набор спрайтов для анимации игрока представлен на рисунке 13.

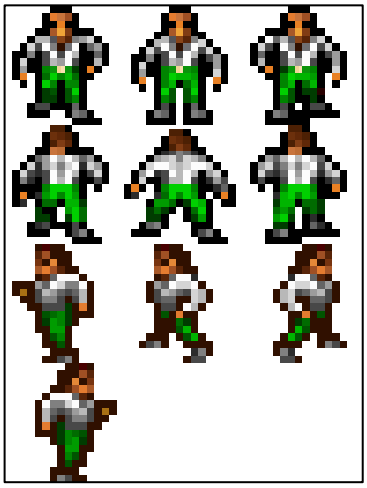

Рисунок 13 – Набор спрайтов для анимации игрока

Для инвентаря используется ряд скриптов: «Inventory.gd», «Item.gd», «PlayerInventory.gd», «Slot.gd». В приложении А продемонстрирован код «Inventory.gd», так как он содержит саму логику перемещения, объединения и регистрации предметов.

Так как описывается компонент игрока, содержание скрипта «PlayerInventory.gd» будет представлено в листинге 2.

Листинг 2 – Скрипт инвентаря игрока

```
extends Node
const NUM_INVENTORY_SLOTS = 12
var inventory = {
    0: ["axe", 1],
    1: ["bugs", 2],
    2: ["bugs", 1],
    3: ["bugs", 2]
}
func add_item(item_name, item_quantity):
    for item in inventory:
        if inventory[item][0] == item_name:
            var stack_size = int(JsoNdata.item_data[item_name]["StackSize"])
            var able_to_add = stack_size - inventory[item][1]
            if able_to_add >= item_quantity:
                inventory[item][1] += item_quantity
                return
            else:
```

```
            inventory[item][1] += able_to_add
            item_quantity = item_quantity - able_to_add
    # item doesn't exist in inventory yet, so add it to an empty slot
    for i in range(NUM_INVENTORY_SLOTS):
        if inventory.has(i) == false:
            inventory[i] = [item_name, item_quantity]
            return
```

Содержимое инвентаря задается при создании игрока. Процесс добавления нового предмета исходит из системы группировки предметов, если подобранный предмет еще не находился в инвентаре, то он занимает свободный слот, если такой существует. Если премдет уже был в инвентаре, то проверяется наполненность существующей группы предметов и только тогда добавляется в группу этих предметов либо в свобоный слот. В случае отсутствия свободных слотов предмет не подбирается.

Затем был сформирован сам интерфейс для инвентаря (рисунок 14). Структурно инвентарь состоит из сеточного контейнера, наполненного необходимым набором слотов, каждый из которых содержит скрипт «Slot.gd», реализующий логику работы родительской ячейки.

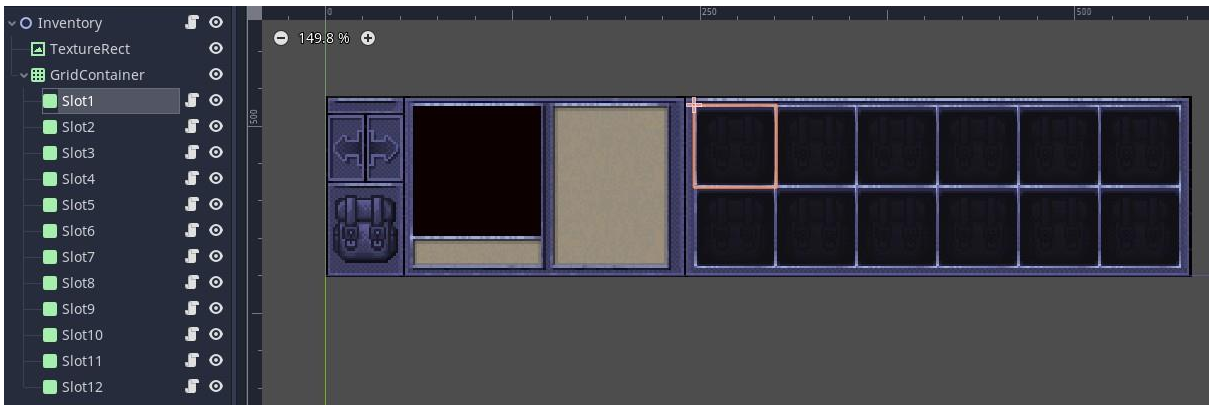

Рисунок 14 – Иерархия компонентов и графическая составляющая инвентаря

Для отображения инвентаря используются переключение видимости и инициализация (листинг 3).

Листинг 3 – Код, инициализирующий видимое окно инвентаря

```
func _input(event)
    if event.is_action_pressed("inventory"):
        $Inventory.visible = !$Inventory.visible
        $Inventory.initialize_inventory()
```

```
func initialize_inventory():
    var slots = inventory_slots.get_children()
    for i in range(slots.size()):
        if PlayerInventory.inventory.has(i):
            slots[i].initialize_item(PlayerInventory.inventory[i][0], Play-
erInventory.inventory[i][1])
```

3.2. Реализация базовых функций редактора

Данный модуль приложения содержит конвертер «.png» файлов в набор тайлов, которые позже можно размещать в окне редактора. Также реализованы функции импорта и экспорта «.json» файлов реализуемых локаций.

В качестве исходных данных для редактора принимаются наборы тайлсетов, разделенные по уникальности согласно типу местности и наличию коллизий на них. Наборы заранее изменяются согласно требованиям редактора: описываются коллизии, указываются тайлы, которые будут интерактивными и на какие внутриигровые объекты они будут ссылаться при создании, остальные тайлы определяются на декоративный слой. На рисунке 15 представлен фрагмент тайлсета пустынных сооружений и декоративных объектов.

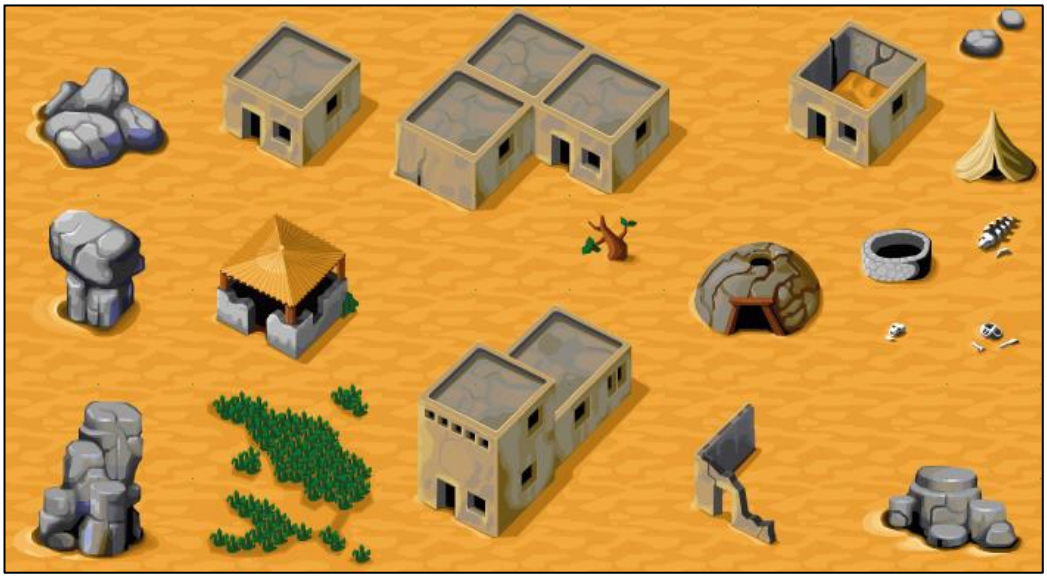

Рисунок 15 – Тайлсет пустынной местности

В скрипте модуля прописана логика управления камерой внутри редактора, все события нажатия существующих кнопок редактора, парсинг и

логика размещения объектов на поле редактирования локации. Фрагмент кода из скрипта «locEditor.gd», отвечающий за открытие файлового диалога для загрузки существующего файла представлен в листинге 4.

Листинг 4 – Фрагмент скрипта «locEditor.gd» для загрузки файлов

```
func _on_FileDialogImportJson_file_selected(path: String) -> void:
    var file := File.new()
    file.open(path, File.READ)
    var text := file.get_as_text()
    var json_result := JSON.parse(text)
    if json_result.error == OK:
        for p in json_result.result["layers"]:
            var layer := _create_layer(p["texture_path"])
            var tm := layer.get_meta("tm") as TileMap
            for c in p["cells"]:
                tm.set_cell(c[1], c[2], c[0])
        for p in json_result.result["objects"]:
            _create_object(p)
    file.close()
```

В коде видно, что «.json» файл разбивается на слои и объекты в данных слоях (элементы заполнения), с помощью описанных в этом же скрипте функций `_create_layer()` и `_create_object()` объединяется содержимое «.json» файла в объект, который можно будет редактировать. Структура «.json» файлов представлена на рисунке 16. Коды создания слоев сохранения файлов и объектов представлены в листингах 2–4 приложения Б.

Рисунок 16 – Структура «.json» файлов локаций

В редакторе также были реализованы виды кистей, для более удобной отрисовки тайлмапа локации. Всего форм кистей две, переключаются формы с помощью кнопок в окне редактора в разделе «Brush Shape», которые вызывают события `_on_ButtonCircle_toggled` или `_on_Button Square_toggled` при нажатии на них. События переключают состояние кисти и вызывают функцию смены формы курсора. Смена же формы курсора происходит в функции `_set_new_cursor_shape()` в скрипте редактора «locEditor.gd». Кроме параметра формы – `_current_shape`, кисть содержит также параметр размера – `_current_size`, устанавливаемых с помощью слайдера в окне редактора, взаимодействие с которым вызывает функцию `_on_HSliderSize_value_changed`. Размер является основным коэффициентом при вычислении формулы формы кисти. На данный момент в редакторе реализовано два вида кистей: круг и квадрат. Функция `_set_new_cursor_shape()` представлена в листинге 5.

Листинг 5 – Функция переключения формы кисти скрипта «locEditor.gd»

```
func _set_new_cursor_shape() -> void:
    for i in _cursor_container.get_children():
        i.queue_free()
    _cursors.clear()

    match _current_shape:
        BrushShape.CIRCLE:
            var half_size := int(floor(_current_size * 0.5))
            for x in range(-half_size, half_size + 1):
                for y in range(-half_size, half_size + 1):
                    if pow(x, 2) + pow(y, 2) <= pow(half_size, 2):
                        var c: Sprite = CURSOR.instance()
                        c.offset_pos = 16 * 4 * Vector2(x, y)
                        _cursor_container.add_child(c)
                        _cursors.push_back(c)
        BrushShape.SQUARE:
            var half_size := int(floor(_current_size * 0.5))
            for x in range(-half_size, half_size + 1):
                for y in range(-half_size, half_size + 1):
                    var c: Sprite = CURSOR.instance()
                    c.offset_pos = 16 * 4 * Vector2(x, y)
                    _cursor_container.add_child(c)
                    _cursors.push_back(c)

        _:
            Pass
```

Для использования редактора и навигации по полю локации используется функция обработки событий нажатия определенных клавиш. Так, для

перемещения по локации используется средняя кнопка мыши (drag в коде), для переключения видимости интерфейса редактора используется «tab» (toggle_ui), использование колесика мыши позволяет масштабировать камеру в редакторе, либо, если зажать «shift», появляется возможность менять размеры кисти, для отрисовки тайлов. В той же функции обработки событий реализованы события нажатия правой и левой кнопки мыши, которые удаляют и размещают объекты по нажатию, соответсвенно. Выше описанный код продемонстрирован в листинге 6.

Листинг 6 – Функция «`_unhandled_input`» скрипта «locEditor.gd»

```
func _unhandled_input(event: InputEvent) -> void:
    if Input.is_action_just_pressed("drag"):
        _initial_drag_pos = get_global_mouse_position()
    if Input.is_action_just_pressed("toggle_gui"):
        _cl.visible = !_cl.visible
    var shift_pressed := Input.is_action_pressed("shift")
    if Input.is_action_just_pressed("zoom_in"):
        if shift_pressed:
            _hslider_size.value = clamp(_current_size + 1, 1, 8)
        else:
            _camera.zoom = Vector2.ONE * clamp(_camera.zoom.x - 0.05, 0.01,
4.0)
    if Input.is_action_just_pressed("zoom_out"):
        if shift_pressed:
            _hslider_size.value = clamp(_current_size - 1, 1, 8)
        else:
            _camera.zoom = Vector2.ONE * clamp(_camera.zoom.x + 0.05, 0.01,
4.0)
    var mouse_pos = get_global_mouse_position() / 64
    var place_pos := Vector2(floor(mouse_pos.x), floor(mouse_pos.y))
    for c in _cursors:
        c.position = place_pos * 64 + Vector2.ONE * 32 + c.offset_pos
    _lbl_position.text = "%s\n%d" % [place_pos, _selected_tile]
    if _current_layer == -1: return
    if Input.is_action_pressed("place"):
        for i in _cursors:

_layer_container.get_child(_current_layer).get_meta("tm").set_cellv(
                Vector2(floor(i.position.x / 64), floor(i.position.y /
64)),
                _selected_tile)
    if Input.is_action_pressed("delete"):
        for i in _cursors:

_layer_container.get_child(_current_layer).get_meta("tm").set_cellv(
                Vector2(floor(i.position.x / 64), floor(i.position.y /
64)),
                -1)
```

Для настройки объектов были прописаны несколько функций: `_on_Object_text_changed` – меняющая название объекта, отображаемого

на локации, _on_Object_deleted – очищающая очередь от выбранного для удаления объекта в редакторе, _on_Object_pos_x_changed и _on_Object_pos_y_changed – позволяющие управлять положением координат объекта на локации и функция _update_obj, которая применяет изменения внесенные с помощью предыдущих функций. Код данных функций продемонстрирован в листинге 7, в том же порядке следования, что был описан выше.

Листинг 7 – Функции работы с объектами скрипта «locEditor.gd»

```
func _on_Object_text_changed(text: String, obj_item: Control) -> void:
    _update_obj(obj_item)

func _on_Object_deleted(obj_item: Control) -> void:
    obj_item.get_meta("obj").queue_free()
    obj_item.queue_free()

func _on_Object_pos_x_changed(text: String, obj_item: Control) -> void:
    _update_obj(obj_item)

func _on_Object_pos_y_changed(text: String, obj_item: Control) -> void:
    _update_obj(obj_item)
func _update_obj(obj_item: Control) -> void:
    var obj := obj_item.get_meta("obj") as Node2D
    obj.get_node("Label").text =
obj_item.get_node("MarginContainer/VBoxContainer/HBoxContainer/Key").text
    obj.position = Vector2(
        64 *
int(obj_item.get_node("MarginContainer/VBoxContainer/HBoxContainer2/PosX").
text),
    64 *
int(obj_item.get_node("MarginContainer/VBoxContainer/HBoxContainer2/PosY").
text))
```

Функции обращаются к объекту по текстовому идентификатору и отправляют объект класса «Control» в основную функцию контроля состояния объекта для его обновления, формируемый объект занимает клетку равную размеру тайла.

3.3. Реализация противников

Для реализации противников был создан класс Enemy для которого использовался компонент «KinematicBody2d». Код основных функций ИИ

противника представлен в листинге 8. Для переключения состояний используется enum с состояниями: SURROUND, ATTACK, HIT. Состояние перемещения устанавливается как изначальное.

Листинг 8 – Код основных функций реализации ИИ противника

```
func _ready():
    var rng = RandomNumberGenerator.new()
    rng.randomize()
    randomnum = rng.randf()
func _physics_process(delta):
    match state:
        SURROUND:
            move(get_circle_position(randomnum), delta)
        ATTACK:
            move(player.global_position, delta)
        HIT:
            move(player.global_position, delta)
            print("HIT")
func move(target, delta):
    var direction = (target - global_position).normalized()
    var desired_velocity =  direction * SPEED
    var steering = (desired_velocity - velocity) * delta * 2.5
    velocity += steering
    velocity = move_and_slide(velocity)
func get_circle_position(random):
    var kill_circle_centre = player.global_position
    var radius = 40
    var angle = random * PI * 2;
    var x = kill_circle_centre.x + cos(angle) * radius;
    var y = kill_circle_centre.y + sin(angle) * radius;
    return Vector2(x, y)
func _on_AttackTimer_timeout():
    state = ATTACK
```

Вывод по третьей главе

В данной главе были представлены листинги скриптов редактора локаций и части модулей объектов, созданных в процессе реализации. Были реализованы основные компоненты редакторов, затем были созданы сами редакторы с описанными в функциональных требованиях функциями. Модули редакторов были внедрены в основную игру.

4. ТЕСТИРОВАНИЕ

Функциональное тестирование было проведено на ноутбуке MSI GF63 Thin 9RCX с операционной системой Windows 10. Результаты тестирований приведены в таблице 1.

Таблица 1 – Результаты тестирования редактора локаций

№	Название теста	Шаги	Ожидаемый результат	Тест пройден?
1	Запустить редактор.	Через главное меню игры нажать на кнопку «Editor».	Запуск редактора карт.	Да
2	Создать новую карту.	1. Нажать на верхней панели на вкладку «File». 2. В выпавшем контекстном меню выбрать функцию «Create new map».	Была создана новая карта, все вкладки открылись без проблем.	Да
3	Изучить интерфейс.	1. Нажать на остальные вкладки на верхней панели. 2. Нажать на любую иконку под вкладками, чтобы открыть боковую панель. 3. Переключиться между вкладками на боковой панели.	Все функции верхней панели редактора работают правильно, боковая панель открывается корректно, происходит переключение вкладок боковой панели и содержание различается в зависимости от выбранного типа объектов.	Да
4	Использовать интерфейс.	1. Выбрать любой объект боковой вкладки и настроить его. 2. Проверить работоспособность окна предпросмотра. 3. Разместить локацию на карте.	Выбранный объект был настроен и корректно отображен в окне предпросмотра. Объект корректно выставлен на карту.	Да
5	Открыть новый тайлсет и проверить его работоспособность.	1. Нажать на кнопку «New Layer». 2. Выбрать .png файл с тайлсетом. 3. Убедиться, что тайлсет загрузился и отобразился корректно. 4. Разместить любое количество элементов на локации.	Выбранный тайлсет был успешно загружен. Отображение объектов в меню выбора объектов было корректно. Объекты были корректно размещены на локации.	Да
6	Сохранить проект.	1. Сохранить проект через вкладку «File» - «Save current map»/«Save as…». 2. Дать название сохраняемой карте.	Карта была успешно сохранена.	Да

Окончание таблицы 1

№	Название теста	Шаги	Ожидаемый результат	Тест пройден?
7	Загрузить созданный проект.	1. Загрузить сохраненный ранее файл локации с помощью кнопки «Load Location». 2. Выбрать нужный файл в открывшемся диалоговом окне. 3. Проверить, что файл загрузился корректно.	Созданная ранее локация была успешно загружена в редакторе.	Да
8	Использовать кисти и функции навигации.	1. Изменить настройки кисти. 2. Использовать горячие клавиши «tab» и «shift» при использовании редактора.	Настройки кисти были успешно изменены и проверены. Все горячие клавиши и клавиши для управления процессом редактирования работают корректно.	Да
9	Проверить проект.	Нажать на кнопку «Test current map» во вкладке «File».	Карта была успешно запущена, все элементы работают корректно.	Да
10	Перезапустить проект.	Выйти из редактора и повторить действия.	Редактор перезапущен, после всех действий все работает корректно.	Да

Вывод по четвертой главе

В рамках данной главы было проведено функциональное тестирование редактора локаций. В ходе тестирования все функциональные тесты были пройдены успешно, все проблемы и ошибки, возникшие в ходе проверок были успешно исправлены.

ЗАКЛЮЧЕНИЕ

В рамках данной работы был разработан редактор локаций для компьютерной игры в жанре Survival RPG на платформе Godot Engine.

Были решены следующие задачи:

1) выполнен анализ предметной области;
2) спроектировано приложение;
3) реализованы модули персонажа и инвентаря;
4) реализованы базовые функции для работы с редактором;
5) реализованы функции сохранения и загрузки локаций;
6) реализовано внедрение редактора в основную игру;
7) проведено тестирование приложения.

В ходе выполнения работы была изучена платформа Godot Engine и встроенный в нее язык GDScript, исследованы возможности создания и использования внутренних плагинов игрового движка. Полученный в ходе разработки редактора локаций опыт может быть использован для создания дальнейших проектов на данном игровом движке.

Направление дальнейших исследований

Дальнейшая работа над редактором будет направлена на увеличение функциональной составляющей приложения и добавление методов редактирования сценария. Также будет уделено внимание улучшению графической составляющей проекта.

ЛИТЕРАТУРА

1. Godot Docs – 3.4 branch. [Электронный ресурс] URL: https://docs.godotengine.org/en/stable (дата обращения: 08.01.2024 г.).

2. Лассер А., Лассер М., Райтер У. BurnTime – Manual / Лассер Альберт, Лассер Мартин, Райтер Уилфред. – Max Design, 1993. – 134 с.

3. GDSript Tutorials. [Электронный ресурс] URL: https://gdscript.com/tutorials (дата обращения: 13.01.2024 г.).

4. Ubisoft | Heroes of might and magic 3. [Электронный ресурс] URL: https://www.ubisoft.com/ru-ru/game/heroes-of-might-and-magic-3-hd (дата обращения: 12.01.2024 г.).

5. Titan quest | Дельфийский оракул. [Электронный ресурс] URL: http://titan-quest.net.ru (дата обращения: 12.01.2024 г.).

6. Godot Engine – ru-docs. [Электронный ресурс] URL: https://godot-ru.readthedocs.io (дата обращения: 13.01.2024 г.).

7. Godot Engine Tutorials. [Электронный ресурс] URL: https://www.davidepesce.com (дата обращения: 13.01.2024 г.).

8. Godot Engine: Use GridMap to create 3D TileMap. [Электронный ресурс] URL: https://www.programmersought.com/article/81814813001 (дата обращения: 11.01.2024 г.).

9. Procedural Generation in Godot. [Электронный ресурс] URL: https://kidscancode.org/blog/2018/12/godot3_procgen (дата обращения: 12.01.2024 г.).

10. Godot Engine Game Development in 24 Hours. [Электронный ресурс] URL: https://files.catbox.moe/fgn61m.pdf (дата обращения: 13.01.2024 г.).

11. Top Game Engines. Hady ElHady | Blog & Resources. [Электронный ресурс] URL: https://instabug.com/blog/game-engines/ (дата обращения: 05.01.2024 г.).

12. Glass BD, Maddox WT, Love BC. Real-Time Strategy Game Training: Emergence of a Cognitive Flexibility Trait | Plos One. [Электронный ресурс] URL: https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0070350/ (дата обращения: 05.01.2024 г.).

13. Bradfield C. Godot Engine Game Development Projects. – Packt Publishing, 2018. – 298 с.

14. Create Your Own Level Editor. [Электронный ресурс] – URL: https://www.kodeco.com/2732-create-your-own-level-editor-part-1-3/ (дата обращения: 18.01.2024 г.).

15. Make Your Life Easier: Build A Level Editor. [Электронный ресурс] – URL: https://gamedevelopment.tutsplus.com/articles/make-your-life-easier-build-a-level-editor--gamedev-356/ (дата обращения: 15.01.2024 г.).

ПРИЛОЖЕНИЯ

Приложение А. Листинг скрипта «Inventory.gd»

Листинг 1 – Скрипт «Inventory.gd»

```
extends Node2D
const SlotClass = preload("res://UI_tries_Inv/Slot.gd")
onready var inventory_slots = $GridContainer
var holding_item = null
func _ready():
    for inv_slot in inventory_slots.get_children():
        inv_slot.connect("gui_input", self, "slot_gui_input", [inv_slot])
    initialize_inventory()
func initialize_inventory():
    var slots = inventory_slots.get_children()
    for i in range(slots.size()):
        if PlayerInventory.inventory.has(i):
          slots[i].initialize_item(PlayerInventory.inventory[i][0], Player-
Inventory.inventory[i][1])
func slot_gui_input(event: InputEvent, slot: SlotClass):
    if event is InputEventMouseButton:
        if event.button_index == BUTTON_LEFT && event.pressed:
          # Currently holding an Item
          if holding_item != null:
            # Empty slot
            if !slot.item:
              slot.putIntoSlot(holding_item)
              holding_item = null
            # Slot already contains an item
            else:
              # Different item, so swap
              if holding_item.item_name != slot.item.item_name:
                var temp_item = slot.item
                slot.pickFromSlot()
                temp_item.global_position = event.global_position
                slot.putIntoSlot(holding_item)
                holding_item = temp_item
              # Same item, so try to merge
              else:
                var stack_size = int(JsoN-
data.item_data[slot.item.item_name]["StackSize"])
                var able_to_add = stack_size - slot.item.item_quantity
                if able_to_add >= holding_item.item_quantity:
                  slot.item.add_item_quantity(holding_item.item_quantity)
                  holding_item.queue_free()
                  holding_item = null
                else:
                  slot.item.add_item_quantity(able_to_add)
                  holding_item.decrease_item_quantity(able_to_add)
          # Not holding an item
          elif slot.item:
            holding_item = slot.item
            slot.pickFromSlot()
            holding_item.global_position = get_global_mouse_position()
func _input(event):
    if holding_item:
        holding_item.global_position = get_global_mouse_position()
```

Приложение Б. Листинги функций скрипта «locEditror.gd»

Листинг 2 – Функция создания слоя тайлов для редактирования

```
func _create_layer(path: String) -> Control:
    var layer := LAYER.instance() as Control
    layer.set_meta("path", path)
    var hbox := layer.get_node("MarginContainer/VBoxContainer/HBoxCon-
tainer")
    hbox.get_node("CheckBox").connect("toggled", self, "_on_Layer_toggled",
[layer])
    hbox.get_node("Name").text = path.get_file()
    hbox.get_node("ButtonUp").connect("pressed", self,
"_on_Layer_moved_up", [layer])
    hbox.get_node("ButtonDown").connect("pressed", self,
"_on_Layer_moved_down", [layer])
    hbox.get_node("ButtonDelete").connect("pressed", self, "_on_Layer_de-
leted", [layer])
    _layer_container.add_child(layer)
    _update_layers()
    var file := File.new()
    file.open(path, File.READ)
    var buffer := file.get_buffer(file.get_len())
    var img := Image.new()
    var data
    if path.ends_with(".png"):
        data = img.load_png_from_buffer(buffer)
    elif path.ends_with(".jpg") or path.ends_with(".jpeg"):
        data = img.load_jpg_from_buffer(buffer)
    var img_tex := ImageTexture.new()
    img_tex.create_from_image(img, 1 | 2)
    file.close()
    var tm := TileMap.new()
    tm.cell_size = Vector2(16, 16)
    var ts := TileSet.new()
    var idx := 0
    for x in range(img_tex.get_width() / 16):
        for y in range(img_tex.get_height() / 16):
            var at := AtlasTexture.new()
            at.atlas = img_tex
            at.region = Rect2(x * 16, y * 16, 16, 16)
            ts.create_tile(idx)
            ts.tile_set_texture(idx, at)
            idx += 1
    tm.tile_set = ts
    tm.scale = Vector2.ONE * 4
    _tm_container.add_child(tm)
    layer.set_meta("tex", img_tex)
    layer.set_meta("tex_path", path)
    layer.set_meta("tm", tm)
    hbox.get_node("CheckBox").emit_signal("toggled", true)
    return layer
```

Листинг 3 – Функция создания файлового диалога для сохранения файла

```
func _on_FileDialogExportJson_file_selected(path: String) -> void:
    var file := File.new()
    file.open(path, File.WRITE)
    var data := {
        "layers": [],
        "objects": {},
    }
```

```
    for idx in range(_layer_container.get_child_count()):
        var lay = _layer_container.get_child(idx)
        var tm := lay.get_meta("tm") as TileMap
        var a := []
        for c in tm.get_used_cells():
            a.push_back([tm.get_cellv(c), c.x, c.y])
        data["layers"].push_back(a)
    for obj_item in _objects_list.get_children():
        data["objects"][obj_item.get_node("MarginContainer/VBoxCon-
tainer/HBoxContainer/Key").text] = {
            "position": [
                int(obj_item.get_node("MarginContainer/VBoxContainer/HBox-
Container2/PosX").text),
                int(obj_item.get_node("MarginContainer/VBoxContainer/HBox-
Container2/PosY").text),
            ],
        }
    file.store_string(JSON.print(data))
    file.close()
```

Листинг 4 – Функция создания объектов

```
func _create_object(data := {}) -> void:
    var obj_item := OBJECT_ITEM.instance() as Control
    var vbox := obj_item.get_node("MarginContainer/VBoxContainer")
    var key := vbox.get_node("HBoxContainer/Key")
    var pos_x := vbox.get_node("HBoxContainer2/PosX")
    var pos_y := vbox.get_node("HBoxContainer2/PosY")
    if data:
        key.text = data["key"]
        pos_x.text = str(data["position"][0])
        pos_y.text = str(data["position"][1])
    key.connect("text_changed", self, "_on_Object_text_changed",
[obj_item])
    vbox.get_node("HBoxContainer/ButtonDelete").connect("pressed", self,
"_on_Object_deleted", [obj_item])
    pos_x.connect("text_changed", self, "_on_Object_pos_x_changed",
[obj_item])
    pos_y.connect("text_changed", self, "_on_Object_pos_y_changed",
[obj_item])
    _objects_list.add_child(obj_item)
    var obj := OBJECT.instance() as Node2D
    _objects_container.add_child(obj)
    obj_item.set_meta("obj", obj)
    _update_obj(obj_item)
```