**МИНОБРНАУКИ РОССИИ**
**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**
**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**
**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**
**Кафедра САПР**

# КУРСОВАЯ РАБОТА

**по дисциплине «Объектно-ориентированное программирование»**

**Тема: Разработка 2D игры на Unity**

Студенты гр. 3352 Гультяев А.С.
Портнов В.В.

Преподаватель _____ Кулагин М.В.

Санкт-Петербург
2024

**ЗАДАНИЕ**

**НА КУРСОВУЮ РАБОТУ**

Студенты: Гультяев А.С., Портнов В.В.

Группа 3352

Тема работы: Разработка 2D игры на Unity

Исходные данные:
C# 12 версии
Uinty 6000.0.30f1

Содержание пояснительной записки:
"Введение", "ТЕОРЕТИЧЕСКИЙ РАЗДЕЛ", "Игровые движки и выбор Unity", "Особенные компоненты", "Алгоритмы и взаимодействие объектов", "Создание графики для игры", "Разработка с использованием Unity", "ПРАКТИЧЕСКИЙ РАЗДЕЛ", "Реализация основных механик игры", "Реализация пользовательского интерфейса", "Диаграмма вариантов использования", "Диаграмма классов объектной модели предметной области", "Спецификация классов", "Код программы", "Заключение", "Ссылка на git"
Предполагаемый объем пояснительной записки:
Не менее 10 страниц.

Дата выдачи задания: 04.12.2024

Дата сдачи реферата: 25.12.2024

Дата защиты реферата: 26.12.2024

Студенты Гультяев А.С.
Портнов В.В.

Преподаватель Кулагин М.В.

## АННОТАЦИЯ

Курсовая работа посвящена созданию 2D-игры на платформе Unity. В рамках проекта разработаны основные игровые механики: управление персонажем (ходьба), сбор предметов (червячков) по заданию, а также пользовательский интерфейс, включающий главное меню с интерактивными кнопками. В процессе разработки использовались методы объектно-ориентированного программирования, сценарии на языке C# и инструменты Unity для создания анимаций и взаимодействия объектов.

Результатом работы является функционирующая игра с базовым набором функций, демонстрирующая основные этапы разработки 2D-игр. Полученные навыки могут быть применены для дальнейшего изучения и разработки более сложных игровых приложений.

## SUMMARY

This coursework focuses on the development of a 2D game using the Unity platform. The project implements core game mechanics, including character movement (walking), collecting items (worms) as part of a task, and a user interface featuring a main menu with interactive buttons. The development process involved object-oriented programming methods, C# scripting, and Unity tools for creating animations and object interactions.

The outcome of this work is a functional game with basic features, showcasing the key stages of 2D game development. The acquired skills can be applied to further study and the creation of more complex game applications.

**СОДЕРЖАНИЕ**


| | | |
|---|---|---|
| | Введение | 5 |
| 1. | ТЕОРЕТИЧЕСКИЙ РАЗДЕЛ | 6 |
| 1.1. | Игровые движки и выбор Unity | 6 |
| 1.2. | Особенные компоненты | 6 |
| 1.3. | Алгоритмы и взаимодействие объектов | 7 |
| 1.4. | Создание графики для игры | 7 |
| 1.5. | Разработка с использованием Unity | 7 |
| 2. | ПРАКТИЧЕСКИЙ РАЗДЕЛ | 8 |
| 2.1. | Реализация основных механик игры | 8 |
| 2.2. | Реализация пользовательского интерфейса | 8 |
| 2.3. | Диаграмма вариантов использования | 11 |
| 2.4. | Диаграмма классов объектной модели предметной области | 11 |
| 2.5. | Спецификация классов | 12 |
| 3. | Код программы | 17 |
| 4. | Заключение | 54 |
| 5. | Ссылка на git | 54 |

# ВВЕДЕНИЕ

Целью данной курсовой работы является создание простой 2D-игры на платформе Unity, которая включает базовые игровые механики: управление персонажем, сбор предметов по заданию, и интерфейс с главным меню. Этот проект позволяет изучить основные этапы разработки игр и закрепить навыки работы с инструментами Unity и языком программирования C#.

Для достижения поставленной цели необходимо решить следующие задачи:

1. Нарисовать все необходимые объекты для графического интерфейса и игры.
2. Разработать систему управления персонажем, позволяющую реализовать движение по игровому пространству.
3. Создать игровую механику сбора предметов (червячков) с учетом заданных условий.
4. Разработать пользовательский интерфейс, включающий главное меню и интерактивные кнопки.
5. Настроить игровой процесс и протестировать работу всех элементов игры.

Данный проект представляет собой не только образовательное задание, но и основу для дальнейшего изучения создания игр, а также возможность применения полученных знаний для реализации более сложных игровых приложений.

# 1. ТЕОРЕТИЧЕСКИЙ РАЗДЕЛ

## 1.1. Игровые движки и выбор Unity

Одним из самых популярных инструментов для разработки игр является игровой движок Unity. Unity представляет собой многоплатформенную среду для разработки, которая поддерживает создание как 2D, так и 3D игр. Он включает в себя мощные инструменты для работы с графикой, анимацией, физикой и звуком.

Для разработки 2D-игр Unity предлагает удобный редактор сцены, где можно работать с объектами, спрайтами, а также настраивать камеру и освещение. Unity использует C# как основной язык программирования, что позволяет создавать гибкие и эффективные игровые механики.

## 1.2. Основные компоненты игры

При разработке 2D-игры следует учитывать несколько ключевых компонентов:

- **Персонажи и их управление**. Одним из важнейших аспектов игры является система управления персонажем. В данном проекте используется механика ходьбы, которая реализуется через обработку ввода с клавиатуры и изменение позиции персонажа на игровом поле. Это позволяет создать ощущение динамичности и взаимодействия с игровым миром.
- **Сбор предметов**. В игре предусмотрена механика сбора предметов, в данном случае — червячков. Для этого используется система коллизий, при которой персонаж взаимодействует с объектами, попадающими в его область. Когда персонаж "приближается" к предмету, он может его собрать, что будет отмечено в игровом процессе.
- **Интерфейс и главное меню**. Пользовательский интерфейс играет важную роль в восприятии игры. В Unity существует возможность легко создавать кнопки, текстовые поля и другие элементы UI. Главное меню включает кнопки для начала игры, настроек и выхода. Эти элементы создаются с помощью компонентов Unity UI, которые обрабатывают события

взаимодействия с пользователем.

### 1.3. Алгоритмы и взаимодействие объектов

Для взаимодействия между объектами игры используется система коллизий. В Unity можно использовать как физические коллайдеры (для взаимодействий с использованием физики), так и триггеры, которые позволяют отслеживать столкновения объектов без применения физики. В данной игре используется простой подход, при котором персонаж собирает предметы, когда пересекает их область.

### 1.4. Создание графики для игры

Хоть Unity и предлагает большое количество готовых решений в виде кнопок, но все таки для создания уникального интерфейса был использован ряд других приложений: Aseprite и PikoPixel. Данные приложения предлагают удобный и понятный интерфейс для создания пиксель-артов, что и необходимо для реализации простой 2D игры.

### 1.5. Разработка с использованием Unity

Процесс разработки игры в Unity включает несколько этапов:

- **Проектирование уровней и объектов**. На этом этапе создаются сцены, размещаются объекты (персонажи, предметы, фон), а также настраиваются их параметры и поведение.
- **Программирование логики игры**. С помощью C# создаются скрипты для управления персонажем, взаимодействия с объектами и обработки пользовательского ввода.
- **Тестирование и отладка**. После реализации основных механик важно протестировать игру, выявить ошибки и оптимизировать работу. Это включает тестирование управления персонажем, правильность сбора предметов и корВ ходе выполнения курсовой работы была разработана 2D-игра на платформе Unity, включающая базовые игровые механики:

управление персонажем, сбор предметов (червячков) по заданию и пользовательский интерфейс с главным меню и интерактивными кнопками.

# 2. ПРАКТИЧЕСКИЙ РАЗДЕЛ

## 2.1. Реализация основных механик игры

Одним из самых популярных инструментов для разработки игр является игровой движок Unity. Unity представляет собой многоплатформенную среду для разработки, которая поддерживает создание как 2D, так и 3D игр. Он включает в себя мощные инструменты для работы с графикой, анимацией, физикой и звуком.

Для разработки 2D-игр Unity предлагает удобный редактор сцены, где можно работать с объектами, спрайтами, а также настраивать камеру и освещение. Unity использует C# как основной язык программирования, что позволяет создавать гибкие и эффективные игровые механики.

## 2.2. Реализация пользовательского интерфейса

Как было сказано ранее, для реализации пользовательского интерфейса использовались такие программы как PikoPixel и Aseprite, также, для программной реализации кнопок и текста использовались встроенные в Unity интефрейсы кнопок и текста (Button и TMP_Text).

При разработке 2D-игры следует учитывать несколько ключевых компонентов:

1) Начальное окно. Данное окно открывается при запуске игры, оно должно содержать минимальные параметры: начало игры и выход из игры. Реализация начального окна приведена на изображении 1.

2) Карта и окружение. Для реализации окружения и карты были использована стилистика леса и тропы из каменных плит. В виде окружения используются деревья, которые равномерно расположены по всей карте. Итоговая карта приведена на рис. 2.

3) Персонаж и NPC. Персонаж выбран все в той же стилистике 2D игры размерами 32 на 32 пикселя. Также добавлен диалог между NPC и персонажем для принятия квеста. Как и все вышесказанное реализован он был в специальной программе. Изображение персонажа и NPC приведены на рис. 3.

4) Окна состояния персонажа. Для состояния персонажа были использованы два параметра — здоровье и энергия. Энергия используется персонажем для ускорения при нажатии на клавишу shift и имеет некоторый заряд, который тратится по мере зажатой клавиши, и точно также восстанавливается по мере того, как клавиша не нажата. Эти окна, вместе с персонажем показаны на изображении 4.

5) NPC для задания. В нашем случае NPC для задания — маленькие червячки, которых пользователь должен собрать для выполнения квеста. Изображения червяков и пример их сбора приведен на рис. 5.


Рис. 1. Начальное окно игры

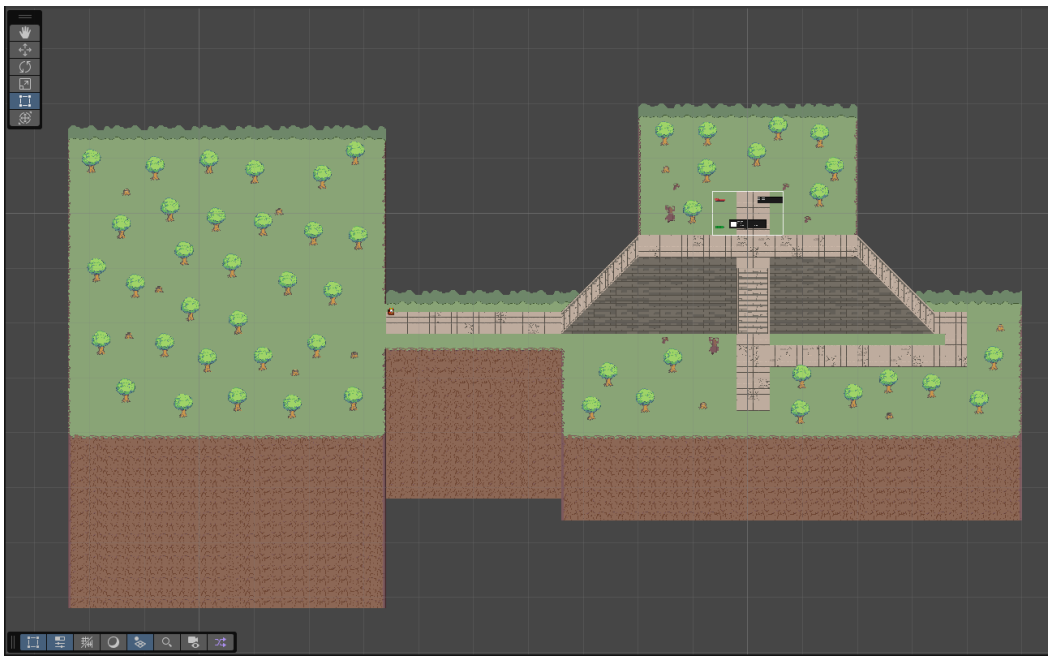

Рис. 2. Карта игры.

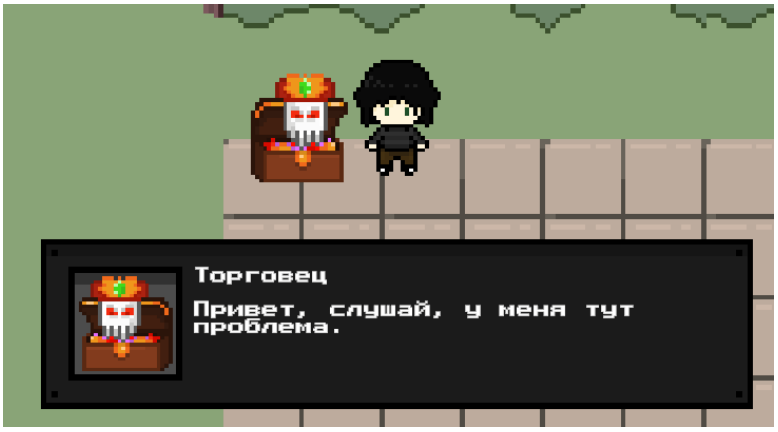


Рис. 3. Персонаж, NPC и диалог между ними

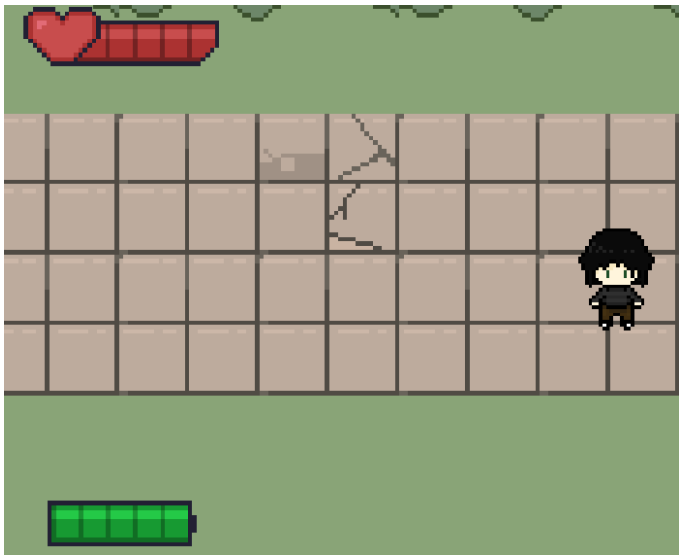

Рис. 4. Окна состояния персонажа

Рис. 5. Червяки NPC

### 2.3. Диаграма вариантов использования

Диаграмма вариантов использования включает в себя работу с внешним миром, а именно — работу с пользователем. В контексте двумерной игры на Unity примерами такого взаимодействия могут быть: взаимодействие пользователя с кнопками, а именно — запуск игры и выход, передвижение персонажа по карте, и другое. Полный список вариантов использования приведен на рисунке 6.

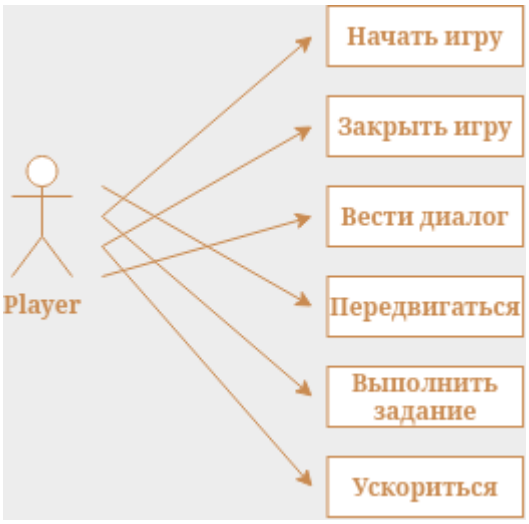


Рис. 6. Пример диаграммы вариантов использования системы

### 2.4. Диаграмма классов объектной модели предметной области

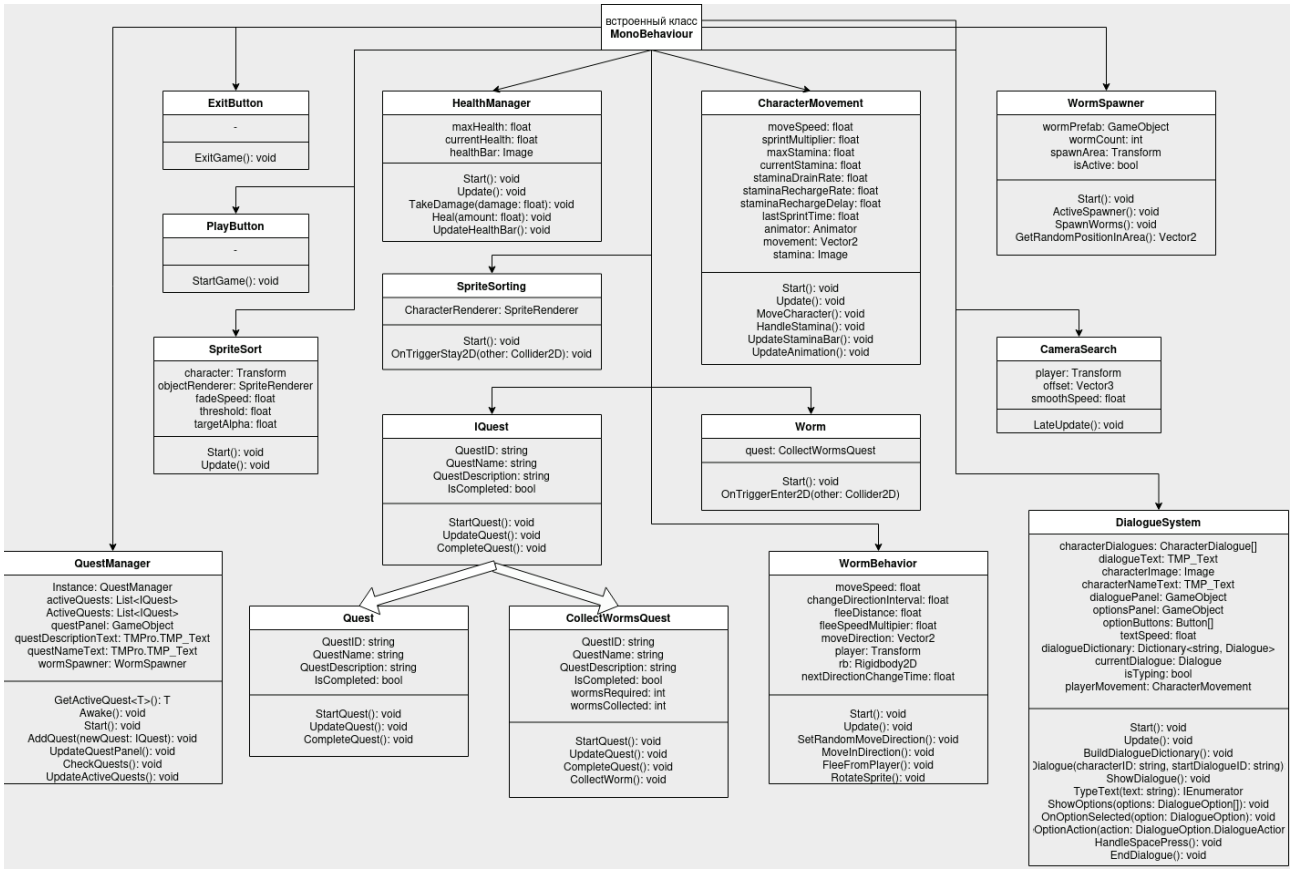


Рис. 7. UML диаграмма классов.

Основным элементом диаграммы классов является класс, которые представляется в виде прямоугольником, состоящим из трех секций: имя класса, атрибуты (свойства, поля) и методы. Атрибуты (свойства, поля) помимо названия могут содержать квалификатор видимости и тип. Методы также могут содержать квалификатор видимости, а также сигнатуру, определяющую тип возвращаемого значения, имена и типы параметров. На данном изображении белой стрелкой изображено наследование, черными стрелочками наследование от встроенного класса.

## 2.5. Спецификация классов

Спецификация классов содержит информацию об имени классов, их методах и полях, а также описание классов, полей и методов.

| *QuestManager* | | |
|---|---|---|
| **Поле** | **Тип** | **Описание** |
| Instance | QuestManager | Паттерн одиночки |
| activeQuests | List<IQuest> | Активные квесты |
| ActiveQuests | List<IQuest> | Активные квесты |
| questPanel | GameObject | Панель квеста |
| questDescriptionText | TMPro.TMP_Text | Описание квеста |
| questNameText | TMPro.TMP_Text | Имя квеста |
| wormSpawner | WormSpawner | Ссылка на спавнер червей |

| *QuestManager* | | | |
|---|---|---|---|
| **Метод** | **Параметры** | **Тип** | **Описание** |
| GetActiveQuest | - | T | Получение активного квеста |
| Awake | - | void | Присвоение или разрушение объекта |
| Start | - | void | Запустить |
| AddQuest | newQuest: IQuest | void | Получить квест |
| UpdateQuestPanel | - | void | Обновить панель квеста |
| CheckQuests | - | void | Проверить квесты |
| UpdateActiveQuests | - | void | Обновить активные квесты |

| *CharacterMovement* | | |
|---|---|---|
| **Поле** | **Тип** | **Описание** |
| moveSpeed | float | Скорость передвижения |
| sprintMuliplier | float | Ускорение |
| maxStamina | float | Максимальная емкость ускорения |
| currentStamina | float | Текущее ускорение |
| staminaDrainRate | float | Скорость потребление энергии |
| staminaRechargeRate | float | Скорость перезарядки энергии |
| staminaRechargeDelay | float | Задержка перезарядки энергии |
| lastSprintTime | float | Последнее время ускорения |
| animator | Animator | Анимация |
| movement | Vector2 | Передвижение |
| stamina | Image | UI емкости |

| *CharacterMovement* | | | |
|---|---|---|---|
| **Метод** | **Параметры** | **Тип** | **Описание** |
| Start | - | void | Запуск |
| Update | - | void | Обновление |
| MoveCharacter | - | void | Передвижение персонажа |
| HandleStamina | - | void | Энергия |
| UpdateStaminaBar | - | void | Обновление энергии |
| UpdateAnimation | - | void | Обновление анимации |

| *Quest* | | |
|---|---|---|
| **Поле** | **Тип** | **Описание** |
| QuestID | string | ID квеста |
| QuestName | string | Название квеста |
| QuestDiscription | string | Описание квеста |
| IsCompleted | bool | Проверка на выполнение квеста |

| *Quest* | | | |
|---|---|---|---|
| **Метод** | **Параметры** | **Тип** | **Описание** |
| StartQuest | - | void | Начать квест |
| UpdateQuest | - | void | Обновить квест |
| CompleteQuest | - | void | Завершить квест |

| *CollectWormsQuest* | | |
|---|---|---|
| **Поле** | **Тип** | **Описание** |
| QuestID | string | ID квеста |
| QuestName | string | Название квеста |
| QuestDiscription | string | Описание квеста |
| IsCompleted | bool | Проверка на выполнение квеста |
| wormsRequired | int | Необходимое количество червей |
| wormsCollected | int | Собранное количество червей |

| *CollectWormsQuest* | | | |
|---|---|---|---|
| **Метод** | **Параметры** | **Тип** | **Описание** |
| StartQuest | - | void | Начать квест |
| UpdateQuest | - | void | Обновить квест |
| CompleteQuest | - | void | Завершить квест |
| CollectWorm | - | void | Собрать червей |

| *WormBehavior* | | |
|---|---|---|
| **Поле** | **Тип** | **Описание** |
| moveSpeed | float | Скорость передвижения |
| changeDirectionInterval | float | Измененить интервал взаимодействия |
| fleeDistance | float | Дистанция взаимодействия |
| fleeSpeedMultiplier | float | Скорость отдаления от персонажа |
| moveDirection | Vector2 | Направление отдаления |
| player | Transform | Персонаж |
| rb | Rigidbody2D | Хитбокс |
| nextDirectionChangeTime | float | Следующее время изменения |

| *WormBehavior* | | | |
|---|---|---|---|
| **Метод** | **Параметрь** | **Тип** | **Описание** |
| Start | - | void | Начать действия |
| Update | - | void | Обновить действия |
| SetRandomMoveDirection | - | void | Задать случайное движение |
| MoveInDirection | - | void | Двигаться в направлении |
| FleeFromPlayer | - | void | Уход от игрока |
| RotateSprite | - | void | Поворот NPC |

| *DialogueSystem* | | |
|---|---|---|
| **Поле** | **Тип** | **Описание** |
| characterDialogues | CharacterDialogue[] | Диалоги персонажа |
| dialogueText | TMP_Text | Текст диалога |
| characterImage | Image | Изображение персонажа |
| characterNameText | TMP_Text | Имя персонажа |
| dialoguePanel | GameObject | Панель диалога |
| optionsPanel | GameObject | Параметры панели |
| optionButton | Button[] | Параметры кнопки |
| textSpeed | float | Скорость появления текста |
| dialogueDictionary | Dictionary<string, Dialogue> | Словарь диалогов |
| currentDialogue | Dialogue | Текущий диалог |
| isTyping | bool | Проверка на ввод текста |
| playerMovement | CharacterMovement | Движение персонажа |

| *DialogueSystem* | | | |
|---|---|---|---|
| **Метод** | **Параметры** | **Тип** | **Описание** |
| Start | - | void | Начать действия |
| Update | - | void | Обновить действия |
| BuildDialogueDictionary | - | void | Создать словарь диалогов |
| StartDialogue | - | void | Наачть диалог |
| ShowDialogue | - | void | Показать диалог |
| TypeText | - | void | Писать текст |
| ShowOptions | - | void | Показать параметры |
| OnOptionSelected | - | void | По выбранному параметру |
| HandleOptionAction | - | void | Выбор параметра |
| HandleSpacePress | - | void | Действия от нажатия пробела |
| EndDialogue | | void | Завершить диалог |

| *HealthManager* | | |
|---|---|---|
| **Поле** | **Тип** | **Описание** |
| maxHealth | float | Максимальное здоровье |
| currentHealth | float | Текущее здоровье |
| healthBar | Image | UI панели |

| *HealthManager* | | | |
|---|---|---|---|
| **Метод** | **Параметры** | **Тип** | **Описание** |
| Start | - | void | Начать использование |
| Update | - | void | Обновить параметры |
| TakeDamage | damage: float | void | Получение урона |
| Heal | amount: float | void | Лечение |
| UpdateHealthBar | - | void | Обновление UI |

| *SpriteSorting* | | |
|---|---|---|
| **Поле** | **Тип** | **Описание** |
| CharacterRenderer | SpriteRenderer | Максимальное здоровье |
| currentHealth | float | Текущее здоровье |
| healthBar | Image | UI панели |

| *SpriteSorting* | | | |
|---|---|---|---|
| **Метод** | **Параметры** | **Тип** | **Описание** |
| Start | - | void | Начать использование |
| OnTriggerStay2D | other: Collider2D | void | Проверка на нахождение в объекте |

| *IQuest* | | |
|---|---|---|
| **Поле** | **Тип** | **Описание** |
| QuestID | string | ID квеста |
| QuestName | string | Название квеста |
| QuestDiscription | string | Описание квеста |
| IsCompleted | bool | Проверка на выполнение |

| *IQuest* | | | |
|---|---|---|---|
| **Метод** | **Параметры** | **Тип** | **Описание** |
| StartQuest | - | void | Начать квест |
| UpdateQuest | - | void | Обновление квеста |
| CompleteQuest | - | void | Завершить квест |

| *WormSpawner* | | |
|---|---|---|
| **Поле** | **Тип** | **Описание** |
| wormPrefab | GameObject | Объект червя |
| wormCount | int | Количество червей |
| spawnArea | Transform | Место появления |
| isActive | bool | Проверка на активность |

| *WormSpawner* | | | |
|---|---|---|---|
| **Метод** | **Параметры** | **Тип** | **Описание** |
| Start | - | void | Начать |
| ActiveSpawner | - | void | Активация спавна |
| SpawnWorms | - | void | Спавн червей |
| GetRandomPositionArea | - | Vector2 | Выбор случайной позиции для появления |

| *SpriteSort* | | |
|---|---|---|
| **Поле** | **Тип** | **Описание** |
| character | Transform | Персонаж |
| objectRenderer | SpriteRenderer | Объект для |
| fadeSpeed | float | Скорость затухания |
| threshold | float | Порог для затухания |
| targetAlpha | float | Цель |

| *SpriteSort* | | | |
|---|---|---|---|
| **Метод** | **Параметры** | **Тип** | **Описание** |
| Start | - | void | Начать |
| Update | - | void | Обновление |

| *Worm* | | |
|---|---|---|
| **Поле** | **Тип** | **Описание** |
| quest | CollectWormsQuest | Квест для сбора |

| *Worm* | | | |
|---|---|---|---|
| **Метод** | **Параметры** | **Тип** | **Описание** |
| Start | - | void | Начало |
| OnTriggerEnter2D | other: Collider2D | void | Взаимодействие объектов |

| *CameraSearch* | | | |
|---|---|---|---|
| **Метод** | **Параметры** | **Тип** | **Описание** |
| LateUpdate | - | void | Последнее обновление |

| *CameraSearch* | | |
|---|---|---|
| **Поле** | **Тип** | **Описание** |
| player | Transform | Максимальное здоровье |
| offset | Vector3 | Текущее здоровье |
| smoothSpeed | float | UI панели |

| *PlayButton* | | |
|---|---|---|
| **Поле** | **Тип** | **Описание** |
| - | - | - |

| *PlayButton* | | | |
|---|---|---|---|
| **Метод** | **Параметры** | **Тип** | **Описание** |
| StartGame | - | void | Перенос на основную сцену |

| *ExitButton* | | |
|---|---|---|
| **Поле** | **Тип** | **Описание** |
| - | - | - |

| *ExitButton* | | | |
|---|---|---|---|
| **Метод** | **Параметры** | **Тип** | **Описание** |
| ExitGame | - | void | Выход из игры |

## Код программы

CameraSearch.cs:

```
using UnityEngine;


public class CameraSearch : MonoBehaviour
{
    public Transform player;      // —сылка на объект игрока
    public Vector3 offset;        // —мещение камеры
    public float smoothSpeed = 0.125f; // —корость плавного движени€


    void LateUpdate()
    {
        if (player != null)
        {
            Vector3 targetPosition = player.position + offset; // ÷елева€
позици€ камеры
            transform.position = Vector3.Lerp(transform.position,
targetPosition, smoothSpeed); // Ћавное движение
        }
    }
}
```

CharacterMovement.cs:

```
using UnityEngine;
using UnityEngine.UI; // Для работы с UI компонентами


public class CharacterMovement : MonoBehaviour
{
    public float moveSpeed = 5f; // Базовая скорость
    public float sprintMultiplier = 1.5f; // Множитель для ускорения
```

```
    public float maxStamina = 100f; // Максимальная выносливость
    private float currentStamina; // Текущая выносливость
    public float staminaDrainRate = 20f; // Сколько выносливости тратится в секунду при спринте
    public float staminaRechargeRate = 10f; // Сколько выносливости восстанавливается в секунду
    private float staminaRechargeDelay = 2f; // Задержка перед началом восстановления выносливости после спринта
    private float lastSprintTime = 0f; // Время последнего спринта

    private Animator animator;
    private Vector2 movement;

    // Переменные для UI элементов
    public Image staminaBar; // Ссылка на компонент Image для отображения выносливости

    void Start()
    {
        animator = GetComponent<Animator>();
        currentStamina = maxStamina; // Изначально устанавливаем полную выносливость
    }

    void Update()
    {
        movement.x = Input.GetAxisRaw("Horizontal");
        movement.y = Input.GetAxisRaw("Vertical");

        UpdateAnimation();
```

```
        MoveCharacter();

        HandleStamina();
    }

    void MoveCharacter()
    {
        // Проверяем, удерживается ли клавиша Shift для ускорения
        bool isSprinting = Input.GetKey(KeyCode.LeftShift) ||
Input.GetKey(KeyCode.RightShift);
        bool isMoving = movement.sqrMagnitude > 0; // Проверка, двигается ли
персонаж

        float currentSpeed = moveSpeed;

        // Если персонаж спринтует и двигается, уменьшаем выносливость
        if (isSprinting && isMoving && currentStamina > 0)
        {
            currentSpeed *= sprintMultiplier; // Увеличиваем скорость
            currentStamina -= staminaDrainRate * Time.deltaTime; // Уменьшаем
выносливость
            lastSprintTime = Time.time; // Обновляем время последнего спринта
        }

        // Если выносливость закончилась, не даём ускоряться
        if (currentStamina <= 0)
        {
            currentStamina = 0; // Останавливаем выносливость на 0
            isSprinting = false; // Останавливаем спринт
```

```
        }

        // Обновляем позицию персонажа
        Vector3 newPosition = new Vector3(movement.x, movement.y,
0f).normalized;
        transform.position += newPosition * currentSpeed * Time.deltaTime;

        // Обновляем заполненность полосы выносливости
        UpdateStaminaBar();
    }

    void HandleStamina()
    {
        // Если спринт не активен и прошло достаточно времени после спринта,
начинаем восстановление выносливости
        if (!Input.GetKey(KeyCode.LeftShift) && !
Input.GetKey(KeyCode.RightShift) && currentStamina < maxStamina)
        {
            // Проверка времени восстановления
            if (Time.time - lastSprintTime >= staminaRechargeDelay)
            {
                currentStamina += staminaRechargeRate * Time.deltaTime; //
Восстанавливаем выносливость
            }
        }

        // Ограничиваем максимальной выносливостью
        if (currentStamina > maxStamina)
        {
            currentStamina = maxStamina;
```

```
        }
    }

    void UpdateStaminaBar()
    {
        // Обновляем заполненность UI изображения
        if (staminaBar != null)
        {
            staminaBar.fillAmount = currentStamina / maxStamina;
        }
    }

    void UpdateAnimation()
    {
        bool isMoving = movement.sqrMagnitude > 0;

        animator.SetFloat("MoveX", movement.x);
        animator.SetFloat("MoveY", movement.y);
        animator.SetBool("IsMoving", isMoving);

        // Ускоряем анимацию, если персонаж бежит
        bool isSprinting = Input.GetKey(KeyCode.LeftShift) ||
Input.GetKey(KeyCode.RightShift);
        animator.speed = isSprinting ? 1.5f : 1f; // Ускоряем анимацию на 50%
при спринте
    }
}
```

## CollectWormsQuest.cs:

```
using UnityEngine;
```

```
public class CollectWormsQuest : MonoBehaviour, IQuest
{
    public string QuestID => "collect_worms"; // ������������ ID
    public string QuestName => "���� ������������";
    public string QuestDescription => IsCompleted ? "������������ ⌀
NPC, ������ ������ ������."
                                                  : $"������������
{wormsCollected}/{wormsRequired} ������������.";

    public bool IsCompleted { get; private set; }

    private int wormsRequired = 5;
    private int wormsCollected = 0;

    public void StartQuest()
    {
        IsCompleted = false;
        wormsCollected = 0;
        Debug.Log("������ ������: ������������
������������.");
    }

    public void UpdateQuest()
    {
        if (wormsCollected >= wormsRequired)
        {
            IsCompleted = true;
            CompleteQuest();
            Debug.Log("��⌀ ⌀⌀⌀⌀⌀⌀⌀⌀ ⌀⌀⌀⌀⌀⌀⌀! ⌀⌀⌀⌀⌀
```

```
        ������.");
            QuestManager.Instance.UpdateQuestDescription(this,
"������ ������ NPC, ����  ����  ����.");
        }
    }

    public void CompleteQuest()
    {
        if (IsCompleted)
        {
            Debug.Log("����  ������! ������ NPC.");

            DialogueTrigger dialogueTrigger =
Object.FindFirstObjectByType<DialogueTrigger>();
            if (dialogueTrigger != null)
            {
                dialogueTrigger.startingDialogueID = "after_quest_1";
            }
            else
            {
                Debug.Log("����  ����-��");
            }
        }
    }

    public void CollectWorm()
    {
        if (IsCompleted) return;
```

```
        wormsCollected++;
        Debug.Log($"������ ��������: {wormsCollected}/{wormsRequired}");

        UpdateQuest();
        QuestManager.Instance.UpdateQuestDescription(this, QuestDescription);
    }
}
```

## DialogueSystem.cs:

```
using UnityEngine;
using UnityEngine.UI;
using TMPro;
using System.Collections;
using System.Collections.Generic;

public class DialogueSystem : MonoBehaviour
{
    [System.Serializable]
    public class DialogueOption
    {
        public string optionText;
        public DialogueAction action;
        public string nextDialogueID;

        public enum DialogueAction
        {
            AcceptQuest,
            DeclineQuest,
            EndQuest,
```

```
        Negotiate,
        ContinueDialogue
    }
}

[System.Serializable]
public class Dialogue
{
    public string dialogueID;
    [TextArea(2, 5)]
    public string text;
    public Sprite characterSprite;
    public string characterName;
    public DialogueOption[] options;
    public string nextDialogueID;
}

[System.Serializable]
public class CharacterDialogue
{
    public string characterID;
    public Dialogue[] dialogues;
}

public CharacterDialogue[] characterDialogues;
public TMP_Text dialogueText;
public Image characterImage;
public TMP_Text characterNameText;
public GameObject dialoguePanel;
```

```
    public GameObject optionsPanel;
    public Button[] optionButtons;
    public float textSpeed = 0.05f;

    private Dictionary<string, Dialogue> dialogueDictionary = new
Dictionary<string, Dialogue>();
    private Dialogue currentDialogue;
    private bool isTyping = false;

    private CharacterMovement playerMovement;

    void Start()
    {
        dialoguePanel.SetActive(false);
        optionsPanel.SetActive(false);

        playerMovement = Object.FindFirstObjectByType<CharacterMovement>();
        BuildDialogueDictionary();
    }

    void Update()
    {
        if (dialoguePanel.activeSelf && Input.GetKeyDown(KeyCode.Space))
        {
            if (isTyping)
            {
                // Завершаем печать текста
                StopAllCoroutines();
                dialogueText.text = currentDialogue.text;
```

```
                isTyping = false;
            }
            else
            {
                // Переход к следующему диалогу
                HandleSpacePress();
            }
        }
    }

    void BuildDialogueDictionary()
    {
        foreach (var character in characterDialogues)
        {
            foreach (var dialogue in character.dialogues)
            {
                if (!dialogueDictionary.ContainsKey(dialogue.dialogueID))
                {
                    dialogueDictionary.Add(dialogue.dialogueID, dialogue);
                }
                else
                {
                    Debug.LogWarning($"Duplicate Dialogue ID detected:
{dialogue.dialogueID}");
                }
            }
        }
    }
```

```
    public void StartDialogue(string characterID, string startDialogueID = null)
    {
        foreach (var character in characterDialogues)
        {
            if (character.characterID == characterID &&
character.dialogues.Length > 0)
            {
                // Если startDialogueID задан, то находим нужный диалог, иначе
начинаем с первого
                if (!string.IsNullOrEmpty(startDialogueID))
                {
                    if (dialogueDictionary.ContainsKey(startDialogueID))
                    {
                        currentDialogue = dialogueDictionary[startDialogueID];
                    }
                    else
                    {
                        Debug.LogWarning($"Диалог с ID {startDialogueID} не
найден.");
                        currentDialogue = character.dialogues[0];  // Используем
первый диалог по умолчанию, если ID не найден
                    }
                }
                else
                {
                    currentDialogue = character.dialogues[0];  // Начинаем с
первого диалога
                }

                ShowDialogue();
```

```
            dialoguePanel.SetActive(true);

            if (playerMovement != null)
                playerMovement.enabled = false;

            return;
        }
    }

    Debug.LogWarning($"Не найдены диалоги для персонажа: {characterID}");
}


void ShowDialogue()
{
    if (currentDialogue == null)
    {
        EndDialogue();
        return;
    }

    // Настройка UI
    characterNameText.text = currentDialogue.characterName;
    characterImage.sprite = currentDialogue.characterSprite;

    StopAllCoroutines();
    StartCoroutine(TypeText(currentDialogue.text));

    // Показываем варианты ответов, если они есть
```

```
        if (currentDialogue.options != null && currentDialogue.options.Length >
0)
        {
            ShowOptions(currentDialogue.options);
        }
        else
        {
            optionsPanel.SetActive(false);
        }
    }

    IEnumerator TypeText(string text)
    {
        isTyping = true;
        dialogueText.text = "";

        foreach (char c in text)
        {
            dialogueText.text += c;
            yield return new WaitForSeconds(textSpeed);
        }

        isTyping = false;
    }

    void ShowOptions(DialogueOption[] options)
    {
        optionsPanel.SetActive(true);
```

```
        for (int i = 0; i < optionButtons.Length; i++)
        {
            if (i < options.Length)
            {
                optionButtons[i].gameObject.SetActive(true);
                optionButtons[i].GetComponentInChildren<TMP_Text>().text =
options[i].optionText;

                DialogueOption option = options[i];
                optionButtons[i].onClick.RemoveAllListeners();
                optionButtons[i].onClick.AddListener(() =>
OnOptionSelected(option));
            }
            else
            {
                optionButtons[i].gameObject.SetActive(false);
            }
        }
    }

    void OnOptionSelected(DialogueOption option)
    {
        optionsPanel.SetActive(false);

        if (!string.IsNullOrEmpty(option.nextDialogueID) &&
dialogueDictionary.ContainsKey(option.nextDialogueID))
        {
            currentDialogue = dialogueDictionary[option.nextDialogueID];
            ShowDialogue();
        }
```

```
    else
    {
        HandleOptionAction(option.action);
    }
}

void HandleOptionAction(DialogueOption.DialogueAction action)
{
    switch (action)
    {
        case DialogueOption.DialogueAction.AcceptQuest:
            CollectWormsQuest wormsQuest = new CollectWormsQuest();
            QuestManager.Instance.AddQuest(wormsQuest);
            Debug.Log("Квест добавлен в систему!");

            EndDialogue();
            break;
        case DialogueOption.DialogueAction.EndQuest:
            QuestManager.Instance.EndQuests();
            Debug.Log("Квест убран из системы");

            EndDialogue();
            break;
        case DialogueOption.DialogueAction.DeclineQuest:
            Debug.Log("Задание отклонено.");
            EndDialogue();
            break;
        case DialogueOption.DialogueAction.Negotiate:
            Debug.Log("Начался процесс торговли.");
```

```
                EndDialogue();
                break;
            case DialogueOption.DialogueAction.ContinueDialogue:
                Debug.Log("Продолжение диалога.");
                EndDialogue();
                break;
        }
    }

    void HandleSpacePress()
    {
        if (!string.IsNullOrEmpty(currentDialogue.nextDialogueID) &&
dialogueDictionary.ContainsKey(currentDialogue.nextDialogueID))
        {
            // Переход к следующему диалогу
            currentDialogue =
dialogueDictionary[currentDialogue.nextDialogueID];
            ShowDialogue();
        }
        else
        {
            // Если нет следующего диалога, завершаем диалог
            EndDialogue();
        }
    }

    public void EndDialogue()
    {
        dialoguePanel.SetActive(false);
```

```
        optionsPanel.SetActive(false);


        if (playerMovement != null)
            playerMovement.enabled = true;
    }
}
```

## DialogueTrigger.cs:

```
using UnityEngine;


public class DialogueTrigger : MonoBehaviour
{
    public string characterID; // ”никальный идентификатор персонажа
    public string startingDialogueID; // ID начального диалога
    private DialogueSystem dialogueSystem;


    public GameObject interactPrompt; // UI подсказка


    private bool isPlayerInRange = false;


    // —сылка на квест
    public CollectWormsQuest collectWormsQuest;


    void Start()
    {
        dialogueSystem = Object.FindFirstObjectByType<DialogueSystem>();
        if (interactPrompt != null)
            interactPrompt.SetActive(false);
    }
```

```
void Update()
{
    if (isPlayerInRange && Input.GetKeyDown(KeyCode.E))
    {
        TriggerDialogue();
    }
}

void TriggerDialogue()
{
    if (dialogueSystem != null)
    {
        dialogueSystem.StartDialogue(characterID, startingDialogueID);
    }
    interactPrompt.SetActive(false);
}

void OnTriggerEnter2D(Collider2D other)
{
    if (other.CompareTag("Player"))
    {
        isPlayerInRange = true;
        interactPrompt.SetActive(true);
    }
}

void OnTriggerExit2D(Collider2D other)
{
    if (other.CompareTag("Player"))
```

```
        {
            isPlayerInRange = false;
            interactPrompt.SetActive(false);
        }
    }
}
```

ExitButton.cs:

```
using UnityEngine;

public class ExitButton : MonoBehaviour
{
    public void ExitGame()
    {
        Debug.Log("Game is exiting..."); // Для проверки в редакторе Unity
        Application.Quit();
    }
}
```

HealthManager.cs:

```
using UnityEngine;
using UnityEngine.UI;

public class HealthManager : MonoBehaviour
{
    public float maxHealth = 100f;
    private float currentHealth;

    public Image healthBar;
```

```
void Start()
{
    currentHealth = maxHealth;

    if (healthBar != null)
    {
        healthBar.fillAmount = 1f;
    }
}

void Update()
{

}

public void TakeDamage(float damage)
{
    currentHealth -= damage;

    if (currentHealth < 0)
    {
        currentHealth = 0;
    }
    UpdateHealthBar();
}

public void Heal(float amount)
{
    currentHealth += amount;
```

```
        if (currentHealth > maxHealth)
        {
            currentHealth = maxHealth;
        }

        UpdateHealthBar();
    }

    private void UpdateHealthBar()
    {
        if (healthBar != null)
        {
            healthBar.fillAmount = currentHealth / maxHealth;
        }
    }
}
```

## IQuest.cs:

```
public interface IQuest
{
    string QuestID { get; }
    string QuestName { get; }
    string QuestDescription { get; }
    bool IsCompleted { get; }

    void StartQuest();
    void UpdateQuest();
    void CompleteQuest();
}
```

PlayButton.cs:

```
using UnityEngine;
using UnityEngine.SceneManagement;


public class PlayButton : MonoBehaviour
{
    public void StartGame()
    {
        Debug.Log("Loading Game Scene..."); // Для проверки
        SceneManager.LoadScene("MainScene"); // Укажите имя сцены
    }
}
```

Quest.cs:

```
using UnityEngine;


public abstract class Quest : IQuest
{
    public string QuestID { get; protected set; }
    public string QuestName { get; protected set; }
    public string QuestDescription { get; protected set; }
    public bool IsCompleted { get; protected set; }


    public virtual void StartQuest()
    {
        Debug.Log($" вест '{QuestName}' начат!");
        IsCompleted = false;
    }
```

```
    public abstract void UpdateQuest();

    public virtual void CompleteQuest()
    {
        IsCompleted = true;
        Debug.Log($" вест '{QuestName}' завершен!");
    }
}
```

## QuestManager.cs:

```
using System.Collections.Generic;
using UnityEngine;

public class QuestManager : MonoBehaviour
{
    // Паттерн одиночки
    public static QuestManager Instance { get; private set; }

    private List<IQuest> activeQuests;

    public List<IQuest> ActiveQuests => activeQuests;

    public GameObject questPanel;
    public TMPro.TMP_Text questDescriptionText;
    public TMPro.TMP_Text questNameText;

    public WormSpawner wormSpawner; // Ссылка на спавнер

    public T GetActiveQuest<T>() where T : class, IQuest
    {
```

```
    Debug.Log("Проверка активных квестов...");

    foreach (var quest in activeQuests)
    {
        Debug.Log($"Проверка: {quest}, тип: {quest.GetType()}");

        if (quest is T matchedQuest)
        {
            Debug.Log($"Сравнение ссылок: {matchedQuest} == {quest}");
            return matchedQuest;
        }
    }

    // Логируем, если квест не был найден
    Debug.Log("Квест нужного типа не найден!");
    return null;
}

private void Awake()
{
    if (Instance == null)
    {
        Instance = this;
    }
    else
    {
        Destroy(gameObject);
    }
```

```
}

private void Start()
{
    activeQuests = new List<IQuest>();
    questPanel.SetActive(false);
}

// Добавление квеста в активные квесты
public void AddQuest(IQuest newQuest)
{
    activeQuests.Add(newQuest);
    UpdateQuestPanel();

    if (newQuest is CollectWormsQuest wormsQuest && wormSpawner != null)
    {
        Debug.Log("Активируем спавнер для квеста: " + wormsQuest.QuestName);
        wormSpawner.ActivateSpawner();
    }
    else
    {
        Debug.LogWarning("Спавнер не найден или квест неверного типа!");
    }

}

public void EndQuests()
{
    CheckQuests();
```

```
    }


    // Обновление панели с информацией о квестах
    public void UpdateQuestPanel()
    {
        if (activeQuests.Count > 0)
        {
            var quest = activeQuests[0];  // Отображаем только первый квест для
упрощения
            questNameText.text = quest.QuestName;
            questDescriptionText.text = quest.QuestDescription;
            questPanel.SetActive(true);
        }
        else
        {
            questPanel.SetActive(false);
        }
    }


    public void UpdateQuestDescription(IQuest quest, string newDescription)
    {
        if (ActiveQuests.Contains(quest))
        {
            // Обновляем текст UI
            questDescriptionText.text = newDescription;
            Debug.Log("Описание квеста обновлено: " + newDescription);
        }
        else
```

```
        {
            Debug.LogWarning("Попытка обновить описание несуществующего
квеста.");
        }
    }

    // Проверка завершения всех активных квестов
    public void CheckQuests()
    {
        for (int i = activeQuests.Count - 1; i >= 0; i--)
        {
            if (activeQuests[i].IsCompleted)
            {
                activeQuests[i].CompleteQuest();
                activeQuests.RemoveAt(i);
                UpdateQuestPanel();
            }
        }
    }

    // Метод для обновления состояния квестов
    public void UpdateActiveQuests()
    {
        foreach (var quest in activeQuests)
        {
            quest.UpdateQuest();
        }
    }
}
```

SpriteSort.cs:

```
using UnityEngine;

public class SpriteSort : MonoBehaviour
{
    public Transform character; // Ссылка на персонажа
    private SpriteRenderer objectRenderer;

    // Скорость изменения прозрачности
    public float fadeSpeed = 5f;

    // Порог, при котором персонаж считается "за" или "перед" объектом
    public float threshold = 0.1f;

    private float targetAlpha = 1f; // Целевая прозрачность

    void Start()
    {
        // Получаем SpriteRenderer объекта, к которому прикреплен этот скрипт
        objectRenderer = GetComponent<SpriteRenderer>();
    }

    void Update()
    {
        if (character != null)
        {
            // Определяем, ниже или выше персонаж по оси Y
            if (character.position.y < transform.position.y - threshold)
            {
```

```
                targetAlpha = 0.5f; // Если персонаж за деревом, сделать дерево
прозрачным
                objectRenderer.sortingOrder = 1; // Отрисовывать за персонажем
            }
            else if (character.position.y > transform.position.y + threshold)
            {
                targetAlpha = 1f; // Если персонаж перед деревом, сделать дерево
полностью непрозрачным
                objectRenderer.sortingOrder = -1; // Отрисовывать перед
персонажем
            }

            // Плавно изменяем прозрачность дерева
            Color currentColor = objectRenderer.color;
            currentColor.a = Mathf.Lerp(currentColor.a, targetAlpha, fadeSpeed *
Time.deltaTime);
            objectRenderer.color = currentColor;
        }
    }
}
```

SpriteSorting.cs:

```
using UnityEngine;

public class SpriteSorting : MonoBehaviour
{
    private SpriteRenderer characterRenderer;

    // Смещение для сравнения по оси Y
    public float positionOffsetTree = 0f;
    public float positionOffsetStump = 0f;
```

```
    void Start()
    {
        // Получаем SpriteRenderer персонажа
        characterRenderer = GetComponent<SpriteRenderer>();
    }

    void OnTriggerStay2D(Collider2D other)
    {
        if (other.CompareTag("Tree"))
        {
            SpriteRenderer otherRenderer = other.GetComponent<SpriteRenderer>();
            if (otherRenderer != null)
            {
                float objectY = other.transform.position.y + positionOffsetTree;
                if (transform.position.y < objectY)
                {
                    otherRenderer.sortingOrder = characterRenderer.sortingOrder
- 1;
                }
                else
                {
                    otherRenderer.sortingOrder = characterRenderer.sortingOrder
+ 1;
                }
            }
        }

        if (other.CompareTag("Stump"))
```

```
        {
            SpriteRenderer otherRenderer = other.GetComponent<SpriteRenderer>();
            if (otherRenderer != null)
            {
                float objectY = other.transform.position.y +
positionOffsetStump;
                if (transform.position.y < objectY)
                {
                    otherRenderer.sortingOrder = characterRenderer.sortingOrder
- 1;
                }
                else
                {
                    otherRenderer.sortingOrder = characterRenderer.sortingOrder
+ 1;
                }
            }
        }
    }
}
```

## Worm.cs:

```
using UnityEngine;


public class Worm : MonoBehaviour
{
    private CollectWormsQuest quest;


    private void Start()
    {
```

```
        // Ќаходим активный квест через QuestManager
        quest = QuestManager.Instance.GetActiveQuest<CollectWormsQuest>();

        if (QuestManager.Instance.GetActiveQuest<CollectWormsQuest>() == null)
        {
            Debug.LogWarning("CollectWormsQuest не активен!");
        }
    }

    private void OnTriggerEnter2D(Collider2D other)
    {
        if (other.CompareTag("Player"))
        {
QuestManager.Instance.GetActiveQuest<CollectWormsQuest>().CollectWorm(); //
”величиваем количество собранных черв€чков
            Destroy(gameObject); // ”дал€ем черв€ка после сбора
        }
    }
}
```

## WormBehavior.cs:

```
using UnityEngine;

public class WormBehavior : MonoBehaviour
{
    public float moveSpeed = 2f;
    public float changeDirectionInterval = 2f;
    public float fleeDistance = 5f;
    public float fleeSpeedMultiplier = 2f;
```

```
    private Vector2 moveDirection;
    private Transform player;
    private Rigidbody2D rb;
    private float nextDirectionChangeTime = 0f;

    void Start()
    {
        rb = GetComponent<Rigidbody2D>();
        player = GameObject.FindWithTag("Player").transform;
        SetRandomMoveDirection();
    }

    void Update()
    {
        if (Time.time >= nextDirectionChangeTime)
        {
            SetRandomMoveDirection();
            nextDirectionChangeTime = Time.time + changeDirectionInterval;
        }

        if (Vector2.Distance(transform.position, player.position) <=
fleeDistance)
        {
            FleeFromPlayer();
        }
        else
        {
            MoveInDirection();
```

```
        }

        RotateSprite();
    }

    void SetRandomMoveDirection()
    {
        float randomAngle = Random.Range(0f, 360f);
        moveDirection = new Vector2(Mathf.Cos(randomAngle),
Mathf.Sin(randomAngle)).normalized;
    }

    void MoveInDirection()
    {
        rb.linearVelocity = moveDirection * moveSpeed;
    }

    void FleeFromPlayer()
    {
        Vector2 fleeDirection = (transform.position -
player.position).normalized;
        rb.linearVelocity = fleeDirection * moveSpeed * fleeSpeedMultiplier;
    }

    void RotateSprite()
    {
        float angle = Mathf.Atan2(rb.linearVelocity.y, rb.linearVelocity.x) *
Mathf.Rad2Deg;

        transform.rotation = Quaternion.Euler(new Vector3(0, 0, angle-90));
```

```
    }
}
```

WormSpawner.cs:

```
using UnityEngine;


public class WormSpawner : MonoBehaviour
{
    public GameObject wormPrefab; // Префаб червяка
    public int wormCount = 5;     // Количество червяков
    public Transform spawnArea;   // Область для спавна

    private bool isActive = false; // Флаг активности спавнера

    void Start()
    {
        gameObject.SetActive(false); // Спавнер отключён по умолчанию
    }

    public void ActivateSpawner()
    {
        if (!isActive)
        {
            isActive = true;
            gameObject.SetActive(true);
            SpawnWorms();
        }
    }

    private void SpawnWorms()
```

```
    {
        for (int i = 0; i < wormCount; i++)
        {
            Vector2 spawnPosition = GetRandomPositionInArea();
            Instantiate(wormPrefab, spawnPosition, Quaternion.identity);
        }
    }

    private Vector2 GetRandomPositionInArea()
    {
        // Получаем случайную точку в пределах области спавна
        float x = Random.Range(spawnArea.position.x - spawnArea.localScale.x /
2,
                               spawnArea.position.x + spawnArea.localScale.x /
2);
        float y = Random.Range(spawnArea.position.y - spawnArea.localScale.y /
2,
                               spawnArea.position.y + spawnArea.localScale.y /
2);
        return new Vector2(x, y);
    }
}
```

## ЗАКЛЮЧЕНИЕ

В ходе выполнения курсовой работы была разработана 2D-игра на платформе Unity, включающая базовые игровые механики: управление персонажем, сбор предметов (червячков) по заданию и пользовательский интерфейс с главным меню и интерактивными кнопками.

Основной целью работы было изучение процесса создания игр на Unity, включая проектирование сцен, разработку игровых объектов и программирование логики игры на языке C#. Реализация проекта позволила приобрести навыки работы с игровым движком Unity, изучить методы взаимодействия объектов и принципы создания интерфейса для игр.

Результатом стал полностью функционирующий прототип игры, демонстрирующий ключевые этапы разработки 2D-игр. Работа имеет образовательный и практический характер, так как освоенные инструменты и подходы могут быть применены для дальнейшего углубления в разработку игр и создания более сложных игровых проектов.

Проект также может быть расширен, например, добавлением новых игровых механик, уровней или элементов графики, что позволит сделать его более увлекательным для пользователей. Курсовая работа достигла своей цели, доказав возможность реализации простых игровых приложений с использованием Unity.

## ССЫЛКА НА github

https://github.com/ZamniProg/GAME