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 компьютерной игры «The Dark Path» на платформе Unity

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
ЮУрГУ – 09.03.04.2024.308-369.ВКР

Научный руководитель,
доцент кафедры СП, к.т.н.
_____ М.В. Сухов

Автор работы,
студент группы КЭ-404
_____ Г.С. Федяшов

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-404
Федяшову Георгию Сергеевичу,
обучающемуся по направлению
09.03.04 «Программная инженерия»

1. **Тема работы** (утверждена приказом ректора от 22.04.2024 г. № 764-13/12)

Разработка компьютерной игры «The Dark Path» на платформе Unity.

2. **Срок сдачи студентом законченной работы:** 03.06.2024 г.
3. **Исходные данные к работе**

3.1. Руководство Unity. [Электронный ресурс] URL: https://docs.unity3d.com/ru/530/Manual/index.html (дата обращения: 29.01.2024 г.).

3.2. Майк Гейг. Разработка игр на Unity за 24 часа. [Электронный ресурс] URL: https://disk.yandex.ru/i/F1ZzqBQkqIll5g (дата обращения: 29.01.2024 г.).

3.3. Бонд Д. Unity и CS, Геймдев от идеи до реализации (для профессионалов). [Электронный ресурс] URL: https://www.rulit.me/books/unity-i-s-gejmdev-ot-idei-do-realizaci-download-649239.html (дата обращения: 29.01.2024 г.).

4. **Перечень подлежащих разработке вопросов**

4.1. Провести анализ предметной области.

4.2. Провести анализ существующих решений.

4.3. Разработать требования к приложению.

4.4. Реализовать игровое приложение и функциональные скрипты.

4.5. Провести тестирование приложения.

5. Дата выдачи задания: 29.01.2024 г.

Научный руководитель,
доцент кафедры СП, к.т.н. М.В. Сухов

Задание принял к исполнению Г.С. Федяшов

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Актуальность

В настоящее время компьютерные игры занимают ведущие позиции в рейтинге самых популярных видов развлечений среди людей различных возрастных категорий. С каждым годом наблюдается неуклонный рост спроса на игровые продукты, способные предложить игрокам не просто времяпрепровождение, но и возможность погружения в уникальные виртуальные миры [1]. В связи с постоянно растущим интересом к играм, сфера разработки компьютерных игр становится крайне важной и стремительно развивающейся областью в индустрии информационных технологий и развлечений.

Особый интерес среди широкого спектра игровых жанров вызывают пиксельные игры [2]. Созданные в уникальном стиле пиксель-арт, эти игры выделяются среди прочих своим необычайным визуальным решением, которое несет в себе ностальгическое очарование эпохи ранних видеоигр. Этот стиль позволяет разработчикам создавать игры с уникальной атмосферой, привлекая внимание игроков своей визуальной составляющей.

Таким образом, разработка пиксельных игр становится все более востребованной задачей, открывающей новые возможности для творчества и инноваций в игровой индустрии. Игры, выполненные в стиле пиксель-арт, предлагают игрокам не просто возвращение к истокам жанра, но и возможность исследования новаторских игровых механик и необычных сюжетных линий, которые делают каждую игру уникальным продуктом.

Постановка задачи

Основной целью выпускной квалификационной работы является разработка компьютерной игры «The Dark Path» на платформе Unity.

Для достижения поставленной цели необходимо решить следующие задачи:

1) изучить литературу по Unity;
2) провести анализ предметной области и существующих решений;

3) разработать требования к приложению;

4) реализовать игровое приложение и скрипты;

5) провести тестирование скриптов для проверки их корректной работы.

Структура и содержание работы

Работа состоит из введения, пяти глав, заключения, литературы и приложений. Объем работы составляет 42 страницы, объем списка литературы – 15 источников.

В первой главе «Анализ предметной области и существующих работ» приведен анализ аналогичных проектов и анализ существующих решений для реализации проекта.

Во второй главе «Анализ требований к программной системе» содержится реализация диаграммы вариантов использования, описание функционирования игры и таблицы спецификации вариантов использования.

В третьей главе «Архитектура программной системы» описана файловая система игры, диаграмма компонентов и диаграмма деятельности игрового процесса.

В четвертой главе «Реализация системы» представлен процесс реализации компонентов системы с демонстрацией листингов и скриншотов пользовательского интерфейса.

В пятой главе «Тестирование системы» приведены результаты тестирования системы.

В приложении А содержатся спецификации вариантов использования игрового приложения.

В приложении Б представлен основной скрипт игровых уровней.

В приложении В приведен скрипт, отвечающий за считывание нажатия клавиш на клавиатуре или жестов пользователя.

1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ И СУЩЕСТВУЮЩИХ РАБОТ

1.1. Анализ аналогичных проектов

Пиксельные игры стали неотъемлемой частью поп-культуры, привлекая миллионы игроков по всему миру своей динамикой, графикой и уникальными игровыми механиками.

Разрабатывая новую игру, стоит стремимся взять лучшее из уже существующих проектов, адаптировав их элементы к нашему уникальному визионерскому видению захватывающей игры.

Pixel Castle Runner

Pixel Castle Runner [3] – это инди-игра, разработанная небольшой командой энтузиастов, ценителей пиксельной графики (рисунок 1).

Рисунок 1 – Игра Pixel Castle Runner

Жанр игры – пиксельный раннер, который сочетает простоту и захватывающий опыт. В игре нет запутанного сюжета, сложных игровых механик и запоминающихся боссов. Вместо этого, ее простота и уникальный визуальный стиль вызывают ностальгию и быстрое погружение в игровой процесс. Для данного проекта идеально подходит концепция бесконечного бега из Pixel Castle Runner.

Loop Hero

Loop Hero [4] от студии Four Quarters – это уникальное сочетание жанров, включая элементы раннера, RPG и стратегии (рисунок 2).

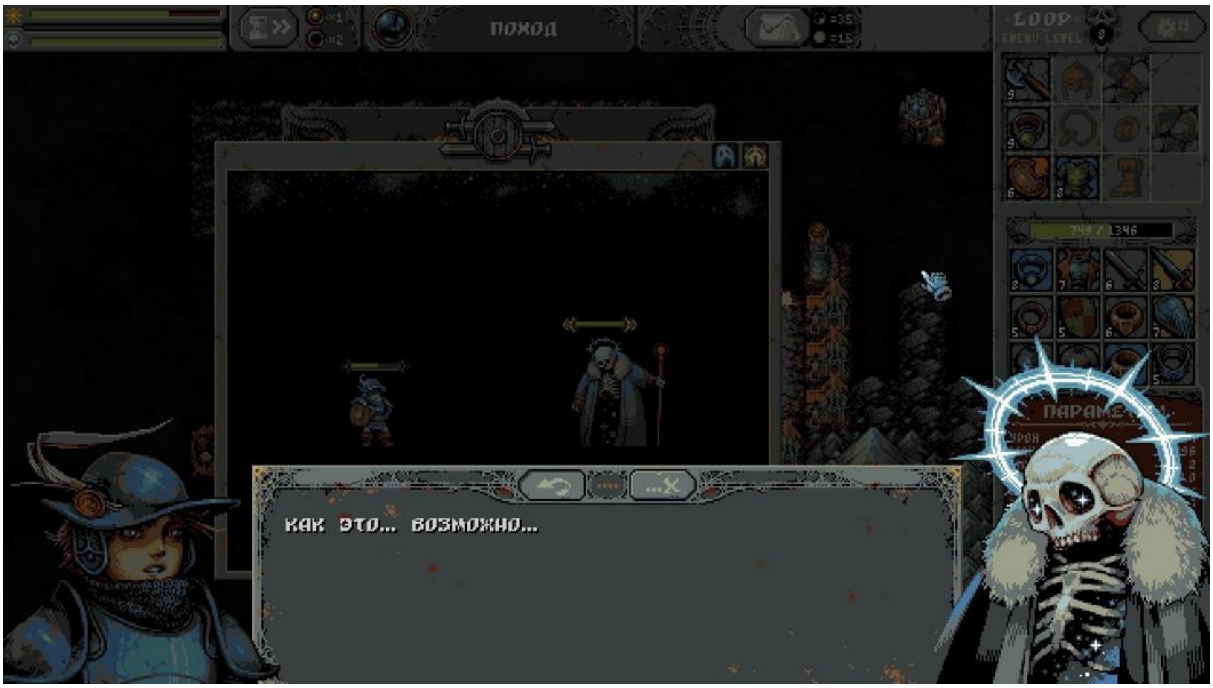

Рисунок 2 – Игра Loop Hero

Игра поражает своим глубоким сюжетом, где главный герой сталкивается с бесконечным циклом сражений и исследований. Loop Hero превосходно демонстрирует, как можно сочетать пиксельную графику с захватывающим повествованием и сложными игровыми механиками. Из Loop Hero необходимо взять идею вовлечения игрока в сюжет через диалоги, постоянное развитие персонажа и мира вокруг него.

Blasphemous

Blasphemous [5] – это игра от испанской студии The Game Kitchen, которая выделяется своей сложностью. Она привлекает пользователей своей атмосферой, сложными испытаниями и интересными боссами. Идеи сложных уровней, для прохождения которых нужно умело управлять игровым персонажем из Blasphemous, могут завлечь игроков, которые постоянно ищут вызов в компьютерных играх. Игровая сцена сражения с босом из игры Blasphemous представлена на рисунке 3.

Рисунок 3 – Игра Blasphemous

1.2. Анализ существующих решений для реализации проекта

Для создания качественной игры необходимо использовать качественные инструменты. Одним из таких инструментов является Unity [6]. Unity – это бесплатная кроссплатформенная среда разработки компьютерных, мобильных или консольных игр. Unity содержит огромное количество технических средств для создания компьютерных игр:

1) встроенные библиотеки и плагины;
2) потоковое аудио и видео;
3) поддержка языка скриптования: C# [7].

Кроме того, Unity имеет подробное руководство [8], огромное количество обучающих курсов, пособий, статей, видеоуроков и документаций.

Вывод по первой главе

В результате анализа предметной области, была выбрана платформа Unity для реализации игры. Так же определено, что в игре должны быть реализованы сложные уровни и диалоги для раскрытия сюжета и описания игрового мира.

2. АНАЛИЗ ТРЕБОВАНИЙ К ПРОГРАММНОЙ СИСТЕМЕ

Целью данной главы является проведение анализа требований пользователей и определение ключевых характеристик и функциональности игры, которые обеспечат эффективный и удобный пользовательский опыт.

2.1. Требования к проектируемой системе

Функциональные требования

Проектируемое приложение должно предоставлять пользователю следующие возможности.

1. Пользователь должен иметь возможность начать новую или продолжить уже начатую игру.
2. Пользователь должен иметь возможность настраивать игру, включать и выключать звуки и музыку с помощью специальных кнопок, которые располагаются в главном меню.
3. Пользователь должен иметь возможность закрыть игровое приложение через главное меню.
4. В ходе игры пользователь должен иметь возможность управлять игровым персонажем, передвигать его по трем направляющим или использовать атаку.
5. В ходе игры пользователь должен иметь возможность улучшать характеристики игрового персонажа через специальною меню прокачки.
6. Приложение должно сохранять накопленный опыт и уровень прокачки персонажа.

Нефункциональные требования

Ниже представлен перечень требований, которым должно соответствовать разрабатываемое приложение.

1. Приложение должно поддерживать следующие ОС: Windows 10 и Windows 11.
2. Приложение должно поддерживать управление игрой с помощью клавиатуры и манипулятора мыши.

2.2. Диаграмма вариантов использования

На основании описанных в пункте 2.1 функциональных требований, была разработана диаграмма вариантов использования игры. UML диаграмма приведена на рисунке 4.

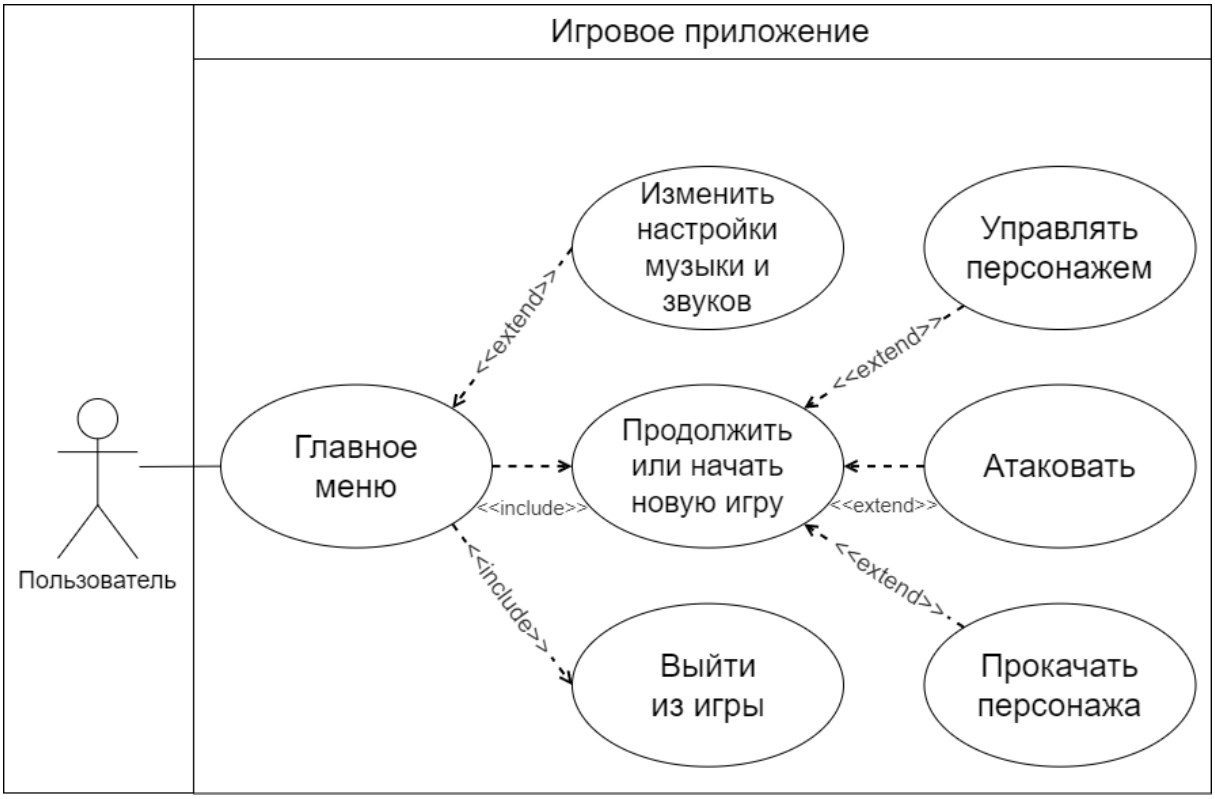

Рисунок 4 – Диаграмма вариантов использования

Основным актером является сам пользователь приложения, которому доступны возможности взаимодействия с приложением. Пользователь является единственным актером.

В приложении А представлены спецификации основных вариантов использования игры «The Dark Path». Варианты использования описывают ключевые действия, доступные игроку в игровом приложении.

Вывод по второй главе

В результате проделанной работы был проведен анализ требований пользователей и определены ключевые характеристики и функциональности игры, которые обеспечивают эффективный и удобный пользовательский опыт.

3. АРХИТЕКТУРА ПРОГРАММНОЙ СИСТЕМЫ

3.1. Общее описание архитектуры системы

Структура игры состоит из четырех основных игровых элементов.

1. Главное меню – сцена, которая загружается при запуске игры. На ней располагаются кнопки «Играть», «Выход», «Об Авторе» и кнопки настройки громкости музыки и общих звуков игры.

2. Сюжетные заставки – сцены, которые загружается, в определенные моменты игры, посвящая игрока в историю игры. На них располагаются изображения и слайды с текстом сюжета игры или диалогом.

3. Игровой уровень – сцена, на которой протекает основной геймплей. Главный персонаж движется вперед, параллельно уворачиваясь от препятствий или разрушая их. Одна сцена соответствует одному игровому уровню. Уровень можно поставить на паузу, нажав кнопку «Пауза» и убрать паузу, нажав кнопку «Продолжить».

4. Меню прокачки персонажа – сцена, в которой пользователь может потратить очки персонажа на его улучшение. Через эту сцену можно перейти на игровой уровень или выйти в главное меню.

Файловая структура игры представлена на рисунке 5.

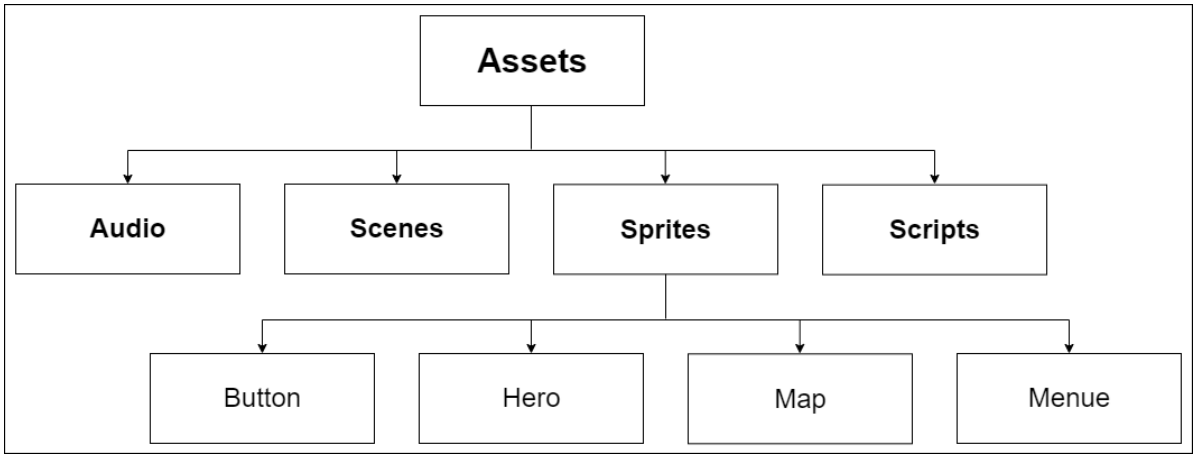

Рисунок 5 – Файловая структура игры

Описание файлов и их содержания:

1) Scenes – содержит в себе сцены игры;
2) Audio – содержит в себе аудиофайлы игры;

3) Scripts – содержит в себе набор скриптов, которые отвечают за механику игры и определяют поведение игровых объектов;

4) Sprites – содержит в себе набор директорий, в которых грамотно расположены: изображения игровых объектов, иконки, кнопки, материалы игрового персонажа, задние фоны и шрифты.

3.2. Общее компонентов, составляющих систему

На рисунке 6 представлена диаграмма компонентов игрового приложения. Данные компоненты содержат в себе классы, описывающие поведение, свойства, параметры и зависимости игровых объектов.

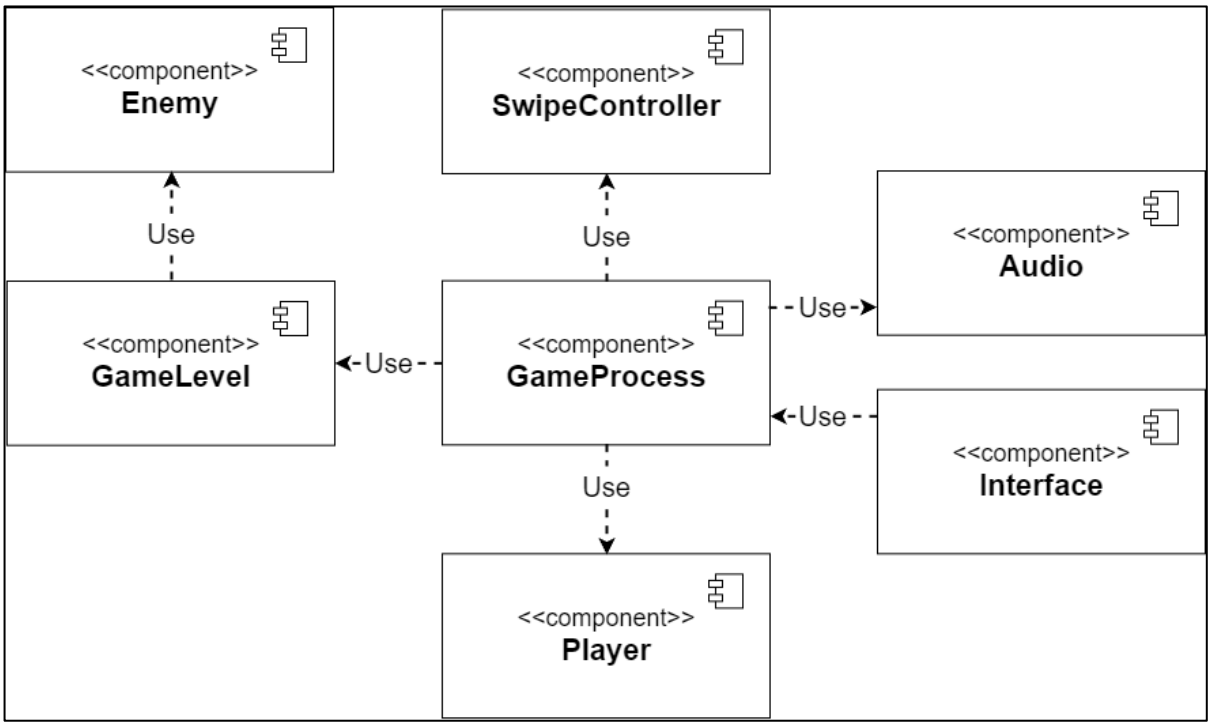

Рисунок 6 – Диаграмма компонентов

Описание компонентов, показанных на рисунке 6 представлено ниже.

1. Player – содержит классы и параметры, связанные с главным игровым персонажем.

2. GameProcess – это набор классов, которые управляют игровым процессом, включая перемещение персонажа, и его взаимодействия с другими игровыми объектами.

3. GameLevel – компонент отображения игрового уровня, который отвечает за перемещение фоновых элементов.

4. Enemy – компонент, который отвечает за поведение противников, удаляет их после убийства.

5. SwipeController – отвечает за обработку пользовательского ввода.

6. Interface – это набор UI элементов: счетчиков, кнопок, изображений, панелей и тумблеров.

7. Audio – компонент, отвечающий за воспроизведение аудио эффектов и музыки.

3.3. Диаграмма деятельности

На рисунке 7 представлена диаграмма деятельности управления персонажем. Эта диаграмма показывает, как протекает игровой процесс, возможности игрока и отклик системы.

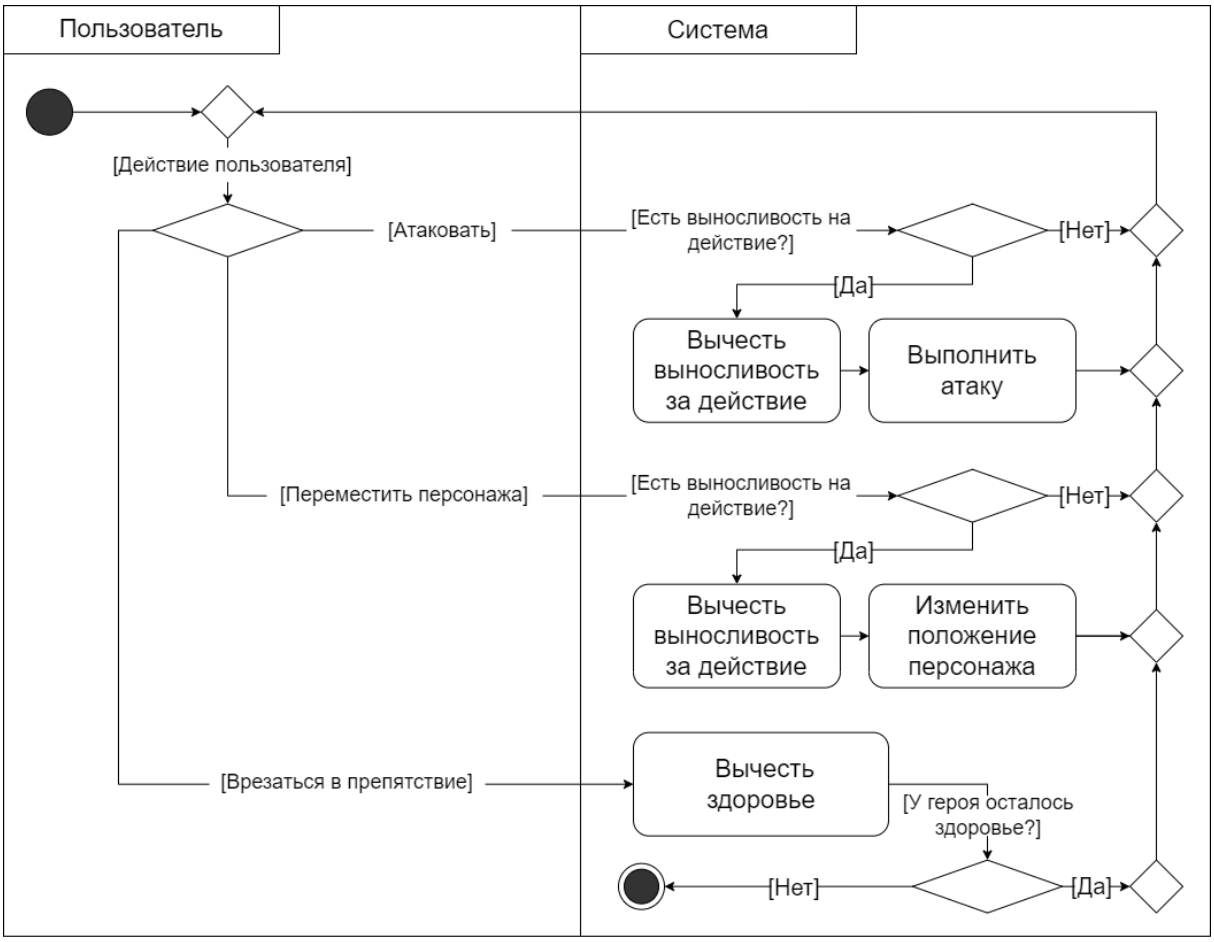

Рисунок 7 – Диаграмма деятельности управления персонажем

Диаграмма деятельности демонстрирует только управление персонажем на игровом уровне, полный игровой процесс компьютерной игры описан ниже.

1. Погружение в сюжет – все начинается с погружения игрока в увлекательный виртуальный мир. Через кат-сцены и диалоги игрок знакомится с ключевыми событиями сюжета, что погружает его в захватывающую атмосферу игры.

2. Главный игровой процесс – основная часть игры представлена игровыми уровнями разной сложности. На каждом уровне персонажу предстоит либо уворачиваться от разнообразных препятствий, либо сражаться с монстрами, которые встречаются на его пути. Эта часть игры требует от игрока ловкости, стратегического мышления и навыков управления персонажем.

3. Процесс прокачки – важным элементом игрового процесса является процесс прокачки персонажа. Здесь каждый пользователь имеет возможность самостоятельно выбирать, какие характеристики развивать, чтобы адаптировать персонажа под свой игровой стиль. Это позволяет каждому игроку индивидуализировать свой опыт игры.

Вывод по третьей главе

В результате проделанной работы была разработана структура игры и файловая система приложения, с кратким описанием ее основных компонентов. Помимо этого, создана диаграмма компонентов для описания структурного аспекта системы и сформирована диаграмма деятельности управления персонажем. Также подробно описан весь игровой процесс компьютерной игры.

4. РЕАЛИЗАЦИЯ СИСТЕМЫ

Для создания игры была выбрана платформа Unity. Она обладает всеми необходимым инструментарием для создания компьютерных, мобильных или консольных игр. На рисунке 8 изображен интерфейс Unity.

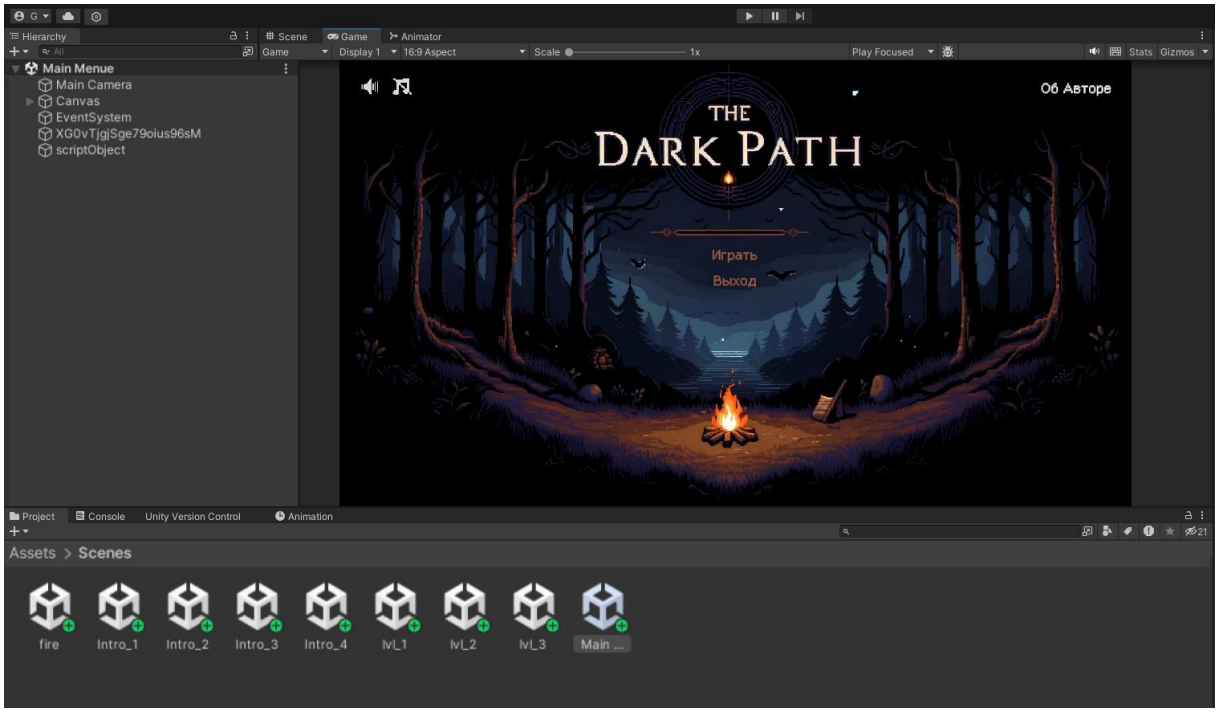

Рисунок 8 – Интерфейс Unity

Инструменты Unity [9], которые будут использованы для разработки игры, представлены ниже.

1. Инструмент «Transform» – базовый инструмент, который позволяет изменять размер, положение и угол поворота объектов.
2. Инструмент «Camera» – базовый инструмент управления камерой, который позволяет настраивать цвет заднего фона и дальность видимости.
3. Инструмент «Light» – базовый инструмент, который позволяет настраивать цвет и интенсивность источника света.
4. Инструмент «Collider» – создает невидимую сетку вокруг объекта, для взаимодействия с другими объектами.
5. Инструмент «Character Controller» – позволяет управлять игровым персонажем.
6. Инструмент «Canvas» – позволяет добавлять UI объекты.

7. Инструмент «Event System» – позволяет управлять событиями, такими как нажатия кнопок или выключателей.

8. Инструмент «Audio Source» – позволяет добавлять и воспроизводить звуки и музыку.

Некоторые игровые объекты были найдены в магазине Unity. Unity Asset Store – это платформа готового контента для разработки, в нем публикуется множество решений с платным или бесплатным доступом.

Для написания и редакции кодов на C# использовался продукт компании Microsoft – Visual Studio.

Аудио сопровождение – звуки и музыка для игры были взяты с сайта «zvukogram» [10]. Аудиофайлы без авторских прав – это значит, что их можно скачать бесплатно и использовать в своей игре.

Для отрисовки объектов и главного героя, использовались программы Adobe Photoshop и Figma. Эти графические редакторы идеально подходят для создания новых изображений и генерации визуальных эффектов. На рисунке 9 представлена разработка окна диалогов в графическом редакторе Figma [11].

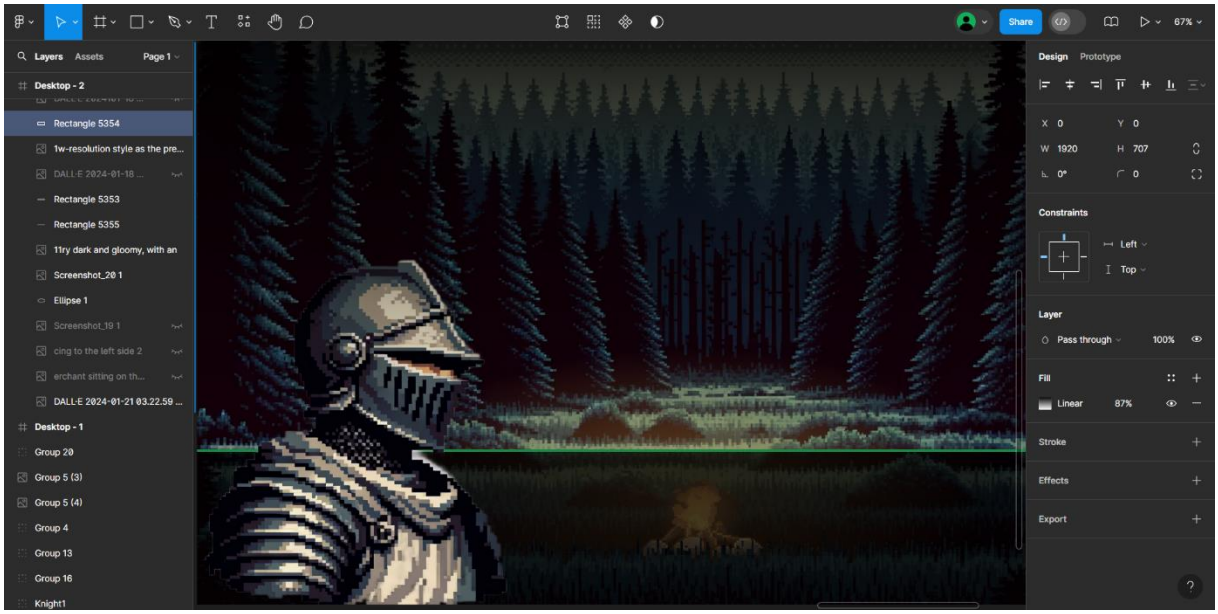

Рисунок 9 – Разработка диалогового окна в Figma

4.1. Реализация персонажа и его анимаций

Кадровая анимация – это искусство создания движущихся изображений чередой кадров. Одним из ключевых моментов в этом процессе является техника смены изображений. Это означает последовательное замещение одного изображения другим с определенной частотой, что создает визуальный эффект движения. Кадровая анимация используется для отображения бега, удара, прыжка и спрыгивания героя.

Каждый кадр главного персонажа был тщательно разработан методом ручной отрисовки. В процессе было уделено большое внимание к деталям и реалистичности пластики персонажа.

Для создания анимации бега использовалось всего четыре кадра, для сохранения баланса между выразительностью и оптимизацией. Первый кадр представляет начальную фазу бега с определенной позой ног и торса. Следующие три кадра отображают поочередные фазы движения ног, передавая естественное движение бегущего персонажа. Смена между этими кадрами происходит настолько быстро, что создается иллюзия бега, кадры бега представлены на рисунке 10.

Рисунок 10 – Кадры анимации бега

Для анимации удара также было использовано четыре кадра. Начальный кадр зафиксирован в момент подготовки к удару, с последующими тремя кадрами, изображающими движение меча в момент удара, кадры анимации удара представлены на рисунке 11.

Рисунок 11 – Кадры анимации удара

Для отображения прыжка и спрыгивания использовался всего 1 кадр анимации. Благодаря высокой динамики игры, персонаж быстро перемещается по игровому полю, что позволяет использовать минимальное количество кадров перемещения, кадр спрыгивания представлен на рисунке 12.

Рисунок 12 – Кадр анимации спрыгивания

Для своевременного отображения различных действий персонажа в игре, была создана специальная последовательность использования анимаций. Логика системы анимаций представлена на рисунке 13.

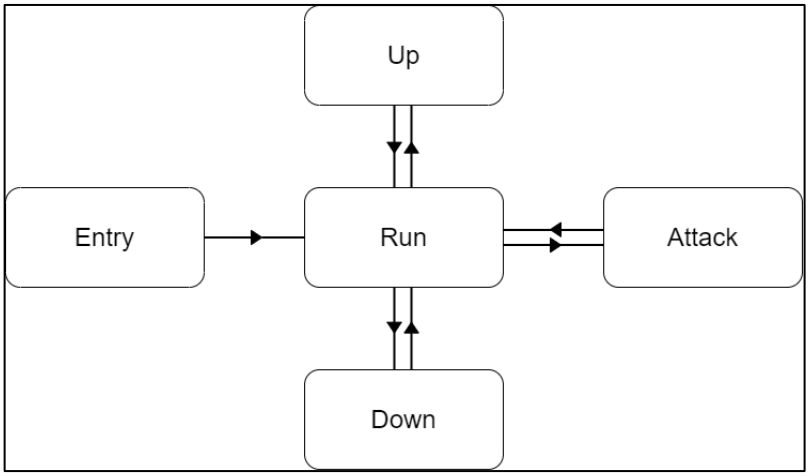

Рисунок 13 – Система анимационных клипов

На рисунке представлены анимационные клипы, которые связаны между собой определенным образом. По умолчанию персонаж стоит, и никакая анимация не воспроизводится. Когда загружается игра, запускается основная анимация бега, при выполнении определенного действия (удара или перемещения) – выполняется анимация этого действия, например, атаки. В листинге 1 показана часть скрипта `Player` отвечающая за атаку, полный листинг представлен в приложении Б.

Листинг 1 – Часть скрипта `Player`, отвечающая за атаку

```
if (SwipeController.action && !isAttacking && endurance > 0)
{
    if (volume == 1)
        PlaySound(sounds[1]);
    animator.SetBool("attak", true); // Включение анимации атаки
    isAttacking = true;
    endurance -= 40f; // Расход выносливости на атаку
    enduranceBar.fillAmount = endurance * 0.01f;
    attackTimer = 0f;
}

if (isAttacking)
{
    attackTimer += Time.deltaTime;
    if (attackTimer >= attackDuration)
    {
        animator.SetBool("attak", false); // Выключение анимации атаки
        isAttacking = false;
    }
}
```

4.2. Реализация пользовательского интерфейса

В данном компоненте реализованы такие подсистемы как взаимодействие игрока с кнопками и тумблерами, отображение счетчика очков и игровые панели.

Все доступные игроку графические объекты, являющие пользовательским интерфейсом, расположены на объекте `Canvas` [12]. `Canvas` – это область, внутри которой находятся все элементы пользовательского интерфейса.

Для реализации кнопок и переключателей использовался базовый функционал Unity, с помощью которого можно создавать на экране UI элементы `Button` и `Toggle`. Задние фоны и иконки создавались вручную, для

большего соотношения общей визуальной стилистики. На рисунке 14 представлены кнопки настроек музыки и аудио эффектов.

Рисунок 14 – Настройки громкости музыки и аудио эффектов

Для реализации счетчика очков был использован постоянно обновляющийся текст. На рисунке 15 представлен пример интерфейса отображения накопленных очков.

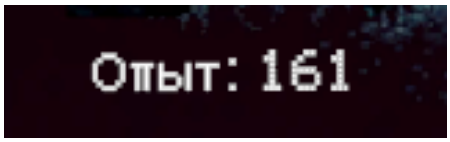

Рисунок 15 – Счетчик очков

Кнопки перехода между сценами реализованы через базовый модуль отвечающий за управление сценами `UnityEngine.SceneManagement` [13] и его метод `LoadScene()`. Сцена главного меню имеет индекс 0, так как она самая первая сцена, которую должен видеть пользователь. Все остальные сцены пронумерованы по логике повествования игры.

Отображение полосок здоровья и выносливости реализовано через изображение прямоугольника с характеристиками: тип `Filled`, метод `Horizontal`, направление `Left`. Когда персонаж теряет здоровье или выносливость, соответствующая полоска с отображением характеристики постепенно исчезает по горизонтали слева на право, отвечает за это переменная `Amount`. На рисунке 16 представлен пример отображения полосок здоровья и выносливости.

Рисунок 16 – Полоски здоровья и выносливости

Всплывающие панели используются практически на всех сценах. Некоторые помогают пользователю лучше разобраться в игре, объясняя механики и управление игры. Другие являются неотъемлемой частью игры, например, на одной из них располагается меню прокачки главного персонажа. На рисунке 17 представлен пример интерфейса улучшения характеристик персонажа.

Рисунок 17 – Панель прокачки персонажа

4.3. Реализация игрового уровня

Игровой уровень состоит из 3 основных слоев, которые выполняют определенную функцию и создают главную атмосферу игры. Первый слой – задний план, он состоит 8 изображений, расположенных друг за другом. Это добавляет игре объем и возможность использовать параллакс эффект. С помощью скрипта `SceneMove`, задние слои движутся вместе с героем, чем дальше слой, тем больше его скорость. В листинге 2 показан код перемещения задних слоев фона.

Листинг 2 – Скрипт `SceneMove`

```
public class SceneMove : MonoBehaviour
{
    // Скорость задается в Unity
    public float speed;

    void FixedUpdate()
```

```
    {
        float moveDistance = speed * Time.deltaTime;
        transform.Translate(Vector3.left * moveDistance);
    }
}
```

Если первый слой отвечает за задний план, то второй слой отвечает за передний. На нем располагаются линии перемещения персонажа и другие визуальные элементы, которые служат дополнением к окружению и никак не взаимодействуют с игровым персонажем.

Финальный слой уровня содержит в себе объекты, с которыми игровой персонаж может взаимодействовать. На этом слое расположены препятствия и монстры, в которых персонаж может врезаться, в следствии чего потеряет часть здоровья. Помимо этого взаимодействия, игрок может разрубать мечом монстров и простые препятствия. Объекты данного слоя имеют отличительные теги, с помощью которых скрипты отличают их от других статичных элементов заднего фона. На рисунке 18 продемонстрированы все слои игрового уровня.

Рисунок 18 – Демонстрация слоев игровой сцены

4.4. Реализация игрового процесса

В данном компоненте реализованы такие подсистемы как перемещение и управление персонажем, а также его взаимодействие с игровыми объектами.

В начале уровня скрипт `Player` считывает уровень каждой характеристики персонажа, их значения напрямую влияют на скорость вертикального перемещения персонажа, его здоровье, выносливость и силу удара.

Персонаж бежит вперед с постоянной скоростью, значение этой переменной задается в редакторе Unity и зависит от игрового уровня, с каждым уровнем, скорость немного увеличивается.

Для реализации управления персонажем в скрипте `SwipeController` (приложение В) идет проверка нажатия кнопки клавиатуры или жеста пользователя. В скрипте `Player` (приложение Б) после инициализации действия – меняется положение игрового персонажа или происходит атака.

Во время игры главный герой получает внутриигровой опыт, который в дальнейшем сможет потратить на усиление своих характеристик. При увеличении количества опыта, его новое значение сразу записывается в память игры, чтобы пользователь имел возможность выйти из игрового уровня в любой момент и не потерять накопленные очки.

Так как в игре есть прогрессия уровней, необходимо записывать пройденные игроком уровни в память игры. Если пользователь открывает игру в первый раз, то его прогресс равен 0, после просмотра вступительного ролика его прогресс увеличивается до 1. После этого игрок может загружать игровые уровни. Прогресс повышается только после прохождения нового уровня.

В игре есть несколько классов объектов. Их взаимодействие с персонажем определяется через `CharacterController` [14] и тег объекта. У не разрушаемого препятствия тег `obstacle`, у монстров и разрушаемых препятствий тег `enemy`, а у финишной линии, которая отвечает за конец уровня тег `finish`. Препятствия реализованы в виде триггера, чтобы персонаж не

врезался в них, и не получал урон от одного препятствия несколько раз. В скрипте `Player` метод `OnTriggerEnter()` отвечает за взаимодействие с препятствиями.

Игровой процесс можно в любой момент поставить на паузу, для этого необходимо нажать на кнопку в верхнем правом углу экрана или нажать кнопку «Escape» на клавиатуре. В листинге 3 продемонстрирован скрипт `PauseGame`, отвечающий за остановку игрового времени.

Листинг 3 – Скрипт `PauseGame`

```
public class PauseGame : MonoBehaviour
{
    [SerializeField] private GameObject pausePanel;
    public int progress;

    private void Start()
    {
        progress = PlayerPrefs.GetInt("PlayerProgress");
    }
    public void pauseGame()
    {
        pausePanel.SetActive(true);
        Time.timeScale = 0;
    }

    public void resumeGame()
    {
        pausePanel.SetActive(false);
        Time.timeScale = 1;
    }

    public void newGame()
    {
        pausePanel.SetActive(false);
        Time.timeScale = 1;
        SceneManager.LoadScene(progress);
    }

    public void pauseToMenue()
    {
        SceneManager.LoadScene(8);
    }

    void Update()
    {
        if (SwipeController.pause)
        {
            pausePanel.SetActive(true);
            Time.timeScale = 0;
        }
    }
}
```

4.5. Реализация звукового сопровождения

Звуковая система аудио эффектов реализована через скрипт `Sounds`, он немного изменяет тональность звука перед его воспроизведением, чтобы одинаковые аудиодорожки звучали по-разному и не надоедали пользователю своей однотипностью. В листинге 4 продемонстрирован скрипт `Sounds`.

Листинг 4 – Скрипт `Sounds`

```
public class Sounds : MonoBehaviour
{
    // массив игровых аудио эффектов
    public AudioClip[] sounds;

    private AudioSource audioSrc => GetComponent<AudioSource>();

    public void PlaySound(AudioClip clip, float volume = 1f, float p1 =
0.85f, float p2 = 1.2f)
    {
        // Пичинг аудиодорожки в случайном диапазоне
        audioSrc.pitch = Random.Range(p1, p2);
        // Проиграть 1 раз аудиоклип
        audioSrc.PlayOneShot(clip, volume);
    }
}
```

Скрипт `Player` наследуется от `Sounds`, что позволяет удобно вызывать нужные аудиодорожки в отвечающим за основной геймплей скрипте. Всего в игре используется 12 различных звуковых эффектов.

Помимо аудио эффектов в игре присутствует музыкальное сопровождение сцен. Пять разных композиций размещены на сцене главного меню, вступительной заставки, игрового процесса, сцены прокачки и сцены диалогов. Скрипт `Music` отвечает за включение фоновой музыки, его отличительная черта от `Sounds` в том, что он не изменяет тональность аудиоклипов, и использует другой `AudioSourse` [15] компонент со включенным режимом зацикливания звука для того, чтобы музыка постоянно играла на сцене. В листинге 5 продемонстрирован скрипт `Music`.

Листинг 5 – Скрипт `Music`

```
public class Music : MonoBehaviour
{
    public AudioClip scene_music;
```

```
    public AudioClip second_music;

    int volume;
    private AudioSource audioSrc => GetComponent<AudioSource>();

    void Start()
    {
        volume = PlayerPrefs.GetInt("ToggleMusic");
        PlayMusic(scene_music);
        PlayMusic(second_music);
    }

    void Update()
    {
        int new_volume = PlayerPrefs.GetInt("ToggleMusic");
        // отслеживание изменений настроек в главном меню
        if (volume != new_volume)
        {
            audioSrc.volume = 1 - new_volume;
            volume = new_volume;
        }

        // отслеживание изменений настроек игрового времени
        if (Time.timeScale == 0)
            audioSrc.volume = 0;
        if (Time.timeScale == 1)
            audioSrc.volume = 1 - volume;
    }

    public void PlayMusic(AudioClip clip)
    {
        audioSrc.PlayOneShot(clip);
        // Музыка играет на фоне, поэтому громкость 0.7
        audioSrc.volume = (1 - volume) * 0.7f;
    }
}
```

Вывод по четвертой главе

В соответствии с поставленными задачами, спроектированной архитектурой и файловой системой было успешно создано игровое приложение. Была описана реализация основных компонентов игры, представлены примеры кадров главного персонажа, листинги исходного кода и скриншоты пользовательского интерфейса.

5. ТЕСТИРОВАНИЕ СИСТЕМЫ

Для любой игры необходимо тестирование. Именно оно позволяет избежать многих багов и ошибок системы. Кроме того, через тестирование можно узнать мнение пользователя о балансе механик и визуальной составляющей игры.

5.1. Функциональное тестирование

В таблице 1 представлены результаты функциональных тестов, проведенных на реализованных скриптах в проекте «The Dark Path» на платформе Unity. В таблице приведены основные сценарии тестирования и их результаты.

Таблица 1 – Результаты функционального тестирования

№	Название теста	Действия	Результат	Тест пройден?
1	Запуск игры	Включить игру	Запускается сцена главного меню	Да
2	Начать игру	Нажать кнопку играть	Запускается вступительная заставка	Да
3	Проверка управления персонажем, движение вверх	1. Нажать кнопку играть 2. Пропустить начальную заставку	Если сверху свободно, и у персонажа есть выносливость, персонаж перемещается	Да
4	Проверка управления персонажем, движение вниз	1. Нажать кнопку играть 2. Пропустить начальную заставку	Если снизу свободно, и у персонажа есть выносливость, персонаж перемещается	Да
5	Проверка управления персонажем, атака	1. Нажать кнопку играть 2. Пропустить начальную заставку	Если есть выносливость, персонаж атакует	Да
6	Проверка получения опыта в игре	1. Нажать кнопку играть 2. Пропустить начальную заставку	Если персонаж жив, то счетчик очков увеличивается	Да
7	Проверка отображения выносливости	1. Нажать кнопку играть 2. Пропустить начальную заставку 3. Сделать атаку или перемещение	Полоска выносливости уменьшается	Да

Окончание таблицы 1

8	Проверка уменьшения здоровья	1. Нажать кнопку играть 2. Пропустить начальную заставку	Полоска здоровья уменьшается	Да
9	Проверка взаимодействия персонажа с окружением	1. Нажать кнопку играть 2. Пропустить начальную заставку 3. Врезаться в препятствие	Главный герой получает урон, здоровье уменьшается.	Да
10	Проверка убийства монстра	1. Нажать кнопку играть 2. Пропустить начальную заставку 3. атаковать монстра	Монстр убил, герой пробегает мимо и не получает урона.	Да
11	Проверка прохождения уровня	1. Нажать кнопку играть 2. Пропустить начальную заставку 3. Пройти уровень	Игрок переходит на сцену заставки	Да
12	Проверка установки паузы	1. Нажать кнопку играть 2. Пропустить начальную заставку 3. Поставить паузу	Игровое время останавливается, открывается меню паузы	Да

5.2. Юзабилити тестирование

Юзабилити-тестирование необходимо для проверки удобства продукта путем тестирования приложения конечным пользователем. В тестировании принимали участие девять игроков с разным игровым опытом. В ходе тестирования пользователям предлагалось запустить игру и поиграть в нее несколько минут. В рамках тестирования был незначительно изменен баланс игры, а также доработан пользовательский интерфейс:

1) уменьшен урон тьмой;
2) скорость восстановления выносливости увеличена;
3) добавлена пояснительная записка в меню прокачки персонажа.

Вывод по пятой главе

Система была протестирована на 12 функциональных тестах. Все тесты были успешно пройдены. Помимо этого, баланс системы был немного изменен после отклика тестовой группы пользователей.

ЗАКЛЮЧЕНИЕ

В ходе выполнения выпускной квалификационной работы была разработана игра «The Dark Path». Также были выполнены следующие поставленные задачи.

1. Проведен анализ предметной области. Проанализированы аналогичные проекты.

2. Составлены функциональные и нефункциональные требования к игре, а также представлена диаграмма вариантов использования и таблицы спецификации ВИ.

3. Спроектирована архитектура игры, описана файловая структура игры и диаграммы компонентов и деятельности.

4. Реализованы все компоненты игры с подробным описанием, скриншотами пользовательского интерфейса и листингами скриптов.

5. Проведено функциональное и юзабилити тестирование системы. Все функциональные тесты пройдены успешно.

К планам дальнейшего развития игры можно отнести:

1) улучшение графического оформления;

2) дальнейшее наполнение игры, то есть добавление боссов и новых уровней;

3) развитие сюжета игры;

4) добавление скинов и образов для главного героя, а также возможность улучшать снаряжение.

ЛИТЕРАТУРА

1. Статья «Чем объясняется популярность компьютерных игр?». [Электронный ресурс] URL: https://dzen.ru/a/X1s0UYJ5tAlGUcCV (дата обращения: 23.05.2024 г.).
2. Статья «Чем берут современные игры с пиксельной графикой». [Электронный ресурс] URL: https://club.dns-shop.ru/blog/t-64-videoigryi/74996-chem-berut-sovremennyie-igryi-s-pikselnoi-grafikoi/ (дата обращения: 23.05.2024 г.).
3. Инди игра «Pixel Castle Runner» [Электронный ресурс] URL: https://www.yiv.com/Pixel-Castle-Runner (дата обращения: 23.05.2024 г.).
4. Официальный сайт игры «Loop Hero» [Электронный ресурс] URL: https://www.loophero.com/ru (дата обращения: 23.05.2024 г.).
5. Официальная страницы игры «Blasphemous» [Электронный ресурс] URL: https://thegamekitchen.com/blasphemous/ (дата обращения: 23.05.2024 г.).
6. Среда разработки Unity. [Электронный ресурс] URL: https://unity.com/ru (дата обращения: 24.02.2024 г.).
7. Документация по C#. [Электронный ресурс] URL: https://docs.microsoft.com/ru-ru/dotnet/csharp/ (дата обращения: 24.02.2024 г.).
8. Руководство Unity. [Электронный ресурс] URL: https://docs.unity3d.com/ru/530/Manual/ (дата обращения: 24.02.2024 г.).
9. Инструменты Unity. [Электронный ресурс] URL: https://unity.com/ru/developer-tools (дата обращения: 24.02.2024 г.).
10. Бесплатные звуки без авторских прав. [Электронный ресурс] URL: https://zvukogram.com/ (дата обращения: 24.02.2024 г.).
11. Файл Figma. [Электронный ресурс] URL: https://www.figma.com/design/1VOplj6lFLSPAFeutampCk/Untitled?node-id=0-1&t=LLBEFVL1mXnd5QDU-0 (дата обращения: 24.02.2024 г.).

12. Canvas | Unity Documentation. [Электронный ресурс] URL: https://docs.unity3d.com/2022.2/Documentation/Manual/UICanvas.html (дата обращения: 24.02.2024 г.).

13. SceneManager | Unity Documentation. [Электронный ресурс] URL: https://docs.unity3d.com/ScriptReference/SceneManagement.SceneManager.html (дата обращения: 24.02.2024 г.).

14. CharacterController | Unity Documentation. [Электронный ресурс] URL: https://docs.unity3d.com/ScriptReference/CharacterController.html (дата обращения: 24.02.2024 г.).

15. AudioSourse | Unity Documentation. [Электронный ресурс] URL: https://docs.unity3d.com/ScriptReference/AudioSource.html (дата обращения: 24.02.2024 г.).

ПРИЛОЖЕНИЯ

Приложение А. Спецификация вариантов использования

Спецификация вариантов использования системы приведена в таблицах 1–6.

Таблица 1 – Спецификация вариантов использования «Начать игру»

Прецедент: Начать новую игру
ID: 1
Краткое описание: Пользователь начинает новую игру
Главные актеры: Пользователь
Второстепенные актеры: Нет
Предусловия: Пользователь находится в главном меню
Основной поток: 1. Игрок нажимает кнопку «Играть» 2. Игрок выбирает «новая игра»
Постусловия: 1. Игровой прогресс обнуляется 2. Запускается игровая сцена
Альтернативные потоки: Нет

Таблица 2 – Спецификация вариантов использования «Продолжить игру»

Прецедент: Продолжить игру
ID: 2
Краткое описание: Пользователь продолжает начатую игру
Главные актеры: Пользователь
Второстепенные актеры: Нет
Предусловия: Пользователь находится в главном меню
Основной поток: 1. Игрок нажимает кнопку «Играть» 2. Игрок выбирает «продолжить»
Постусловия: 1. Считывается игровой прогресс 2. Запускается игровая сцена
Альтернативные потоки: Нет

Таблица 3 – Спецификация вариантов использования «Настроить игру»

Прецедент: Изменить настройки звуков в игре
ID: 3
Краткое описание: Пользователь изменяет настройки звука в игре
Главные актеры: Пользователь
Второстепенные актеры: Нет
Предусловия: Пользователь находится в главном меню
Основной поток: Игрок изменяет положение кнопок отвечающих за звук и музыку в игре
Постусловия: 1. Изменяется настройка звука 2. измененное состояние записывается в память игры
Альтернативные потоки: Нет

Таблица 4 – Спецификация вариантов использования «Закрыть игру»

Прецедент: Закрыть игру
ID: 4
Краткое описание: Пользователь решает выйти из игрового приложения
Главные актеры: Пользователь
Второстепенные актеры: Нет
Предусловия: Пользователь находится в главном меню
Основной поток: Игрок нажимает кнопку «Выйти»
Постусловия: Игровое приложение закрывается
Альтернативные потоки: Нет

Таблица 5 – Спецификация вариантов использования «Управление»

Прецедент: Управление игровым персонажем
ID: 5
Краткое описание: Пользователь перемещает главного персонажа
Главные актеры: Пользователь
Второстепенные актеры: Нет

Предусловия: Пользователь находится на сцене игрового уровня
Основной поток: Игрок нажимает кнопку перемещения
Постусловия: Игровой персонаж перемещается
Альтернативные потоки: Нет

Таблица 6 – Спецификация вариантов использования «Прокачка»

Прецедент: Прокачка характеристик персонажа
ID: 6
Краткое описание: Пользователь тратит очки опыта на улучшение характеристики игрового персонажа
Главные актеры: Пользователь
Второстепенные актеры: Нет
Предусловия: Пользователь находится в меню прокачки
Основной поток: Игрок нажимает прокачивает выбранный параметр
Постусловия: 1. Вычитается опыт за прокачку 2. Уровень характеристики повышается
Альтернативные потоки: Нет

Приложение Б. Листинг основного игрового скрипта

Листинг 1 – Скрипт `Player`

```
using System.Collections;
using System.Collections.Generic;
using Unity.Burst.CompilerServices;
using UnityEngine;
using UnityEngine.SceneManagement;
using UnityEngine.UI;

public class Player : Sounds
{
    private CharacterController controller;
    [SerializeField] private GameObject DeathPanel;
    [SerializeField] private GameObject EndPanel;
    [SerializeField] private GameObject FirstPanel;
    [SerializeField] private GameObject SecondPanel;

    int volume; // Параметр звуковых эффектов

    public Animator animator; // Аниматор персонажа
    public float forwardSpeed = 5f; // Скорость движения вперед по оси X

    private int verticalLineToMove = 1; // Текущая вертикальная позиция
персонажа
    public float lineDistance = 2; // Расстояние между вертикальными лини-
ями
    public float verticalSpeed; // Скорость вертикального перемещения

    public bool isAttacking = false;
    private float attackTimer = 0f;
    private float attackDuration = 0.2f; // Длительность атаки

    public Image enduranceBar; // Индикатор выносливости
    public float endurance = 100f;
    public float enduranceRegeneration; // Скорость восстановления выносли-
вости

    public Image healthBar; // Индикатор здоровья
    public float health;
    public float healthDamage; // Урон, получаемый персонажем

    public int health_char; // Характеристика здоровья
    public int endurance_char; // Характеристика выносливости
    public int dexterity_char; // Характеристика ловкости
    public int power_char; // Характеристика силы
    public int faith_char; // Характеристика веры

    public int monster_damage; // Урон монстров
    public int obstacle_damage; // Урон каменных объектов
    public int progress; // Прогресс игрока

    void Start()
    {
        Time.timeScale = 1;

        progress = PlayerPrefs.GetInt("PlayerProgress");
        volume = PlayerPrefs.GetInt("ToggleSound");

        if (progress == 1)
            Education();
```

```
    controller = GetComponent<CharacterController>();

    health_char = PlayerPrefs.GetInt("health_char", 0);
    endurance_char = PlayerPrefs.GetInt("endurance_char", 0);
    dexterity_char = PlayerPrefs.GetInt("dexterity_char", 0);
    power_char = PlayerPrefs.GetInt("power_char", 0);
    faith_char = PlayerPrefs.GetInt("faith_char", 0);

    health = 100;
}

void Update()
{
    // Движение вверх
    if (SwipeController.swipeUp && verticalLineToMove < 2)
    {
        if (endurance > 0)
        {
            PlaySound(sounds[0], volume);
            verticalLineToMove++;
            animator.SetBool("Up", true);
            endurance -= 20f; // Расход выносливости на перемещение
            enduranceBar.fillAmount = endurance * 0.01f;
        }
    }

    // Движение вниз
    if (SwipeController.swipeDown && verticalLineToMove > 0)
    {
        if (endurance > 0)
        {
            PlaySound(sounds[0], volume);
            verticalLineToMove--;
            animator.SetBool("Down", true);
            endurance -= 20f; // Расход выносливости на перемещение
            enduranceBar.fillAmount = endurance * 0.01f;
        }
    }

    // Атака
    if (SwipeController.action && !isAttacking && endurance > 0)
    {
        PlaySound(sounds[1], volume);
        animator.SetBool("attak", true);
        isAttacking = true;
        endurance -= 40f; // Расход выносливости на атаку
        enduranceBar.fillAmount = endurance * 0.01f;
        attackTimer = 0f;
    }

    if (isAttacking)
    {
        attackTimer += Time.deltaTime;
        if (attackTimer >= attackDuration)
        {
            animator.SetBool("attak", false);
            isAttacking = false;
        }
    }

    // Восстановление выносливости
```

```
        if (!isAttacking && endurance < 100f)
        {
            endurance += (enduranceRegeneration * Time.deltaTime) * (1 + 2f
* endurance_char) / 2;
            enduranceBar.fillAmount = endurance * 0.01f;
        }

        // Получение урона
        health -= (healthDamage - (1 + 0.4f * faith_char) - (1 + 0.4f *
health_char)) * Time.deltaTime;
        healthBar.fillAmount = health * 0.01f;

        // Смерть
        if (health <= 0)
        {
            if (progress == 1)
                SceneManager.LoadScene(1);
            else
            {
                DeathPanel.SetActive(true);
                Time.timeScale = 0;
            }
        }
    }

    private void FixedUpdate()
    {
        // Движение вперед по оси X
        Vector3 direction = Vector3.right * forwardSpeed * Time.deltaTime;

        // Расчет новой позиции Y
        float verticalTargetY = (verticalLineToMove - 1) * lineDistance;
        float newY = Mathf.MoveTowards(transform.position.y, verticalTar-
getY, (verticalSpeed + (1 * dexterity_char)) * Time.deltaTime);

        // Создание новой позиции
        Vector3 newPosition = new Vector3(transform.position.x + direc-
tion.x, newY, transform.position.z);

        // Перемещение персонажа
        controller.Move(newPosition - transform.position);

        // Выключение анимаций движения после достижения целевой позиции
        if (Mathf.Approximately(transform.position.y, verticalTargetY))
        {
            animator.SetBool("Up", false);
            animator.SetBool("Down", false);
        }
    }

    // Столкновение
    private void OnTriggerEnter(Collider hit)
    {
        if (hit.gameObject.tag == "enemy")
        {
            if (isAttacking)
            {
                PlaySound(sounds[3], volume);
                Destroy(hit.gameObject);
                float currentExperience = PlayerPrefs.GetFloat("Experi-
ence", 0.0f);
```

```
                currentExperience += progress*5;
                PlayerPrefs.SetFloat("Experience", currentExperience);
            }
            else
            {
                PlaySound(sounds[2], volume);
                health -= monster_damage;
            }
        }

        if (hit.gameObject.tag == "obstacle")
        {
            if (isAttacking && power_char > 4)
            {
                PlaySound(sounds[3], volume);
                Destroy(hit.gameObject);
                float currentExperience = PlayerPrefs.GetFloat("Experi-
ence", 0.0f);
                currentExperience += progress * 10;
                PlayerPrefs.SetFloat("Experience", currentExperience);
            }
            else
            {
                PlaySound(sounds[2], volume);
                health -= obstacle_damage;
            }
        }

        if (hit.gameObject.tag == "finish")
        {
            PlayerPrefs.SetInt("PlayerProgress", progress + 1);
            PlayerPrefs.Save();

            Time.timeScale = 0;
            EndPanel.SetActive(true);
        }
    }
    public void NewGame()
    {
        DeathPanel.SetActive(false);
        SceneManager.LoadScene(progress);
    }

    public void ToFire()
    {
        SceneManager.LoadScene(8);
    }

    public void ToMenue()
    {
        SceneManager.LoadScene(0);
    }

    public void EndLevel()
    {
        SceneManager.LoadScene(progress+4);
    }

    public void Education()
    {
        Time.timeScale = 0;
```

```
        FirstPanel.SetActive(true);
    }
    public void EducationNext()
    {
        FirstPanel.SetActive(false);
        SecondPanel.SetActive(true);
    }
    public void EducationEnd()
    {
        SecondPanel.SetActive(false);
        Time.timeScale = 1;
    }
}
```

Приложение В. Листинг скрипта перемещения персонажа

Листинг 1 – Скрипт `SwipeController`

```
public class SwipeController : MonoBehaviour
{
    public static bool tap, swipeLeft, swipeRight, swipeUp, swipeDown, ac-
tion, pause;
    private bool isDraging = false;
    private Vector2 startTouch, swipeDelta;

    private void Update()
    {
        tap = swipeDown = swipeUp = swipeLeft = swipeRight = action = pause
= false;

        // ПК-версия
        if (Input.GetMouseButtonDown(0))
        {
            tap = true;
            isDraging = true;
            startTouch = Input.mousePosition;
        }
        else if (Input.GetMouseButtonUp(0))
        {
            isDraging = false;
            Reset();
        }

        // Мобильная версия
        if (Input.touches.Length > 0)
        {
            if (Input.touches[0].phase == TouchPhase.Began)
            {
                tap = true;
                isDraging = true;
                startTouch = Input.touches[0].position;
            }
            else if (Input.touches[0].phase == TouchPhase.Ended || In-
put.touches[0].phase == TouchPhase.Canceled)
            {
                isDraging = false;
                Reset();
            }
        }

        // Клавиатурный ввод
        if (Input.GetKeyDown(KeyCode.W) || Input.GetKeyDown(KeyCode.UpAr-
row))
        {
            swipeUp = true;
            Reset();
        }
        else if (Input.GetKeyDown(KeyCode.S) || Input.Get-
KeyDown(KeyCode.DownArrow))
        {
            swipeDown = true;
            Reset();
        }
        else if (Input.GetKeyDown(KeyCode.A) || Input.Get-
KeyDown(KeyCode.LeftArrow))
        {
            swipeLeft = true;
```

```
            Reset();
        }
        else if (Input.GetKeyDown(KeyCode.D) || Input.Get-
KeyDown(KeyCode.RightArrow))
        {
            swipeRight = true;
            Reset();
        }
        else if (Input.GetKeyDown(KeyCode.Space))
        {
            action = true;
        }
        else if (Input.GetKeyDown(KeyCode.Escape))
        {
            pause = true;
        }

        // Расчет дистанции
        swipeDelta = Vector2.zero;
        if (isDraging)
        {
            if (Input.touches.Length > 0)
                swipeDelta = Input.touches[0].position - startTouch;
            else if (Input.GetMouseButton(0))
                swipeDelta = (Vector2)Input.mousePosition - startTouch;
        }

        // Проверка на пройденность расстояния
        if (swipeDelta.magnitude > 100)
        {
            // Определение направления
            float x = swipeDelta.x;
            float y = swipeDelta.y;
            if (Mathf.Abs(x) > Mathf.Abs(y))
            {
                if (x < 0)
                    swipeLeft = true;
                else
                    swipeRight = true;
            }
            else
            {
                if (y < 0)
                    swipeDown = true;
                else
                    swipeUp = true;
            }

            Reset();
        }
    }

    private void Reset()
    {
        startTouch = swipeDelta = Vector2.zero;
        isDraging = false;
    }
}
```