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 «Электронные вычислительные машины»

ДОПУСТИТЬ К ЗАЩИТЕ
Заведующий кафедрой ЭВМ
_____Д. В. Топольский
«___»_____ 2022 г.

РАЗРАБОТКА КОМПЬЮТЕРНОЙ ИГРЫ В ЖАНРЕ RPG С УСОВЕРШЕНСТВОВАННОЙ СИСТЕМОЙ АНАЛИЗА ИГРОВЫХ ПЕРСОНАЖЕЙ

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
К ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ
ЮУРГУ-090301.2022.212 ПЗ ВКР

Руководитель работы,
к.т.н., доцент каф. ЭВМ
_____В. А. Парасич
«___»_____2022 г.

Автор работы,
студент группы КЭ-405
_____В. А. Калимуллин
«___»_____2022 г.

Нормоконтролёр,
к.пед.н., доцент каф. ЭВМ
_____ М. А. Алтухова
«___»_____2022 г.

Челябинск 2022

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 «Электронные вычислительные машины»

УТВЕРЖДАЮ
Заведующий кафедрой ЭВМ
_____ Д. В. Топольский
«___»_____2022 г.

ЗАДАНИЕ

на выпускную квалификационную работу бакалавра

студенту группы КЭ-405
Калимуллину Владиславу Айдаровичу,
обучающемуся по направлению
09.03.01 «Информатика и вычислительная техника»

Тема работы: «Разработка компьютерной игры в жанре RPG с усовершенствованной системой анализа игровых персонажей» утверждена приказом по университету от «12» декабря 2021 г. № 308/141

Срок сдачи студентом законченной работы: 1 июня 2022 г.

Исходные данные к работе.

1) Платформа Unity / Unity Technologies. – Текст. Изображение (подвижное ; двухмерное) : электронные // Unity : [официальный сайт]. – URL: https://unity.com/ru/products/unity-platform (Дата обращения: 30.04.2022)

2) Unity Documentation : [официальный сайт] / Unity Technologies. – URL: https://docs.unity3d.com/Manual/index.html (Дата обращения: 30.04.2022) – Текст. Изображение (неподвижное ; двухмерное) : электронные.

Перечень подлежащих разработке вопросов:

1) проанализировать главные критерии жанра RPG;

2) провести исследование предпочтений целевой аудитории игры;

3) проанализировать существующие игры в жанре RPG;

4) определить вид, платформу и среду разработки игры;

5) разработать компьютерную игру в жанре RPG;

6) провести тестирование разработанной игры.

Дата выдачи задания: 1 декабря 2021 г.

Руководитель работы _____/*В. А. Парасич*/

Студент _____/*В. А. Калимуллин*/

КАЛЕНДАРНЫЙ ПЛАН

Этап	Срок сдачи	Подпись руководителя
Введение и обзор литературы. Анализ предметной области с целью выявления аналогичных проектов. Анализ основных технических решений	07.02.2022	
Составление функциональных и нефункциональных требований. Составление архитектуры и описание данных	07.03.2022	
Реализация приложения	04.04.2022	
Тестирование, отладка, эксперименты	10.05.2022	
Компоновка текста работы и сдача на нормоконтроль	16.05.2022	
Подготовка презентации и доклада	24.05.2022	

Руководитель работы _____/*В. А. Парасич*/

Студент _____/*В. А. Калимуллин*/

АННОТАЦИЯ

В. А. Калимуллин. Разработка компьютерной игры в жанре RPG с усовершенствованной системой анализа игровых персонажей. – Челябинск: ФГАОУ ВО «ЮУрГУ (НИУ)», ВШЭКН; 2022, 41 с., 19 ил., библиогр. список – 11 наим.

В рамках выпускной квалификационной работы производится разработка компьютерной игры в жанре RPG.

Проводится анализ процесса разработки компьютерной игры, используемых инструментов. Приведены данные о важности для игроков различных аспектов жанра RPG. Определены основные критерии создания игр в данном жанре, выбора среды разработки. Результатом работы является разработанный прототип компьютерной игры в жанре RPG с усовершенствованной системой анализа игровых персонажей.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Компьютерные игры, будучи одной из самых быстроразвивающихся отраслей человеческой деятельности, на сегодняшний день насчитывают более трех миллиардов игроков по всему миру [1]. Их популярность можно объяснить желанием людей удовлетворить потребность в развлечениях. Одним из самых известных и популярных игровых жанров является жанр «RPG». Отправными точками для данного жанра можно считать текстовые игры, выходившие в конце XX века, а также настольную ролевую игру Dungeons & Dragons, разработанную в 1970 году.

RPG (Role-Playing Game) – жанр компьютерных игр, основанный на элементах игрового процесса традиционных настольных ролевых игр. В ролевой игре игрок управляет одним или несколькими персонажами, каждый из которых описан набором численных характеристик, списком способностей и умений. Игрок отыгрывает собственного персонажа, руководствуясь при этом характером своей роли и внутренними убеждениями персонажа в рамках игровых реалий. Действия игроков составляют сюжет игры [2;3].

Таким образом, основным критерием игр данного жанра является возможность отыгрывать роль своего персонажа путем развития его характеристик и умений, а также принятия решений, влияющих на сюжет игры.

Целью выпускной квалификационной работы является разработка прототипа компьютерной игры в жанре RPG с усовершенствованной системой анализа игровых персонажей. Для достижения поставленной цели необходимо решить следующие задачи:

1) проанализировать главные критерии жанра RPG;

2) провести исследование предпочтений целевой аудитории игры;

3) проанализировать существующие игры в жанре RPG;

4) определить вид, платформу и среду разработки игры;

5) разработать компьютерную игру в жанре RPG;

6) провести тестирование разработанной игры.

1 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

Опрос игроков, проведенный командой Larian Studios [4], определил предпочитаемые игроками аспекты данного жанра (рисунок 1). При проведении опроса рассматривались следующие аспекты:

1) сюжет;

2) боевая система;

3) развитие персонажа;

4) реиграбильность;

5) испытание навыков игрока;

6) система выпадения предметов;

7) ремесло;

8) кооператив;

9) сражения между игроками.

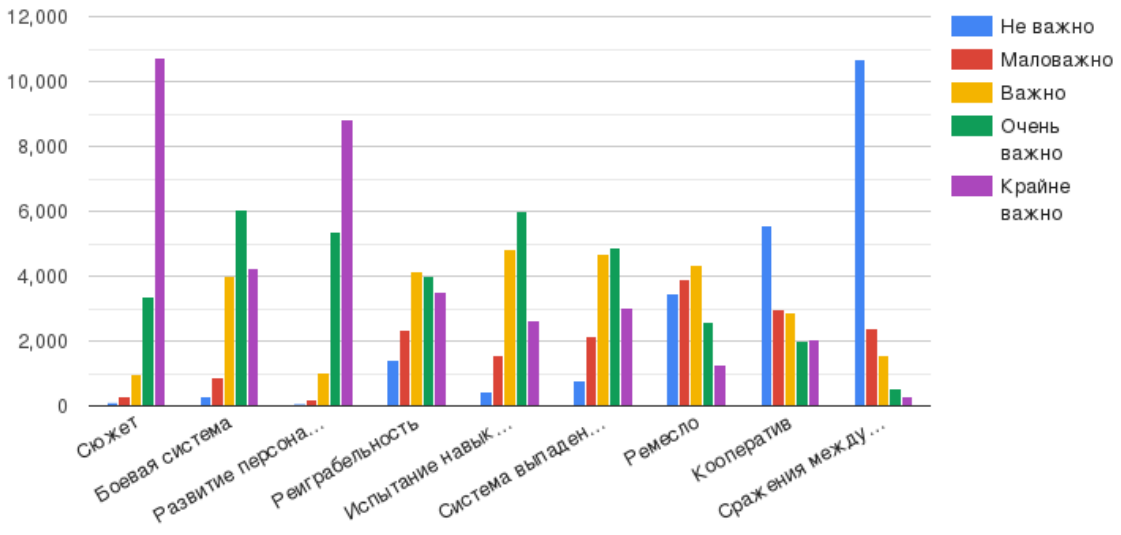

Рисунок 1 – Важность аспектов жанра

1.1 Обзор аналогов

Kenshi – это компьютерная ролевая игра с открытым миром и видом сверху (рисунок 2). На создание этой игры потребовалось 12 лет. Первые 6 лет разработка велась одним человеком, однако позже к проекту присоединились программист, писатель и 3D-художник [5]. Сюжет в игре отсутствует, историю создает игрок, благодаря свободе действий. Бои в данной игре происходят в реальном времени. Персонаж и противник автоматически атакуют друг друга при отдаче соответствующей команды. Развитие персонажа выражено прокачкой характеристик при выполнении определенных действий. Ввиду отсутствия сюжета игры, игрок сам ставит себе цель. Реиграбельность зависит от воображения игрока. Боевая система не испытывает навыки игрока. Из противника выпадают экипированные им предметы. Присутствует возможность создания предметов. Игра одиночная. Стоимость игры – 515 рублей.

Рисунок 2 – Kenshi

Path of Exile – это изометрическая сетевая ролевая игра, разработанная компанией Grinding Gear Games (рисунок 3). Игра победила в номинации «Ролевая игра года» (2013), в 2020 году выиграла премию BAFTA 2020 года в номинации «Развивающаяся игра» [6;7]. Сюжет линейный, без возможности выбора. Бои в данной игре происходят в реальном времени. Боевая система

представляет собой набор способностей, которые персонаж может экипировать. Способности применяются при нажатии соответствующей клавиши. Развитие персонажа выражено развитием характеристик и навыков персонажа путем прокачки соответствующих ячеек в древе навыков. Большое количество навыков и способностей приводит к высокой реиграбельности. Боевая система не испытывает навыки игрока напрямую. Высокая сложность игры требует глубокого понимания при развитии персонажа. Сложность выражается в эффективности сочетания характеристик и способностей. При неправильной прокачке игроки предпочитают начать игру заново. Выпадение предметов случайное, каждый предмет имеет базовые показатели, а также случайные модификаторы. Данная система приводит к получению большого количества предметов с разными параметрами. Присутствует возможность создания предметов, изменения их модификаторов и параметров. Игроки могут играть в группе до 6 человек. Игра в группе усиливает противников. Присутствует возможность сражения между игроками, но данный режим игры непопулярен ввиду низкой проработанности и плохого баланса. Игра бесплатная, присутствует внутриигровой магазин.

Рисунок 3 – Path of Exile

Серия игр Dark Souls – это компьютерные игры в жанре RPG, разработанные компанией From Software (рисунок 4). Серия состоит из трех игр, имеющих между собой минимальные отличия. Каждая игра получила высокие оценки от критиков и игроков [8;10]. Сюжет в игре простой, нелинейность выражена разными концовками. Высокая степень проработки истории мира. Бои в данной игре происходят в реальном времени. Боевая система представляет собой сочетание возможностей атаки и защиты. Присутствует несколько видов физической атаки, магические способности, возможность блока и уклонения. Развитие персонажа выражено развитием характеристик путем накопления определенного количества опыта при победе над противником. Разнообразие снаряжения, способностей, характеристик и концовок приводит к высокой реиграбельности. Высокая сложность игры требует от игрока высокого уровня навыков для победы над противником. Сложность заключается в необходимости своевременных уклонений от атак противника, контроле выносливости персонажа и ограничении возможности восстановления здоровья. При поражении, персонаж теряет весь накопленный опыт и возвращается к последнему месту отдыха, при этом все враги восстанавливаются. Каждый противник имеет фиксированный набор выпадающих предметов. Предметы выпадают с определенным шансом. Отсутствует возможность создания предметов. Кооператив и сражения между игроками представляют собой возможность призвать другого игрока в свой мир в фиксированных местах. Призванный игрок находится в игре до выполнения определенной задачи, после чего исчезает. Стоимость Dark Souls – 1199 рублей, Dark Souls II – 1699 рублей, Dark Souls III – 3499 рублей.

Рисунок 4 – Dark Souls III

1.2 Анализ основных технологических решений

При создании прототипа игры должна быть определена среда разработки. Необходимо сделать выбор между использованием готового игрового движка и созданием собственного решения. Преимуществами готового игрового движка являются:

1) универсальность;

2) наличие готовых плагинов;

3) наличие готовых шаблонов.

Преимуществом самостоятельно разработанного движка является узкая направленность на данный проект, что приводит к повышению производительности высокобюджетных проектов. Таким образом, ввиду большей скорости разработки, наличия готовых шаблонов и отсутствия значимого преимущества самостоятельно разработанной среды, использование готового решения является наиболее предпочтительным.

При выборе движка, рассмотрим статистику самых популярных решений, основываясь на количестве выпускаемых проектов (рисунок 5).

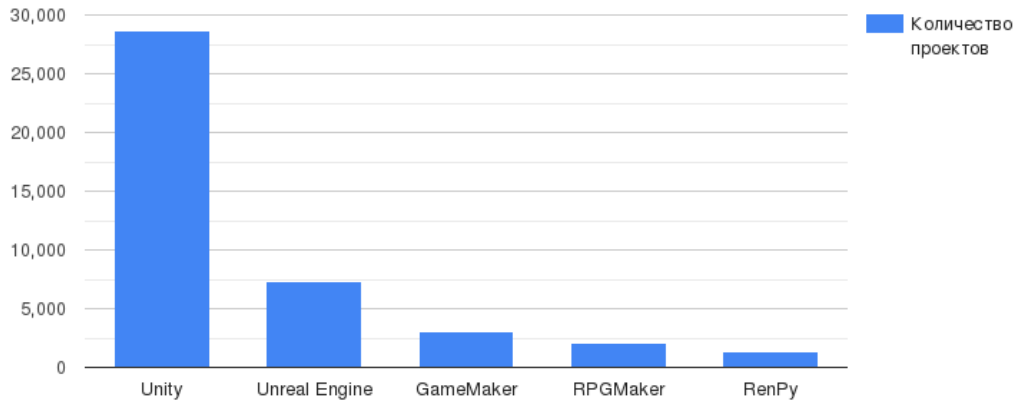

Рисунок 5 – Популярные среды разработки

Движок RenPy предназначен для создания визуальных новелл, ввиду чего не подходит для рассматриваемой предметной области. RPGMaker и GameMaker предназначены для разработчиков без навыков программирования. Таким образом, ввиду простоты использования, данные среды обладают ограниченным функционалом и не предназначены для крупных проектов.

Выбор между Unity и Unreal Engine делается с учетом требований каждого конкретного проекта. Игры категории AAA предпочтительно создавать на Unreal Engine ввиду больших возможностей редактора и гибкой архитектуры, однако движок требователен к техническому обеспечению и навыкам специалистов. Также в данной среде отсутствует возможность создания 2D проектов. Движок Unity в свою очередь больше подходит для проектов в стиле 2D и 2.5D, обладает большим количеством шаблонов и простотой разработки по сравнению с Unreal Engine за счет использования языка C#. Была выбрана среда разработки Unity ввиду низкой масштабности проекта, универсальности движка, наличия готовых шаблонов и меньших системных требований.

1.3 Вывод по первой главе

Полученные данные позволяют сделать вывод, что самыми важными аспектами жанра являются сюжет и развитие персонажа. Проработка данных элементов является первостепенной задачей создания продукта в жанре RPG. Возможность совместной игры и поединки между игроками не являются приоритетными и могут быть упрощены или устранены в пользу более важных аспектов. Также, игроки делают выбор в пользу изометрического вида и вида от третьего лица, вид от первого лица является менее популярным.

2 ОПРЕДЕЛЕНИЕ ТРЕБОВАНИЙ

2.1 Функциональные требования

Исходя из концепции игры и результатов анализа предметной области, были выделены следующие требования:

1) пользователь должен иметь возможность управлять передвижением персонажа;

2) пользователь должен иметь возможность взаимодействовать с окружающими предметами и противниками;

3) пользователь должен иметь возможность повышать характеристики персонажа;

4) пользователь должен иметь возможность сражаться с противниками;

5) пользователь должен иметь возможность менять экипировку персонажа;

6) пользователь должен иметь возможность хранить экипировку в инвентаре.

Диаграмма вариантов использования представлена на рисунке 6:

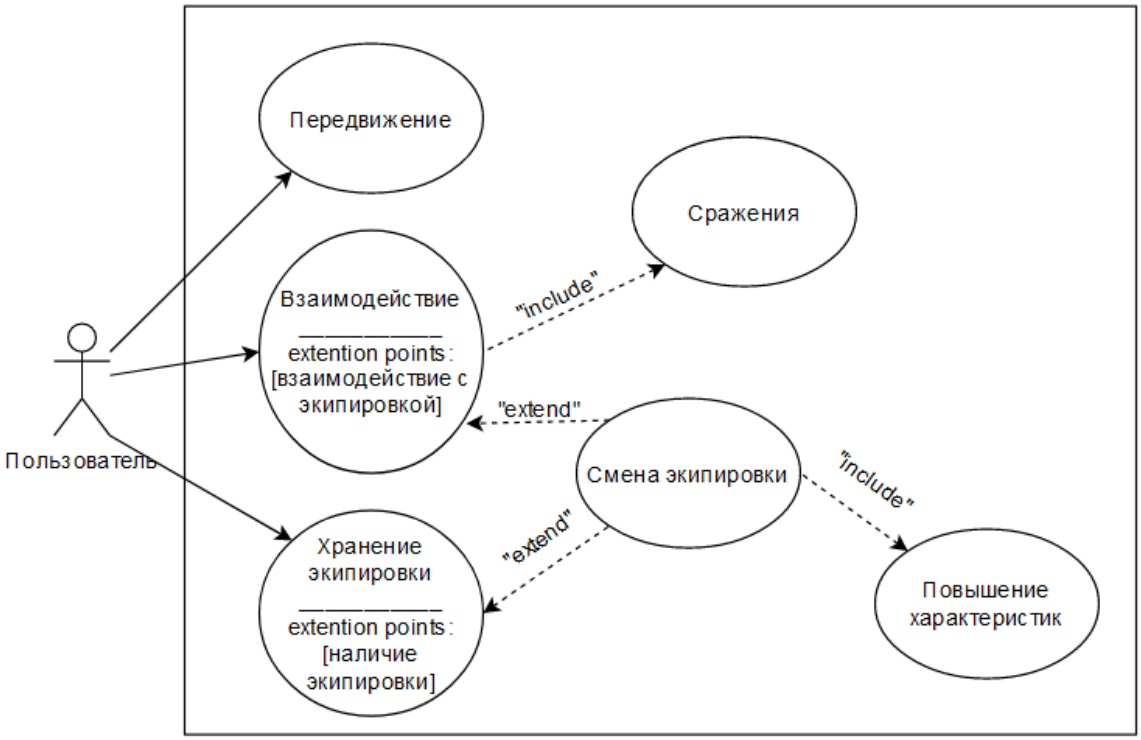

Рисунок 6 – Диаграмма вариантов использования

2.2 Нефункциональные требования

1) приложение должно работать на операционной системе Windows;

2) приложение должно быть реализовано средствами языка программирования C# и среды разработки Unity 2021.3.3f1.

Минимальными требованиями для работы в Unity являются [11]:

1) операционная система Windows 7 (SP1+), Windows 10 или Windows 11;

2) центральный процессор с поддержкой набора инструкций SSE2;

3) графический адаптер с поддержкой DX10, DX11, DX12;

4) Microsoft Visual Studio не старше версии 2015.

Для создания трехмерных моделей, построения скелета и настройки анимации используется программное обеспечение Blender.

3 ПРОЕКТИРОВАНИЕ

3.1 Архитектура предлагаемого решения

Разрабатываемое приложение состоит из следующих функциональных блоков (рисунок 7):

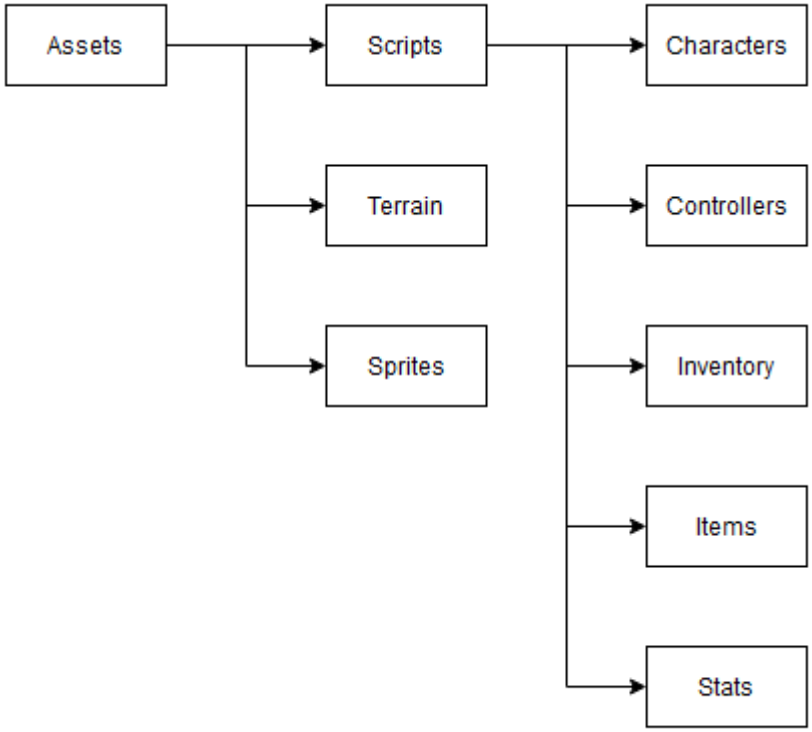

Рисунок 7 – Функциональные блоки приложения

3.2 Описание данных

Блок Terrain является модулем настройки формы и размера ландшафта, а также применения к ландшафту особых эффектов, в том числе расстановка травы и деревьев. Блок Sprites используется для хранения двухмерных изображений. В блоке Script хранятся функции и классы, разделенный на 5 категорий. Блок Characters хранит модели персонажей, а также их параметры анимации и связанные модели экипировки. В блоке Controllers хранятся функции, управляющие передвижением персонажей, параметрами движения и поворота камеры, поиском кратчайшего пути, поведением противников. В

блоке Inventory хранятся классы, определяющие взаимодействие с инвентарем. В блоке Items хранятся функции, от которых зависит взаимодействие персонажа с предметами и экипировкой. В блоке Stats хранятся классы, определяющие характеристики персонажей, модификаторы характеристик.

4 РЕАЛИЗАЦИЯ

4.1 Реализация приложения

Приложение представляет собой игру в жанре RPG в открытом мире. Главный герой – неизвестный житель деревни. Он замечает, что у старосты деревни есть проблема и решает узнать, может ли помочь старосте. Ответ старосты на вопрос главного героя будет зависеть от экипировки игрока и мнении неигрового персонажа об игроке. Без достаточного количества экипировки или низкой степени доверия, староста не согласится принять помощь от игрока. Соответственно, староста не просто выдает задание, а использует усовершенствованную систему анализа персонажа для подтверждения готовности игрока к выполнению задания. Данная экипировка повышает характеристики игрока, являясь тем самым средством развития персонажа. Таким образом, игрок должен найти в игровом мире экипировку, после чего староста согласится принять помощь главного героя и скажет, что деревне угрожают четыре скелета. Игрок, используя найденную экипировку, должен победить скелетов, помогая старосте, после чего отчитаться и получить похвалу старосты.

Приложение было реализовано путем применения классов, написанных на языке программирования C#.

Класс CameraController – служит для привязки камеры к модели персонажа, управление приближением и поворотом камеры путем считывания ввода клавиш и кнопок мыши. Проверка выполняется каждый кадр. Положение камеры корректируется исходя из текущих действий персонажа. Метод Update отвечает за приближение и отдаление камеры, поворот камеры по горизонтали.

Листинг 1 – метод Update класса CameraController

```
 void Update()
    {
        currentZoom -= Input.GetAxis("Mouse ScrollWheel") * zoomSpeed;
        currentZoom = Mathf.Clamp(currentZoom, minZoom, maxZoom);
        currentYaw += Input.GetAxis("Horizontal") * yawSpeed *
Time.deltaTime;    }
```

Метод LateUpdate отвечает за движение камеры за персонажем.

Листинг 2 – метод LateUpdate класса CameraController

```
void LateUpdate ()
    {
        transform.position = target.position - offset*currentZoom;
        transform.LookAt(target.position + Vector3.up * pitch);
        transform.RotateAround(target.position, Vector3.up, currentYaw);
    }
```

Класс PlayerMotor – определяет передвижение персонажа. Позволяет следовать за выбранным объектом, поворачиваться к выбранному объекту лицом, предотвращает столкновения. Метод Update используется для следования за выбранной целью.

Листинг 3 – метод Update класса PlayerMotor

```
void Update()
    {
        if(target != null)
        {
            agent.SetDestination(target.position);
            FaceTarget();
        }
    }
```

Метод MoveToPoint отвечает за движение в выбранную точку.

Листинг 4 – метод MoveToPoint класса PlayerMotor

```
public void MoveToPoint (Vector3 point)
    {
        agent.SetDestination(point);
    }
```

Метод FollowTarget предотвращает столкновение и прохождение моделей персонажей друг через друга.

Листинг 5 – метод FollowTarget класса PlayerMotor

```
public void FollowTarget (Interactable newTarget)
    {
        agent.stoppingDistance = newTarget.radius*0.8f;
        agent.updateRotation = false;
        target = newTarget.interactionTransform;
    }
```

Метод StopFollowingTarget используется для снятия выделения с цели.

Листинг 6 – метод StopFollowingTarget класса PlayerMotor

```
public void StopFollowingTarget()
    {
        agent.stoppingDistance = 0f;
        agent.updateRotation = true;
        target = null;
    }
```

Метод FaceTarget отвечает за поворот головы персонажа в сторону цели.

Листинг 7 – метод FaceTarget класса PlayerMotor

```
void FaceTarget()
    {
        Vector3 direction = (target.position -
transform.position).normalized;
        Quaternion lookRotation = Quaternion.LookRotation(new
Vector3(direction.x, 0f, direction.z));
        transform.rotation = Quaternion.Slerp(transform.rotation,
lookRotation, Time.deltaTime * 5f);
    }
```

Класс PlayerController – служит для управления передвижением персонажа, позволяет выбирать точку перемещения или объект следования (рисунок 8).

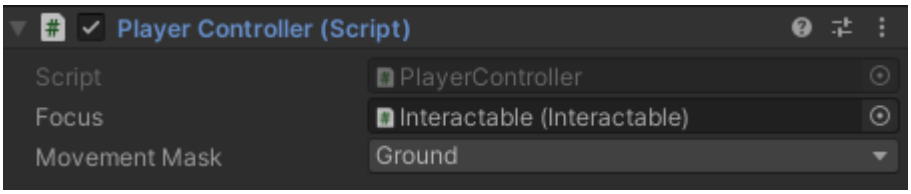

Рисунок 8 – Объект Interactable выбран в качестве объекта следования

Метод Update позволяет перемещаться в выбранную точку с помощью левой кнопки мыши и следовать за выбранным объектом с помощью правой кнопки мыши.

Листинг 8 – метод Update класса PlayerController

```
void Update()
    {
        if (EventSystem.current.IsPointerOverGameObject())
            return;
        if (Input.GetMouseButtonDown(0))
         {
             Ray ray = cam.ScreenPointToRay(Input.mousePosition);
             RaycastHit hit;
             if(Physics.Raycast(ray, out hit, 100, movementMask))
             {
                 motor.MoveToPoint(hit.point);
                RemoveFocus();
             }
         }

         if (Input.GetMouseButtonDown(1))
         {
             Ray ray = cam.ScreenPointToRay(Input.mousePosition);
             RaycastHit hit;
             if (Physics.Raycast(ray, out hit, 100))
             {
                Interactable interactable =
hit.collider.GetComponent<Interactable>();
                if(interactable != null)
                {     SetFocus(interactable);      }
             }
         }
    }
```

Метод SetFocus используется для выбора цели.

Листинг 9 – метод SetFocus класса PlayerController

```
void SetFocus (Interactable newFocus)
    {
        if (newFocus != focus)
        {
            if (focus != null)
                focus.OnDefocused();
            focus = newFocus;
            motor.FollowTarget(newFocus);
        }

        newFocus.OnFocused(transform);
    }
```

Метод RemoveFocus позволяет снять выделение с выбранной цели.

Листинг 10 – метод RemoveFocus класса PlayerController

```
void RemoveFocus()
    {
        if (focus != null)
            focus.OnDefocused();
        focus = null;
        motor.StopFollowingTarget();
    }
```

Класс EnemyController – управляет искусственным интеллектом противников. Отвечает за передвижение, степень агрессии, переход в режим агрессии (рисунок 9).

Рисунок 9 – Визуализация радиуса агрессии противника

Метод Update отвечает за нападение на персонажа, переход в режим агрессии и проверку характеристик игрока.

Листинг 11 – метод Update класса EnemyController

```
void Update()
	{
		float distance = Vector3.Distance(target.position, 
transform.position);
		if (distance <= lookRadius)
		{
			agent.SetDestination(target.position);
			if (distance <= agent.stoppingDistance)
			{
				CharacterStats targetStats = 
target.GetComponent<CharacterStats>();
				if (targetStats != null)
				{
					combat.Attack(targetStats);
				}
				FaceTarget();
			}
		}
	}
```

Метод FaceTarget отвечает за поворот противника лицом к игроку.

Листинг 12 – метод FaceTarget класса EnemyController

```
void FaceTarget()
	{
		Vector3 direction = (target.position - 
transform.position).normalized;
		Quaternion lookRotation = Quaternion.LookRotation(new 
Vector3(direction.x, 0, direction.z));
		transform.rotation = Quaternion.Slerp(transform.rotation, 
lookRotation, Time.deltaTime * 5f);
	}
```

Метод OnDrawGizmosSelected отвечает за визуализацию радиуса агрессии противника в редакторе.

Листинг 13 – метод OnDrawGizmosSelected класса EnemyController

```
void OnDrawGizmosSelected()
	{
		Gizmos.color = Color.red;
		Gizmos.DrawWireSphere(transform.position, lookRadius);
	}
```

Класс Inventory – представляет собой список, хранящий предметы, содержащихся в инвентаре персонажа. Инвентарь может быть лишь один. Метод Add добавляет предмет в инвентарь при наличии свободного места.

Листинг 14 – метод Add класса Inventory

```
public bool Add (Item item)
    {
        if(!item.isDefaultItem)
        {
            if (items.Count >= space)
            {
                return false;
            }
            items.Add(item);
            if (onItemChangeCallback != null)
                onItemChangeCallback.Invoke();
        }
        return true;
    }
```

Метод Remove удаляет предмет из инвентаря.

Листинг 15 – метод Remove класса Inventory

```
public void Remove(Item item)
    {
        items.Remove(item);
        if (onItemChangeCallback != null)
            onItemChangeCallback.Invoke();
    }
```

InventoryUI – отвечает за интерфейс инвентаря (рисунок 10). Позволяет открывать и закрывать инвентарь.

Рисунок 10 – Интерфейс инвентаря

Метод Update отвечает за открытие и закрытие инвентаря.

Листинг 16 – метод Update класса InventoryUI

```
void Update()
	{
		if (Input.GetButtonDown("Inventory"))
		{
			inventoryUI.SetActive(!inventoryUI.activeSelf);
		}
	}
```

Метод UpdateUI добавляет предметы из списка в слоты графического интерфейса инвентаря.

Листинг 17 – метод UpdateUI класса InventoryUI

```
void UpdateUI()
	{
		for (int i = 0; i < slots.Length; i++)
		{
			if (i < inventory.items.Count)
			{
				slots[i].AddItem(inventory.items[i]);
			}
			else
			{
				slots[i].ClearSlot();
			}
		}
	}
```

Класс Slot – управляет слотами для предметов инвентаря. Устанавливает изображения полученным предметам, позволяет избавиться от предмета. Метод AddItem устанавливает предмет в ячейку инвентаря, присваивает изображение и активирует кнопку удаление предмета.

Листинг 18 – метод AddItem класса Slot

```
public void AddItem(Item newItem)
	{
		item = newItem;
		icon.sprite = item.icon;
		icon.enabled = true;
		removeButton.interactable = true;
	}
```

Метод ClearSlot очищает слот инвентаря, удаляет изображение, отключает кнопку удаления предмета.

Листинг 19 – метод ClearSlot класса Slot

```
public void ClearSlot()
	{
		item = null;
		icon.sprite = null;
		icon.enabled = false;
		removeButton.interactable = false;
	}
```

Метод OnRemoveButton удаляет предмет при нажатии кнопки удаления.

Листинг 20 – метод OnRemoveButton класса Slot

```
public void OnRemoveButton()
	{
		Inventory.instance.Remove(item);
	}
```

Метод UseItem позволяет использовать предмет, удаляя его при этом из инвентаря.

Листинг 21 – метод UseItem класса Slot

```
public void UseItem()
	{
		if (item != null)
		{
			item.Use();
		}
	}
```

Класс Equipment – используется для управления снаряжением, определения характеристик снаряжения (рисунок 11).

Рисунок 11 – Экипировка снаряжения из инвентаря

Метод Use позволяет экипировать предмет, убирая его из инвентаря.

Листинг 22 – метод Use класса Equipment

```
public override void Use()
	{
		base.Use();
		EquipmentManager.instance.Equip(this);
		RemoveFromInventory();
	}
```

Item – определяет основные параметры предметов, позволяет использовать и избавляться от предмета. Метод Use позволяет использовать предмет. Переопределяется дочерними классами.

Листинг 23 – метод Use класса Item

```
public virtual void Use()
	{
		Debug.Log("Using " + name);
	}
```

Метод RemoveFromInventory позволяет удалить текущий предмет из инвентаря.

Листинг 24 – метод RemoveFromInventory класса Item

```
public void RemoveFromInventory()
	{
		Inventory.instance.Remove(this);
	}
```

Класс ItemPickup – позволяет поднимать предметы. Метод Interact представляет собой взаимодействие с предметом. Вызывает метод PickUp. Переопределяется дочерними классами. Метод PickUp позволяет добавить предмет в инвентарь, удаляя его при этом из игрового мира.

Листинг 25 – методы Interact и PickUp класса ItemPickup

```
 public override void Interact()
    {
        PickUp();
    }
 void PickUp()
    {
        bool wasPickedUp = Inventory.instance.Add(item);
        if (wasPickedUp)
            Destroy(gameObject);
    }
```

Класс Stat – используется для изменения характеристик. Позволяет изменить значение характеристик, добавить к ним модификатор. Метод GetValue позволяет получить измененное модификаторам значение характеристики.

Листинг 26 – метод GetValue класса Stat

```
public int GetValue()
	{
		int finalValue = baseValue;
		modifiers.ForEach(x => finalValue += x);
		return finalValue;
	}
```

Метод AddModifier позволяет добавить модификатор. Метод RemoveModifier позволяет снять модификатор с характеристики.

Листинг 27 – методы AddModifier и RemoveModifier класса Stat

```
public void AddModifier(int modifier)
	{
		if (modifier != 0)
			modifiers.Add(modifier);
	}
public void RemoveModifier(int modifier)
	{
		if (modifier != 0)
			modifiers.Remove(modifier);
	}
```

Класс CharacterStats – отвечает за максимальное и текущее количество очков жизни персонажа, его уровень атаки и защиты, силы, ловкости, а также показателя внешнего вида снаряжения. Позволяет рассчитывать получаемый урон.

Метод TakeDamage рассчитывает получаемый персонажем урон, вычитает полученное значение из текущего количества очков жизни.

Листинг 28 – метод TakeDamage класса CharacterStats

```
public void TakeDamage(int damage, int enemyLook)
	{
		damage -= (armor.GetValue() +
dex.GetValue())*look.GetValue()/enemyLook;
		damage = Mathf.Clamp(damage, 0, int.MaxValue);
		currentHealth -= damage;
		Debug.Log(transform.name + " takes " + damage + " damage.");
		if (OnHealthChanged != null)
		{
			OnHealthChanged(maxHealth, currentHealth);
		}
		if (currentHealth <= 0)
		{
			Die();
		}
	}
```

Метод Die отвечает за падение очков жизни до нуля, используется как основа для переопределения дочерними классами.

Класс PlayerStats – управляет характеристиками игрока и изменяет модификаторы характеристик при смене снаряжения (рисунок 12).

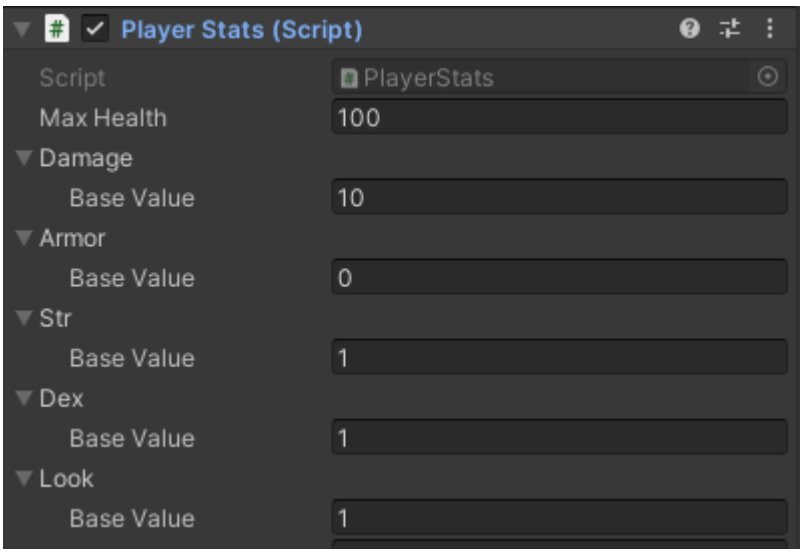

Рисунок 12 – Базовые характеристики персонажа

Метод OnEquipmentChanged отвечает за изменение модификаторов от характеристик при смене снаряжения. Метод Die отвечает за поражение игрока.

Листинг 29 – методы OnEquipmentChanged и Die класса PlayerStats

```
void OnEquipmentChanged(Equipment newItem, Equipment oldItem)
	{
		if (newItem != null)
		{
			armor.AddModifier(newItem.armorModifier);
			damage.AddModifier(newItem.damageModifier);
			str.AddModifier(newItem.strModifier);
			dex.AddModifier(newItem.dexModifier);
			look.AddModifier(newItem.lookModifier);
		}
		if (oldItem != null)
		{
			armor.RemoveModifier(oldItem.armorModifier);
			damage.RemoveModifier(oldItem.damageModifier);
			str.AddModifier(oldItem.strModifier);
			dex.AddModifier(oldItem.dexModifier);
			look.AddModifier(oldItem.lookModifier);
		}
	}
public override void Die()
	{
		base.Die();
		PlayerManager.instance.KillPlayer();
	}
```

Класс EnemyStats – отвечает за поражение противника. Метод Die удаляет противника при достижении его очков жизни значения 0.

Класс CharacterAnimator – управляет анимацией персонажа. Отвечает за плавность перехода анимации. Метод Update изменяет коэффициент скорости,

что приводит к более плавному переходу анимации перемещения, вход в боевое состояние.

Листинг 30 – метод Update класса CharacterAnimator

```
protected virtual void Update()
    {
        float speedPercent = agent.velocity.magnitude / agent.speed;
        animator.SetFloat("speedPercent", speedPercent,
locomationAnimationSmoothTime, Time.deltaTime);
        animator.SetBool("inCombat", combat.InCombat);
    }
```

Класс CharacterCombat – отвечает за боевую систему, определяет задержку между атаками (рисунок 13).

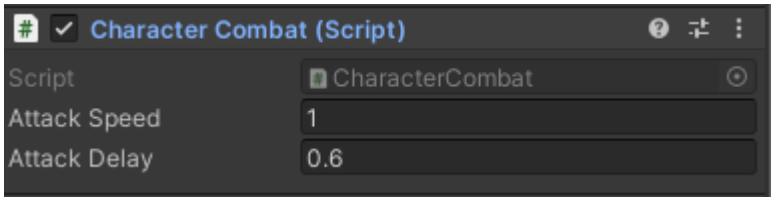

Рисунок 13 – Определение задержки между атаками

Метод Update управляет задержкой между атаками. Метод Attack запускает процесс нанесения урона.

Листинг 31 – методы Update и Attack класса CharacterCombat

```
void Update()
      {
            attackCooldown -= Time.deltaTime;
            if (Time.time - lastAttackTime > combatCooldown)
            {
                  InCombat = false;
            }
      }
public void Attack(CharacterStats targetStats)
      {
            if (attackCooldown <= 0f)
            {
                  opponentStats = targetStats;
                  if (OnAttack != null)
                        OnAttack();
                  attackCooldown = 1f / attackSpeed;
                  InCombat = true;
                  lastAttackTime = Time.time;
            }

      }
```

Класс Enemy – определяет взаимодействие с противником, переопределяет алгоритм взаимодействия, инициирует атаку. Метод Interact переводит персонажа в боевой режим, служит для атаки.

Листинг 32 – метод Interact класса Enemy

```
public override void Interact()
	{
		base.Interact();
		CharacterCombat playerCombat =
playerManager.player.GetComponent<CharacterCombat>();
		if (playerCombat != null)
		{
			playerCombat.Attack(myStats);
		}
	}
```

Класс EquipmentManager – управляет экипировкой, может добавлять и удалять предметы. Метод Equip определяет тип предмета экипировки, определяет подходящий слот персонажа, помещает предмет в слот.

Листинг 33 – метод Equip класса EquipmentManager

```
public void Equip(Equipment newItem)
	{
		int slotIndex = (int)newItem.equipSlot;
		Equipment oldItem = Unequip(slotIndex);
		if (onEquipmentChanged != null)
		{
			onEquipmentChanged.Invoke(newItem, oldItem);
		}
		currentEquipment[slotIndex] = newItem;
		AttachToMesh(newItem, slotIndex);
	}
```

Метод Unequip служит для снятия экипировки. При наличии надетой экипировки, метод помещает предмет в слот инвентаря и освобождает активный слот персонажа. Метод UnequipAll снимает все предметы.

Листинг 34 – методы Unequip и UnequipAll класса EquipmentManager

```
public Equipment Unequip(int slotIndex)
	{
		Equipment oldItem = null;
		if (currentEquipment[slotIndex] != null)
		{
			oldItem = currentEquipment[slotIndex];
			inventory.Add(oldItem);
			SetBlendShapeWeight(oldItem, 0);
			if (currentMeshes[slotIndex] != null)
			{
				Destroy(currentMeshes[slotIndex].gameObject);
			}
			currentEquipment[slotIndex] = null;
			if (onEquipmentChanged != null)
			{
				onEquipmentChanged.Invoke(null, oldItem); }
		}
		return oldItem;
	}
public void UnequipAll()
	{
		for (int i = 0; i < currentEquipment.Length; i++)
		{	Unequip(i);	}
		EquipDefaults();  }
```

Метод EquipDefaults экипирует стандартное снаряжение. Метод Update снимает все экипированные предметы по нажатию соответствующей клавиши.

Класс Interactable – регулирует взаимодействие с объектами, устанавливает радиус взаимодействия, запускает переопределяемый процесс взаимодействия (рисунок 14).

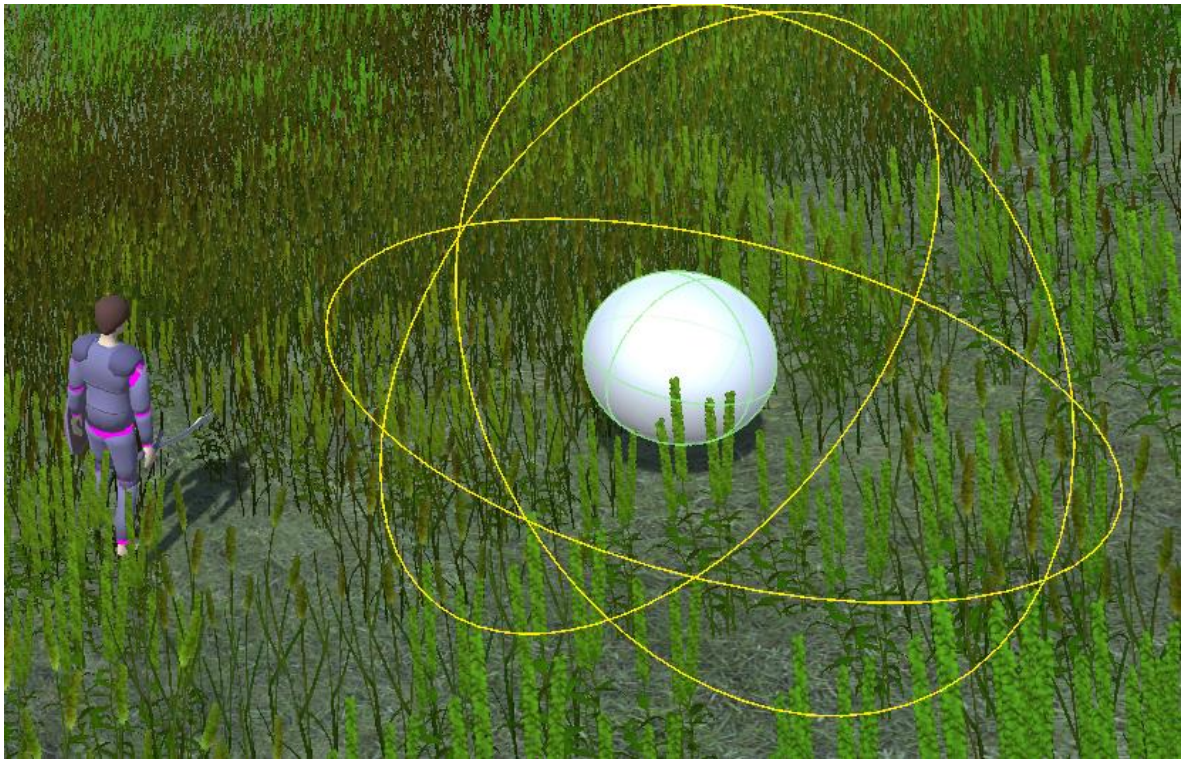

Рисунок 14 – Визуализация радиуса взаимодействия с объектом

Метод Update измеряет расстояние до выделенного объекта и, при нахождении на достаточно близкой дистанции, инициирует процесс взаимодействия.

Листинг 35 – метод Update класса Interactable

```
void Update()
    {
        if(isFocus && !hasInteracted)
        {
            float distance = Vector3.Distance(player.position,
interactionTransform.position);
            if(distance <= radius)
            {
                Interact();
                hasInteracted = true;
            }
        }
    }
```

Метод OnFocused устанавливает готовность для взаимодействия. Метод OnDefocused снимает готовность взаимодействия во избежание повторения

одного и того же действия. Метод OnDrawGizmosSelected визуализирует дальность взаимодействия объекта в редакторе.

Листинг 36 – методы OnFocused, OnDefocused и OnDrawGizmosSelected класса Interactable

```
public void OnFocused (Transform playerTransform)
    {
        isFocus = true;
        player = playerTransform;
        hasInteracted = false;
    }
public void OnDefocused()
    {
        isFocus = false;
        player = null;
        hasInteracted = false;
    }
void OnDrawGizmosSelected()
    {
        if (interactionTransform == null)
            interactionTransform = transform;
        Gizmos.color = Color.yellow;
        Gizmos.DrawWireSphere(interactionTransform.position, radius);
    }
```

Класс PlayerManager – регулирует состояние игрока. Метод KillPlayer запускает сцену поражения.

Класс Dialogue – хранит текст диалогов неигровых персонажей. Не имеет методов.

Класс DialogueTrigger – инициирует начало диалога с неигровыми персонажами. Содержание диалога зависит от выбранного персонажа. Метод Interact передает текст диалога, а также переменные, необходимые анализа игрока в метод StartDialogue класса DialogueManager.

Листинг 37 – метод Interact класса DialogueTrigger

```
public override void Interact()
      {
            base.Interact();
            CharacterStats targetStats =
target.GetComponent<CharacterStats>();
            Debug.Log("Working as intended");
            FindObjectOfType<DialogueManager>().StartDialogue(dialogue,
targetStats.look.GetValue() * targetStats.reputation > approveValue,
targetStats.counter == completeValue);
      }
```

Класс DialogueManager – позволяет включать и отключать окно диалога, выводить имя неигрового персонажа и текста диалога на экран. Метод StartDialogue отвечает за появление окна диалога, определение выводимого

текста в зависимости от значения параметров approval и completed. Параметры зависят от результата анализа игрока неигровым персонажем. Таким образом, если, согласно оценке неигрового персонажа, игрок не способен выполнить его задание, задание не будет выдано игроку (рисунок 15).

Рисунок 15 – Игрок не получил задание

Если, согласно оценке неигрового персонажа, игрок способен выполнить его задание (рисунок 16), задание будет выдано игроку (рисунок 17).

Рисунок 16 – Неигровой персонаж согласился выдать задание игроку

Рисунок 17 – Выдача задания

Для выполнения данного задания, игроку необходимо победить указанных противников (рисунок 18).

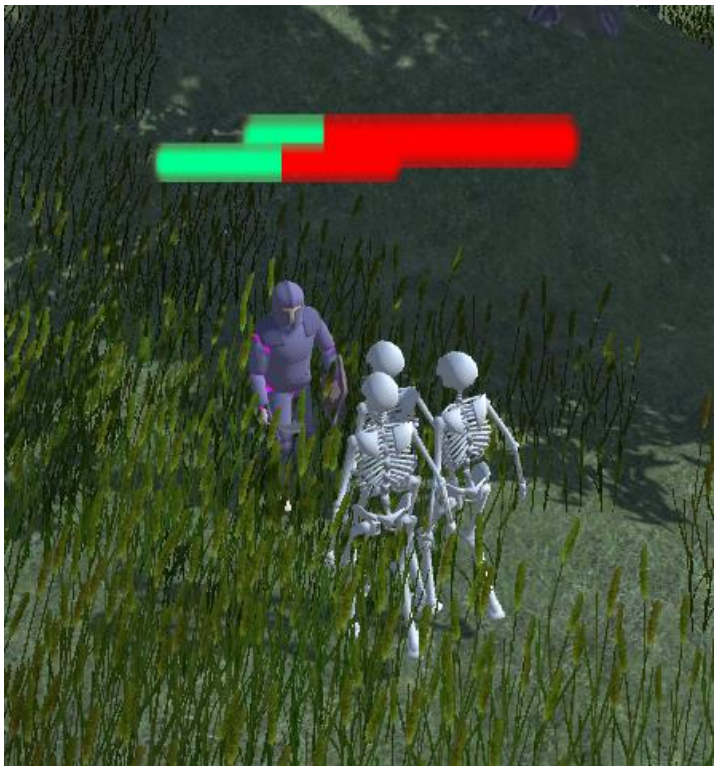

Рисунок 18 – Сражение персонажа с противниками

После победы, игроку необходимо вернуться к неигровому персонажу для сдачи задания (рисунок 19).

Рисунок 19 – Сдача задания игроком

Листинг 38 – метод StartDialogue класса DialogueManager

```
public void StartDialogue(Dialogue dialogue, bool approval, bool completed)
	{
		animator.SetBool("IsOpen", true);

		nameText.text = dialogue.name;

		sentences.Clear();
		if (completed)
		{
			foreach (string sentence in dialogue.sentencesCompleted)
			{
				Debug.Log(sentence);
				sentences.Enqueue(sentence);
			}
		}
		else if (approval)
		{
			foreach (string sentence in dialogue.sentencesApproved)
			{
				Debug.Log(sentence);
				sentences.Enqueue(sentence);
			}
		}
		else
		{
			foreach (string sentence in dialogue.sentencesDenied)
			{
				Debug.Log(sentence);
				sentences.Enqueue(sentence);
			}
		}
		DisplayNextSentence();
	}
```

5 ТЕСТИРОВАНИЕ

В качестве метода тестирования будет проверено соответствие полученного программного обеспечения поставленным функциональным требованиям (таблица 1).

Таблица 1 – Тестирование приложения на соответствие функциональным требованиям

Требование	Действие	Результат
Пользователь должен иметь возможность управлять передвижением персонажа	Персонаж передвигается путем нажатия левой кнопки мыши, находит кратчайший путь до выбранной точки, анимации ходьбы и бега плавно переходят друг в друга благодаря модификатору скорости	Тест пройден
Пользователь должен иметь возможность взаимодействовать с окружающими предметами и противниками	Пользователь может взаимодействовать с окружением путем нажатия правой кнопки мыши. В зависимости от цели, переопределяется алгоритм взаимодействия	Тест пройден
Пользователь должен иметь возможность повышать характеристики персонажа	Пользователь может повышать характеристики благодаря модификаторам экипировки	Тест пройден
Пользователь должен иметь возможность сражаться с противниками	При взаимодействии с противником, персонаж переходит в режим атаки, сражается с противником, может победить или проиграть	Тест пройден
Пользователь должен иметь возможность менять экипировку персонажа	Пользователь способен надевать на персонажа экипировку из инвентаря, или же снимать, помещая ее обратно в инвентарь	Тест пройден
Пользователь должен иметь возможность хранить экипировку в инвентаре	Инвентарь пользователя способен вмещать до 20 предметов, имеется возможность выбросить или использовать предмет из инвентаря, надеть экипировку.	Тест пройден

Вывод. Было проведено тестирование разработанного приложения на соответствие функциональным требованиям. Результаты теста показали, что приложение удовлетворило все 6 требований.

ЗАКЛЮЧЕНИЕ

В ходе выполнения выпускной квалификационной работы были решены следующие задачи:

1) проанализированы главные критерии жанра RPG;

2) проведено исследование предпочтений целевой аудитории игры;

3) проанализированы существующие игры в жанре RPG;

4) определен вид, платформа и среда разработки игры;

5) разработана компьютерная игра в жанре RPG;

6) проведено тестирование разработанной игры.

Результатом работы стал прототип компьютерной игры в жанре RPG с усовершенствованной системой анализа игровых персонажей, удовлетворяющий всем поставленным функциональным требованиям. В дальнейшем планируется расширение игровых возможностей, переход на иные игровых платформы.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Самитов, А. Исследование: число увлекающихся видеоиграми людей превысило 3 миллиарда / А. Самитов. – Текст : электронный // DTF : [блог-сайт]. – 15 августа 2020. – URL: https://dtf.ru/gameindustry/189500-issledovanie-chislo-uvlekayushchihsya-videoigrami-lyudey-prevysilo-3-milliarda (дата обращения: 30.04.2022).

2. Компьютерная ролевая игра / Фонд Викимедиа. – Текст : электронный // Википедия : [сайт]. – 2022. – URL: https://ru.wikipedia.org/wiki/Компьютерная_ролевая_игра (дата обращения: 30.04.2022).

3. Ролевая игра / Фонд Викимедиа. – Текст : электронный // Википедия : [сайт]. – 2022. – URL: https://ru.wikipedia.org/wiki/Ролевая_игра (дата обращения: 30.04.2022).

4. Бабаев, В. Исследование: игроки предпочитают RPG продолжительностью 40-60 часов / В. Бабаев. – Текст. Изображение (неподвижное ; двухмерное) : электронные // DTF : [блог-сайт]. – 19 августа 2017. – URL: https://dtf.ru/gamedev/9414-issledovanie-igroki-predpochitayut-rpg-prodolzhitelnostyu-40-60-chasov (дата обращения: 30.04.2022).

5. About Us / Lo-Fi Games. – Текст. Изображение (неподвижное ; двухмерное) : электронные // Kenshi : [официальный сайт]. – 2022. – URL: https://lofigames.com/about/about-us/ (дата обращения: 30.04.2022).

6. Об игре / Grinding Gear Games. – Текст : электронный // Path of Exile : [официальный сайт]. – 2022. – URL: https://ru.pathofexile.com/game (дата обращения: 30.04.2022).

7. Path of Exile / Фонд Викимедиа. – Текст : электронный // Википедия : [сайт]. – 2022. – URL: https://ru.wikipedia.org/wiki/Path_of_Exile (дата обращения: 30.04.2022).

8. Dark Souls: Prepare to Die Edition / Red Ventures. – Текст. Изображение (неподвижное ; двухмерное) : электронные // Metacritic : [агрегатор рецензий].

– 2022. – URL: https://www.metacritic.com/game/pc/dark-souls-prepare-to-die-edition (дата обращения: 30.04.2022).

9. Dark Souls II: Scholar of the First Sin / Red Ventures. – Текст. Изображение (неподвижное ; двухмерное) : электронные // Metacritic : [агрегатор рецензий]. – 2022. – URL: https://www.metacritic.com/game/playstation-4/dark-souls-ii-scholar-of-the-first-sin (дата обращения: 30.04.2022).

10. Dark Souls III / Red Ventures. – Текст. Изображение (неподвижное ; двухмерное) : электронные // Metacritic : [агрегатор рецензий]. – 2022. – URL: https://www.metacritic.com/game/pc/dark-souls-iii (дата обращения: 30.04.2022).

11. System requirements for Unity 2021 LTS / Unity Technologies. – Текст : электронный // Unity Documentations : [официальный сайт]. – 2021. – URL: https://docs.unity3d.com/Manual/system-requirements.html#desktop (дата обращения: 16.05.2022).