

МИНИСТЕРСТВО ОБРАЗОВАНИЯ ОРЕНБУРГСКОЙ ОБЛАСТИ  
ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ПРОФЕССИОНАЛЬНОЕ  
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ  
«ОРЕНБУРГСКИЙ КОЛЛЕДЖ ЭКОНОМИКИ И ИНФОРМАТИКИ»  
(ГАПОУ ОКЭИ)

## КУРСОВОЙ ПРОЕКТ

*ОКЭИ 09.02.07. 9023. 15 У*  
(код документа)

Курсовая работа  
по МДК 01.01 «Разработка программных модулей»

Создание «2D» видеоигры на «Unity»

Количество листов 44

Руководитель Егурнова Е.Н.

Разработал Логинов С.А.

Защищен с оценкой \_\_\_\_\_

Оренбург 2023

## **Аннотация**

Курсовой проект на тему: «Создание 2D видеоигры на Unity».

Ключевые слова: игровой движок, инди игры, стек разработки, «Canvas», тестирование программного обеспечения, «Unity», «C#».

В отчете подробно рассмотрен процесс создания игры с использованием встроенных инструментов игрового движка «Unity» и встроенного языка программирования «C#».

Отчет включает в себя 44 страницы, 49 рисунков и 4 таблицы.

## Содержание

Введение .....	4
1 Анализ предметной области .....	7
2 Проектирование приложения .....	9
3 Разработка программного обеспечения.....	11
3.1 Описание технологического стека разработки.....	11
3.2 Описание алгоритма работы.....	12
3.3 Описание интерфейса пользователя .....	34
4 Тестирование приложения.....	36
4.1 План тестирования.....	36
4.1 Оценка результатов проведения тестирования.....	37
Заключение .....	40
Список источников .....	41
Приложение А .....	42
Приложение Б.....	43
Приложение В .....	44

					<i>ОКЭИ 09.02.07 9023. 15 У</i>		
<i>Изм.</i>	<i>Лист</i>	<i>№ докум.</i>	<i>Подп.</i>	<i>Дата</i>			
<i>Разраб.</i>		<i>Логинов С.А.</i>			<i>МДК 01.01 «Разработка программных модулей»</i>		
<i>Провер.</i>		<i>Егурнова Е.Н.</i>					
						<i>Лит.</i>	<i>Лист</i>
						<i>п</i>	
						<i>Листов</i>	
						<i>Отделение информационных технологий, гр. 4пк2</i>	

## Введение

В современном мире компьютерные игры стали неотъемлемой частью развлечения миллионов людей по всему миру. Данное направление активно развивается и имеет множество перспективных направлений в медиаиндустрии. Времена, когда разработкой игр занимались только крупные компании и средства для их разработки были доступны только внутри их компаний ушли в прошлое. Технологии разработки доступны для всех и позволяют даже одному разработчику создать свою игру, не прибегая к чужой помощи. На данный момент большинство проектов разрабатываются при помощи игровых движков, которые включают в себя множество полезных функций, упрощающих разработку компьютерных игр. Однако, несмотря на большое количество многофункциональных инструментов, которые будут рассмотрены далее, от разработчика всё равно требуется очень большое количество труда и знаний, чтобы создать действительно хороший продукт.

Игровой движок («Game Engine») — это программное обеспечение, разработанное для упрощения и ускорения процесса создания компьютерных игр. Этот инструмент предоставляет разработчикам все необходимые компоненты и функции для проектирования, создания, редактирования и запуска игровых проектов.

Основные компоненты игрового движка включают в себя:

- графический движок («Graphics Engine»): отвечает за отображение графики, работу с «2D» и «3D» моделями, освещение, текстуры и другие визуальные аспекты игры;

- физический движок («Physics Engine»): обеспечивает моделирование физики в игре, такие как коллизии, гравитация, силы, движение объектов и взаимодействие между ними;

- искусственный интеллект («Artificial Intelligence»): позволяет создавать поведение неигровых персонажей, так называемых «Non-Player Character», управлять их действиями и решениями на основе заранее заданных алгоритмов;

- аудио-движок («Audio Engine»): отвечает за воспроизведение звуковых эффектов, музыки и звуковой атмосферы в игре;

- инструменты разработки («Development Tools»): включают в себя редакторы сцен, анимаций, скриптов, интерфейсов и другие инструменты для удобства работы разработчиков;

- система управления ресурсами («Resource Management System»): оптимизирует работу с ресурсами игры (текстуры, звуки, модели), управляя их загрузкой, выгрузкой и оптимизацией для улучшения производительности;

- скриптовые языки («Scripting Languages»): предоставляют средства для написания скриптов и программирования игровой логики.

Игровые движки бывают коммерческими и бесплатными с открытым исходным кодом. Они позволяют разработчикам сосредоточиться на

креативном процессе создания игры, а не на написании с нуля всех необходимых компонентов. Кроме того, они часто предоставляют возможности многоплатформенной разработки, позволяя выпускать игры на различных устройствах, таких как компьютеры, консоли, мобильные устройства.

Также с помощью игрового движка разработчики могут значительно ускорить процесс создания игр, снизить затраты времени и ресурсов на разработку, а также получить доступ к мощным инструментам для реализации своих идей в интерактивной форме.

На сегодняшний день игровая индустрия развивается невероятно быстро, количество активных игроков и прибыль растет с каждым годом. Все это указывает на то, что разработка видеоигр является одним из перспективнейших направлений в сфере разработки программных продуктов.

Можно точно сказать, что разрабатываемая игра будет достаточно популярна среди выделенной целевой аудитории. Этому способствуют выбранные жанр и стилизация проекта. Жанр «бродилка» представляет собой простой и понятный, а также интересный игровой процесс. Обладая очень низким порогом вхождения, такие игры доступны очень широкому кругу людей, в том числе тем, кто играми как средством проведения собственного досуга мало интересуется.

Так же стоит помнить, что современное общество перенасыщено крупными проектами, требующие от игрока глубокого понимания механики игрового процесса. Для их прохождения требуется довольно большое количество времени. На их фоне игра с простым и понятным геймплеем будет смотреться более предпочтительно среди людей, которым сложно уделять достаточно много времени.

На основе вышеперечисленного легко выделяется цель проекта.

Цель курсового проекта заключается в создании компьютерной игры жанра «2D - бродилка» с помощью игрового движка «Unity».

Для достижения поставленной цели необходимо выполнить следующие задачи:

- проектирование игры: определение концепции игры, механик геймплея и создание игрового мира;
- реализация игровых механик: создание управления персонажем, взаимодействия с окружением и других игровых элементов;
- визуальное оформление: разработка графического стиля, создание анимаций и интерфейса пользователя;
- тестирование и оптимизация: проверка функционала, исправление ошибок и оптимизация производительности игры.

Практическая значимость полученных результатов заключается в том, что полученные результаты имеют важное практическое значение в контексте разработки игр. Созданная «бродилка» не только представляет собой полноценную игру, но и отражает ключевые аспекты работы над игровым проектом, начиная от идеи и заканчивая рабочей версией. Полученный опыт и

навыки играют важную роль в дальнейшей разработке игры и программировании.

Для реализации проекта будут задействованы программные ресурсы, перечисленные ниже:

- игровой движок «Unity»: интегрированная среда разработки для создания игр;
- графические инструменты: «Photoshop» и «Aseprite»;
- инструменты программирования: «Visual Studio Code», для написания кода.

Также для реализации будут задействованы следующие технические ресурсы:

- процессор: Ryzen 5 2600;
- видеокарта: Nvidia GeForce GTX 1660 SUPER;
- оперативная память: 16 ГБ;
- твердотельный накопитель: SSD объемом 256 ГБ;
- операционная система: Windows 10 Pro.

Перспективы расширения функционала приложения: полученный в ходе выполнения задач результат - лишь начальная точка для дальнейшего улучшения и расширения функционала приложения. Возможные направления развития включают в себя добавление новых уровней, введение дополнительных игровых механик, усовершенствование графики и адаптацию для различных платформ.

					ОКЭИ 09.02.07 9023 15 У	Лист
Изм.	Лист	№ докум.	Подп.	Дата		6

## 1 Анализ предметной области

Разработка компьютерных игр — процесс создания компьютерных игр (видеоигр). Разработкой видеоигр занимается разработчик, который может быть представлен как одним человеком, так и фирмой. Обычно крупномасштабные коммерческие игры разрабатываются командами разработчиков в пределах компании, специализирующейся на играх для персонального компьютера или консолей. Как правило, разработку финансирует другая, более крупная компания-издатель, которая по окончании разработки занимается изданием игры и связанными с ним тратами. Реже компании-издатели могут содержать внутренние команды разработчиков, или же компания-разработчик может разрабатывать игры за свой счет и распространять их без участия издателей, например, средствами цифровой дистрибуции (инди-игры).

Разработка наиболее крупнобюджетных игр, таких как «AAA-игры», может стоить десятки миллионов долларов США, причем в течение последних десятилетий эти бюджеты непрерывно росли, как и численность команд разработчиков и сроки разработки.

Благодаря развитию рынка инди-игр, многие разработчики компьютерных игр получили возможность работать над своими игровыми проектами без финансовых и юридических обязательств перед компаниями-издателями.

Инди-игры (англ. «Indie games», от англ. «independent video games» — «независимые компьютерные игры») — это компьютерные игры, созданные отдельными разработчиками или небольшими коллективами без финансовой поддержки издателя компьютерных игр. Распространение осуществляется посредством каналов цифровой дистрибуции, то есть сейчас существует множество платформ, на которые независимый разработчик может выкладывать свои игры, самые известные из них: «Steam», «Epic Games Store», «Itch.io» и «VK Play». Масштаб явлений, связанных с инди-играми, ощутимо возрастает со второй половины 2000-х годов, в основном ввиду развития новых способов онлайн-дистрибуции и средств разработки.

Разработка инди игр на «Unity» представляет собой процесс создания игровых приложений в двухмерном или трехмерном формате при помощи программной среды «Unity». Развитие данной области происходит в контексте постоянного усовершенствования инструментов и технологий, направленных на облегчение и ускорение процесса разработки.

Сущности и явления данной области разработки перечислены ниже:

– интегрированная среда разработки («IDE»): «Unity» предоставляет разработчикам универсальную среду с графическим интерфейсом для создания, редактирования и программирования игровых элементов;

– графический движок и рендеринг: «Unity» обладает инструментами для создания и управления «2D» графикой, позволяющими реализовывать анимацию, работать с текстурами, спрайтами и анимированными объектами;

– программирование и скриптинг: использование языка программирования «C#» для создания игровой логики, взаимодействия объектов, создания и редактирования скриптов, определяющих поведение элементов игры;

– управление ресурсами: эффективное управление звуками, изображениями, анимациями, и другими ресурсами игры с помощью специальных инструментов «Unity».

Отношения и процессы, определяющие сферу деятельности данной области:

– цикл разработки игры: включает этапы планирования, дизайна, программирования, тестирования и выпуска игры, каждый из которых требует четкой организации и согласования усилий всей команды разработчиков;

– командная работа и коммуникация: в разработке игр важна эффективная командная работа, включая взаимодействие между программистами, дизайнерами, художниками, звукорежиссерами и другими специалистами;

– техническая поддержка и обновления: после выпуска игры важно обеспечивать техническую поддержку, исправлять ошибки, так называемые баги, и предоставлять обновления для улучшения игрового опыта игроков;

– монетизация и распространение: определение стратегий монетизации игры (платная/бесплатная с возможностью покупок внутри игры), выбор платформ для распространения («Steam», мобильные платформы и другие).

Разработка инди игр на «Unity» представляет собой многоаспектный процесс, требующий особый подход, глубоких знаний технических аспектов игровой разработки и умения эффективно управлять проектом от начала до выпуска готовой игры на рынок.

Разрабатываемый проект относится к инди-разработке и развивается только за счет средств разработчиков. Благодаря развитию программного обеспечения разработки компьютерных игр, команде разработчиков не требуется тратить несколько лет на разработку игрового движка. Это позволяет сразу же приступить к непосредственной работе над игровым проектом и значительно сокращает время его разработки.



## 2 Проектирование приложения

Проектируемое приложение «Wander 2D» – это игра в жанре «бродилка» и расположением камеры «top - down», в которой главный герой исследует игровой мир, взаимодействуя с различными предметами. Данная игра предназначена для широкого круга пользователей с простыми механиками и интуитивно понятным коротким игровым процессом.

Для успешной реализации данного продукта, нужно сформулировать ряд требований.

Требования по геймплейной составляющей – игрок появляется на первом уровне, чтобы попасть на следующий, ему нужно найти некоторое количество монет. Чтобы получить те самые монеты, игрок должен изучать различные локации, присутствующие на карте. Изучая локации, игрок должен решать ряд простых задач такие как передвинуть предмет, принести предмет и так далее. Выполнение данных задач поощряется выдачей монет, которые необходимы игроку для успешного прохождения игры.

Игра должна иметь двумерную графику и вид сверху.

В игре должны присутствовать две локации: лес и замок. Каждая локация уникальна, то есть локации должны отличаться дизайном и загадками.

Загадки должны быть достаточно простыми, например требуется принести какую – либо вещь или переместить нужный объект в необходимое место для получения монеты.

Интерфейс игры состоит из главного меню и игровой панели. В главном меню игрок может выбрать уровень, сбросить свой прогресс или выйти из игры. Игровая панель находится в верхней части экрана и состоит из двух компонентов: изображение монетки, которые нужно собирать и непосредственно их количество.

Управление осуществляется при помощи клавиш, представленных ниже:

- «w» - вверх;
- «a» - влево;
- «s» - вправо;
- «d» - вниз.

В ходе анализа предметной области и на основе необходимых требований были определены функциональные и нефункциональные требования к игровому приложению и возможности, которые должны предоставляться пользователю приложения.

Функциональные требования определяют функциональность программного обеспечения, то есть описывают, какое поведение должна предоставлять разрабатываемая система. Разрабатываемая система должна удовлетворять следующим функциональным требованиям:

- в приложении должно присутствовать меню для более удобного начала игры;
- в меню должна присутствовать кнопка для выхода из игры;

					ОКЭИ 09.02.07 9023 15 Ч	Лист
Изм.	Лист	№ докум.	Подп.	Дата		9

- пользователь должен иметь возможность выбора уровня;
- пользователь должен иметь возможность сбрасывать свой прогресс;
- пользователь должен иметь возможность взаимодействовать с окружающими объектами;
- пользователь должен иметь возможность подбирать монеты;
- пользователь должен иметь возможность подбирать предметы;
- пользователь должен иметь возможность видеть текущее число подобранных монет.

Нефункциональные требования описывают свойства и ограничения, накладываемые на систему. Для реализации приложения были выявлены следующие требования:

- приложение должно быть создано на игровом движке «Unity» с использованием языка программирования «C#»;
- обеспечение стабильной работы приложения на настольных компьютерах с операционной системой семейства «Windows», версии не ниже «Windows 7»;
- создание удобного пользовательского интерфейса для облегчения взаимодействия с игрой;
- возможность дальнейшего расширения функционала и добавления новых элементов.

В ходе анализа требований была разработана «UML-диаграмма» вариантов использования компьютерной игры, с которой можно ознакомиться в «Приложение А».

## 3 Разработка программного обеспечения

### 3.1 Описание технологического стека разработки

Стек разработки — это набор технологий, инструментов, языков программирования, фреймворков и других компонентов, используемых при создании программного обеспечения или приложений. Этот стек определяет инфраструктуру, которая будет использоваться для разработки, тестирования, внедрения и поддержки проекта.

Основные компоненты стека разработки могут включать в себя:

- языки программирования;
- фреймворки и библиотеки;
- базы данных;
- среда разработки и инструменты управления версиями;
- серверные и клиентские технологии.

Стек разработки зависит от типа проекта, требований к производительности, предпочтений команды разработчиков и других факторов. Правильно выбранный стек может ускорить процесс разработки, повысить производительность и обеспечить лучшую масштабируемость и надежность приложения.

При разработке «2D» игры будет использован стек, представленный ниже:

- язык программирования «C#»;
- игровой движок «Unity Engine».

Язык программирования «C#» - является основным языком программирования в «Unity». Данный язык используется для написания скриптов — программы, написанные на языке программирования «C#», которые определяют поведение и действия объектов в игре. Они позволяют управлять движением, взаимодействием и внешним видом объектов. С помощью скриптов можно создавать сложные анимации, реагировать на пользовательский ввод и управлять игровой логикой.

«Unity Engine» — это кроссплатформенный игровой движок, который широко используется для разработки компьютерных игр, виртуальной и дополненной реальности, тренировочных симуляторов, а также для создания интерактивных «2D» и «3D» приложений.

Обоснование выбора данного стека заключается в том, что «Unity Engine» предоставляет мощные инструменты и гибкие возможности для создания «2D» игр, обеспечивая удобство работы с графикой, анимациями, физикой и аудио. Это позволяет сосредоточиться на самом процессе разработки игры, ускоряя время разработки. Также данный движок обладает богатым комьюнити, то есть Unity имеет огромное сообщество разработчиков, готовых поддержать и помочь в решении проблем. В свою очередь, язык программирования «C#» обладает мощными возможностями объектно-

					ОКЭИ 09.02.07 9023 15 Ч	Лист
Изм.	Лист	№ докум.	Подп.	Дата		11

ориентированного программирования, и интеграция с «Unity API» обеспечивают высокую производительность и эффективность в разработке.

Выбранный стек разработки обладает рядом преимуществ, такими как продуктивность разработки, масштабируемость и портативность и поддержка сообщества.

Продуктивность разработки заключается в том, что «C#» обеспечивает удобство при программировании игровой логики, а «Unity Engine» предоставляет интуитивно понятный интерфейс и множество инструментов для ускорения разработки.

Масштабируемость «Unity» позволяет разрабатывать игры для различных платформ, таких как персональные компьютеры, мобильные устройства, консоли и веб. Это обеспечивает большую аудиторию и возможность масштабирования проекта.

Поддержка сообщества выражается большим сообществом разработчиков «Unity», которые активно обмениваются опытом, предоставляет обучающие материалы и помощь, что существенно облегчает разработку и устранение возможных проблем.

Использование игрового движка «Unity» с языком программирования «C#» обеспечивает необходимый инструментарий для создания «2D» игр, упрощает процесс разработки и обеспечивает возможность достижения широкой аудитории игроков.

## 3.2 Описание алгоритма работы

В качестве основы для рассмотрения алгоритма работы программы, была взята диаграмма последовательности для первого уровня игры, с которой можно ознакомиться в «Приложение Б».

Изначально пользователь будет иметь папку с файлами готовой игры, как показано на «Рисунок 1».







	MonoBleedingEdge	26.12.2023 0:18	Папка с файлами	
	Wander2D_BurstDebugInformation_DoN...	26.12.2023 0:18	Папка с файлами	
	Wander2D_Data	26.12.2023 0:18	Папка с файлами	
	UnityCrashHandler64.exe	26.12.2023 0:18	Приложение	1 089 КБ
	UnityPlayer.dll	26.12.2023 0:18	Расширение при...	30 049 КБ
	Wander2D.exe	26.12.2023 0:18	Приложение	651 КБ

Рисунок 1 – Папка с игрой.

Чтобы запустить приложение, пользователь должен выбрать «Wander2D.exe».

После запуска игры, перед пользователем появляется окно с выбором уровня, который можно увидеть на «Рисунок 2».



Рисунок 2 – Главное меню.

В главном меню пользователь может выбрать уровень, сбросить прогресс или же выйти из игры.

Изначально пользователю доступен только первый уровень, по мере прохождения игры открывается следующий. Код для проверки прогресса пользователя представлен на «Рисунок 3».

```
0 references
void Start()
{
    levelComplete = PlayerPrefs.GetInt("LevelComplete");
    levelTwo.interactable = false;

    switch(levelComplete){
        case 1:
            levelTwo.interactable = true;
            break;
    }
}
```

Рисунок 3 – Код, регулирующий прогресс.

Прогресс отслеживается при помощи класса «PlayerPrefs», который предназначен для простого способа сохранения и чтения данных между сеансами игры. Он используется для хранения простых данных, таких как настройки игры, состояния уровней, счет игрока и других параметров. «PlayerPrefs» сохраняет данные в виде пар ключ-значение и хранит их между сеансами игры даже после

её закрытия. Это удобный способ сохранения информации без необходимости использования файлов или баз данных.

Далее после прохождения игры, последующие уровни будут доступны в главном меню, как показано на «Рисунок 4».



Рисунок 4 – Второй уровень доступен.

Чтобы сбросить прогресс прохождения игры, пользователю достаточно кликнуть по кнопке «RESET», которая расположена снизу экрана главного меню. Код для работы данной кнопки представлен на «Рисунок 5».

```
0 references
public void Reset(){
    levelTwo.interactable = false;
    PlayerPrefs.DeleteAll();
}
```

Рисунок 5 – Код для кнопки «RESET».

После ее нажатия пользователь полностью чистит свой прогресс прохождения, возвращаясь к изначальному состоянию, как было показано ранее при запуске игры.

В нижнем правом углу меню расположена кнопка, которая позволит пользователю выйти из игры. Код для реализации данного элемента представлен на «Рисунок 6».



```

0 references
public void Exit(){
    Application.Quit();
}

```

Рисунок 6 – Код для выхода из игры.

После выбора первого уровня, перед пользователем генерируется игровой мир и интерфейс, о котором стоит упомянуть позже. Сгенерированный мир и интерфейс представлен на «Рисунок 7».



Рисунок 7 – Игровой мир.

Управление персонажем осуществляется при помощи клавиш «w» - вверх, «a» - влево, «s» - вниз, «d» - вправо. Для реализации данной возможности используется код, представленный на «Рисунок 8».

```

public class PlayerController : MonoBehaviour
{
    1 reference
    public float speed = 8;
    6 references
    private Vector2 direction;
    3 references
    private Rigidbody2D rb;
    3 references
    [SerializeField] Animator animator;

    0 references
    void Start()
    {
        rb = GetComponent<Rigidbody2D>();
    }

    0 references
    void Update()
    {
        direction.x = Input.GetAxis("Horizontal");
        direction.y = Input.GetAxis("Vertical");

        animator.SetFloat("Horizontal", direction.x);
        animator.SetFloat("Vertical", direction.y);
        animator.SetFloat("Speed", direction.sqrMagnitude);

        // if(Input.GetKeyDown(KeyCode.Escape)){
        //     SceneManager.LoadScene("MainMenu");
        // }
    }

    0 references
    private void FixedUpdate()
    {
        rb.MovePosition(rb.position + direction * speed * Time.fixedDeltaTime);
    }
}

```

Рисунок 8 – Код для передвижения персонажа.

В коде можно заметить такой элемент как «animator». Он служит для переключения анимаций передвижения персонажа в зависимости от направления движения, которые необходимо задать в самом игровом движке, как показано на «Рисунок 9».

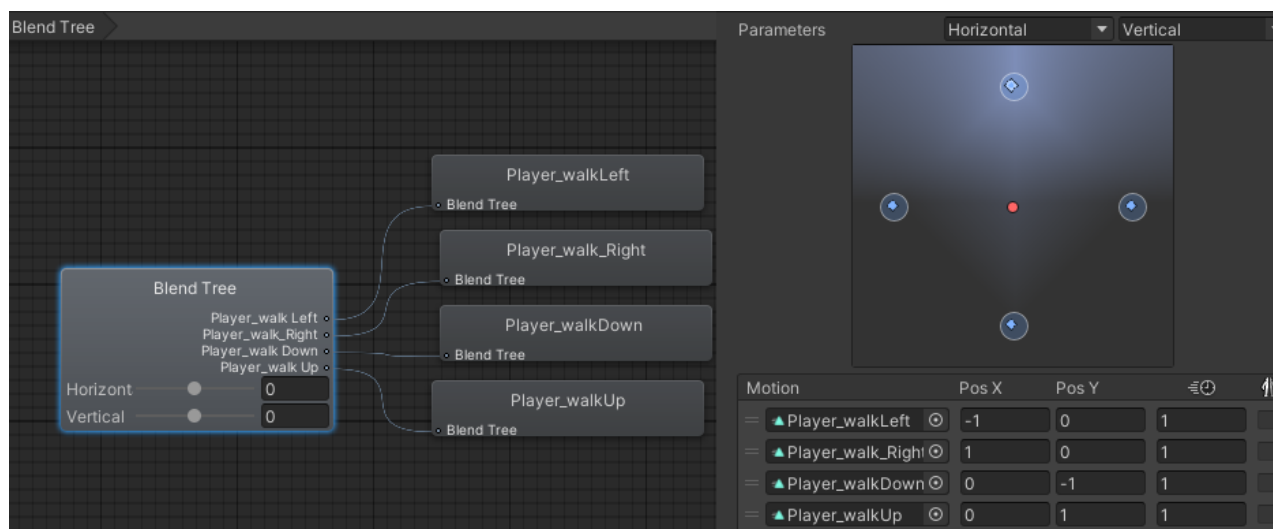


Рисунок 9 – Анимации персонажа.

Изм.	Лист	№ докум.	Подп.	Дата

ОКЭИ 09.02.07 9023 15 У

Лист

16



Для того чтобы понять, что требуется от игрока, пользователь, должен найти дверь, которая является выходом на следующий уровень. Рядом с ней стоит «NPC», который требует некоторое количество монет для открытия двери. Данная локация представлена на «Рисунок 10».



Рисунок 10 – Игровая локация с выходом.

На данной локации размещен «Box Collider 2D» типа «IsTrigger», который проверяет игрока на количество собранных монет. Поле данного коллайдера представлено на «Рисунок 11».



Рисунок 11 – Коллайдер для проверки игрока.

Изм.	Лист	№ докум.	Подп.	Дата

ОКЭИ 09.02.07 9023 15 У

Лист

17

Далее пользователь должен начать исследовать локацию, для нахождения монет, которые помогут ему продвинуться дальше.

Первой из локаций рассмотрим «Алтарь». Здесь размещена достаточно простая загадка, суть которой заключается в передвижении вазы в центр алтаря. Данная локация представлена на «Рисунок 12».



Рисунок 12 – Локация «Алтарь».

Для того что бы ваза двигалась, в окне игрового движка, а именно в «Inspector», был добавлен компонент «Rigidbody 2D», и в настройках данного компонента выставлены настройки, показанные на «Рисунок 13».

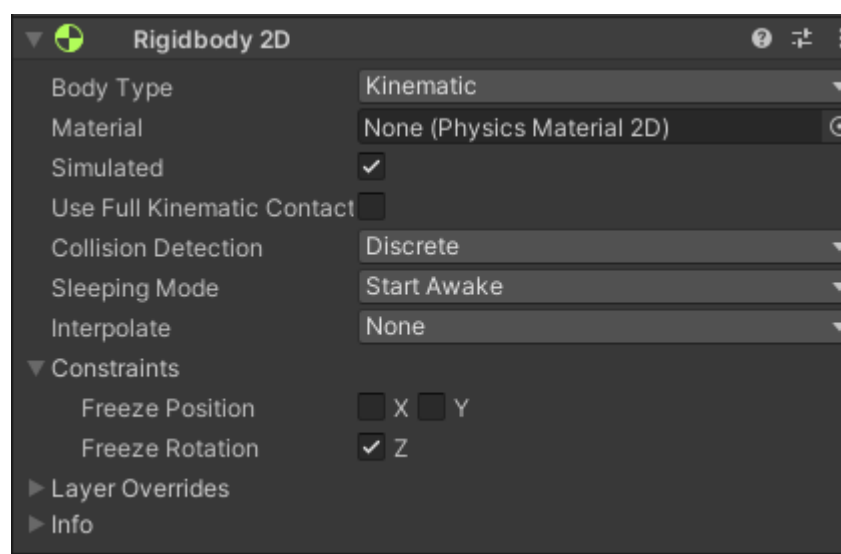


Рисунок 13 – Настройка вазы.

Также для данного объекта необходимо указать тэг «Vase», как показано на «Рисунок 14».

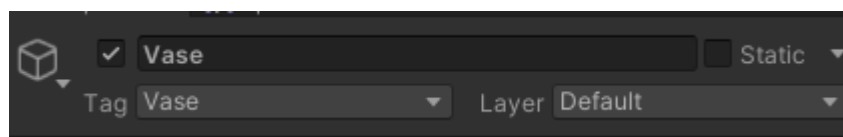


Рисунок 14 – Тэг для вазы.

Для передвижения вазы, игроку достаточно подойти к ней и начать толкать в нужную сторону. Для решения данной загадки достаточно подвинуть вазу в центр алтаря, расположенного правее. После этого, игрок получит первую монету, как показано на «Рисунок 15».



Рисунок 15 – Решение первой загадки.

Для реализации данной загадки был использован «Box Collider 2D» типа «IsTrigger», который проверяет объект, который зашел в него, на наличие тега «Vase». Код данной проверки показан на «Рисунок 16».

```

0 references
public class TriggerObj : MonoBehaviour
{
    3 references
    public GameObject signs;
    3 references
    public GameObject coin;

    0 references
    private void OnTriggerEnter2D(Collider2D other) {
        if(other.CompareTag("Vase")){
            signs.SetActive(true);
            coin.SetActive(true);
        }
    }

    0 references
    private void OnTriggerStay2D(Collider2D other) {
        if(other.CompareTag("Vase")){
            signs.SetActive(true);
            coin.SetActive(true);
        }
    }

    0 references
    private void OnTriggerExit2D(Collider2D other) {
        if(other.CompareTag("Vase")){
            signs.SetActive(false);
            coin.SetActive(false);
        }
    }
}

```

Рисунок 16 – Проверка объекта алтарем.

Далее игроку нужно подобрать данную монету. Здесь также имеется проверка, которая представлена на «Рисунок 17».

```

0 references
public class Coins : MonoBehaviour
{
    0 references
    private void OnTriggerEnter2D(Collider2D collision) {
        if(collision.CompareTag("Player")){
            CoinsText.Coin += 1;
            Destroy(gameObject);
        }
    }
}

```

Рисунок 17 – Код для монет.

Также нужно отметить, что после подбора изменяется цифра в интерфейсе пользователя, как показано на «Рисунок 18».



Рисунок 18 – Изменение количества монет.

Это происходит в ходе выполнения кода, представленного выше и кода, представленного на «Рисунок 19», который отвечает за вывод на экран количества монет игрока.

```

public class CoinsText : MonoBehaviour
{
    3 references
    public static int Coin = 0;
    2 references
    Text text;

    0 references
    void Start()
    {
        text = GetComponent<Text>();
    }

    0 references
    void Update()
    {
        text.text = Coin.ToString();
    }
}

```

Рисунок 19 – Код для вывода количества монет.

Также количество монет отслеживается в скрипте «PlayerStats», который предназначен для слежения за статистикой игрока. Данный код представлен на «Рисунок 20».

```

3 references
public int coinCount = 0;
3 references
public int flowerCount = 0;
1 reference
public CoinsText coinsText;

0 references
private void OnTriggerEnter2D(Collider2D other) {
    if(other.CompareTag("Coin")){
        coinCount += 1;
    }
}

```

Рисунок 20 – Статистика монет игрока.

Далее игроку нужно пройти к локации «Статуя». Здесь нужно принести цветок, который расположен недалеко и очень сильно выделяется, как показано на «Рисунок 21».



Рисунок 21 – Цветок.

На цветке расположен «Box Collider 2D» типа «IsTrigger», который проверяет, что объект с тэгом «Player» зашел в него. Код для работы данного коллайдера представлен на «Рисунок 22».

```

0 references
public class Flower : MonoBehaviour
{
    0 references
    public GameObject flower;

    0 references
    private void OnTriggerEnter2D(Collider2D other) {
        if(other.CompareTag("Player")){
            Destroy(gameObject);
        }
    }
}

```

Рисунок 22 – Код для цветка.

Для его подбора, достаточно подойти к нему и он будет добавлен в статистику игрока. За этот процесс отвечает код, представленный на «Рисунок 23».

```

0 references
public class PlayerStats : MonoBehaviour
{
    2 references
    public int health = 1;
    3 references
    public int coinCount = 0;
    3 references
    public int flowerCount = 0;
    1 reference
    public CoinsText coinsText;

    0 references
    private void OnTriggerEnter2D(Collider2D other) {
        if(other.CompareTag("Coin")){
            coinCount += 1;
        }

        if(other.CompareTag("Flower")){
            flowerCount += 1;
        }
    }
}

```

Рисунок 23 – Статистика цветков игрока.

Далее этот цветок необходимо отнести к статуе, как демонстрируется на «Рисунок 24».

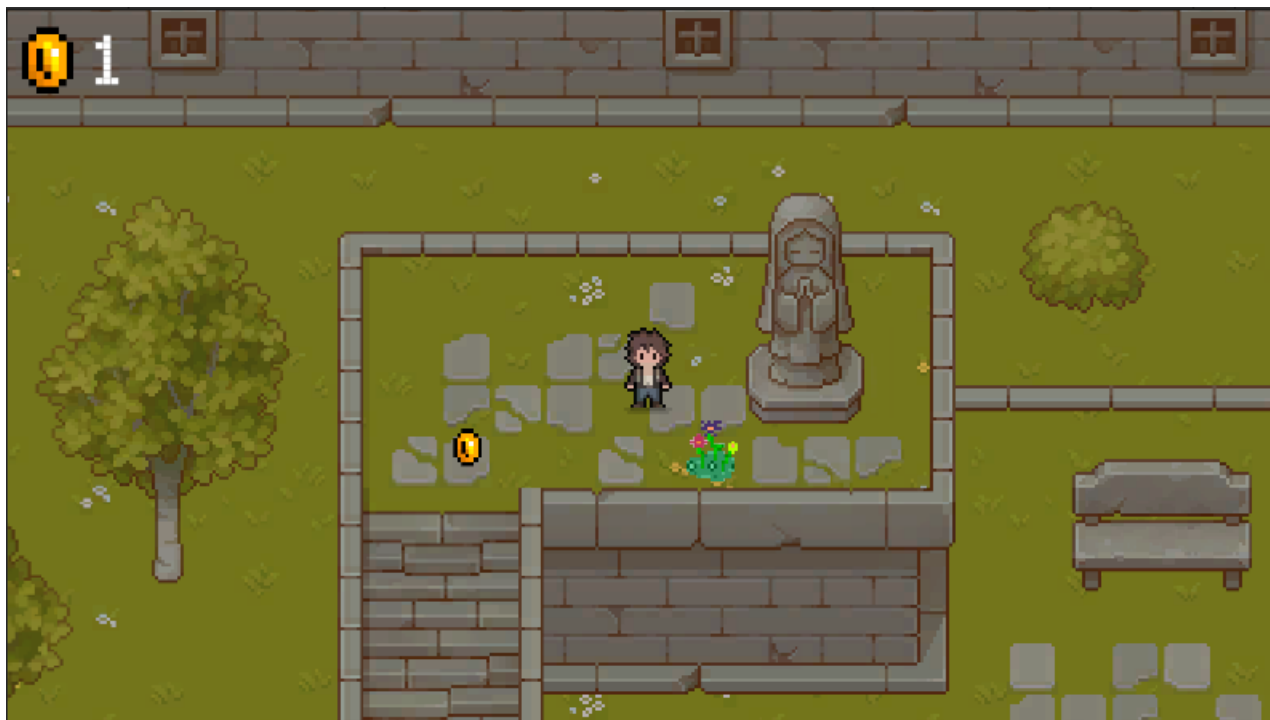


Рисунок 24 – Выполненный квест.

На статуе присутствует «Box Collider 2D» типа «IsTrigger», который проверят, что у игрока имеется необходимый объект. Он вычитается из статистики и квест считается выполненным после размещения цветка у статуи, и игрок получает еще одну монету, которую нужно подобрать.

После подбора монеты, счетчик монет изменится, как в статистике игрока, так и в интерфейсе игры, как показано на «Рисунок 25».



Рисунок 25 – Подбор второй монеты.

Изм.	Лист	№ докум.	Подп.	Дата

ОКЭИ 09.02.07 9023 15 У

Лист

24



Изменение статистики и цифры и интерфейсе игры осуществляется тем же способом, что был рассмотрен ранее.

Далее игроку нужно найти локацию «Кладбище». На данной локации монета просто лежит на земле, как показано на «Рисунок 26».



Рисунок 26 – Локация «Кладбище».

После подбора монеты статистика игрока и цифра в интерфейсе изменятся, как показано на «Рисунок 27».



Рисунок 27 – Подбор третьей монеты.

После сбора всех монет, игроку необходимо вернуться к выходу на следующий уровень.

Рассмотренный ранее коллайдер, для проверки количества монет у игрока использует код, представленный на «Рисунок 28».

```
public PlayerStats playerStats;
2 references | 2 references
public GameObject door, openDoor;

0 references
private void OnTriggerEnter2D(Collider2D other) {
    if(other.CompareTag("Player")){
        foreach(Animator animator in panel){
            animator.SetTrigger("IsTrigger");
        }
        if(playerStats.coinCount >= 3){
            door.SetActive(false);
            openDoor.SetActive(true);
        }
        else{
            door.SetActive(true);
            openDoor.SetActive(false);
        }
    }
}
```

Рисунок 28 – Код для проверки количества монет.

Также данный скрипт отвечает за то, что бы открывать или закрывать двери, в зависимости от количества монет у игрока.

Теперь, когда пользователь имеет нужное количество монет, дверь, для прохода на следующий уровень, открывается, как показано на «Рисунок 29».



Рисунок 29 – Успешная проверка.

После открытия двери, игрок может пройти на следующий уровень, зайдя в нее. Для переноса игрока на следующую локацию используется скрипт, представленный на «Рисунок 30».

```

0 references
public class NextLevel : MonoBehaviour
{
    0 references
    private void OnTriggerEnter2D(Collider2D other) {
        LevelController.instance.IsEndGame();
    }
}

0 references
void NextLevel(){
    SceneManager.LoadScene(sceneIndex + 1);
    coinsText.ResetCoinCount();
}

```

Рисунок 30 – Код для перехода.

Также стоит отметить, что при переходе на следующий уровень, все монеты игрока сбрасываются.

Далее игрок оказывается на следующем уровне, в котором также нужно собрать необходимое количество монет.

На этом уровне имеются противники «Зомби», которые охраняют монеты. Рассмотрим реализацию данных противников.

Изначально противники бегают по заданным точкам, случайно выбирая их. Рассмотрим пример на одном из «Зомби».

Для начала были заданы начальные переменные для патрулирования, как показано на «Рисунок 31».

```

1 reference
public float speed = 3;
1 reference
public float chaseSpeed = 4;
2 references
public float chaseDistance = 7f;
4 references
private float waitTime;
2 references
public float startWaitTime;
4 references
public Transform[] moveSpots;
4 references
public int randomSpot;

```

Рисунок 31 – Переменные для патрулирования.

Далее в методе «Start» было указано время задержки на точках и непосредственно получение их, как продемонстрировано на «Рисунок 32».

```
0 references
void Start()
{
    waitTime = startWaitTime;
    randomSpot = Random.Range(0, moveSpots.Length);
}
```

Рисунок 32 - Метод «Start».

Поле этого в методе «Update» был создан скрипт для перемещения противника от точки к точке, как показано на «Рисунок 33».

```
transform.position = Vector2.MoveTowards(transform.position, moveSpots[randomSpot].position, speed * Time.deltaTime);
if(Vector2.Distance(transform.position, moveSpots[randomSpot].position) < 0.2f){
    if(waitTime <= 0){
        randomSpot = Random.Range(0, moveSpots.Length);
        waitTime = startWaitTime;
    }
    else{
        waitTime -= Time.deltaTime;
    }
}
```

Рисунок 33 – Код перемещения противника.

Чтобы задать точки для перемещения противника в «Unity», нужно было данный скрипт добавить в ранее созданного противника и добавить пустые объекты на игровую локацию. Далее данные пустые объекты нужно добавить в массив, как показано на «Рисунок 34».

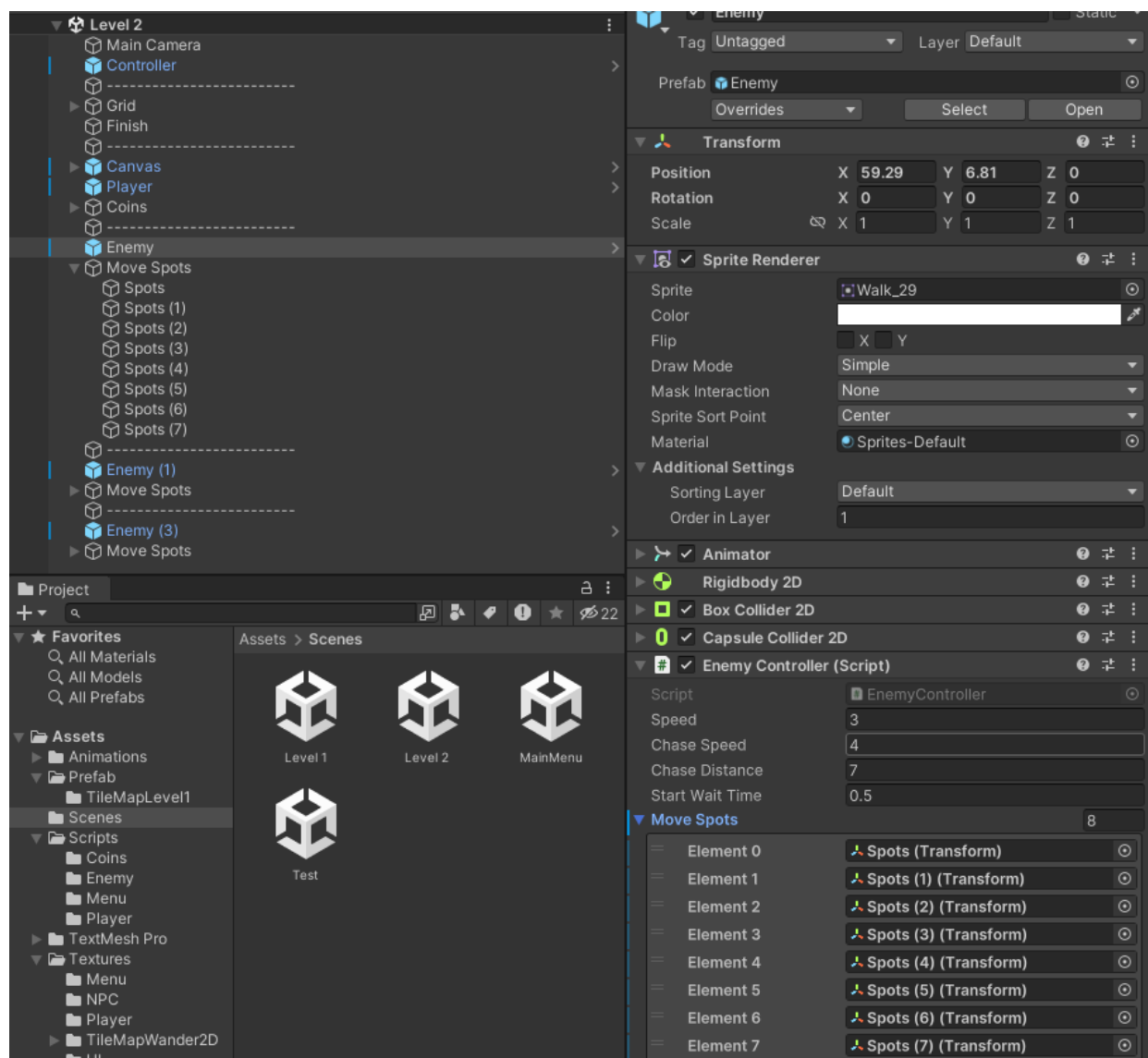


Рисунок 34 – Добавление точек.

Чтобы реализовать режим преследования, когда игрок попадает в поле зрения «Зомби», и возвращение в режим патрулирования, когда игрок отбегает на достаточное расстояние, нужно было, для начала, добавить в скрипт, рассмотренный ранее, новые переменные и в метод «Start» добавить получение информации о положении игрока на карте. Данный код представлен на «Рисунок 35».

```

private Transform player;
1 reference
public float chaseSpeed = 4;
2 references
public float chaseDistance = 7f;

3 references
private bool isChasing = false;

0 references
void Start()
{
    waitTime = startWaitTime;
    randomSpot = Random.Range(0, moveSpots.Length);
    player = GameObject.FindGameObjectWithTag("Player").GetComponent<Transform>();
}

```

Рисунок 35 – Добавление новых переменных.

В метод «Update» были добавлены условия для преследования, как показано на «Рисунок 36».

```

0 references
void Update()
{
    float distanceToPlayer = Vector2.Distance(transform.position, player.position);

    if (distanceToPlayer <= chaseDistance)
    {
        isChasing = true;
    }
    else if (distanceToPlayer > chaseDistance * 2)
    {
        isChasing = false;
    }

    if (isChasing)
    {
        transform.position = Vector2.MoveTowards(transform.position, player.position, chaseSpeed * Time.deltaTime);
    }
    else{
        transform.position = Vector2.MoveTowards(transform.position, moveSpots[randomSpot].position, speed * Time.deltaTime);
        if(Vector2.Distance(transform.position, moveSpots[randomSpot].position) < 0.2f){
            if(waitTime <= 0){
                randomSpot = Random.Range(0, moveSpots.Length);
                waitTime = startWaitTime;
            }
            else{
                waitTime -= Time.deltaTime;
            }
        }
    }
}
}

```

Рисунок 36 – Код для преследования.

Первое условие проверяет то, что игрок находится в поле зрения врага. Второе условие проверяет то, что игрок ушел на достаточное расстояние от противника. Следующее условие исходит от первого, то есть если игрок попадает в поле зрения «Зомби», то враг начинает перемещаться по координатам игрока.

Если же игрок ушел из поля зрения противника, то после окончания преследования «Зомби» возвращается в режим патрулирования.

Также был добавлен урон, наносимый противником. Для этого в ранее рассмотренном скрипте была создана переменная, которая принимает значение здоровья игрока, а также метод, который работает с «Box Collider 2D» в режиме «IsTrigger». Код данной функции представлен на «Рисунок 37».

```
1 reference
public PlayerStats playerHealth;

3 references
private bool isChasing = false;

0 references
private void OnTriggerEnter2D(Collider2D other) {
    if(other.CompareTag("Player")){
        playerHealth.health -= 1;
    }
}
```

Рисунок 37 – Код для нанесения урона.

Теперь игровому персонажу необходимо добавить здоровье, при потере которого, уровень будет перезапущен и все собранные монеты сбросятся. Реализация представлена на «Рисунок 38».

```
2 references
public int health = 1;

0 references
private void Update() {
    if(health <= 0){
        coinsText.ResetCoinCount();
        SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex);
    }
}
```

Рисунок 38 – Код для здоровья игрока.

Теперь рассмотрим на примере, как работают данные функции. При загрузке уровня, «Зомби» бежит от точки к точке, как показано на «Рисунок 39».



Рисунок 39 – Перемещение противника.

Далее нужно подойти к врагу. При попадании игрока в поле зрения «Зомби», противник начинает преследовать игрока, как показано на «Рисунок 40».



Рисунок 40 – Преследование.

После того как игрок отбежит от врага на достаточное расстояние, «Зомби» вернётся в режим патрулирования, как продемонстрировано на «Рисунок 41».





Рисунок 41 – Возвращение в режим патрулирования.

Реализация сбора монет осуществлена также как и на первом уровне.

После сбора необходимого количества монет пользователь сможет закончить игру, как показано на «Рисунок 42».



Рисунок 42 – Конец игры.

После прохождения данного уровня, игрок будет иметь к нему доступ, так как все данные о прохождении были записаны в «PlayerPrefs».

Дополнительно ознакомиться с структурой кода можно в «Приложение В».

### 3.3 Описание интерфейса пользователя

Интерфейс пользователя в игре реализован при помощи внутреннего компонента игрового движка «Unity». Данный компонент называется «Canvas». «Canvas» – это область, внутри которой находятся все элементы «UI» (пользовательского интерфейса). Все элементы «UI» должны быть дочерними этому «Canvas».

Все элементы выполнены с использованием пиксельных спрайтов, чтобы соответствовать стилю игры. Также элементы обладают необходимыми цветами, сочетающиеся с сеттингом игры.

В главном меню пользователь имеет интерфейс, представленный на «Рисунок 43».



Рисунок 43 – Интерфейс главного меню.

В данном случае структура «Canvas» имеет следующий вид, показанный на «Рисунок 44».

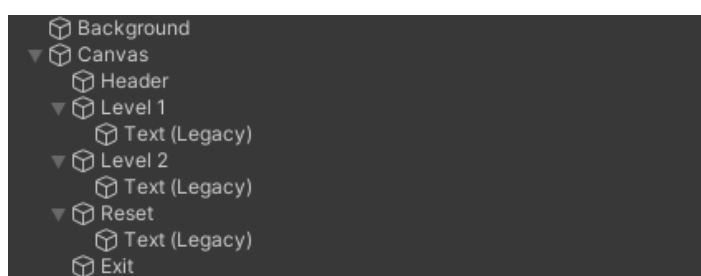


Рисунок 44 – Структура «Canvas» в меню.

В данном случае элемент «Header» — это обычный текст, который расположен сверху интерфейса. Элементы «Level 1» и «Level 2» являются кнопками, которые включают в себя дочерние элементы «Text (Legacy)», с текст внутри кнопок.

В игре интерфейс представляет собой набор из спрайта монеты, которые игроку нужно собирать и цифра, которая отображает текущее количество монет у игрока, как показано на «Рисунок 45».



Рисунок 45 – Внутриигровой интерфейс.

В данном случае «Canvas» имеет структуру, которая показана на «Рисунок 46».

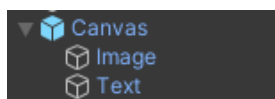


Рисунок 46 – Структура «Canvas» в игре.

В элементе «Image» содержится спрайт монеты, это можно увидеть в вкладке «Inspector», как показано на «Рисунок 47».

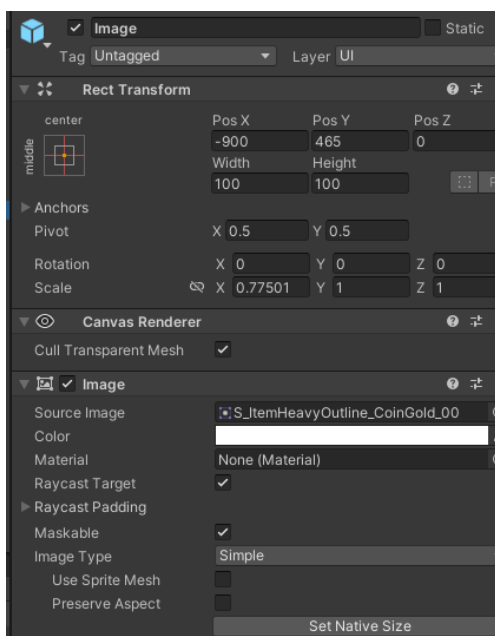


Рисунок 47 – Спрайт «Image».

К элементу «Text» прикреплен скрипт, который считает количество монет, подобранные игроком. Реализация данного объекта была рассмотрена ранее.

## 4 Тестирование приложения

### 4.1 План тестирования

Тестирование программного обеспечения — проверка соответствия реальных и ожидаемых результатов поведения программы, проводимая на конечном наборе тестов, выбранном определённым образом. Перед началом тестирования, нужно определиться с его планом проведения.

«Unit – тестирование» будет выполнять проверку работы простой механики, реализованной в игре. Данный вид тестирования выбран для проверки передвижения персонажа по локации, а также для проверки переключения анимаций в зависимости от направления движения героя.

Данный вид тестирования будет проводиться при помощи встроенного инструмента «Unity» - «Test Runner». Также для проверки нужно прописать код, представленный на «Рисунок 48».

```
0 references
public class MoveTest
{
    [Test]
    0 references
    public void MoveTestSimplePasses()
    {
        GameObject playerObject = new GameObject();
        PlayerController playerController = playerObject.AddComponent<PlayerController>();
        Rigidbody2D rb = playerObject.AddComponent<Rigidbody2D>();
        Animator animator = playerObject.AddComponent<Animator>();

        playerController.speed = 8;
        playerController.animator = animator;

        playerController.Start();
        playerController.Update();
        playerController.FixedUpdate();

        Assert.AreEqual(playerController.rb, rb);
        Assert.IsNotNull(playerController.animator);
        Assert.AreEqual(playerController.speed, 8f);
    }
}
```

Рисунок 48 – Код для проведения тестирования.

«Тест – кейс» предназначен для проверки всего функционала, реализованного в игре. С помощью данного вида тестирования будут проверены на корректную работу следующие аспекты игры:

- запуск игрового уровня;
- подбор предметов;
- переключение состояния у противника.

Данное тестирование будет проводиться в ходе прохождения игры, все результаты будут внесены в таблицу.

«Интеграционное тестирование» будет проверять как различные компоненты игры взаимодействуют между собой. В данном тесте будет рассмотрено, то, что статистика монет у игрока соответствует количеству монет, отображённых в интерфейсе игры.

#### 4.1 Оценка результатов проведения тестирования

В ходе выполнения «Unit - тестирования» программа выдала положительный результат, представленный на «Рисунок 49».

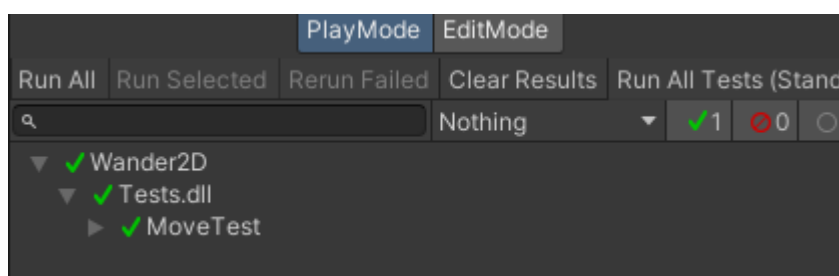


Рисунок 49 – Успешное проведение «Unit - тестирования».

Далее рассмотрим таблицу результатов тестирования игры с помощью «Тест – кейс».

Был проведен тест запуска игрового уровня, результаты представлены в «Таблица 1».

Таблица 1 – Тест запуска игрового уровня

<b>Название:</b>	Тест запуска уровня	
<b>Действие</b>	<b>Ожидаемый результат</b>	<b>Результат теста</b>
<b>Предусловие:</b>		
Запустить приложение	Отображается главное меню	Пройден
<b>Шаги теста:</b>		
Выбрать уровень	Загружается выбранный уровень	Пройден

Был проведен тест подбора предметов, результаты тестирования представлены в «Таблица 2».

Таблица 2 – Тест подбора монет, расположенных на локации

<b>Название:</b>	Тест подбора монет	
<b>Действие</b>	<b>Ожидаемый результат</b>	<b>Результат теста</b>
Предусловие:		
Запустить один из уровней	Уровень запущен	Пройден
Шаги теста:		
Прикоснуться к монете	Монета пропадает и зачитывается игроку в статистику	Пройден
Прикоснуться к цветку	Цветок пропадает и засчитывается игроку в статистику	Пройден

Был проведен тест на переключение состояний врага с патрулирования на преследование и наоборот, результаты тестирования представлены в «Таблица 3».

Таблица 3 – Тест переключения состояния противника

<b>Название:</b>	Тест состояний у противника	
<b>Действие</b>	<b>Ожидаемый результат</b>	<b>Результат теста</b>
Предусловие:		
Запустить второй уровень	Уровень запущен	Пройден
Шаги теста:		
Подойти к врагу на близкое расстояние	Враг начинает преследовать игрока	Пройден
Отбежать от врага на большое расстояние	Враг возвращается к патрулированию	Пройден

Теперь необходимо рассмотреть результаты интеграционного тестирования. Результаты представлены в «Таблица 4».

Таблица 4 – Тест соответствия монет

<b>Название:</b>	Тест на соответствие статистика монет игрока на верное их отображение внутри игрового интерфейса	
<b>Действие</b>	<b>Ожидаемый результат</b>	<b>Результат теста</b>
<b>Предусловие:</b>		
Запустить игровой уровень	Уровень запущен	Пройден
<b>Шаги теста:</b>		
Подойти к монете	Монета подобрана и защитилась в статистику игрока	Пройден
Проверить интерфейс	Число монет, отображаемых внутри интерфейса, соответствует их количеству в статистике	Пройден

Все проделанные тесты были успешно пройдены итоговой версией приложения, следовательно, приложение работает корректно.

## Заключение

Целью курсового проекта была разработка «2D» игры с видом сверху, позволяющей пользователям погрузиться в мир жанра игр "бродилок". Используя популярный игровой движок «Unity» и язык программирования «C#» для написания скриптов, была создана игра, предоставляющая возможность пользователям погрузиться в уникальный геймплей данного жанра.

Начальным этапом процесса разработки был тщательный анализ предметной области, что позволило выявить ключевые требования к будущей игре. В процессе работы над проектом был накоплен ценный опыт работы с игровым движком «Unity». Этот опыт позволил не только успешно завершить проект, но и обнаружить новые функциональности и возможности платформы, что станет основой для разработки более качественных проектов в будущем.

В процессе написания скриптов для реализации игровых механик возникали некоторые трудности. Однако благодаря обширному комьюнити «Unity», которое предоставляет множество альтернативных решений и советов, эти сложности были успешно преодолены.

Разработанный продукт обладает потенциалом для дальнейших улучшений. Планируется расширение игры новыми механиками для создания более интересного и насыщенного геймплея. Дополнительно планируется улучшение пользовательского интерфейса с помощью эффектов "Fade In" и "Fade Out" при переходах между уровнями, что придаст игре большую динамику.

Также не стоит забывать про добавление новых анимаций для героев, локаций, предметов и врагов. Это позволит создать более живой и игровой мир, который погрузит игрока еще глубже в игровой процесс.

В целом, данная игра представляет собой не только завершенный проект, но и отправную точку для дальнейших творческих разработок в области игровой индустрии.



## Список источников

- 1      Официальная документация игрового движка «Unity». Режим доступа URL: <https://docs.unity3d.com/Manual/> - дата обращения: 01.12.2023;
- 2      Работа с «TileMap». Режим доступа URL: <https://habr.com/ru/articles/412765/> - дата обращения: 01.12.2023;
- 3      Реализация передвижения персонажа. Режим доступа URL: [https://unityhub.ru/guides/peredvizhenie-personazha-v-unity-2d-i-3d\\_59](https://unityhub.ru/guides/peredvizhenie-personazha-v-unity-2d-i-3d_59) - дата обращения: 05.12.2023;
- 4      Анимации в «Unity». Режим доступа URL: <https://learn.unity.com/tutorial/animating-a-sprite-with-the-2d-animation-package#/> - дата обращения: 05.12.2023;
- 5      Компонент «Box Collider» в «Unity». Режим доступа URL: <https://www.codinblack.com/colliders-and-triggers-in-unity3d/> - дата обращения: 07.12.2023;
- 6      Работа с «Canvas». Режим доступа URL: <https://stfalcon.com/ru/blog/post/unity-meets-canvas> - дата обращения: 09.12.2023;
- 7      Работа с сценами в «Unity». Режим доступа URL: <https://thiscodedoesthis.com/change-scenes-in-unity/> - дата обращения: 11.12.2023;
- 8      Создание патрулирующего врага. Режим доступа URL: [https://unityhub.ru/guides/kak-sdelat-patrulirovanie-personazha-v-unity\\_58](https://unityhub.ru/guides/kak-sdelat-patrulirovanie-personazha-v-unity_58) - дата обращения: 15.12.2023;
- 9      Получение урона игроком. Режим доступа URL: <https://stackru.com/questions/40350705/zastavte-igroka-poluchat-uron-v-unity-2d> - дата обращения: 16.12.2023;
- 10     Создание преследующего врага. Режим доступа URL: [https://unityhub.ru/guides/presledovanie-vrazheskogo-personazha-za-igrokom\\_5](https://unityhub.ru/guides/presledovanie-vrazheskogo-personazha-za-igrokom_5) - дата обращения: 26.12.2023.

# Приложение А (обязательно)

## Диаграмма прецедентов

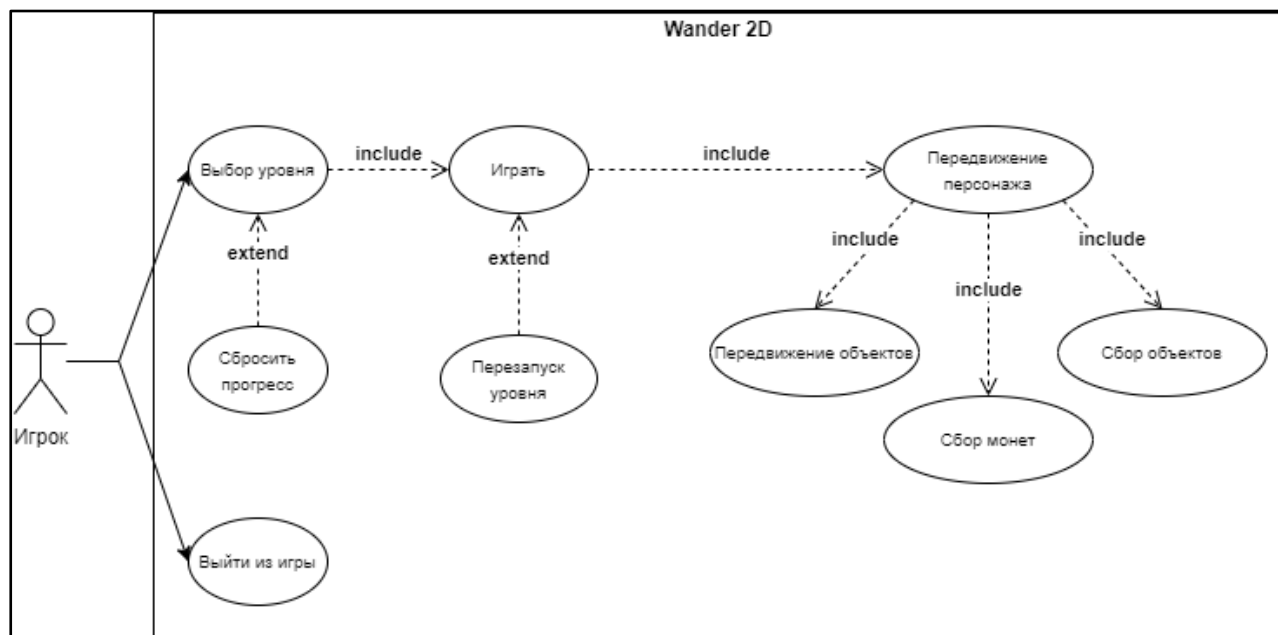


Рисунок А.1 – Диаграмма прецедентов.

## Приложение Б (обязательно)

### Диаграмма деятельности

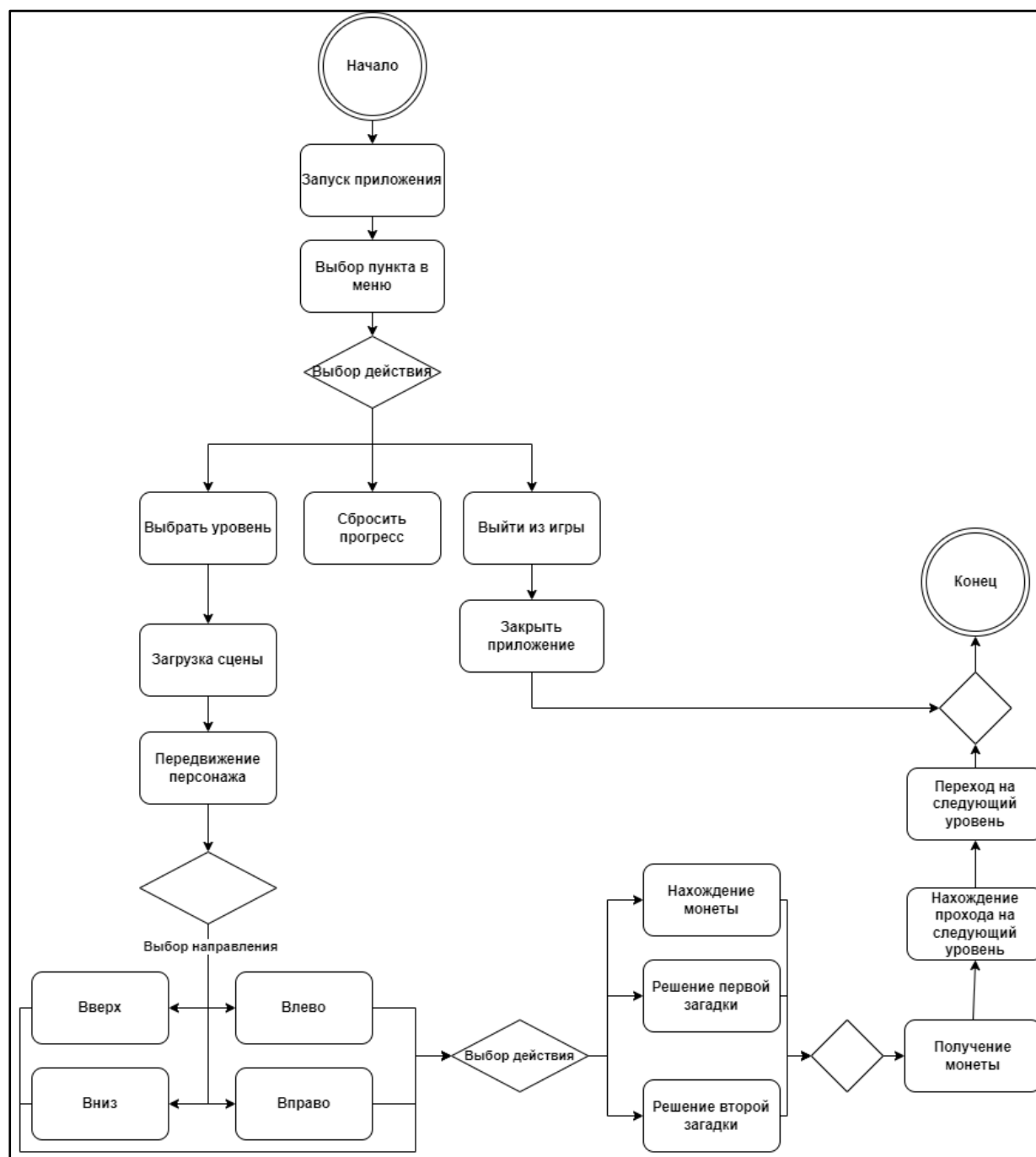


Рисунок Б.1 – Диаграмма деятельности.

# Приложение В (обязательно)

## Диаграмма классов

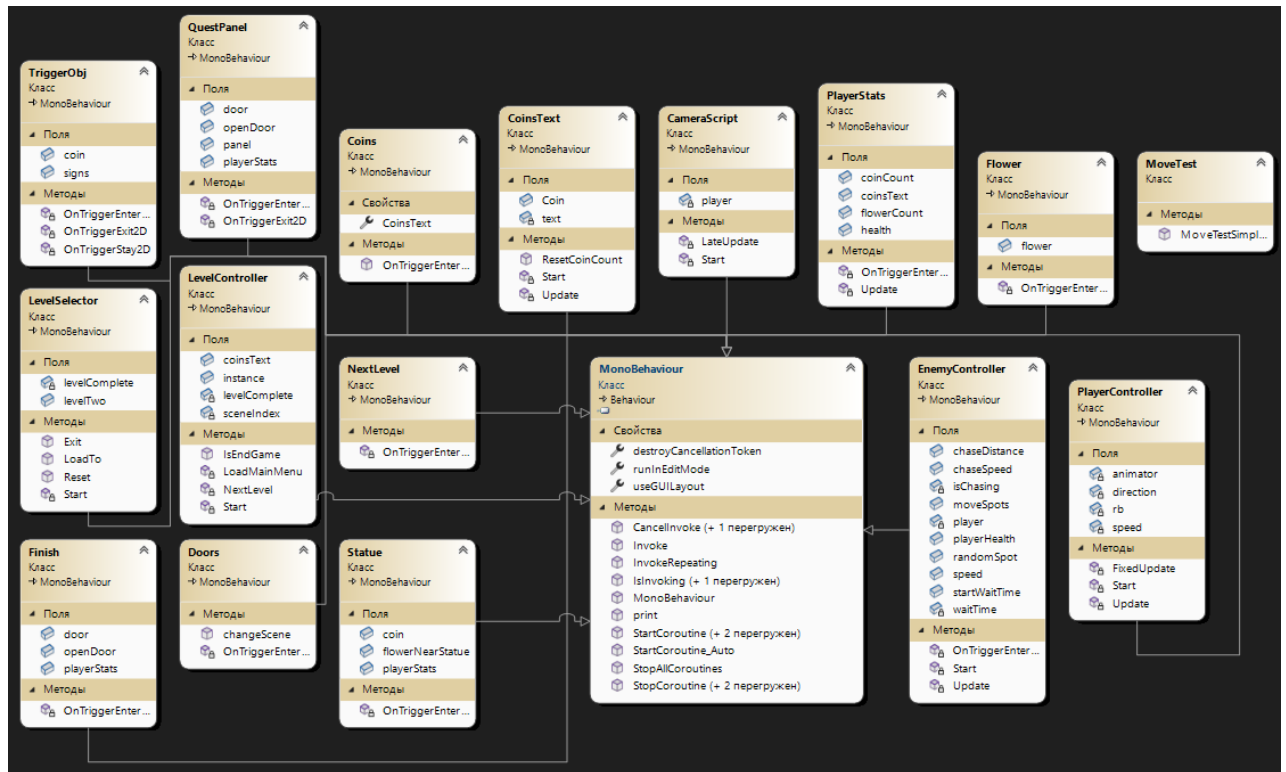


Рисунок В.1 – Диаграмма классов.