

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет
вычислительной техники

Кафедра
«Вычислительная техника»

Направление подготовки: 09.03.01 «Информатика и вычислительная техника»

Профиль подготовки: «Вычислительные машины, комплексы, системы и сети»

БАКАЛАВРСКАЯ РАБОТА

на тему

ИГРОВОЕ ПРИЛОЖЕНИЕ ПОД ОС
АНДРОИД

Студент


(подпись, дата) 13.06.23

Буданов Дмитрий Сергеевич

Руководитель


(подпись, дата) 13.06.23

Юрова О. В

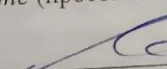
Нормоконтролёр


(подпись, дата) 13.06.23

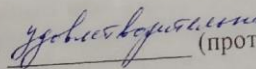
Бычков А. С.

Работа допущена к защите (протокол зас. кафедры от 02.06.2023г. № 14)

Заведующий кафедрой

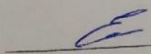

(подпись)

М.А. Митрохин
(фамилия, инициалы)

Работа защищена с отметкой  (протокол заседания ГЭК от

20.06.2023 № 9)

Секретарь ГЭК



Е.И. Гурин

Пенза 2023

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет
вычислительной техники

Кафедра
«Вычислительная техника»

«Утверждаю»
Заведующий кафедрой
М.А. Митрохин
«26» 04 2023 г.

**ЗАДАНИЕ
НА БАКАЛАВРСКУЮ РАБОТУ**

1. Студент Буданов Дмитрий Сергеевич, гр.19ВВВ1 факультета вычислительная техника

2. Тема работы: «Игровое приложение под ОС Андроид»

Тема утверждена приказом ПГУ № 458/о от "26" 04 2023г.

3. Руководитель бакалаврской работы к.т.н., доцент каф. ВТ ПГК Юрова О. В.

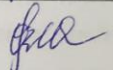
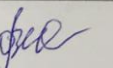
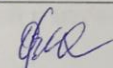
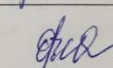
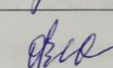
4. Задание на работу (назначение разработки, исходные данные и т.п.)

Разработать игровое приложение под ОС Андроид. Приложение должно иметь яркую графику, интуитивно понятный игровой процесс, несколько уровней, обладать отзывчивым управлением, качественным звуковым сопровождением, интерфейсом и быть адаптированным к разным разрешениям экранов.

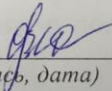
5. Перечень подлежащих разработке вопросов

Требуется произвести анализ предметной области: изучить существующие игровые приложения под ОС Андроид. Выбрать язык программирования для приложения и среду разработки, изучить и понять нюансы разработки игрового приложения.

6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1	Анализ предметной области	28	26.04.2023-29.04.2023	
2	Анализ требований	16	30.04.2023-02.05.2023	
3	Выбор технологий, применяемых при проектировании и разработке	16	03.05.2023-05.05.2023	
4	Разработка игрового приложения	96	06.05.2023-17.05.2023	
5	Тестирование и исправление ошибок	50	18.05.2023-25.05.2023	
6	Оформление пояснительной записки	48	26.05.2023-31.05.2023	

Дата выдачи задания "26" 04 2023г.

Руководитель бакалаврской работы _____
(подпись, дата) 

Задание к исполнению принял студент _____
(подпись, дата) 

Работу к защите допустить
Декан факультета _____
(подпись, дата)  А.Р. Рязанов
20.06.2023

Реферат

Пояснительная записка содержит 80 листов, 23 рисунка, 2 таблицы, 3 приложения

ИГРОВОЕ ПРИЛОЖЕНИЕ, *ANDROID*, *UNITY*, ПЛАТФОРМЕР, *MICROSOFT VISUAL STUDIO 2022*, *C#*

В рамках данной работы была поставлена цель - разработать игровое приложение в жанре платформера для операционной системы *Android*. Основными требованиями были интуитивно понятный интерфейс и соответствие правилам данного жанра.

Для достижения этой цели был проведен анализ требований и изучение существующих решений в области разработки игр.

В процессе создания приложения были использованы различные инструменты разработки, такие как язык программирования *C#*, среда разработки *Unity* и интегрированная среда разработки программного обеспечения *Microsoft Visual Studio 2022*.

В результате успешно выполненной работы было создано и отлажено приложение, которое удовлетворяет большинству поставленных задач.

					ПГУ 1.090301.02.001 ПЗ			
Изм.	Лист	№ докум.	Подпись	Дата	Игровое приложение под ОС Андроид Пояснительная записка	Лит.	Лист	Листов
Разраб.		Буданов Д.С.					2	80
Провер.		Юрова О.В.				ФВТ, каф ВТ		
Н. Контр		Бычков А.С.				гр. 19ВВВ1		
Утверд.		Митрохин М.А.						

Содержание

Перечень сокращений и обозначений.....	5
ВВЕДЕНИЕ.....	6
1. Игровое приложение под ОС Андроид.....	7
1.1 Анализ предметной области и постановка задачи.....	7
1.1.2 Анализ существующих решений для реализации проекта.....	8
1.1.3 Обзор аналогов разрабатываемого игрового приложения	13
1.1.4 Постановка задачи	18
1.1 Выбор инструментов разработки.....	19
1.2.1 Среда разработки	19
1.2.2 Язык программирования	20
1.3 Проектирование приложения.....	21
1.3.1 Вводная информация.....	21
1.3.2 Проектирование базовой логики и основных компонентов.....	22
1.3.3 Структура уровней.....	22
1.3.4 Графический интерфейс.....	24
1.4. Реализация.....	24
1.4.1 Вводная информация.....	24
1.4.2 Базовая логика и основные компоненты	26
1.4.3 Реализация анимаций	32
1.4.4 Реализация ловушек и препятствий.....	34
1.4.5 Графический интерфейс.....	38
1.5.Тестирование	40
1.5.1 Результаты тестирования	40
1.5.2 Результаты тестирования	44
1.5.3 Вывод по результатам тестирования	47
2. Охрана труда.....	49
2.1 Требования к организации рабочего места с ПК.....	49
2.2 Требования к освещению помещений и рабочих мест	50

2.3 Режим труда и отдыха при работе с компьютером	50
Заключение	53
Список использованных источников	54
Приложение А – Листинг программы.....	55
Приложение Б – <i>UML</i> - диаграммы.....	66
Приложение В – Презентация.....	71

Перечень сокращений и обозначений

В настоящей работе применяют следующие сокращения и обозначения
ОС – операционная система

3D (*Three-dimensional*) – компьютерная трехмерная графика

2D (*Two-dimensional*) – компьютерная двухмерная графика

API (*Application Programming Interface*) – интерфейс прикладного программирования, который представляет собой набор определений и протоколов для создания и интеграции прикладного программного обеспечения.

IDE (*Integrated Development Environment*) – интегрированная среда разработки.

SDK (*Software Development Kit*) - набор инструментов для разработки программного обеспечения в одном устанавливаемом пакете

					ПГУ1.090301.02.001 ПЗ	Лист
						5
Изм.	Лист	№ докум.	Подпись	Дата		

Введение

В современном мире смартфоны стали важной частью нашей жизни. Эти устройства позволяют нам общаться, выполнять разные задачи, получать информацию и проводить время в развлекательных приложениях. Одними из самых популярных развлекательных приложений являются игры. Они представляют собой огромную индустрию, в которой находятся миллионы пользователей по всему миру. Эти игры сочетают в себе уникальные элементы геймплея и дизайна, которые делают их по-своему захватывающими и увлекательными. Поэтому разработка мобильных игр – это ключевая отрасль в индустрии разработки программного обеспечения.

Одной из наиболее распространенных мобильных операционных систем является *Android*, которая разработана компанией *Google*. Данная система используется на миллионах смартфонов, планшетов и других устройств по всему миру. Такая система имеет множество различных функций и возможностей для разработчиков. Поэтому, разработка мобильных игровых приложений на ОС *Android* является интересной и актуальной темой.

1 Игровое приложение под ОС Андроид

1.1 Анализ предметной области и постановка задачи

1.1.1 Анализ темы

Мобильные игры возникли в результате развития смартфонов и планшетов, которые стали неотъемлемыми устройствами в повседневной жизни людей.

Настоящий прорыв в мобильных играх произошел с выпуском первых смартфонов, таких как *iPhone* и смартфонов на базе операционной системы *Android*. Такие смартфоны предлагали мощные, на то время, процессоры, качественные графические возможности и сенсорные экраны, что создало новые перспективы для игровой индустрии.

В 2008 году были запущены первые магазины приложений, такие как *App Store* от *Apple*, а также *Goggle Play* для устройств с системой *Android*. Это предоставило разработчикам возможность создавать и распространять свои игры напрямую на мобильных устройствах. Такое событие позволило независимым разработчикам стимулировать рост игровой индустрии на мобильных платформах.

За последние десятилетие мобильные игровые приложения получили огромное развитие. Технические возможности смартфонов значительно улучшились, что позволило разработчикам создавать более сложные и визуально улучшенные игры. Также, благодаря распространению высокоскоростного интернета и развитию мобильных сетей произошел бум многопользовательских игр, в которых игроки взаимодействуют с друг другом в реальном времени.

Такое развитие и широчайшее распространение смартфонов сделало мобильные игры популярным способом развлечения и отдыха.

Мобильные игры предлагают широкий спектр жанров таких, как: аркады, головоломки, стратегии, ролевые игры и многие другие. Такое

					ПГУ1.090301.02.001 ПЗ	Лист
						7
Изм.	Лист	№ докум.	Подпись	Дата		

разнообразие игровых жанров даёт возможность каждому пользователю найти что-то для себя и получить удовольствие от игрового процесса. Также стоит отметить, что игры на смартфонах обладают рядом уникальных преимуществ, таких как мобильность и удобство использования.

Разработка мобильных игр требует специализированных знаний и навыков. Разработчики мобильных игр на *Android* используют такие языки программирования, как *Java* или *C#*, а также используют специализированные инструменты разработки. Разработчики также должны для обеспечения оптимальной производительности и пользовательского опыта учитывать особенности мобильных устройств такие, как ограниченные ресурсы и различные размеры экранов.

Кроме того, важно учитывать требования и предпочтения аудитории при разработке игр. Разные возрастные группы и интересы пользователей требуют разных жанров, дизайна и механик игры. Проведение исследований рынка мобильных игр и анализ конкурентов помогут определить потребности целевой аудитории и создать игру, которая будет привлекательной и успешной на рынке мобильных игр.

Важным аспектом разработки мобильных игр является монетизация. Существует несколько моделей монетизации, таких как реклама, внутриигровые покупки и премиум-версии. Разработчики должны обдуманно подходить к выбору модели монетизации, чтобы обеспечить прибыльность проекта и предложить своим игрокам комфортный игровой опыт.

1.1.2 Анализ существующих решений для реализации проекта

В настоящее время доступно множество платных и бесплатных инструментов для разработки игр. Этот спектр инструментов варьируется от простых библиотек до крупных платформ с обширным функционалом [1]. Для разработки качественного приложения нужно выбрать готовую среду разработки, специально предназначенную для создания игр. Для этого рассмотрим некоторые из них.

Unity - это кросс-платформенный игровой движок, разработанный компанией *Unity Technologies*. Этот движок является одним из самых популярных и широко используемых инструментов разработки игр, включая игры для ОС *Android*.

Основные особенности *Unity*:

- кросс-платформенность. *Unity* позволяет разрабатывать игры для разных платформ таких, как: *Android*, *iOS*, *Windows*, *macOS*, *Linux*, *Xbox*, *PlayStation*. Это позволяет увеличить охват аудитории;

- 2D и 3D разработка. *Unity* поддерживает как игры с двухмерной графикой, так и с трехмерной. Это означает, что в данном инструменте вы сможете создавать как простые 2D платформеры, так и более сложные трехмерные проекты с реалистичной графикой и физикой;

- интуитивный интерфейс и редактор. *Unity* предоставляет удобный и простой в использовании визуальный редактор, позволяющий создавать игровые сцены, настраивать объекты, применять графические эффекты и многое другое. Благодаря такой простоте в использовании даже новички могут начать разрабатывать игры без необходимости написания большого количества кода;

- скриптинг на *C#*. В *Unity* для создания игровой логики используется язык программирования *C#*. Этот язык является очень популярным и мощным языком;

- расширяемость. *Unity* поддерживает сторонние плагины и ресурсы, что даёт возможность расширить функциональность и возможности движка. Благодаря этому, становится возможным использование готовых решений из *Asset Store* или разработка своих собственных инструментов для оптимизации и улучшения проектов;

- сообщество и поддержка. *Unity* имеет большое и активное сообщество разработчиков. Вы можете обмениваться опытом, найти обучающие материалы

и ресурсы для изучения *Unity*. Кроме того, *Unity Technologies* предоставляет документацию, видеоуроки и форумы для помощи разработчикам.

Таким образом *Unity* является мощным и гибким инструментом для создания игр, особенно для мобильных платформ, таких как *Android*. Он обладает множеством возможностей и поддержкой кросс-платформенной разработки, что делает его популярным среди разработчиков игр.

Android Studio - это интегрированная среда разработки (IDE) для создания приложений под операционную систему *Android*. Это официальная IDE, разработанная и поддерживаемая *Google*, специально предназначенная для создания мобильных приложений для устройств на базе ОС *Android*. Рассмотрим особенности данной среды разработки:

- профессиональная разработка под *Android*. *Android Studio* предоставляет все необходимые инструменты и функции для разработки приложений под *Android*. Оно поддерживает полный цикл разработки, включая создание проектов, редактирование, отладку кода и так далее;

- интеграция с *Android SDK*. *Android Studio* поставляется с полным комплектом *Android SDK*, который содержит все необходимые библиотеки, инструменты и документацию для разработки под ОС *Android*, что позволяет легко получить доступ к последним версиям *Android API* и использовать их в своих проектах;

- удобный редактор кода. *Android Studio* предлагает мощный редактор кода с функциями автодополнения, проверки синтаксиса, рефакторинга и другие инструменты, которые упрощают процесс написания кода для разработки приложений. Данная среда также поддерживает различные языки программирования такие, как: *Java* и *Kotlin*;

- визуальный редактор интерфейса. *Android Studio* имеет визуальный редактор интерфейса, позволяющий создавать пользовательские интерфейсы с помощью функции *drag-and-drop*;

- инструменты для отладки и профилирования. *Android Studio* предлагает инструменты для отладки и профилирования приложений, которые дают возможность разработчикам исправлять ошибки, оптимизировать производительность и анализировать использование ресурсов приложения;

- поддержка для различных устройств и версий *Android*. *Android Studio* поддерживает разработку приложений для всех устройств на базе *Android*, включая смартфоны, планшеты, *Android TV*, *Android Auto* и *Android Wear*. Оно также поддерживает различные версии данной ОС, что позволяет создавать приложения, которые поддерживаются огромным количеством устройств.

Итак, *Android Studio* предоставляет все необходимые инструменты и ресурсы для разработки мобильных приложений под ОС *Android*. Оно позволяет разработчикам создавать инновационные и привлекательные для пользователей приложения, а также эффективно управлять процессом разработки.

Godot Engine - это бесплатный и открытый исходный кросс-платформенный игровой движок, предназначенный для создания 2D, 3D игр и других приложений. Этот движок разрабатывается и поддерживается сообществом *Godot Engine Community*, имеет интуитивный интерфейс и мощные функциональные возможности. Рассмотрим основные особенности данного движка:

- кросс-платформенность. *Godot Engine* поддерживает различные операционные системы, включая *Windows*, *macOS*, *Linux*, *Android*, *iOS* и множество других платформ, что позволяет создавать приложения, которые могут быть запущены на самых различных устройствах;

- интуитивно понятный редактор сцен. *Godot Engine* предлагает удобный визуальный редактор сцен, позволяющий разработчикам создавать игровые объекты и компоненты, расставлять и настраивать элементы

игрового мира без необходимости написания кода, что упрощает и ускоряет процесс разработки;

- гибкая система скриптинга. *Godot Engine* поддерживает несколько языков программирования: *GScript* (схожий с *Python*), *C#*, *VisualScript* и другие. Разработчики могут выбрать наиболее удобный для них язык;

- работа с 2D и 3D графикой. *Godot Engine* обладает мощными возможностями для создания как 2D, так и 3D графики. Данный движок предлагает множество инструментов и эффектов, таких как анимации, спрайты, освещение, частицы и так далее. Это предоставляет разработчикам возможность создавать впечатляющий визуальный стиль для своих игр или приложений;

- расширяемость и модульность. *Godot Engine* имеет модульную архитектуру и позволяет разработчикам создавать свои собственные плагины и расширения, служащие для добавления новых функциональных возможностей в движок;

- бесплатность и открытый исходный код. *Godot Engine* является полностью бесплатным и распространяется под лицензией *MIT*. Это означает, что разработчики могут свободно использовать и изменять исходный код движка, а также создавать коммерческие приложения без каких-либо ограничений.

Таким образом, *Godot Engine* получил признание в разработческом сообществе благодаря своей гибкости, мощности и бесплатному распространению. Он предоставляет разработчикам все необходимые инструменты и ресурсы для создания уникальных игр и приложений на самых разных платформах.

Исходя из всего сказанного выше была сформирована таблица 1.

					ПГУ1.090301.02.001 ПЗ	Лист
						12
Изм.	Лист	№ докум.	Подпись	Дата		

Таблица 1 – Сравнение платформ для разработки мобильных игровых приложений

Редактор	Достоинства	Недостатки
<i>Unity</i>	Кроссплатформенность, высокая функциональность, крупное сообщество, удобство в работе и компиляции, большое количество библиотек расширяющих функционал.	Закрытый исходный код, платные продвинутые функции
<i>Android Studio</i>	Официальная среда разработки для <i>Android</i> , Широкий набор инструментов и функциональность	Сложность освоения, требовательность к ресурсам
<i>Godot Engine</i>	Бесплатность и открытый код, легкость использования, интегрированный редактор	Ограничение экосистемой, недостаток документации

1.1.3 Обзор аналогов разрабатываемого игрового приложения

Рассмотрим мобильную игру *Geometry Dash*. *Geometry Dash* – это аркадная 2D игра от *RobTop Games*, которая напоминает очень популярную игру *Flappy Bird* и другие подобные аркады своей интенсивной динамикой. Она заставляет игрока сосредоточиться и не отрываться ни на секунду, пока не будет пройден каждый мучительный уровень.

В данной игре представлены два существующих режима игры: режим практики, где можно создавать точки сохранения в любой части уровня, и обычный режим, который уже через несколько минут натянет ваши нервы,

словно струны только что настроенной гитары. *Geometry Dash* изображена на рисунке 1.1.

Ошибки в *Geometry Dash* не прощаются, и даже небольшая оплошность отправит игрока в самое начало уровня. Вы будете проходить одно и то же, пока не достигнете финиша или не сдадитесь и не найдете более расслабляющую альтернативу этой игры. Чтобы поддерживать желание играть снова и снова, игра использует потрясающее и качественное музыкальное сопровождение, которое играет только в обычном режиме. Заметно, что разработчики позаботились о возможности загрузки этих саундтреков прямо из игры, для этого есть специальная кнопка в главном меню.



Рисунок 1.1 – *Geometry Dash*

Графика в *Geometry Dash* имеет неоновый стиль, который хорошо сочетается с электронной музыкой и высоким темпом игры. Управление осуществляется простыми нажатиями в любой части экрана. Игра реагирует на нажатия с небольшой задержкой, к которой можно привыкнуть, хотя не заметить ее присутствия нельзя.

Эта задержка может быть немного разочаровывающей для пользователей, особенно при выполнении сложных движений и точных прыжков. Однако, с привычкой и практикой, игроки могут научиться справляться с этой неприятной особенностью управления и достигать максимальной точности в своих движениях.

Несмотря на недостатки, *Geometry Dash* всегда предлагает вызов и приятные награды для своих игроков. Каждый уровень полон опасностей и сложного расположения платформ, требующего мгновенной реакции и точности. Пройти уровень с первого раза становится настоящим достижением, вызывающим чувство удовлетворения и приводящим к желанию продолжать игру, чтобы преодолеть следующий испытание.

Далее рассмотрим игру *Super Mario Run*. *Super Mario Run* - это популярная аркадная игра, разработанная и выпущенная компанией *Nintendo* для устройств на платформе *Android*. Она является официальной адаптацией знаменитой серии игр *Super Mario Bros.*, которая стала иконой в мире видеоигр.

Игра *Super Mario Run* представляет собой платформер, в котором главный герой Марио автоматически бежит вперед, а игроку нужно управлять его прыжками и выполнением различных трюков, чтобы преодолеть препятствия, собрать монеты и спасти принцессу Пич. Управление осуществляется с помощью простых жестов, что делает игру доступной для всех возрастных категорий. Пример геймплея данной игры изображен на рисунке 1.2.



Рисунок 1.2 – *Super Mario Run*

Одной из ключевых особенностей *Super Mario Run* является ее простота и интуитивность. Игра имеет непрерывный режим, в котором игроки просто прыгают по уровням, стараясь достичь финиша. Кроме того, есть режим "*Toad Rally*", где игроки соревнуются с другими пользователями, собирая максимальное количество монет и привлекая туристов-грибочков.

Графика в игре выполнена в ярком и привлекательном стиле, вдохновленном классическими играми серии *Super Mario*. Звуковое сопровождение также выполнено на высоком уровне, с воссозданием знаменитых звуков и мелодий из оригинальных игр.

Super Mario Run является бесплатной игрой с возможностью покупки полной версии для доступа ко всем уровням и функциям. В игре также есть элементы внутриигровых покупок, которые позволяют приобретать дополнительные персонажи и улучшения.

В целом, *Super Mario Run* - это отличная аркадная игра для любителей платформеров. Она сочетает простоту управления с захватывающим геймплеем и великолепным визуальным исполнением.

Последняя игра, которую мы рассмотрим, *Yeah Bunny 2*. *Yeah Bunny 2* - это веселая и захватывающая игра в жанре платформера для устройств на платформе *Android*.

Главным героем игры является милый кролик, который отправляется в захватывающее путешествие, чтобы спасти свою возлюбленную, которую похитил злобный лис.

Геймплей *Yeah Bunny 2* является простым и интуитивно понятным. Вы будете управлять кроликом, прыгая по платформам и собирая морковки на своем пути. Кролик может выполнять двойные прыжки и использовать различные предметы и специальные умения, чтобы преодолеть преграды и победить врагов. Уровни разнообразны и содержат различные ловушки и головоломки, которые требуют от вас ловкости и реакции.

Графика в игре яркая и привлекательная, с ретро-стилизрованными пиксельными визуальными эффектами. Музыкальное сопровождение также веселое и подходит к игровой атмосфере. На рисунке 1.3 показана игра *Yeah Bunny 2*.

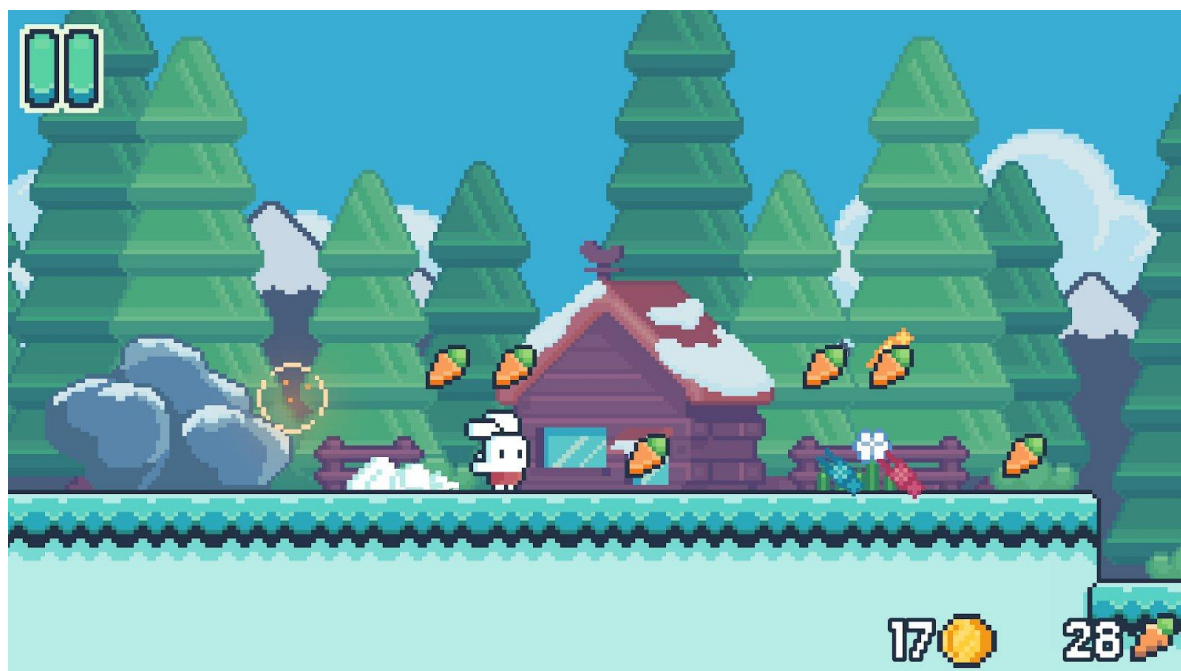


Рисунок 1.3 – *Yeah Bunny 2*

Игра является бесплатной с внутриигровыми покупками для покупки дополнительных предметов и улучшений. Однако она также предлагает возможность играть и прогрессировать без совершения покупок.

В целом, *Yeah Bunny 2* - это веселая и захватывающая игра, которая подходит для всех любителей платформеров. Она предлагает простой и привлекательный геймплей, красочную графику и разнообразие уровней и контента.

1.1.4 Постановка задачи

Необходимо разработать игровое приложение под ОС Android. Приложение должно удовлетворять следующие требования:

- интуитивно понятный игровой процесс;
- яркая и привлекательная графика;
- несколько уровней с разнообразными и интересными ловушками и препятствиями;
- важно обеспечить баланс между сложностью игры и доступностью для всех уровней игроков. Игра должна быть в меру сложная, для того чтобы бросать пользователям испытание;
- реализовать качественное и увлекательное звуковое сопровождение;
- управление в платформере должно быть интуитивным и отзывчивым;
- адаптировать приложение к разным разрешениям экранов и аппаратным возможностям устройств;
- реализовать интерфейс пользователя включающий в себя:

1) главное меню с кнопкой начала игры и кнопкой закрытия приложения;

2) внутриигровой интерфейс с кнопкой выхода и инструментами управления персонажем.

В связи с анализом предметной области была разработана диаграмма вариантов использования главного меню игры. Данная диаграмма продемонстрирована на рисунке 1.4.

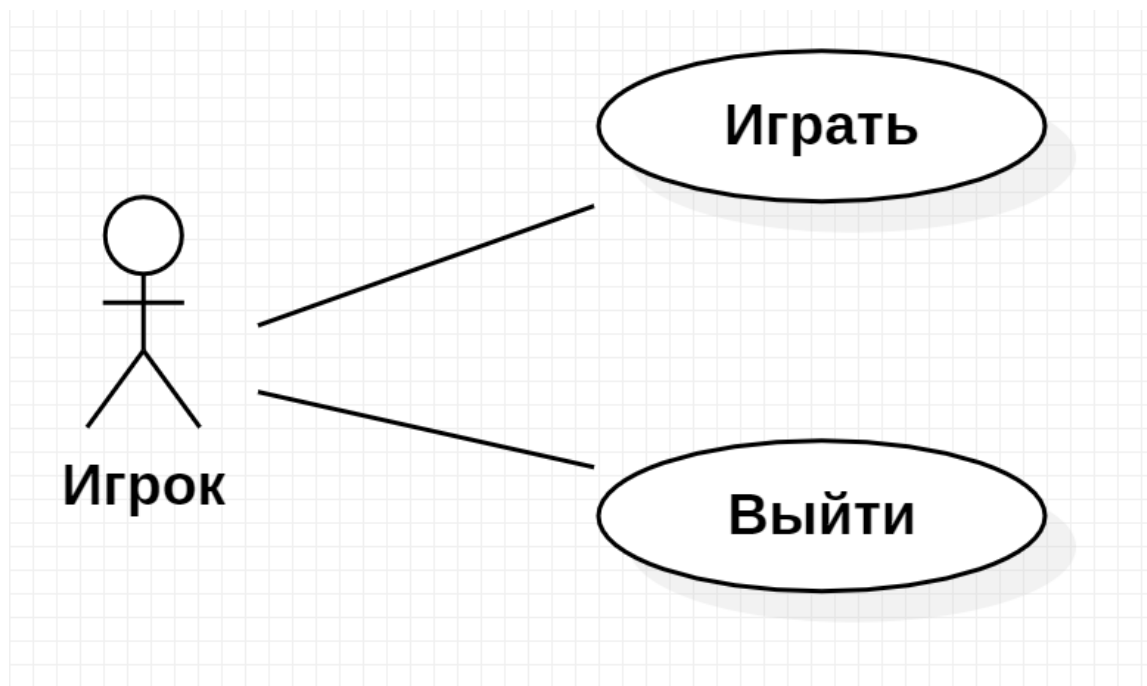


Рисунок 1.4 – Диаграмма использования главного меню

1.1 Выбор инструментов разработки

1.2.1 Среда разработки

Была выбрана среда разработки *Unity*, о которой были рассмотрены все достоинства и недостатки выше. Графический интерфейс редактора прост и представлен на рисунке 1.5.

Интерфейс состоит из нескольких блоков. В нижнем левом углу находится блок "*Project*", который используется для работы с ресурсами проекта. В центре расположено окно "*Game*", где отображается то, что пользователь увидит при запуске игры, а рядом с ним блок "*Scene*", позволяющий работать с графическими элементами игры. Вся работа происходит в этом окне. В правой части находится блок "*Inspector*", который отображает информацию о выбранном объекте и его свойствах. Слева расположено окно "*Hierarchy*", отображающее все элементы, используемые в проекте [2].

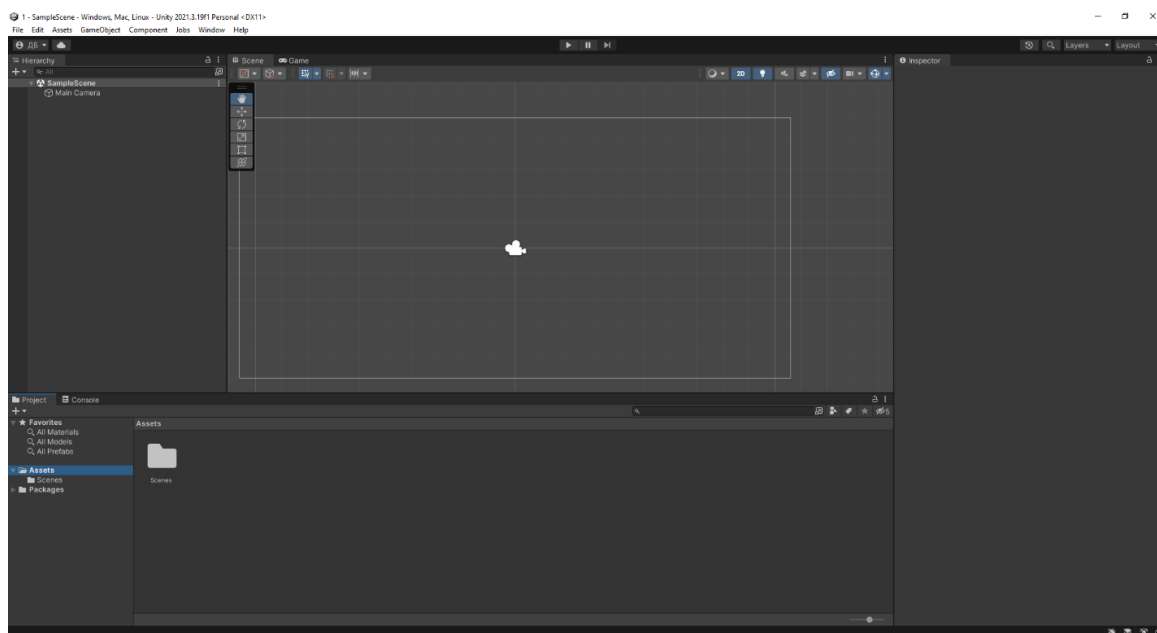


Рисунок 1.5 – Интерфейс Unity

Код приложения будет разрабатываться с использованием *Microsoft Visual Studio 2022*. *Microsoft Visual Studio* - это набор продуктов компании *Microsoft*, включающих интегрированную среду разработки программного обеспечения и другие инструментальные средства. Эти продукты позволяют разрабатывать как консольные приложения, так и приложения с графическим интерфейсом для платформ *Windows*, *Xbox*, *Windows Phone*, *Android*, *macOS* и *iOS*.

1.2.2 Язык программирования

В процессе разработки был выбран язык программирования *C#*. *C#* был разработан и выпущен *Microsoft* в 2001 году. Это строго типизированный объектно-ориентированный язык программирования, который предоставляет современным разработчикам гибкость и легкость для создания программного обеспечения. Он также является кроссплатформенным языком программирования. Приложения, написанные на *C#*, используют среду выполнения *.NET* и библиотеки классов. В настоящее время *.NET* представляет собой платформу и среду программирования для кроссплатформенной разработки. Благодаря платформе *.NET* можно

					ПГУ1.090301.02.001 ПЗ	Лист 20
Изм.	Лист	№ докум.	Подпись	Дата		

создавать приложения, которые могут быть запущены на платформах *Windows, Linux* и *macOS*.

В процессе разработки приложений в *Unity* мы используем среду выполнения *Mono*. Она представляет собой реализацию *.NET Framework*, основанную на стандартах *ECMA* для *C#* и *Common Language Runtime*. Следует также упомянуть стандартный класс *Unity - MonoBehaviour*, который является базовым классом для всех скриптов. При помощи этого класса программист может получать доступ к игровым объектам сцены, например, к объекту персонажа, чтобы заставить его двигаться. Если создать класс без наследования от *MonoBehaviour*, то он будет являться обычным классом *.NET*, и его нельзя будет использовать напрямую с *Unity* редактором.

1.3 Проектирование приложения

1.3.1 Вводная информация

Важно отметить, что основные этапы проектирования проекта для игры под *Android* выполнялись параллельно друг другу. Это было обусловлено тем, что для разработки программного кода не требовались готовые анимации или другие ресурсы. Для проверки функциональности кода было достаточно использовать стандартные примитивы *Unity*. Такой подход значительно увеличивает скорость работы над проектом. С самого начала проект можно разделить на несколько составных частей для удобной реализации: основная игровая логика, включающая игрока и окружение; графический интерфейс.

Важно учесть, что наше приложение может запускаться не на всех мобильных устройствах из-за системных требований. Поэтому игра должна быть оптимизирована и не требовать значительных ресурсов. Так же из-за различных размеров дисплея, его разрешения и соотношения сторон, игровая сцена может выходить за рамки экрана или наоборот. Поэтому важно оптимизировать изображение под все возможные виды дисплеев.

Всё приложение фактически состоит из нескольких сцен, в которых размещаются объекты игры. Эти сцены могут использоваться для создания главного меню, отдельных уровней и других элементов игры. В каждой сцене можно размещать объекты окружения, преграды, декорации, поэтапно создавая дизайн игры.

1.3.2 Проектирование базовой логики и основных компонентов

Перед началом создания игры необходимо выбрать жанр. В рамках данной работы мы будем разрабатывать приложение в жанре платформер. Суть платформера заключается в том, что игрок должен, минуя различные препятствия, дойти до конечной точки. Основной игровой механикой платформеров является способность персонажа прыгать на различные платформы, избегать препятствий и собирать предметы.

Суть игры пройти все уровни с одной попытки. Если главный герой умер на каком-либо из уровней, то игра начинается снова с первого уровня. Игра будет состоять из 4 уровней, каждый из которых будет более сложным, чем предыдущий. В игре будет возможность собирать фрукты, которые будут приносить игроку очки. На экране будет отображаться таймер с текущим временем, чтобы игрок мог ставить рекорды по прохождению уровней.

В задачи игрока входит перемещение в горизонтальных направлениях при помощи джойстика, а также прыжки при помощи кнопки.

1.3.3 Структура уровней

Структура уровней включает в себя разнообразные платформы и ловушки, создающие вызовы для игрока.

Платформы - это поверхности, на которых персонаж может перемещаться, стоять и прыгать. Они могут иметь различные формы, размеры и свойства, предоставляя игроку возможность передвигаться по уровню. Рассмотрим встречающиеся типы платформ:

- горизонтальные платформы. Платформы, расположенные на одном уровне горизонтально. Игрок может перемещаться по ним вправо или влево, стоять или прыгать с них;

- вертикальные платформы. Платформы, которые движутся вверх или вниз. Игрок может использовать их для перемещения по вертикальной оси, прыгая на них или спускаясь с них. Вертикальные платформы помогают преодолеть высотные различия на уровне;

- подвижные платформы. Платформы, которые движутся по заданному пути. Они могут перемещаться горизонтально, вертикально или в обоих направлениях. Подвижные платформы требуют от игрока точности и времени, чтобы перейти с одной платформы на другую.

Далее рассмотрим ловушки. Ловушки - это интерактивные элементы, предназначенные для создания вызовов и препятствий. Они увеличивают сложность игры и проверяют навыки игрока. В игре встречаются следующие виды ловушек:

- шипы. Острые объекты, расположенные на земле, которые причиняют урон персонажу при контакте. Чтобы избежать поражения, игрок должен быть внимателен и точно прыгать;

- пилы. Острые вращающиеся объекты, представляющие опасность для персонажа. Игрок должен аккуратно избегать контакта с ними, чтобы не потерпеть неудачу;

- враги. Противники, такие как летающие головы с шипами, двигаются по заданному пути. Если персонаж касается врага, он терпит неудачу. Игрок должен быть внимателен и выбрать правильный момент для перемещения.

Каждый уровень будет содержать различные комбинации платформ и ловушек, а их количество и сложность будут увеличиваться по мере прохождения игры. Кроме того, на уровнях будут размещены фрукты, которые игрок может собирать для получения очков. С каждым последующим уровнем количество и виды фруктов будут меняться. Фрукты

будут расположены как по пути к финишу, так и в некоторых ответвлениях от дороги, что будет заставлять игрока задержаться на уровне.

Таким образом от уровня к уровню игрок будет знакомится с различными сложностями и нарабатывать свои навыки в их преодолении.

1.3.4 Графический интерфейс

Согласно требованиям проекта, в игре должен быть реализован интуитивно понятный интерфейс, включающий следующие элементы: главное меню, элементы управления, счетчики очков и времени.

Главное меню будет содержать две кнопки. Первая кнопка предназначена для начала игры, вторая кнопка – для выхода из приложения.

Во время игры на экране будут реализованы 2 счетчика: счетчик времени прохождения уровней и счетчик очков. Также будет присутствовать рекорд лучшего времени прохождения игры и кнопка выхода в главное меню.

Элементы управления включают в себя джойстик для движения героя по горизонтали и кнопку прыжка.

Все интерфейсные элементы будут разработаны с учетом простоты и интуитивности, чтобы игрок мог быстро освоить управление и пользоваться функциональностью игры без лишних сложностей.

1.4 Реализация

1.4.1 Вводная информация

MonoBehaviour - это базовый класс в *Unity*, который предоставляет функциональность для создания компонентов игровых объектов. Он является основным строительным блоком для разработки игр в *Unity* и используется для управления поведением и состоянием игровых объектов.

Класс *MonoBehaviour* наследуется от класса *UnityEngine.Object* и является основным компонентом, который может быть добавлен к игровому объекту [3]. Каждый *MonoBehaviour* может содержать код, который

определяет его поведение, реагирует на события и выполняет действия в игре.

MonoBehaviour предоставляет ряд методов, которые можно переопределить в пользовательском коде. Некоторые из наиболее часто используемых методов *MonoBehaviour* включают:

- *Start()*. Вызывается единожды при запуске сцены и используется для инициализации компонента;
- *Update()*. Вызывается каждый кадр и используется для обновления состояния компонента;
- *FixedUpdate()*. Вызывается с постоянной фиксированной частотой, независимо от кадровой частоты, и используется для физических вычислений и обновления компонента.

Также на всех объектах, с которыми сталкивается персонаж, используется компонент *Collider2D*. *Collider2D* - это компонент в *Unity*, который используется для определения границ и обнаружения столкновений объектов в двумерной среде. *Collider2D* определяет форму и размеры объекта, которые будут использоваться для взаимодействия с другими коллайдерами в игровом мире.

Когда компонент *Collider2D* прикрепляется к игровому объекту, он создает область, которую другие объекты могут засечь или пересечь. *Collider2D* представляет различные формы, такие как прямоугольник, круг, полигон или составные формы, и обеспечивает точное обнаружение столкновений между объектами.

Collider2D также может использоваться совместно с компонентом *Rigidbody2D*, чтобы обеспечить реалистичное поведение при столкновении объектов. Когда коллайдеры объектов пересекаются или сталкиваются, *Unity* отправляет сообщения и вызывает методы обработки столкновений, которые могут быть использованы для реагирования на столкновения и взаимодействия с другими объектами.

1.4.2 Базовая логика и основные компоненты

Мы разрабатываем 2D игру, где главный герой представлен в виде спрайта. Главный герой имеет несколько навешенных компонентов, включая *BoxCollider2D* и *Rigidbody2D*, а также два скрипта - *MoveController* и *Health*. Эти компоненты и скрипты играют важную роль в функциональности главного героя, обеспечивая его перемещение, взаимодействие с окружающим миром и управление его здоровьем.

Основные характеристики персонажа реализованы в скрипте *MoveController*. В методе *FixedUpdate* реализовано передвижение персонажа. Метод получает горизонтальный ввод от компонента *Joystick* (*joystick.Horizontal*) и устанавливает горизонтальную скорость главного героя (*rb.velocity.x*). Если главный герой не смотрит вправо и горизонтальный ввод больше 0, вызывается метод *Flip()*, чтобы изменить направление взгляда. Если главный герой смотрит вправо и горизонтальный ввод меньше 0, также вызывается метод *Flip()*.

Метод *Flip()* изменяет направление взгляда главного героя, поворачивая его вокруг оси Y. Он меняет значение переменной *facingRight* на противоположное, а затем масштабирует локальную шкалу объекта по оси X, чтобы повернуть его в нужную сторону. Метод *jumpButton* вызывается при нажатии кнопки прыжка. Если герой на земле, его скорость изменяется на вертикальную силу прыжка.

Метод *Update()* проверяет, находится ли главный герой на земле путем использования метода *Physics2D.OverlapCircle()*.

В данном скрипте двойной прыжок реализован с помощью переменных *extraJump* и *extraJumpvalue*.

Переменная *extraJump* отвечает за количество доступных дополнительных прыжков, а переменная *extraJumpvalue* задает исходное значение этой переменной.

В методе *jumpButton()* проверяется состояние *isGrounded* (находится ли персонаж на земле). Если персонаж находится на земле (*isGrounded == true*), то ему разрешается выполнить прыжок, устанавливая скорость *rb.velocity* по вертикали равной *jumpForce* (силе прыжка) вверх.

Если персонаж не находится на земле (*isGrounded == false*) и у него остались дополнительные прыжки (*extraJump > 0*), то он может совершить дополнительный прыжок. В этом случае также устанавливается скорость *rb.velocity* по вертикали равной *jumpForce* и значение переменной *extraJump* уменьшается на единицу (*extraJump--*).

Если у персонажа не осталось дополнительных прыжков (*extraJump == 0*) и он не находится на земле, то ничего не происходит.

Таким образом, при нажатии на кнопку прыжка (*jumpButton()*), персонаж может выполнить обычный прыжок, а также использовать ограниченное количество дополнительных прыжков в воздухе. Это реализует механику двойного прыжка.

В скрипте *Health* реализован метод *TakeHit*. Данный метод уничтожает персонажа, сбрасывает счетчик очков и затем загружает главную меню.

Все поверхности на уровнях в игре построены при помощи *Tilemap*. Наш *Tilemap* использует компоненты *Tilemap Collider2D*, который добавляет коллайдер каждой отдельной клеточке с тайлом, и *Composite Collider2D*, который объединяет все тайлы в один нераздельный коллайдер. На рисунке 1.6 приведен пример создания тайлов.

Переход на следующий уровень происходит при соприкосновении с объектом *Checkpoint*. Функционал этого объекта реализован в скрипте *Finish*. Когда персонаж касается объекта *Checkpoint* вызывается метод *NextLevel*, который загружает следующую сцену. Объект представлен на рисунке 1.7 ниже.

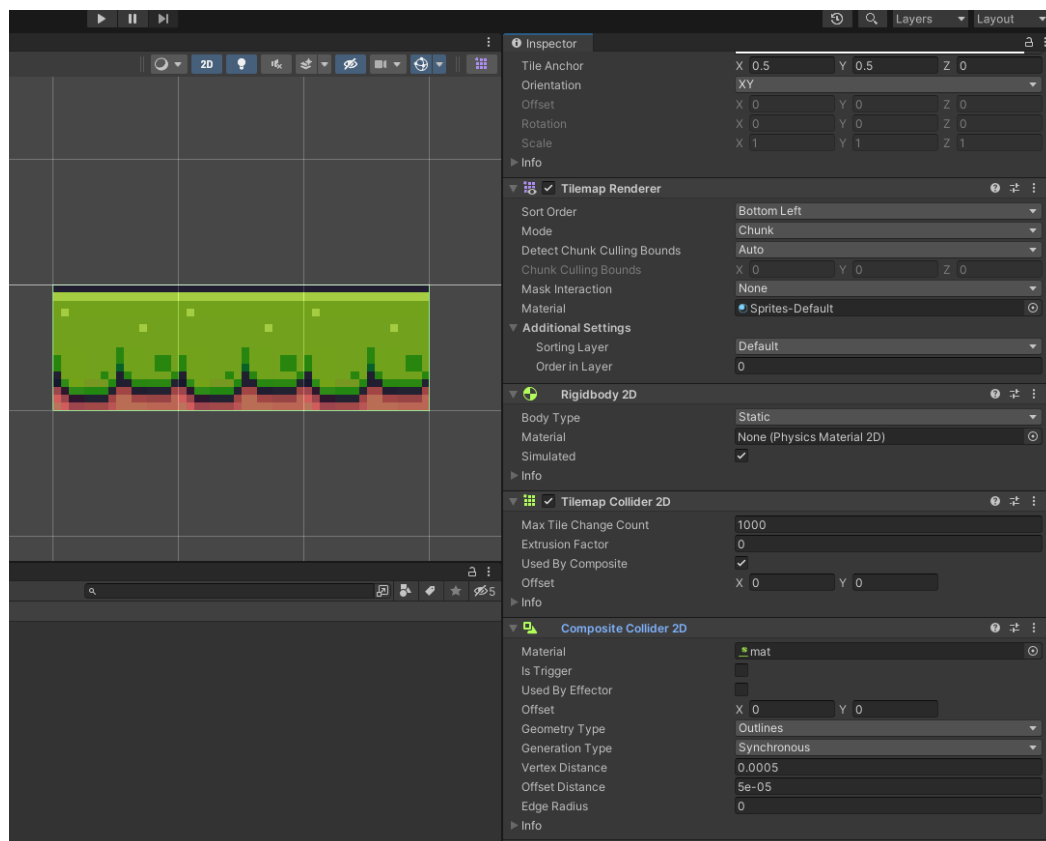


Рисунок 1.6 –Создание тайлов



Рисунок 1.7 – Объект Checkpoint

На всех четырех уровнях, реализованных в игре, присутствуют объекты в виде фруктов, которые при соприкосновении с героем исчезают и дают игроку некоторое количество очков. Все свойства этих объектов реализованы в скрипте *Point*. При соприкосновении с героем вызывается метод *OnTriggerEnter2D(Collider2D collision)*, который уничтожает фрукт и прибавляет к счетчику очков баллы. А также в скрипте *MoveController* вызывается звук уничтожения фрукта при помощи кода представленного ниже.

```
private void OnTriggerEnter2D(Collider2D collision)
{
    if (collision.CompareTag("point"))
    {
        AudioSource.PlayOneShot(audioclip);
    }
}
```

Также на каждой сцене есть музыкальное сопровождение. Оно реализовано при помощи компонента *AudioSource*. На рисунке 1.8 приведен пример настроек компонента.

В ходе реализации игрового приложения были разработаны диаграмма использования игры и диаграмма деятельности. Диаграмма использования игры представлена на рисунке 1.9. Диаграмма деятельности представлена на рисунке 1.10

.

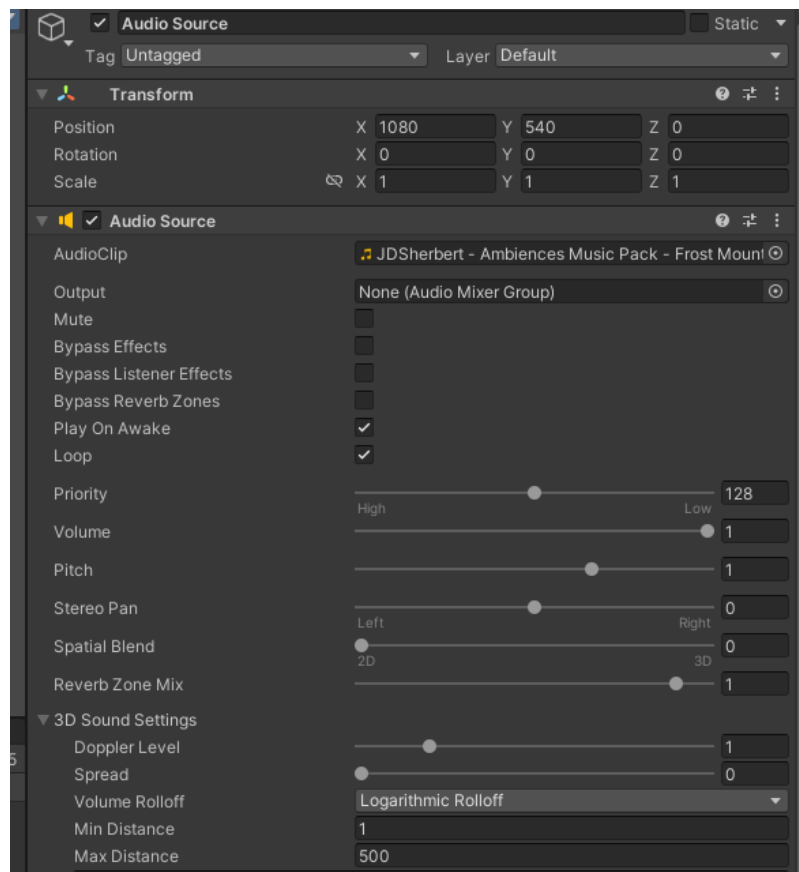


Рисунок 1.8 – Настройки компонента *AudioSource*

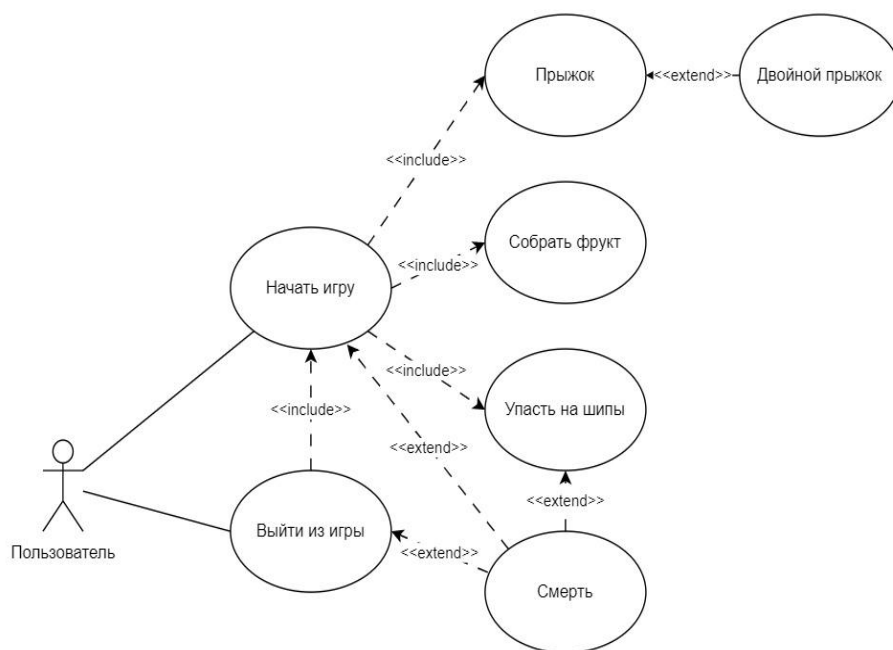


Рисунок 1.9 – Диаграмма использования приложения

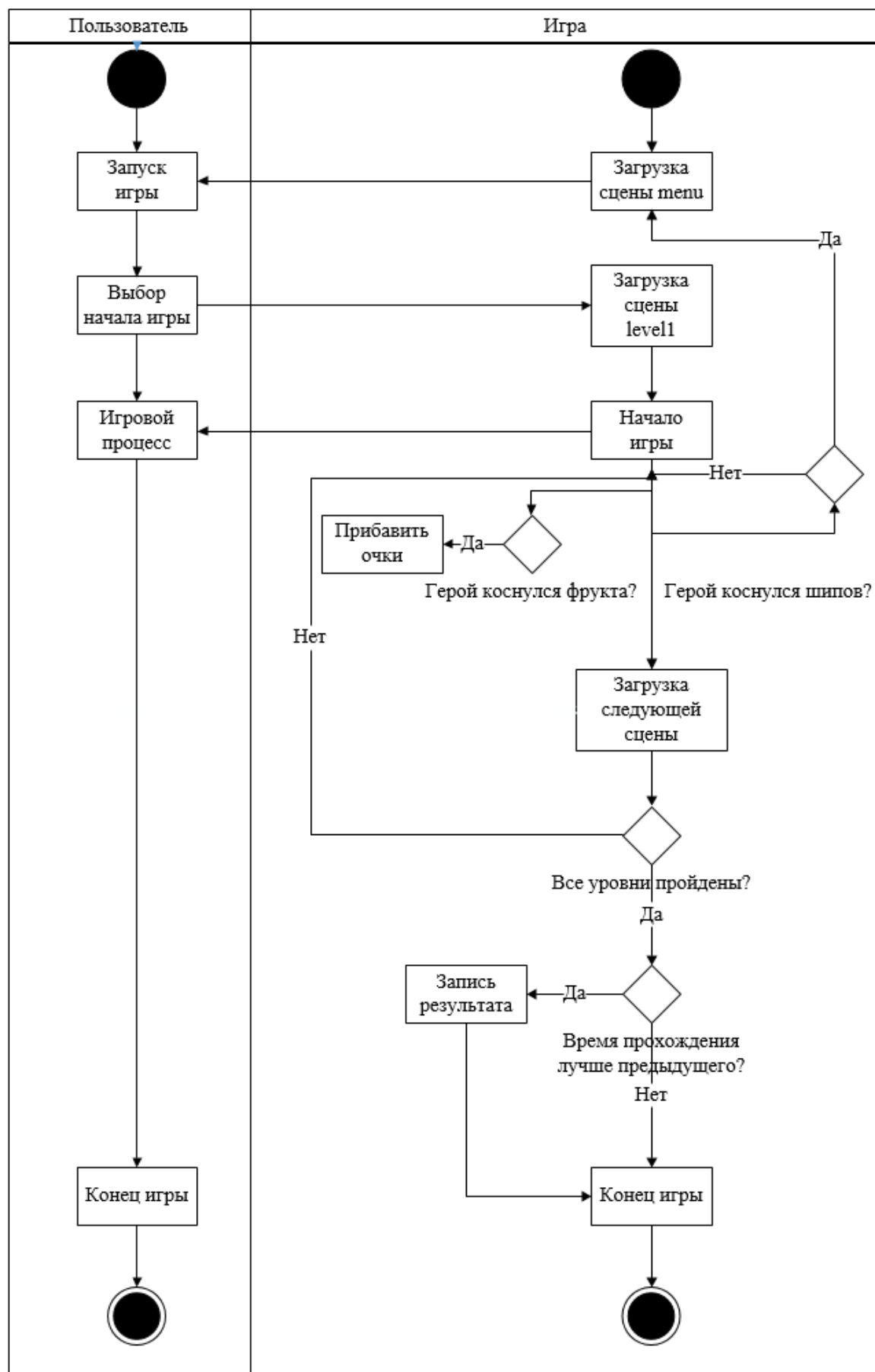


Рисунок 1.10 – Диаграмма деятельности

1.4.3 Реализация анимаций

В *Unity* существует несколько видов анимаций, которые могут быть использованы для создания живых и динамичных игровых объектов [4]. Давайте рассмотрим некоторые из них:

- спрайтовые анимации. Спрайтовые анимации являются одним из самых распространенных типов анимаций в *Unity*. Они основаны на последовательном воспроизведении спрайтов, создавая иллюзию движения. В *Unity* можно создавать спрайтовые анимации с помощью спрайтовых атласов или анимационных клипов. Спрайты могут меняться с определенной скоростью или в зависимости от действий игрока или других событий;

- риггинг и скелетная анимация. Риггинг и скелетная анимация позволяют создавать более сложные и гибкие анимации для персонажей и объектов. Они основаны на использовании скелетных систем, состоящих из костей (бонов) и связанных с ними спрайтов или моделей. Риггинг позволяет задавать различные кости и ограничения движения, а скелетная анимация позволяет анимировать персонажей, деформировать их форму и создавать реалистичные движения.

В данной игре были использованы спрайтовые анимации. Все использованные спрайты находятся в свободном доступе. Пример набора спрайтов представлен на рисунке 1.11 ниже.



Рисунок 1.11 – Пример набора спрайтов

Для создания анимации используется инструмент *Animation*. В этом инструменте можно настроить время через которое будет отображаться следующий спрайт в анимации. Данный инструмент показан на рисунке 1.12.

После создания всех анимаций нужно настроить переходы между ними в окне *Animator*. Переходы определяют условия, при которых происходит переход от одного состояния к другому. Например, переход "idle" в состояние "run" может быть активирован при условии того, что персонаж касается земли и движется по горизонтали.

У главного героя в данной игре есть 5 состояний: *idle*(спокойствие), *run*(бег), *jump*(прыжок), *double jump*(двойной прыжок) и *fall*(падение). В параметрах окна *Animator* создано 2 переменные: "IsRunning" типа *Bool* и "fall" типа *Integer*. В зависимости от значений этих переменных меняются анимации. На рисунке 1.13 представлен *Animator* для персонажа.

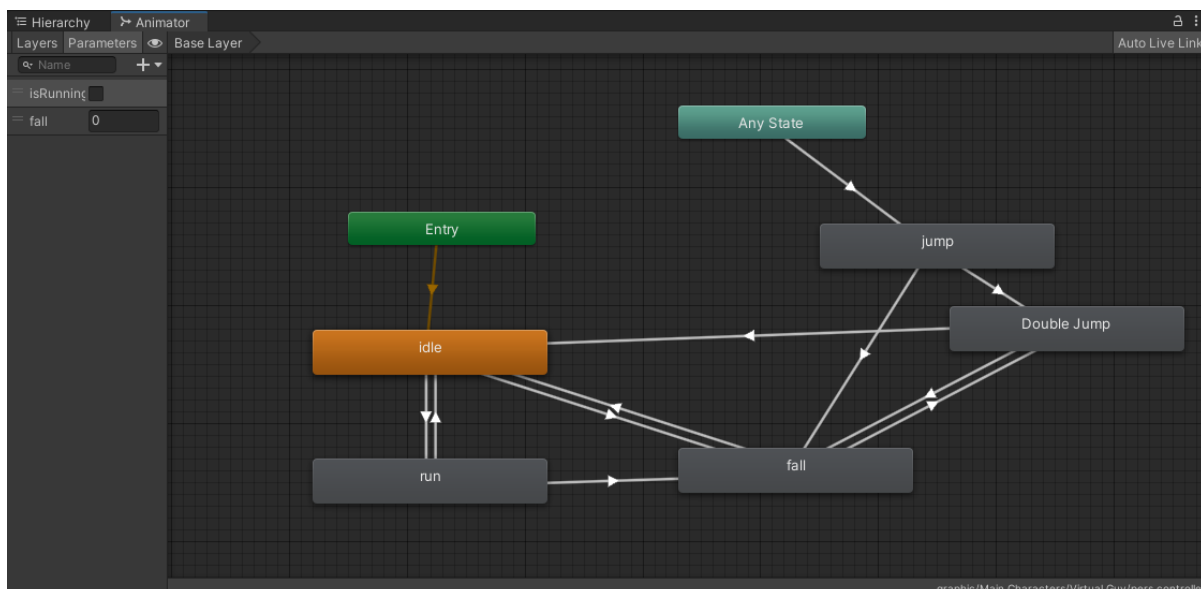


Рисунок 1.12 – Окно *Animator*

1.4.4 Реализация ловушек и препятствий

В игре встречаются несколько видов ловушек. Первые из них это шипы, которые реализованы объектом *Spikes*. К шипам привязаны 2 компонента - это *BoxCollider2D* и скрипт *Collision Damage*. Данный скрипт, при соприкосновении шипов с игроком, вызывает метод *TakeHit* из скрипта *Health*, который был описан ранее. Пример шипов представлен на рисунке 1.14.

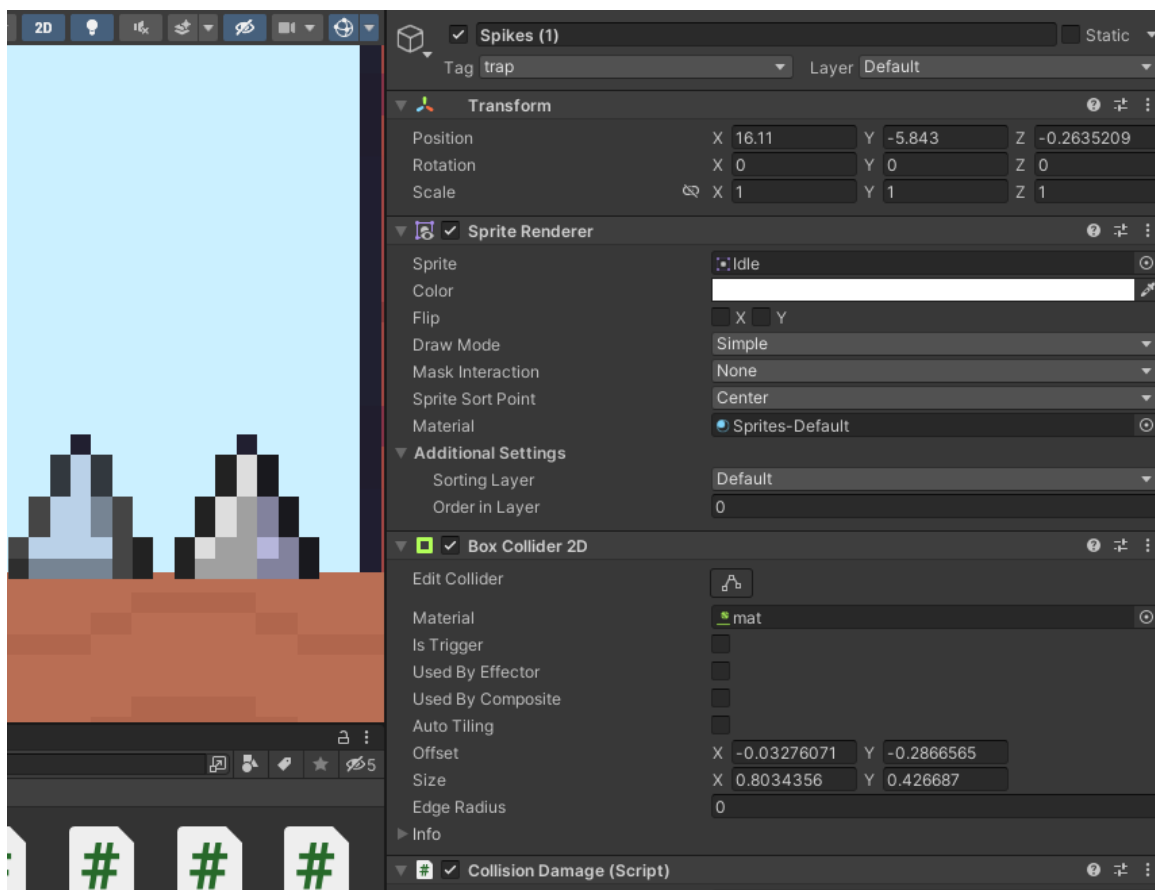


Рисунок 1.13 - Шипы

В данной игре на третьем и четвертом уровне встречается препятствие в виде летающей платформы. Платформа движется от одного заранее заданного объекта до другого. Поведение платформы прописано в скрипте *platform*. В методе *Update* происходит движение платформы и проверка достижения целевой позиции. Метод *OnCollisionEnter2D(Collision2D collision)* вызывается, когда происходит столкновение платформы с персонажем. Этот метод делает платформу родительским объектом персонажа. Это делается для того, чтобы персонаж двигался вместе с платформой.

Следующей ловушкой является летающая голова с шипами, называемая *Spike Head*. Данный объект использует скрипт *platform* и *Collision Damage*, которые были описаны ранее. Также этот объект использует компонент *Audio Source*, с помощью которого реализован 3D звук. При

приближении персонажа к *Spike Head* мы можем, услышать издаваемый им, звук, который можно услышать только в определенном радиусе. Пример настройки 3D звука представлен на рисунке 1.15.

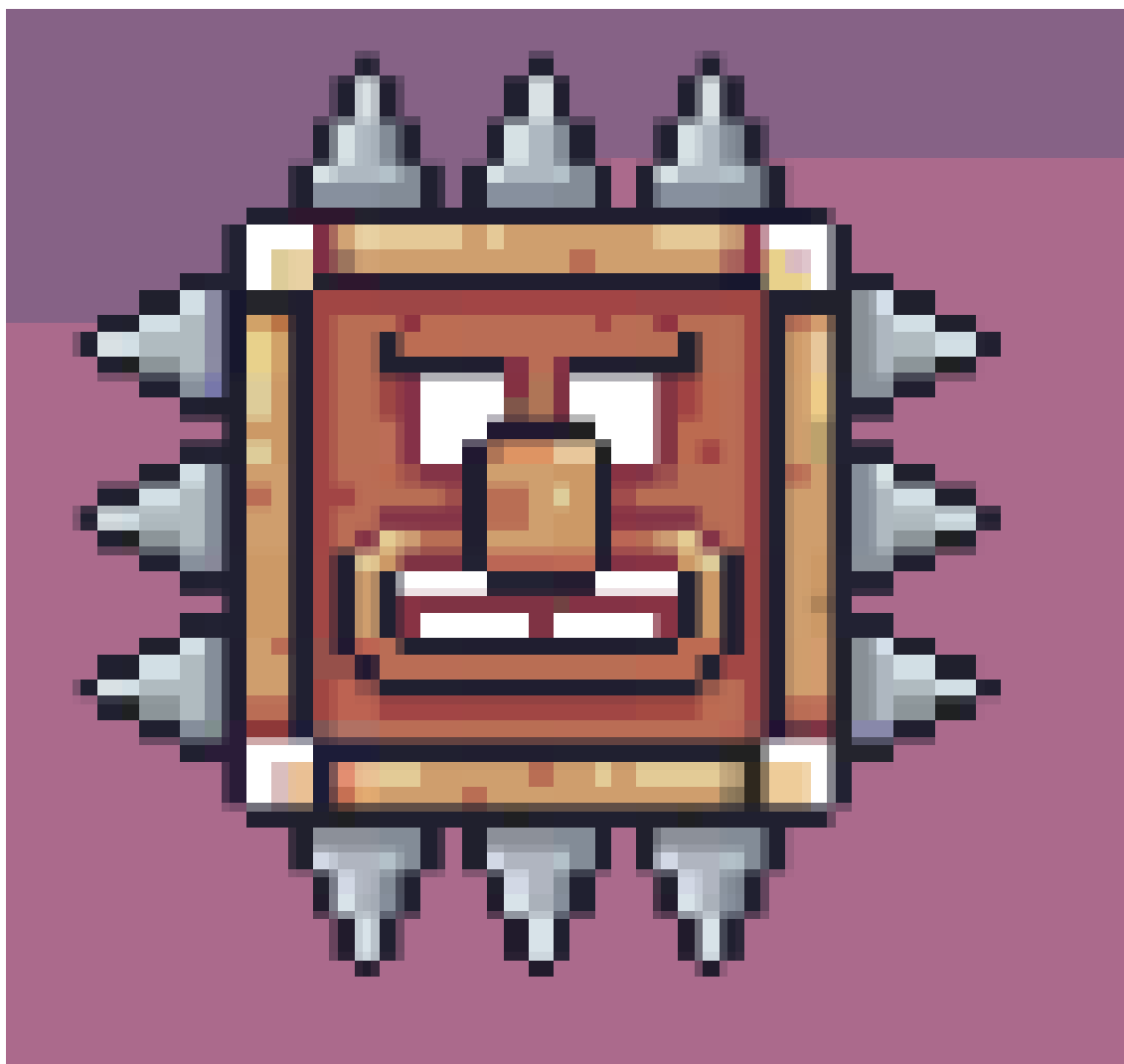


Рисунок 1.14 – *Spike Head*

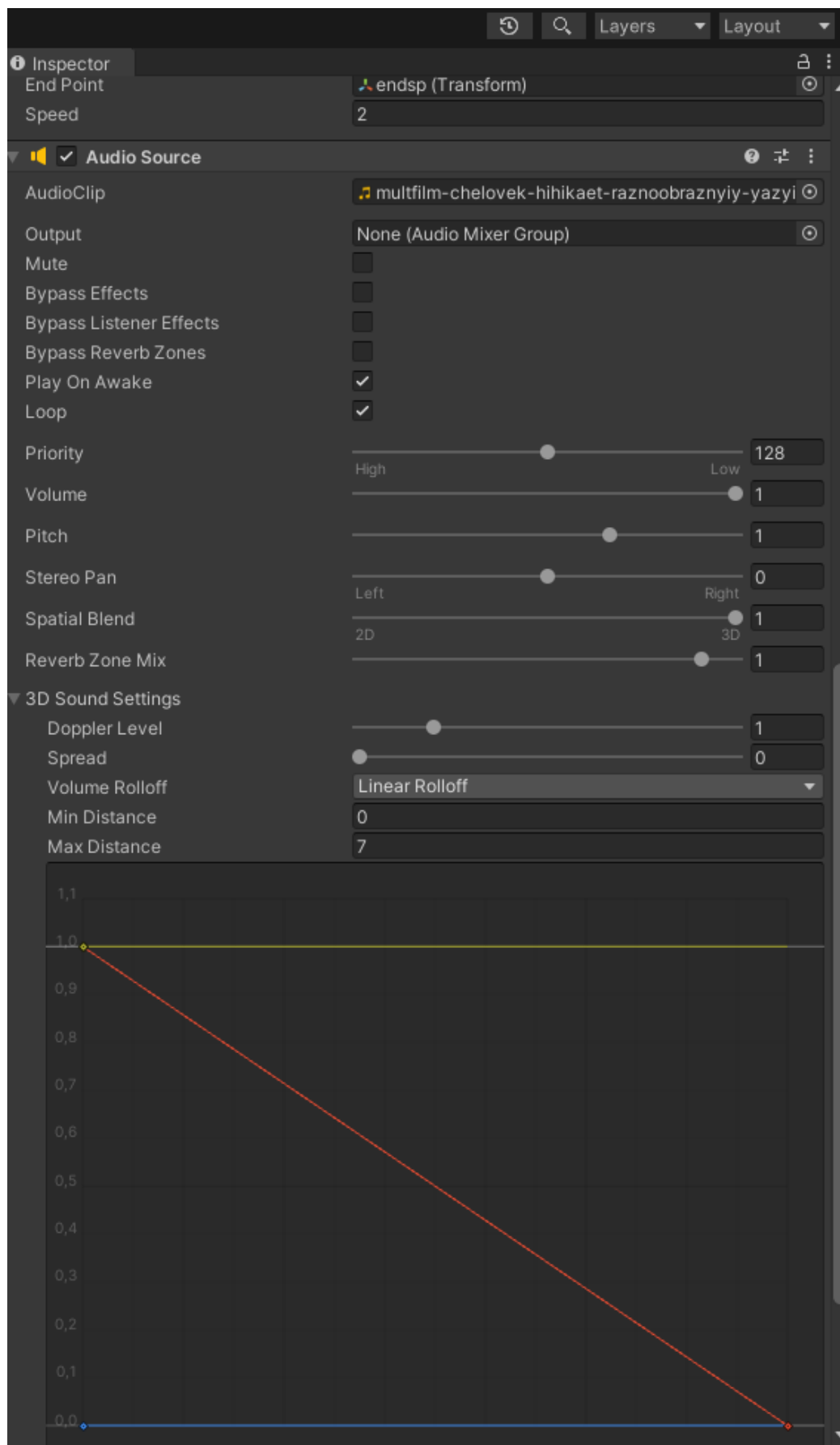


Рисунок 1.15 – Настройка 3D звука

1.4.5 Графический интерфейс

Графический интерфейс игры включает в себя элементы управления и различные счетчики. В главном меню присутствуют две основные кнопки: кнопка запуска игры и кнопка выхода из игры. Они позволяют игроку начать игру или закрыть приложение, с помощью скрипта *play*. При нажатии на кнопку начала игры вызывается метод *level1*, который загружает первый уровень. При нажатии на кнопку выхода, вызывается метод *ExitGame*, который закрывает приложение. Интерфейс главного меню показан на рисунке 1.16 ниже.



Рисунок 1.16 – Интерфейс главного меню

Элементы управления, связанные с персонажем, включают в себя джойстик для движения по горизонтали и кнопку прыжка. Джойстик позволяет игроку контролировать горизонтальное перемещение персонажа, перемещая его влево или вправо. Кнопка прыжка позволяет персонажу выполнять прыжок для перехода через препятствия или достижения высоких платформ. Элементы управления показаны на рисунке 1.17 ниже.



Рисунок 1.17 – Элементы управления

Также на каждом уровне реализована кнопка выхода в главное меню. При нажатии на эту кнопку загружается главное меню и сбрасывается счетчик очков.

В игре также присутствуют счетчик времени, счетчик очков и время рекорда, которые предоставляют игроку дополнительную информацию о процессе игры и их достижениях.

Счетчик времени отображает текущее время, прошедшее с начала игры. Это позволяет игроку контролировать свою производительность и стараться улучшить свои результаты. Данный счетчик реализован в скрипте *Timer*, при помощи корутины. Корутина (*Coroutine*) - это специальный тип функции в языке программирования, который позволяет выполнять определенные задачи в асинхронном режиме.

Время рекорда отображает лучшее время прохождения игры. Оно дает игроку ориентир и стимулирует его попытаться превзойти этот рекорд и установить новый лучший результат. Это создает элемент соревнования и мотивации для игрока, чтобы он стремился к улучшению своих навыков и достижений. Этот элемент интерфейса реализован при помощи метода *CompleteLevel* в скрипте *Timer*. Когда персонаж касается финиша на последнем уровне вызывается метод *CompleteLevel*. Данный метод работает поэтапно:

- проверка наличия запущенной корутины *timerCoroutine*. Это условие проверяет, активна ли корутина, отвечающая за отсчет времени уровня. Если переменная *timerCoroutine* не равна *null*, значит корутина запущена;

- остановка корутины и сброс переменной *timerCoroutine*. С помощью метода *StopCoroutine()* корутина прерывается и перестает выполняться. После этого переменная *timerCoroutine* устанавливается в значение *null*, чтобы показать, что корутина больше не активна;

- проверка текущего времени и обновление рекорда. В этой части кода происходит сравнение текущего времени *currentTime* с лучшим временем *bestTime* или проверка, является ли *bestTime* равным 0. Если текущее время меньше, чем лучшее время или *bestTime* равно 0, выполняются следующие действия:

1) обновление лучшего времени *bestTime* значением текущего времени *currentTime*;

2) обновление текста рекорда на экране с помощью компонента *recordText*. В тексте отображается значение лучшего времени, округленное до двух знаков после запятой;

3) сохранение лучшего времени в *PlayerPrefs*: *PlayerPrefs* - это класс в Unity, который позволяет сохранять и получать данные между сеансами игры. В данном случае, лучшее время *bestTime* сохраняется с помощью метода *SetFloat()* и ключа "*BestTime*". Это позволяет сохранить рекордное время для будущих игровых сеансов.

1.5. Тестирование

1.5.1 Результаты тестирования

Тестирование игрового проекта является неотъемлемой частью процесса разработки игр. Оно позволяет выявить и исправить ошибки, проверить функциональность и стабильность игры, а также убедиться в соответствии игрового процесса заявленным требованиям.

					ПГУ1.090301.02.001 ПЗ	Лист
						40
Изм.	Лист	№ докум.	Подпись	Дата		

На данном этапе реализации проекта нецелесообразно проводить обширное тестирование, поскольку в игре реализована только небольшая часть функциональностей, которые будут доработаны в последующей разработке. Поэтому проведенное тестирование (Таблица X) является лишь небольшой проверкой на соответствие основным функциональным требованиям.

В рамках данного проекта игровое приложение успешно прошло тестирование (Таблица 2) на различных мобильных устройствах, включая *Realme 10*, *Redmi 9*, *Honor X* и *Me Pad 5*.

Таблица 2 – Тест - план

Назначение	Действие	Ожидаемый результат	Полученный результат	Итог
Проверка запуска на разных смартфонах	Запустить игру на разных смартфонах	Корректный запуск и после отображение главного меню	Игра запустилась на всех тестируемых смартфонах	Пройден
Проверка корректного отображения интерфейса на разных смартфонах	Просмотреть главное меню и первый уровень на разных смартфонах	Корректное отображение интерфейса	Интерфейс отображается корректно	Пройден

Назначение	Действие	Ожидаемый результат	Полученный результат	Итог
Проверка корректного счета очков	Собрать несколько фруктов	Значение счетчика “Score” будет увеличиваться	Значение счетчика увеличилось	Пройден
Проверить корректность счетчика времени	Некоторое время побегать по уровню	Корректное отображение времени проведенного на уровне	Корректное отображение времени	Пройден
Проверка исправности отображения рекорда	Пройти игру	Время прохождения игры будет отображаться корректно	Время отображается корректно	Пройден
Проверка сохранения рекорда после перезапуска приложения	Пройти игру, выйти из нее и снова запустить	После перезапуска рекорд сохранится	Рекорд сохранился	Пройден
Проверка работы перехода между уровнями	Пройти уровень	В конце уровня героя переносит на следующий уровень	Персонажа перенесло на следующий уровень	Пройден

Назначение	Действие	Ожидаемый результат	Полученный результат	Итог
Корректность завершения игры после победы	Пройти игру	После прохождения всех уровней появляется победная сцена	Появилась победная сцена	Пройден
Проверка корректности завершения игры во время прохождения	На уровне нажать кнопку выхода в меню	При нажатии кнопки откроется главное меню	После нажатия открылось главное меню	Пройден
Проверка корректного взаимодействия с объектами	Собрать несколько фруктов,	При соприкосновении с фруктами, они должны исчезнуть;	Фрукты исчезли	Пройден
Проверка работоспособности смерти героя	Упасть на шипы	При соприкосновении героя с шипами, должно появиться главное меню	После падения на шипы появилось главное меню	Пройден

На данном этапе разработки проекта тестирование демонстрирует, что игра удовлетворяет большинству предоставленных функциональных требований, которые были определены на начальном этапе проекта.

1.5.2 Результаты тестирования

После запуска приложения нас встречает главное меню. При нажатии на кнопку в центре экрана происходит запуск игры. Приложение может быть закрыто при нажатии на красную кнопку, расположенную в левом нижнем углу экрана. Главное меню представлено на рисунке 1.18.



Рисунок 1.18 – Главное меню

Если выбрать кнопку запуска, то происходит загрузка первого уровня. Первый уровень представляет из себя ознакомительную локацию, которая не почти не представляет сложности для прохождения. Игрок учится преодолевать препятствия, прыгать, подбирать фрукты. На первой локации игрок знакомится с шипами, встречает первый фрукт – это яблоко, которое дает 1 балл при соприкосновении. Первый уровень представлен на рисунке 1.19.

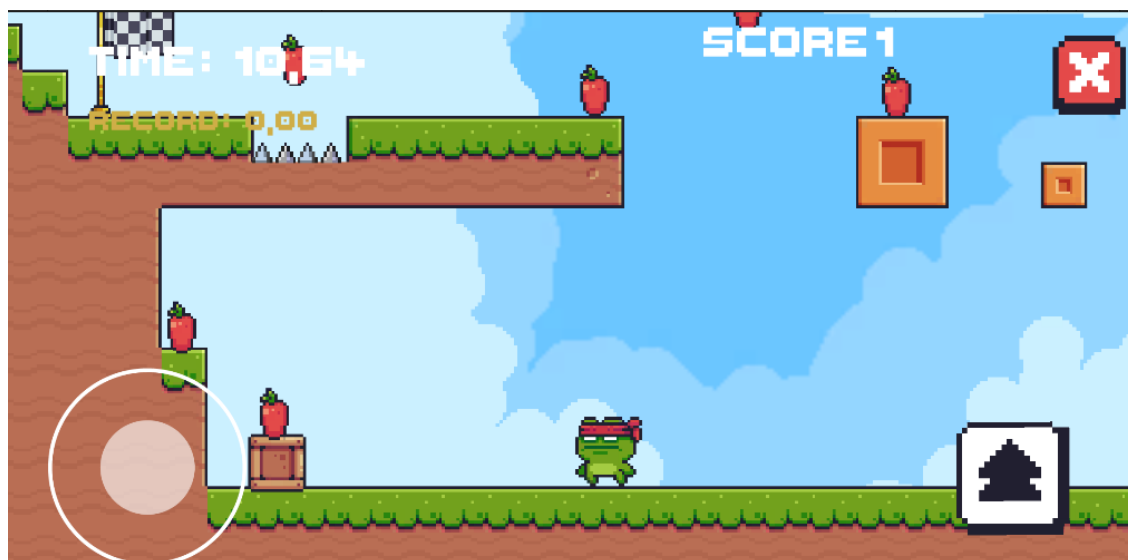


Рисунок 1.19 – Первый уровень

Второй уровень - это локация из парящих платформ. Этот уровень представляет собой полосу, которая проверяет точность в управлении персонажем, если игрок ошибётся то упадет на шипы, и персонаж умрет. В данной локации появляется новый вид фруктов – арбуз. Арбуз приносит игроку 2 очка. Второй уровень изображен на рисунке 1.20.

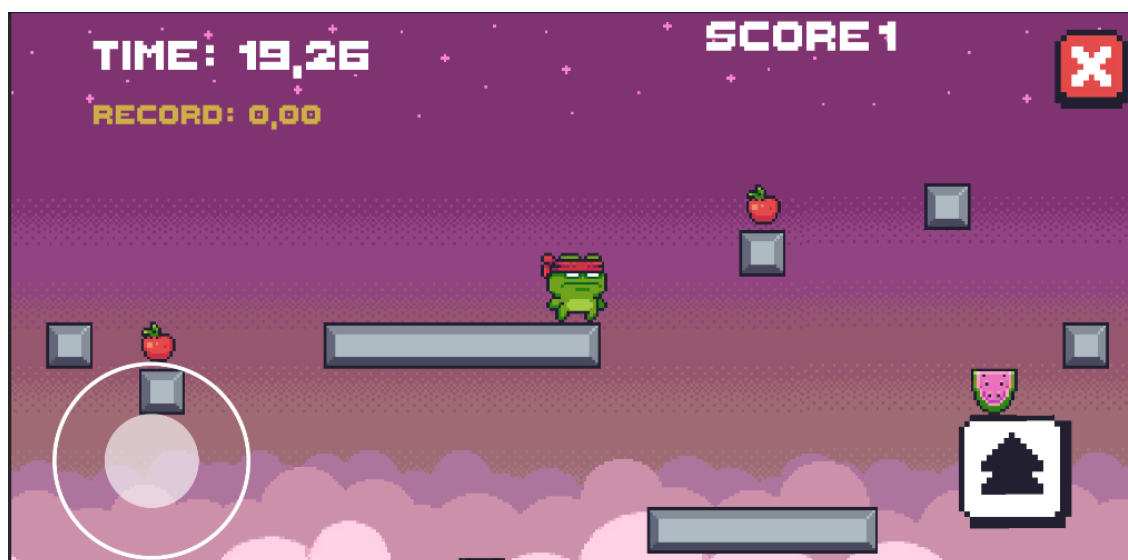


Рисунок 1.20 – Второй уровень

На третьем уровне игроку нужно подниматься вверх локации по парящим платформам. Здесь игрок знакомится с двигающимися

платформами, летающими головами с шипами, которые убивают героя, и появляются вишни, которые приносят 3 очка. На рисунке 1.21 изображен третий уровень.

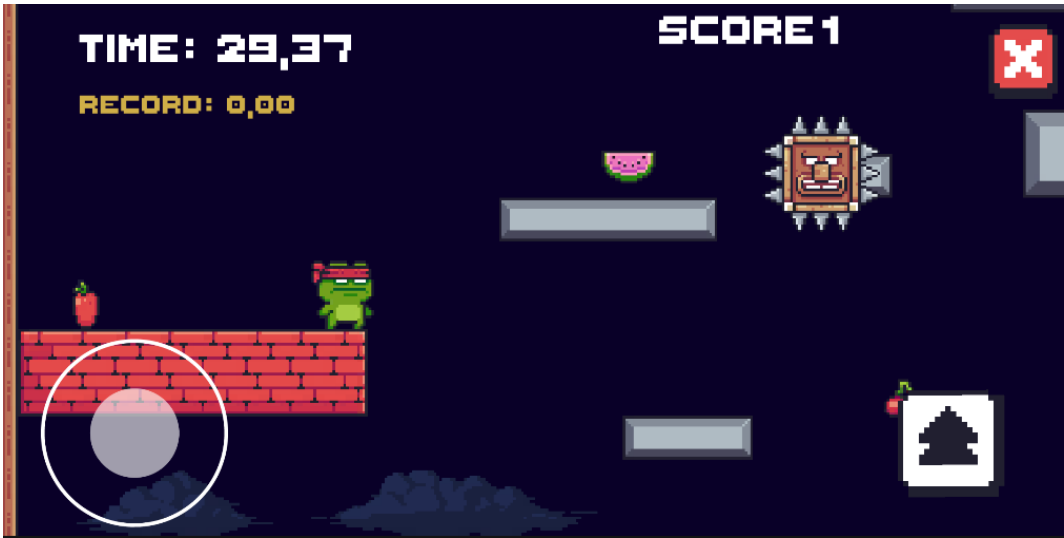


Рисунок 1.21 – Третий уровень

Четвертый уровень совмещает в себе все предыдущие уровни. Здесь встречается полоса парящих платформ, подъем вверх, движущиеся платформы и летающие головы. Также на уровне появляется новый фрукт – банан, который приносит 4 очка. Четвертый уровень изображен на рисунке 1.22.

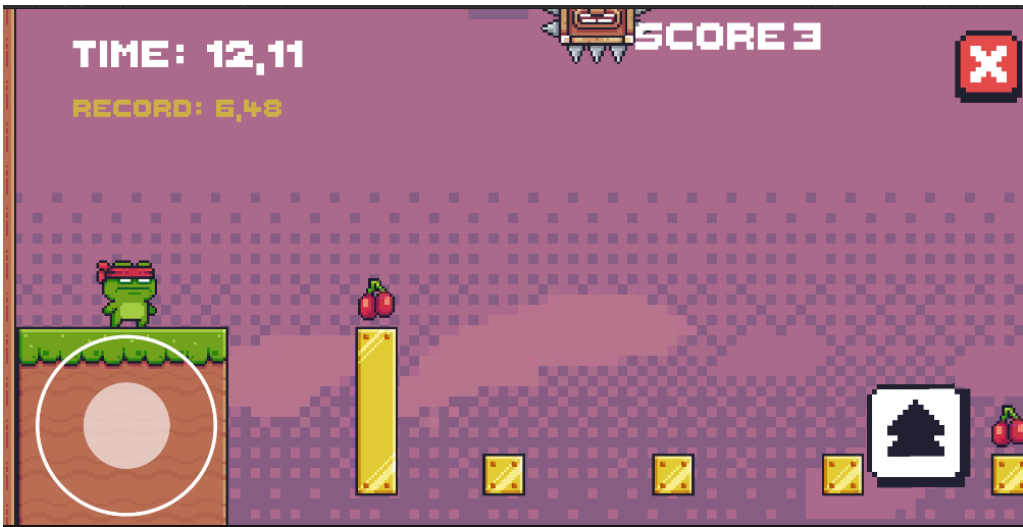


Рисунок 1.22 – Четвертый уровень

После прохождения игроком четвертого уровня появляется победная сцена. На этой сцене сохраняется рекорд времени прохождения. И игрок вознаграждается танцем героя игры. Победная сцена представлена на рисунке 1.23.

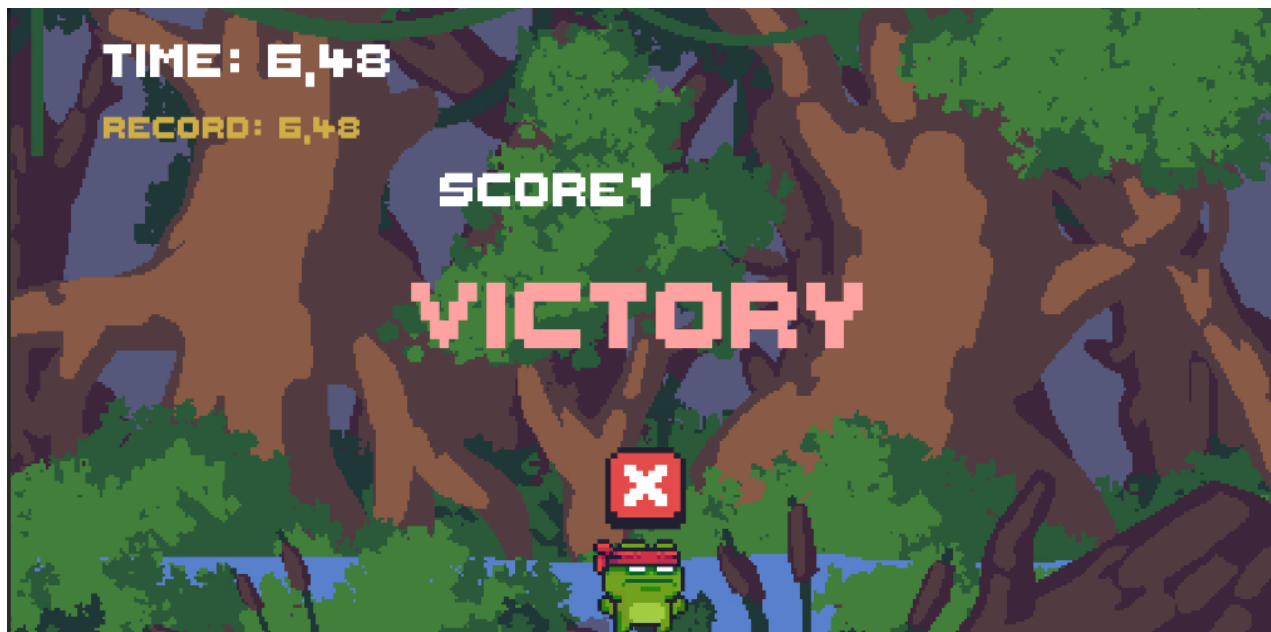


Рисунок 1.23 – Победная сцена

1.5.3 Вывод по результатам тестирования

Стоит понимать, что данный проект не является полноценной законченной игрой, и по своей сути представляет собой прототип. Вызвано это тем, что для создания полноценного продукта требуется огромное количество времени. К примеру, разработка даже небольшой полноценной мобильной игры может занимать более года.

В настоящий момент невозможно точно определить будущие системные требования для проекта, поскольку многие элементы еще не реализованы. На данный момент мы имеем только базовые игровые механики, и в будущем возможны значительные изменения в технических характеристиках игры.

Однако, на текущем этапе разработки можно сказать, что для запуска данного прототипа потребуется смартфон, соответствующий минимальным требованиям, рекомендуемым для работы игрового движка *Unity*. Это позволяет предположить, что прототип будет работать на большинстве смартфонов, удовлетворяющих указанным минимальным требованиям. Однако, следует учитывать, что с развитием проекта и добавлением новых функциональных возможностей системные требования могут измениться.

					ПГУ1.090301.02.001 ПЗ	Лист
						48
Изм.	Лист	№ докум.	Подпись	Дата		

2 Охрана труда

2.1 Требования к организации рабочего места с ПК

При расстановке рабочих мест в помещении необходимо соблюдать определенные требования. Согласно СанПиН 2.2.2/2.4.1340-03, между столами должно быть расстояние не менее 2 метров, а между мониторами - не менее 1 метра. Площадь рабочего места пользователя должна быть не менее 4,5 квадратных метров.

Если продолжительность работы не превышает четырех часов в день, минимальная площадь одного рабочего места для пользователя старше 18 лет составляет 5 квадратных метров.

Рекомендуется изолировать рабочие места с персональным компьютером для работы, требующей умственного напряжения или высокой концентрации внимания, друг от друга перегородками высотой 1,5-2,0 метра.

Расстояние между монитором и глазами пользователя должно быть 60-70 сантиметров, но не менее 50 сантиметров с учетом размеров знаков и символов. Клавиатура должна располагаться на расстоянии 10-30 сантиметров от края, обращенного к пользователю, или на регулируемой по высоте рабочей поверхности, отделенной от основной столешницы.

Пользовательское стул (кресло) должно обеспечивать правильную позу при работе, чтобы предотвратить усталость и искривление позвоночника в области шеи, плеч и спины. При выборе рабочего стула (кресла) следует учитывать рост пользователя, продолжительность и характер работы с компьютером. Стул (кресло) должно быть подъемно-поворотным, с возможностью регулировки угла наклона спинки и высоты подлокотников. Каждая регулировка должна быть независимой, легко осуществляемой и иметь надежную фиксацию. Поверхности сиденья, спинки и других элементов стула (кресла) должны быть полумягкими, нескользящими, слабо электризующимися и воздухопроницаемыми.

2.2 Требования к освещению помещений и рабочих мест

В компьютерных залах необходимо обеспечить искусственное и естественное освещение. Свет должен падать на рабочее место пользователя с левой стороны из оконного проема. Коэффициент естественного освещения должен быть не менее 1,2% в зонах со снежным покровом и не менее 1,5% на остальной территории.

Искусственное освещение в помещениях с компьютерами должно быть равномерным. Освещенность на поверхности столов должна составлять от 300 до 500 лк. Рекомендуется использовать светильники для местного освещения, которые помогут подсветить документы. При этом местное освещение не должно создавать бликов на поверхности экрана и увеличивать освещенность экрана более 300 лк.

Яркость светящихся поверхностей, находящихся в поле зрения пользователя, не должна превышать 200 кд/м². Яркость бликов на экране монитора не должна превышать 40 кд/м². Для снижения бликов рекомендуется размещать монитор таким образом, чтобы избегать попадания естественного или местного освещения прямо на экран [3].

Для искусственного освещения помещений с компьютерами рекомендуется использовать светильники с зеркализированными решетками. Светильники должны располагаться в виде сплошных или прерывистых линий сбоку от рабочих мест, параллельно линии зрения пользователя в зависимости от размещения компьютеров. Защитный угол светильников должен быть не менее 40 градусов.

2.3 Режим труда и отдыха при работе с компьютером

Режим работы и отдыха при работе на компьютере предусматривает соблюдение определенной длительности непрерывной работы и установленных перерывов, которые зависят от продолжительности рабочей смены и вида трудовой деятельности. Виды работы на компьютере

					ПГУ1.090301.02.001 ПЗ	Лист 50
Изм.	Лист	№ докум.	Подпись	Дата		

классифицируются на три группы: работа по считыванию информации с экрана, работа по вводу информации и творческая работа с высоким умственным напряжением или высокой концентрацией внимания. Если в течение рабочей смены пользователь выполняет разные виды работ, его деятельность относится к той группе работ, на выполнение которой тратится не менее 50% времени рабочей смены.

Количество и длительность регламентированных перерывов, а также их распределение в течение рабочей смены, определяются в зависимости от категории работ на компьютере и продолжительности рабочей смены. При 8-часовой рабочей смене и работе на компьютере регламентированные перерывы следует устанавливать следующим образом:

- первая категория работ: перерыв через 2 часа от начала смены и через 2 часа после обеденного перерыва, каждый длительностью 15 минут;
- вторая и третья категории работ: перерыв через 1,5-2,0 часа от начала рабочей смены и через 1,5-2,0 часа после обеденного перерыва, каждый длительностью 20 минут, или перерыв продолжительностью 15 минут через каждый час работы.

При 12-часовой рабочей смене регламентированные перерывы должны устанавливаться в первые 8 часов работы аналогично перерывам при 8-часовой смене. В течение последних 4 часов работы, независимо от категории и вида работ, каждый час устанавливается перерыв продолжительностью 15 минут.

Продолжительность непрерывной работы на компьютере без регламентированного перерыва не должна превышать 2 часа. При работе на компьютере в ночную смену продолжительность регламентированных перерывов увеличивается на 60 минут, независимо от категории и вида трудовой деятельности.

Кроме регламентированных перерывов, эффективными являются нерегламентированные перерывы длительностью 1-3 минуты. Во время

регламентированных перерывов рекомендуется выполнять комплекс упражнений и гимнастики для глаз, пальцев рук, а также массаж. Комплексы упражнений следует менять через 2-3 недели.

Для пользователей, выполняющих работу с высоким уровнем напряжения, рекомендуется предусматривать психологическую разгрузку во время регламентированных перерывов и в конце рабочего дня в специально оборудованных помещениях.

Во время отдыха рекомендуется заниматься пассивным и активным отдыхом, такими как занятия на тренажерах, плавание, езда на велосипеде, бег, игры в теннис, футбол, лыжи, аэробика, прогулки по парку, лесу, экскурсии, прослушивание музыки и т.д. Рекомендуется проводить курс витаминотерапии в течение месяца дважды в год (весной и поздней осенью). Курение на рабочих местах и в помещениях с компьютерами должно быть категорически запрещено.

					ПГУ1.090301.02.001 ПЗ	Лист
						52
Изм.	Лист	№ докум.	Подпись	Дата		

Заключение

В настоящее время люди все чаще предпочитают использовать смартфоны в качестве развлекательной платформы. В результате этого роста пользователей, игровая аудитория также увеличивается, и разработчики игр всё больше начинают создавать свои приложения на Android.

В ходе работы над проектом были изучены новые методы программирования и проектирования систем, а также значительно улучшены навыки разработки программ на языке C# в среде разработки Unity.

В рамках выпускной квалификационной работы было создано игровое приложение под ОС Android с интуитивно понятным пользовательским интерфейсом. В приложении реализованы все стандартные элементы игры в жанре платформер. Данный проект успешно достиг большинства поставленных целей.

В будущем игра может быть дальше совершенствована в соответствии с потребностями пользователей.

Список использованных источников

1. Лучшие современные движки для разработок приложений [Электронный ресурс]. URL: <https://habr.com/ru/company/timeweb/blog/659891/>. (дата обращения: 29.04.2023)
2. *Unity* [Электронный ресурс]. URL: <https://docs.unity3d.com/Manual/>. (дата обращения: 5.05.2023)
3. Харрисон Ферроне. *Learning C# by Developing Games with Unity* [Электронный ресурс]. URL: <https://www.programmer-books.com/wp-content/uploads/2019/04/Learning-C-by-Developing-Games-with-Unity-5.x-2nd-Edition.pdf>. (дата обращения: 10.05.2023)
4. *Unity Forums* [Электронный ресурс]. URL: <https://forum.unity.com/>. (дата обращения: 13.05.2023)
5. СНиП 23-05-95 – Естественное и искусственное освещение [Электронный ресурс]. URL: <https://docs.cntd.ru/document/871001026>. (дата обращения: 27.05.2023)

Приложение А

(обязательное)

Игровое приложение под ОС Андроид

Листинг программы

					ПГУ1.090301.02.001 ПЗ	Лист
						55
Изм.	Лист	№ докум.	Подпись	Дата		

Файл *MoveController.cs*

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

public class MoveController : MonoBehaviour
{
    public float speed;

    public float jumpForce;
    private float moveInput;
    public Joystick joystick;

    private Rigidbody2D rb;

    private bool facingRight = true;
    private bool isGrounded;
    public Transform feetPos;
    public float checkRadius;
    public LayerMask whatIsGround;
    private Animator anim;
    private int extraJump;
    public int extraJumpvalue;
    public AudioClip audioclip;
    private AudioSource audioSource;

    void Start()
    {
        audioSource = GetComponent<AudioSource>();
        anim = GetComponent<Animator>();
        rb = GetComponent<Rigidbody2D>();
        extraJump = extraJumpvalue;
    }
    private void OnTriggerEnter2D(Collider2D collision)
    {
        if (collision.CompareTag("point"))
        {
            audioSource.PlayOneShot(audioclip);
        }
    }

    void Update()
    {
        isGrounded = Physics2D.OverlapCircle(feetPos.position,
        checkRadius, whatIsGround);

        if (isGrounded == false && rb.velocity.y >=.1f)
        {
            anim.SetInteger("fall", 2);
        }
    }
}
```

```

    }
    if (isGrounded == false && rb.velocity.y <= -.1f)
    {
        anim.SetInteger("fall", 1);
    }
    if (isGrounded == false && extraJump == 0 &&
rb.velocity.y >= .1f)
    {
        anim.SetInteger("fall", 4);
    }
    if (isGrounded == true)
    {
        anim.SetInteger("fall", 0);
        extraJump = extraJumpvalue;
    }
}

public void jumpButton()
{

    if(isGrounded== true )
    {
        rb.velocity = Vector2.up * jumpForce;
    }
    if (isGrounded == false && extraJump > 0)
    {
        rb.velocity = Vector2.up * jumpForce;
        extraJump--;
    }
    else if (extraJump == 0 && isGrounded == false)
    {
    }

}

void FixedUpdate()
{
    moveInput = joystick.Horizontal;
    rb.velocity = new Vector2(moveInput * speed,
rb.velocity.y);
    if(facingRight == false && moveInput > 0)
    {
        Flip();
    }
    else if(facingRight == true && moveInput < 0)
    {
        Flip();
    }
    if(moveInput == 0)
    {

```

```

        anim.SetBool("isRunning", false);
    }
    else if ((moveInput <= -0.01f || moveInput >= 0.01f) &&
isGrounded == true)
    {
        anim.SetBool("isRunning", true);
    }
}

void Flip()
{
    facingRight = !facingRight;
    Vector3 Scaler = transform.localScale;
    Scaler.x *= -1;
    transform.localScale = Scaler;
}
}

```

Файл *Health.cs*

```

using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;

public class Health : MonoBehaviour
{
    public int health;
    public int maxHealth;
    public LevelRestart levelrestart;

    public void TakeHit(int damage)
    {
        health -= damage;
        if (health <= 0)
        {
            pointText.Point = 0;
            Destroy(gameObject);

            SceneManager.LoadScene("menu");
        }
    }
    public void SetHealth(int bonusHealth)
    {
        health += bonusHealth;
        if (health > maxHealth)
        {
            health= maxHealth;
        }
    }
}

```

```

    }
}

```

Файл *CollisionDamage.cs*

```

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

public class CollisionDamage : MonoBehaviour
{
    public int collisionDamage = 10;
    public string collisionTag;
    private void OnCollisionEnter2D(Collision2D coll)
    {
        Health health = coll.gameObject.GetComponent<Health>();
        health.TakeHit(collisionDamage);
    }
}

```

Файл *Finish.cs*

```

using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;
public class Finish : MonoBehaviour
{
    private void OnTriggerEnter2D(Collider2D collision)
    {
        if (collision.CompareTag("Player"))
        {
            NextLevel();
        }
    }
    public void NextLevel()
    {
        var index = SceneManager.GetActiveScene().buildIndex;
        SceneManager.LoadScene(index + 1);
    }
}

```

Файл *Finish.cs*

```

using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;

public class LevelRestart : MonoBehaviour
{

```

```

    public void RestartLevel()
    {
        int currentSceneIndex =
SceneManager.GetActiveScene().buildIndex;
        SceneManager.LoadScene(currentSceneIndex);
    }
}

```

Файл *persistent.cs*

```

using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;

public class persistent : MonoBehaviour
{
    void Start()
    {
        DontDestroyOnLoad(gameObject);
    }
    void Update()
    {
        if (SceneManager.GetActiveScene().name == "menu")
        {
            Destroy(gameObject);
        }
    }
}

```

Файл *platform.cs*

```

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

public class platform : MonoBehaviour
{
    public Transform startPoint;
    public Transform endPoint;
    public float speed = 2f;
    private bool movingToEndPoint = true;

    private void Update()
    {
        Vector3 targetPosition = movingToEndPoint ?
endPoint.position : startPoint.position;

        transform.position =
Vector3.MoveTowards(transform.position, targetPosition, speed *
Time.deltaTime);
    }
}

```

```

        if (transform.position == targetPosition)
        {
            movingToEndPoint = !movingToEndPoint;
        }
    }
    private void OnCollisionEnter2D(Collision2D collision)
    {
        if (collision.gameObject.CompareTag("Player"))
        {
            collision.transform.SetParent(transform);
        }
    }
    private void OnCollisionExit2D(Collision2D collision)
    {
        if (collision.gameObject.CompareTag("Player"))
        {
            collision.transform.SetParent(null);
        }
    }
}

```

Файл *point.cs*

```

using System.Collections;
using System.Collections.Generic;
using UnityEngine;
public class Point : MonoBehaviour
{
    private void OnTriggerEnter2D(Collider2D collision)
    {
        if (collision.CompareTag("Player"))
        {
            pointText.Point += 1;
            Destroy(gameObject);
        }
    }
}

```

Файл *pointText.cs*

```

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

public class pointText : MonoBehaviour
{
    public static int Point;
    Text text;
    void Start()

```

```

    {
        text = GetComponent<Text>();
    }
    void Update()
    {
        text.text = Point.ToString();
    }
}

```

Файл *Record.cs*

```

using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;

public class Record : MonoBehaviour
{
    public Timer complete;
    void Start()
    {
        complete =
GameObject.Find("GameManager").GetComponent<Timer>();
    }
    private void OnTriggerEnter2D(Collider2D collision)
    {
        if (collision.CompareTag("Player"))
        {
            NextLevel();
            complete.CompleteLevel();
        }
    }
    public void NextLevel()
    {
        var index = SceneManager.GetActiveScene().buildIndex;
        SceneManager.LoadScene(index + 1);
    }
}

```

Файл *ReloadOnDestroyScene.cs*

```

using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;

public class ReloadOnDestroyScene : MonoBehaviour
{
    private void OnDestroy()
    {
        pointText.Point = 0;
    }
}

```

```

        SceneManager.LoadScene("menu");
    }
}

```

Файл *spikehead.cs*

```

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

public class spikehead : MonoBehaviour
{
    public Transform startPoint;
    public Transform endPoint;
    public float speed = 2f;
    private bool movingToEndPoint = true;
    private void Update()
    {
        Vector3 targetPosition = movingToEndPoint ?
endPoint.position : startPoint.position;
        transform.position =
Vector3.MoveTowards(transform.position, targetPosition, speed *
Time.deltaTime);
        if (transform.position == targetPosition)
        {
            movingToEndPoint = !movingToEndPoint;
            gameObject.SetActive(true);
        }
    }
}

```

Файл *Timer.cs*

```

using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
public class Timer : MonoBehaviour
{
    public float currentTime;
    public Text timeText;
    public Text recordText;
    private Coroutine timerCoroutine;
    public float bestTime;
    void Start()
    {
        StartTimer();
    }
    public void StartTimer()
    {
        timerCoroutine = StartCoroutine(UpdateTimer());
    }
}

```

```

        bestTime = PlayerPrefs.GetFloat("BestTime", 0f);
        recordText.text = "Record: " +
bestTime.ToString("0.00");
    }
    private IEnumerator UpdateTimer()
    {
        while (true)
        {
            currentTime += Time.deltaTime;
            timeText.text = "Time: " +
currentTime.ToString("0.00");
            yield return null;
        }
    }
    public void CompleteLevel()
    {
        if (timerCoroutine != null)
        {
            StopCoroutine(timerCoroutine);
            timerCoroutine = null;

            if (currentTime < bestTime || bestTime == 0f)
            {
                bestTime = currentTime;
                recordText.text = "Record: " +
bestTime.ToString("0.00");
                PlayerPrefs.SetFloat("BestTime", bestTime);
                Debug.Log(bestTime);
            }
        }
    }
}

```

Файл *play.cs*

```

using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;

public class play : MonoBehaviour
{
    public void level1()
    {
        SceneManager.LoadScene("level1");
    }
    public void ExitGame()
    {
        Application.Quit();
    }
}

```

}

Файл *play.cs*

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;
public class backtomenu : MonoBehaviour
{
    public void OpenMenu()
    {
        SceneManager.LoadScene("Menu");
        pointText.Point = 0;
    }
}
```

					ПГУ1.090301.02.001 ПЗ	Лист
						65
Изм.	Лист	№ докум.	Подпись	Дата		

Приложение Б

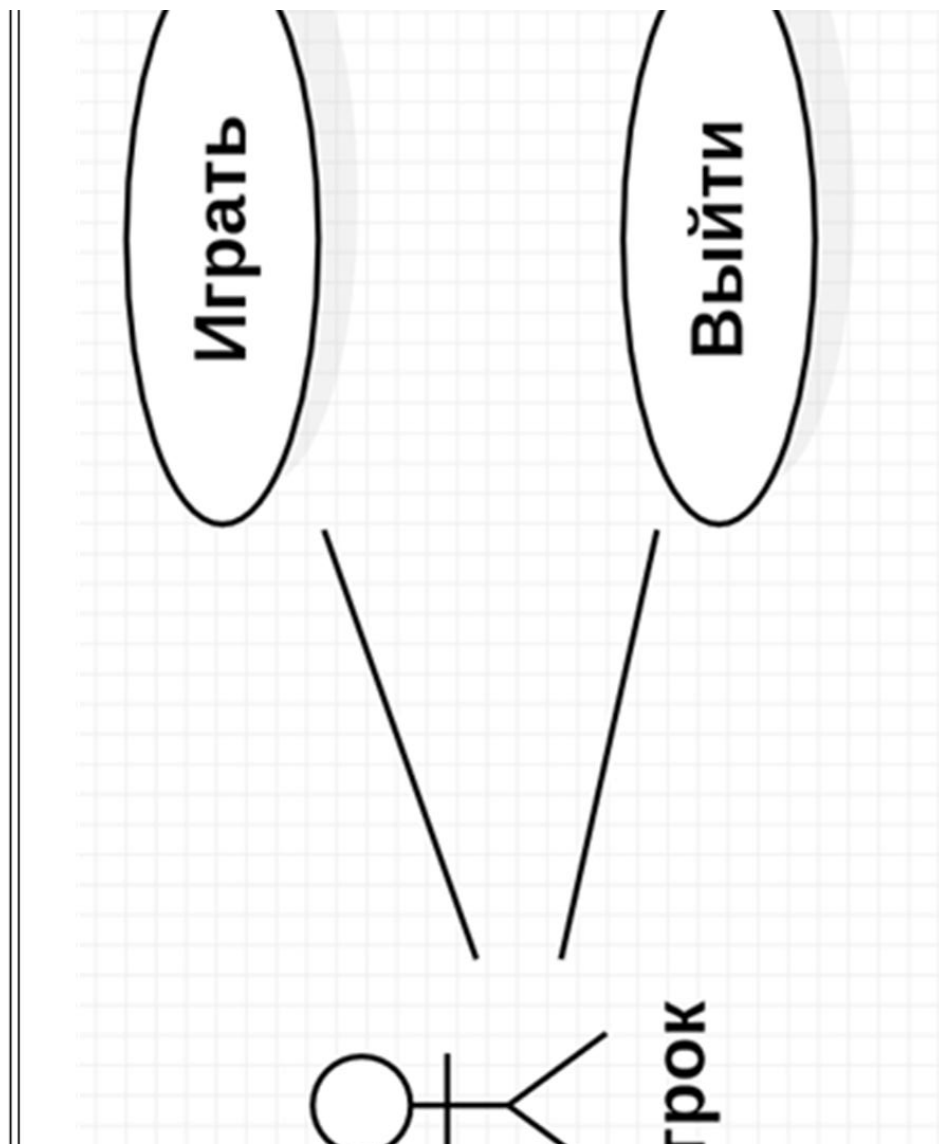
(обязательное)

Игровое приложение под ОС Андроид

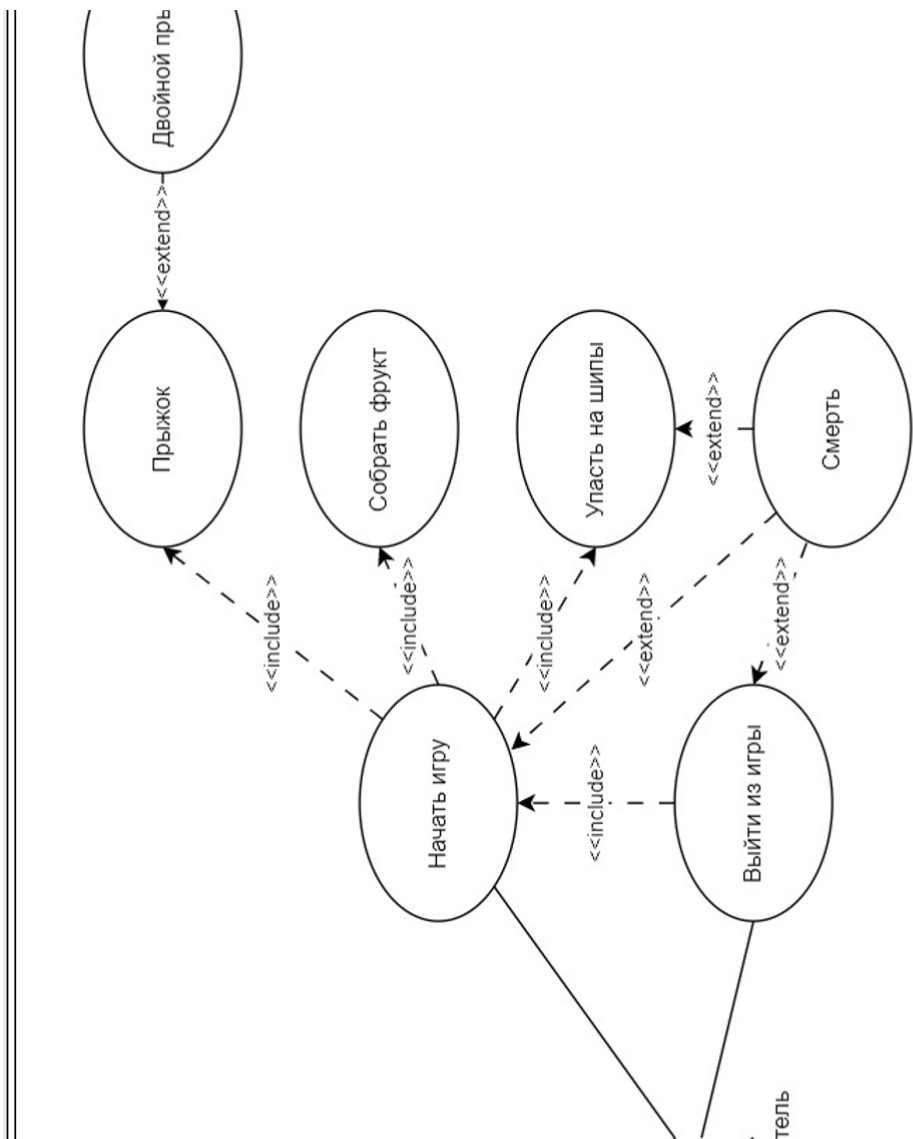
UML диаграммы

					ПГУ1.090301.02.001 ПЗ	Лист
						66
Изм.	Лист	№ докум.	Подпись	Дата		

Приложение Б. 1 – UML – диаграмма прецедентов меню

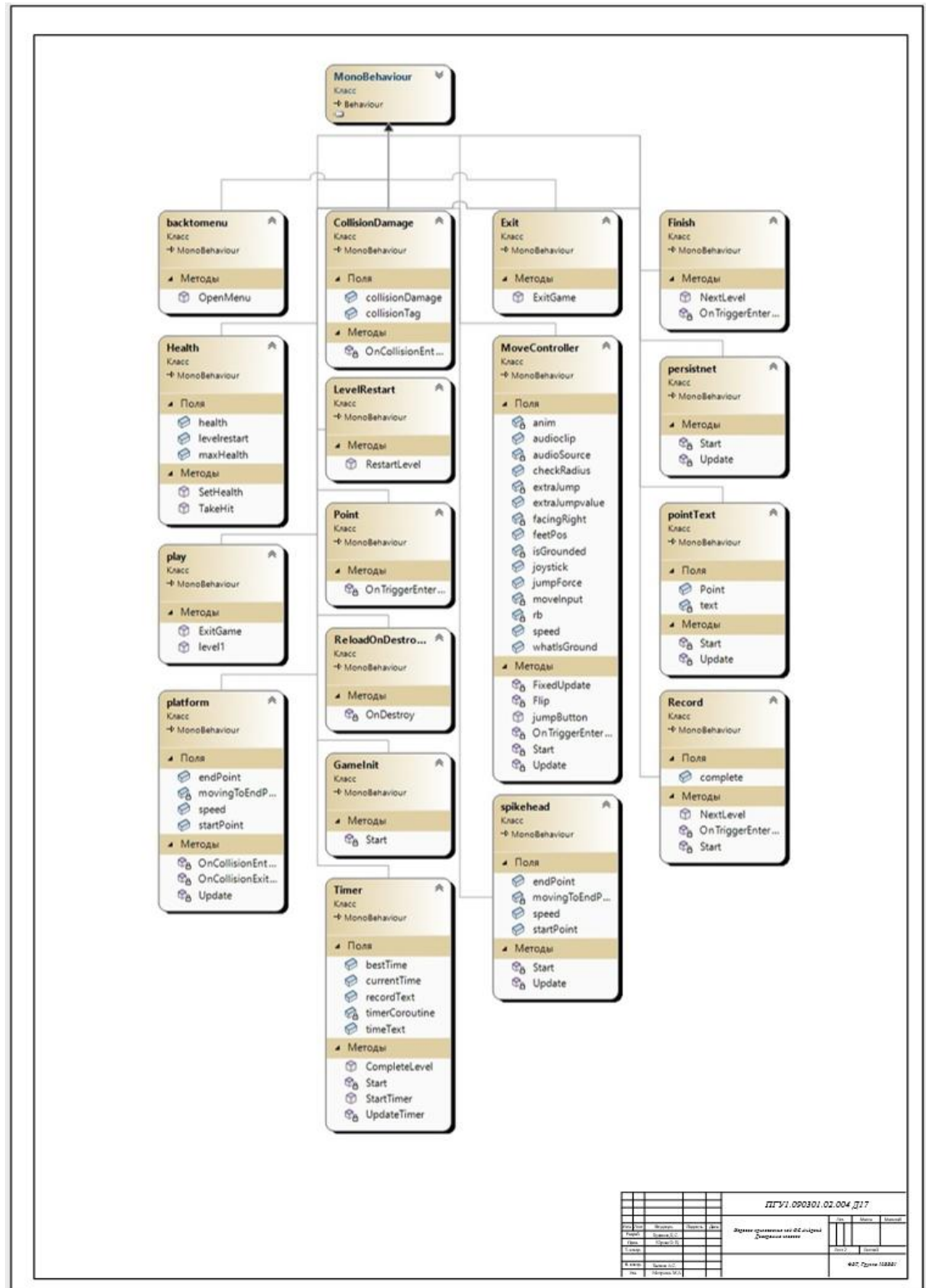


Приложение Б. 2 – UML – диаграмма прецедентов

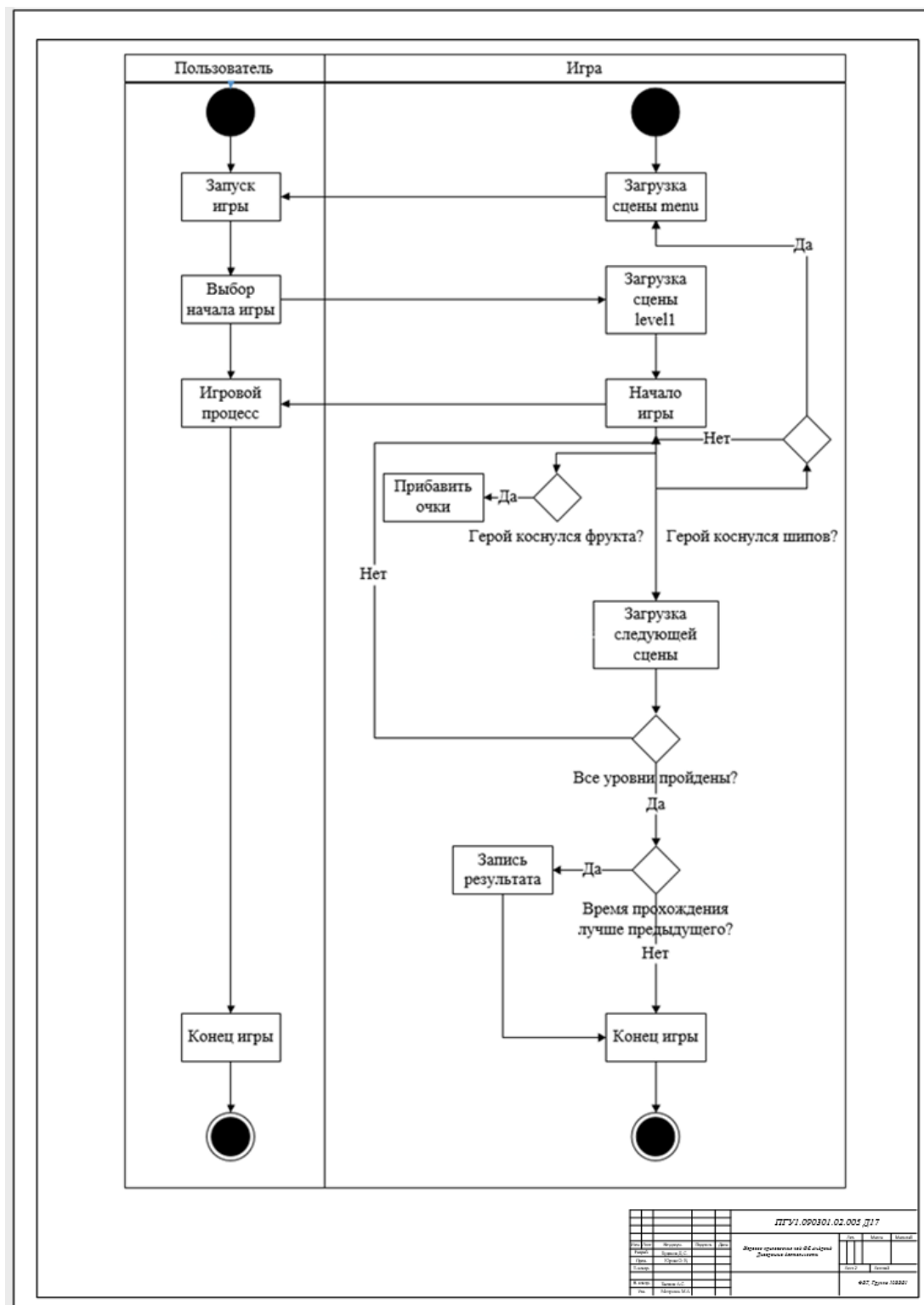


Изм.	Лист	№ докум.	Подпись	Дата

Приложение Б. 3 – UML – диаграмма классов



Приложение Б. 4 – UML – диаграмма деятельности



ПГУ1.090301.02.001 Д17									
Имя	Фамилия	Отчество	Дата	Подпись	Место	Дата	Подпись	Место	Дата
Имя	Фамилия	Отчество	Дата	Подпись	Место	Дата	Подпись	Место	Дата
Имя	Фамилия	Отчество	Дата	Подпись	Место	Дата	Подпись	Место	Дата
Имя	Фамилия	Отчество	Дата	Подпись	Место	Дата	Подпись	Место	Дата
Имя	Фамилия	Отчество	Дата	Подпись	Место	Дата	Подпись	Место	Дата
Имя	Фамилия	Отчество	Дата	Подпись	Место	Дата	Подпись	Место	Дата
Имя	Фамилия	Отчество	Дата	Подпись	Место	Дата	Подпись	Место	Дата
Имя	Фамилия	Отчество	Дата	Подпись	Место	Дата	Подпись	Место	Дата
Имя	Фамилия	Отчество	Дата	Подпись	Место	Дата	Подпись	Место	Дата

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ1.090301.02.001 ПЗ

Приложение В

(обязательное)

Игровое приложение под ОС Андроид

Презентация

					ПГУ1.090301.02.001 ПЗ	Лист
						71
Изм.	Лист	№ докум.	Подпись	Дата		

ИГРОВОЕ ПРИЛОЖЕНИЕ ПОД ОС АНДРОИД

Автор работы:

студент группы 19ВВВ1 Буданов Д. С.

Научный руководитель:

к. т. н., доцент Юрова О. В.

Цель работы

Цель работы – разработать игровое приложение под ОС Андроид.
Приложение должно удовлетворять следующим требованиям:

- Интуитивно понятный игровой процесс
- Яркая и привлекательная графика
- Несколько уровней
- Игра должна быть в меру сложная
- Качественное и увлекательное звуковое сопровождение
- Адаптация к разным разрешениям экранов

Мобильные устройства – неотъемлемая часть нашей жизни



Мобильные игры в настоящее время являются одним из наиболее перспективных сегментов игровой индустрии.



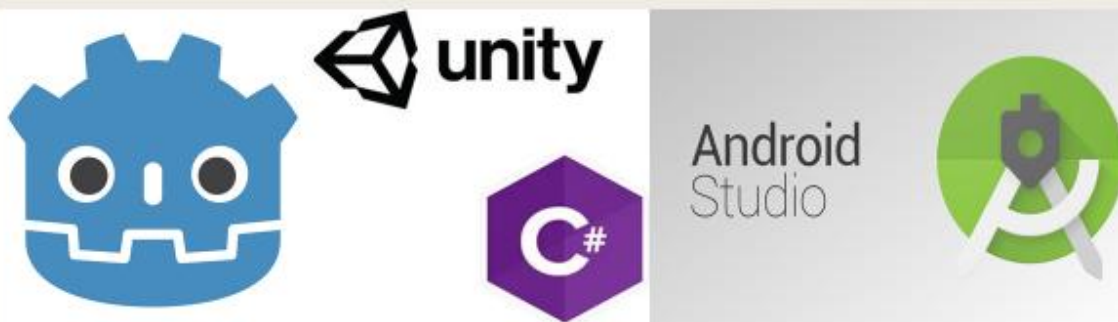
Изм.	Лист	№ докум.	Подпись	Дата

ПГУ1.090301.02.001 ПЗ

Лист

73

Выбор среды разработки



Выбор инструментов скриптинга



Изм.	Лист	№ докум.	Подпись	Дата

ПГУ1.090301.02.001 ПЗ

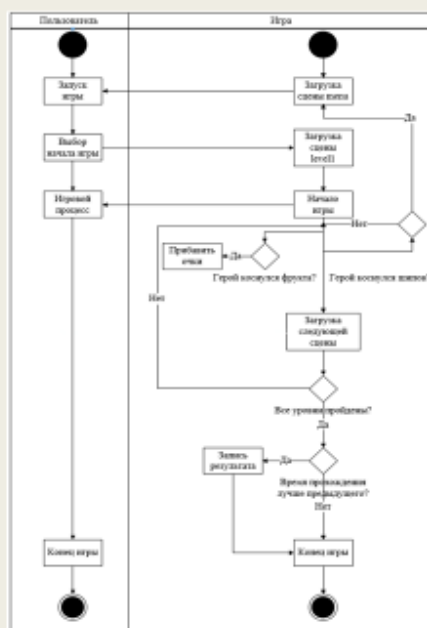
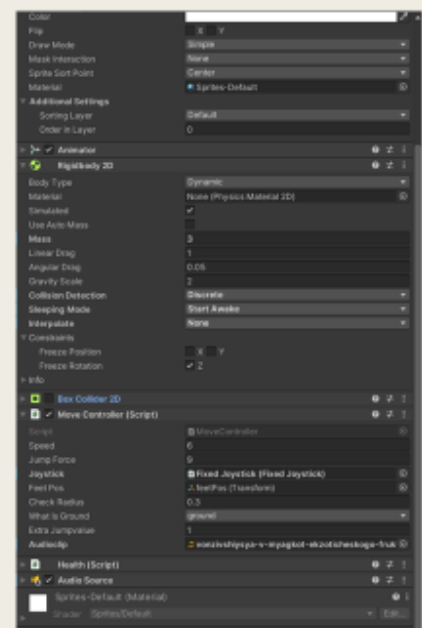
Лист

74

Диаграмма прецедентов



Базовая логика



Реализация анимаций

Бег:



Двойной прыжок:



Спокойствие:



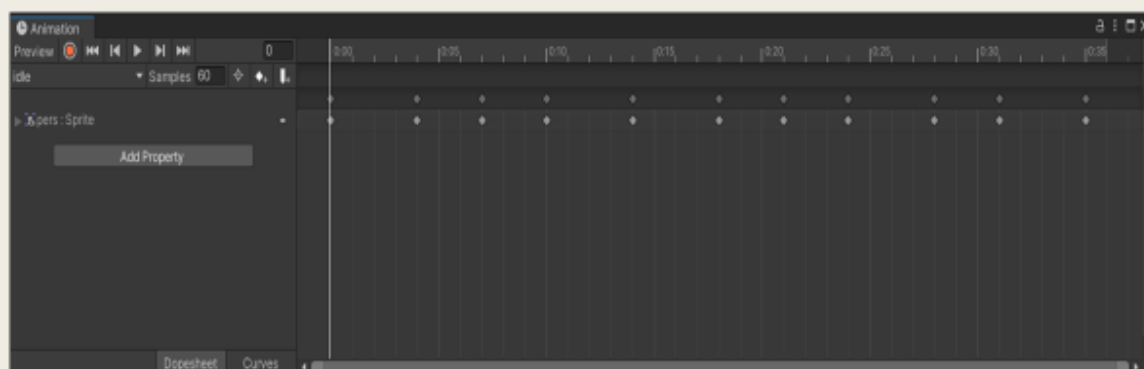
Прыжок:



Падение:



Реализация анимаций



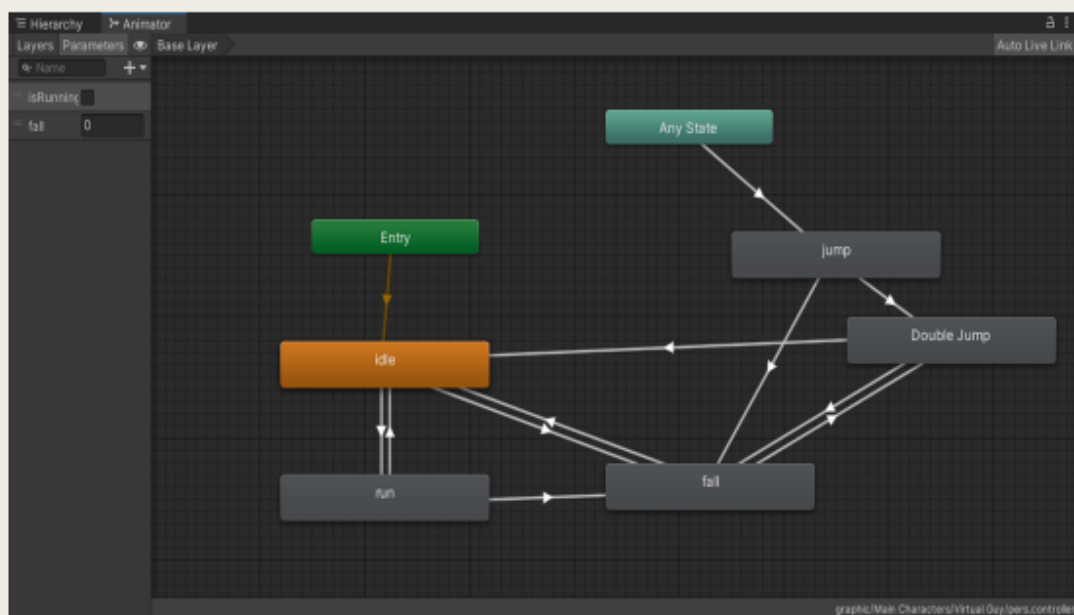
Изм.	Лист	№ докум.	Подпись	Дата

ПГУ1.090301.02.001 ПЗ

Лист

76

Реализация анимаций



Структура уровней



Изм.	Лист	№ докум.	Подпись	Дата

ПГУ1.090301.02.001 ПЗ

Лист

77

Структура уровней



Интерфейс главного меню



Изм.	Лист	№ докум.	Подпись	Дата

ПГУ1.090301.02.001 ПЗ

Лист
78

Интерфейс



Заключение

В ходе работы над проектом были изучены новые методы программирования и проектирования систем, а также значительно улучшены навыки разработки программ на языке С# в среде разработки Unity.

В рамках выпускной квалификационной работы было создано игровое приложение под ОС Android с интуитивно понятным пользовательским интерфейсом. В приложении реализованы все стандартные элементы игры в жанре платформер. Данный проект успешно достиг большинства поставленных целей.

В будущем игра может быть дальше совершенствована в соответствии с потребностями пользователей.

Изм.	Лист	№ докум.	Подпись	Дата

Спасибо за
внимание!

					ПГУ1.090301.02.001 ПЗ	Лист
						80
Изм.	Лист	№ докум.	Подпись	Дата		