МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное бюджетное образовательное учреждение
высшего образования
«Тольяттинский государственный университет»

Институт математики, физики и информационных технологий
(наименование института полностью)

Кафедра «Прикладная математика и информатика»
(наименование)

01.03.02 Прикладная математика и информатика
(код и наименование направления подготовки, специальности)

Компьютерные технологии и математическое моделирование
(направленность (профиль) / специализация)

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА (БАКАЛАВРСКАЯ РАБОТА)

на тему «Моделирование и визуализация физики объектов для мобильной игры на платформе Unity»

Обучающийся К.Д. Проскурнов  (личная подпись)
(И.О. Фамилия)

Руководитель доктор ф.-м. наук, профессор А.И. Сафронов
(ученая степень, звание, И.О. Фамилия)

Консультант к.п.н., доцент Т.С. Якушева

Тольятти 2023

Аннотация

Название дипломной работы: «Моделирование и визуализация физики объектов для мобильной игры на платформе Unity».

Дипломная работа состоит из пояснительной записки на 55 страниц, введения на 3 страницы, включая 38 рисунков, списка 24 источников, в том числе 5 источников на иностранном языке и трёх приложений, трех частей, заключения.

Ключевым вопросом дипломной работы является моделирование и визуализация физики объектов в игре. Мы затрагиваем проблему физических свойств внутри игровых объектов.

Целью дипломной работы является разработка игрового приложения для мобильных устройств на платформах IOS и Android.

Объектом дипломной работы является создание физических свойств объектов и их реалистичная визуализация в играх.

Предметом дипломной работы являются технологии моделирования и физической визуализации объектов для разработки мобильных игр на платформе Unity, а также анализ эффективности этих технологий на примере созданной мобильной игры.

Дипломную работу можно разделить на несколько логически связанных частей, которыми являются: анализ существующих подходов к реализации физики в играх; создание математической и физической основы для игровой механики и алгоритмов действия объектов; обоснование необходимости использования метода; выбор программного обеспечения для создания игры; выбор программного обеспечения для создания графики; приведение примеров других подобных проектов; программная реализация игры.

В конце исследования мы представим работу успешно созданного игрового проекта, который затем смогут использовать люди по всему миру.

Подводя итоги, мы хотели бы подчеркнуть, что данная работа актуальна не только для этого проекта, но и для будущих более крупных проектов.

Abstract

The title of the graduation work is: «Object physics modelling and visualisation for a mobile game on the Unity platform».

The thesis consists of an explanatory note of 55 pages, an introduction of 3 pages including 38 figures, a list of 24 sources including 5 foreign language sources and three appendices, three parts, a conclusion.

The key issue of the thesis is the modelling and visualising the physics of objects in the game. We touch on the problem of physical properties within game objects.

The aim of the work is to develop a game application for mobile devices on the IOS and Android platforms.

The object of the graduation work to create physical properties of objects and their realistic visualisation in games.

The subject of the thesis is the technologies of modelling and physical visualisation of objects for the development of mobile games on the Unity platform, as well as the analysis of the effectiveness of these technologies on the example of a created mobile game.

The graduation work may be divided into several logically connected parts, which are: analysis of existing approaches to physics implementation in games; creation of mathematical and physical basis for game mechanics and object action algorithms; justification of the necessity to use the method; selection of software for game creation; selection of software for graphics creation; giving examples of other similar projects; software implementation of the game.

At the end of the study, we will present the work of a successfully created game project that can then be used by people all over the world.

To summarise, we would like to stress that this work is not only relevant to this project, but also to future larger projects where this experience will also be needed.

Оглавление

Введение

«Что такое компьютерная игра? Компьютерная игра (или видеоигра) - это организация игрового процесса на компьютере между двумя людьми (и более), посредством специальной компьютерной программы.

Появились первые компьютерные игры в 1950-ых - 1960-ых годах. Конечно же, они были не такими, какими мы привыкли видеть игры - это были простейшие примитивные игровые программы. В настоящее время игры - это многомиллиардная индустрия развлечений, в которой присутствует большое количество производителей игр» [1].

«Компьютерные игры в настоящее время выделяются в отдельную область знаний. Для некоторых категорий игр даже проводятся соревнования, в которых игроки соперничают друг с другом. Появилось даже такое понятие как "киберспорт". Есть попытки выделить компьютерные игры как отдельный вид искусства, что не очень удивляет, ведь, порою, некоторые игры так поражают воображение своей красотой, яркостью, красочностью, реалистичностью, которые при этом могут сочетаться с потрясающей музыкой, идеально подобранной для того или иного момента игры» [2].

«Игры зачастую делаются по какому-нибудь кассовому фильму или книге. Бывает и наоборот - фильм снимается по популярной игре, как например, фильм "Макс Пейн" был снят по мотивам небезызвестной одноименной игры "Max Payne". Если компьютерная игра пользуется популярностью, то, как правило, делается продолжение игры - может появиться и вторая, и третья часть, а то и больше. Кроме того, что бывают большие и мини-игры (казуальные игры), игры для мобильных телефонов, игровые приставки к телевизору, простые и сложные игры - есть ещё и деление игр по жанрам. Причем, это деление является основным, так как жанр присутствует в любой игре» [3]. Касаемо этой работы жанр игры будет казуальным.

В игре будет использоваться баллистика.

«Баллистика — раздел механики, изучающий, прежде всего, движение снарядов в поле тяжести Земли. В шутерах движение снарядов, очевидно, основной элемент игры. Но и здесь есть нюансы. Есть игры, где выстрел выглядит очень правдоподобно (например, Battlefield и Call of Duty). Наряду с ними есть игры, где вместо привычной криволинейной траектории используется Hitscan (ХитСкан). Это так называемая система выстрел — попадание, где вместо объекта выходящего из оружия снаряда используется луч. Hitscan встречается в Quake 1 и 2, Wolfenstein 3D и многих других» [23].

Касаемо этого проекта баллистика будет использоваться при демонстрации выстрела из пушки(артиллерии). Суть игры: мощным и точным выстрелом из пушки поразить противника, который прячется за деревянными стенками. Стенки служат как прикрытие для врага. Противников в игре всего 3. После того, как игрок поразит всех трёх противников, он выигрывает и переходит на следующий уровень, если же игрок не убивает всех трёх, то он проигрывает и может начать игру заново.

Цель работы – постановка задачи максимально воссоздать физику объектов в игре, а также создать полностью готовую игру по выше описанному сценарию под платформу Android и IOS.

Объект исследования – внутри игровая физика объектов и система классической схемы метания ядра(снаряда).

Предмет исследования – технологии моделирования и визуализации физики объектов для разработки мобильных игр на платформе Unity, а также анализ эффективности использования данных технологий на примере созданной мобильной игры.

Гипотеза: процессы метания, применяемые в игре, могут быть смоделированы на основе физических и математических формул, используемых при описании работы классической баллистической системы.

Задачи исследования:

1. Изучение классической схемы метания, используемой в баллистических системах.

2. Изучить особенности создания физики для внутри игровых объектов.

3. Провести анализ существующих математических подходов для реализации физики в игре.

4. Разработать схему выстрела из пушки, полёт ядра и схему поражения ядром противника.

5. Создание пользовательского управления и разработка главного меню для пользователя.

6. Реализация программы, то есть игры.

Результаты исследования могут быть использованы для создания другого проекта или же для расширение данного проекта, так же для разработки усовершенствований выше приведённых задач исследований.

Основными **методами исследования** в данной работе являются:

– Математическое и физическое моделирование, включающее в себя реализацию выстрела из пушки и полёта ядра.

– Численное моделирование, которое позволяет отслеживать нажатия мышкой и получить расположение нажатия.

Научная новизна данной работы заключается в разработке реалистичной схемы выстрела из пушки.

Практическая значимость исследования заключается в возможности использования разработанных схем для создания более больших проектов. И для разработчика, это также огромный практический опыт, который будет получен в этом проекте. Кроме того, исследование может быть полезным для студентов, занимающихся разработкой программных решений и технологий, а также для обобщения и систематизации знаний об информационных технологиях.

Глава 1 Изучение существующих подходов в реализации игры

1.1 Среда разработки. Разновидность среды разработки

«Видеоигра — игра с использованием изображений, сгенерированных электронной аппаратурой. Другими словами, видео игра является электронной игрой, которая базируется на взаимодействии человека и устройства посредством визуального интерфейса, например, телевизора, монитора компьютера или телефона. Первой зарегистрированной игрой в 1947 г., которую можно отнести к видеоигре, является Ракетный симулятор, интерфейсом которого выступало устройство на базе электронно-лучевой трубки (ЭЛТ)» [4].

«Как создаются игры? В 2023 году игры в основном создаются в программах/игровых движках которые специально предназначенные для этого. Существует довольно много игровых движков, в которых можно создать свою собственную видео игру. Рассмотрим наиболее популярные игровые движки на момент 2023 года» [5].

1. «Unity - кроссплатформенная среда разработки компьютерных игр, разработанная американской компанией Unity Technologies. Unity позволяет создавать приложения, работающие на более чем 25 различных платформах, включающих персональные компьютеры, игровые консоли, мобильные устройства, интернет-приложения и другие. Выпуск Unity состоялся в 2005 году и с того времени идёт постоянное развитие. Основными преимуществами Unity являются наличие визуальной среды разработки, межплатформенной поддержки и модульной системы компонентов. К недостаткам относят появление сложностей при работе с многокомпонентными схемами и затруднения при подключении внешних библиотек. На Unity написаны тысячи игр, приложений, визуализации математических моделей, которые охватывают множество платформ и жанров. При этом Unity используется как крупными разработчиками, так и независимыми студиями» [6]. (Рисунок 1)

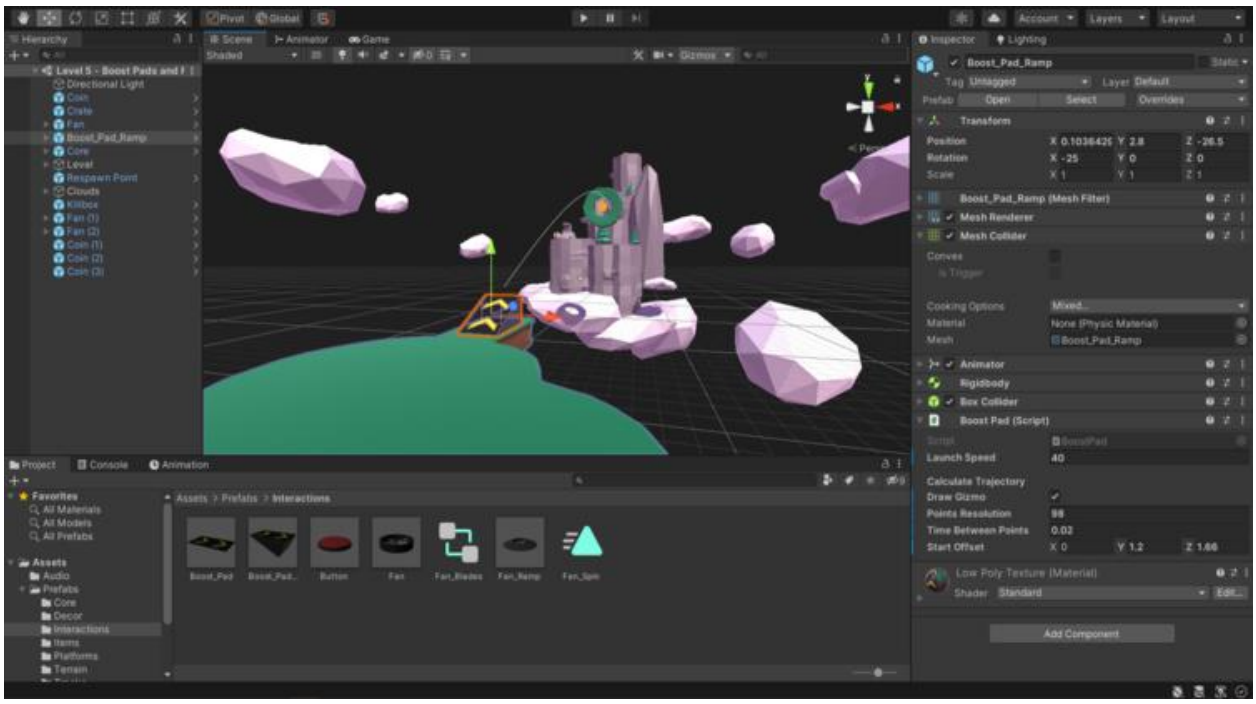

Рисунок 1 - Интерфейс Unity.

2. «Unreal Engine — игровой движок, разрабатываемый и поддерживаемый компанией Epic Games. Первой игрой на этом движке был шутер от первого лица Unreal, выпущенный в 1998 году. Хотя движок первоначально был предназначен для разработки шутеров от первого лица, его последующие версии успешно применялись в играх самых различных жанров, в том числе стелс-играх, файтингах и массовых многопользовательских ролевых онлайн-играх» [7]. «В прошлом движок распространялся на условиях оплаты ежемесячной подписки; с 2015 года Unreal Engine бесплатен, но разработчики использующих его приложений обязаны перечислять 5 % роялти от общемирового дохода с некоторыми условиями. Написанный на языке C++, движок позволяет создавать игры для большинства операционных систем и платформ: Microsoft Windows, Linux, Mac OS и Mac OS X; консолей Xbox, Xbox 360, Xbox One, PlayStation 2, PlayStation 3, PlayStation 4, PlayStation 5, PSP, PS Vita, Wii, Dreamcast, GameCube и др., а также на различных портативных устройствах, например, устройствах Apple (iPad, iPhone), управляемых системой iOS и прочих» [8]. (Рисунок 2)

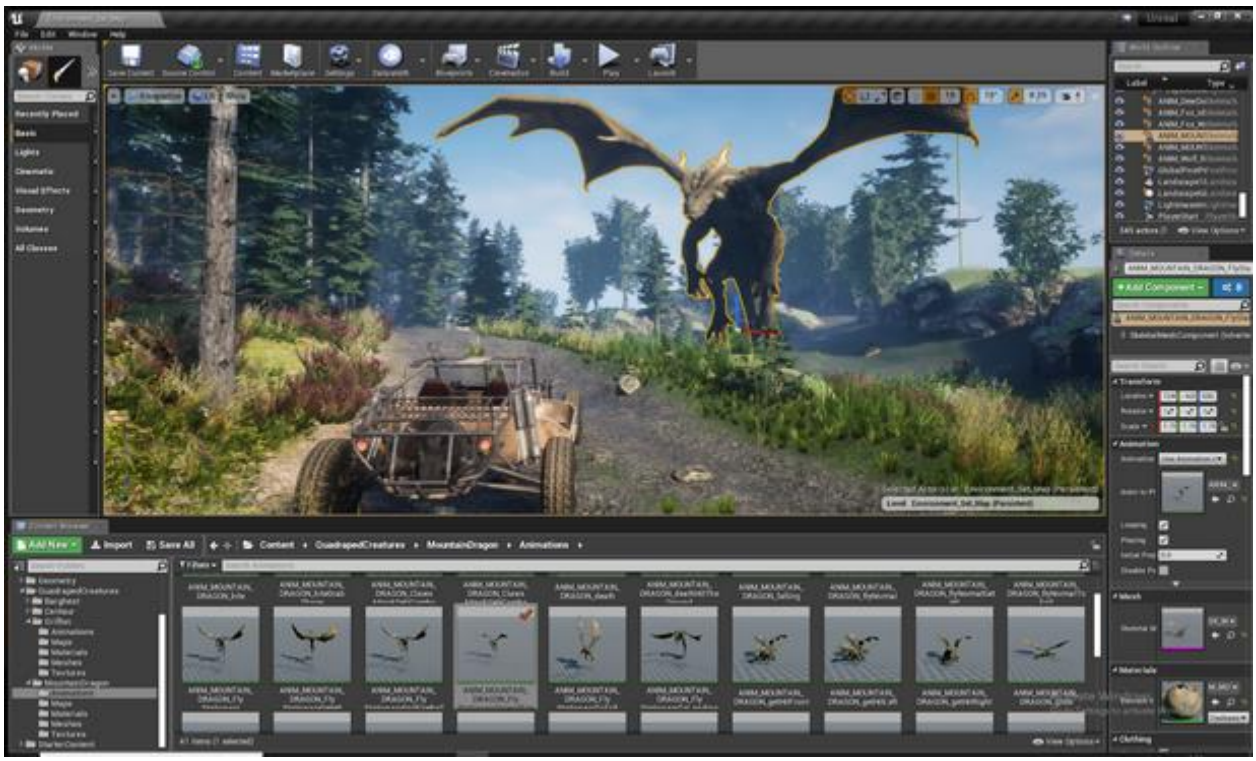

Рисунок 2 - Интерфейс Unreal Engine.

3. «Godot Engine - открытый кроссплатформенный 2D и 3D игровой движок под лицензией MIT, который разрабатывается сообществом Godot Engine Community. До публичного релиза в виде открытого ПО движок использовался внутри некоторых компаний Латинской Америки. Среда разработки запускается на Android, HTML5, Linux, macOS, Windows, BSD и Haiku и может экспортировать игровые проекты на ПК, консоли, мобильные и веб-платформы» [9]. «Задача Godot — быть максимально интегрированной и самодостаточной средой для разработки игр. Среда позволяет разработчикам создавать игры с нуля, не пользуясь более никакими инструментами, за исключением тех, которые необходимы для создания игрового контента (элементы графики, музыкальные треки и т. д.). Процесс программирования также не требует внешних инструментов (хотя при необходимости использовать внешний редактор, это можно сделать относительно легко)» [10]. (Рисунок 3)

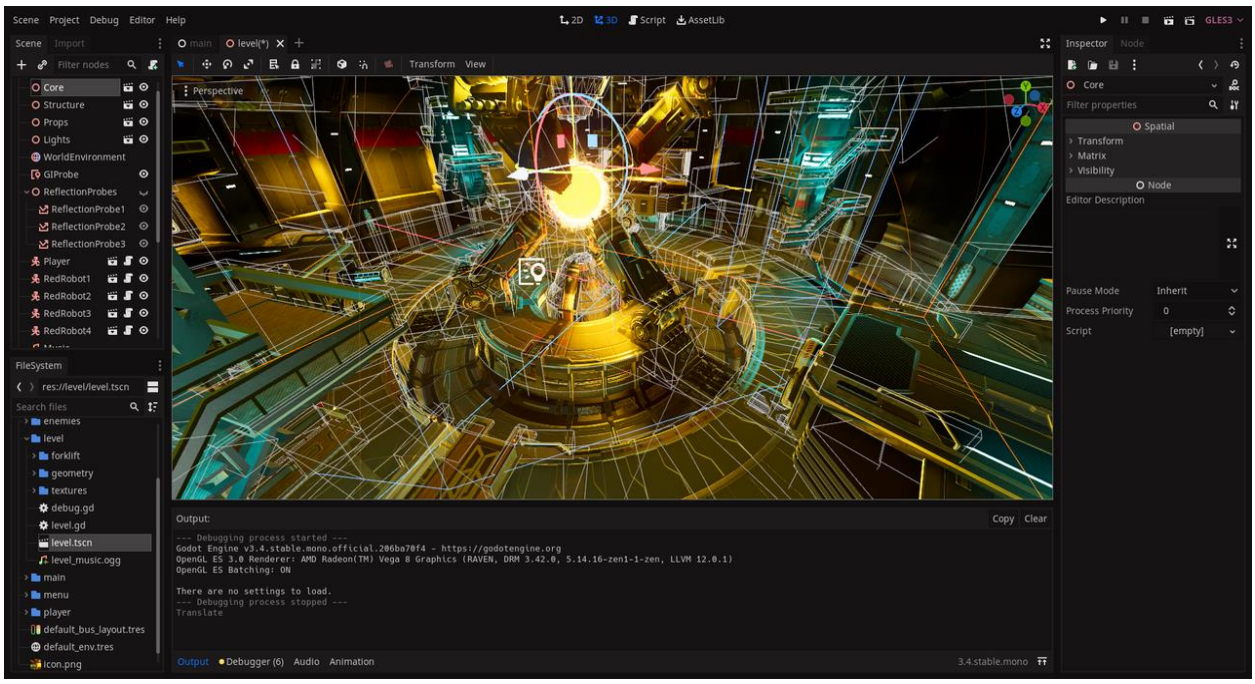

Рисунок 3 - Интерфейс Godot Engine.

Непосредственно в этой работе будет использоваться движок Unity. Так как он является самым популярным, а это значит, что и всяческой информации, уроков, статей также больше. Также Unity очень прост для понимания начинающим разработчикам. Ещё для Unity нужен редактор кота IDE, в котором будет удобно прописывать игровые скрипты. Для этого очень хорошо подходит IDE под названием Microsoft Visual Studio.

«Microsoft Visual Studio — линейка продуктов компании Microsoft, включающих интегрированную среду разработки программного обеспечения и ряд других инструментов» [11]. «Данные продукты позволяют разрабатывать как консольные приложения, так и игры, и приложения с графическим интерфейсом, в том числе с поддержкой технологии Windows Forms, UWP а также веб-сайты, веб-приложения, веб-службы как в родном, так и в управляемом кодах для всех платформ, поддерживаемых Windows, Windows Mobile, Windows CE, .NET Framework, .NET Core, .NET, MAUI, Xbox, Windows Phone .NET Compact Framework и Silverlight. После покупки компании Xamarin корпорацией Microsoft появилась возможность разработки IOS и Android программ» [12].

1.2 Описание принципов разработки игрового приложения

Чтобы создать качественную и интересную игру, необходимо соблюдать некоторые шаги создания или другими словами этапы, которые также ускорят процесс создания.

Этапы создания игры:

- создание концепта: Этот этап включает в себя продумывание общей концепции игры и дизайна. Это нужно для того, чтобы во время разработки игры разработчик не потерял первоначальную концепцию игры;
- продумывание логики игры: Это нужно для проверки гипотез, оценки игрового процесса и технических характеристик, продумывания монетизации и проведения различных тестов игровых механик;
- тестирование прототипа: Тестирование предоставляет возможность увидеть предварительную версию игры, чтобы лучше понимать цельный образ проекта. После тестирования прототипа возможно будут поправки в логике игры или в концепции;
- дизайн и разработка: Этот этап включает в себя доработку прототипа и адаптацию дизайна под него. Дизайн игры, а именно так называемые спрайты, их можно найти в интернете бесплатно, также можно купить у разработчика, либо создать свой дизайн игры самому, в таких программах как Blender, Adobe Illustrator, Adobe Photoshop (рисунок 4). Последний вариант очень спорный, потому что иногда разработчик не обладает художественным искусством;

Рисунок 4 — Дизайн игры.

- тестирование: Тестирование мобильной игры нужно для выявления различных ошибок, багов;
- релиз: в заключении, создаются запоминающиеся скриншоты для магазина приложений, прорабатываются заголовки и описания, и игра выгружается в App Store или Google Play.

В результате, благодаря всем этим этапам получится хорошо проработанная мобильная игра, которая будет радовать игроков по всему Миру.

Естественно выше предоставленные этапы создания игры не являются обязательными. Конечно, каждый разработчик либо некоторые копании, разрабатывают игры по-своему, но что-то общее есть у всех.

1.3 Анализ подобных приложений для определения основных функций и структурных компонентов

Ранее уже обсуждалась какова будет суть игры, что нужно делать чтобы выиграть в этой игре, но без визуального представления, суть игры понимается сложно. Поэтому необходимо рассмотреть несколько подобных игр из которых можно будет взять пример.

1. Первая игра, которая похоже на будущую игру этого проекта называется Fragger. Fragger — платформер (создан в 31 октября 2010 году), в котором цель игрока - взорвать всех врагов. На протяжении 130 уровней в четырех мирах игрок устанавливает взрывчатку и ловушки, чтобы отвоевать территорию у бандитов. Имеется сводная таблица рекордов и лучших бойцов. (Рисунок 5)

Рисунок 5 – Fragger - игра платформер.

Игра работает так: У персонажа несколько гранат, игроку необходимо бросить гранату как можно точнее, чтобы попасть в противника. Противники стоят неподвижно. Естественно, преграды за которыми стоят противники всяческие затрудняют и усложняют попадание. Чтобы кинуть гранату, игроку

необходимо: 1 - навести указатель(стрелку) в сторону противника, 2 - рассчитать силу броска персонажа, 3 - и наконец рассчитать свободное падение (гравитацию) самой гранаты, не так уж и просто. В этой игре уделено огромное внимание именно броску гранаты. В этом и весь интерес этой игры, а именно метко метнуть гранату.

2. Вторая игра называется - King Oddball. Игра - 2012 года выпуска, примерно похожая суть игры как в игре Fragger но механика игры то есть персонажем(снарядом) уже отличается. (Рисунок 6)

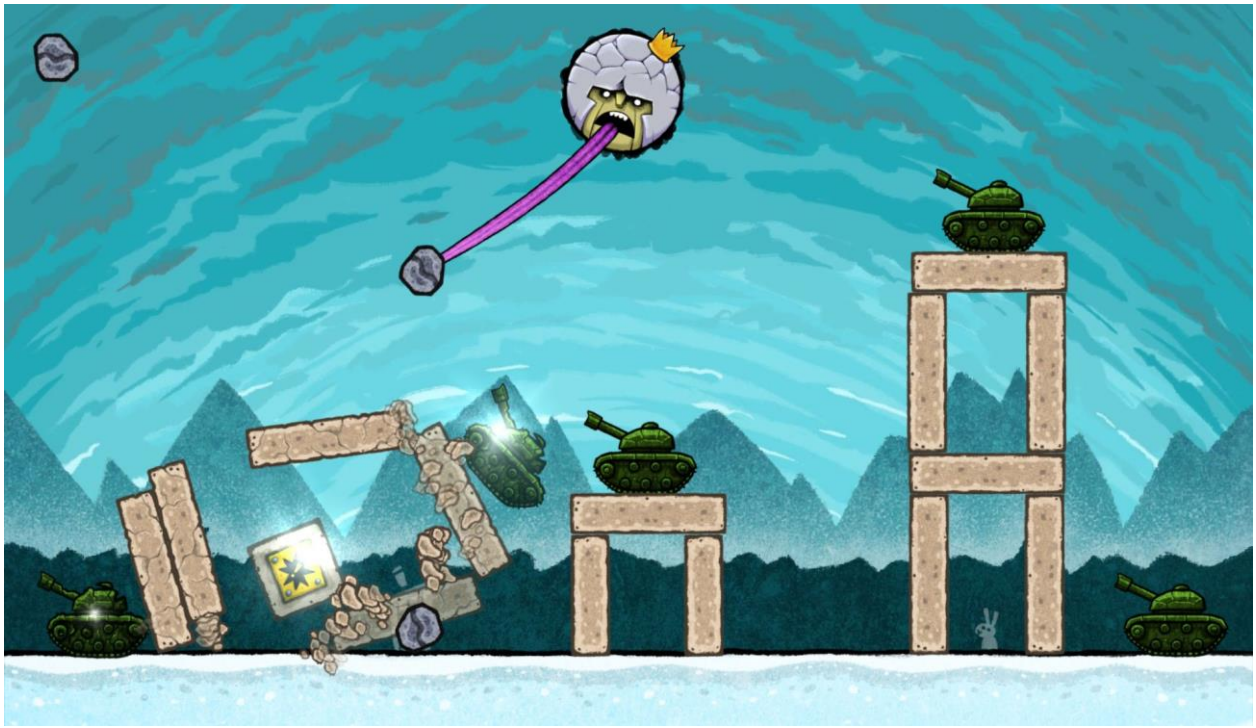

Рисунок 6 - King Oddball — казуальная игра.

Суть игры такова: У игрока есть 4 снаряда(камня), также на карте разбросаны противники(танки). Король раскачивает валун языком, а игрок должен отпустить камень, нажав на экран. Необходимо рассчитывать этот момент точно, чтобы уничтожить как можно больше целей каждым валуном, то есть необходимо предугадать по времени в какой момент сбросить камень, понимать, как камень полетит если опустить его и поэтому учитывать нужно гравитацию, скорость расстояние. Чтобы достичь вершины мастерства, нужно

научится просчитывать, как валуны будут катиться, подскакивать и отлетать при взрыве. Сделать обваливающиеся конструкции союзниками игрока!

3. Третья игра называется - The Archers 2. Год выхода - Август 2017 года. The Archers 2 – PVP-шутер для одного пальца, в котором игроку предстоит уничтожать врагов стрелами из лука. Также, как и предыдущие выше описанные игры имеет похожую схему метания только уже стрелы. Графика в The Archers 2 минимальная, однотонная, но от этого интерес к ней не чуть не уменьшается. (Рисунок 7)

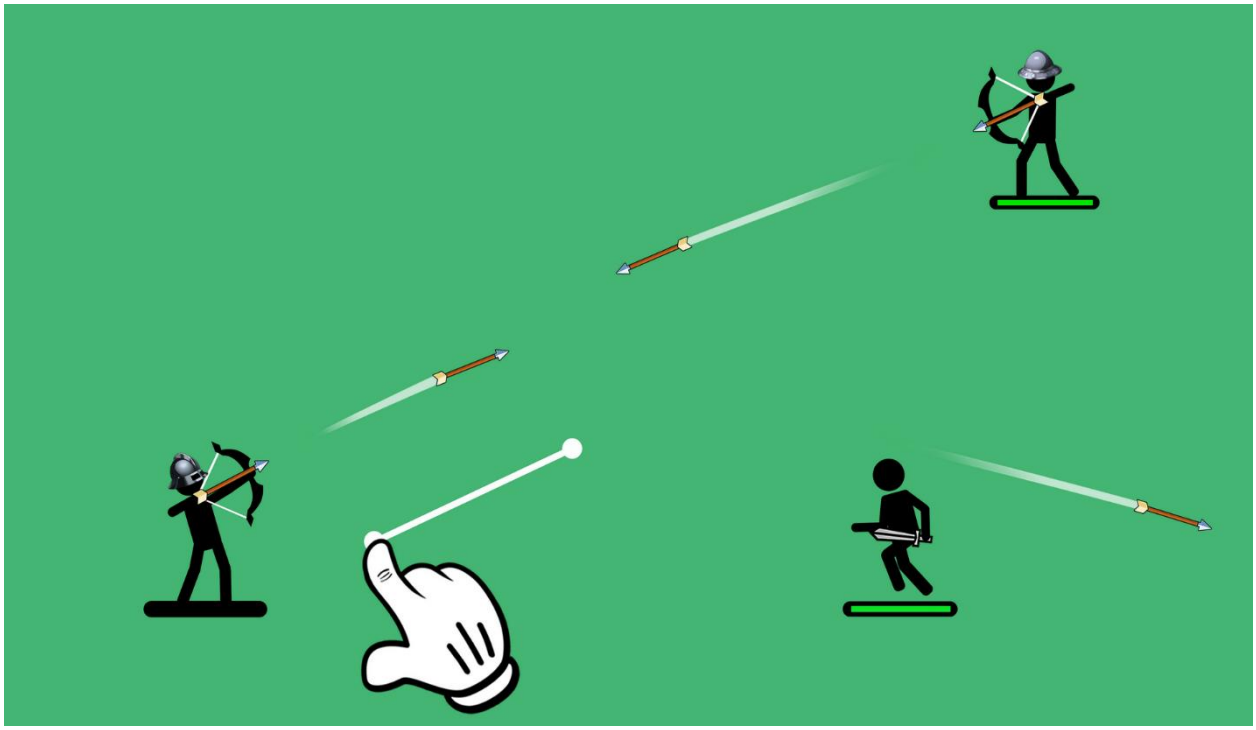

Рисунок 7 – The Archers 2 — казуальная игра пвп.

Динамичный таймкиллер, в котором победа зависит от скорости реакции игрока. Игроку в одиночку предстоит уничтожить бесконечные волны бегущих противников. Победить можно если попасть либо 2 раза в туловище, либо 1 раз в голову (хедшот). В отличии от подобных забав, тут нет никаких намеков, на то, куда полетит снаряд, а потому пару выстрелов придется потратить на пристрелку. Выстрел и полёт стрелы в этой игре приближён к реальному выстрелу из боевого лука, поэтому чтобы попасть в противника,

необходимо целится чуть выше так как на стрелу действуют законы физики, также следить за силой выстрела. В игре два режима – single player и multiplayer. Первый является компанией, где каждый последующий враг будет в новом месте. Второй же заключается в сражениях с живыми людьми. Правила здесь немного отличаются. Так, между выстрелами есть небольшая перезарядка и два попадания по ногам недостаточно для победы.

На этом обзор игр закончен. Примером из обозреваемых игр для этого проекта можно взять полёт/метания снаряда. Необходимо понять какие математические/физические формулы нужно использовать чтобы в игре воссоздать приближенную к реальному миру физику объектов. А именно гравитация, сила притяжение (рисунок 8).

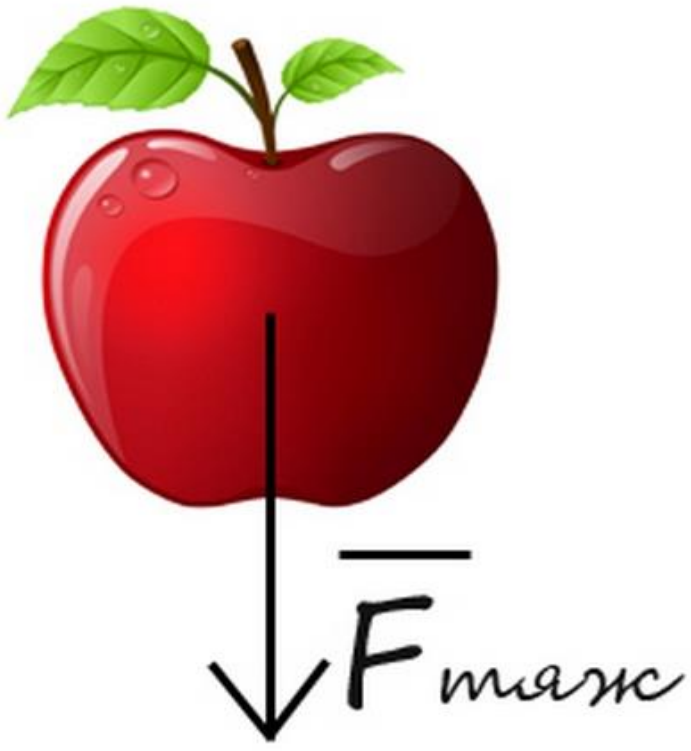

Рисунок 8 – Сила притяжения.

вес снаряда, взаимодействие снаряда с другими физическими телами (врагами, преградами) и многое другое.

В главе 1 пункте 1.1 (Среда разработки. Разновидность среды разработки), было рассмотрено, какие бывают среды разработки (движки) для игр, а именно было рассмотрено 3 движка (Unity, Godot Engine, Unreal Engine) и был выбран самый наилучший вариант для поставленной задачи. В качестве редактора кода был выбран Microsoft Visual Studio C#, так как он на сегодняшний день является один из самых лучших редакторов кода.

В пункте 1.2 (Описание принципов разработки игрового приложения), был выбран графический конструктор Adobe Photoshop, для создания игровых текстур. Определены и описаны этапы создания игры и основные требования к игре.

В пункте 1.3 (Анализ подобных приложений для определения основных функций и структурных компонентов), было рассмотрено три подобных проекта, из которых взяты подходы к реализации данного проекта.

Глава 2 Создание математической и физической основы игровых механик и алгоритмов действий объектов

2.1 Полёт снаряда. Физические свойства.

Как ранее было сказано, в игре будет метаться снаряд либо ядро, которое будет поражать условных противников. Необходимо сделать так чтобы снаряд метался очень реалистично, как в реальной жизни. К счастью это возможно сделать, так как движок Unity оснащён многими физическими свойствами и естественно позволяет это сделать.

«Для начала нужно уяснить отличие компьютерной физики от реальной. Реальная физика действует непрерывно (во всяком случае обратное не доказать на текущий момент). Компьютерная физика, как и компьютер действуют дискретно, т.е. не можем вычислять её непрерывно» [13].

Первое свойство, которым нужно оснастить снаряд — это физическая масса. Для этого используется второй закон Ньютона. «Второй закон Ньютона — дифференциальный закон механического движения, описывающий зависимость ускорения тела от равнодействующей всех приложенных к телу сил и массы тела. Один из трёх законов Ньютона. Основной закон динамики.

«Объектом, о котором идёт речь во втором законе Ньютона, является материальная точка, обладающая неотъемлемым свойством — инерцией, величина которой характеризуется массой. В классической (ньютоновской) механике масса материальной точки полагается постоянной во времени и не зависящей от каких-либо особенностей её движения и взаимодействия с другими телами» [14].

«Второй закон Ньютона в его наиболее распространённой формулировке, справедливой для скоростей, много меньших скорости света, утверждает: в инерциальных системах отсчёта ускорение, приобретаемое материальной точкой, прямо пропорционально вызывающей его силе, не зависит от её природы, совпадает с ней по направлению и обратно

пропорционально массе материальной точки» [15].

Обычно этот закон записывается в виде формулы:

где: $\bar{a}$ – ускорение тела,

$\bar{F}$ – сила, приложенная к телу,

m – масса тела.

$$\bar{a} = \frac{\bar{F}}{m}, \quad (1)$$

Или в ином виде:

$$m\bar{a} = \bar{F}, \quad (2)$$

где: $\bar{a}$ – ускорение тела,

$\bar{F}$ – сила, приложенная к телу,

m – масса тела.

Формулировка второго закона Ньютона с использованием понятия импульса:

$$\frac{d\bar{p}}{dt} = \bar{F}, \quad (3)$$

где: $d\bar{p}$ – вектор изменения импульса тела,

$d\bar{t}$ и $\bar{F}$ – вектора импульса силы.

В инерциальных системах отсчёта производная импульса материальной точки по времени равна действующей на неё силе:

$$\bar{p} = m\bar{v}, \quad (4)$$

где: $\bar{v}$ – импульс (количество движения) точки,

m – масса.

Второе физическое свойство, которое необходимо для полёта снаряда это — гравитация или Закон всемирного тяготения (Ньютона гравитация). «Классическая теория тяготения Ньютона (Закон всемирного тяготения Ньютона) — закон, описывающий гравитационное взаимодействие в рамках классической механики. Этот закон был открыт Ньютоном около 1666 года, опубликован в 1687 году в «Началах» Ньютона. Закон гласит, что сила F гравитационного притяжения между двумя материальными точками с массами m_1 и m_2, разделёнными расстоянием r, действует вдоль соединяющей

их прямой, пропорциональна обеим массам и обратно пропорциональна квадрату расстояния.

То есть:

$$F = G \cdot \frac{m_1 \cdot m_2}{r^2}, \tag{5}$$

где: G – гравитационная постоянная, равная: 6,67430(15)·10−11 м³/(кг·с²),

m_1 и m_2 – масса,

r – расстояние.

В теории тяготения Ньютона ускорение точечного или маленького тела под действием гравитационной силы всегда в точности равно напряжённости гравитационного поля в точке, в которой находится тело, определяемой как отношение:

$$\bar{g} = \bar{F} / m, \tag{6}$$

где: F – сила гравитационного притяжения,

m – масса.

Гравитационное поле в теории Ньютона является потенциальным, в связи с этим для его описания можно использовать гравитационный потенциал φ. В случае, если поле создаётся расположенной в начале координат точечной массой M, гравитационный потенциал определяется формулой:

$$\varphi(\bar{r}) = -G \frac{M}{r}, \tag{7}$$

где: φ – гравитационный потенциал,

$\varphi(\bar{r})$ – гравитационное поле,

G – гравитационная постоянная,

M– точечной массой,

r – расстояние.

В общем случае, когда плотность вещества ρ распределена произвольно, φ удовлетворяет уравнению Пуассона:

$$\Delta\varphi(\bar{r}) = -4\pi G p(\bar{r}) \tag{8}$$

где: Δ – оператор Лапласа,

$\varphi(\bar{r})$ – гравитационное поле,

p – плотность вещества,

G – гравитационная постоянная,

$\bar{r}$ – радиус вектор,

π – 3,14.

Решение данного уравнения записывается в виде:

$$\varphi(\bar{r}) = -G\int_{v} \frac{p(\bar{r}'dV')}{|\bar{r}-\bar{r}'|} + C, \qquad (9)$$

Здесь: $\bar{r}$ – радиус-вектор точки, в которой определяется потенциал,

$\bar{r}'$ – радиус-вектор элемента объёма dV′ с плотностью вещества,

$p(\bar{r}')$ – интегрирование охватывает все такие элементы;

C – произвольная постоянная чаще всего ее принимают равной нулю, как это сделано в формуле выше для одного точечного источника.

Сила притяжения, действующая в гравитационном поле на материальную точку с массой m, связана с потенциалом формулой:

$$\bar{F}(\bar{r}) = -m\Delta\varphi(\bar{r}), \qquad (10)$$

где: $\bar{F}(\bar{r})$ – Сила притяжения, действующая в гравитационном поле,

m – масса,

Δ – оператор Лапласа,

$\varphi(\bar{r})$ – гравитационное поле.

Если поле создаётся точечной массой M, расположенной в начале координат, то на точку массой m действует сила:

$$\bar{F}(\bar{r}) = -G\frac{mM}{r^3}\cdot\bar{r}, \qquad (11)$$

где: $\bar{F}(\bar{r})$ – Сила притяжения, действующая в гравитационном поле,

M – точечная масса, расположенная в начале координат,

m – масса,

$\bar{r}$ – радиус-вектор точки, в которой определяется потенциал,

G – гравитационная постоянная,

r – расстояние между массами.

«Величина этой силы зависит только от расстояния r между массами, но не от направления радиус-вектора $\bar{r}$» [16].

Третье физическое свойство — это скорость. «Мгновенной скоростью (или чаще просто скоростью) материальной точки называется физическая величина равная первой производной от радиуса – вектора точки по времени (t). Обозначают скорость обычно буквой v. Это векторная величина. Математически определение вектора мгновенной скорости записывается как:

$$\bar{v} = \frac{d\bar{r}}{dt}, \tag{12}$$

где: $\bar{v}$ – скорость,

$d\bar{r}$ – радиус,

dt – время.

Скорость имеет направление указывающее направление движения материальной точки и лежит на касательной к траектории ее движения. Модуль скорости можно определить, как первую производную от длины пути (s) по времени:

$$v = \frac{ds}{dt} = s, \tag{13}$$

где: v – скорость,

ds – длина пути,

dt – время.

«Скорость характеризует быстроту перемещения в направлении движения точки по отношению рассматриваемой системе координат» [17].

Четвёртое физическое свойство — это трение. Сила трения равна произведению коэффициента трения скольжения на силу реакции опоры и вычисляется по формуле (Рисунок 9).

Рисунок 9 – Сила трения.

Далее необходимо рассмотреть физические свойства других объектов в игре, таких как земля(ground), противники (монстры), деревянные укрытия/стойки.

2.2 Объекты в игре и их физические свойства.

Необходимо понять, как происходит столкновения ядра и других физических объектов в игре.

Столкновения (Collisions) играют важную роль в компьютерных играх. Это, пожалуй, не конкретная механика, а объемный пласт взаимодействия между игровыми объектами. Система столкновений Unity работает за счет коллайдеров (Colliders). «Коллайдер - это компонент, который представляет собой невидимые "границы" объекта. Часто они совпадают с формой самого объекта (как в реальном мире), хотя это и не обязательно. Физический движок Unity поддерживает разнообразные формы коллайдеров:

– boxcollider – форма прямоугольного параллелепипеда;

– spherecollider – сфера;

– capsulecollider – капсула (математически, сфероид или эллипсоид вращения);

– meshcollider – кастомная форма 3Д-меша, соответствующая форме

самого меша;

– boxcollider2D – прямоугольник;

– circlecollider2D – круг;

– polygoncollider2D – кастомная форма 2Д-спрайта, повторяющая форму самого спрайта.

В данном проекте будут использоваться сферический коллайдер (SphereCollider) и прямоугольный коллайдер (BoxCollider2D). Сферический коллайдер будет использоваться на снарядах и на противниках. А прямоугольный коллайдер будет использоваться на деревянных укрытий, земли, стенок. «Необходимо рассмотреть, как выглядит математика сферического коллайдера» [18]. «Из аналитической геометрии известно, что если у есть сфера с центром (x_0, y_0, z_0) и радиус r, то все точки (x, y, z) находящиеся на этой сфере можно описать, как: » [19]

$$(x - x_0)^2+(y - y_0)^2+(z - z_0)^2 = r^2, \quad (14)$$

где: (x_0, y_0, z_0) – центр сферы,

r – радиус.

а запись в векторной форме этого выражения будет выглядеть:

$$||P - C||^2 = r^2, \quad (15)$$

где: P - это точка на сфере,

C - это точка центра сферы,

r – радиус.

Что эквивалентно:

$$dot(P - C, P - C) = r^2, \quad (16)$$

где: функция dot - это скалярное произведение и для него в Unity3d есть функция Vector3.Dot. Уравнение же прямой выглядит, как:

$$p(t) = o + t * d, \quad (17)$$

где: p(t) – это точка на прямой,

o – это начало луча,

d – это направление луча. При существовании пересечения окружности и луча.

$$P = p(t), \quad (18)$$

И подставив всё в исходное уравнение окружности и раскрыв все скобки получим уравнение вида:

$$t^2 * dot(d, d) + 2t * dot(d, o - C) + dot(o - C, o - C) - r^2 = 0, \quad (19)$$

где: t – точка на прямой,

C – это центр.

что в свою очередь является стандартным квадратным уравнением вида:

$$a * t^2 + b * t + c = 0, \quad (20)$$

где: a = dot(d, d) – координата x,

b = 2 * dot(d, o – c) – координата y,

c = doc(o – c, o – c) – r^2 – координата z.

или: a = dot(direction, direction) – координата x,

b = 2 * dot(direction, origin – center) – координата y,

c = doc(origin – center, origin – center) – $radius^2$ – координата z.

и приходим к дискриминанту, что t так же равно

$$t = (-b + - \sqrt{(b^2 - 4ac)}) / 2a, \quad (21)$$

Триггеры.

«Триггеры - это те же коллайдеры» [21]. «Но триггеры "физически прозрачны". Другими словами, объекты, помеченные как триггеры*, не являются твердыми телами и пропускают любое другое тело сквозь себя» [20]. «Триггеры в основном используют как некие зоны, или области, попадание в которые влечет за собой какие-то последствия» [24].

Например, начало анимации, изменение переменных, уничтожение объектов и т.д. Другое распространенное использование триггеров - это обнаружение столкновений/входа или выхода объектов из определенной зоны. «В целом, триггеры предоставляют более гибкий и мощный способ управления поведением объектов внутри игры или приложения» [22].

В этом проекте триггерами будут стенки, которые стоят где заканчивается карта и видимость игрока. А также триггером будет пушка из

которой будут вылетать снаряды.

В главе 2 (Создание математической и физической основы игровых механик и алгоритмов действий объектов), в пункте 2.1 (Полёт снаряда. Физические свойства), были расписаны формулы, которые необходимы для создания игровой физики (формула массы тела, скорость тела, закон всемирного тяготения или гравитация, сила трения).

В пункте 2.2 (Объекты в игре и их физические свойства), было рассмотрено система столкновения с физическими объектами и противниками. Также были описаны разновидность разнообразных форм коллайдеров (boxcollider, spherecollider, capsulecollider, meshcollider, boxcollider2D, circlecollider2D, polygoncollider2D). Подробно был расписан сферический коллайдер, а именно его математика. И наконец было описано что такое триггер, где он применяется и физические свойства.

На этом вторая глава заершена.

Глава 3 Программная реализация игры

3.1 Концептуальная модель

Концептуальная модель необходима для определения структурных элементов предметной области и обозначения отношений между ними. В результате концептуальная модель представляет собой логическую структуру рассматриваемой области.

Концептуальная модель - это первый и самый важный шаг в создании игрового проекта. на этом этапе гейм дизайнер создает и описывает свои идеи в специальном документе. Издание должно было получить документ, описывающий игру как конечный продукт, а также первоначальное исследование всех элементов игры. Далее документ используется тестировщиками, производителями, дизайнерами, программистами и инвесторами.

Поскольку данный игровой проект создается начинающим разработчиком, имеет смысл описывать концепцию только в виде тезисов и в дальнейшем изменять их по мере решения практических задач.

При описании концептуальной модели игрового проекта были сформулированы следующие тезисы:

Жанр: Головоломка, артиллерия.

Режим: Одиночная.

Графика: Векторная.

Пространство: 2D.

Вид камеры: Вид с боку.

Цель: Уничтожить монстров точными выстрелами.

Мир: Мир состоит из плоской зелёной земли и красивых гор на фоне.

Развертывание: Размещение на сайте под доменом .io.(Play Store)

Средство: Unity.

Язык программирования: C#.

Таким образом, в этих тезисах были определены все детали игры, описаны игровые объекты и определены средства проектирования, которые не были необходимы для проекта.

3.2 Отрисовка дизайна

На этом этапе необходимо реализовать внешний вид игры, представить элементы пользовательского интерфейса и спрайты в векторе и определить вид с камеры.

Графика в 2D играх может иметь 3 вида камеры:

- вид сверху;
- вид сбоку;
- ортографическая проекция.

Вид сбоку: такой обзор также называют двумерным. Такая камера позволяет увидеть всё, что происходит в игре. Это важно для файтингов и платформеров, потому что в них игрок должен понять, откуда движется враг, как лучше добраться из точки А в точку Б и так далее. (Рисунок 10)

Рисунок 10 — Вид сбоку.

Расширенный взгляд сверху: обеспечивает более полное понимание ситуации. Он является важным элементом в играх с большим числом оппонентов, где непрерывно необходимо быть настороже. Например, во многих играх в стиле Diablo количества непиятелей могут исчисляться десятками и сотнями. (Рисунок 11).

Рисунок 11 — Вид сверху.

В ортографических проекциях направление лучей проецирования перпендикулярно плоскости проецирования, где сами лучи направлены вдоль осей X, Y, Z. Благодаря этому получается плоское изображение объекта в трех направлениях. На рисунке 12 представлено различие ортографической проекции от 3D.

Рисунок 12 - Ортографическая проекция

Выбор пал на вид сбоку. Так как этого будет более чем достаточно для этого проекта.

С помощью программы Adobe Illustrator 2017 были созданы векторные изображения с использованием примитивов. Как упоминалось ранее, векторное изображение можно легко масштабировать без потери качества, что позволяет сначала создавать изображения, а затем настраивать их в соответствии с желаемым разрешением. Разрешение каждого изображения в пикселях может быть любым. Все изображения сохраняются в растровом формате PNG, поскольку движок Unity не принимает векторные форматы. Поэтому рекомендуется сразу определять размеры изображений, чтобы избежать потери качества при передаче изображений в среду Unity. Графические элементы в компьютерной графике называются спрайтами. На рисунке 13 показаны игровые спрайты.

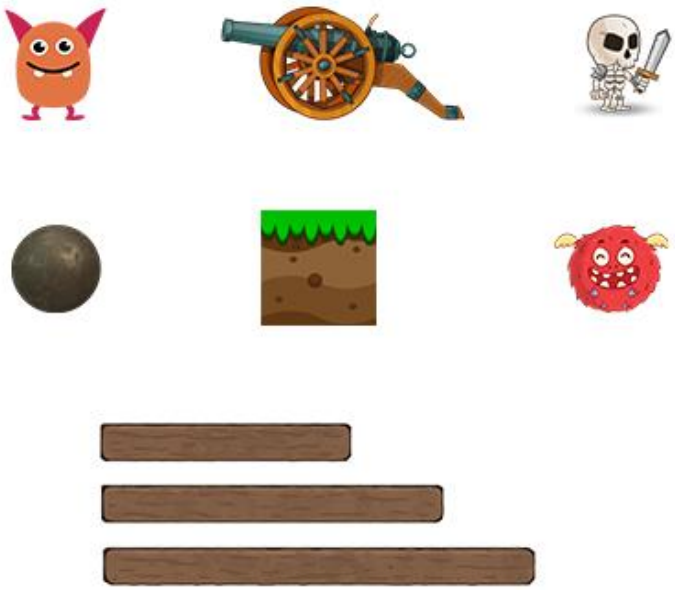

Рисунок 13 — Игровые спрайты.

Затем были созданы элементы интерфейса. Они необходимы для взаимодействия пользователей с оборудованием компьютера. Это взаимодействие осуществляется с помощью кнопок, текстовых полей ввода и

других графических элементов. на рисунке 14 показаны созданные элементы интерфейса - это различные панели управления для отображения информации на экране и кнопки.

Рисунок 14 — Элементы интерфейса.

В 2D играх камера обычно расположена с боку, поэтому необходимо создать спрайт, который будет отвечать за задний фон или так называемый background. У него нет каких-то физический свойств он просто нужен для красоты и приятной атмосферы в игре. (Рисунок 15)

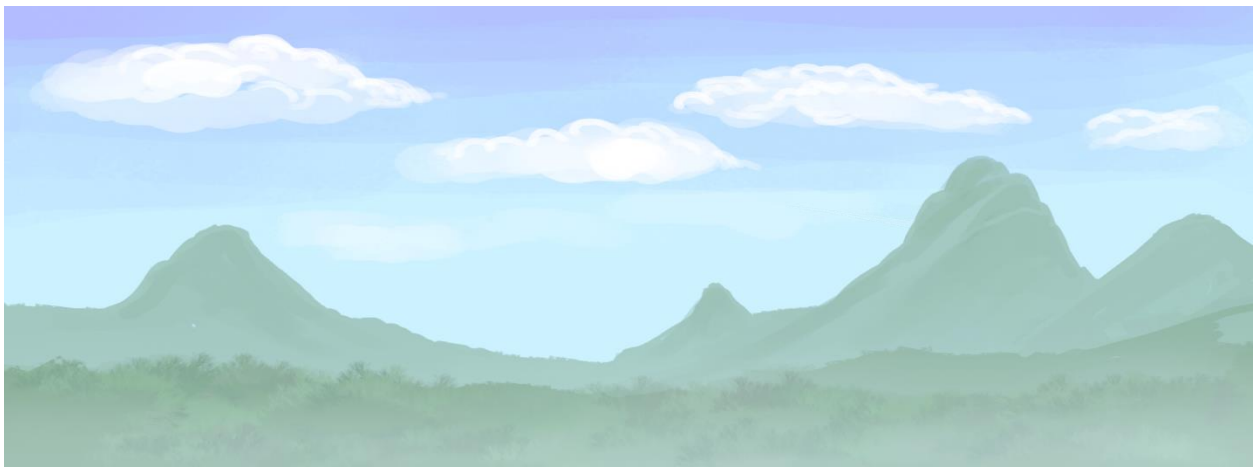

Рисунок 15 — Задний фон

3.3 Создание 2D макета в среде Unity

3.3.1 Основы Unity

В области компьютерных разработок пользовательский интерфейс (UI) представляет собой информационную среду, которая отображается на экране компьютера и с которой взаимодействует пользователь. Для создания UI ранее требовалось написание большого числа скриптов, однако сейчас существует объектная модель, которая позволяет создавать интерфейсы с помощью визуализации. Для создания любого элемента UI в Unity необходимо нажать правую кнопку мыши в окне иерархии, выбрать UI и соответствующий графический элемент. На рисунке 16 представлены виды UI в Unity.

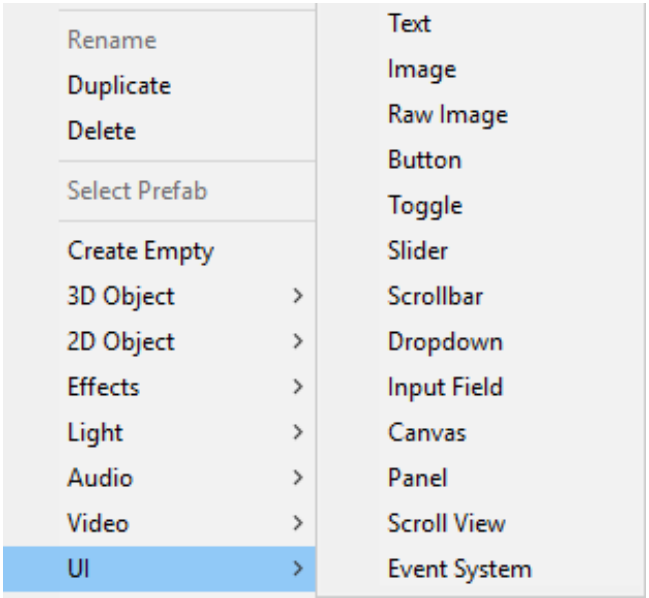

Рисунок 16 – UI в Unity

Каждый из графических элементов создается в контексте объекта Canvas. В Unity объектно-ориентированная модель создания интерфейса предусматривает, что все элементы должны быть ассоциированы с Canvas.

Canvas (Полотно)

Canvas – это компонент, который отображает графические элементы (картинки, текст, кнопки и т.д.). Canvas изображен на рисунке 17.

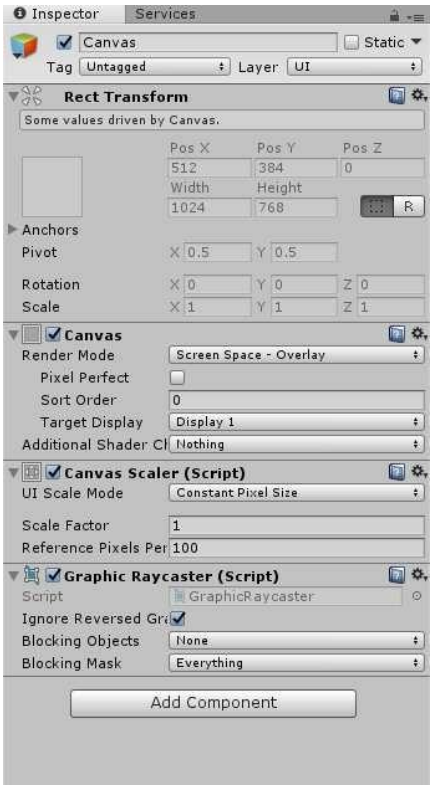

Рисунок 17 – Canvas

Canvas автоматически меняется в зависимости от разрешения экрана. Свойства Canvas изображены на рисунке 18.

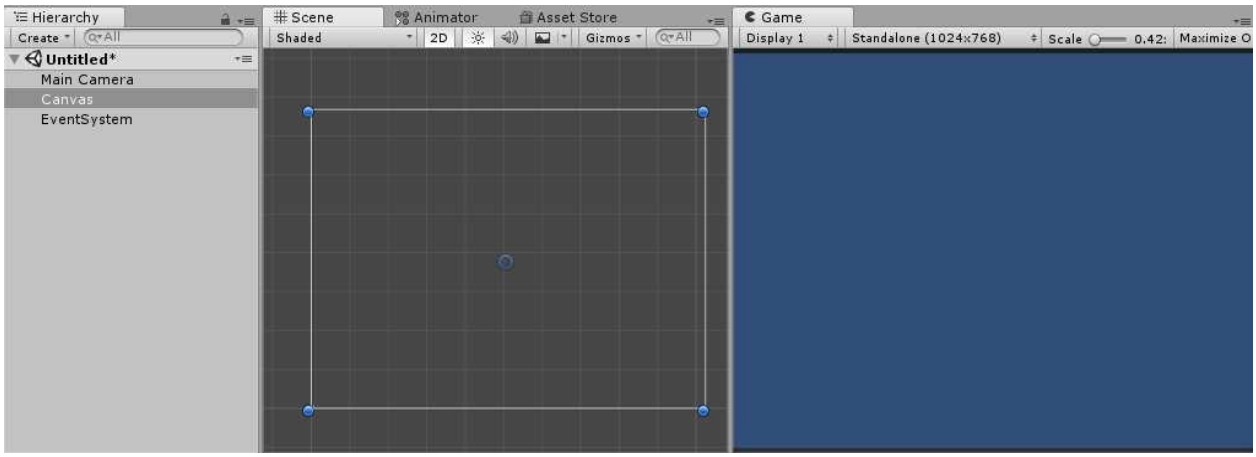

Рисунок 18 – Свойства Canvas

Режим Render Mode (рисунок 19) позволяет пользователю настроить, как графические элементы внутри Canvas будут отображаться на экране.

Режим Screen Space - Overlay позволяет отображать все графические элементы поверх игровой сцены, при этом размеры Canvas автоматически подстраиваются под размеры экрана. Режим Screen Space - Camera позволяет отображать все графические элементы с помощью камеры. Для этого нужно поместить элемент Camera в свойство Render Camera. Изменяя настройки камеры, можно влиять на внешний вид Canvas. В режиме World Space все элементы пользовательского интерфейса находятся в 3D пространстве и считаются обычными 3D объектами.

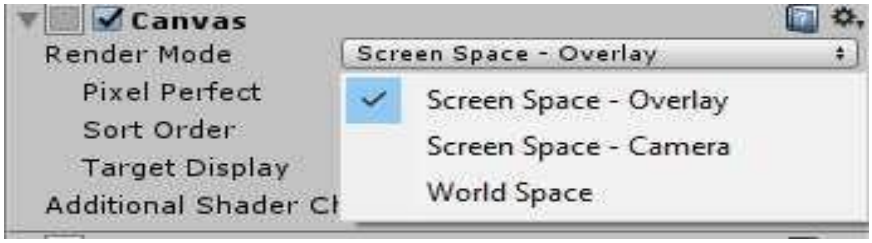

Рисунок 19 – Render Mode

Любой созданный на Canvas графический элемент можно привязать к одной из девяти точек (к углам, серединам сторон, либо к центру экрана). Для этого в панели Rect transform (рисунок 20).

Рисунок 20 – Rect Transform

На рисунке 21 показана привязка графического элемента с помощью Anchor Presets.

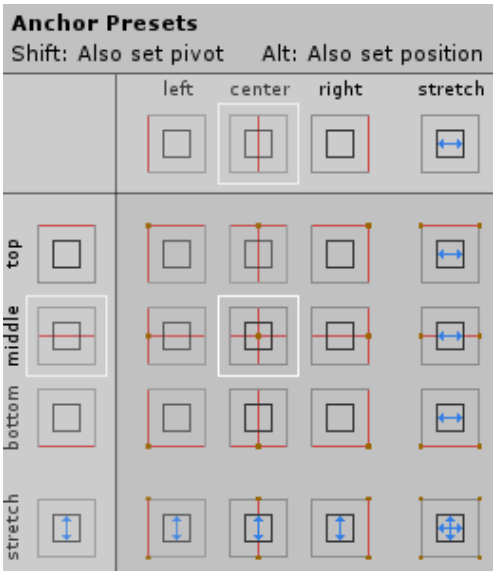

Рисунок 21 – Anchor Presets

Координаты элемента на панели React зависят от якорной точки, которая выбирается. Каждый элемент имеет Pivot свойство, которое определяет точку на нем, от которой будут отсчитываться координаты (рисунок 20).

Text - это элемент, который отображает заданный текст. Image - это элемент, позволяющий размещать изображение на экране. Размер объекта картинки по умолчанию одинаковый, однако с помощью кнопки "Set Native Size" его можно подогнать под размер спрайта.

Button - это составной элемент, содержащий дочерний элемент Text на кнопке. Нажатие на кнопку может обрабатываться с помощью свойства Button и функции-обработчика, которую необходимо передать в качестве пропса и описать в отдельном скрипте.

Toggle - сложный элемент, состоящий из нескольких изображений и текста. Единственное событие данного элемента срабатывает при изменении его состояния.

Sider - элемент управления, который настраивает различные параметры, такие как громкость звука или яркость экрана. Благодаря встроенному свойству, слайдер может закрашивать область левее ползунка.

Scrollbar - элемент, логически похожий на Slider, используется в более сложных компонентах, чтобы взаимодействие со Scrollbar имело какой-то

эффект.

Dropdown - элемент, представляющий собой раскрывающийся список. Элементы списка задаются в настройках самого компонента.

Input Field - текстовое поле, в которое пользователь может вводить данные. Он имеет много настроек, таких как Placeholder, content type и корректировка мигающего курсора. Элемент может быть однострочным или многострочным. (Рисунок 22)

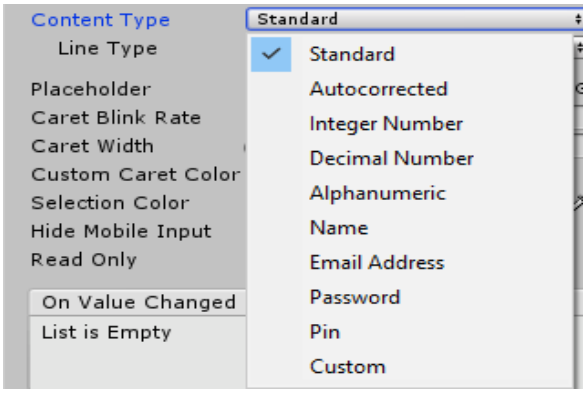

Рисунок 22 – Content type

Данный элемент интерфейса отличается от всех предыдущих элементов наличием двух событий, которые срабатывают при работе с ним. Первое событие срабатывает при любом изменении, а второе - после того, как пользователь закончил печатать.

Подложка Panel может быть использована в качестве основы для любого другого интерфейса.

Комбинированный элемент Scroll View основывается на компоненте Scroll Rect. Для его нормального функционирования необходимо наличие нескольких свойств: площади покрытия, внутри которой элемент будет прокручивать весь контент, контента и Scrollbar, который будет прокручивать элементы.

3.3.2 Создание пользовательского интерфейса

Создадим сцены: Menu и Main. Menu будет содержать 3 кнопки: Play – играть, Options – настройки, Exit – выход. Main – основная сцена, в которой будут происходить все игровые действия.

Переставим созданные сцены в Scenes in build, как на рисунке 23.

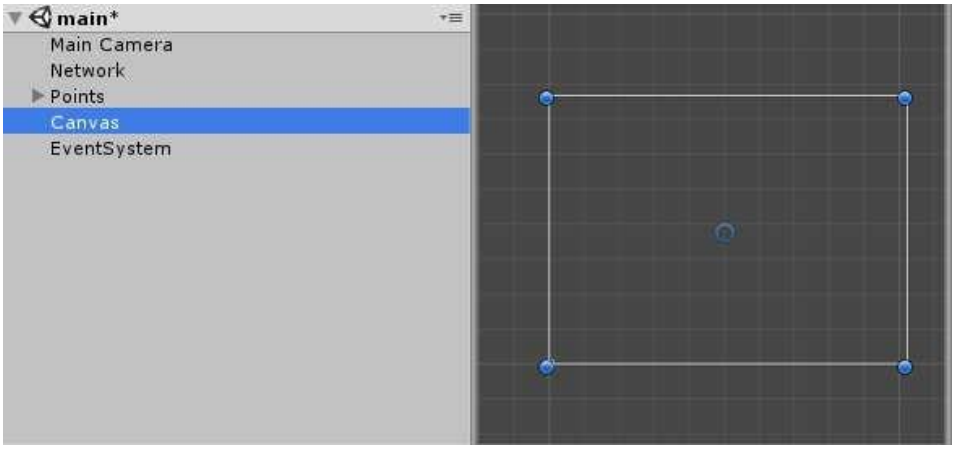

Рисунок 23 – Параметры сборки

Переходим к процессу формирования сцен. Необходимо открыть сцену "Menu" и создать холст "Canvas", на котором будут располагаться элементы пользовательского интерфейса (рисунок 24).

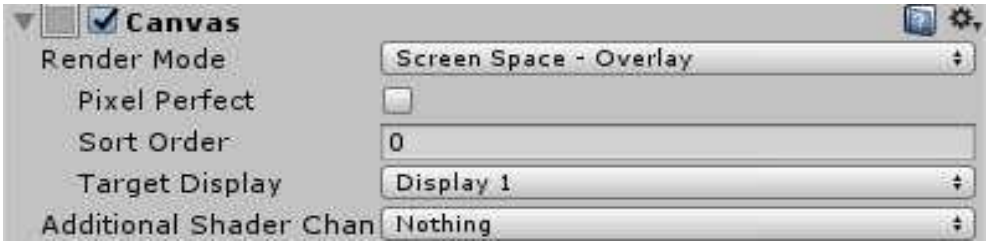

Рисунок 24 – Создание Canvas в сцене Menu

На рисунке 25 изображен инспектор canvas.

Рисунок 25 – Инспектор Canvas

Для выполнения задания необходимо использовать свойство Render mode - Screen Space - Overlay и настроить цвет панели и добавить картинку на фон. Для этого нужно установить фоновое фото на компонент Image в Инспекторе панели. Также необходимо создать папку Sprites и поместить туда игровые спрайты и элементы интерфейса (рисунок 26).

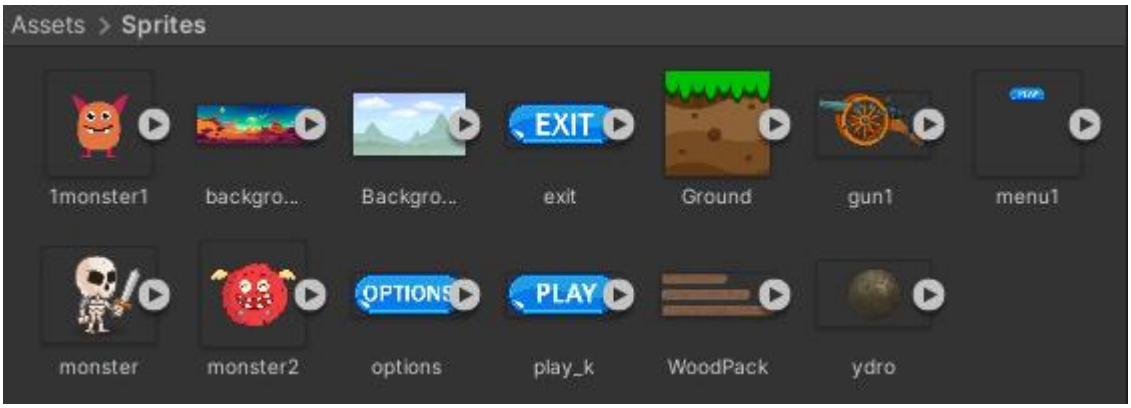

Рисунок 26 – Содержимое папки Sprites

Сцена Menu должна содержать следующее:

– название игры;
– кнопку Play, для создания сервера;
– кнопку Options для настройки игры;
– кнопку Exit для выхода из игры;
– фоновый рисунок.

Создадим дочерние UI объекты для Canvas: 1 изображения (Background), 3 кнопки (Play, Options и Exit) (рисунок 27).

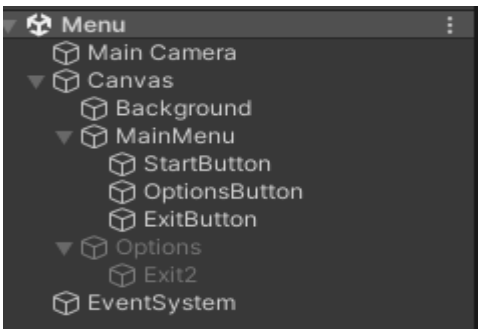

Рисунок 27 – Дочерние объекты canvas

В элемент Background поместим картинку фона, настроим разрешение и прикрепим на Canvas. Аналогично поступим с кнопками Play, Options и Exit. На рисунке 28 изображен итоговый вид сцены Menu.

Рисунок 28 – Итоговый вид сцены Menu

Аналогично сформируем сцену Main. На рисунке 29 изображен итоговый вариант сцены:

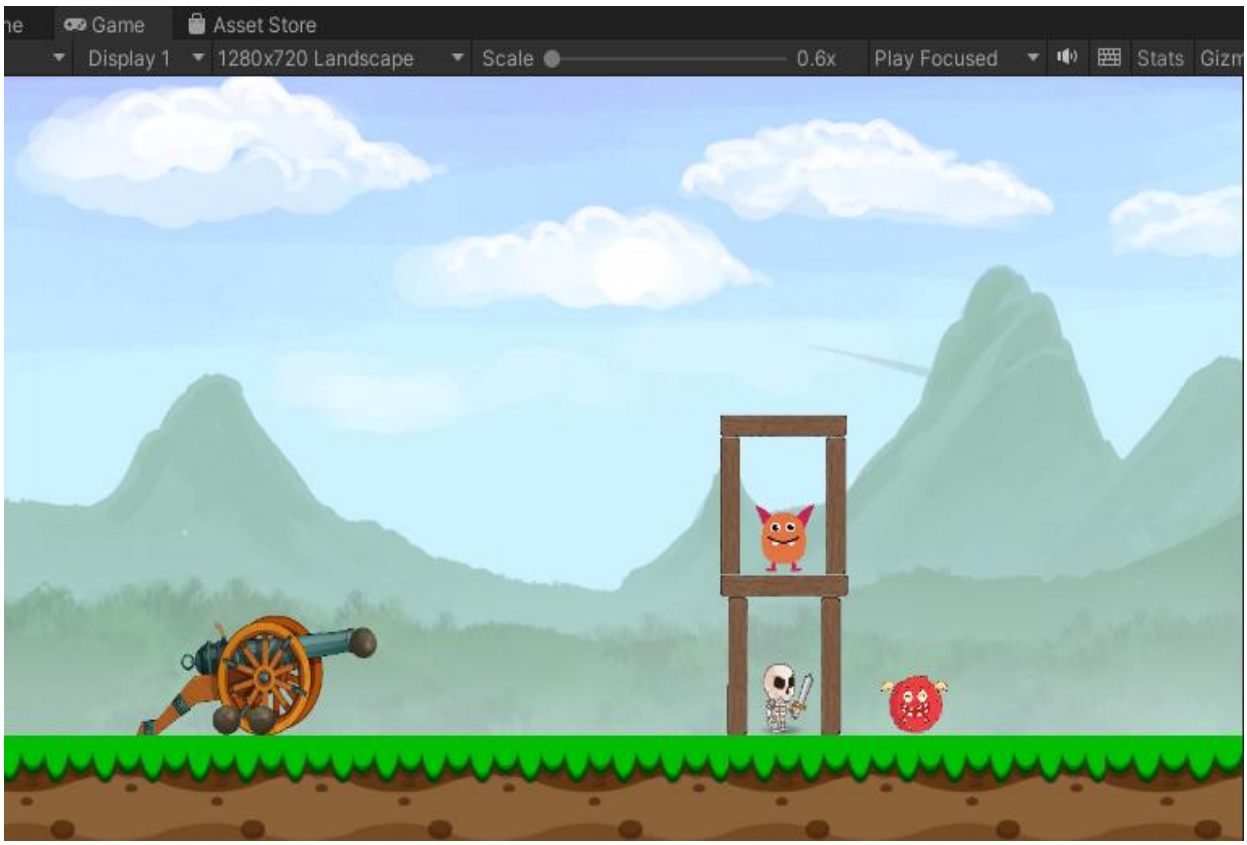

Рисунок 29 – Итоговый вид сцены Main

3.4 Скриптинг

В Unity скрипты рассматриваются в качестве объектов, отвечающих за поведение. Как и другие компоненты в Unity, они могут быть прикреплены ко бъектам. Это можно увидеть в окне инспектора.

Скрипты позволяют реализовывать любые варианты поведения объекта. Скрипт – это инструмент, с помощью которого можно выбрать поведение для любого объекта в игре. Это могут быть персонажи, объекты окружения, или, например, расширение функциональности игрового геймплея. Скрипты могут быть использованы даже для создания графических эффектов или реализации искусственного интеллекта. Каждый скрипт содержит код, написанный на языке программирования, который управляет поведением объекта.

3.4.1 Меню

Чтобы кнопки в меню функционировали корректно необходимо написать скрипт. Первая кнопка, которая запускает игру называется Play. В файле MainMenu.cs прописывается функция PlayGame (рисунок 30).

```
Ссылок: 0
public void PlayGame()
{
    SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex + 1);
}
```

Рисунок 30 – Функция PlayGame

Эта функция просто перебрасывает пользователя на сцену Level1, то есть на первый уровень. После создания скрипта необходимо его активировать, то есть прикрепить на объект(кнопку). Для этого выбирается кнопка, в инспекторе в разделе Button/on Click прикрепляется созданный ранее скрипт рисунок 31.

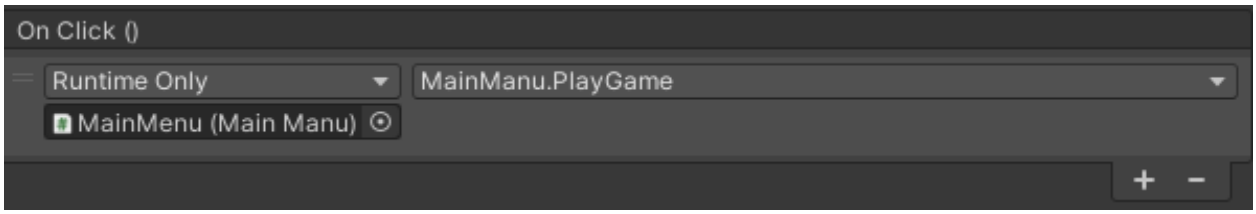

Рисунок 31 – On Click

Следующая кнопка выход или Exit. Предназначенная для того чтобы пользователь смог выйти из игры. Также в файле MainMenu.cs создаётся функция ExitGame рисунок 32. (Приложение В)

```
Ссылок: 0
public void ExitGame()
{
    Application.Quit();
    Debug.Log("Игра закрылась");

}
```

Рисунок 32 – Функция ExitGame

И точно также скрипт прикрепляется к объекту и вызывается функция при нажатии мышкой рисунок 33.

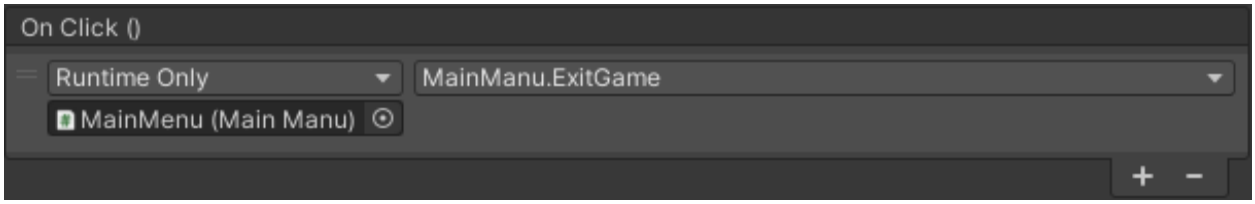

Рисунок 33 – On Click ExitGame

3.4.2 Снаряд

Теперь необходимо прописать скрипт, который будет отвечать за механику и физику игры, а именно: полёт снарядов, обработка нажатий мышкой, точка появления снаряда. Файл будет называться Bomb.cs. Сначала необходимо создать переменные (рисунок 34). Переменная BombRigid – для наделения физическим свойством снаряд. ShootRigid – для наделения физическим свойством пушку. BombPrefab – префаб для снаряда. «Префаб (prefab) — это шаблон для объекта в игровом движке Unity. С помощью префабов можно создать «образец» предмета с определенными свойствами, а потом использовать такие предметы на всей игровой сцене. Если изменить префаб, то изменятся все объекты, созданные на его основе» [6]. BombSpawnerPos – нужен для того чтобы определять текущую позицию снаряда, maxDistance – нужен для того чтобы задать ограничения по силе выстрела снаряда, isPressed – для обработки нажатий.

```
{
    [SerializeField] private Rigidbody2D BombRigid; //
    [SerializeField] public Rigidbody2D ShootRigid; //

    [SerializeField] public GameObject BombPrefab; //
    [SerializeField] public Transform BombSpawnerPos;

    [SerializeField] private float maxDistance = 2f;

    [SerializeField] private bool isPressed = false;
```

Рисунок 34 – Переменные

Следующим шагом объявляется функция Start – эта функция присваивает объектам при старте игры физические свойства (Rigidbody2D) рисунок 35.

```
    private void Start()
    {
        BombRigid = GetComponent<Rigidbody2D>();
    }
```

Рисунок 35 – Функция Start

Следующая функция Update которая вызывается раз за кадр, зависит от мощности устройства. В этой функции описан алгоритм выстрела снаряда из пушки. (Рисунок 36)

```
private void Update()
{
    if (isPressed == true) // если нажата
    {
        /*BombRigid.position = Camera.main.ScreenToWorldPoint(Input.mousePosition);*/

        Vector2 mousePos = Camera.main.ScreenToWorldPoint(Input.mousePosition);

        if(Vector2.Distance(mousePos, ShootRigid.position) > maxDistance)
        {
            BombRigid.position = ShootRigid.position + (mousePos - ShootRigid.position).normalized * maxDistance;
        }
        else
        {
            BombRigid.position = mousePos;
        }
    }
}
```

Рисунок 36 – Функция Update.

Математическое представление формулы выстрела:

PB – позиция бомбы.

TVP – позиция точки выстрела.

MP – позиция мыши.

MD – максимальная дистанция отдаления ядра (сила выстрела) равна 2.

Norm – нормализованный вектор. При нормализации вектор сохраняет то же направление, но его длина составляет 1 или 0. Если вектор слишком мал, чтобы его можно было нормализовать, будет возвращен нулевой вектор.

$$PB = TVP + (Norm(MP - TVP)) \times MD \quad (22)$$

Далее идёт сопрограмма или корутина рисунок 37.

```
IEnumerator LetGo()
{
    yield return new WaitForSeconds(0.1f);



    gameObject.GetComponent<SpringJoint2D>().enabled = false;
    this.enabled = false;
    Destroy(gameObject, 8);// объект удаляется через пять секунд после того, как был отпущен

    yield return new WaitForSeconds(3);
    if(BombPrefab != null)
    {
        BombPrefab.transform.position = BombSpawnerPos.position;
    }
    else
    {
        SceneManager.LoadScene(0); // если птички кончились, то сцена перезапускается
    }
}
}
```

Рисунок 37 – Корутина LetGo

Coroutine (корутины), или сопрограммы — это блоки кода, которые работают асинхронно, то есть по очереди. В нужный момент исполнение такого блока приостанавливается с сохранением всех его свойств, чтобы запустился другой код. Когда управление возвращается к первому блоку, он продолжает работу. В результате программа выполняет несколько функций одновременно.

Coroutine (корутины) также позволяют избежать блокировки

выполнения потока, что повышает эффективность работы программы, особенно в случаях, когда один поток должен выполнять несколько задач одновременно. Кроме того, корутины существенно упрощают написание асинхронного кода, делая его более легким для понимания и отладки. Однако, использование корутин требует более внимательного подхода к управлению памятью и синхронизации оных, чтобы избежать проблем с утечками памяти и неожиданным поведением программы.

Корутина LetGo() нужна для того чтобы, после выстрела появлялся ещё один снаряд, то есть так называемый спаунер. Также корутина обрабатывает условие того что если снаряды закончились, то уровень игры обновляется. Также обрабатывает условие чтобы снаряд после выстрела мог существовать 8 секунд, после чего он удаляется. (Приложение А)

3.4.3 Монстры

В скрипте Monster.cs описан метод столкновения ядра с монстром. (Рисунок38)

```
5   public class Monster : MonoBehaviour
6   {
7       [SerializeField] private float stamina = 5f; // выносливость
8
9       private void OnCollisionEnter2D(Collision2D collision) // метод для столкновения
0       {
1           if (collision.relativeVelocity.magnitude > stamina) // если сила удара больше, чем выносливость
2           {
3               Destroy(gameObject); // то объект удаляется
4           }
5       }
6   }
```

Рисунок 38 – Скрипт Monster

Создаётся переменная, в которой будет хранится выносливость монстра, то есть 5 единиц. Далее в функции OnCollisionEnter2D – описывается метод столкновения, а именно если сила удара больше чем выносливость монстра, то он удаляется/убивается. (Приложение Б)

На этом программирование игровых объектов и игровой логики завершено. Теперь осталось провести тестирование на ошибки и недоработки, а также настроить игровые параметры для достижения наилучшего игрового опыта. После этого игра будет готова к релизу в разных интернет магазинах и сможет привлечь множество игроков.

В главе 3 (Программная реализация игры), в пункте 3.1 была описана концептуальная модель куда входят следующие тезисы: жанр, режим, графика, пространство, вид камеры, цель, мир, развёртывание, средство, язык программирования.

В пункте 3.2 был создан дизайн игры (текстуры, спрайты).

В пункте 3.3 были созданы главная сцена и сцена меню.

В пункте 3.4 создавался скрипт на языке C# для сцены меню, также в подпункте 3.4.2 для полёта снаряда бала разработана формула и сама схема выстрела.

И наконец в подпункте 3.4.3 (монстры), был внедрён искусственный интеллект с помощью скрипта.

Заключение

В ходе выполнения выпускной квалификационной работы на тему «Моделирование и визуализация физики объектов для мобильной игры на платформе Unity» были рассмотрены физические свойства объектов в движке Unity, реализована схема метания снаряда и столкновение с монстрами. Для достижения поставленных целей в работе были рассмотрены различные аспекты физики объектов, такие как взаимодействие объектов с окружающей средой, кинематика движения, силы трения и гравитации. Были рассмотрены методы моделирования и визуализации физических законов, в том числе методы расчета физических параметров и методы отображения их на экране.

Во время подготовки, проведения исследований были изучены материалы, связанные с физическими законами и математическими расчетами, проводимыми для реализации метания снаряда.

Для проведения исследований и подготовки к созданию игры была поставлена задача определения стиля игры, выбора движка или программы, с помощью которой удавалось бы создавать игры, направление игр, проводить выбор используемой в игре графики, проводить выбор используемых камер, а также подбор программ, в которых будет создаваться графика игры.

Также была реализована программа, то есть само приложение, на платформу IOS и Android. Реализация адаптации к разным по типу устройствам, а именно, к мощности устройства и к разрешению экрана.

Цель создания игры достигнута. Проведено тестирование и исправление всяческих ошибок, которые могли бы проявиться при создании игры, а также при её использовании.

После того, как игра была реализована, можно загружать её в общее пользование на такие популярные сайты-магазины как Play Market, App Store, NashStore – (российский аналог Play Market), развивать этот проект и далее выпускать новые версии этой игры.

Список используемой литературы и используемых источников

1. Введение в геймдизайн: Основные понятия и принципы проектирования игр [Электронный ресурс]. // vc.ru. – Режим доступа: https://vc.ru/10495-gamedev-challenges (дата обращения: 09.04.2023).
2. Геймер платит за все [Электронный ресурс]. // ПРОФИЛБ. – Режим доступа: http://www.profile.ru/economics/item/117304-gejmer-platit-za-vse (дата обращения: 01.04.2023).
3. Дипломный проект [Электронный ресурс]. // GameDev. – Режим доступа: https://gamedev.ru/flame/forum/?id=185009 (дата обращения: 06.04.2023).
4. Диплом777 [Электронный ресурс]. // Высшая школа экономики. – Режим доступа: https://www.hse.ru/data/2013/06/03/1285529298/Диплом777.docx (дата обращения: 06.03.2018).
5. Жанры онлайн игр [Электронный ресурс]. // TOP MMO GAMES. Режим доступа: http://tmmog.ru/zhanry-onlajjn-igr/ (дата обращения: 01.04.2023).
6. Игра в программе Unity 3D [Электронный ресурс]. // allbest. – Режим доступа: https://knowledge.allbest.ru/programming/3c0b65635a2bc69b5c53a89521316c37.html (дата обращения: 03.04.2023).
7. Курс «Starter» по Unity 3D [Электронный ресурс]. // ITVDN Видеокурсы по программированию. – Режим доступа: https://itvdn.com/ru/video/unity-3d (дата обращения: 11.04.2023).
8. Многопользовательская игра [Электронный ресурс]. // Wikipedia. Режим доступа https://ru.wikipedia.org/wiki/ (дата обращения: 01.04.2023).
9. Программа Adobe Illustrator [Электронный ресурс]. // Меморис. – Режим доступа: http://memoirs.ru/other/page2/adobe/programma-adobe-illustrator.html (дата обращения: 01.04.2023).

10. Разработка программного обеспечения [Электронный ресурс]. // Studfiles. – Режим доступа: https://studfiles.net/preview/3021970/page:2/ (дата обращения: 01.04.2023).

11. Семь этапов создания игры: от концепта до релиза [Электронный ресурс]. // habr. – Режим доступа: https://habr.com/company/miip/blog/308286/ (дата обращения: 10.04.2023).

12. Birgigt L. Shell cartridges, 3724377, 1973.

13. C Sharp [Электронный ресурс]. Автор: Андерс Хейлсберг // Wikipedia. – Режим доступа: https://ru.wikipedia.org/wiki/C_Sharp (дата обращения: 01.04.2023).

14. Cels A., Simonin R. Improvement in or relating to ballistic devices and projectiles, 1273208, 1972.

15. Featured Unity Books [Editor's Picks] Joe Hocking – Режим доступа: https://geni.us/TB9J (дата обращения 06.04.2023)

16. Game Development With Unity for .NET Developers, guide. Author: Jiadong Chen. New York, 1950. 462 pp.

17. IO-игры – лучший жанр для начинающего разработчика [Электронный ресурс]. // gkh11. – Режим доступа: https://www.gkh11.ru/news/io_igry_luchshij_zhanr_dlja_nachinajushhego_razrabotchika/2016-12-07-2107 (дата обращения: 02.04.2023).

18. MonoDevelop [Электронный ресурс]. // Unity | Documentation. – Режим доступа: https://docs.unity3d.com/ru/current/Manual/MonoDevelop.html (дата обращения: 01.04.2023).

19. Unity User Manual (2018.1) [Электронный ресурс]. // Unity Documentation. – Режим доступа: https://docs.unity3d.com/Manual/index.html (дата обращения: 01.04.2023).

20. Unity C# [Электронный ресурс]. // itProger. – Режим доступа: https://itproger.com/course/unity-csharp (дата обращения: 21.04.2023).

21. Unity Game Development Cookbook: Essentials for Every Game. Author: Paris Buttfield-Addison, Jon Manning, and Tim Nugent

https://books.google.ru/books/about/Unity_Game_Development_Cookbook.html?id=q_2MDwAAQBAJ&redir_esc=y (дата обращения 10.04.2023)

22. Unity User Manual 2021.3 (LTS) [Электронный ресурс]. // Unity Documentation. – Режим доступа: https://docs.unity3d.com/Manual/index.html

23. Unity (игровой движок) [Электронный ресурс]. // Wikipedia. – Режим доступа: https://ru.wikipedia.org/wiki/Unity (дата обращения: 01.04.2023).

24. 2D Unity [Электронный ресурс]. // itProger. – Режим доступа: https://itproger.com/course/2d-unity-game (дата обращения: 11.04.2023)

Приложение А

Реализация физики снарядов

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement; // библиотека для перехода между сценами

public class Bomb : MonoBehaviour
{
   [SerializeField] private Rigidbody2D BombRigid; // Rigidbody для снаряда
   [SerializeField] public Rigidbody2D ShootRigid; // Rigidbody для пушки

   [SerializeField] public GameObject BombPrefab; // префаб для снаряда
   [SerializeField] public Transform BombSpawnerPos; // позиция снаряда

   [SerializeField] private float maxDistance = 2f; // максимальный радиус окружности, куда можно увести снаряд

   [SerializeField] private bool isPressed = false; // нажатие кнопки (изначально не нажата)


   private void Start()
   {
      BombRigid = GetComponent<Rigidbody2D>();
   }
   private void Update()
   {
      if (isPressed == true) // если нажата
      {
         /*BombRigid.position = Camera.main.ScreenToWorldPoint(Input.mousePosition);*/

         Vector2 mousePos = Camera.main.ScreenToWorldPoint(Input.mousePosition);

         if(Vector2.Distance(mousePos, ShootRigid.position) > maxDistance)
         {
            BombRigid.position = ShootRigid.position + (mousePos - ShootRigid.position).normalized * maxDistance;
         }
         else
         {
            BombRigid.position = mousePos;
         }
      }
   }
   private void OnMouseDown()
   {
      isPressed = true;
      BombRigid.isKinematic = true;

   }
   private void OnMouseUp()
   {
      isPressed = false;
      BombRigid.isKinematic = false;

      StartCoroutine(LetGo());
   }
   IEnumerator LetGo()
   {
      yield return new WaitForSeconds(0.1f);
```

```
        gameObject.GetComponent<SpringJoint2D>().enabled = false;
        this.enabled = false;
        Destroy(gameObject, 8);// объект удаляется через восемь секунд после того, как был отпущен

        yield return new WaitForSeconds(3);
        if(BombPrefab != null)
        {
            BombPrefab.transform.position = BombSpawnerPos.position;
        }
        else
        {
            SceneManager.LoadScene(0); // если птички кончились, то сцена перезапускается
        }
    }
}
```

Приложение Б

Скрипт для реализации физики монстров

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

public class Monster : MonoBehaviour
{
   [SerializeField] private float stamina = 5f; // выносливость

   private void OnCollisionEnter2D(Collision2D collision) // метод для столкновения
   {
      if (collision.relativeVelocity.magnitude > stamina) // если сила удара больше, чем выносливость
      {
         Destroy(gameObject); // то объект удаляется
      }
   }
}
```

Приложение В

Скрипт для реализации сцены меню

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

public class MainManu : MonoBehaviour
{
    public void PlayGame()
    {
        SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex + 1);
    }

    public void ExitGame()
    {
        Application.Quit();
```