Министерство цифрового развития, связи и массовых коммуникаций Российской Федерации
Ордена Трудового Красного Знамени
федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский технический университет связи и информатики»

Разрешаю
допустить к защите
Зав. Кафедрой

_____
_____20_____г.

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА НА ТЕМУ
__“Разработка мобильной игры в жанре RougeLike”
_____
_____

Студент: Мазур Михаил Григорьевич_____

Руководитель: Павлов С.В_____

Москва 2024г.

Министерство цифрового развития, связи и массовых коммуникаций Российской Федерации
Ордена Трудового Красного Знамени
федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский технический университет связи и информатики»

Кафедра Сетевые информационные технологии и сервисы
(название полностью)

«Утверждаю»
Зав. кафедрой
«___ ».

З А Д А Н И Е
на выпускную квалификационную работу

Студенту Мазур Михаилу Григорьевичу ЗБСТ1953 гр.

Направление (специальность): Информационные системы и технологии (09.03.02)

Форма выполнения выпускной квалификационной работы
бакалаврская работа
(Дипломный проект, дипломная работа, магистерская диссертация, бакалаврская работа)

Тема выпускной квалификационной работы:
“Разработка мобильной игры в жанре RogueLike”
Утверждена приказом ректора №
от20 г.

1. ГОСТ 2.105-95. Общие требования к текстовым документам. 2. Содержание расчетно-пояснительной записки (перечень подлежащих разработке вопросов) Введение	Объем работы в % и сроки выполнения по разделам

1. Анализ предметной области	5% до 10.12.23
2. Методы разработки	35% до 27.12.2023
3. Разработка проекта	30% до 12.01.2024
Заключение	25% до 01.02.2024
	5% до 10.02.2024
3. Вопросы конструктивных разработок	
4. Разработка вопросов по экологии и безопасности жизнедеятельности	
5. Технико-экономическое обоснование (подлежащее расчету)	
6. Перечень графического материала (с точным указанием обязательных чертежей)	
1. Диаграмма прецедентов	

7. Консультанты по ВКР (с указанием относящихся к ним разделов проекта):

_____ _____
(подпись) (ФИО)

_____ _____
(подпись) (ФИО)

8. Срок сдачи студентом законченной ВКР: _____

Дата выдачи задания: _____

Руководитель _____ Павлов С.В. КТН. Доцент кафедры СИТиС
(подпись) (ФИО)

Почасовая нагрузка _____
(штатная или почасовая)

Задание принял к исполнению _____
(подпись студента)

Примечание: Настоящее задание прилагается к законченной ВКР

Отзыв руководителя на выпускную квалификационную бакалаврскую работу

Мазур Михаила Григорьевича

«Разработка мобильной игры в жанре RogueLike"

представленную к защите по направлению 09.03.02 –

Информационные системы и технологии

Характеристика работы

Актуальность темы выпускной работы определяется тем, что в представленной бакалаврской работе рассматриваются важные структуры и методы их применения в процессе разработки мобильной игры. В данной работе проанализированы наиболее распространенные способы оптимизации кода. Разработана архитектура сохранения игровых данных. Тем самым Мазур М.Г. показал, что обладает способностью:

понимать сущность и значение информации в развитии современного информационного общества;

владеть основными методами, способами и средствами получения, хранения, переработки информации;

Бакалаврская работа Мазур М.Г. содержит не только актуальные методы разработки мобильных приложений, но и тщательный их анализ.

2.Оценка проявленных компетенций

Считаю, что стандартные компетенции у Мазур М.Г. сформированы на высоком уровне. Выпускная квалификационная работа бакалавра заслуживает оценки отлично, а Мазур М.Г. - присвоения степени «бакалавр».

3. Характеристика поведенческих аспектов деятельности студента в период работы над ВКР

При выполнении бакалаврской работы Мазур М.Г. проявил себя как инициативный и трудолюбивый студент, и сложившийся специалист в области информационных технологий.

Руководитель ВКР

Павлов С.В.

КТН. Доцент кафедры СИТиС.

Аннотация

В данной работе ведется разбор процесса разработки мобильной игры. Работа берет начало с описания сферы разработки, о том, как она зарождалась, каким образом люди пришли, к тому, чтобы развивать мобильную игровую индустрию.

Далее идет описание использовавшихся ресурсов и инструментов для разработки. Рассмотрение их плюсов и причин их выбора лично мной. По той же причине ведется рассуждение об определенных методах, объектах и структурах, использовавшихся в процессе разработки.

По завершению знакомства со сферой, в рамках которой происходила разработка, следует переход к основной части работы, в которой рассказывается непосредственно о процессе разработки, уточняются детали и сложности отдельных этапов разработки, а также подводятся итоги каждого этапа с пояснениями выбранного решения.

В конце идет небольшой обзор некоторых механик игры, её сути и возможностей, которые могут быть использованы игроком.

Содержание

Введение.

Моя ВКР является разбором процесса разработки мобильной игры, в котором будет проводиться поэтапный обзор, каждого отдельного аспекта этого пути, поэтому для начала, дабы сохранить порядок повествования, будет затронут исторический этап и речь пойдет о том, как человечество пришло к тому, что телефоны нужны не только для того, чтобы общаться на расстоянии, но и для получения исключительно интересного опыта, заключающегося в проведении своего досуга за игрой в дали от уже привычных на тот момент стационарных персональных компьютеров.

Рассмотрим более подробно само зарождение мобильных технологий.

Как только стукнул двадцать первый век, микроэлектроника сделала быстрый, резкий скачок в развитии, следствием чего стало повсеместное появление телефонов, которые так же быстро трансформировались во всем уже привычные смартфоны. Таким образом уже в нулевых можно было поиграть в игры на раскладушках “Моторола”, раздвигушках “Нокиа”, чуть позже пользоваться достоинствами системы передачи данных блютуз и так далее.

По ходу развития телефоны стали машинами, которые могли не только связывать людей на расстоянии, но и способны заниматься чем-то большим. Соответственно пустующую нишу стали в скором порядке заполнять различными приложениями. У программистов появилась новая платформа и целая веха в развитии своих знаний и навыков. Конечно же не обошлось и без так любимых всем игр, которые в то время создавались на одном, ведомым только самим создателям, энтузиазме без всяческих программ, помогающих в их создании. Тем не менее, на энтузиазме далеко не уедешь, железо смартфонов того времени оставляло желать лучшего, поэтому игры были примитивными, невзрачными и попросту ограниченными в их разнообразии, что побудило искать новые пути в упрощении создания приложений для досуга. Появление

операционных систем, таких как “IOS” и “Android”, предоставило стандарты для создания более сложных и интересных игр. Позже подтянулись целые площадки для выкладывания на них игр и соответственно скачивания их пользователем, такие как “GooglePlay” и “AppStore”.

Именно это и послужило фундаментом для мобильного гейминга.

После быстрого распространения смартфонов среди широкого круга людей, игры стали более разнообразными и интересными, множество проектов получив крупную аудиторию стали развиваться и улучшать своё качество. Сильным толчком послужило распространение социальных сетей, в которых люди могли обсуждать друг с другом интересные им игры, а с появлением многопользовательских проектов там же в них и играть. Стоит ли говорить, что с начала десятых годов мобильные игры стали повсеместной обыденностью. Индустрия мобильного гейминга стала привлекать внимание не только рядовых пользователей, но серьезные киберспортивные команды. Крупные компании стали инвестировать миллионы в этот сегмент развлечений, который стал одним из самых быстроразвивающихся на тот момент.

Как бы то ни было, данный сегмент разработки так же, как и все остальные сталкивается с трудностями и конкуренцией. Разработчики игр на смартфоны постоянно ищут новые решения и инновации, которые позволят им перетянуть долю пользователей с других платформ. И все это на фоне постоянного негодования игроков от рекламы, которая является основным источником дохода в игростроения особенно сейчас, когда этим могут заниматься огромное число индивидуальных разработчиков. При этом крупным студиям по-прежнему приходится мирится с ограниченным аппаратным обеспечением на смартфонах и продолжать держать конкуренцию с играми на стационарных персональных компьютерах.

1. Анализ предметной области

1.1 Основные платформы для смартфонов.

Операционная система “Android”, порожденная компанией Google – это гибкая и удобная платформа для разработки приложений на смартфоны, которая сочетает в себе открытость и масштабируемость. Это отражается в том, что производители могут кастомизировать интерфейс своих устройств, добавлять новые функции и даже создавать свои собственные версии “Android”. На платформе “Android” базируются тысячи мобильных игр и приложений. Так же данная платформа отличается широким спектром поддерживаемых устройств. В их число входят множество видов смартфонов планшеты и даже ноутбуки. Одним из основополагающих приятных отличительных черт платформы “Android”, является возможность устанавливать на неё приложения из сторонних источников, у ОС очень высокая адаптивность к разным видам приложений. Однако это сказывается на безопасности.

Существует так же популярная операционная система “IOS”, созданная ребятами из компании “Apple” исключительно для их собственной продукции. Основополагающим фактором для данной операционной системы является её крайне параноидальная закрытость от внешней среды. Это означает, что почти всё программное обеспечение, которое существует на данной платформе, либо было создано её собственными разработчиками, либо было тщательно проверено и допущено до работы на данной операционной системе.

Примечательной особенностью “IOS” является её общий для всех продуктов компании интерфейс, который подходит как для их смартфонов “iPhone” так и для “AppleTV”, “Apple Watch”, “MacBook” и так далее. Их собственный магазин “AppStore”, предлагает исключительно отобранную базу приложений для скачивания на их устройства и не терпит отклонений от заданной нормы, что повышает безопасность всех устройств и защищает пользователей от них самих, дабы исключить использование ими неблагоприятного программного обеспечения. Но ограничения есть

ограничения и для многих пользователей они являются бременем, с которым тем не менее приходится мириться дабы продолжать пользоваться продукцией компании “Apple” . И конечно есть ради чего мирится, “IOS” предлагает простоту и плавность использования. Опыт работы и обыденного использования данной платформы стабилен и не вызывает дискомфорта. К тому же платформа более оптимизирована чем у конкурентов, а это значит, что пользователи смогу получить опыт игры в большем числе игровых проектов.

По итогу эти две платформы являются по своей сути диаметрально противоположными по своим принципам, хоть и предоставляют одинаковые основные возможности в виде упрощенной и структурированной разработки приложений на смартфоны и прочую умную технику. Выбор одной из них зависит от предпочтений пользователя в свободе действий, стабильности, качества работы, удобства, широкого выбора возможностей, гибкости в действиях и безопасности. Для разработчика эти платформы однотипны, разница лишь в портировании игры между ними и использовании разных инструментов при программировании.

1.2 Unity: Взгляд на, используемый в процессе создания проекта, движок для разработки игр.

Сердце и почки каждого проекта – это движок, на котором он собирается и поддерживается, до тех пор, пока волна хайпа держит его на плаву, поэтому стоит уделить отдельно внимание его обзору. Мой выбор пал на детище разработчиков из Unity Technologies под названием Unity.

Unity - это интегрированная среда разработки (IDE) и кроссплатформенный движок, созданный Unity Technologies. Он является одним из наиболее популярных инструментов для создания обычных игр, игр виртуальной реальности (VR), а также приложений и симуляторов для различных платформ.

Достоинства:

- Обучение и Лицензирование:
    - Unity предоставляет бесплатную версию с некоторыми ограничениями, что делает его доступным для новичков. Для более продвинутых проектов и коммерческого использования доступны различные планы лицензирования, включая Unity Pro.
- Язык программирования:
    - Unity поддерживает несколько языков программирования. C# является основным языком, который широко используется разработчиками в Unity, благодаря своей простоте и эффективности, его я и использую. Однако, Unity также поддерживает JavaScript и Boo, и IronPython, и IronRuby, и Lua, и C, и C++, и Rust.
- Asset Store:
    - Unity Asset Store представляет собой маркетплейс, где разработчики могут обмениваться, продавать или бесплатно распространять свои ассеты, такие как модели, текстуры, звуки, скрипты и многое другое. Это упрощает процесс разработки, позволяя использовать готовые ресурсы и сэкономить время.
- Сообщество и Документация:

    - Unity обладает огромным и активным сообществом разработчиков, где можно найти ответы на вопросы, обменяться опытом и иногда получить поддержку. Официальная документация Unity также обширна и доступна, что делает процесс обучения и разработки более удобным.
- Кроссплатформенность:
    - В Unity просто невероятная кроссплатформенность, всем известен тот факт, что на этом движке делаются игры под множество платформ, к тому же на разных языках. Список платформ: Android, macOS, IOS, Windows, WebGL, Linux, PlayStation и так далее.
- Графика и Анимация:
    - Unity обладает мощным движком графики, который поддерживает различные техники рендеринга, включая шейдеры, освещение, тени и постобработку. Он также включает инструменты для создания 2D и 3D анимаций, что позволяет разработчикам легко воплощать свои идеи в жизнь.
- Современные Технологии:
    - Unity активно поддерживает современные технологии, такие как виртуальная реальность (VR) и дополненная реальность (AR). Это делает его отличным выбором для разработки игр и приложений в области развлечений, образования и бизнеса.
- Инструменты Разработки:
    - Unity предоставляет разработчикам множество инструментов для удобной работы. Визуальный редактор позволяет проектировать сцены без необходимости в глубоких знаниях программирования. Unity также поддерживает различные фреймворки и библиотеки для расширения функциональности.

Исходя из всего вышеперечисленного можно сделать вывод, что Unity - это мощный и гибкий инструмент для создания разнообразных приложений и игр. С его помощью разработчики получают доступ к широкому спектру возможностей и инструментов, что делает Unity востребованным выбором в индустрии разработки игр и программного обеспечения, который в полной мере справляется со всеми потребностями в разработке моего проекта. Движок Unity имеет богатый функционал и удобный интерфейс, часть которого можно увидеть на рисунке 1.

Рисунок 1 – Пример интерфейса движка Unity.

Вывод по разделу

В разделе проанализированы различные методы создания игр и выбрана система управления контентом Unity.

2. **Методы разработки**

2.1 C#: Хороший вариант для использования в разработке игр.

C# (младший брат C++ он же C++++) — это язык программирования, который разработали ребята из примечательной компании под названием Microsoft, он имеет у себя за пазухой такие полезные качества как простота, эффективность и многие другие очень полезные достоинства. Такой прекрасный язык был впервые представлен в 2000 году тут же стал популярен в сфере игростроения и программирования приложений, включая программное обеспечение для Windows, веб-приложения, мобильные приложения и игры на платформе Unity. Был выбран мной для разработки проекта за простоту освоения, надежность в компиляции, обширное множество источников и многие другие преимущества приведенные и описанные ниже.

Мой любимый язык C(шарп) является полностью объектно-ориентированным языком программирования, что означает, что весь код организован вокруг объектов, которые включают в себя данные и функции, работающие с этими данными. Это способствует более логичному и структурированному подходу к программированию. Так же одним из преимуществ C(шарп) является его простота использования. Синтаксис C(шарп) это примитивный английский язык, он легок для понимания, особенно для новичков в программировании. Это делает язык доступным для широкого круга тех, кто хочет делать игры.

C(шарп) разработали с учетом идеи "написания один раз, запуск на любой платформе". Это достигается с использованием платформы .NET, которая предоставляет среду выполнения для C(шарп) на различных платформах, включая Windows, macOS и Linux. Это сделало C(шарп) привлекательным для разработки кроссплатформенных приложений.

C(шарп) отличается сильной типизацией, что означает, что тип каждой переменной должен быть объявлен явно, если речь не идет о локальной переменной. Это уменьшает количество ошибок в процессе компиляции и повышает надежность программного кода. С появлением многозадачности и асинхронного программирования, C(шарп) стал поддерживать возможности управления параллельными задачами. Это позволяет разработчикам эффективно использовать мощности современных многоядерных процессоров.

Стоит заметить, что C(шарп) тесно интегрирован с платформой .NET и другими технологиями Microsoft. Это включает в себя доступ к библиотекам .NET, работу с Windows-службами и поддержку различных инструментов разработки, таких как Visual Studio. Этот язык активно развивается, и новые версии принесли много полезных функций и улучшений. Microsoft также активно развивает инструментарий для разработки на C(шарп), предоставляя разработчикам средства для создания современных и высокопроизводительных приложений.

Хоть C(шарп) и был создан достаточно давно, он продолжает поддерживаться и остается языком программирования, который олицетворяет современные стандарты и требования. С его помощью разработчики могут создавать разнообразные и инновационные приложения, а активное сообщество и поддержка со стороны Microsoft не дают ему скатиться.

Использование C(шарп) в Unity является одним из ключевых факторов успеха этого движка для разработки игр. Вот несколько достоинств, которые делают C(шарп) привлекательным выбором для разработки в Unity:

- имеет лаконичный и читаемый синтаксис, что делает его отличным выбором для новичков в программировании и обеспечивает быстрое вхождение в создание игр. Это важно, так как Unity активно привлекает широкую аудиторию, включая инди-разработчиков и студентов.

- является основным языком программирования для Unity, и движок предоставляет множество API и библиотек, которые облегчают работу разработчиков. Интеграция C(шарп) и Unity обеспечивает эффективное взаимодействие между кодом и редактором, упрощает процесс создания и изменения игровых объектов.
- C(шарп) и Unity позволяют легко создавать кроссплатформенные игры. Код, написанный на C(шарп), может быть скомпилирован и выполнен на различных платформах, включая Windows, macOS, iOS, Android, Linux и другие. Это обеспечивает разработчикам широкий охват аудитории.
- Unity Asset Store, богатый магазин активов, содержит множество готовых к использованию ресурсов и библиотек, написанных на C(шарп). Разработчики могут экономить время и усилия, используя готовые компоненты, такие как скрипты, эффекты и анимации, написанные на C(шарп).
- предоставляет разработчикам доступ к широким возможностям языка программирования, включая сильную типизацию, событийную модель, LINQ (Language Integrated Query), асинхронное программирование и т.д.

Работа с кодом происходила в программе Visual Studio. Эта программа имеет удобный интерфейс и её советуют сами разработчики движка Unity. Интерфейс редактора кода вы можете увидеть на рисунке 2.

Исходя из всего вышеперечисленного, C(шарп) - современный, объектно-ориентированный язык программирования, разработанный Microsoft. Он отличается чистым синтаксисом, поддержкой ООП и является кроссплатформенным благодаря платформе .NET. Язык предлагает богатые возможности стандартной библиотеки и широкий спектр инструментов для создания разнообразных приложений, от десктопных до веб-серверов и мобильных приложений.

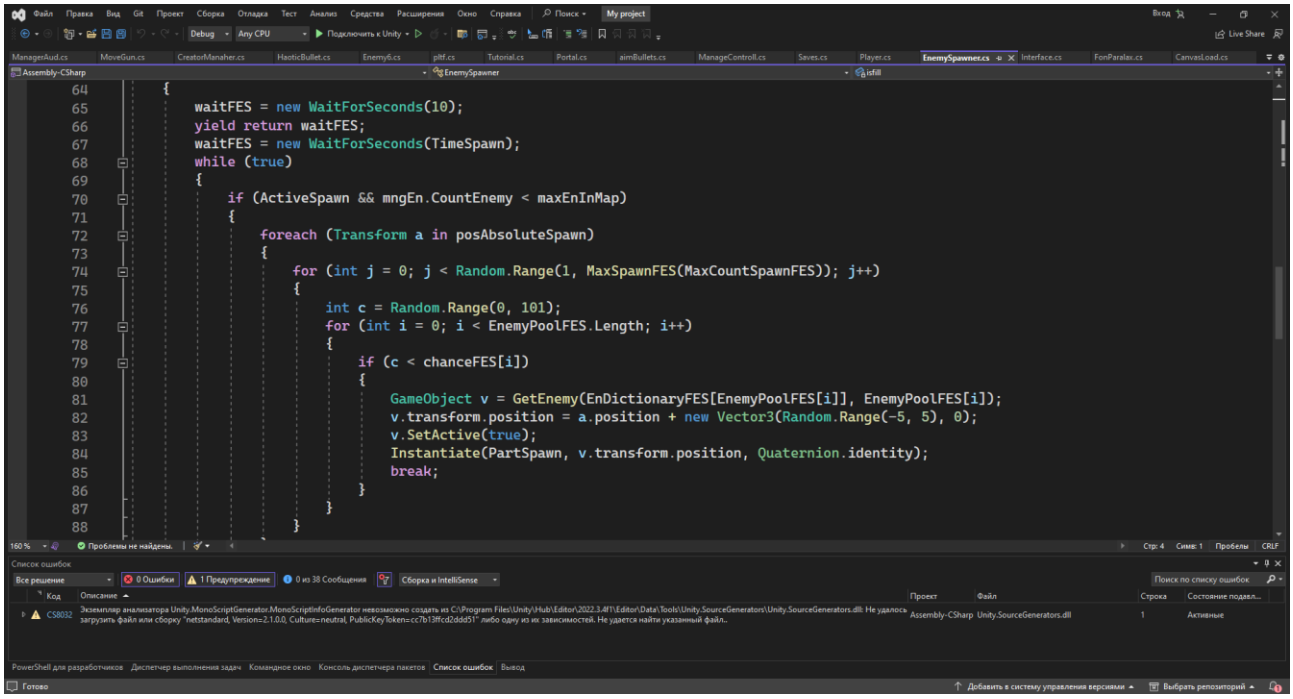


Рисунок 2 – интерфейс VisualStudio.

2.2 Использование абстрактных классов.

Абстрактные классы в языке программирования С(шарп) представляют собой особую концепцию объектно-ориентированного подхода, который позволяет создавать шаблоны для классов, от которых могут наследоваться другие классы. Они обладают уникальными свойствами, которые делают их полезными при проектировании и разработке приложений. В моей работе я использовал абстрактные классы для создания гибких шаблонов классов для различных объектов, таких как враги, различные виды пуль, виды оружия и бонусы. Каждый из этих типов объектов имеет общие характеристики, которые являются неотъемлемой частью их базовой структуры.

Функционал этих классов имеет характерную черту, в них требуется использовать ключевое слово “abstract”. Они могут содержать как абстрактные методы, так и обычные методы, свойства, поля и события. Но есть нюанс, такие классы нельзя создавать и использовать напрямую.

Классический вид такого класса:

abstract class Bullet

```
{
    // Абстрактный метод
    public abstract void InTarget();

    // Обычный метод
    public void Move()
    {
        Console.WriteLine("Moving");
    }
}
```

Абстрактные методы в абстрактных классах представляют собой функции без тела, которые помечаются ключевым словом "abstract". Они задают форму (сигнатуру) метода, который должен быть реализован в классах-наследниках. Для использования абстрактных классов их нужно унаследовать. Подклассы обязаны предоставить реализацию всех абстрактных методов, определенных в базовом абстрактном классе.

```
class ClassicBullet: Bullet
{
    // Реализация абстрактного метода
    public override void InTarget()
    {
        Console.WriteLine("Hit");
    }
}
```

Использование абстрактных классов:

Такие классы как абстрактные не могут использоваться как самостоятельная единица или объект, из используют классы наследники. Так что сам абстрактный класс как экземпляр не используется.

Преимущества использования абстрактных классов:

- Реюзабельность: такие классы как шаблон для производных классов, обеспечивают структуру и методы, которые должны быть реализованы.
- Полиморфизм: позволяет использовать объекты производных классов через ссылки на базовый абстрактный класс, что обеспечивает полиморфизм.
- Расширяемость: позволяет легко добавлять новые производные классы, не затрагивая базовую логику в абстрактном классе.

Ограничения:

- Этот класс не имеет возможности самостоятельного существования, однако может иметь методы с реализацией.
- Любые методы свойства должны быть реализованы в классах наследниках.
- Любой класс, имеющий у себя на борту метод или свойство с типом “abstract”, должен быть помечен как абстрактный.

Вот так вот абстрактные классы могут предоставить эластичный механизм для создания структур иерархий классов, давая свойства полиморфизма и расширяемости объекта.

2.3 Оптимизация кода.

Чтобы игра не сбоила и не было задержек между кадрами, нужно оптимизировать код, это важно. Эффективный и оптимизированный код обеспечивает более плавный игровой процесс, уменьшает использование ресурсов и обеспечивает лучшую производительность. Немного слов об основных возможностях оптимизации кода в Unity:

- Использование Object Pooling:

Object Pooling — это техника, при которой создаются заранее определенные объекты, которые могут быть повторно использованы вместо создания и уничтожения новых экземпляров. Это особенно полезно для часто используемых объектов, таких как взрывы, снаряды или враги. В моем проекте данный метод используется в классе ответственном за появление врагов, так же на подобной основе работает пул с пулями и в некоторых других моментах так же задействована данная система. Пример реализации на рисунке 3.

```
Ссылок: 25
public GameObject GetBullet()
{
    for (int i = 0; i < pooledbullets.Count; i++)
    {
        if (!pooledbullets[i].activeInHierarchy)
        {
            return pooledbullets[i];
        }
    }
    if (isfill)
    {
        GameObject obj = Instantiate(bulletUse);
        pooledbullets.Add(obj);
        return obj;
    }
    return null;
}
```

Рисунок 3 – пример оптимизации Object Pooling.

- Оптимизация циклов:

Стоит избегать сложных операций внутри циклов, таких как выделение памяти. Помним, что чем быстрее будет выполнен цикл, тем лучше. Если цикл выполняется очень часто, даже небольшие изменения могут иметь большое

влияние на производительность. Подобного стоит избегать, однако это не всегда является возможным. Пример подобной ошибки на рисунке 4.

```
Ссылок: 7
public override void Move()
{
    Quaternion w = transform.rotation * Quaternion.Euler(0, 0, Random.Range(-Range, Range));
    transform.rotation = Quaternion.Lerp(transform.rotation, w, SpeedAim * Time.deltaTime);
    base.Move();
}
```

Рисунок 4 – пример ошибки, выделение памяти в цикле(метод вызывается в Update())

- Batching:

Можно использовать функциональность батчинга Unity, чтобы объединять множество мелких объектов в один, что уменьшит количество вызовов отрисовки. Благодаря функционалу движка unity это делается без вмешательства в код, силами интерфейса данная функция показана на рисунке 5.

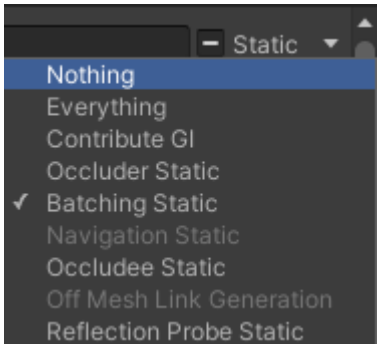


Рисунок 5 – функция батчинга в Unity

- Оптимизация Скриптов:

Использование FixedUpdate вместо Update. FixedUpdate вызывается с фиксированным интервалом времени и часто более подходит для физики.

- Кэширование ссылок:

если имеется несколько вызовов, к примеру GetComponent внутри цикла, можно сохранить ссылку на компонент в переменной, чтобы избежать избыточных вызовов. Пример кэширования на рисунке 6.

```
Сообщение Unity | Ссылок: 0
private void Awake()
{
    instance = this;
    tf = GetComponent<Transform>();
    rb = GetComponent<Rigidbody2D>();
    saves = Saves.instance;
    mnr.pl = this;
    mnr.plTf = tf;
    mnr.mg = Gun;
```

Рисунок 6 – кэширование ссылок на компоненты.

- Кеширование и Оптимизация Материалов:

Combine Meshes: если есть много объектов с одним материалом, можно объединить их в один объект для уменьшения вызовов рендеринга. Использование GPU Instancing: позволяет отрисовывать множество экземпляров одного объекта с одним вызовом отрисовки. Эти возможности в основном используются для 3D графики, поэтому в своем проекте данный способ оптимизации я не использовал, однако о нем все же важно знать.

- Оптимизация Алгоритмов:

Проведение анализа своих алгоритмов и структур данных. Например, если вы выполняете много поисков в списке, возможно, стоит рассмотреть использование словарей или множеств. Пример использования множеств на рисунке 7

```
public Sprite[] poolSprites;
public List<ModuleSpawnBullets> poolModules = new()
{   new GunAk0(),
    new ShotGun1(),
    new Aksu2(),
    new Cz3(),
    new DuableShot4(),
    new Pp5(),
    new Rifle6(),
    new RPK7(),
    new Val8(),
    new DualPistol9(),
    new Blaster10(),
    new BMG11(),
    new Famas12(),
    new FastShot13(),
    new Uzi14(),
    new MiniGun15(),
    new BlusterTwo16(),
    new HzFire17(),
    new MoreUzi18(),
    new Pistol19(),
    new RailGun20(),
    new ShotOne21(),
    new TheeGun22(),
    new ThreePistol23(),
    new TwoGun24()
};
public List<GameObject> poolBullets = new();
public List<SmallModules> smallModules = new() { new MoreDamage(), new MoreHp(), new MoreJumps(), new MoreSpeed(), new MoreMaxHp() };
```

Рисунок 7 – класс с множествами объектов.

- Отключение Компонентов:

Отключить ненужные компоненты в объектах, когда они не используются это хорошая практика. Это может уменьшить нагрузку на процессор и память. Пример отключения компонентов на рисунке 8.

```
Ссылок: 1
public void OnDeadMove()
{
    EnSpwn.ActiveSpawn = false;
    Dead = true;
    rb.Sleep();
    StarsPart.SetActive(false);
    canv.gameObject.SetActive(false);
    MenuOnDead.SetActive(true);
    BtnMenu.SetActive(false);
    saves.PlayerDEAD = Dead;
    Gun.joystick.gameObject.SetActive(false);
    UpBtn();
    anim.SetBool("Dead", true);
}
```

Рисунок 8 – Отключение компонентов

- Профилирование(сбор характеристик работы кода):

Использование инструмента профилирования, такой как Unity Profiler, для понимания узких мест в коде.

- Асинхронная Загрузка Ресурсов:

Использовать асинхронную загрузку ресурсов для улучшения производительности приложения и избегания задержек.
Явно асинхронный код в моем проекте не используется, так как в Unity асинхронные операции имеют определенные ограничения по обрабатываемым объектам, а также это банально излишне и трудоемко.

По итогу оказывается, что оптимизация кода в Unity — это скрупулезный процесс, в нем требуется внимательное тестирование на разных устройствах и условиях. Регулярная проверка и профилирование кода помогут поддерживать высокую производительность приложения.

2.4 Особенность метода Unity DontDestroyOnLoad.

Метод DontDestroyOnLoad в Unity представляет собой удобный способ сохранения объекта между сценами. Когда сцена в Unity загружается, все объекты в текущей сцене обычно уничтожаются, чтобы освободить ресурсы и предотвратить утечки памяти. Однако с использованием DontDestroyOnLoad, вы можете указать Unity не уничтожать определенный объект при загрузке новой сцены. Данный метод активно используется в проекте и так же является одно из особенностей его оптимизации, повышает удобство и работу с некоторыми важными объектами и увеличивает гибкость кода и его функционал.
Использование метода DontDestroyOnLoad:

```
public class SomeThing: MonoBehaviour
{
void Start()
        {
                // Применить DontDestroyOnLoad к текущему объекту
                DontDestroyOnLoad(gameObject);
        }
}
```

В данном примере, при запуске сцены, объект, к которому применен данный скрипт, не будет уничтожен при загрузке новых сцен. Это может быть полезно, например, для сохранения объектов, которые представляют глобальные данные или настройки.

DontDestroyOnLoad обычно используется в методе Start объекта, который нужно сохранить. Метод Start вызывается после создания объекта и до начала выполнения сцены.

Если объект, к которому применен DontDestroyOnLoad, содержит другие объекты, они также не будут уничтожены при загрузке новой сцены.
Нужно помнить, что объекты, созданные в сцене после того, как был применен DontDestroyOnLoad, по-прежнему будут уничтожены при переходе на новую сцену. DontDestroyOnLoad применяется только к объекту, к которому он был вызван.
Применение в практике:

- Музыкальный плеер: если есть музыкальный плеер, который должен проигрывать музыку на протяжении всей игры, можно применить DontDestroyOnLoad к объекту, содержащему аудио компонент.
- Глобальные настройки: если есть объект, который хранит глобальные настройки, такие как уровень звука или управление, этот объект можно сохранить между сценами.
- Меню: если есть стартовое меню, которое должно оставаться видимым на протяжении всей игры, можно использовать DontDestroyOnLoad, чтобы сохранить объект, представляющий это меню.

Таким образом можно сделать вывод, что, используя метод DontDestroyOnLoad, можно эффективно управлять объектами, которые должны оставаться активными в течение всей жизни игры и переходить между сценами Unity без необходимости повторной инициализации.

2.5 Использование типа данных ScriptableObject.

ScriptableObject в Unity - это специальный тип данных, который позволяет создавать объекты для хранения информации, без привязки к конкретному объекту в игре. Он полезен для хранения разнообразных данных, таких как настройки, текстуры, звуки и другие, и предоставляет удобный способ организации информации в проекте. Одна из ключевых особенностей ScriptableObject заключается в том, что он может существовать независимо от текущей сцены, что делает его удобным для повторного использования данных в различных частях проекта. Он также обеспечивает легкость редактирования через редактор Unity, что упрощает процесс работы с данными. Пример использования ScriptableObject может включать в себя создание объекта для хранения настроек уровня или характеристик персонажа.

Основные плюсы данного типа:

- ScriptableObject можно использовать для хранения данных. Это может включать в себя различные виды информации, такие как текстуры, звуки, настройки, списки и другие объекты данных.
- Объекты типа ScriptableObject могут существовать независимо от текущей сцены. Это означает, что они могут использоваться в различных частях проекта без необходимости повторного создания или дублирования.
- Вы можете создавать экземпляры ScriptableObject динамически через код. Это особенно удобно, когда требуется генерировать большое количество объектов данных.
- ScriptableObject легко редактировать в самом движке Unity. Можно просматривать и изменять данные объекта через инспектор Unity, что делает процесс работы с данными более удобным.

Исходя из вышеперечисленного, используя ScriptableObject можно упростить разработку проекта и легко решить задачи, которые с трудом можно было бы решить, не используя ScriptableObject.

Вид данного типа объекта в инспекторе Unity показан на рисунке 24.

Рисунок 24 – ScriptableObject в инспекторе.

Способ обозначения ScriptableObject в коде показан на рисунке 25.

```
[CreateAssetMenu(fileName = "EnemyHpManager", menuName = "Managers/EnemyHpManager")]
Скрипт Unity | Ссылок: 9
public class ManagerForEn : ScriptableObject
{
    public static ManagerForEn Instance;
    public Transform plTf;
```

Рисунок 25 – ScriptableObject в коде.

2.6 Система сохранений, использующаяся в проекте.

Система сохранений в формате JSON в комбинации с PlayerPrefs в Unity может предоставить гибкость и удобство для сохранения и загрузки данных игре. PlayerPrefs используется для хранения данных в постоянной памяти между сеансами игры, а сохранение в формате JSON может быть полезным, когда нужно сохранять сложные структуры данных или большие объемы информации.

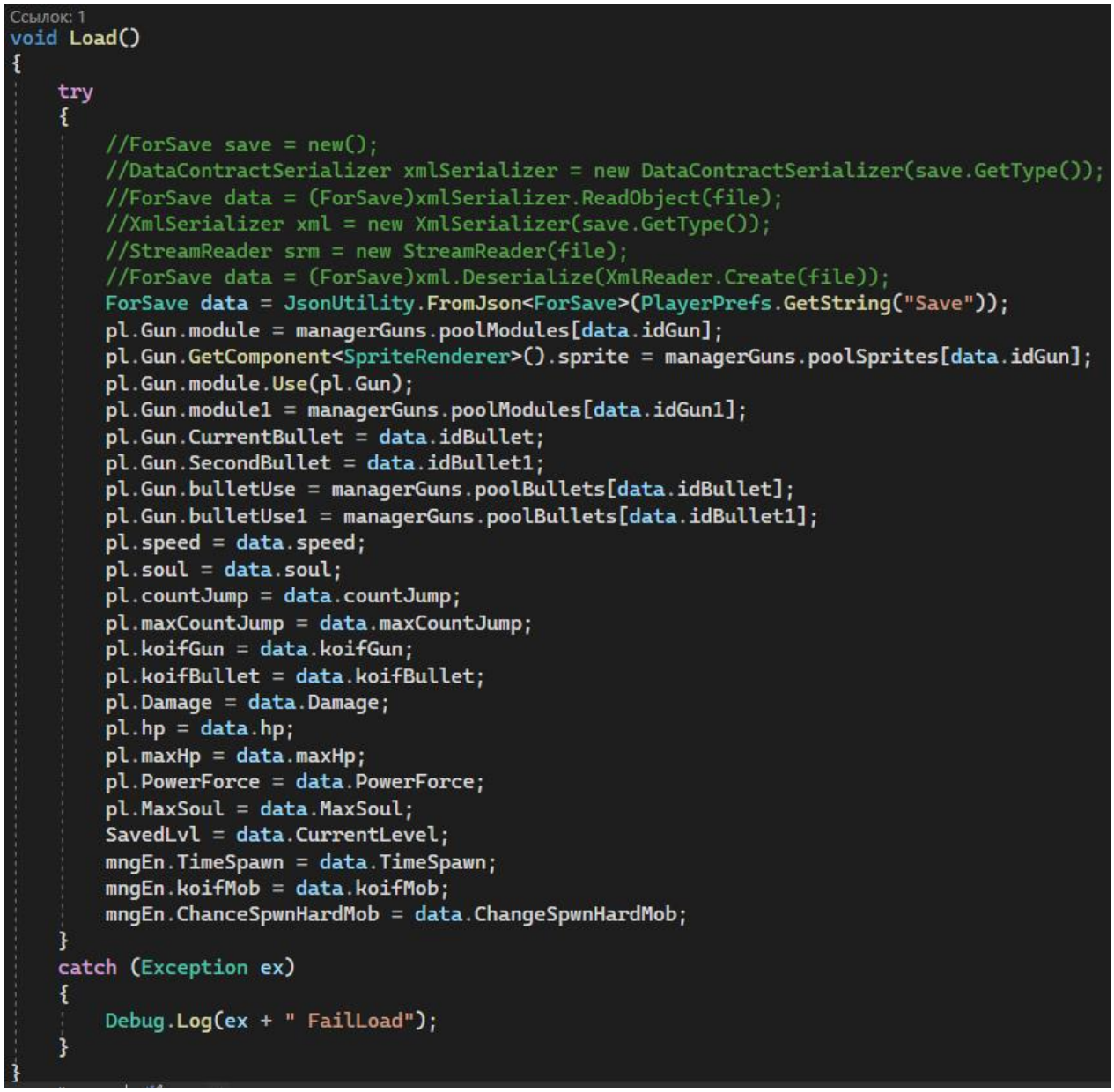
Пример реализации сохранений показан на рисунке 10

```
public void Save()
{
    try
    {
        ForSave data = new()
        {
            MaxSoul = pl.MaxSoul,
            hp = pl.hp,
            maxHp = pl.maxHp,
            soul = pl.soul,
            PowerForce = pl.PowerForce,
            koifBullet = pl.koifBullet,
            koifGun = pl.koifGun,
            Damage = pl.Damage,
            speed = pl.speed,
            countJump = pl.countJump,
            maxCountJump = pl.maxCountJump,
            idGun = pl.Gun.CurrentGun,
            idGun1 = pl.Gun.SecondGun,
            idBullet = pl.Gun.CurrentBullet,
            idBullet1 = pl.Gun.SecondBullet,
            CurrentLevel = SavedLvl,
            TimeSpawn = mngEn.TimeSpawn,
            koifMob = mngEn.koifMob,
            ChangeSpwnHardMob = mngEn.ChanceSpwnHardMob,
        };

        PlayerPrefs.SetString("Save", JsonUtility.ToJson(data));
        //FileStream file =
        //File.Create(Application.persistentDataPath + "/Save.dat");
        //DataContractSerializer bf = new DataContractSerializer(data.GetType());
        //MemoryStream streamer = new MemoryStream();
        //bf.WriteObject(streamer, data);
        //streamer.Seek(0, SeekOrigin.Begin);
        //file.Write(streamer.GetBuffer(), 0, streamer.GetBuffer().Length);
        //file.Close();
    }
    catch (Exception ex)
    {
        Debug.LogError(ex + "FailedSave");
    }
}
```

Рисунок 10 – использование системы сохранений.

Пример загрузки сохранений показан на рисунке 11



```
Ссылок: 1
void Load()
{
    try
    {
        //ForSave save = new();
        //DataContractSerializer xmlSerializer = new DataContractSerializer(save.GetType());
        //ForSave data = (ForSave)xmlSerializer.ReadObject(file);
        //XmlSerializer xml = new XmlSerializer(save.GetType());
        //StreamReader srm = new StreamReader(file);
        //ForSave data = (ForSave)xml.Deserialize(XmlReader.Create(file));
        ForSave data = JsonUtility.FromJson<ForSave>(PlayerPrefs.GetString("Save"));
        pl.Gun.module = managerGuns.poolModules[data.idGun];
        pl.Gun.GetComponent<SpriteRenderer>().sprite = managerGuns.poolSprites[data.idGun];
        pl.Gun.module.Use(pl.Gun);
        pl.Gun.module1 = managerGuns.poolModules[data.idGun1];
        pl.Gun.CurrentBullet = data.idBullet;
        pl.Gun.SecondBullet = data.idBullet1;
        pl.Gun.bulletUse = managerGuns.poolBullets[data.idBullet];
        pl.Gun.bulletUse1 = managerGuns.poolBullets[data.idBullet1];
        pl.speed = data.speed;
        pl.soul = data.soul;
        pl.countJump = data.countJump;
        pl.maxCountJump = data.maxCountJump;
        pl.koifGun = data.koifGun;
        pl.koifBullet = data.koifBullet;
        pl.Damage = data.Damage;
        pl.hp = data.hp;
        pl.maxHp = data.maxHp;
        pl.PowerForce = data.PowerForce;
        pl.MaxSoul = data.MaxSoul;
        SavedLvl = data.CurrentLevel;
        mngEn.TimeSpawn = data.TimeSpawn;
        mngEn.koifMob = data.koifMob;
        mngEn.ChanceSpwnHardMob = data.ChangeSpwnHardMob;
    }
    catch (Exception ex)
    {
        Debug.Log(ex + " FailLoad");
    }
}
```

Рисунок 11 – загрузка сохранений.

В этом примере, метод Save создает объект data, преобразует его в JSON-строку и сохраняет в PlayerPrefs под ключом "Save". Метод Load проверяет наличие сохраненных данных, затем извлекает JSON-строку из PlayerPrefs, преобразует ее обратно в объект data и использует данные.

Ранее использовался способ сохранения путем записи данных в формат xml, однако возникла ошибка и я использовал Json.

Однако можно воспользоваться и закомментированным методом, он надежнее в плане сохранности файлов, но не так удобен на практике и скорость записи под вопросом, разные источники выдают разную информацию по поводу скорости записи данных в файл с помощью рассматриваемых вариантов.

Примечания:

- Осторожность с объемом данных: PlayerPrefs может использоваться для хранения небольших объемов данных. Если данные слишком велики, можно столкнуться с ограничениями памяти или производительности.
- Шифрование данных: важно помнить, что данные, хранящиеся в PlayerPrefs, могут быть легко доступны и изменены, поэтому не рекомендуется использовать PlayerPrefs для хранения важной информации. Если нужна большая безопасность, стоит рассмотреть использование других методов хранения данных.
- Дополнительные параметры: можно использовать PlayerPrefs для хранения дополнительных параметров, таких как настройки звука или уровень сложности, а использование JSON позволяет сохранять более сложные данные, такие как массивы или списки.
- Управление версиями: при изменении структуры данных в объекте (например, добавление новых полей), нужно убедиться, что имеется механизм управления версиями данных, чтобы избежать ошибок при загрузке старых сохраненных данных.
- Тестирование: нужно внимательно тестировать систему сохранений, чтобы убедиться, что она работает корректно в различных сценариях и на разных устройствах.

Таким образом комбинирую способы предоставляемые Unity в их библиотеках мы можем добиться удобного и быстрого способа сохранения данных игрока всего за несколько строчек кода. Пусть метод и не является лучшим в плане безопасного хранения данных, однако в рамках мобильной игры этим фактором можно пренебречь в пользу удобства и скорости использования системы сохранений в разработке.

Выводы по разделу

В данном разделе проведен анализ использованного языка программирования, а также систем и решений, которые были реализованы в проекте.

3.Разработка проекта

3.1 Шаги в разработке проекта.

Разработка проекта игры на смартфон – это творческий процесс. В процессе проектирования я придерживался следующих этапов.

- Определить базу:
    - Платформа: для какой платформы будет предназначена моя игра? IOS, Android, обе?
      Мой проект собран под Android так как считаю его более распространенным ПО.
    - Целевая аудитория: Кто моя целевая аудитория? Дети, подростки, взрослые? Это поможет определить тематику и сложность игры.
      Я решил, что проект будет нацелен на детей и подростков так как на мой взгляд это основная пользовательская база мобильных игр.
- Исследование рынка:
    - Анализ конкуренции: какие существующие игры в моем жанре популярны. Что делает их успешными? Какие пробелы в рынке я могу заполнить?
      В ходе исследования было найдено ограниченное количество аналогов планируемого проекта, которые пользовались большой популярностью, однако не являлись точным отражением моей задумки.
- 3. Определить жанр:
    - Выбор жанра: определиться с жанром игры. Это может быть головоломка, аркада, казуальная, стратегия, RPG и т.д.
      Сам я выбрал жанр RogueLike, смысл которого цикличное прохождение уровне и развитие персонажа, и полное или почти полное обнуление персонажа при проигрыше.

- Смешивание жанров: рассмотрите возможность смешивания нескольких жанров для создания уникального опыта.
  В этом пункте обеспечен полет фантазии, однако на данный момент разработка нового типа жанра опасна, так как уже выделены основные категории интереса пользователей, а эксперименты с новыми жанрами все реже пользуются спросом.

- 4. Разработать уникальный геймплей:
  - Инновации: рассуждать о том, как внести инновации в геймплей. Что-то, что сделает игру выдающейся.
    В своем проекте я не опираюсь на какую-либо понравившуюся игру, однако сложно назвать геймплей уникальным, так как почти все уже было в том или ином виде показано ранее другими разработчиками.
  - Простота использования: учесть удобство управления на смартфоне. Простота использования – важный фактор успеха.
- Создать увлекательный сюжет:
  - Оригинальный сюжет: разработать интересный и оригинальный сюжет. Эмоциональная связь с игроком может сделать игру более привлекательной.
    В моем проекте он как таковой отсутствует, однако задумка игры моего жанра и не должна раскрывать сюжет.
  - Персонажи: создать запоминающихся персонажей, которые будут взаимодействовать с игроком.
    В проекте имеются 10 видов врагов, которые характерно отличаются друг от друга.
- Разобраться с монетизацией:
  - Модель монетизации: решить, как зарабатывать на своей игре. Будет ли это платной игрой, бесплатной с встроенными покупками, рекламой и т.д.

В мой проект будет интегрирована реклама, так как она способна обеспечить пассивный и долгосрочный доход.

- Дизайн Интерфейса и Графики:
    - Привлекательный дизайн: Дизайн интерфейса и графика должны быть привлекательными и соответствовать тематике игры.
      Графика, как все остальные аспекты проекта были разработаны мной. Если конкретно говорить о графике, то её я проектировал исходя из своих вкусов.
    - Адаптация игры: обеспечить адаптацию под разные размеры экранов и разрешения. Эту функцию так же можно использовать в самом редакторе.
- Проектирование Прототипа:
    - Прототипирование: создать прототип игры, чтобы понять, как будет работать геймплей. Это поможет выявить потенциальные проблемы и внести изменения.
      На этот этап зачастую тратится больше всего времени.
- Тестирование:
    - Тестирование бета-версии: запустите бета-версию игры для получения обратной связи от тестеров. Моими тестерами были друзья.
    - Итеративное улучшение: учесть отзывы и постоянно улучшать игру.
- Маркетинг и Реклама:
    - Создание маркетинговой стратегии: определить, как будет продвигаться игра. Она может включать в себя использование социальных сетей, партнерство с блоггерами, рекламу и т.д.
    - Промо-материалы: подготовить привлекательные и информативные материалы для рекламы.
- Запуск и Поддержка:

- Запуск игры: отправить игру в магазины приложений и отслеживать ее производительность.
- Поддержка и обновления: регулярно выпускать обновления, учитывая отзывы пользователей и добавляя новый контент.

Данная методика позволяет создать проект избегая издержек времени и сложностей с планированием последующих действий.

3.2 Рассмотрение аналогов разрабатываемой игры.

Есть множество способов посмотреть аналоги определенной игры: IOS AppStore и Google Play Store: просмотреть секции с рекомендациями и похожими играми, которые отображаются на странице конкретной игры. Обычно магазины приложений используют алгоритмы рекомендаций для выявления игр с схожей тематикой или геймплеем.
SteamDB позволяет просматривать игры на платформе Steam и искать их по различным параметрам, включая схожие игры. GOG.com (Good Old Games). GOG также предоставляет возможность искать игры и просматривать их по жанрам и тематике.

Reddit и похожие сайты. В соответствующих разделах Reddit, посвященных играм, можно найти темы с запросами аналогов для конкретных игр. Форумы игровых сайтов: посещайте форумы игровых сайтов и общайтесь с сообществом, чтобы узнать о схожих играх. Множество роликов на YouTube так имеют ценность для анализа чужих проектов. Многие видеоблогеры и каналы на YouTube создают обзоры и сравнения игр. Можно воспользоваться поиском чтобы изучить множество аналогов конкретной игры. Обзоры и статьи на игровых сайтах также могут содержать рекомендации похожих игр. Имеет смысл поискать рекомендации от других игроков в социальных сетях. Зачастую сообщества игр активно обсуждают схожие проекты. Многие игровые блоги и

ресурсы посвящают статьи и обзоры играм. Ну и спросить у друзей. Если у вас есть друзья, увлеченные играми, попросить их порекомендовать аналоги.

Тем не менее, нужно помнить, что в поиске аналогов важно учитывать различные аспекты, такие как жанр, геймплей, атмосфера и тематика. Комбинация этих факторов может позволить найти игру близкой к той, которая разрабатывается.

3.3 Разработка дизайна игры.

Разработка дизайна мобильной игры — это творческий процесс, включающий в себя множество аспектов, направленных на создание увлекательного и привлекательного визуального опыта для игроков. Основными этапами этого процесса являются концепция, интерфейс и визуальное оформление.

Первый шаг в разработке дизайна — это создание концепции игры. Это важный этап, на котором определяются основные идеи, стиль и общая атмосфера проекта. Дизайнеры стремятся выделить уникальные черты игры, которые сделают ее запоминающейся. Далее следует работа над интерфейсом. Здесь важно создать удобное и интуитивно понятное управление, чтобы игрок мог легко взаимодействовать с игрой. Оптимальное размещение элементов управления, понятные иконки, адаптивность к разным размерам экранов — все это учитывается при проектировании интерфейса.

Визуальное оформление игры играет ключевую роль. Дизайнеры занимаются созданием персонажей, уровней, фонов, анимаций и других визуальных элементов. Важно поддерживать единый стиль и цветовую палитру, чтобы создать гармоничное визуальное восприятие. Также стоит уделить внимание анимации, которая придает жизнь игровым объектам. Плавные и реалистичные движения способны значительно улучшить впечатление от игры. Важным аспектом является тестирование дизайна на реальных пользователях.

Получение обратной связи помогает выявить слабые места и внести необходимые изменения.

В заключении приходим к выводу, что разработка дизайна мобильной игры — это процесс, требующий внимания к деталям и умения понимать предпочтения целевой аудитории. В конечном итоге успешный дизайн создает приятное визуальное восприятие, способствует вовлечению игроков и делает игру запоминающейся.

На рисунках 12, 13 и 14 представлены некоторые атласы с текстурами, сделанными мной для использования в проекте.

Рисунок 12 – Логотип игры.

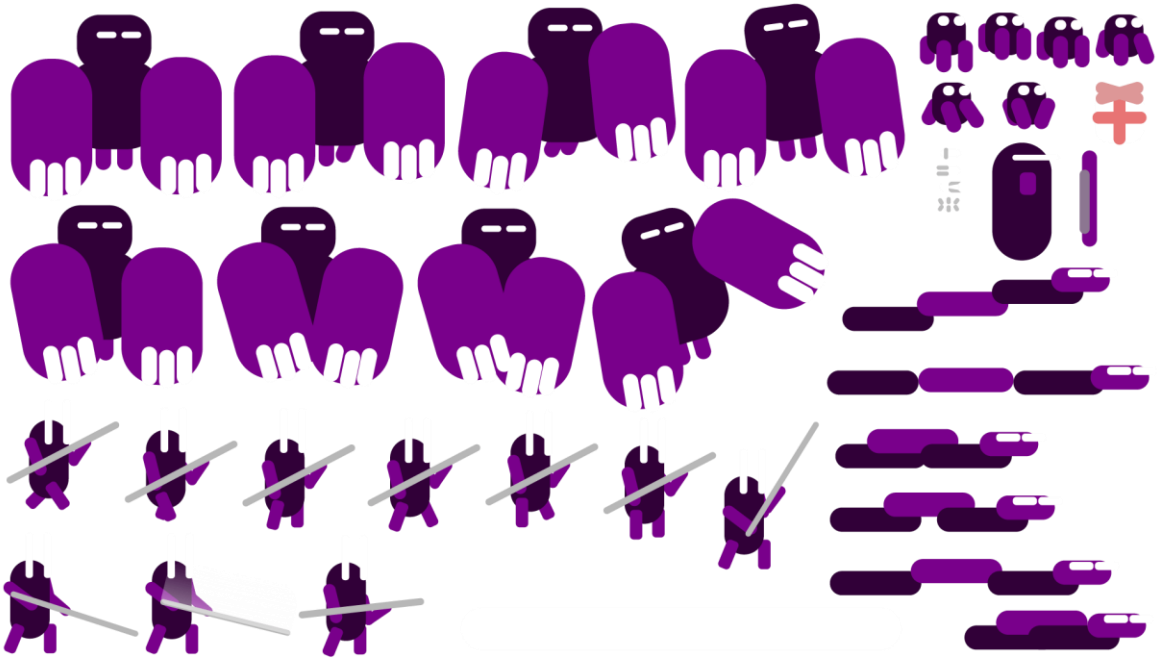


Рисунок 13 – атлас с нескольким видами врагов.

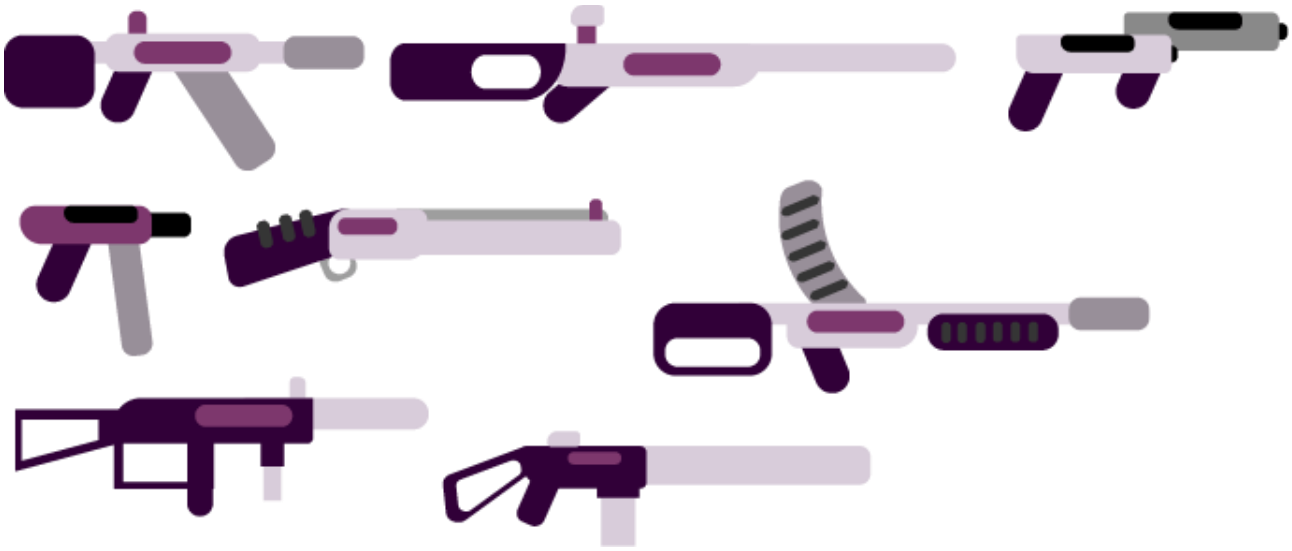


Рисунок 14 – атлас с несколькими видами оружия.

3.4 Разработка структуры кода.

Создание индивидуальной структуры кода для игры — это процесс, требующий внимания к множеству аспектов. Важно начать с четкого определения основных компонентов игры, таких как персонажи, миры, взаимодействия и игровая логика.

В начале проектирования кода необходимо выделить ключевые элементы и определить, как они будут взаимодействовать друг с другом. Это может включать в себя создание классов для персонажей, объектов и игровых сцен. Каждый класс должен быть ответственен за конкретные функциональности, чтобы код был модульным и легко поддерживаемым.

Важно также учесть принципы ООП (объектно-ориентированное программирование) и стремиться к созданию объектов, которые могут быть использованы в различных частях игры. Это позволит сделать код более гибким и уменьшить дублирование.

Организация игровой логики и управления состоянием игры также требует внимания. Использование конечных автоматов для управления различными состояниями игры может упростить код и сделать его более понятным. Кроме того, стоит разделять графический интерфейс и игровую логику для повышения читаемости и облегчения изменений в интерфейсе без воздействия на игровую механику. Помимо этого, следует активно использовать комментарии в коде для пояснения сложных частей или важных решений. Это поможет не только самим, но и другим разработчикам легче понимать структуру и цели кода.

Наконец, регулярно проводить тестирование кода, чтобы обнаруживать и исправлять возможные ошибки на ранних этапах разработки. Взаимодействие с сообществом разработчиков также может быть полезным для постоянного улучшения структуры кода.

В общем и целом, халатное отношение к данному этапу карается сотнями часами потраченными на понимание и перестройку необдуманной структуры. К

данному этапу стоит подходить с наибольшим вниманием, так как он является связующим звеном между всеми остальными этапами.

3.5 Тестирование.

Тестирование игры в Unity включает в себя проверку функциональности, производительности, стабильности и общего пользовательского опыта.

Тестирование игры на Unity должно включать в себя такие этапы как:

- Тестирование функциональности:
    - Тестирование игровых механик: проверить, что все игровые механики работают корректно. Нужно включить в процесс - тестирование управления, взаимодействия с объектами, анимации и другие игровые элементы.
    - Тестирование уровней: проверить уровни на наличие ошибок и убедитесь, что они проходимы. Проведите тестирование всех путей и альтернативных сценариев.
- Тестирование производительности:
    - Оптимизация ресурсов: убедиться, что ресурсы (модели, текстуры, звуки) оптимизированы для мобильных устройств.
    - Профилирование кода: использовать инструменты Unity Profiler и другие средства для анализа производительности игры. Постараться устранить узкие места в коде.
- Тестирование совместимости:
    - Тестирование на разных устройствах: проверить работу игры на разных устройствах с разными версиями операционных систем. Это включает в себя тестирование на различных моделях смартфонов и планшетов.
    - Тестирование разрешений: убедиться, что игра корректно отображается на различных разрешениях экранов.
- Тестирование стабильности:

    - Тестирование на утечки памяти: проверить игру на утечки памяти. Используйте Unity Profiler и другие инструменты для мониторинга использования памяти.
    - Тестирование на вылеты и ошибки: провести тестирование на вылеты и ошибки. Логировать ошибки и вылеты, чтобы быстро их идентифицировать и исправить.
- Тестирование интерфейса пользователя (UI):
    - Тестирование навигации: проверить, что пользователь может легко пользоваться меню и настройками.
- Тестирование сохранения и загрузки:
    - Тестирование системы сохранения: проверить, что сохранение и загрузка данных работают корректно. Тестирование восстановления после перезапуска игры.
- Тестирование на реальных пользователях:
    - Тестирование бета-версии: дать бета-версию игры ограниченному кругу тестеров. Собрать обратную связь для улучшения качества игры.
    - Мониторинг игровых сообществ: просматривать форумы, обсуждения и отзывы, чтобы быстро реагировать на обнаруженные проблемы.
- Ведение документации тестирования:
    - Создание тестовых случаев: создать тестовые случаи для каждого аспекта игры. Это поможет систематизировать тестирование и упростит повторное тестирование при внесении изменений.

По итогу нужно помнить, что тестирование – непрерывный процесс, который нужно постоянно поддерживать, и важно внимательно следить за обратной связью от пользователей и регулярно улучшать игру. Большую часть

информации можно получить с помощью инструмента профайлер, предоставляемого движком Unity. Пример профайлера на рисунке 15.

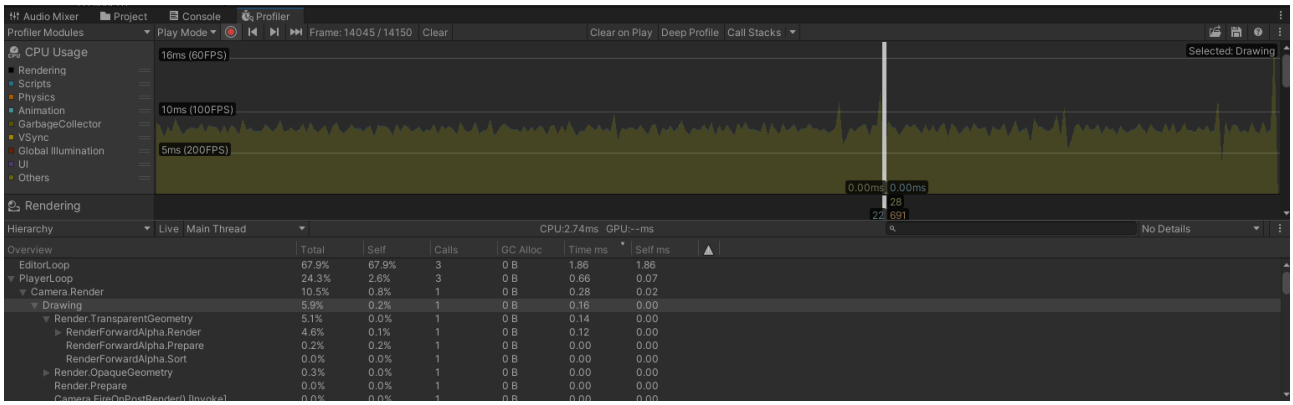


Рисунок 15 – инструмент профайлер.

3.6 Обзор игры.

Пройдя через все вышеперечисленные этапы, я спроектировал и разработал следующий проект.

Пройдемся по порядку.

Зайдя в игру, нас встретит главное меню, которое показано на рисунке 16,

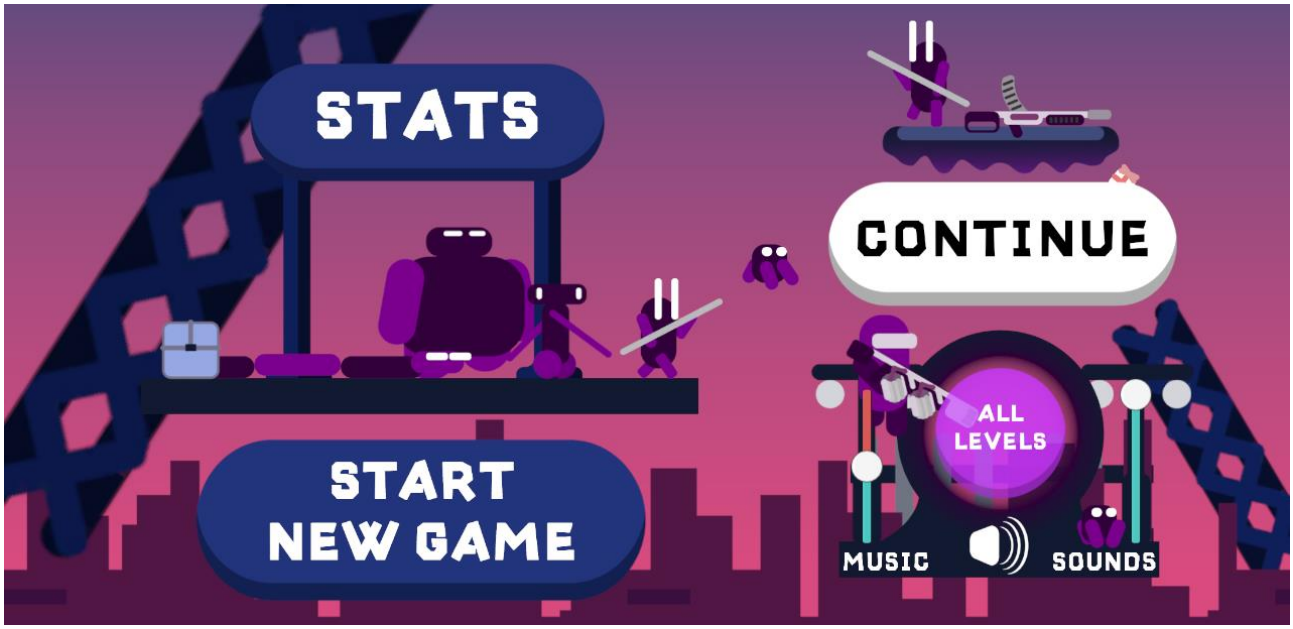


Рисунок 16 – главное меню

функционал которого будет сводится к самым важным кнопкам, а именно:

Начало новой игры

Туториал, он же уровень для обучения основным механикам.

Статистика, раздел для просмотра статистики игр.

Настройки звука

Так же в зависимости от условий меню может меняться: может появиться кнопка “Продолжить”, кнопка туториал пропадет после его прохождения, а также после прохождения всех уровней появится кнопка “Левела”, открывающая возможность начинать игру с любого уровня. Изменения показаны на рисунке 17.

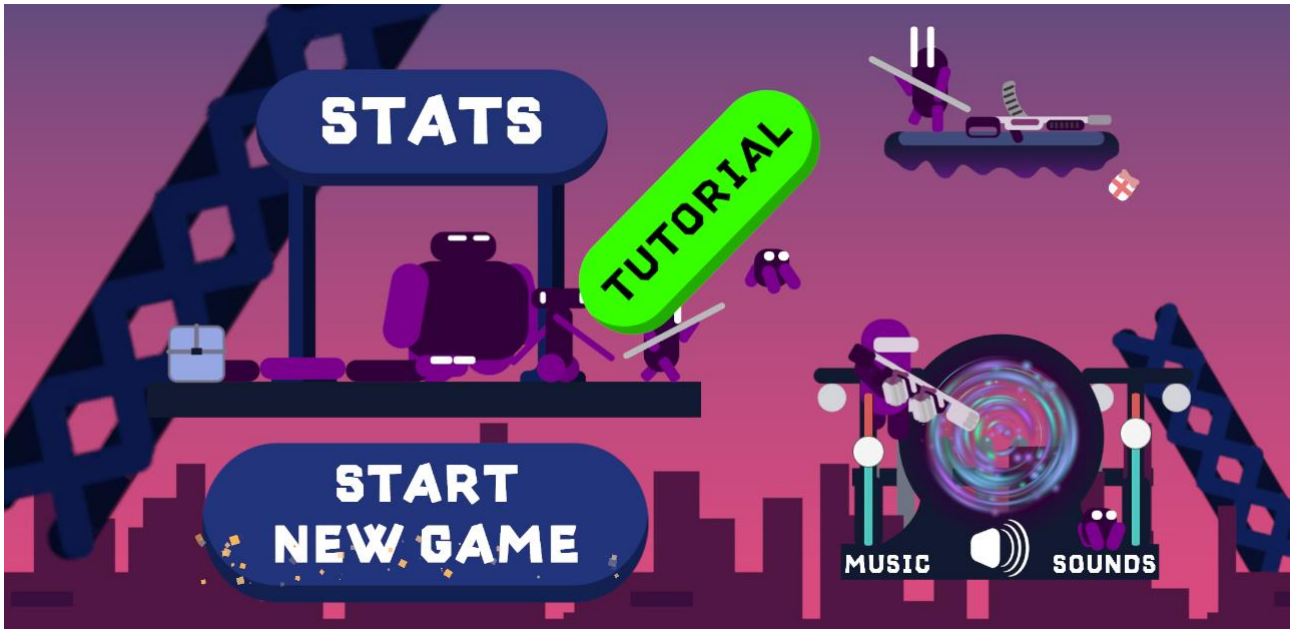


Рисунок 17 – главное меню с изменениями.

Далее затрону основные механики игры:

Стрельба с возможностью смены оружия:

Данная механика реализована следующим образом. Абстрактный класс описывает основной функционал всех оружий, затем в классах наследниках переопределяются методы и устанавливаются свои уникальные значения переменных.

```
Public abstract class ModuleSpawnBullets
    {
        public MoveGun mg; //ссылка на скрипт логики оружия
        public GameObject obj; //переменная пустого объекта класса GameObject
        public abstract int choose { get; set; } //целочисленное св-во индекс спрайта оружия
        public abstract float Timershot { get; }//св-во с не целочисл. знач.
Скорострельности
        public abstract void Use(MoveGun gun);//метод с параметром MoveGun
        public abstract void Spawn(Quaternion where, Vector2 vec);//метод с параметрами угла
и позиции
        public virtual void Clear() {Debug.Log("WasClear")}
        // виртуальный метод, который можно не инициализировать в дочерних классах
Нужен для инициализации инструкций при смене оружия, если они требуются
    }
```

Стоит заострить внимание, что ни один из классов не наследует тип MonoBehavior, основной класс, который реализуют все объекты на сцене.

Это связано с спецификой оружия, которое по своей сути является просто картинкой, классы оружия в свою очередь выполняют простую задачу создания пуль с заданным количеством, в заданной точке, с заданным углом и в общем задают определенный паттерн поведения, разные вариации которых создают впечатление разного типа оружия.

Достоинством же такой системы является её малый вес и простота добавления контента, ибо все классы лежат в одном скрипте, который является созданным мной пространством имен, в котором находятся и многие другие классы, префабы, они же заготовленные объекты в памяти, не занимают места, а доступ к каждому типу оружия максимально упрощен. Результат действия такой системы на рисунках 18 и 19. Всего в игре 25 видов оружия.

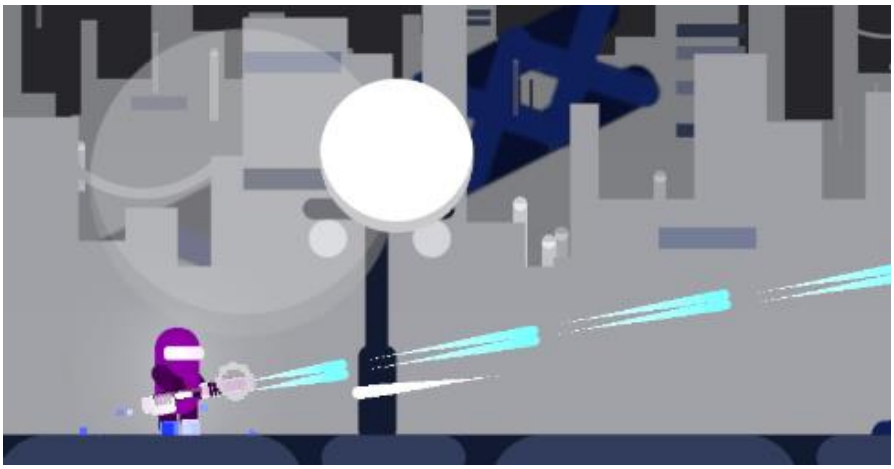


Рисунок 18 – вариант стрельбы 1.

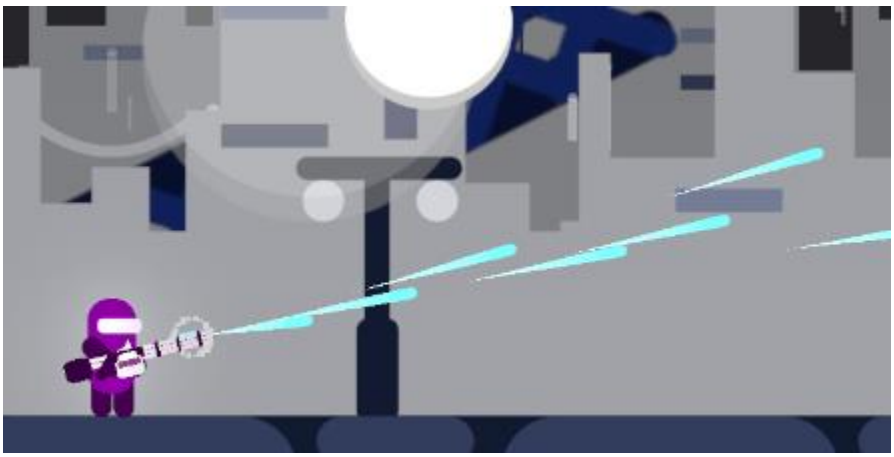


Рисунок 19 – вариант стрельбы 2.

Система врагов схожа, однако для них уже требуется отдельный скрипт с наследованием класса типа MonoBehavior, так как скрипты непосредственно взаимодействуют с физическими объектами. Так же для каждого врага создается префаб. Однако их так же легко создавать, так как базовые поля уже определены в абстрактном классе и всё что остается это присвоить им соответствующие переменные и при необходимости переопределить абстрактные методы.

Данным способом работает и система бонусов и пуль.
Разные виды пуль можно комбинировать с разными видами оружия. Пример на рисунках 20 и 21.

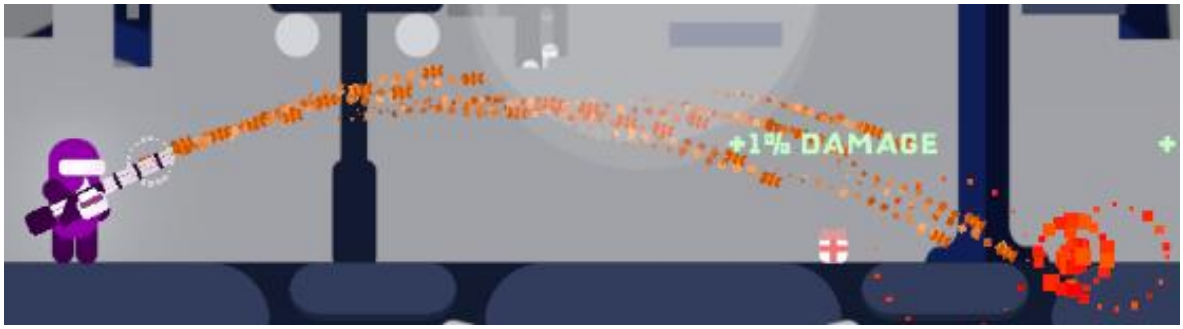


Рисунок 20 – комбинирование оружия и разных пуль 1.

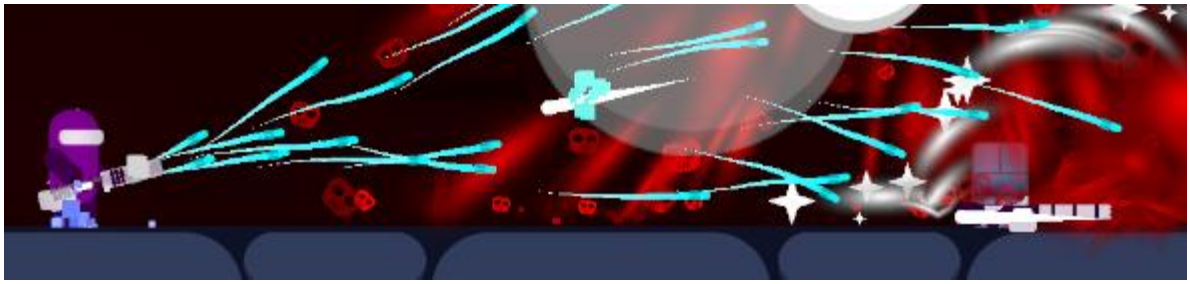


Рисунок 21 – комбинирование оружия и разных пуль 2.

Система сложности динамическая и подстраивается под игрока. С повышением количества здоровья и урона игрока растет и здоровье, и урон монстров. За это отвечает объект, который создает врагов и общий скрип управляющий их настройками.

Основная задача игрока перейти на следующий уровень путём накопления определенного количества валюты, которая дается за уничтожение врагов. В процессе, игрок усиливается бонусами, получает новые оружия и пули, от которых понемногу меняется геймплей. Длиться игра может условно бесконечно, после чего игрок может выйти в меню и просмотреть свою статистику. Скриншот одного из уровней и пример меню на Рисунке 22 и 23

Рисунок 22 – Пример игрового управления

Рисунок 23 – Пример уровня

Выводы по разделу

На данный момент проект выглядит таким образом и имеет свои недостатки, однако развитие его продолжается. В процессе разработки и поддержки игры имеется возможность добавления новых механик и контената, её баланса и так далее, поэтому уже в ближайшем будущем игра обрастет новыми чертами, которые могут кардинально изменить её геймплей.

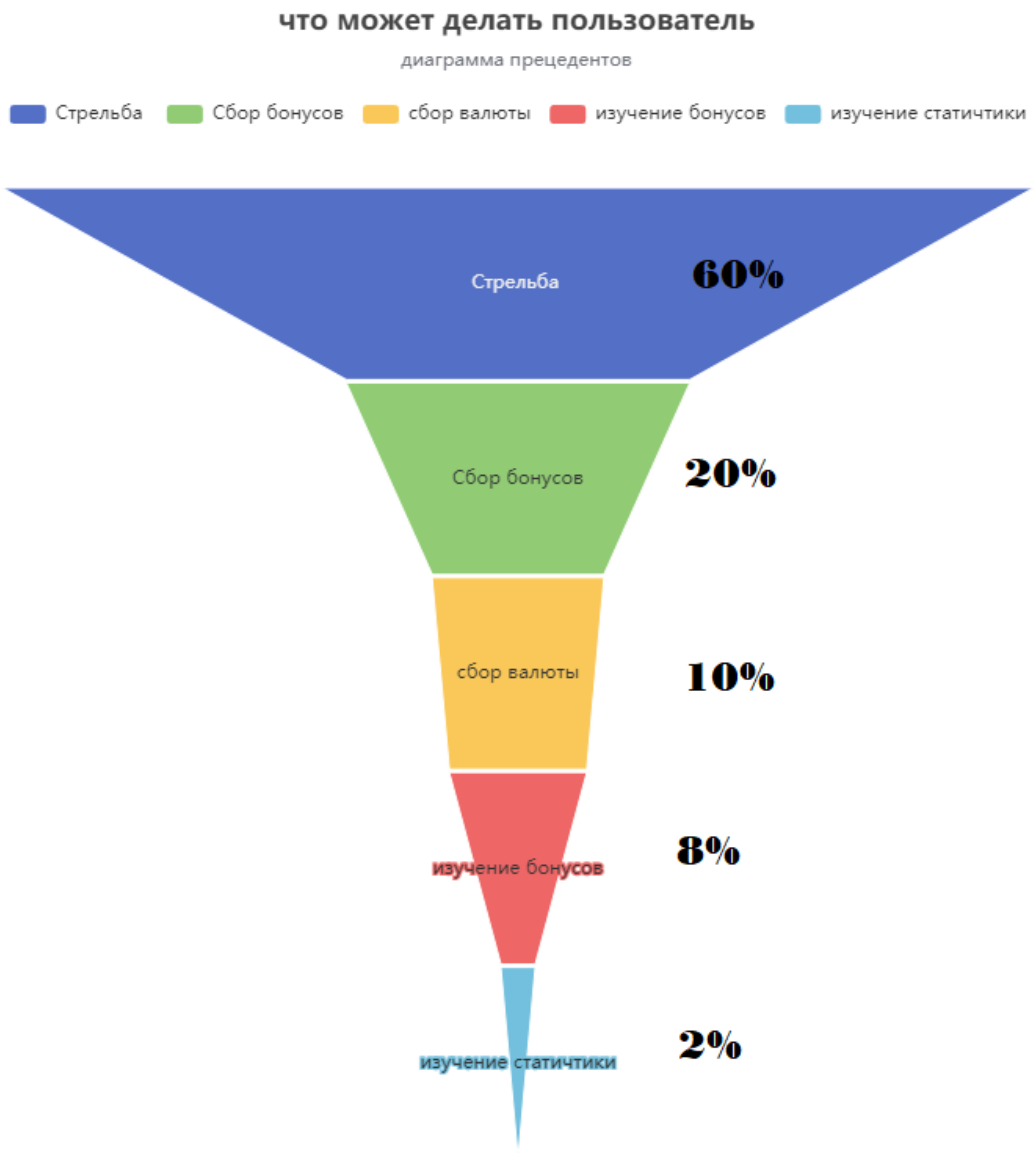


Рисунок 24 – Диаграмма занятости пользователя

Заключение.

Завершение разработки мобильной игры на Unity — это результат тщательной и трудоемкой работы разработчика. Каждый этап, начиная от проектирования и кодирования, до тестирования и оптимизации, требует внимания и профессионального подхода. Важно не только создать увлекательный геймплей, но и обеспечить стабильность приложения и его хорошую производительность.

Внимание к дизайну ведь в разработке он как вода на континенте со всем известным названием Африка, поскольку определяет внешний вид, механику, атмосферу и теплое отношение игрока к игре через его глаза. Качественный дизайн способен создать уникальный и запоминающийся игровой мир, который заставит каждого обратить его взор на себя и заинтересует игроков. Он также влияет на геймплей, делая его увлекательным, сбалансированным и максимально, как только можно понятным. Дизайн заставляет игрока влюбиться в игру глазами, а после прощупать остальными органами чувств.

Определение структуры кода влияет на весь процесс разработки, от самого проектирования идеи до момента первых и последних тестов. Общая структура позволяет понимать свой код на всей протяжённости разработки и дает некий стержень, вокруг которого крутиться всё построение. В команде правильно спроектированная структура позволяет всем и каждому взаимодействовать и работать сообща без лишних вопросов, облегчает интеграцию кода людей, которые мало кооперировались. В общем очень важный момент со структурой, это база.

Что качается тестирования проекта, то тут стоит придерживаться правила разбиения на этапы. По завершении каждого нужно проверять наработки. Этапы могут быть масштабными, такие как новые механики и маленькими, такие как наладка какого-либо локального интерфейса. По завершении основной части проекта следует общее тестирование, которое должно выявить общие недочеты в геймплее и неточности в механиках. Зачастую некоторые

механики приходится вырезать из конечного продукта, а где-то создавать новые. Естественно, тут же и проверяется производительность.

Итоговым продуктом будет совокупный труд над всеми этапами разработки, при этом работая над каждым этапом нельзя терять из виду общую картину конечной игры. Крайне неприятно идти к чему то, а по итогу увидеть совершенно не то, чего добивался всё это время.

Часто ошибкой является забрасывание проекта после его выпуска. Множество пользователей, которые продолжали играть в игру надеясь, что некоторые недочеты будут исправлены покидают игру в скором времени после игнорирования этих самых недочетов разработчиком. Поэтому важно поддерживать проект в течение ещё какого-то времени после завершения разработки.

Таким образом, успешное завершение проекта требует не только технической компетенции, но и управленческого мастерства, а также готовности к долгосрочной поддержке игры после ее выпуска.

Список использованных источников.

1. Официальная документация Unity: https://docs.unity.com/
2. Unity Learn: https://learn.unity.com/
3. Unity YouTube Channel: https://www.youtube.com/@unity
4. Brackeys YouTube Channel: https://www.youtube.com/@Brackeys
5. Unity Forum: https://forum.unity.com/forums/
6. GitHub: https://github.com/
7. Microsoft-learn: https://learn.microsoft.com/ru-ru/dotnet/csharp/
8. StackOverflow: https://stackoverflow.com/
9. LeetCode: https://leetcode.com/
10. Habr: https://habr.com/
11. Metanit: https://metanit.com/sharp/

Приложения

1. https://github.com/Mishany-M/Game