МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«ТЮМЕНСКИЙ ИНДУСТРИАЛЬНЫЙ УНИВЕРСИТЕТ»
Высшая школа цифровых технологий
Кафедра математики и прикладных информационных технологий

ДОПУСТИТЬ К ЗАЩИТЕ
Зав. кафедрой математики и прикладных информационных технологий
_____ Барбаков О. М.
«1» Июля 2024 г.

Разработка игры на движке Godot с добавлением искусственного интеллекта

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
к бакалаврской диссертации

НОРМОКОНТРОЛЕР:

_____ *Осинцева М. А.*

РУКОВОДИТЕЛЬ:
доцент, канд. пед. наук
_____ *Спирин И. С.*

РАЗРАБОТЧИК:
обучающийся группы РИСб-20-1
_____ *Никитин М. Д.*

Бакалаврская диссертация защищена с оценкой _____
Секретарь ГЭК
_____ *Арясова Д. В.*

Тюмень, 2024

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«ТЮМЕНСКИЙ ИНДУСТРИАЛЬНЫЙ УНИВЕРСИТЕТ»
Высшая школа цифровых технологий
Кафедра бизнес-информатики и математики

ДОПУСТИТЬ К ЗАЩИТЕ
Зав. кафедрой математики и прикладных информационных технологий
_____ Барбаков О. М.
«1» Июля 2024 г.

ЗАДАНИЕ
На выпускную квалификационную работу (бакалаврскую работу)

Ф.И.О. обучающегося: Никитин М. Д.
Ф.И.О. руководителя ВКР: Спирин И. С.
Тема ВКР: «Разработка игры на базе движка godot с добавлением искусственного интеллекта»
утверждена приказом по Высшей школе цифровых технологий от 19.03.2024 г.
№ 03-2000-04-03/18
Срок предоставления завершенной ВКР на кафедру «29» июня 2024 г.
Исходные данные к ВКР получены при изучении деятельности ИП «CODDY».

Содержание пояснительной записки

Наименование главы, раздела	% от объема ВКР	Дата выполнения
1 Анализ предметной области	25%	04.04.2024
2 Создание интерфейса	50%	04.05.2024
3 Разработка геймплейной составляющей	25%	04.06.2024

Дата выдачи задания «19» марта 2024 г.

Задание принял к исполнению «19» марта 2024 г.

Аннотация

Ключевые слова: код, программа, функции, значения, Godot, игровой движок, разработка игр, компьютерная игра, среда разработки.

Целью данной выпускной квалификационной работы является разработка игры на движке Godot.

Объектом исследования – игра на движке Godot.

Предмет исследования – процесс разработки игры на движке Godot.

Результатом работы является игра на движке Godot.

Теоретической основой работы послужили электронные ресурсы, статьи, учебные пособия и техническая документация в области разработки игр на движке Godot.

Выпускная квалификационная работа состоит из введения, трех глав, заключения, списка использованной литературы, приложения.

Код программы можно найти по ссылке https://github.com/NikitinMaksim/Coursework_2023

Abstract

Keywords: code, program, functions, values, Godot, game engine, game development, computer game, development environment.

The purpose of this final qualification work is to develop a game on the Godot engine.

The object of research is a game on the Godot engine.

The subject of research is the process of developing a game on the Godot engine.

The result of the work is a game on the Godot engine.

The theoretical basis of the work is electronic resources, articles, tutorials and technical documentation in the field of game development on the Godot engine.

The final qualification work consists of an introduction, three chapters, a conclusion, a list of references, and an appendix.

Program code can be found at the link https://github.com/NikitinMaksim/Coursework_2023

ОГЛАВЛЕНИЕ

ОПРЕДЕНИЯ, ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

Компьютерная игра – компьютерная программа, служащая для организации игрового процесса (геймплея), связи с партнёрами по игре, или сама выступающая в качестве партнёра.

Геймплей – компонент игры, отвечающий за взаимодействие игры и игрока. Геймплей описывает, как игрок взаимодействует с игровым миром, как игровой мир реагирует на действия игрока и как определяется набор действий, который предлагает игроку игра.

Игровой движок – базовое программное обеспечение компьютерной игры.

Игровой ассет или игровой ресурс – цифровой объект, преимущественно состоящий из однотипных данных, неделимая сущность, которая представляет часть игрового контента и обладает некими свойствами. Понятие «игрового ассета» используется при разработке компьютерных игр по отношению к тем элементам контента, которые обрабатываются ресурсной системой как неделимые (атомарные, элементарные) сущности.

Игровой искусственный интеллект – набор программных методик, которые используются в компьютерных играх для создания иллюзии интеллекта в поведении персонажей, управляемых компьютером. Игровой ИИ, помимо методов традиционного искусственного интеллекта, включает также алгоритмы теории управления, робототехники, компьютерной графики и информатики в целом.

Игровая механика – набор правил и способов, реализующий определённым образом некоторую часть интерактивного взаимодействия игрока и игры. Все множество игровых механик игры формируют конкретную реализацию её игрового процесса или геймплея.

Godot – это универсальный 2D и 3D игровой движок, спроектированный для поддержки всех видов проектов.

GDScript – это высокоуровневый, объектно-ориентированный, императивный язык программирования с постепенной типизацией, созданный для Godot. Он использует синтаксис на основе отступов, аналогичный таким языкам, как Python. Его цель – оптимизация и тесная интеграция с Godot Engine, обеспечивая большую гибкость при создании и интеграции контента.

Aseprite – это редактор изображений с доступным исходным кодом, предназначенный в первую очередь для рисования и анимации в стиле пиксель-арт. Он работает на Windows, macOS и Linux и содержит различные инструменты для редактирования изображений и анимации, такие как слои, кадры, поддержка тайловых карт, интерфейс командной строки и другие.

Тайловая, плиточная или знакоместная графика (от англ. tile – плитка) – метод создания больших изображений (как правило, уровней в компьютерных играх) из маленьких фрагментов одинаковых размеров.

ChipTone – бесплатный инструмент для генерации звуковых эффектов, распространяющийся по лицензии CC0, что означает, любые коммерческие и некоммерческие проекты могут выполняться без разрешения автора приложения.

Основными элементами в Godot являются:

Узел (Или нода) – самый маленький строительный блок игры. В качестве свойств могут иметь имя, принадлежность к потоку обработки, свойства обработки (Обрабатывается ли узел только во время паузы, только когда нет паузы, всегда, никогда или наследует это свойство от родителя). К каждому узлу можно прикрепить не больше одного скрипта. Существует огромное количество узлов под самые разные задачи.

Скрипт – файл, содержащий в себе определенный код. В Godot скрипты обрабатываются только если они прикреплены к узле или

загружены отдельно путем подключения автозагрузки (Принцип проектирования «Одиночка» иди «Singleton»).

Одиночка – порождающий шаблон проектирования, гарантирующий, что в однопоточном приложении будет единственный экземпляр некоторого класса, и предоставляющий глобальную точку доступа к этому экземпляру.

Сцена – многократно используемый шаблон. В сцена может быть чем угодно, начиная от персонажа или оружия и заканчивая интерфейсом, домом или отдельным уровнем. В Godot сцены очень гибки и одновременно выполняют роль шаблонов и сцен из других игровых движков. Также следует упомянуть что одной из сцен присваивается титул главной, и именно она загружается первой при запуске игры.

Дерево сцены – все сцены игры собираются в дереве сцен, буквально дереве из сцен. И, так как сцены – деревья узлов, дерево сцены также является деревом узлов.

Сигнал – это шаблон «Наблюдателя» в исполнении движка Godot.

Шаблон «Наблюдателя» – это поведенческий шаблон проектирования. Также известен как «подчинённые». Реализует у класса механизм, который позволяет объекту этого класса получать оповещения об изменении состояния других объектов и тем самым наблюдать за ними.

Шина – в программирование, класс "Издатель" из паттерна "Издатель-подписчик". При возникновении события в «издателе», вызываются методы «подписчика(ов)» на это событие.

Ресурс – обобщенное название всех контейнеров данных. Ресурсом может быть изображение, скрипт, анимация, звуковой файл, сцена и т.д. Также Godot позволяет создавать свои шаблоны ресурсов, содержащие в себе другие ресурсы.

Хитбокс – в компьютерных играх тело, которое проверяется на столкновение.

ВВЕДЕНИЕ

С развитием информационных технологий и ростом популярности видеоигр, их разработка становится все более актуальной и востребованной областью в ИТ-индустрии. Видеоигры одновременно являются средством развлечения, популяризации новых технологий, а также обучения и образования.

Неотъемлемой частью разработки видеоигр стали игровые движки. Они являются специализированной средой разработки и позволяют разработчикам сосредоточиться на создании контента и геймплея, а не о написании функционала с нуля. Одним из самых быстрорастущих игровых движков в индустрии, на данный момент, является Godot. Имея полностью бесплатную модель пользования, открытый код, свой собственный язык программирования, предназначенный специально для разработки игр и разработчиков, активно прислушивающихся к мнению пользователей, данный движок набирает популярность как среди разработчиков, так и среди инвесторов.

Программисты Хуан Линьетски (англ. Juan Linietsky) и Ариель Манзур (англ. Ariel Manzur) начали разработку движка в 2007 году, и через некоторое время он уже использовался как закрытый программный продукт в некоторых компаниях. В феврале 2014 года разработчики приняли решение сделать технологию открытой, и выложили исходники движка под лицензией MIT на GitHub под названием Godot.

15 декабря 2014 движок достиг версии 1.0, первой стабильной версии нового продукта. В ней были добавлены дополнительные шейдеры, navimesh и lightmapping.

4 ноября 2015 проект Godot присоединился к организации Software Freedom Conservancy.

23 февраля 2016 вышла стабильная версия 2.0. Основные акценты в её разработке ставились на существенное повышение удобства работы в движке через рабочее окружение.

22 июня 2016 проекту Godot от Mozilla Open Source Support (MOSS) «Mission Partners» присуждена сумма в $20,000 для того, чтобы добавить в проект поддержку WebSocket, WebAssembly и WebGL 2.0.

Актуальностью данной работы является почти полное отсутствие механики динамической сложности в представителях данного жанра.

Цель исследования – разработка игры на движке Godot.

Объект исследования – игра на движке Godot.

Предмет исследования – процесс разработки игры на движке Godot.

Для достижения поставленной цели были сформулированы следующие задачи:

- проанализировать предметную область;
- разработать основной геймплей и минимальную графику;
- реализовать динамически изменяющуюся сложность;
- протестировать правильность работы всех систем.

Для написания выпускной квалификационной работы были задействованы разнообразные источники информации, включая научные публикации, учебные материалы и электронные ресурсы.

Глава 1 Анализ предметной области

1.1 Характеристика предметной области

Перед созданием видеоигры необходимо ответить на ряд важных вопросов. Данные ответы предоставят достаточно полную информацию для написания дизайн-документа, что, соответственно, значительно ускорит и упростит процесс разработки.

Определение целевой аудитории. Как самый большой сегмент геймеров, целевой аудиторией были выбраны мужчины в возрасте от 16 до 24 лет, владеющие небольшим количеством свободного времени.

Далее предстоял выбор платформы. Ввиду материальных ограничений, разработка консольных игр не является доступным вариантом, соответственно остается мобильный и ПК гейминг. Исходя из статистики, хоть мобильный гейминг и растет, спрос на ПК гейминг все еще сохраняет свои позиции. Однако, разработка на ПК является более простым и быстрым вариантом ввиду гораздо большего разброса производительности, разрешения экрана и объема памяти на мобильных устройствах. Также, в России, ПК гейминг все еще является приоритетным. Исходя из этих факторов, а также учитывая личные предпочтения и доступность людей для будущего закрытого бета тестирования, платформой разработки был выбран ПК.

Далее необходимо было определить продолжительность игрового процесса. Это является настолько же важным параметром, как и платформа, ввиду того что разные целевые аудитории предпочитают разную продолжительность игры. В целях ускорения тестирования было решено остановиться на длительности игры в 5 минут. Такая продолжительность игры позволяет сократить время, необходимое на тестирование баланса игры, а также позволяет войти в нишу игр, доступных в короткий период времени,

например обеденный перерыв или 15-ти минутная разгрузка в процессе выполнения домашнего задания.

Определение игрового процесса. Игровой процесс должен быть динамичным и полноценным, соответственно стратегические игры и игры жанра Idle не подходят. Пазлы также не являются хорошим вариантом, ввиду необходимости расслаблять человека, а не напрягать. Мой выбор пал на жанр шутеров (стрелялок). Данный жанр являлся популярным на протяжении всей истории видеоигр и все еще не начал терять популярности. Для поддержания вовлеченности игроков дополнительным жанром был выбран rogue-lite. Суть этого жанра заключается в случайных улучшениях, получаемых во время каждой отдельной игровой сессии, улучшением характеристик между сессиями и процедурной генерации врагов и локаций. Основной игровой цикл был выбран следующий: Игрой заходит в игру, выбирает персонажа и оружие, после чего на протяжении 5 минут старается пережить нападение постоянно увеличивающихся волн врагов, параллельно получая опыт и улучшения. В конце он получает валюту для улучшений между игровыми сессиям, тратит эту валюту и повторяет цикл пока не закончится время или игра не наскучит ему.

Определение графики. Выбор графики был совершен исключительно исходя из личных умений ввиду того, что в наше время больше ценится стиль, а не фотореализм в графике. Создание графики также должно было занимать минимальное количество времени при сохранении приемлемых результатов. Соответственно, идеальным вариантом стала графика в стиле пиксель-арт и разрешением 32 на 32 пикселя. При необходимости допускается возможность увеличения размера холста или размера самих пикселей.

Определения дизайна уровня. Дизайн уровней должен быть простым и не отвлекать игрока от геймплея. Создание атмосферы планируется за счет использования визуальных шейдеров и эффектов в целях экономии времени

на создание ассетов уровней. Оптимальным вариантом в таком случае являются подземные локации, где большая часть уровня будет скрыта темнотой.

Определение истории (Сюжета). Исходя из выбора подземной локации и механики получения улучшений, главный герой должен обладать причиной быть под землей и навыками для модификации своего оружия. Таким образом была выбрана роль инженера-шахтера, решено посадить главного героя внутрь робота, а врагов сделать инопланетными подземными существа ввиду отсутствия подземных хищников на нашей планете. Также, исходя из того, что игрок остается один против множества врагов, было решено, что по сюжету, он упал в пещеру уровнем ниже и ждет подмоги. Приходом подмоги, по совместительству, можно объяснить ограничение по времени.

Определение персонажей. Персонажи в игре должны соответствовать работе инженера-шахтера. Соответственно, они должны быть изобретательными, старательными, возможно грубые или недовольные условиями труда.

Определение пользовательского интерфейса. Пользовательский интерфейс должен четко отражать нужную информацию, но при этом оставлять как можно больше пространства на экране не занятым. Оптимальным размещением стали 3 шкалы в левом верхнем углу и одна шкала в центре снизу.

Самым главным средством разработки является движок, соответственно, с него выбор и начался. Самыми крупным на данный момент являются Unity, Unreal Engine и Godot.

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.

Плюсы Unity:

- самое большое сообщество из трех вариантов;
- встроенная реклама и аналитика;
- возможность разработки на множество платформ;
- бесплатная версия для игр с доходом меньше ста тысяч долларов;
- самый большой выбор готовых ассетов;
- самый большой выбор обучающих материалов;
- язык программирования C#;
- большие возможности самого движка.

Минусы Unity:

- не удобная структура и интерфейс;
- высокий вес файлов на выходе;
- ошибки, не исправленные на протяжении многих версий;
- несмотря на количество обучающего материала и ассетов, очень сложный для новичка движок;
- 3D графика приемлема, но уступает Unreal Engine.

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 и файтингах.

Плюсы Unreal Engine:

- лучшая графика из трех вариантов;
- удобные инструменты моделирования уровней;
- большое сообщество и количество обучающего материала;
- встроенный визуальный скриптинг;
- бесплатный;
- готовых ассетов меньше количественно, но лучше по качеству.

Минусы Unreal Engine:

- большой вес и программы и финальной игры;
- очень высокий порог входа;
- интерфейс хоть и удобный, но очень объемный;
- плохая производительность.

Godot – 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.

Плюсы Godot:

- лучшая документация из трех вариантов;
- бесплатный;
- размер сообщества уступает только двум другим движкам;
- открытый исходный код, быстрое исправление багов;
- работает на всех платформах и позволяет экспортировать почти на все платформы;
- производительный, игры имеют малый вес;
- удобная организация структуры файлов;
- поддержка GDScript, неполная поддержка C#, C++;
- возможность поиска документации внутри приложения.

Минусы Godot:

- относительно молодой движок, существует необходимость устанавливать дополнительные модули;
- плохая производительность с большим количеством объектов;
- небольшое количество обучающих материалов, частая смена версий и последующее устаревание старых материалов;

– малое количество успешных игр и общая популярность движка.

Unreal Engine данному проекту не подходит ввиду объема и сроков, данный движок больше подходит для крупных проектов, создаваемых командой профессионалов. Unity же хоть и является одним из самых популярных движков, последние несколько лет страдает от плохого руководства и постепенно теряет популярность ввиду этого. Также, мой личный опыт работы с Unity оставил впечатление гораздо хуже, чем Godot. Исходя из всего этого, выбор движка пал на Godot.

Выбор языка программирования соответственно, был сделан при выборе движка, ведь Godot полностью построен вокруг использования языка GDScript и использование других языков хоть и увеличит производительность, но значительно увеличит время разработки.

Выбор программы для рисования ассетов имеет значение только в трех критериях: удобство, стоимость и лицензия на использование. Ввиду предыдущего опыта использования Aseprite, их лицензии на свободное использование созданных ассетов в разработке игр, а также низкой цены (435 рублей на момент написания курсовой), был выбран именно он. Для создания звуков было решено использовать единственный удобный полностью бесплатный редактор ChipTone, распространяемый на itch.io по лицензии, позволяющей выполнять любые манипуляции, не запрашивая разрешения автора.

1.2 Анализ аналогов

Главным вдохновением, и, одновременно, конкурентами являются три игры: Nova drift, 20 minutes until dawn и vampire survivors. Все три игры являются представителями жанра rogue-lite с видом сверху и два из них также являются стрелялками.

Vampire Survivors – компьютерная игра в жанрах shoot’em up и roguelike, разработанная и изданная Лукой Галанте. Игра была первоначально выпущена по модели раннего доступа в 2021 году; полные версии для macOS и Windows вышли в октябре 2022 года.

Основными элементами геймплея являются:

– ограничение по времени 30 минут (в некоторых случаях меньше), после чего игроку засчитывается победа;
– улучшение характеристик между игровыми сессиями;
– случайный выбор улучшений во время игровой сессии;
– большое количество врагов, постепенно заполняющих весь экран;
– все оружие стреляет автоматически, от игрока требуется только ходить и выбирать улучшения.

Рисунок 1.1 – Vampire survivors

20 minutes until dawn – это игра стрелялка в жанре rogue-lite от Flanne. Игрок управляет персонажем, который сражается с непрерывными волнами монстров, с целью пережить натиск до рассвета.

Основными элементами геймплея являются:

– ограничение по времени в 20 минут, после чего игроку засчитывается победа;
– улучшение характеристик между игровыми сессиями;
– случайный выбор улучшений во время игровой сессии;
– все улучшения разделены на ветки по 4, где первое открывает второе и третье, а второе или третье открывает четвертое и самое сильное;
– суперулучшения, появляющиеся если игрок выбрал определенную комбинацию обычных;
– выбор персонажа и оружия перед началом игры;
– большое количество врагов, постепенно заполняющих весь экран;
– игрок может как сам выбирать направление стрельбы, так и позволить игре выбирать направление автоматические.

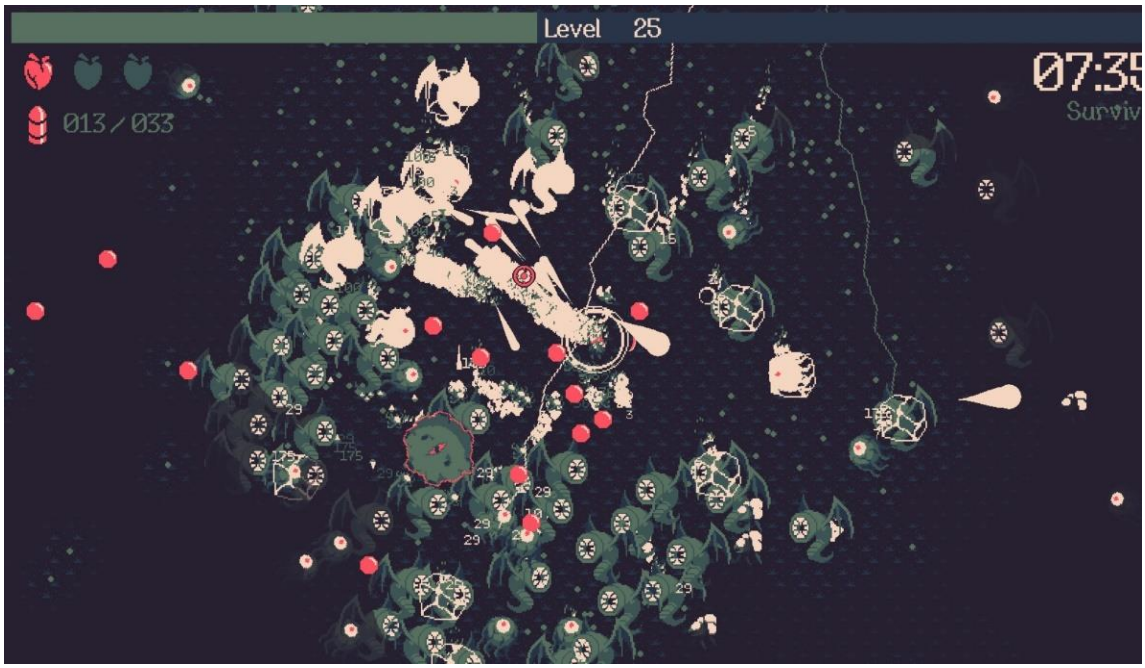

Рисунок 1.2 – 20 minutes until dawn

Nova drift – это космический шутер в жанре rogue-lite, который объединяет в себе впечатления от классических аркадных игр и некоторые возможности современных ARPG.

Основными элементами геймплея являются:

- отсутствие ограничения по времени, вместо этого игра может продолжаться бесконечно;
- между игровыми сессиями открываются новые улучшения, но отсутствует улучшение характеристик;
- случайный выбор улучшений во время игровой сессии;
- все улучшения разделены на ветки по 4, где первое открывает второе и третье, а второе или третье открывает четвертое и самое сильное;
- суперулучшения, появляющиеся если игрок выбрал определенную комбинацию обычных;
- выбор персонажа и оружия в начале игры, а также возможность заменить его вместо получения нового улучшения;
- враги появляются волнами, новая волна приходит или когда предыдущая уничтожена или если у игрока уходит слишком много времени;
- игрок может как сам выбирать направление стрельбы, но движение и стрельбы происходит в одну сторону, также, игрок останавливается не сразу и может скользить по полю боя.

Рисунок 1.3 – Nova drift

Исходя из этого анализа, игроки хотят:

- большое количество врагов;
- различные улучшения между игровыми сессиями;
- большое количество улучшений, взаимодействующих друг с другом во время самого игровой сессии;
- контроль над направлением стрельбы;
- вариативность при выборе персонажа и оружия;
- демоверсию перед покупкой.

И игрокам не нравятся:

- слишком долгие игровые сессии;
- персонажи, отличающиеся только характеристикам;
- излишнее облегчение игры за счет улучшений между игровыми сессиями;
- излишняя легкость или сложность.

В своей игре я хочу совместить все самое лучшее, при этом избежав недостатков и одновременно привести новшество в жанр, используя искусственный интеллект, отвечающий за динамическую сложность.

Основная уникальность моего проекта заключается в динамически подстраивающейся под игрока сложности.

1.3 Анализ задачи

Задачи, которые необходимо выполнить:

- проанализировать предметную область;
- обеспечить интуитивно удобную функциональность приложения;
- реализовать работу в автономном режиме;
- предоставить правильности работы программы.

Для выполнения поставленных задачи нам необходимо:

- спроектировать модель работы программы;
- спроектировать макет графического приложения;
- разработать программу в приложении PyCharm Community Edition;
- протестировать функциональность и работу программы.

Проектирование модели и макета графического приложения необходимы, чтобы составить визуальный проект будущего приложения, необходимый, для создания самого приложения.

Глава 2 Создание интерфейса

2.1 Создание схемы меню и переходов между ними

Игра должна состоять из семи меню и одного интерфейса. Первое меню (Или так называемое «Главное меню») отображается первым после запуска игры и из него совершается переход во все остальные меню.

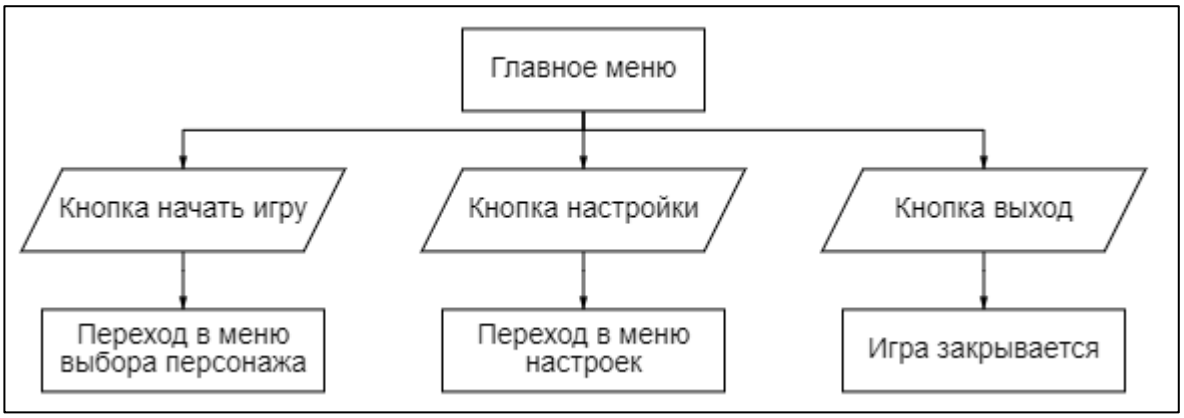

Рисунок 2.1 – Схема главного меню

Вторым меню является меню настроек. Оно состоит из трех вкладок: настроек видео, аудио и управления. Кнопки сохранения, отмены и сброса до стандартных настроек присутствуют во всех трех вкладках. Меню настроек может вызываться сразу из двух мест: из главного меню и из меню паузы во время игры. При закрытии открывается то меню, из которого открывалось меню настроек.

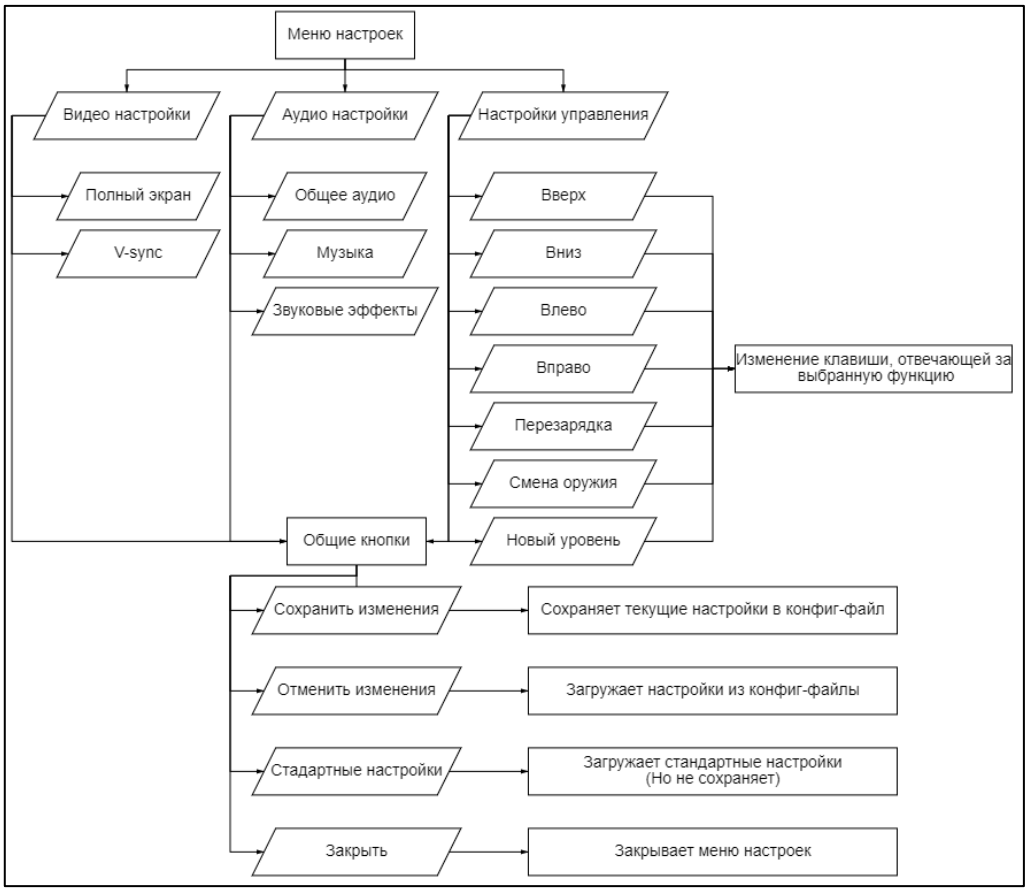

Рисунок 2.2 – Схема меню настроек

Третьим меню является меню выбора персонажа. Здесь игроку предоставлены 3 списка, первый список позволяет выбрать своего персонажа, второй оружие, расходующее патроны, третий оружие, расходующее топливо. При выборе персонажа, его характеристики, изображение и дрон-помощник появляются в правой части экрана. Также отсюда игрок может перейти обратно в главное меню, в меню покупки постоянных улучшений или начать игру.

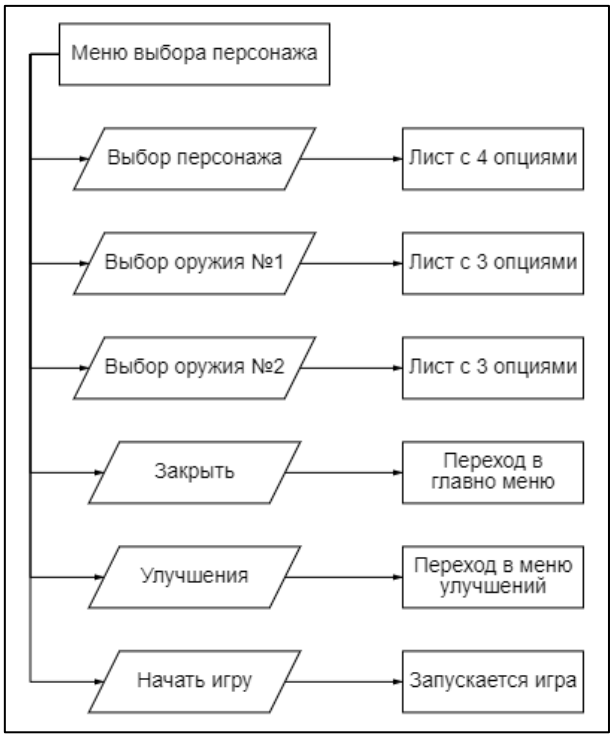

Рисунок 2.3 – Схема меню выбора персонажа

Четвертым меню является меню покупки улучшений. В левой части экрана находятся 6 кнопок, рядом с которыми написан текущий уровень характеристики, максимальный уровень характеристики и цена улучшения. При нажатии на кнопку уровень повышается, и валюта вычитается из хранилища игрока. В правой верхней части показывается оставшееся число валюты. Игрок может сбросить все уровни улучшений до 0 и вернуть себе потраченную валюту или сохранить изменения и вернуться в меню выбора персонажа.

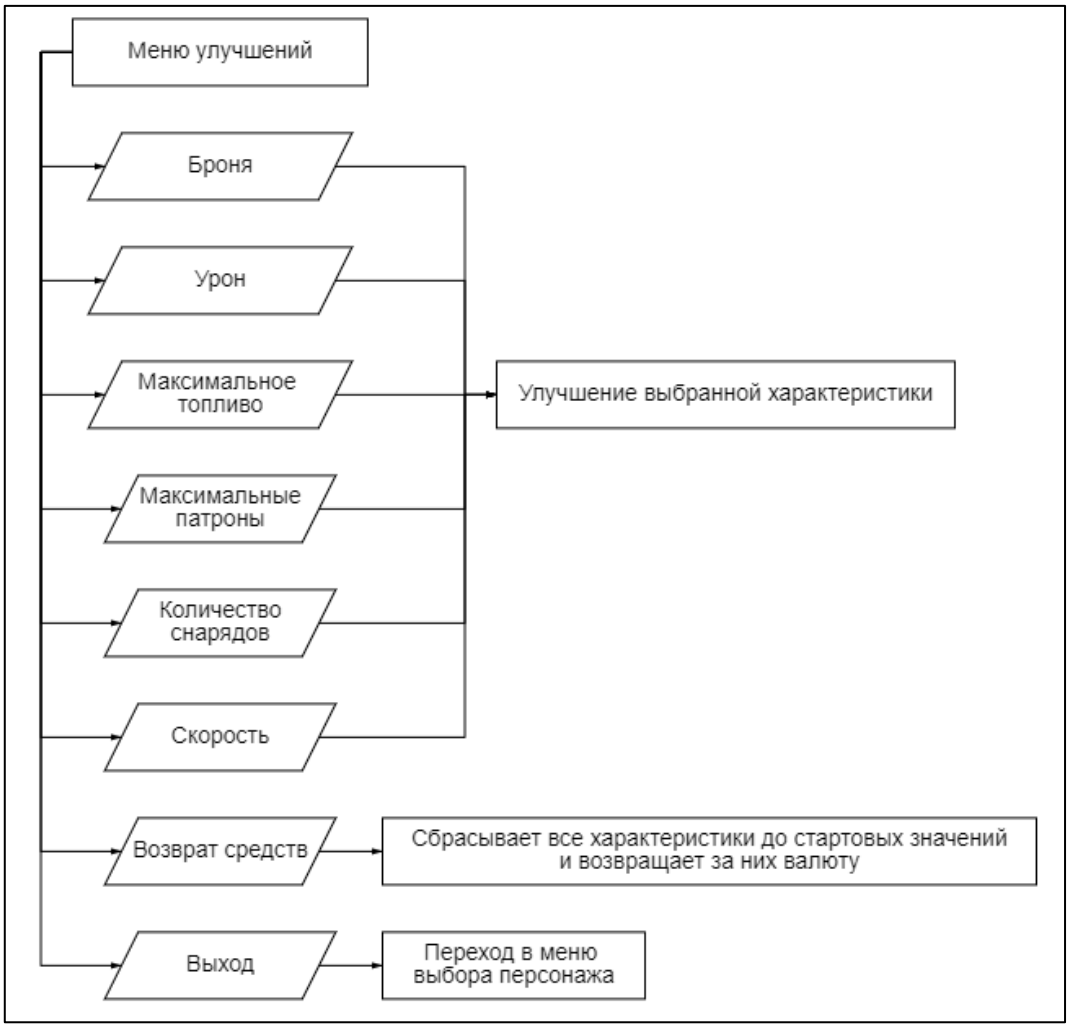

Рисунок 2.4 – Меню покупки улучшений

Следующим по логической цепочке идет игровой интерфейс. Во время игры вся информация отображается по краям экрана, с целью не мешать видимости игрока. Слева в несколько строк идут показатели: здоровья игрока, оставшееся количество патронов, оставшееся количество топлива. Сверху в центре экрана находится секундомер, ведущий счет времени, прошедшего с начала игры. Снизу в центре экрана находится полоска опыта и, при наличии достаточного количества опыта, уведомление о возможности повысить уровень. Также, вокруг персонажа периодически появляются стрелки, указывающие на местоположение ящиков с припасами и залежей руды (В схеме именуемые «Точки интереса»).

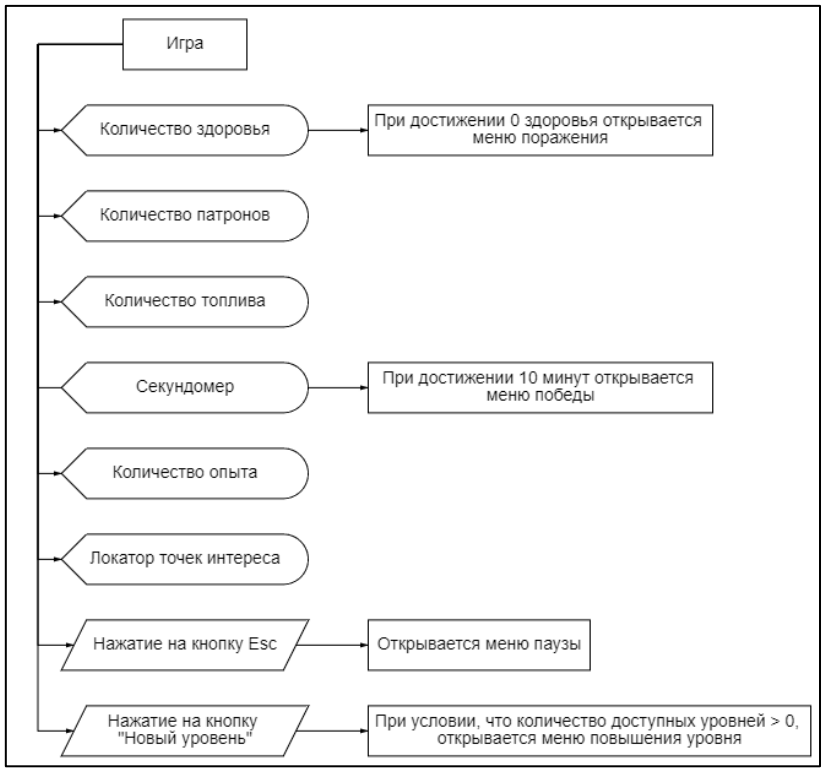

Рисунок 2.5 – Схема игрового интерфейса

Пятым меню является меню паузы. Из меню паузы игрок может вернуться к игре, зайти в меню настроек, выйти в главное меню или полностью закрыть игру.

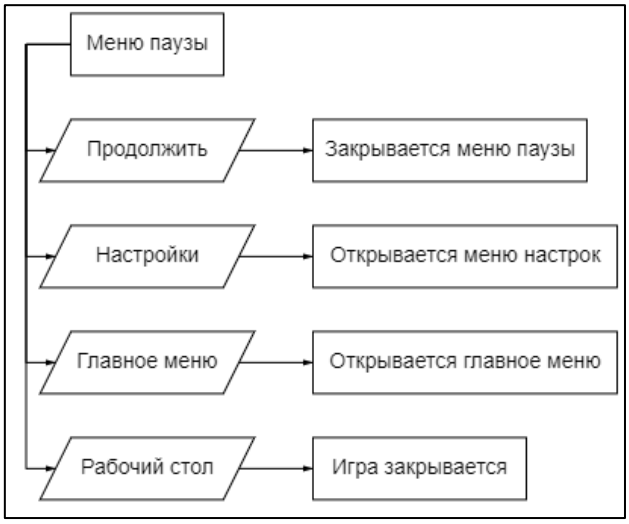

Рисунок 2.6 – Схема меню паузы

Шестым является меню повышения уровня. Открыть его можно в том случае, когда игрок набрал достаточно опыта. При открытии игроку даются на выбор до трех разных улучшений, в зависимости от того, сколько улучшений осталось в массиве.

Рисунок 2.7 – Схема меню повышения уровня

При наведении на улучшение, в правой половине экрана появляется дерево улучшений, состоящее из одного снизу, соединенного с двумя выше него, слева и справа и еще одного сверху, соединенного с двумя ниже.

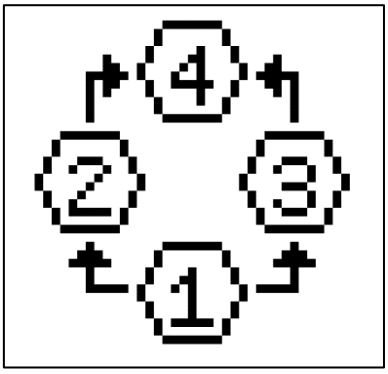

Рисунок 2.8 – Пустое дерево улучшений

Последним, седьмым меню является меню победы/поражения. Оно показывается всякий раз, когда выполняется одно из условий. Если здоровье игрока упало до нуля, тогда это меню появляется с надписью «You lost». Если же игроку удалось дожить до десяти минут, это меню появляется с надписью «You won!».

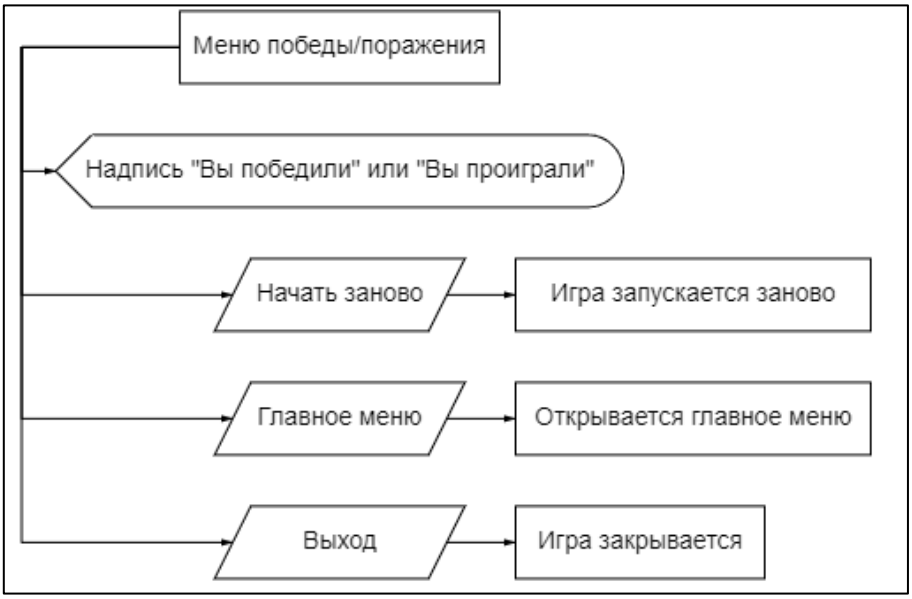

Рисунок 2.9 – Схема меню победы/поражения

Находясь в этом меню, игрок может начать заново, пропуская выбор персонажа, выйти в главное меню или же закрыть игру.

Весь интерфейс и меню создается с использованием нарисованных вручную текстур в стиле пиксель-арт и бесплатного шрифта под названием «Pixeloid-sans».

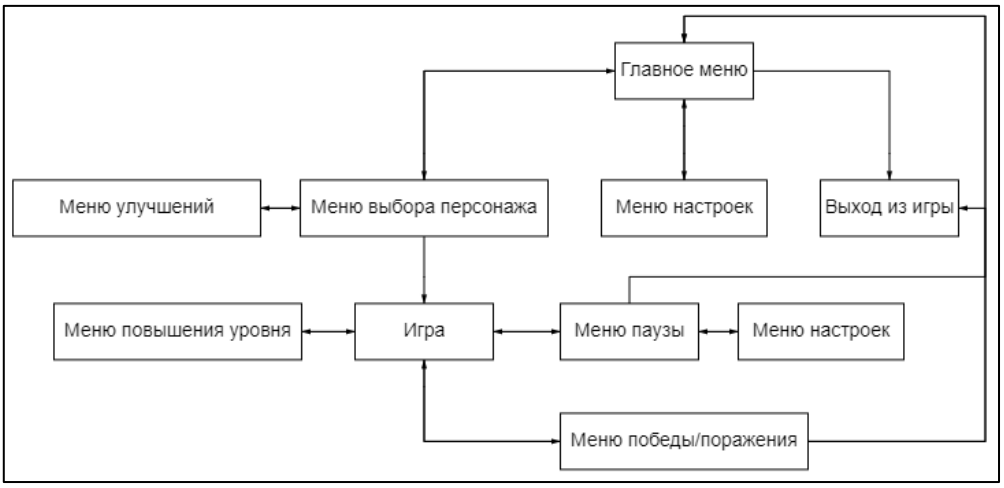

Рисунок 2.10 – Схема переходов между меню

После завершения создания отдельных схем меню, была создана полная схема переходов между меню. Две копии меню настроек являются идентичными и разделены в схеме с целью избежать неправильной интерпретации.

2.2 Разработка меню

В Godot, все, начиная от интерфейсов и заканчивая игровыми персонажами состоит из различных узлов. Все узлы можно разделить на 4 категории по предназначению, «Node2D», «Node3D», «Control» и прочие. В данной главе будут рассмотрены узлы категории «Control». Данная категория отвечает за различные интерфейсы, в том числе кнопки, подписи, изображения, видео, графы и, самый важный, контейнер. Контейнеры во многом похожи на различные свойства, используемые в CSS. Благодаря контейнерам можно автоматически создавать отступы, столбцы, строки, решетки, панели, вкладки и прокручиваемые списки, а также центрировать дочерние объекты или динамически делить содержимое пополам.

По принципу использования все меню делятся на две категории, полная смена сцены и наложение сверху. Ввиду того, что меню настроек используется путем наложения, перед разработкой главного меню, было

разработано меню настроек. В меню настроек используется «TabContainer» – узел, позволяющий переключаться между дочерними объектами, используя мышку.

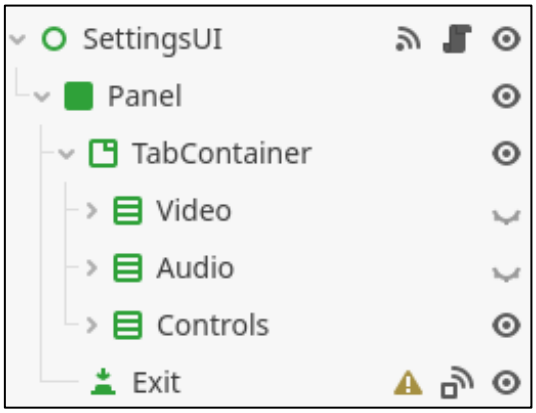

Рисунок 2.11 – Структура меню настроек

Каждую вкладку рассмотрим отдельно. Первой созданной вкладкой были настройки аудио. Каждая вкладка состоит из основного «VBoxContainer», который распределяет все дочерние объекты по строчкам и множества «HBoxContainer», распределяющих свое содержимое по строкам. Настройки аудио содержат в себе три слайдера, основной, отвечающий за общую громкость, слайдер музыки и слайдер звуковых эффектов. Для удобства пользователя, при изменении значения слайдера звуковых эффектов, воспроизводится звук стрельбы, чья громкость соответствует выбранному значению.

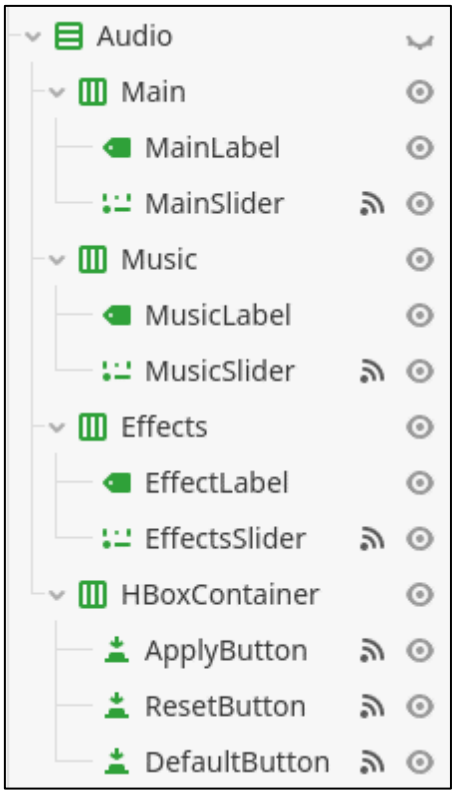

Рисунок 2.12 – Структура вкладки настроек аудио

Также, в каждой вкладке находится отдельный набор из трех кнопок: сохранить изменения, сбросить изменения и стандартные настройки. Подробно работа этих кнопок была описана в первое разделе второй главы.

Работа с громкостью в Godot происходит через аудио шины. Есть основная, или же мастер шина и подчиненные ей шина музыки и шина эффектов. Благодаря такому делению, изменения шины музыки или эффектов имеют эффект, пропорциональный уровню мастер шины.

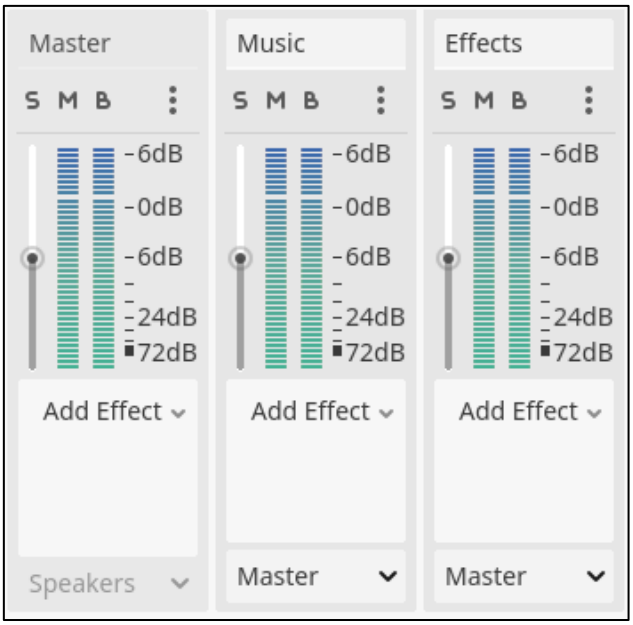

Рисунок 2.13 – Стандартные аудио шины

Основным принципом разработки в Godot принято считать «Сигналы вверх, прямой вызов вниз», что означает, например, при нажатии на кнопку, если необходимо использовать метод дочернего объекта, то его метод вызывается напрямую. Если же требуется вызвать метод родительского объекта, вместо этого используются сигналы. Все кнопки меню настроек через сигналы подключены к своим личным методам в головном скрипте «SettingsUI.gd».

```
func _on_main_slider_value_changed(value):
	AudioServer.set_bus_volume_db(0,linear_to_db(value))

func _on_music_slider_value_changed(value):
	AudioServer.set_bus_volume_db(1,linear_to_db(value))

func _on_effects_slider_value_changed(value):
	AudioServer.set_bus_volume_db(2,linear_to_db(value))
```

Изменение аудио шин происходит через обращение к глобальному классу «AudioServer», после чего вызывается метод «set_bus_volume_db» и аргументами передается id шины и необходимое значение.

```
func _on_apply_button_pressed():
	var err = config.load(savepath)
	if err != OK:
		print("Ошибка:", str(err))
	config.set_value("Audio","Master",db_to_linear(AudioServer.get_bus_volume_db(0)))
	config.set_value("Audio","Music",db_to_linear(AudioServer.get_bus_volume_db(1)))
	config.set_value("Audio","Effects",db_to_linear(AudioServer.get_bus_volume_db(2)))
	config.save(savepath)
func _on_reset_button_pressed():
	config.load(savepath)
	$Panel/TabContainer/Audio/Main/MainSlider.value = config.get_value("Audio","Master")
	$Panel/TabContainer/Audio/Music/MusicSlider.value = config.get_value("Audio","Music")
	$Panel/TabContainer/Audio/Effects/EffectsSlider.value = config.get_value("Audio","Effects")
func _on_default_button_pressed():
	$Panel/TabContainer/Audio/Main/MainSlider.value = db_to_linear(-5.9)
	$Panel/TabContainer/Audio/Music/MusicSlider.value = db_to_linear(-5.9)
	$Panel/TabContainer/Audio/Effects/EffectsSlider.value = db_to_linear(-5.9)
```

Сохранение данных происходит за счет встроенного в GDScript класса «ConfigFile». Его использование позволяет сохранять напрямую на компьютере данные настроек, после чего загружать их при каждом запуске. Сохранение происходит через функцию «set_value», куда аргументом передается по порядку, раздел, потом ключ и далее значение. Разделом в данном проекте выступает название вкладки, то есть «Audio», а ключем название конкретной настройки, в случае аудио это «Master», «Music» и «Effects». Загрузка данных происходит также через класс «ConfigFile», но вместо «set_value» используется «get_value». «get_value» принимает два аргумента, первый это раздел, второй это ключ, после чего возвращает значение, хранящееся по указанному расположению. Стандартные настройки указывались в коде вручную.

Во вкладке видео можно изменить всего две настройки, это вертикальная синхронизация и полный экран.

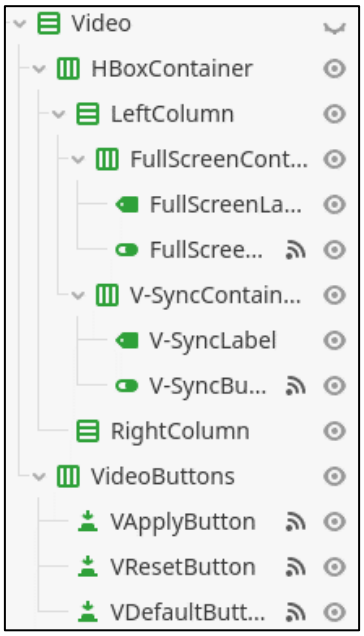

Рисунок 2.14 – Структура вкладки настроек видео

Изменение настроек видео, по аналогии с настройками аудио, происходит через глобальный класс «DisplayServer». В этом классе

вызывается метод «window_set_mode», в случае с включением полного экрана и «window_set_vsync_mode», в случае включения вертикальной синхронизации.

```
func _on_full_screen_button_toggled(toggled_on):
	if toggled_on:
	DisplayServer.window_set_mode(DisplayServer.WINDOW_MODE_FULLSCREEN)
	else:
	DisplayServer.window_set_mode(DisplayServer.WINDOW_MODE_MAXIMIZED)
func _on_v_sync_button_toggled(toggled_on):
	if toggled_on:
	DisplayServer.window_set_vsync_mode(DisplayServer.VSYNC_ENABLED)
	else:
	DisplayServer.window_set_vsync_mode(DisplayServer.VSYNC_DISABLED)
```

Ввиду специфики GDScript, режим работы видео принимает только значения, соответствующие ENUM, встроенному в класс «DisplayServer». Из-за этого появляется необходимость проверять, включена кнопка или нет вместо того, чтобы напрямую сохранять это значение в файл настроек.

```
func _on_v_apply_button_pressed():
	var err = config.load(savepath)
	if err != OK:
		print("Ошибка:", str(err))
	config.set_value("Video","FullScreen",DisplayServer.window_get_mode())
	config.set_value("Video","VSync",DisplayServer.window_get_vsync_mode())
	config.save(savepath)
```

```
func _on_v_reset_button_pressed():
	if config.get_value("Video","FullScreen") == DisplayServer.WINDOW_MODE_FULLSCREEN:
	$Panel/TabContainer/Video/HBoxContainer/LeftColumn/FullScreenContainer/FullScreenButton.button_pressed = true
	else:
	$Panel/TabContainer/Video/HBoxContainer/LeftColumn/FullScreenContainer/FullScreenButton.button_pressed = false
	if config.get_value("Video","VSync") == DisplayServer.VSYNC_ENABLED:
		$"Panel/TabContainer/Video/HBoxContainer/LeftColumn/V-SyncContainer2/V-SyncButton".button_pressed = true
	else:
		$"Panel/TabContainer/Video/HBoxContainer/LeftColumn/V-SyncContainer2/V-SyncButton".button_pressed = false
func _on_v_default_button_pressed():
	$Panel/TabContainer/Video/HBoxContainer/LeftColumn/FullScreenContainer/FullScreenButton.button_pressed = true
	$"Panel/TabContainer/Video/HBoxContainer/LeftColumn/V-SyncContainer2/V-SyncButton".button_pressed = false
```

Последним была разработана вкладка настроек управления. Для реализации этой вкладки было необходимо создать отдельную сцену для кнопки изменения одной горячей клавиши, после чего добавить необходимое количество таких кнопок в основную сцену.

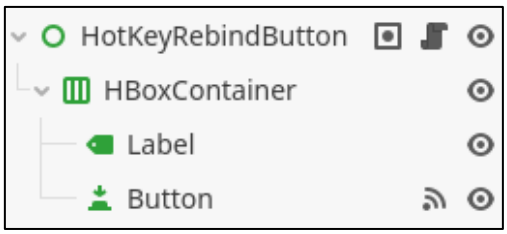

Рисунок 2.15 – Структура сцены «Кнопка смены горячей клавиши»

Каждая кнопка является членом пользовательского класса «HotKeyRebindButton». У этого класса есть экспортируемая переменная, называющаяся «action_name». При инициализации объекта срабатывает функция «_ready», отключающая обработку нажатий клавиш и вызывающая две другие функции, одна из которых назначает подпись рядом с кнопкой, вторая назначает подпись самой кнопки.

```
func _ready():
	set_process_unhandled_key_input(false)
	set_action_name()
	set_text_for_key()
func set_action_name() -> void:
	label.text = "Unassigned"
	match action_name:
		"move_left":
			label.text = "Move left"
		"move_right":
			label.text = "Move right"
		"move_up":
			label.text = "Move up"
		"move_down":
			label.text = "Move down"
		"swap_weapon":
			label.text = "Swap weapon"
		"reload":
			label.text = "Reload"
		"LevelUp":
			label.text = "Level up"
func set_text_for_key() -> void:
	var action_events = InputMap.action_get_events(action_name)
	var action_event = action_events[0]
	var action_keycode =
OS.get_keycode_string(action_event.physical_keycode)
```

```
	button.text = "%s" % action_keycode
```

При нажатии на кнопку, её текст меняется на «Press any key», включается обработчик нажатий клавиш у этой кнопки и выключается у всех остальных. После нажатия на любую клавишу, текст кнопки меняется, обработчик нажатий клавиш выключается, а горячая клавиша в настройках проекта меняется. Изменение происходит за счет вызова глобального класса «InputMap» и методов «action_erase_events» и «action_add_event».

```
func _on_button_toggled(toggled_on):
	if toggled_on:
		button.text = "Press any key"
		set_process_unhandled_key_input(toggled_on)
		for i in get_tree().get_nodes_in_group("hotkey_button"):
			if i.action_name != self.action_name:
				i.button.toggle_mode = false
				i.set_process_unhandled_key_input(false)
		get_tree().get_first_node_in_group("exitbutton").disabled  =
true
	else:
		for i in get_tree().get_nodes_in_group("hotkey_button"):
			if i.action_name != self.action_name:
				i.button.toggle_mode = true
				i.set_process_unhandled_key_input(false)
		set_text_for_key()
		get_tree().get_first_node_in_group("exitbutton").disabled  =
false
func _unhandled_key_input(event):
	if not event.is_action("pause"):
		rebind_action_key(event)
	button.button_pressed = false
func rebind_action_key(event)->void:
	InputMap.action_erase_events(action_name)
```

```
InputMap.action_add_event(action_name,event)
set_process_unhandled_key_input(false)
set_text_for_key()
set_action_name()
```

Для работы с клавишами необходимо использовать один из трех методов класса «InputEventKey», «get_key_label», «get_keycode» или «get_physical_keycode». «get_key_label» возвращает локализированное значение буквы, привязанной к клавише. То есть, при использовании английской раскладки, он вернет, например, букву Q, а при использовании русской, вернет букву Й. «get_keycode» возвращает код самой клавиши, не учитывая различия в физической раскладке. То есть, если взять в пример американские и германские клавиатуры. В американской клавиатуре используется раскладва QWERTY, а в германской QWERTZ. Если использовать «get_keycode» на американской клавише Y и потом запустить, используя германскую клавиатуру, то код будет указывать на другое физическое место клавиши. «get_physical_keycode» в свою очередь, возвращает одно и тоже физическое место нахождения клавиши.

Сохранение, сброс и стандартные настройки особенностей не имеют, ввиду чего в рассмотрении не нуждаются.

Главное меню, в сравнении с меню настроек, имеет простую структуру, состоящую из заднего фона, трех кнопок, одного скрипта и узла, представляющего из себя копию сцены меню настроек.

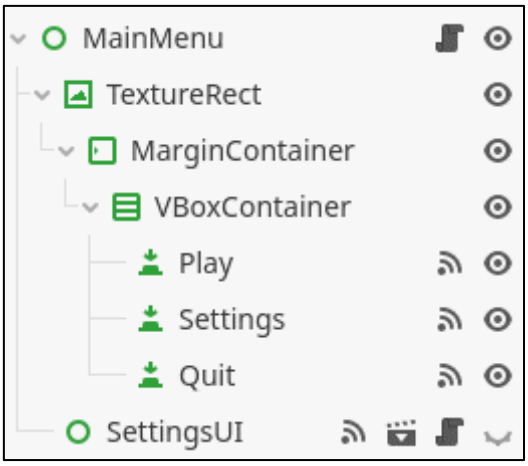

Рисунок 2.16 – Структура главного меню

Используя сигналы, кнопки подключены к главному скрипту и при нажатии выполняют необходимую функцию. Кнопка «Play», переключает сцену на меню выбора персонажа, кнопка «Settings» делает видимым меню настроек, и кнопка «Quit» закрывает приложение.

По аналогии с главным меню и меню настроек, меню покупки улучшений накладывается сверху на меню выбора персонажа, соответственно, сначала необходимо разработать его и потом только меню выбора персонажа.

Для создания меню улучшений, сперва было необходимо создать сцену-шаблон кнопки покупки улучшений и «Singleton», хранящий в себе все информацию об улучшениях, то есть их название, цена, текущий уровень, максимальный уровень, сила эффекта (Процент или фиксированное число в зависимости от вида улучшения). Для хранения информации было решено создать ресурс. Ресурсы в Godot представляют собой гибкий инструмент хранения и обработки информации.

```
extends Resource
class_name MetaUpgradesStats
#Название, текущий уровень, максимальный уровень, цена, сила эффекта
@export var total_points: int = 0
@export var upgrades: Dictionary = {
```

```
    "damage": ["Damage",0,3,50,10],
    "armor": ["Armor",0,3,50,1],
    "speed": ["Speed",0,3,50,5],
    "projectile": ["Projectile",0,3,150,1],
    "max_ammo": ["Max Ammo",0,3,100,10],
    "max_fuel": ["Max Fuel",0,3,100,10]
}
```

Далее была создана сцена, содержащая в себе только узел и скрипт, прикрепленный к нему. Это стандартная архитектура «одиночек» в Godot. Этот скрипт хранит в себе объекты нужного класса, а также имеет все необходимые для работы сеттеры и геттеры.

```
func set_meta_upgrades(stats):
    metaupgrades = stats
func get_meta_upgrades():
    return metaupgrades
func get_remaining_points():
    return metaupgrades.total_points
func subtract_points(amount: int):
    metaupgrades.total_points -= amount
    ResourceSaver.save(metaupgrades, "user://upgrades.tres")
func add_points(amount: int):
    metaupgrades.total_points += amount
    ResourceSaver.save(metaupgrades, "user://upgrades.tres")
```

После этого была создана кнопка покупки улучшения. Структура кнопки состоит из контейнера, подписи и самой кнопки. Скрипт прикреплен к контейнеру и нажатии кнопки через сигнал подключено к этому скрипту.

Логика кнопки включает в себя 4 метода:

- установка данных. Этот метод принимает в качестве аргументов название характеристики, подпись, текущий и максимальный уровни и стоимость, после чего обновляет подписи.
- обновление подписи. Меняет текст кнопки и подписи на текущий и максимальный уровень, а также стоимость или надпись «Full»
- нажатие на кнопку. При условиях того, что очков улучшений больше, чем стоимость и текущий уровень меньше, чем максимальный, очки вычитаются и уровень улучшения улучшается на один.
- возврат. Этот метод вызывается извне и возвращает все очки, затраченные на это улучшение.

При открытии меню улучшений, для каждого улучшения, записанного в словаре ресурса, создается отдельная кнопка с данными из словаря. Соответственно, это позволяет очень легко изменять то, сколько и какие улучшения можно купить.

Также в меню улучшений находится подпись с количеством оставшихся очков улучшения и две кнопки: выход и возврат средств. Кнопка выхода, соответственно, закрывает меню улучшений, а кнопка возврата средств активирует метод возврата у всех кнопок улучшений, сбрасывая тем самым все улучшения до нулевого уровня и возвращая все потраченные очки.

После разработки меню улучшений, было разработано меню выбора персонажа.

Рисунок 2.17 – Меню выбора персонажа

В левой части экрана расположено три списка, верхний позволяет выбрать персонажа, средний – оружие, использующее патроны, нижний – оружие, использующее топливо. При выборе персонажа, справа от него появляются его характеристика: изображение и описание личного дрона-помощника, стартовое здоровье, скорость, максимальный запас патронов и максимальный запас топлива.

В нижней части экрана находятся три кнопки: закрыть, улучшения и начать игру. Кнопка закрывать возвращает игрока в главное меню, кнопка улучшения открывает меню улучшений, и кнопка начать игру совершает переход в игровую сцену.

2.3 Разработка игровых интерфейсов

При входе в игру, пользователь видит перед собой главного героя в центре экрана, полоску здоровье сверху слева, под ней полоску патронов, под ней полоску топлива, в центре сверху таймер, при достижении которым 10 минут игра оканчивается, полоску снизу, означающую прогресс получения следующего уровня и, в зависимости от того, есть ли на карте в данный

момент точки интереса, стрелки, указывающие на эти самые точки. Одна точка интереса появляется с самого начала – это канистра топлива с надписью, в переводе с английского «Используй топливо с умом».

Рисунок 2.18 – Игровой интерфейс

Управление интерфейсом происходит через одиночку «Singnal_bus». Это узел с скриптом, в котором объявлены сигналы и любой другой скрипт может вызвать срабатывание этих сигналов. Благодаря такой архитектуре, создается меньше прямых зависимостей и увеличивается модульность, что, в свою очередь, повышает эффективность и скорость работы.

Меню паузы представляет из себя 4 кнопки, наложенные поверх игрового процесса. Сверху вниз это кнопки:

- продолжить. Закрывает меню паузу и возобновляет игровой процесс.
- настройки. Открывает копию сцены меню настроек, описанную выше.
- главное меню. Возвращает в главное меню.
- выйти из игры. Закрывает приложение.

Рисунок 2.19 – Внешний вид меню паузы

При нажатии на кнопку пробел и при условии того, что игрок набрал достаточно опыта, открывается меню повышения уровня. В левой части игроку предоставляется выбор из трех разных, случайно выбранных улучшений, для каждого из них предоставлено изображение, название и текстовое описание эффекта. В правой части отображается дерево того улучшения, на которое игрок последним навелся. При наведении на улучшения в древе, показывается их описание.

Рисунок 2.20 – Внешний вид меню улучшений

Последнее меню, которое было разработано – это меню победы/поражения. Оно появляется в двух случаях: таймер доходит до

отметки в 10 минут, это победа или игрок получает урон после потери последней единицы здоровья, это поражение. В зависимости от того, победил игрок или нет, в этом меню будет разная надпись, однако остальной функционал останется неизменным.

Рисунок 2.21 – Внешний вид меню победы/поражения

В этом меню присутствуют три кнопки:

- начать сначала. Заново запускает игровую сцену, сохраняя выбор оружия и персонажа.
- главное меню. Возвращает игрока в главное меню.
- выйти из игры. Закрывает приложение.

Таким образом, разработка меню и интерфейсов была завершена, следующей задачей стала разработка основных игровых механик.

Глава 3 Разработка геймплейной составляющей

3.1 Разработка персонажей

Основой любой комплексной системы в Godot является принцип композиции. Все функции объекта разбиваются на мельчайшие удобные для реализации элементы и за счет нужных узлов и скриптов реализуются. Так, например, главный герой состоит из данных компонентов:

- главный узел класса «CharacterBody2d»;
- форма хитбокса;
- узлы, составляющие визуал:
    а) текстура;
    б) проигрыватель анимаций;
    в) дерево анимаций;
    г) текстура оружия;
- маркер дула;
- 5 таймеров:
    а) время между выстрелами;
    б) время перезарядки;
    в) время траты топлива;
    г) время ярости (Одно из улучшений);
    д) время неуязвимости при получении урона;
- узел «Control», отвечающий за пользовательский интерфейс:
    а) текст;
    б) шкала прогресса, показывающая одновременно остаток обоймы в процентах и прогресс перезарядки;
- зона, используемая для отбрасывания врагов при получении урона и её форма.

«CharacterBody2D» представляет из себя физический объект, способный реагировать на столкновения и оптимизированный для создания персонажей в двухмерном пространстве, ввиду чего и был выбран. Текстура оружия имеет собственный скрипт, отвечающий за наведение оружия на курсор игрока.

```
extends Sprite2D
func _process(_delta):
	look_at(get_global_mouse_position())
	if (rotation_degrees<90 or rotation_degrees>270):
		flip_v = false
	else:
		flip_v = true
	rotation_degrees = wrapf(rotation_degrees, 0, 360)
```

Изначально необходимо было дать персонажу возможность двигаться. Самое базовое движение, достаточное для технического демо, создается используя всего 5 строчек.

```
func _physics_process(_delta):
	direction=Input.get_vector("move_left","move_right","move_up","mo
ve_down").normalized()
	if direction:
	velocity=direction*body.speed*fuel_move_speed_modifier*(1+float(m
odifiers["speed"])/100))
	else:
		velocity = Vector2.ZERO
	move_and_slide()
```

Переменные «fuel_move_speed_modifier» и «modifiers[“speed”]» были добавлены позже с целью добавить возможность изменять скорость движения персонажа сторонними модификаторами. Первая переменная

отвечает за модификатор скорости, зависящий от количества топлива, второй же отвечает за прочие модификаторы, будь то улучшения купленные до начала игры или полученные за повышение уровня.

Следующим была разработка логики стрельбы. Изначально планировалось иметь возможность изменять количество снарядов на любое удобное, соответственно, необходимо было разработать формулу нахождения угла вылета снаряда из дула. Разброс в данном случае фиксированный.

Формула угла вылета была выбрана следующая: за каждый X от 0 до количества снарядов – угол наклона ствола - угол разброса/2 + угол разброса/(количество снарядов-1)*X.

Для хранения всех данных были созданы два ресурса: «BodyData» и «WeaponData».

Шаблоны ресурсов в Godot оформляются путем создания скрипта, базирующегося на классе «Resource», после чего им присваивается имя класса и, используя команду @export var задаются все необходимые переменные и их стандартное значение.

```
extends Resource
class_name BodyData

@export var name: String = "Default name, please change"
@export var armor_platings: int = 3
@export var speed: float = 300
@export var max_ammo: int = 300
@export var max_fuel: int = 150
@export var fuel_usage_every_sec: float = 1
@export var sprite_sheet: CompressedTexture2D
@export var drone: PackedScene
@export var drone_description: String
```

Далее используя пкм в файловой системе, можно создавать инстансы данного ресурса и присваивать им значения. На примере стандартного корпуса приведен пример интерфейса редактирования ресурса.

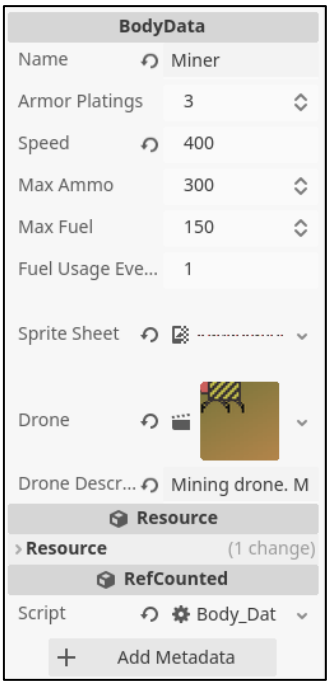

Рисунок 3.1 – Интерфейс редактирования ресурса

По такому же принципу был создан ресурс, хранящий в себе данные об оружии с данными характеристиками:

- наносимый урон;
- вместимость обоймы;
- стоимость одного выстрела;
- скорострельность;
- время перезарядки;
- разброс;
- количество снарядов;
- количество врагов, которые пуля способна пробить;
- количество отскоков между врагами;
- текстура;

– смещение дула (необходимо для создания пули на конце оружия, а не в середине);
– скорость пули;
– максимальная дальность пули.

Также этот ресурс содержал в себе свойство «снаряд», которое имеет типа «упакованная сцена». За счет этого можно отдельно создать снаряд (Будь то пуля, удар дрелью и т.д.), настроить его поведение, внешний вид и хитбокс, после чего просто использовать как свойство ресурса и игра будет автоматически создавать нужный снаряд при выстреле.

Используя экспортируемые переменные, игроку добавляется ресурс BodyData и два ресурса WeaponData, которые используются для определения всех характеристик и внешнего вида персонажа. Загрузка этих ресурсов происходит через одиночку «SingletonDataHolder», уже используемый нами для хранения данных о купленных улучшениях.

Вся информация об улучшениях после загрузки хранится в словаре для удобства доступа.

```
var modifiers = {
	"damage":0,
	"attack_speed":0,
	"pierce":0,
	"max_ammo":0,
	"magazine_size":0,
	"bounce":0,
	"projectile":0,
	"spread":0,
	"max_fuel":0,
	"speed":0,
	"is_split_active":false,
	"is_backwards_fire_active":false,
	"is_rampup_active":false,
```

```
    "is_armor_crafting_active":false,
    "is_rage_active":false,
    "is_armor_to_damage_active":false,
    "is_ammo_crafting_active":false,
    "is_fuel_drops_active":false
}
```

Булевые переменные означают нетипичные улучшения, такие как стрельба за спину, генерация топлива и подобные тому улучшения. Остальные же представляют из себя численные изменения характеристик главного героя.

3.2 Разработка системы улучшений

Для разработки системы улучшений сперва было необходимо создать ресурс, хранящий в себе всю информацию о каждом отдельном улучшении. Было выбрано хранить информацию о:

- внешнем виде улучшения;
- названии улучшения;
- дереве, к которому оно относится;
- уровне улучшения;
- описании улучшения;
- характеристиках, которые улучшение изменяет.

```
extends Resource
class_name UpgradeInfo

@export var image: Texture2D
@export var name: String
@export var tree: String
@export var tier: int
```

```
@export_multiline var description: String
@export var stats = {
	"damage":0,
	"attack_speed":0,
	"pierce":0,
	"max_ammo":0,
	"magazine_size":0,
	"bounce":0,
	"projectile":0,
	"spread":0,
	"max_fuel":0,
	"speed":0,
	"armor":0,
	"is_split_active":false,
	"is_backwards_fire_active":false,
	"is_rampup_active":false,
	"is_armor_crafting_active":false,
	"is_rage_active":false,
	"is_armor_to_damage_active":false,
	"is_ammo_crafting_active":false,
	"is_fuel_drops_active":false
}
```

Следующим шагом было создание экземпляра ресурса для каждого улучшения. В общей сумме было создано двадцать различных улучшений, разделенных на пять веток.

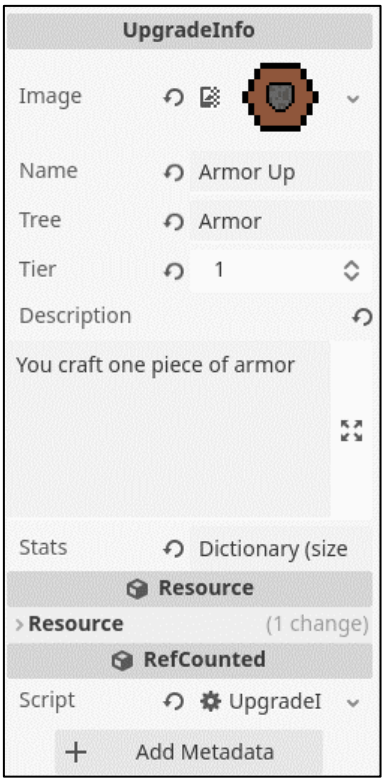

Рисунок 3.2 – Пример окна редактирования улучшения

Во время запуска игры, для каждого улучшения создается пустой узел, к которому прикрепляется скрипт, содержащий в себе функцию, вызываемую при выборе данного улучшения и ресурс конкретного улучшения. Далее они сортируются между доступными и недоступными для выбора. Изначально на выбор доступны только те улучшения, чей уровень («Tier») – это единица. После выбора такого улучшения, в список доступных добавляются улучшения из этого дерева, но с уровнями два и три. После выбора одного из них, в список добавляется улучшение четвертого уровня.

```
extends Node
class_name Upgrade
@export var Info: UpgradeInfo
func OnTake():
	for Child in $"../StashedUpgrades".get_children():
		if Child.Info.tree == Info.tree:
			if Info.tier == 1:
```

```
                    if Child.Info.tier==2 or Child.Info.tier==3:
                        Child.reparent(get_parent())
            if Info.tier == 2 or Info.tier==3:
                    if Child.Info.tier==4:
                        Child.reparent(get_parent())
for stat in Info.stats:
     if typeof(Info.stats[stat])==TYPE_INT:
          if Info.stats[stat]!=0:
SignalBus.modify_player_stats.emit(stat,Info.stats[stat])
     else:
          if Info.stats[stat]!=false:
SignalBus.modify_player_stats.emit(stat,Info.stats[stat])
```

Также, при выборе улучшения, происходит перебор словаря и отправка сигналов изменения характеристик игрока в случае, когда это необходимо.

Для реализации визуальной составляющей, во время запуска игры, помимо создания узлов для каждого улучшения, создается словарь, хранящий в себе информацию о всех улучшениях и через сигнал отправляется в сцену «меню повышения уровня». Там, используя данные из этого словаря, создается визуал всех деревьев улучшений. Варианты выбора улучшений также отправляются через сигнал при нажатии на кнопку повышения уровня.

3.3 Разработка системы динамической сложности

Следующим важным элементом геймплея были враги, каждый враг имеет одинаковый набор характеристик, содержащихся в ресурсе. Здоровье, скорость, урон, названия и количество выпадающего с него опыта. Также они имеют схожие узлы, хитбокс, текстура и т.д. Однако, они отличаются поведением.

Самый маленький враг имеет стайное поведение и, если рядом с ним никого нет, но он видит союзника, они идут друг к другу и только находясь рядом идут в сторону игрока, в противном случае они по одному идут к игроку.

Большие враги идут напрямую к игроку.

А враги-стрелки подходят на определенное расстояние, встают на месте и начинают обстреливать игрока, подходя ближе только если игрок отойдет от них.

И у всех врагов есть общий алгоритм избегания столкновений друг с другом. Если противник видит еще одного или больше противника слишком близко к себе, они начинают идти ровно противоположно друг другу.

С каждого врага при смерти выпадает опыт, который игрок может собрать и при сборе достаточного количества опыта выполнить улучшение.

Все эти враги призываются за счет узла «ИИ директор», действующего по следующему алгоритму:

ИИ директор получает и тратит очки, адаптируясь под успешность игрока. Некоторые подсчеты происходят независимо от времени:

- оружие при выстреле добавляет очки по формуле урон+количество выстрелов в минуту/20;
- при потере бронепластины игроком ИИ теряет 10 очков;
- при потере последней бронепластины ИИ получает 10 очков (Не компенсируется обычной потерей, призвано добить игрока).

Остальные же производятся каждые 15 секунд в данном порядке (Модификатор очков стартует с 0%)

- +15*(количество полных минут с начала игры+1);
- ии подсчитывает количество нанесенного урона ближнего и дальнего боя, сравнивая их процентное соотношение и, если разница больше 10%, выставляет приоритет врагам, противоположным преобладающему типу. (Если враг использует оружие дальнего боя,

на него начинает идти больше крупных врагов ближнего боя и наоборот, если игрок использует оружие ближнего боя, начинает идти больше врагов-стрелков);

- ии подсчитывает соотношение убитых врагов к появившимся и, если игрок убил >80%, игра добавляет к модификатору очков 15%;
- если топливо >80%, игра добавляет к модификатору очков 10% и уменьшает приоритет сброса топлива на следующую минуту;
- если топливо <20%, игра вычитает из модификатора очков 5% и увеличивает приоритет сброса топлива на следующую минуту;
- если патроны >80%, игра добавляет к модификатору очков 10% и уменьшает приоритет сброса патронов на следующую минуту;
- если патроны <20%, игра вычитает из модификатора очков 5% и увеличивает приоритет сброса патронов на следующую минуту;
- ии подсчитывает финальные очки по формуле (Все полученные за 15 секунд очки) *(100%+модификатор);

После всего этого, ИИ тратит очки следуя следующей логике:

- приоритеты всех видов сброса помощи (Бронепластины, патроны, топливо и ничего(всегда4)) суммируются, и эта сумма присваивается к максимуму. К примеру, у игрока меньше 20% патронов и 50% топлива. Соответственно, приоритеты будут выглядеть так: Бронепластины – 1, патроны – 2, топливо – 1, ничего – 4. Всем этим событиям присваивается соответствующая количеству вероятность. 12.5%, 25%, 12.5% и 50% соответственно. Далее используя это распределение случайно выбирается одно событие из четырех;
- выбирается распределение врагов. 50% врагов всегда будут маленькими и быстрыми, остальные 50% будут разделены между сильными и медленными бойцами ближнего боя или бойцами дальнего боя соответственно приоритету. Если приоритета нет, 25 на 25%, если у одного из типов есть приоритет, то 35 на 15%.;

– очки делятся между типами врагов соответственно процентному соотношению и за следующие 15 секунд призывается количество врагов равное их доле разделить на их стоимость.

ЗАКЛЮЧЕНИЕ

В ходе выполнения выпускной квалификационной работы были осуществлены все основные задачи написания программы, проведен анализ предметной области «Разработка игра на движке Godot», углублены и закреплены знания по программированию на языке GDscript и разработке в среде разработки Godot.

Цель выпускной квалификационной работы выполнена – проект игры в программе Godot на языке программирования GDScript.

Следующие задачи были выполнены:

- проанализировать предметную область;
- разработать основной геймплей и минимальную графику;
- реализовать динамически изменяющуюся сложность;
- протестировать правильность работы всех систем.

В первой главе выпускной квалификационной работы была проанализирована предметная область.

Во второй главе представлено описание разработки меню и интерфейсов.

Третья глава направленна непосредственно на разработку геймплейной составляющей проекта.

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. Гэзеуэй Д. Введение в дизайн игровых систем. Пошаговое руководство по созданию сбалансированных игр / Д. Гэзеуэй – Москва: Бомбора, 2023. – 448 с. – Текст : непосредственный.
2. Зубек Р. Элементы гейм-дизайна. Как создавать игры, от которых невозможно оторваться / Р. Зубек – М.: Бомбора, 2022. – 272 с. – Текст : непосредственный.
3. Найстром Р. Паттерны программирования игр / Р. Найстром. – М.: Экскмо, 2022. – 432 с. – Текст : непосредственный.
4. Ходент С. Мозг игрока. Как нейронауки и UX влияют на дизайн видеоигр / С. Ходент – М.: Бомбора, 2023. – 288 с. – Текст : непосредственный.
5. Хёйзинга Й. Homo ludens. Человек играющий / Й. Хёйзинга – М.: Азбука, 2022. – 400 с. – Текст : непосредственный.
6. Шелл Д. Геймдизайн: Как создать игру, в которую будут играть все / Д. Шелл. – М.: Альпина Паблишер, 2019. – 640 с. – Текст : непосредственный.
7. Шрейер Д. Кровь, пот и пиксели / Д. Шрейер. – М.: Литрес, 2018. – 331 с. – Текст : непосредственный.
8. Building a UI in godot [Сайт]. – URL: https://youtu.be/i8ySmMGx--0?list=PL5N89Bry725MaCTKQRWhVoV58JCVikQqe – Текст : электронный.
9. Custom resources – a godot workflow game changer [сайт]. – URL https://youtu.be/vzRZjM9MTGw?list=PL5N89Bry725MaCTKQRWhVoV58JCVikQqe – Текст : электронный.
10. Making infinite background for your game (Godot tutorial) [Сайт]. – URL: https://youtu.be/Qd_vJ1fjYHQ?list=PL5N89Bry725MaCTKQRWhVoV58JCVikQqe – Текст : электронный.

11. Resource gathering RPG in godot 4 tutorial series [Сайт]. – URL: https://youtube.com/playlist?list=PLyH-qXFkNSxlANk9EwZmbtECBfbHeW68-&si=Ux7Yh7ut8kDP58Y2 – Текст : электронный.

12. Save countless lines of code with animations in godot [Сайт]. – URL: https://youtu.be/5KBNGKYV-vU?list=PL5N89Bry725MaCTKQRWhVoV58JCVikQqe – Текст : электронный.

13. Starter state machines in godot 4 [Сайт]. – URL: https://youtu.be/oqFbZoA2lnU – Текст : электронный.

14. Using composition to make more scalable games in godot [Сайт]. – URL: https://youtu.be/rCu8vQrdDDI?list=PL5N89Bry725MaCTKQRWhVoV58JCVikQqe – Текст : электронный.

15. Официальная документация Godot [Сайт]. – URL: https://docs.godotengine.org/ru/4.x/ – Текст : электронный.

16. Официальная страница reddit Godot [Сайт]. – URL: https://www.reddit.com/r/godot/ – Текст : электронный.

17. Официальный форум Godot [Сайт]. – URL: https://forum.godotengine.org/ – Текст : электронный.