Федеральное государственное автономное
образовательное учреждение
высшего образования
«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт космических и информационных технологий

институт

Вычислительная техника

кафедра

УТВЕРЖДАЮ
Заведующий кафедрой
_____ О.В. Непомнящий
подпись инициалы, фамилия
«____» _____ 2022 г.

БАКАЛАВРСКАЯ РАБОТА

Сетевая компьютерная игра типа шутер

тема

09.03.01 «Информатика и вычислительная техника»

код и наименование направления

Руководитель	_____	Доцент, канд. физ.-мат. наук	К. В. Коршун
	подпись, дата	должность, ученая степень	инициалы, фамилия
Выпускник	_____		Д. А. Сахно
	подпись, дата		инициалы, фамилия
Нормоконтролер	_____	Доцент, канд. физ.-мат. наук	К. В. Коршун
	подпись, дата	должность, ученая степень	инициалы, фамилия

Красноярск 2022

РЕФЕРАТ

Выпускная квалификационная работа по теме «Сетевая компьютерная игра типа шутер» содержит 88 страниц текстового документа, 21 иллюстрацию, 9 использованных источников, 1 таблицу.

СЕТЕВАЯ КОМПЬЮТЕРНАЯ ИГРА. ШУТЕР. GODOT.

Целью выпускной квалификационной работы является разработка сетевой компьютерной игры типа «шутер» в режиме реального времени.

Основные задачи:

1. Анализ предметной области, определение инструментов для разработки;

2. Проектирование игры;

3. Разработка игры.

В ходе работы был определен необходимые требования, спроектирована и разработана игра.

Результатом работы является разработанная компьютерная игра, отвечающая всем выдвинутым требованиям в проектировании и поставленной задаче.

СОДЕРЖАНИЕ

Введение....4
1 Анализ предметной области....5
1.1 Постановка задачи....5
1.2 Анализ инструментов для разработки....5
1.3 Вывод....9
2 Проектирование игры....11
2.1 Концепция игры....11
2.2 Общая программная структура....18
2.3 Концепция сетевого модуля....24
2.4 Программная структура сетевого модуля....25
2.5 Вывод....33
3 Разработка игры....34
3.1 Структура и состав проекта....34
3.2 Подробное описание алгоритма....38
3.2.1 Основной класс (Glb)....38
3.2.2 Сетевые классы (NetShared, NetServer, NetClient)....42
3.2.3 Вспомогательные классы (env, g_*, fetchserver)....58
3.2.4 Текстовые уведомления (notifier)....60
3.2.5 Классы главного и игрового меню....62
3.2.6 Игровая логика и игровой интерфейс....67
3.2.7 Игровая арена....78
3.3 Инструкция по сборке....79
3.4 Демонстрация разработанной игры....83
3.5 Вывод....86
Заключение....87
Список использованных источников....88

ВВЕДЕНИЕ

Современная индустрия развлечений находится в состоянии постоянного, беспрерывного развития. Корпорации финансируют разные проекты в стадии зачатия, покупают и перепродают их, в надежде, что в конце концов некоторые из них перевернут всю индустрию с ног на голову, а значит и принесут крупную прибыль, вернут должное инвесторам. Одно из наиболее активных, прибыльных и при этом затратных видов развлечений являются компьютерные игры.

С таким крупным оборотом денег и огромным числом заинтересованных лиц, игровая индустрия должна была разработать некоторый стандартизированный вид, что и произошло. В основе средней компьютерной игры ложится ряд принципов, реализующихся в соответствующем этой средней игре игровом движке. Так как игровой движок реализует все эти стандарты, его можно использовать для разработки разных игр, однако одни игровые движки более специализированные, чем другие. Некоторые из них заточены под определенный жанр игр, другие представляют собой всеохватывающий комплекс инструментов для разработки игры любого жанра. Эти фундаментальные особенности можно реализовывать самостоятельно, то есть, можно создать собственный игровой движок, а можно просто прибегнуть к уже готовому движку для корпоративной и любительской разработки компьютерных игр.

Несмотря на постоянное развитие, сетевые компоненты компьютерных игр, работающих и играющихся в реальном времени, по сей день прибегают к архитектурам, которые были разработаны и доведены до ума в конце девяностых и в начале нулевых. Как показала многолетняя практика, клиент-серверной архитектуры с рассылкой снимков игрового мира достаточно для большей части экшн-ориентированных компьютерных игр.

1 Анализ предметной области

1.1 Постановка задачи

Цель работы: разработать сетевую компьютерную игру типа «шутер».

Отличительные характеристики игр типа «шутер»:

– Игра в режиме реального времени;

– Стрельба как основной способ ведения игрового процесса.

Функциональные требования:

– Игра должна быть сетевой, причем обязательно разработать игру так, чтобы сетевые компоненты разработанной компьютерной игры выполнялись в режиме реального времени, то есть, игровой процесс должен проходить без остановок по каким-либо мнимым игровым условиям, все игроки одновременно воздействуют на происходящее в игре. Это следует из первой отличительной характеристики игры типа «шутер».

1.2 Анализ инструментов для разработки

Исходя из цели настоящей работы был выделен ряд инструментов, которые можно использовать для достижения цели настоящей работы.

OpenGL

Графическое API. Главными особенностями являются кроссплатформенность и аппартное ускорение. Поддерживается тремя основными операционными системами по умолчанию, Linux, macOS и Windows. Реализации этого API служат интерфейсом для низкоуровневых аппаратных операций в отношении использования графического процессора. Используется для отрисовки трехмерной графики, может имитировать двухмерную графику. Имеет лицензионное соглашение «Open-source»,

позволяющее свободное использование в коммерческих и учебных целях [4].

SDL

Мультимедийная библиотека. Кроссплатформенная и со встроенной реализацией OpenGL, а значит и с аппаратным ускорением. Реализует программный интерфейс к графическому отображению, трехмерному и двухмерному, интерфейс к звуковым устройствам и интерфейс к периферийным устройствам ввода. Поддерживает Windows, Mac OS X, Linux, iOS, и Android. Написана на C и работает с C++ по умолчанию, есть возможность использовать библиотеку и с другими языками, такими, как C# и Python. Имеет лицензионное соглашение «zlib», позволяющее свободное использование в коммерческих и учебных целях [5].

DirectX

Мультимедийное API и пакет средств разработки. Поддерживается полностью только на Windows. Работает с C++. Реализует программный интерфейс к графическому отображению, трехмерному и двухмерному, интерфейс к звуковым устройствам и интерфейс к периферийным устройствам ввода. Имеет специфичное лицензионное соглашение как часть пакета «Windows SDK», оно позволяет свободное использование в коммерческих и учебных целях на операционной системе Windows [6].

Game Networking Sockets

Сетевая библиотека, более известная под названием «Steam Networking Sockets». Имеет кроссплатформенность, поддерживает Windows, Linux, macOS, мобильные и консольные платформы. Доступно как общее сетевое

API для видео игр любого типа. Использует одновременно и TCP, и UDP. Шифрует весь трафик при необходимости. Публикуется под лицензионным соглашением «BSD 3-Clause», позволяющим свободное использование в коммерческих и учебных целях [7].

ENet

Сетевая библиотека. Разработана специально для сетевых компьютерных игр типа шутер, работающих в реальном времени. Оперирует исключительно UDP протоколом и «TCP-подобным» UDP протоколом. Эта библиотека не имеет серьезных лицензионных ограничений и доступна полностью бесплатно для учебного и коммерческого использования [8].

Unity

Игровой движок. Поддерживает только трехмерную графику, двухмерная имитируется через трехмерную. Включена нативная поддержка развертывания своего проекта на десктопные (Windows, Linux, MacOS) и мобильные (Android, iOS) операционные системы. Как игровой движок реализует в одном комплексе интерфейсы к графическому выводу, звуковой подсистеме, периферийным устройствам ввода. Поддерживает целый комплекс функций, присущих компьютерным играм. Сетевой функционал приемлем для выполнения настоящей работы. Функциональные возможности движка ограничены в зависимости от используемой версии, в бесплатной версии ограничены сетевые возможности. Лицензионное соглашение позволяет использовать игру, сделанную на движке, в учебных целях. Относительно коммерческого использования есть целый ряд нюансов. Программирование в движке реализуется на языке C#.

Unreal Engine

Игровой движок. Поддерживает только трехмерную графику, двухмерная имитируется через трехмерную. Включена нативная поддержка развертывания своего проекта на десктопные (Windows, Linux, MacOS) и мобильные (Android, iOS) операционные системы. Как игровой движок реализует в одном комплексе интерфейсы к графическому выводу, звуковой подсистеме, периферийным устройствам ввода. Поддерживает целый комплекс функций, присущих компьютерным играм. Сетевой функционал приемлем для выполнения настоящей работы. Лицензионное соглашение позволяет использовать игру, сделанную на движке, в учебных целях. Относительно коммерческого использования необходимо отдавать процент разработчикам движка. Программирование в этом движке реализуется на языке C++.

GameMaker

Игровой движок. Поддерживает только двухмерную графику, поддержка трехмерной находится в состоянии разработки. Функциональные возможности движка ограничены в зависимости от используемой версии, Развертка проекта на любые платформы, кроме специальной платформы разработчика движка «GX.games», не поддерживается. Поддерживает целый комплекс функций, присущих компьютерным играм. Сетевой функционал приемлем для выполнения настоящей работы. Лицензионное соглашение позволяет использовать игру, сделанную на движке, в учебных целях. Относительно коммерческого использования есть целый ряд нюансов. Программирование в движке реализуется на собственном языке GML, основанный на C++.

Godot Engine

Игровой движок. Поддерживает двухмерную графику и трехмерную графику. Включена нативная поддержка развертывания своего проекта на десктопные (Windows, Linux, MacOS) и мобильные (Android, iOS) операционные системы. Поддерживает целый комплекс функций, присущих компьютерным играм. Сетевой функционал приемлем для выполнения настоящей работы. Лицензионное соглашение позволяет использовать разработанную на основе движка игру в коммерческих и учебных целях без ограничений. Программирование в движке реализуется преимущественно на собственном Python-подобном объектно-ориентированном языке GDScript с динамическим типом.

1.3 Вывод

Временные ограничения по выполнению настоящей работы не позволяют начать работать над данным проектом на более низком уровне на основе исключительно библиотек, поэтому было решено, что необходимо будет выбрать именно игровой движок.

Из игровых движков был выбран Godot Engine. Из всех рассмотренных игровых движков он является единственным движком, который точно не имеет функциональных ограничений, так как у него нет никаких платных планов и разделений на разные версии движка в зависимости от соответствующего плана. Получается, что разработка на движке будет упираться исключительно в возможности движка. Его встроенный Python-подобный объектно-ориентированный язык GDScript с динамическим типом позволяет легко разрабатывать проверки концепций. По собственным измерениям и опыту, движок Godot собирает проекты быстрее своих конкурентов и из всех движков он наименее требователен к аппаратным

ресурсам со стороны разработчика. Более того, в движок Godot встроена имплементация библиотеки ENet, разработанной специально под сетевые шутеры, работающие в режиме реального времени.

Для графики будет использоваться редактор растровых изображений Paint.NET и свободные изображения из Интернета.

2 Проектирование игры

2.1 Концепция игры

В левом верхнем углу игры, независимо от текущего состояния игры, должны быть текстовые уведомления. Содержание текстовых уведомлений и их вызов произвольны, при инициализации текстовых уведомлений первое и единственное значение это «Notifier is working». Максимальное количество текстовых уведомлений в один взятый момент равно 6. Если поступает текстовое уведомление и оно превышает максимальное количество то самое старое текстовое уведомление удаляется. Текстовые уведомления очищаются начиная с самого старого с интервалом в 4 секунды. Самое старое находится выше всех, в самом углу. Наиболее молодое, напротив, находится на 5 позиций ниже, где позиция — высота одного текстового уведомления.

На рисунке 1 приведен макет главного меню игры и навигационных путей в виде карточек. Жирная буква в окружности на левом верхнем углу у карточки является идентификатором этой карточки и этого подменю в рамках настоящего проектирования. Простой текст изображен текстом без границ в границах карточки. Нажимаемые кнопки изображены прямоугольниками. Если на левом верхнем углу нажимаемой кнопки есть окружность с некоторой буквой в ней, то это означает, что по нажатию этой кнопки откроется подменю, идентифицируемое этой буквой, остальные подменю скроются из вида. Поля ввода изображены овалами.

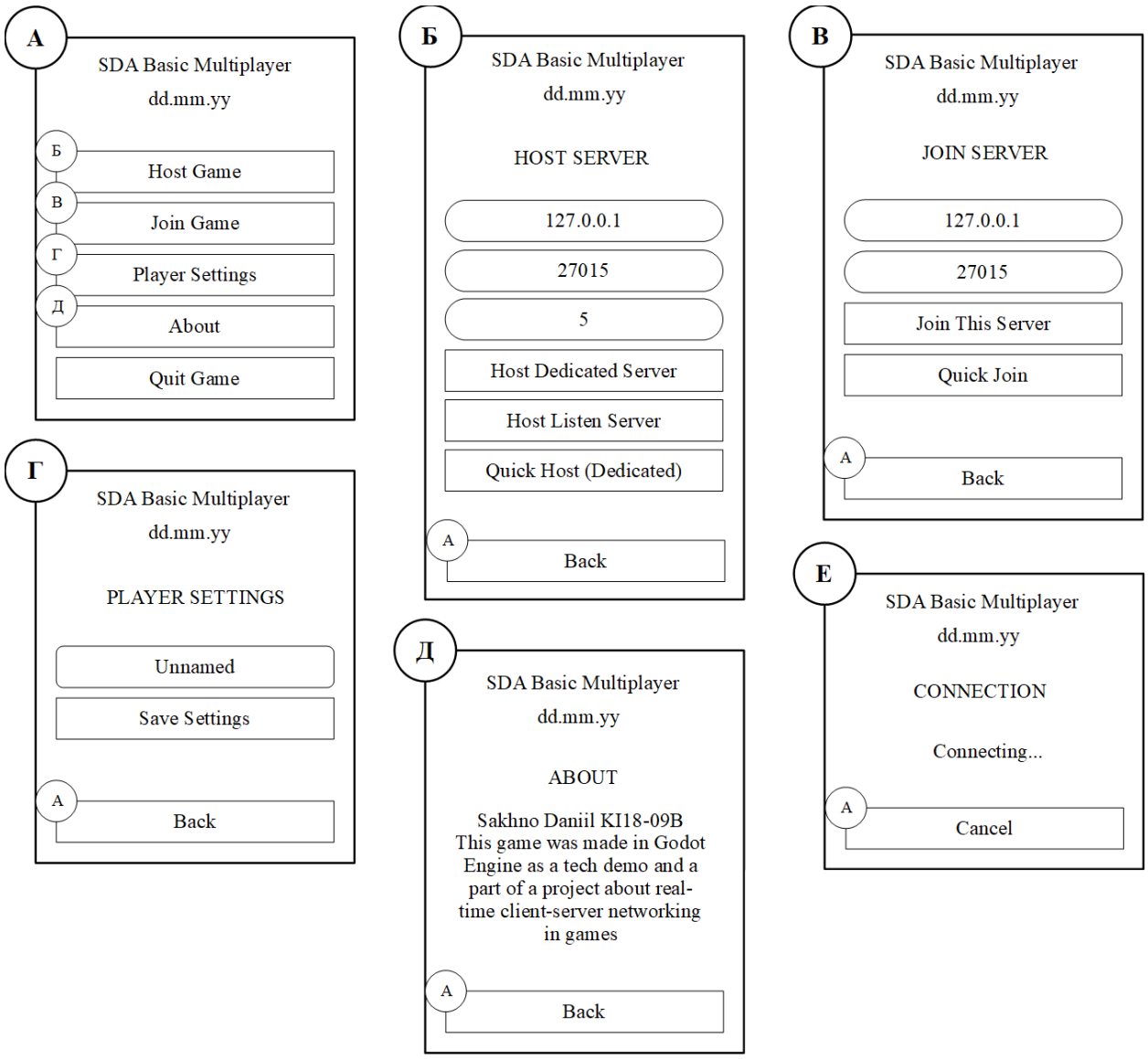

Рисунок 1 — Макет главного меню и навигационных путей в виде карточек

Запуская игру игрок видит главное меню. Главное меню это единственное средство навигации у пользователя по основным программным возможностям, содержит целый ряд различных подменю, которые на момент запуска скрыты от пользователя, за исключением подменю, доступного по умолчанию. Открытие скрытых подменю происходит через навигацию посредством нажатия кнопок из подменю, открытого на момент. Есть специальные подменю, открывающиеся только при определенных условиях, вызванных выборами при навигации.

На всех карточках, а значит и на всех подменю сверху есть название

игры «SDA Basic Multiplayer» и дата выпуска настоящей версии самой игры в формате «dd.mm.yy», где dd – день, mm – месяц, yy – год. У всех подменю, кроме подменю А, есть наименование текущего подменю. Это видно на карточках Б, В, Г, Д и Е.

По умолчанию открыто подменю А из рисунка 1. Это основное подменю в главном меню и должно быть первым объектом что видит пользователь по запуску компьютерной игры. Играет роль связывающего звена между остальными подменю из рисунка 1.

Подменю Б позволяет пользователю создать игровой сервер. Если рассматривать элементы карточки начиная с самого высокого и заканчивая самым низким, то самое первое поле ввода это адрес создаваемого сервера. Второе поле ввода это порт создаваемого сервера. Третье поле ввода это количество максимальных игроков на сервере. Первая нажимаемая кнопка это создание «Dedicated» (выделенного) сервера на основе введенных данных в поля ввода. Вторая нажимаемая кнопка это создание «Listen» сервера на основе введенных данных в поля ввода. Третья нажимаемая кнопка это создание выделенного сервера с константными значениями. Адрес это строка «192.168.0.100», порт равен числу 27015, количество максимальных игроков равно числу 5.

В случае с выделенным сервером окно компьютерной игры, создавшее сервер, не будет участвовать в игровом процессе и не будет занимать игровое место, тем самым оставляя количество игроков пустым после запуска сервера, что позволяет подключиться такому количеству игроков, которое и было указано в третьем поле. В случае с «Listen» сервером создавший сервер так же является и полноценным игроком. Получается, что в таком случае возможных подключений к серверу должно быть на одну единицу меньше. При создании любого сервера должно прийти текстовое уведомление сообщающее об этом. Если произошла ошибка то об этом тоже должно прийти текстовое уведомление. При создании открывается подменю Е.

Подменю В позволяет пользователю присоединиться к игровому серверу. Если рассматривать элементы карточки начиная с самого высокого и заканчивая самым низким, то самое первое поле ввода это адрес создаваемого сервера. Второе поле ввода это порт создаваемого сервера. Первая нажимаемая кнопка берет значения полей ввода этого подменю и пытается с помощью них установить подключение к серверу. Вторая нажимаемая кнопка делает HTTP запрос в отношении некоторого константного URL где содержится зашифрованные перетасовкой соответствующей ASCII кодировки адрес и порт. Если они были найдены и успешно расшифрованы то совершается попытка присоединиться к серверу с этим адресом и портом. При присоединении открывается подменю Е.

Подменю Г позволяет пользователю изменить его отображаемое имя во время игры и сохранить эти изменения. Если рассматривать элементы карточки начиная с самого высокого и заканчивая самым низким, то самое первое поле ввода это отображаемое имя, по умолчанию равное «Unnamed». Самая первая нажимаемая кнопка сохраняет введенное в самое первое поле ввода имя если его длина больше 0 и меньше 10.

Подменю Д содержит описание настоящей компьютерной игры, кем и для чего оно было написано.

Подменю Е это специальное подменю, открываемое при создании сервера или подключения к серверу, его содержание меняется в зависимости от того, создается сервер или подключаются к серверу. В конечном счете оно должно говорить пользователю что сейчас происходит и предоставлять возможность отменить создание сервера или подключение к серверу. Если рассматривать элементы карточки начиная с самого высокого и заканчивая самым низким, то самое первая нажимаемая кнопка преждевременно заканчивает все попытки создания сервера или подключения к серверу и возвращается к подменю А.

На рисунке 2 приведен макет игрового меню.

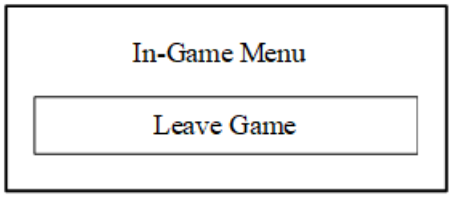

Рисунок 2 — Макет игрового меню

Во время игрового процесса главное меню пропадает и заменяется на игровое меню, которое по умолчанию скрыто, но его можно вызвать, нажав на клавишу «Esc», и таким же образом скрыть. Визуальный стиль у игрового меню такое же, как и у главного меню, но значительно меняется содержание и возможности из меню — игровое меню просто предоставляет возможность завершить текущую игровую сессию и больше ничего.

На рисунке 3 приведен макет таблицы счета игроков.

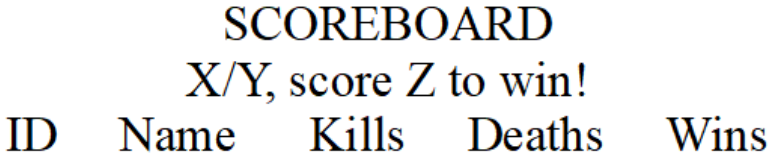

Рисунок 3 — Макет таблицы счета игроков

В макете таблицы счета игроков X это количество игроков на сервере, Y это максимальное количество игроков на сервере, Z это необходимое количество убийств для победы. Строка с ID, Name, Kills, Deaths и Wins это один взятый экземпляр игрока в таблице счета. ID – это идентификатор игрока, присвоенный используемой библиотекой ENet при подключении к серверу или создании сервера. Name – это отображаемое имя игрока, которое он задал в настройках до подключения к серверу. Kills – это количество убийств на момент, Deaths – это количество смертей на момент, Wins – это количество побед.

Чтобы открыть таблицу счета игроков во время игры необходимо нажать клавишу «Tab». При победе таблица появляется сама по себе.

На рисунке 4 приведен макет игровой арены. Жирные черные полоски являются преградами через которые игроки и снаряды не могут проходить. Черными крестиками отмечены точки возрождения и появления игроков.

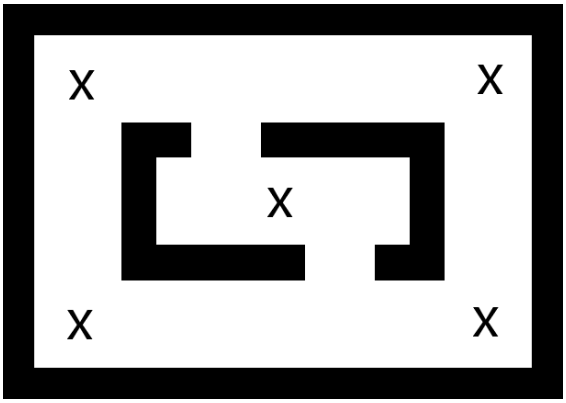

Рисунок 4 — Макет игровой арены

Игровая арена, иначе называемая картой, состоит из преград, препятствующих передвижению всех двигающихся игровых элементов, и пространства, по которому можно передвигаться. На арене присутствует пять точек возрождения и появления игроков.

На рисунке 5 приведен макет игрока и снаряда.

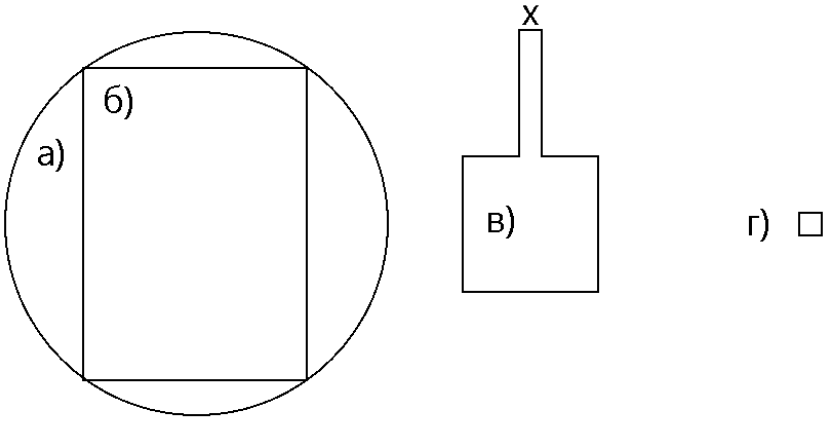

Рисунок 5 — Макет игрока-танка, его турели, тела, коллизии и снаряда.

а) коллизия танка относительно преград на игровой арене и снарядов,

б) «тело» танка, в) «турель» танка, г) площадь коллизии у снаряда

Игрок это есть некоторая боевая машина, назовем эту машину танком. Он двигается по двухмерному пространству и стреляет в сторону указателя мыши пользователя, которого представляет соответствующий танк. Танк визуально делится на две части, «тело» и «турель», однако на деле представлен одним классом. Само такое разделение должно уже дать понять, какая часть за что отвечает. «Тело» - мобильная часть танка, визуально демонстрирующая направление движения, «турель» - часть танка, отвечающая за стрельбу. Под стрельбой понимается появление экземпляра снаряда на определенной позиции наследуемой от «турели», эта позиция отмечена символом «X» на макете турели, рисунке 5.

Игрок двигается с помощью клавиш «W», «A», «S» и «D». Клавиша «W» двигает игрока вверх по двухмерному пространству, «A» двигает игрока влево, «S» двигает игрока вниз, «D» двигает игрока вправо. Стрельба идет при зажатии левой кнопки мыши с небольшим интервалом.

Турель постоянно высчитывает угол между изначальным ее положением, представленным как двухмерный вектор и заданным классом по умолчанию, и вектором, длина которого равна расстоянию от игрока до курсора мыши, начало вектора соответствует положению игрока в игровом мире, конец вектора соответствует положению мыши в игровом мире.

Снаряды, которыми стреляет «турель» этих танков летят в направлении, заданным турелью до того, пока не долетят до игрока или препятствия, либо пока не истечет время их жизни.

Во время игрового процесса камера пользователя преследует соответствующий ему экземпляр класса игрока и перемещается слегка в направлении турели для того, чтобы было лучше видно куда он целится. Если экземпляр игрока отсутствует в следствии смерти игрока или в следствии наблюдения будучи создателем выделенного сервера то положением камеры можно манипулировать теми же клавишами «W», «A»,

«S» и «D» и по тому же принципу, что и движением игрока.

Будучи мертвым или создателем выделенного сервера можно нажать клавишу «T» для обнуления позиции камеры в игровом мире. У создателей выделенного сервера камера заметно отдалена от игрового мира в сравнении с камерой игроков.

Во время игрового процесса внизу и посередине окна отображается здоровье игрока. Если игрок мертв, то отображается текст «PENDING SPAWN». Если был создан выделенный сервер то отображается текст «SPECTATING».

Правила игры. Сам отдельно взятый многопользовательский игровой процесс представляет из себя сетевую и игровую сессию в реальном времени, где переменное множество игроков сражается друг с другом на выделенной игровой арене до тех пор, пока какой-либо из них не наберет некоторый максимум игровых очков, после чего первому набравшему необходимый порог очков причисляется победа и соревнование начинается заново с обнуленными убийствами и смертями у всех игроков. Когда один игрок убивает другого ему причисляется одно очко. Игроки убивают друг друга с помощью снарядов, появляющимися в игровом мире посредством стрельбы с некоторой задержкой.

2.2 Общая программная структура

Игровой движок Godot предоставляет основу для проекта игры. На рисунке 6 приведена диаграмма с абстрактными классами, составляющими собой основу для компьютерной игры на движке Godot. У этих классов только приведены те константы, переменные, сигналы и методы, которые должны использоваться в рамках данной работы. При появлении новой константы, переменной, сигнала или метода относительно наследуемого класса добавляется символ «+» в начале строчки этой константы,

переменной, сигнала или метода. Три точки в конце означают, что у класса есть еще и ряд других параметров, но он скрыт и сведен к этим тремя точкам, так как он не используется в рамках настоящей работы. В первом блоке класса написано его наименование, остальные блоки класса изображаются на диаграмме только при условии, что имеется хотя бы один используемый параметр в рамках настоящей работы параметр. Первый блок после названия описывает используемые константы и используемые переменные, второй блок описывает используемые сигналы, третий и последний описывает используемые методы.

Краткое описание абстрактных классов, изображенных на рисунке 6:

– «Object» это фундаментальный абстрактный класс движка Godot. Все классы в этом движке должны наследовать как минимум от него. Это класс общего назначения и он не решает сам по себе даже такие задачи, как освобождение памяти при отсутствии указателей на нее;

– «Node» это основной абстрактный класс. Служит фундаментом для большей части пользовательских классов. В этот класс встроена поддержка RPC-запросов и тега network_master, целочисленной переменной, изменяющейся только посредством метода set_network_master(). Используется для присваивания «владельца» экземплярам класса;

– «CanvasItem» это основной абстрактный класс для двухмерных классов. Из него используются переменные по окраске отображающихся графических элементов и их видимости;

– «SceneTree» это специальный абстрактный класс, он автоматически ведет запись древа всех активных экземпляров пользовательских классов. Он так же содержит указатель на сетевой класс «network_peer» который должен будет обрабатывать сетевые запросы по RPC;

– «CircleShape2D» и «RectangleShape2D» это двухмерные фигуры для вычисления столкновений;

– «OS» ведет общую информацию о программе и предоставляет

методы для изменения параметров программы. Используется для получения прошедших с момента запуска игры миллисекунд и изменения оглавления окна игры;

– «Node2D» основной двухмерный класс, определяет позицию, вращение и масштаб в игровом мире;

– «Control» основной класс для интерфейса. Используется его переменная о размере элемента интерфейса;

– «KinematicBody2D» основной класс для двигающихся двухмерных объектов в игровом мире. Используется его метод «move_and_slide» для одновременно движения и обработки столкновений с преградами;

– «HTTPRequest» класс для создания HTTP запросов;

– «CanvasLayer» используется для фиксирования позиции элементов интерфейса относительно окна игры;

– «Timer» счетчик времени;

– «NetworkedMultiplayerENet» встроенная в Godot имплементация библиотеки ENet. Используются его методы, переменные и сигналы для создания сервера и подключения к нему, решает сессионные задачи. Присваивает случайные целочисленные идентификаторы подключающимся клиентам, сервер всегда имеет идентификатор 1;

– «Sprite» и «AnimatedSprite» это графическое изображение в игровом мире и его двигающаяся версия соответственно;

– «TileMap» необходим для создания игровой арены;

– «Label» это элемент интерфейса, простой текст;

– «HBoxContainer» и «VBoxContainer» необходимы для выравнивания элементов интерфейса по центру экрана;

– «BaseButton» и «Button» это общий абстрактный класс для нажимаемых кнопок и его более практичный наследник соответственно;

– «LineEdit» это элемент интерфейса, поле ввода;

– «Camera2D» это двухмерная камера в игровом мире;

– «AudioStreamPlayer2D» проигрывает определенный звуковой файл;

– «ColorRect» закрашенный прямоугольник как элемент интерфейса.

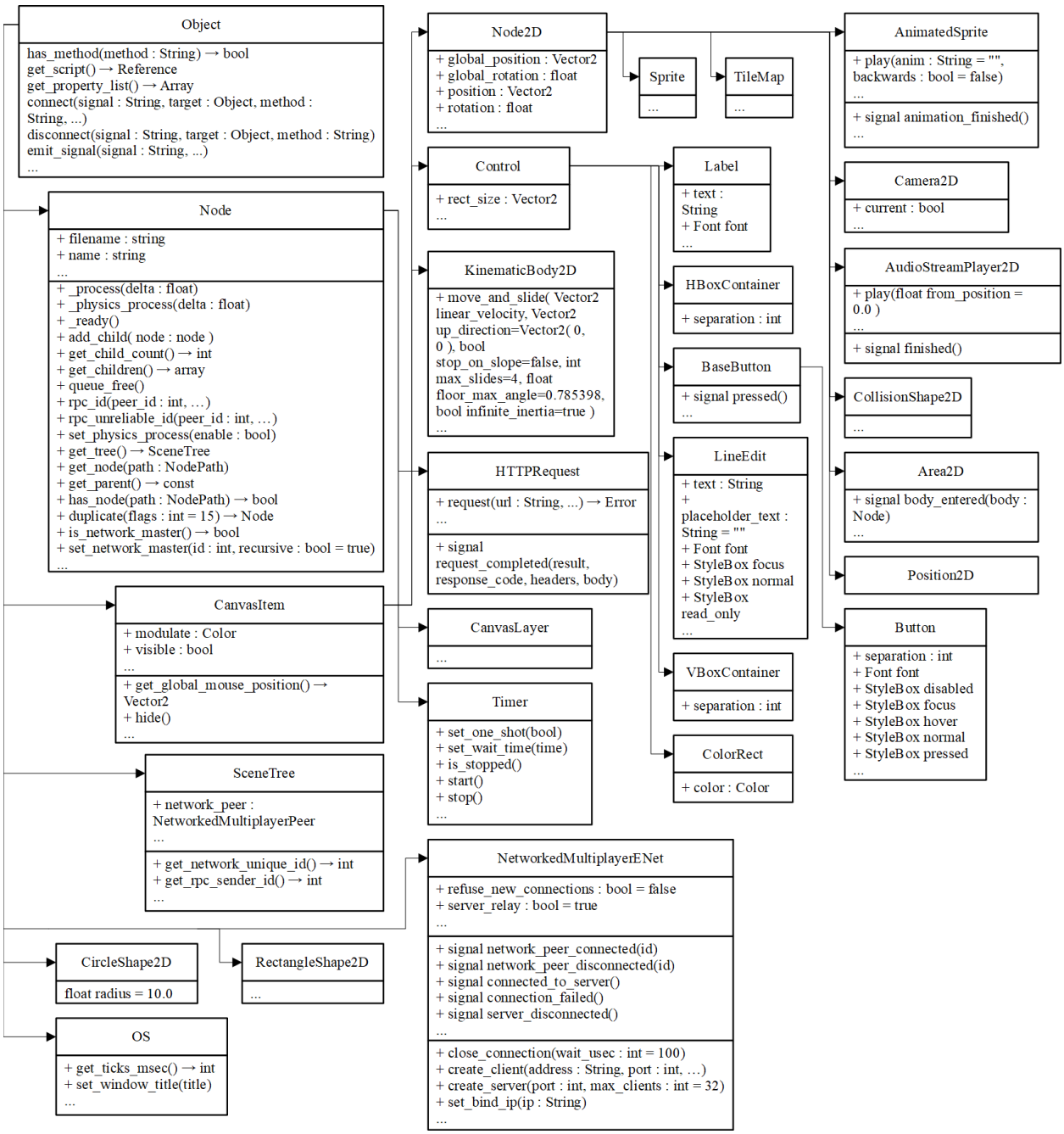

Рисунок 6 — Диаграмма наследования абстрактных классов

На рисунке 7 приведена иерархия наследования классов, разработанных специально для достижения цели и решения задач настоящей работы. Изображаются здесь классы по такому же принципу, что и в

предыдущей диаграмме. Введен «–» как маркер перезаписанного в рамках класса наследуемого метода.

Краткое описание классов, изображенных на рисунке 7:

– «explosion», «hit», «explosion_sound» и «hit_sound». Это визуальные и звуковые эффекты при гибели игрока или попадании по игроку снарядом другого игрока;

– «player_bullet» это снаряд, которым стреляют игроки;

– «tmpl_label» элемент интерфейса, простой текст. Отличается от встроенного в Godot тем, что автоматически расставляет переносы в тексте;

– «Glb» специальный вспомогательный класс, меняет состояния игры, создает и удаляет экземпляры несетевых классов;

– «devout_camera» камера в игровом мире;

– «hp_points» элемент интерфейса, отображающий здоровье экземпляра класса игрока пользователя в мире. Если он мертв, то пишет «PENDING SPAWN». Если пользователь не играет, будучи создателем выделенного сервера, пишет «SPECTATING»;

– «hud_scoreboard» это элемент интерфейса, таблица счета игроков;

– «env», «g_world», «g_persistent», «g_hud», «g_menus», «g_notifiers» это ряд вспомогательных классов;

– «fetchserver» осуществляет HTTP-запросы;

– «map_simple» игровая арена;

– «player_full» класс игрока. Решает все задачи, присущие конкретно игроку, стрельбу, передвижение и тому подобное;

– «menu_main» главное меню;

– «menu_ingame» меню в игре;

– «notifier» класс с текстовыми уведомлениями в левом верхнем углу окна игры.

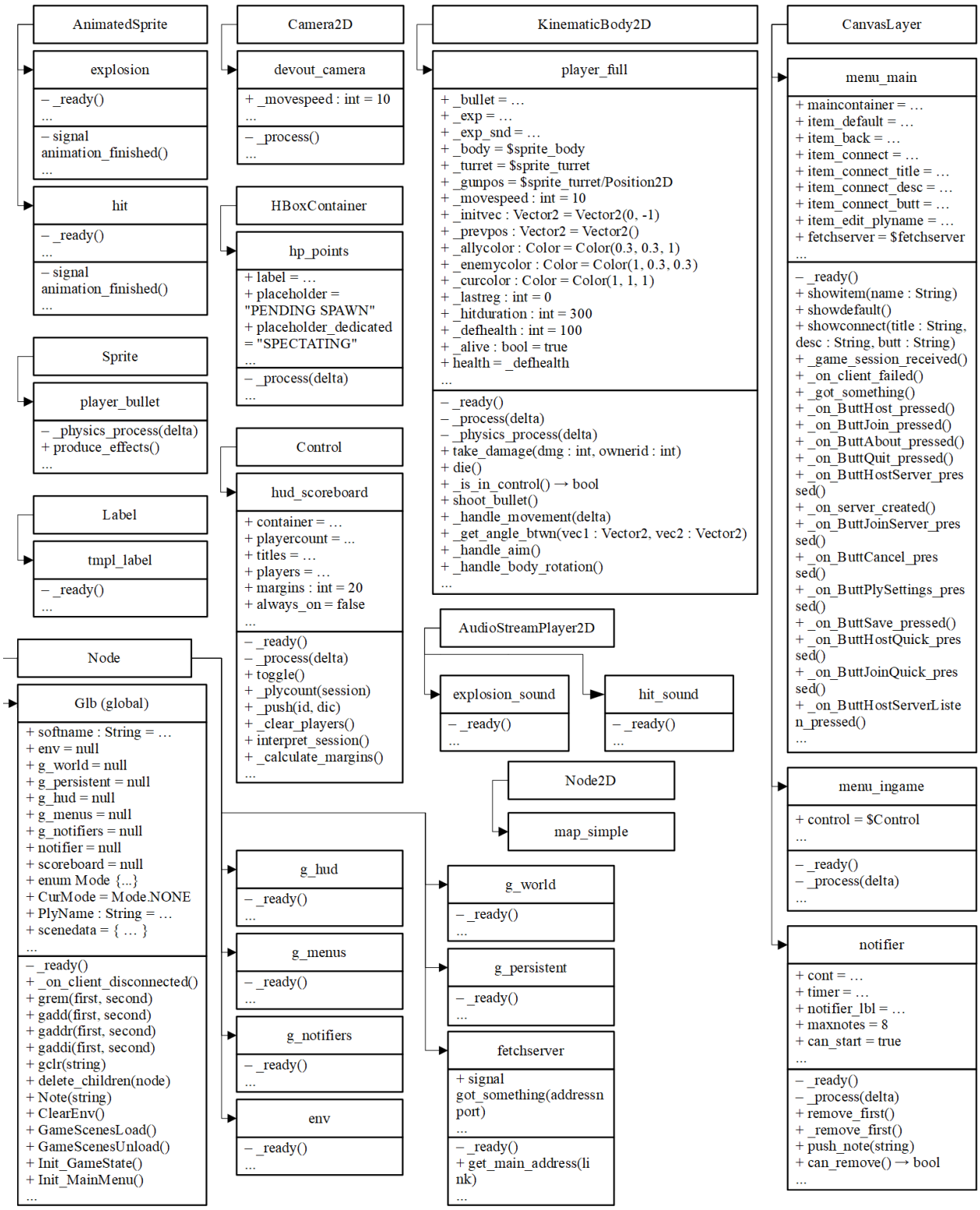

Рисунок 7 — Разработанная иерархия наследования

2.3 Концепция сетевого модуля

Игрок подключившись к серверу должен отправить ему свое отображаемое имя. По отправке отображаемого имени игрока сервер вносит в массив таблицы счета, после чего начинается активный обмен данными.

Эталон сетевой синхронизации приведен на рисунке 8.

Рисунок 8 — Эталон сетевой синхронизации

Серверный сетевой модуль 60 раз в секунду собирает и отправляет всем подключенным клиентам снимок игрового мира. С переменной частотой из-за вероятной потери пакетов и задержки серверный сетевой модуль должен принимать выполняемый клиентами код относительно собственного экземпляра класса игрока. Сервер ожидает такие параметры от каждого клиента, как положение в игровом мире, вращение и действия. При получении он их применяет относительно того экземпляра класса танка-игрока, который соответствует игроку, приславшему эти значения. Если игрок кого-то убил или выступил победителем, это надежно отправляется всем клиентам с учетом потери пакетов.

Клиентский сетевой модуль 60 раз в секунду собирает переменные экземпляра класса танка-игрока, принадлежащего конкретно этому клиенту,

и отправляет собранные переменные с допустимой потерей пакетов. Рекурсивно относительно экземпляра класса собираются положение в игровом мире, положение относительно экземпляров классов-наследников экземпляра класса танка-игрока и то же самое с вращением. При стрельбе клиентский модуль отправляет по надежному каналу, гарантирующему прибытие информации, уведомление о том, что он выстрелил.

2.4 Программная структура сетевого модуля

На рисунке 9 изображена программная структура сетевой части. NetShared – смежный класс, NetServer – класс сервера, NetClient – класс клиента. Node – абстрактный класс, все сетевые классы это его наследники.

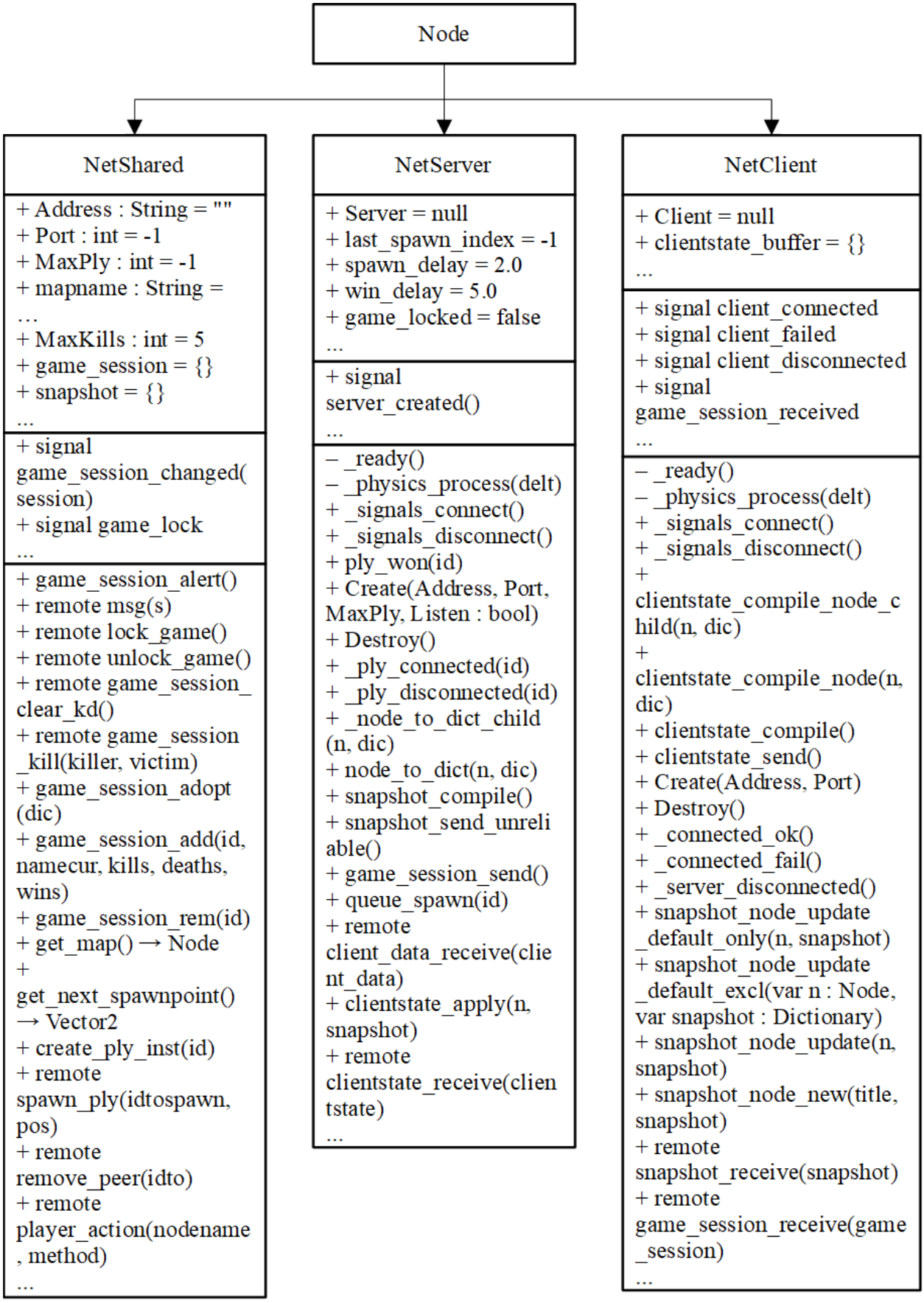

Рисунок 9 — Программная структура сетевой части

Далее идет объяснение сетевых классов с диаграммы на рисунке 9.

«Смежная» часть, класс «NetShared». Задает взаимодействия с игровым процессом со стороны сервера и клиента. Сервер и клиент будут вызывать эти функции, клиент, по большей части, будет это делать по указанию от сервера.

Должны быть следующие переменные:

– Address. Переменная строкового типа, содержит адрес текущей сервера;

– Port. Переменная целочисленного типа, содержит порт текущего сервера;

– MaxPly. Переменная целочисленного типа, содержит максимальное количество игроков;

– mapname. Переменная строкового типа, содержит название класса игровой арены;

– MaxKills. Переменная целочисленного типа, содержит максимальное количество убийств до достижения победы;

– game_session. Переменная типа ассоциативный массив. Содержит всех игроков, допущенных до игры. Играет роль таблицы счета игровых очков;

– snapshot. Переменная типа ассоциативный массив. Содержит последний снимок мира, сделанный сервером или полученный клиентом от сервера.

Должны быть следующие сигналы:

– signal game_session_changed(session). Вызывается когда переменная game_session подверглась каким-либо изменениям.

– signal game_lock(). Вызывается когда игра была окончена после победы какого-либо игрока и когда игра начинается заново.

Должны быть следующие функции, где «remote» - ключевое слово, означающее, что эта функция может вызываться посредством RPC через

сеть:

– game_session_alert() уведомляет таблицу счета об изменениях;

– remote msg(s) при вызове сервером отправляет всем игрокам на сервере текстовое уведомление;

– remote lock_game() забирает возможность контроллировать собственный экземпляр игрока;

– remote unlock_game() удаляет все экземпляры игроков;

– remote game_session_clear_kd() очищает убийства и смерти у всех игроков;

– remote game_session_kill(killer, victim) засчитывает убийство одного игрока другим. При преодолении порога максимального количества убийств на серверной стороне обрабатывает победу преодолевшего порог;

– game_session_adopt(dic) присваивает информацию о счете игроков и уведомляет таблицу счета об этом;

– game_session_add(id, namecur, kills, deaths, wins) добавляет нового игрока в таблицу счета и уведомляет таблицу счета об этом;

– game_session_rem(id) убирает игрока из таблицы счета и уведомляет таблицу счета об этом;

– get_map() возвращает указатель на игровую арену;

– get_next_spawnpoint() возвращает следующую точку возрождения и появления игрока;

– create_ply_inst(id) создает экземпляр игрока относительно пользователя с идентификатором id;

– spawn_ply(idtospawn, pos) создает экземпляр игрока относительно пользователя с идентификатором id и добавляет его в игровой мир на определенной позиции pos;

– remove_peer(idto) безвозвратно удаляет экземпляры игрока из игрового мира.

– player_action(nodename, method) вызывает метод method у экземпляра

класса игрока nodename.

Серверная часть, класс «NetServer». Помимо сетевых функций задает правила игры.

Должны быть следующие переменные:

– Server. Указатель на экземпляр класса «NetworkedMultiplayerENet»;

– last_spawn_index. Целочисленный тип. Последняя точка возрождения, на которой появился игрок;

– spawn_delay. Тип с плавающей точкой. Задержка перед появлением игрока после подключения к серверу или после смерти;

– win_delay. Тип с плавающей точкой. Задержка после победы игрока перед началом новой игры;

– game_locked. Логический тип. Находится ли игра в стадии подготовки к тому, чтобы начать игру заново.

Должен быть следующий сигнал:

– signal server_created(). Вызывается когда был успешно создан сервер.

Должны быть следующие методы:

– _signals_connect(). Подключает сигналы экземпляра класса «SceneTree» к настоящему экземпляру класса NetServer;

– _signals_disconnect(). Отключает сигналы экземпляра класса «SceneTree»;

– _ready(). Запускается при начальной загрузки экземпляра класса, выключает от выполнения метод _physics_process(delta);

– _physics_process(delta). Используется для отправки снимков игрового мира с фиксированной частотой 60 раз в секунду;

– ply_won(id). Причисляет победу игроку с идентификатором id и временно приостанавливает игру. После восстановления обнуляет убийства и смерти всех игроков;

– Create(Address, Port, MaxPly, Listen). Создает экземпляр класса «NetworkedMultiplayerENet» в режиме сервера с адресом Address, портом Port

и максимальным количеством игроков MaxPly и присваивает его указателю Server. Listen это логический параметр. Если он положительный, то создавший сервер так же будет числиться игроком и займет одно место на сервере, в противном случае он просто будет наблюдать за игрой;

– Destroy(). Немедленно уничтожает сервер;

– _ply_connected(). Вызывается при установлении нового подключения к серверу, сервер уведомляется об этом и ожидает информации о клиенте;

– _ply_disconnected(). Вызывается при отключении игрока от сервера. Все на сервере уведомляются об этом текстовым сообщением, игрок удаляется из таблицы счета и экземпляр класса игрока удаляется из игрового мира.;

– node_to_dict_child() и node_to_dict(). Методы для рекурсивного прохода по экземпляру класса игрока для создания на основе его синхронизируемых параметров ассоциативного массива.

– snapshot_compile(). Проходится по всем синхронизируемым между клиентами и сервером экземплярам класса игрока и генерирует на их основе снимок мира, вызывая последовательно node_to_dict() относительно каждого из них;

– snapshot_send_unreliable(). Отправляет без учета потери пакетов снимок игрового мира всем игрокам зафиксированным в переменной game_session;

– game_session_send(). Отправляет всем игрокам, зафиксированным в переменной game_session, текущее значение переменной game_session;

– queue_spawn(id). С ожиданием создает экземпляр класса игрока для пользователя с идентификатором id;

– remote client_data_receive(client_data). Принимает отображаемое имя игрока и записывает его в переменную game_session;

– clientstate_apply(n, snapshot). Применяет полученные от игрока параметры экземпляра класса игрока относительно него;

– remote clientstate_receive(clientstate). Получает информацию о экземпляре игрока клиента в его локальном игровом мире.

Клиентская часть, класс «NetClient».

Должны быть следующие переменные:

– Client. Указатель на экземпляр класса «NetworkedMultiplayerENet»;

– clientstate_buffer. Ассоциативный массив. Содержит отправляемую серверу информацию об подконтрольном клиенту экземпляре класса игрока.

Должны быть следующие сигналы:

– signal client_connected. Вызывается когда было успешно совершено подключение к серверу вплоть до фиксации в переменной game_session;

– signal client_failed. Вызывается когда не удалось подключиться к серверу;

– signal client_disconnected. Вызывается когда клиент был отключен от сервера;

– signal game_session_received. Вызывается когда переменная game_session изменилась.

Должны быть следующие методы:

– _signals_connect(). Подключает сигналы экземпляра класса «SceneTree» к настоящему экземпляру класса NetClient;

– _signals_disconnect(). Отключает сигналы экземпляра класса «SceneTree»;

– _ready(). Запускается при начальной загрузки экземпляра класса, выключает от выполнения метод _physics_process(delta);

– _physics_process(delta). Используется для отправки серверу параметров экземпляра класса игрока с фиксированной частотой 60 раз в секунду;

– clientstate_compile_node_child(n, dic) и clientstate_compile_node(n, dic). Используются для рекурсивной сборки информации об экземпляре класса игрока и выдачи ее в виде ассоциативного массива;

– clientstate_compile(). Проходится по всем подконтрольным клиенту экземплярам класса игрока и вызывает clientstate_compile_node() относительно каждого из них. Из полученных от вызова метода ассоциативных массивов собирает один целый ассоциативный массив и присваивает его переменной clientstate_buffer;

– clientstate_send(). Используется для отправки clientstate_buffer серверу;

– Create(Address, Port). Создает экземпляр класса «NetworkedMultiplayerENet» в режиме клиента с адресом Address, портом Port и присваивает его указателю Client, тем самым совершает попытку подключения к серверу с адресом Address и портом Port;

– Destroy(). Немедленно уничтожает клиентское соединение;

– _connected_ok(). Вызывается при успешном подключении к серверу;

– _connected_fail(). Вызывается при неудачном подключении к серверу;

– _server_disconnected(). Вызывается при отключении от сервера;

– snapshot_node_update_default_only(n, snapshot). Обновляет только положение в игровом мире у экземпляра класса игрока n на основе снимка snapshot;

– snapshot_node_update_default_excl(n, snapshot). Обновляет только здоровье у экземпляра класса игрока n на основе снимка snapshot;

– snapshot_node_update(n, snapshot). Обновляет уже имеющийся экземпляр класса игрока, присваивая ему новые значения, полученные от сервера.;

– snapshot_node_new(title, snapshot). Создает новый экземпляр класса игрока, который ранее не был зафиксирован клиентом, на основе полученного снимка игрового мира. Вызывает snapshot_node_update_default_only() и snapshot_node_update_default_excl() для установки положения в игровом мире и параметров игровой логики соответственно;

– remote snapshot_receive(snapshot). Удаленно вызывается сервером для отправки снимка игрового мира;

– remote game_session_receive(game_session). Удаленно вызывается сервером для отправки таблицы игрового счета.

2.5 Вывод

Были спроектированы игровая и сетевая части в рамках настоящей работы на концептуальном и классовом уровнях.

Концепция игры состоит из текстовых уведомлений, главного меню, игрового меню, таблицы счета, игрока.

На основе этого была разработана общая программная структура, использующая следующие абстрактные классы предоставляемые движком Godot: Object, Node, CanvasItem, SceneTree, CircleShape2D, RectangleShape2D, OS, Node2D, Control, KinematicBody2D, HTTPRequest, CanvasLayer, Timer, NetworkedMultiplayerENet, Sprite, TileMap, Label, HBoxContainer, BaseButton, LineEdit, VBoxContainer, ColorRect, AnimatedSprite, Camera2D, AudioStreamPlayer2D, CollisionShape2D, Area2D, Position2D и Button.

Спроектированы следующие классы в иерархии наследования: explosion, hit, player_bullet, tmpl_label, Glb, devout_camera, hp_points, hud_scoreboard, g_hud, g_menus, g_notifiers, env, g_world, g_persistent, fetchserver, player_full, explosion_sound, hit_sound, map_simple, menu_main, menu_ingame и notifier.

Концепция сетевого модуля состоит из смежного класса NetShared, к которому обращаются остальные и в котором хранят информацию о текущей игровой сессии, класса сервера NetServer, класса клиента NetClient. Все сетевые модули наследуют от абстрактного класса Node.

3 Разработка игры

3.1 Структура и состав проекта

На рисунке 10 приведена файловая структура настоящего проекта.

Рисунок 10 — Файловая структура проекта

Проект содержит в себе следующие виды и форматы файлов:

– Файлы с построением класса из редактора Godot (*.tscn).

– Вспомогательные файлы графического стиля (*.tres), все из которых кроме одного (default_env.tres) находятся по пути «./menus/templates/*».

– Файлы с исходным кодом классов на Python-подобном языке программирования GDScript (*.gd).

– Ресурсы с звуковой информацией (*.wav). Для этих ресурсов выделена следующая специальная директория (путь) «./sfx/*».

– Шрифтовые ресурсы (*.ttf), находятся по пути «./assets/*».

– Ресурсы, содержащие графическое изображение (*.png). Такие ресурсы доступны в директории «./assets/*», некоторые группируются добавлением в дополнительную директорию по все тому же пути, например, «./assets/player/*».

– Файлы с правилами использования ресурсов в рамках проекта в среде Godot (*.import).

– Основные конфигурационные файлы, свойственные любому проекту в среде Godot (*.project, *.cfg).

В таблице 1 приведены классы проекта, их название, назначение и количество строк в исходном коде.

Таблица 1 — Классы проекта

№	Название	Назначение	Кол-во строк
1	Glb (global.gd)	Глобальный класс. Ответственный за смену состояний, добавление и удаление экземпляров классов из памяти. Содержит указатели к некоторым	155
2	NetShared (net/net_shared.gd)	Глобальный класс. Функции обработки и хранения информации применяемые в сетевой сессии. Хранит общую информацию о текущей или последней сетевой сессии. Глобальные игровые события, затрагивающие всех клиентов и сервер. Взаимосвязан с классами NetServer и NetClient.	169
3	NetServer (net/net_server.gd)	Глобальный класс. Работа сервера, поддержка информации об игровой сессии, обработка запросов от подключающихся клиентов, их аутентификация и добавление в игровой мир. Снятие снимков игрового мира с серверной стороны и рассылка этого снимка между всеми клиентами с фиксированной частотой. Взаимосвязан с классами NetShared и NetClient.	194
4	NetClient (net/net_client.gd)	Глобальный класс. Соединение с сервером, отправка информации о себе серверу для «регистрации» в перечне подключенных клиентов. Снятие симулируемого на клиентской стороны кода и отправка его серверу. Взаимосвязан с классами NetShared и NetServer.	192
5	Вспомогательные классы (env, g_hud, g_menus, g_notifiers, g_world, g_persistent)	Инициализация ряда вспомогательных классов.	36

Продолжение таблицы 1

№	Название	Назначение	Кол-во строк
6	Главное меню (menu_main.gd)	Основная навигация по всем программным возможностям. С помощью настоящего класса пользователь может создать сервер, присоединиться к серверу как клиент, поменять собственное отображаемое имя во время игры, посмотреть информацию об игре и выйти из нее.	152
7	HTTP запрос (fetchserver.gd)	Специальный вспомогательный класс. Обращается по HTTP к заранее определенной странице за адресом и портом сервера и обрабатывает полученные данные.	50
8	Упоминания (notifier.gd)	Отображает в текстовом виде многие события в игре.	52
9	Игровое меню (menu_ingame.gd)	Позволяет выйти из игровой сессии в главное меню. При выходе заканчивается работа сетевой сессии.	20
10	Камера (devout_camera.gd)	Следует за игроком в игровом мире или ожидает его появления если сетевая сессия работает в режиме клиента. Позволяет просто наблюдать за игрой если сеть работает в режиме сервера.	18
11	Танк-игрок (player_full.gd)	Представление подключившихся игроков в игровом мире. Главный и единственный способ игроков влиять на игровой процесс и соревноваться с другими игроками.	133
12	Снаряд (player_bullet.gd)	Вызывается классом танка-игрока. Движется в заданном танком-игроком направлении до тех пор, пока не встретится с преградой или оппонентом.	38

Окончание таблицы 1

№	Название	Назначение	Кол-во строк
13	Отображение здоровья (hp_points.gd)	Отображает здоровье игрока или пишет об ожидании появления на игровой арене.	11
14	Счет всех игроков (hud_scoreboard.gd)	Отображает всех успешно подключившихся клиентов к серверу и информацию о них, идентификатор, имя и игровые очки.	45

3.2 Подробное описание алгоритма

3.2.1 Основной класс (Glb)

Файл с исходным кодом: «./global.gd» [9].

Наследует от абстрактного класса «Node».

Объявляет константу строкового типа «softname» с значением, равным «SDA Basic Multiplayer 20.05.2022».

Glb объявляет ряд временно пустых указателей к экземплярам классов, к этим указателям потом будут присвоены адреса экземпляров классов либо ими же при их инициализации, либо вызовом методов в настоящем классе. Перечень этих указателей: env, g_world, g_persistent, g_hud, g_menus, g_notifiers, notifier, scoreboard.

Объявляет «Mode» типа «enum» со следующими возможными значениями: NONE = 0, SERVER = 1, CLIENT = 2.

Инициализирует переменную «CurMode» значением «Mode.NONE».

Содержит переменную строкового типа PlyName, это отображаемое имя игрока.

Объявляет переменную «scenedata» типа ассоциативный массив, работающую как кэш с путями к подгружаемым во время непосредственно игры экземплярам классов и указатель к уже подгруженным экземплярам

классов в памяти в случае, если это уже произошло. «scenedata» состоит из других ассоциативных массивов, ключи которых являются наименованиями подгружаемых классов. У этих ассоциативных массивов есть ключи «path» и «res». Пути к файлу класса хранятся в переменной с ключом «path», указатель к блоку памяти с экземпляром класса в переменной с ключом «res». Таким образом, эталон записи об одном конкретном классе в переменной «scenedata»:

“scene” : { “res” : null, “path” : “res://scenes/scene.tscn” }

При этом значение «scenedata» по умолчанию:

```
scenedata = {
"ply": { "res": null, "path": "res://actors/player_full.tscn" },
"map": { "res": null, "path": "res://maps/map_simple.tscn"},
"ingame": { "res": null, "path": "res://menus/menu_ingame.tscn"},
"main": { "res": null, "path": "res://menus/menu_main.tscn"},
"camera": { "res": null, "path": "res://actors/devout_camera.tscn"},
"scoreboard": { "res": null, "path": "res://hud/hud_scoreboard.tscn"},
"hp": { "res": null, "path": "res://hud/hp_points.tscn"}
}
```

Glb это главный менеджер классовой наследственности в проекте. Для этого были разработаны следующие методы:

– grem(first : String, second : String). Обращается к экземпляру класса, на который указывает указатель «env», и просит у него перечень его экземпляров-наследников. Из этих наследников он ищет экземпляр с наименованием, соответствующим значению параметра «first». Если он был найден, то теперь у этого экземпляра он ищет экземпляр с названием, соответствующим параметру «second». Если этот экземпляр был найден, то он его удаляет из перечня наследников и из памяти.

– gadd(first : String, second : String). Обращается к экземпляру класса, на который указывает указатель «env», и просит у него перечень его

экземпляров-наследников. Из этих наследников он ищет экземпляр с наименованием, соответствующим значению параметра «first». Если он был найден, то он подгружает класс по пути, переданному параметром «second», и создает его экземпляр. Созданный экземпляр добавляется в перечень экземпляров-наследников экземпляра с названием «first».

– gaddr(first : String, second). Обращается к экземпляру класса, на который указывает указатель «env», и просит у него перечень его экземпляров-наследников. Из этих наследников он ищет экземпляр с наименованием, соответствующим значению параметра «first». Если он был найден, то он создает экземпляр класса, переданного параметром «second». Созданный экземпляр добавляется в перечень экземпляров-наследников экземпляра с названием «first».

– gaddi(first : String, second), бращается к экземпляру класса, на который указывает указатель «env», и просит у него перечень его экземпляров-наследников. Из этих наследников он ищет экземпляр с наименованием, соответствующим значению параметра «first». Если он был найден, то он добавляет экземпляр класса, переданный параметром-указателем «second», в перечень экземпляров-наследников экземпляра с наименованием «first».

– delete_children(node), удаляет всех экземпляров-наследников у экземпляра класса, переданного по указателю параметром «node».

– gclr(string). Обращается к экземпляру класса, на который указывает указатель «env», и просит у него перечень его экземпляров-наследников. Из этих наследников он ищет экземпляр с наименованием, соответствующим значению параметра «string». Если такой был найден, то у этого экземпляра удаляются все экземпляры-наследники.

– ClearEnv(). Последовательно вызывает метод gclr() несколько раз, передавая ему следующие строки: «g_world», «g_persistent», «g_menus» и «g_hud».

– GameScenesLoad(). Проходится по всем ассоциативным массивам в «scenesdata» кроме ассоциативного массива «main» и использует их дочерние ключи, а именно загружает в память класс по пути в значении ключа «path» и передает указатель на этот блок памяти в дочерний ключ «res».

– GameScenesUnload(). делает противоположное действие относительно «GameScenesLoad()». Выгружает из памяти классы и присваивает значения null указателям по ключу «res» в «scenedata».

– Init_GameState(). Вызывает метод «GameScenesLoad()». Вызывает gclr() с параметром «g_menus». Вызывает gaddr() с параметрами «g_world» и scenedata.map.res. Вызывает gaddr() с параметрами «g_menus» и scenedata.ingame.res. Вызывает gaddr() с параметрами «g_world» и scenedata.camera.res. Создает экземпляр класса по пути scenedata.scoreboard.res и присваивает указатель на него переменной scoreboard. Вызывает метод gaddi() с параметрами «g_hud» и scoreboard. Вызывает gaddr() с параметрами «g_hud» и scenedata.hp.res. Соединяет сигнал «game_session_changed» от NetShared с методом «interpret_session» класса по указателю scoreboard.

– Init_MainMenu(). завершает клиентскую и серверную сетевые сессии, вызывая NetServer.Destroy(), если CurMode равен Mode.SERVER и NetClient.Destroy(), если CurMode равен Mode.CLIENT. Вызывает методы ClearEnv() и GameScenesUnload(). Подгружает класс по пути scenedata.main.path и присваивает адрес на него ключу scenedata.main.res. Вызывает метод gaddr() с параметрами «g_menus» и scenedata.main.res. Очищает указатель scenedata.main.res. Присваивает CurMode значение Mode.NONE.

Был разработан следующий метод для создания текстовых уведомлений в верхнем левом углу экрана из любого класса в связи с глобальностью настоящего класса:

– func Note(string). обращается к классу по указателю «notifier» и при

возможности вызывает метод «push_note» с параметром «string». Так же пишет параметр string в консоль отладки.

При собственной инициализации Glb присваивает оглавлению окна игры значение константы «softname» и связывает сигнал «client_disconnected» от класса NetClient с собственным методом «_on_client_disconnected».

Метод «_on_client_disconnected()» вызывает метод Init_MainMenu().

3.2.2 Сетевые классы (NetShared, NetServer, NetClient)

NetShared.

Файл с исходным кодом: «./net/net_shared.gd» [9].

Наследует от абстрактного класса «Node».

Объявляет следующие переменные:

– Address строкового типа, инициализируется пустой строкой;

– Port целочисленного типа, инициализируется значением -1;

– MaxPly целочисленного типа, инициализируется значением -1;

– mapname строкового типа, инициализируется значением «map_simple»;

– MaxKills целочисленного типа, инициализируется значением 5;

– game_session типа ассоциативный массив, инициализируется пустым ассоциативным массивом. Эталон заполнения этого массива:

{

1: { “name” : “Unnamed”, “kills” : 0, “deaths” : 0, “wins” : 0 }

...

};

– snapshot типа ассоциативный массив, инициализируется пустым ассоциативным массивом.

Объявляет сигнал game_session_changed с параметром session и сигнал

game_lock.

Методы настоящего класса и подробное описание их алгоритмов:

– game_session_alert(). Вызывает сигнал game_session_changed с параметром game_session;

– remote msg(s : String). Вызывает метод Glb.Note с параметром s. Если сеть работает в режиме сервера, то всем игрокам, зафиксированным в переменной game_session, кроме самого сервера, отправляется RPC запрос «msg» с параметром s с учетом потери пакетов;

– remote lock_game(). Проходит по наследникам экземпляра класса, на который указывает Glb.g_persistent, и выполняет относительно каждого из них метод set_network_master(-1). Если сеть работает в режиме сервера, то всем игрокам, зафиксированным в переменной game_session, кроме самого сервера, отправляется RPC запрос «lock_game»;

– remote unlock_game(). Проходит по наследникам экземпляра класса, на который указывает Glb.g_persistent, и удаляет из из перечня наследников и из памяти программы. Если сеть работает в режиме сервера, то всем игрокам, зафиксированным в переменной game_session, кроме самого сервера, отправляется RPC запрос «unlock_game»;

– remote game_session_clear_kd(). Проходит по всем ассоциативным массивам в game_session и присваивает ключам «kills» и «deaths» значение 0. Если сеть работает в режиме сервера, то всем игрокам, зафиксированным в переменной game_session, кроме самого сервера, отправляется RPC запрос «game_session_clear_kd». Вызывает метод game_session_alert();

– remote game_session_kill(var killer : int, var victim : int). Если игрок с идентификатором killer зафиксирован в переменной game_session, то берется значение ключа «kills» у соответствующего этому игроку ассоциативного массива в game_session и увеличивается на 1. Если игрок с идентификатором victim зафиксирован в переменной game_session, то берется значение ключа «deaths» у соответствующего этому игроку ассоциативного массива в

game_session и увеличивается на 1. Вызывает метод game_session_alert(). Если сеть работает в режиме сервера, то всем игрокам, зафиксированным в переменной game_session, кроме самого сервера, отправляется RPC запрос «game_session_kill» с параметрами killer и victim. Если вызывается этот модуль со стороны сервера и убийства у игрока killer достигли порога убийств, необходимого для побед, вызывается метод ply_won(killer) с параметром killer настоящего метода у экземпляра класса NetServer;

– game_session_adopt(var dic : Dictionary). Ме параметра dic переменной game_session. Вызывает метод game_session_alert(). Если переменная CurMode у Glb равна Glb.Mode.SERVER то вызывается метод game_session_send() у экземпляра класса NetServer;

– game_session_add(var id : int, var namecur : String, var kills : int, var deaths : int, var wins : int). Добавляет ассоциативный массив в переменную game_session с ключом id. У добавляемого ассоциативного массива следующие ключи: «name», значение которого соответствует параметру namecur, «kills», значение которого соответствует параметру kills, «deaths», значение которого соответствует параметру deaths, «wins», значение которого соответствует параметру wins. Вызывает метод game_session_alert(). Если переменная CurMode у Glb равна Mode.SERVER, то вызывается метод game_session_send() у экземпляра класса NetServer;

– game_session_rem(var id). Удаляет ассоциативный массив с ключом id из переменной типа ассоциативный массив game_session. Если переменная CurMode у Glb равна Mode.SERVER, то вызывается метод game_session_send() у экземпляра класса NetServer;

– get_map(). Обращается к экземпляру, на который указывает указатель g_world у Glb, находит и возвращает в перечне его наследников экземпляр с названием, соответствующим переменной mapname. Если не было найдено возвращает null;

– get_next_spawnpoint(). Объявляет внутреннюю переменную map,

инициализирует ее значением, возвращаемым вызовом метода get_map(). Если map не равен null и указывает на экземпляр, содержащий среди собственного перечня экземпляров-наследников экземпляр с названием «spawnpoints», то продолжает дальнейшее выполнение. В противном случае возвращает двухмерный вектор с x и y равными 0. Объявляет переменную n и присваивает ей экземпляр с названием «spawnpoints». Объявляет переменную c и присваивает ей количество экземпляров-наследников экземпляра «spawnpoints». Если переменная last_spawn_index у NetServer меньше нуля, то ей присваивается случайное значение от нуля до переменной c, не включительно. В противном случае если last_spawn_index меньше c, то last_spawn_index увеличивается на 1, иначе last_spawn_index присваивается значение 0. Возвращает двухмерный вектор, соответствующий позиции в игровом мире экземпляра-наследника по индексу равному last_spawn_index из массива со всеми экземплярами-наследниками экземпляра «spawnpoints»;

– create_ply_inst(id : int). Создает экземпляр класса игрока, присваивает ему название, соответствующее параметру id, вызывает относительно него метод set_network_master(id) и возвращает этот экземпляр;

– remote spawn_ply(var idtospawn : int, var pos : Vector2 = Vector2(0,0)). Объявляет переменную id, которой присваивается значение, соответствующее идентификатору вызвавшего этот метод пользователя с помощью вызова метода get_tree().get_rpc_sender_id(). Если id равно 1 (удаленный вызов от сервера), то объявляется переменная instance, ей присваивается значение, возвращаемое вызовом метода create_ply_inst с параметром idtospawn. Экземпляр класса, на который теперь указывает переменная instance, добавляется в перечень экземпляров-наследников относительно экземпляра, на который указывает g_persistent у Glb. Если id не равно 1, но при этом get_tree().get_network_unique_id() возвращает 1, что означает, что запущен режим сервера, то объявляется переменная spawnpoint, ей присваивается значение, возвращаемое вызовом метода

get_next_spawnpoint(). Объявляется переменная instance, ей присваивается значение, возвращаемое вызовом метода create_ply_inst(id) с параметром idtospawn. Экземпляр класса игрока, на который указывает instance, добавляется в перечень экземпляров-наследников относительно экземпляра, на который указывает g_persistent у Glb. Параметру global_position, отвечающему за положение в игровом мире, экземпляра класса игрока, на который указывает instance, присваивается значение переменной spawnpoint. Относительно каждого игрока в game_session, кроме самого сервера, выполняется RPC запрос на настоящий метод «spawn_ply» с параметрами idtospawn и spawnpoint;

– remote remove_peer(var idto : int). Объявляет переменную id и присваивает ей значение, возвращаемое вызовом get_tree().get_rpc_sender_id(). Если id равно 1 или 0 то все экземпляры-наследники с network_master соответствующим параметру idto у экземпляра, на который указывает Glb.g_persistent, удаляются из перечня экземпляров наследников и из памяти игры. Если id равен 0 то всем игрокам в game_session, кроме сервера, посылается RPC-запрос на настоящий метод с параметром idto;

– remote player_action(nodename : String, method : String). Алгоритм продолжает работу только в том случае, если параметры nodename и method имеют длину больше нуля и не начинаются с символа ‘_’. Объявляется переменная id, инициализируется значением, возвращаемым вызовом get_tree().get_rpc_sender_id(). Объявляется переменная myid, инициализируется значением, возвращаемым вызовом get_tree().get_network_unique_id(). Если id == 0 и myid > 1, то при наличии экземпляра с названием nodename в перечне экземпляров-наследников относительно экземпляра, на который указывает Glb.g_persistent, у экземпляра с названием nodename вызывается метод method при его наличии. В противном случае, если myid == 1 и id > 1 или id == 0, то выполняется все

то же самое, что и при первом условии, но сначала проверяется соответствие параметра network_master у экземпляра с названием nodename переменной id. Если не соответствует то выполнение прерывается. При соответствии еще выполняется относительно всех игроков в game_session, кроме сервера, RPC-запрос на настоящую функцию с параметрами nodename и method.

NetServer.

Файл с исходным кодом: «./net/net_server.gd» [9].

Наследует от абстрактного класса «Node».

Объявляет следующие переменные:

– Server динамического типа. Инициализируется пустым значением null;

– last_spawn_index целочисленного типа. Инициализируется значением -1;

– spawn_delay типа числа с плавающей запятой. Инициализируется значением 2.0;

– win_delay типа числа с плавающей запятой. Инициализируется значением 5.0;

– game_locked логического типа. Инициализируется значением false.

Объявляет сигнал server_created.

Методы настоящего класса и подробное описание их алгоритмов:

– _signals_connect(). Подключает сигналы network_peer_connected и network_peer_disconnected экземпляра класса «SceneTree» к методам _ply_connected(id) и _ply_disconnected(id) экземпляра класса NetServer соответственно;

– _signals_disconnect(). Отключает сигналы, подключенные в функции _signals_connect();

– _ready(). Вызывает set_physics_process(false) при загрузке экземпляра для отключения выполнения _physics_process(_delta);

– _physics_process(_delta). Выполняется 60 раз в секунду. Если

Glb.CurMode == Glb.Mode.SERVER то вызывает snapshot_compile() и snapshot_send_unreliable();

– ply_won(var id : int). Если id в game_session, то game_locked присваивается значение false. Параметру refuse_new_connections экземпляра, на который указывает Server, присваивается значение true. Вызывается метод lock_game() у NetShared. Вызывается метод msg(s) у NetShared, где s = NetShared.game_session[id]["name"] + " won by scoring " + str(NetShared.MaxKills) + " kills!". Метод str() превращает другие типы в строковые. Оператор + на строковых переменных это операция конкатенации. У игрока с идентификатором id в game_session увеличивается значение с ключом «wins» на 1. Вызывается метод game_session_send(). Создается таймер, ожидающий win_delay секунд. После этого game_locked = false, Server.refuse_new_connections = false, вызываются методы game_session_clear_kd() и unlock_game() у NetShared;

– Create(Address : String, Port : int, MaxPly : int, Listen : bool). CurMode у Glb присваивается значение Mode.SERVER. Вызывается _signals_connect(). Значениям Address, Port и MaxPly у NetShared присваиваются одноименные параметры. Создается экземпляр «NetworkedMultiplayerENet» и Server становится указателем на него. Параметру server_relay новосозданного экземпляра присваивается значение false. Вызывается метод Server.set_bind_ip() с параметром NetShared.Address. Создается переменная error целочисленного типа. Если Listen == true, то переменной error присваивается значение, возвращаемое вызовом Server.create_server() с параметрами NetShared.Port и NetShared.MaxPly – 1. Иначе вызывается этот же метод, но с параметрами NetShared.Port и NetShared.MaxPly, и возвращаемое значение так же присваивается error. Если error > 0, то возникла ошибка. Вызывается метод Note() у Glb с параметром, равным «"ERROR " + str(error) + ": Can't create server, destroying it"». Вызывается метод Destroy() и прекращается дальнейшее выполнение функции,

возвращается error. Параметр network_peer у SceneTree становится указателем на экземпляр «NetworkedMultiplayerENet», вызвав get_tree().network_peer = Server. Вызывается Note у Glb с параметром «"Server created as " + NetShared.Address + ":" + str(NetShared.Port) + ", unique id: \"" + str(get_tree().get_network_unique_id()) + "\""». Вызывается set_window_title() у OS с параметром "Server " + NetShared.Address + ":" + str(NetShared.Port). Запускается _physics_process, вызвав set_physics_process(true). Вызывается сигнал server_created. Если Listen равен true, то вызывается game_session_add у NetShared с параметрами get_tree().get_network_unique_id(), str(Glb.PlyName), 0, 0 и 0. Вызывается queue_spawn() с параметром, равным значению, возвращаемым вызовом get_tree().get_network_unique_id();

– Destroy(). CurMode у Glb присваивается значение Mode.NONE. Вызывается set_physics_process(false). Если Server не равен null то вызывается метод close_connection() у экземпляра, на который он указывает. Указатель network_peer у SceneTree обнуляется вызовом get_tree().network_peer = null. Вызывается Glb.Note с параметром "Server destroyed". Вызывается метод _signals_disconnect(). Вызывается OS.set_window_title с Glb.softname как параметром;

– _ply_connected(id). В консоль отладки пишет "Player with ID \"" + str(id) + "\" is connecting";

– _ply_disconnected(id). Вызываются метод msg() у NetShared с параметром "\"" + NetShared.game_session[id]["name"] + "\" (" + str (id) + ") left", game_session_rem() у NetShared с параметром id, remove_peer() у NetShared с параметром id, game_session_send().

– node_to_dict_child(n : Node, dic : Dictionary). Рекурсивно проходит по каждому наследнику экземпляра n, начиная с него самого. Создает ассоциативный массив с ключом «default» в ассоциативном массиве dic. Если у n есть параметры «global_position», «global_rotation», «position», «rotation», то относительно каждого из них создаются одноименные ключи в

ассоциативном массиве «default» со значениями, равными значениям этих параметров. Если есть еще наследники, создается ассоциативный массив с ключом «children» в dic. Относительно каждого наследника, название которого не начинается на ‘@’, создается ассоциативный массив с ключом, соответствующим названию наследника, в ассоциативном массиве с ключом «children» и вызывается метод node_to_dict_child(), первый параметр — указатель на наследника, второй параметр — указатель на ассоциативный массив с ключом, соответствующим названию наследника, в ассоциативном массиве с ключом «children»;

– node_to_dict(n : Node, dic : Dictionary). Создает ключ «path» у ассоциативного массива, переданного указателем dic, с значением равным файловому пути к классу экземпляра n. Создается ключ «netmaster»у dic, ему присваивается значение параметра network_master у экземпляра n вызовом n.get_network_master(). Создается ключ «default» в dic со значением пустого ассоциативного массива. Если у экземпляра n есть параметр, называемый «global_position», то создается ключ «global_position» со значением, равным значению этого параметра, у ассоциативного массива с ключом «default». Точно так же с параметрами «global_rotation», «position», «rotation». Создается переменная scr, ей присваивается экземпляр скрипта экземпляра n, вызвав n.get_script(). Если он не пустой, то есть, если класс, экземпляром которого n является имеет скрипт, то создается ключ «script» в dic с значением пустого ассоциативного массива. Относительно каждой переменной, объявленной в скрипте и исходном коде класса экземпляра n, не начинающаяся на символ “_”, записывается в ассоциативный массив с ключом «script» значение переменной с ключом, равным названию переменной. Если наследников у экземпляра n больше нуля, то создается ассоциативный массив с ключом «children» в dic. Относительно каждого наследника экземпляра n, у которого не начинается название на «@», создается ассоциативный массив с ключом, соответствующим названию

наследника, в ассоциативном массиве «children». Вызывается метод node_to_dict_child(), где первый параметр название это указатель на экземпляр-наследник, второй параметр это указатель на ассоциативный массив с названием экземпляра в массиве с ключом «children»;

– snapshot_compile(). Присваивает snapshot у NetShared пустой ассоциативный массив. Относительно всех экземпляров-наследников у экземпляра, на который указывает Glb.g_persistent, выполняется следующее: создается ассоциативный массив с ключом соответствующим названию экземпляра в snapshot, вызывается метод node_to_dict(), первый параметр — указатель на экземпляр-наследник, второй — указатель на только что созданный ассоциативный массив с названием экземпляра;

– snapshot_send_unreliable(). Относительно каждого игрока в game_session у NetShared, кроме сервера, вызывает RPC-запрос без учета потери пакетов на метод «snapshot_receive» у NetClient. Параметр равен snapshot у NetShared;

– game_session_send(). Создает ассоциативный массив dict, в котором ключ «game_session» равен game_session у NetShared, "maxply" равен MaxPly у NetShared, "maxkills" равен MaxKills у NetShared. Относительно всех игроков в game_session, кроме сервера, вызывает RPC-запрос с учетом потери пакетов на метод «game_session_receive» у NetClient, где параметр равен массиву dict;

– queue_spawn(id). С задержкой, равной spawn_delay секунд, выполняет следующее: если game_locked равен false вызывает метод spawn_ply() у NetShared с параметром id.;

– remote client_data_receive(var client_data : Dictionary). Объявляет переменную id со значением get_tree().get_rpc_sender_id(). Если в параметре client_data типа ассоциативный массив есть ключ «plyname» и его длина больше нуля и меньше 10 то вызывается метод game_session_add() у NetShared с параметрами id, значение ключа «plyname», 0, 0 и 0. Вызывается

метод msg у NetShared с параметром "\"" + str(client_data["plyname"]) + "\" (" + str (id) + ") joined". Вызывается queue_spawn с параметром id;

– clientstate_apply(var n : Node, var snapshot : Dictionary). Рекурсивно проходится по экземпляру n и по всем его наследникам. Относительно каждой итерации присваивает значения с ключами «global_position», «global_rotation», «position», «rotation» одноименным параметрам экземпляра n;

– remote clientstate_receive(var clientstate : Dictionary). Объявляет переменную id со значением get_tree().get_rpc_sender_id(). Если id находится в game_session, то относительно каждого ключа первого уровня в ассоциативном массиве clientstate проверяется, есть ли экземпляр-наследник у экземпляра, на который указывает Glb.g_persistent, с названием равным этому ключу. Если есть, то объявляется переменная n, указывающая на этот экземпляр с названием, равным ключу. Если у этого экземпляра параметр network_master равен id, что проверяется вызовом n.get_network_master() == id, то вызывается clientstate_apply(), где первый параметр равен n, второй это указатель на ассоциативный массив с ключом-названием в ассоциативном массиве clientstate.

NetClient.

Файл с исходным кодом: «./net/net_client.gd» [9].

Наследует от абстрактного класса «Node».

Объявляет следующие переменные:

– Client. Динамический тип. Инициализируется пустым значением null;

– clientstate_buffer. Тип ассоциативного массива, инициализируется пустым ассоциативным массивом;

Объявляет сигналы client_connected, client_failed, client_disconnected, game_session_received.

Методы настоящего класса и подробное описание их алгоритмов:

– _signals_connect(). Подключает сигналы connected_to_server,

connection_failed и server_disconnected экземпляра класса «SceneTree» к методам _connected_ok(), _connected_fail() и _server_disconnected() экземпляра класса NetClient соответственно;

– _signals_disconnect(). Отключает сигналы, подключенные в функции _signals_connect();

– _ready(). Вызывает set_physics_process(false) при загрузке экземпляра для отключения выполнения _physics_process(_delta);

– _physics_process(_delta). Выполняется 60 раз в секунду. Если Glb.CurMode == Glb.Mode.CLIENT то вызывает clientstate_compile() и clientstate_send();

– clientstate_compile_node_child(n : Node, dic : Dictionary). Рекурсивно проходит по каждому наследнику экземпляра n, начиная с него самого. Если у n есть параметры «global_position», «global_rotation», «position», «rotation», то относительно каждого из них создаются одноименные ключи в ассоциативном массиве dic со значениями, равными значениям этих параметров. Если есть еще наследники, создается ассоциативный массив с ключом «children» в dic. Относительно каждого наследника, название которого не начинается на ‘@’, создается ассоциативный массив с ключом, соответствующим названию наследника, в ассоциативном массиве с ключом «children» и вызывается метод clientstate_compile_node_child(), первый параметр — указатель на наследника, второй параметр — указатель на ассоциативный массив с ключом, соответствующим названию наследника, в ассоциативном массиве с ключом «children»;

– clientstate_compile_node(n : Node, dic : Dictionary). Если у экземпляра n есть такие параметры, как «global_position», «global_rotation», «position» и «rotation», то в dic создаются одноименные ключи и им присваиваются значения этих параметров. Если у n есть параметр «filename», то в dic создается ключ «path» и ему присваивается значение этого параметра. Если у экземпляра n есть наследники, то в dic создается ассоциативный массив с

ключом «children». В отношении каждого из наследников, название которых не начинается на символ ‘@’, в ассоциативном массиве «children» создается ассоциативный массив с ключом, равным названию наследника и вызывается метод clientstate_compile_node_child(), где первый параметр — указатель на наследника, второй параметр — указатель на новосозданный ассоциативный массив;

– clientstate_compile(). Присваивает переменной clientstate_buffer пустой ассоциативный массив. Относительно каждого экземпляра-наследника у экземпляра, на который указывает Glb.g_persistent, выполняется сравнение параметра network_master с значением, возвращаемым вызовом get_tree().get_network_unique_id(). Если они равны, то в clientstate_buffer создается ассоциативный массив с ключом, равным названию этого экземпляра-наследника и вызывается метод clientstate_compile_node(), первый параметр — указатель на наследующий экземпляр, второй — указатель на ассоциативный массив с ключом, соответствующим названию экземпляра-наследника;

– clientstate_send(). RPC-вызов на сервер относительно метода «clientstate_receive» у NetServer, передает как параметр clientstate_buffer;

– Create(Address : String, Port : int). Присваивает переменной CurMode у Glb значение Mode.CLIENT. Вызывает метод _signals_connect(). Значениям Address и Port у NetShared присваиваются одноименные параметры. Создается экземпляр «NetworkedMultiplayerENet» и переменная Client становится указателем на него. Объявляется переменная err, ей присваивается значение, возвращаемое вызовом Client.create_server() с параметрами Address у NetShared и Port у NetShared. Параметру network_peer у SceneTree присваивается значение переменной Client вызовом get_tree().network_peer = Client. Вызывается метод Note у Glb с параметром "CLIENT created (" + str(err) + "), unique id: \"" + str(get_tree().get_network_unique_id()) + "\"". Вызывается метод

set_window_title у OS с параметром «"Client " + str(get_tree().get_network_unique_id())»;

– Destroy(). CurMode у Glb присваивается значение Mode.NONE. Вызывается set_physics_process(false). Если Client не равен null то вызывается метод close_connection() с параметром 0 у экземпляра, на который он указывает. Указатель network_peer у SceneTree обнуляется вызовом get_tree().network_peer = null. Вызывается Glb.Note с параметром "Client destroyed". Вызывается метод _signals_disconnect(). Вызывается OS.set_window_title с Glb.softname как параметром;

– _connected_ok(). Вызывает сигнал client_connected. Вызывает метод Note у Glb с параметром "SUCCESS: Connected to server as a client!". Создает переменную client_data типа ассоциативный массив и присваивает ей значение PlyName у Glb с ключом «plyname». Делает RPC-запрос на сервер в отношении метода client_data_receive() у NetServer с параметром client_data. Включает выполнение метода _physics_process вызовом set_physics_process(true);

– _connected_fail(). Вызывает сигнал client_failed. Вызывает метод Destroy(). Вызывает метод Note у Glb с параметром "Error: Connection failed";

– _server_disconnected(). Вызывает сигнал client_disconnected. Вызывает метод Destroy(). Вызывает метод Note у Glb с параметром "Server disconnected us";

– snapshot_node_update_default_only(var n : Node, var snapshot : Dictionary). Рекурсивный метод. Если в параметре типа ассоциативный массив snapshot есть ключ «default», то на каждый ключ в ассоциативном массиве с ключом «default» берется параметр у экземпляра n, соответствующий названию ключа в массиве «default», и присваивается ему значение ключа в массиве «default». Если в snapshot есть ключ «children» и у n есть экземпляры-наследники, то относительного каждого из экземпляров-наследников выполняется метод snapshot_node_update_default_only(), где

первый параметр это указатель на наследующий экземпляр, второй параметр это указатель на ключ в массиве «children», соответствующий названию наследующего экземпляра. Вызывается метод только если название экземпляра-наследника не начинается на символ ‘@’ и его название есть в массиве с ключом «children»;

– snapshot_node_update_default_excl(var n : Node, var snapshot : Dictionary). Рекурсивный метод. Если в snapshot есть ключ «script», то метод проходится по каждому ключу в ассоциативном массиве с ключом «script» и присваивает значения этих ключей параметрам экземпляра n с названиями, соответствующими этим ключам. Если в snapshot есть ключ «children» и у n есть экземпляры-наследники, то относительного каждого из экземпляров-наследников выполняется метод snapshot_node_update_default_excl(), где первый параметр это указатель на наследующий экземпляр, второй параметр это указатель на ключ в массиве «children», соответствующий названию наследующего экземпляра. Вызывается метод только если название экземпляра-наследника не начинается на символ ‘@’ и его название есть в массиве с ключом «children»;

– snapshot_node_update(var n : Node, var snapshot : Dictionary). Рекурсивный метод. Если в snapshot есть ключ «default», то метод проходится по каждому ключу в ассоциативном массиве с ключом «default» и присваивает значения этих ключей параметрам экземпляра n, где названия параметров соответствуют этим ключам. Если в snapshot есть ключ «script», то метод проходится по каждому ключу в ассоциативном массиве с ключом «script» и присваивает значения этих ключей параметрам экземпляра n, где названия параметров соответствуют этим ключам. Если в snapshot есть ключ «children» и у n есть экземпляры-наследники, то относительного каждого из экземпляров-наследников выполняется метод snapshot_node_update(), где первый параметр это указатель на наследующий экземпляр, второй параметр это указатель на ключ в массиве «children», соответствующий названию

наследующего экземпляра. Вызывается метод только если название экземпляра-наследника не начинается на символ ‘@’ и его название есть в массиве с ключом «children»;

– snapshot_node_new(var title : String, var snapshot : Dictionary). Объявляет переменную строкового типа index, инициализирует ее пустой строкой. Цикленно проходится по всем ключам в переменной scenedata у Glb. Если находится ключ, в значении типа ассоциативный массив которого есть ключ «path», равный значению ключа «path» в параметре snapshot, то переменной index присваивается соответствующий ключ из scenedata и цикл прекращается. Создается экземпляр на основе класса в ключе «res» у массива с ключом «index», переменной instance присваивается указатель на этот экземпляр. Экземпляру присваивается название со значением параметра title, параметру network_master у экземпляра присваивается значение ключа «netmaster» у snapshot вызовом метода instance.set_network_master() с параметром ключа «netmaster» у snapshot. Вызывается метод snapshot_node_update_default_excl(), где первый параметр это переменная instance, второй это snapshot. Вызывается метод gaddi у Glb, где первый параметр это строка «g_persistent», второй параметр это переменная instance. Вызывается метод snapshot_node_update_default_only(), где первый параметр это переменная instance, второй это snapshot;

– remote snapshot_receive(var snapshot : Dictionary). snapshot у NetShared присваивается копия массива, переданного параметром snapshot. Названия каждого наследующего экземпляра у экземпляра, на который указывает Glb.g_persistent, проверяются на наличие соответствующих ключей в snapshot. Если не было найдено соответствующих ключей, то при наличии у наследующего экземпляра метода die() вызывается он. Если метода нет, то наследующий экземпляр удаляется из перечня наследующих экземпляров и из памяти. Теперь проверяется каждый ключ в ассоциативном массиве snapshot. Если в наследующих экземплярах у экземпляра, на который

указывает Glb.g_persistent, нет ни одного экземпляра с названием, соответствующим ключу из snapshot, то относительно этого ключа вызывается метод snapshot_node_new(), где первый параметр это ключ, второй это его значение типа ассоциативный массив. Относительно тех ключей, у которых есть соответствующие экземпляры, вызывается метод snapshot_node_update_default_excl(), где первый параметр это указатель на экземпляр, второй это указатель на значение ключа. Этот метод вызывается, если ключ «netmaster» в значении ключа с соответствующим наследником соответствует идентификатору настоящего клиента, который получается вызовом get_tree().get_network_unique_id(). В противном случае вызывается метод snapshot_node_update(), где первый параметр это указатель на экземпляр, второй это указатель на значение ключа;

– remote game_session_receive(var game_session : Dictionary). Если вызов get_tree().get_rpc_sender_id() возвращает 1, то есть был совершен RPC-вызов этой функции со стороны сервера, то MaxKills у NetShared присваивается значение ключа «maxkills» у параметра game_session, MaxPly у NetShared присваивается значение ключа «maxply» у параметра game_session. Вызывается метод game_session_adopt() у NetShared, как параметр ему передается значение ключа «game_session» у параметра game_session. Вызывается сигнал game_session_received.

3.2.3 Вспомогательные классы (env, g_*, fetchserver)

env.

Файл с исходным кодом: «./groups/env.gd». Конфигурационный файл класса: «./groups/env.tscn» [9].

Метод настоящего класса и подробное описание его алгоритма:

– _ready(). Выполняется при создании экземпляра. Указателю env у Glb присваивается адрес настоящего экземпляра. Вызывается глобальный метод

randomize(), который задает случайный seed на основе времени операционной системы у генератора случайных чисел движка.

g_hud.

Файл с исходным кодом: «./groups/g_hud.gd» [9].

Метод настоящего класса и подробное описание его алгоритма:

– _ready(). Выполняется при создании экземпляра. Указателю g_hud у Glb присваивается адрес настоящего экземпляра.

g_menus.

Файл с исходным кодом: «./groups/g_menus.gd» [9].

Метод настоящего класса и подробное описание его алгоритма:

– _ready(). Выполняется при создании экземпляра. Указателю g_menus у Glb присваивается адрес настоящего экземпляра.

g_notifiers.

Файл с исходным кодом: «./groups/g_notifiers.gd» [9].

Метод настоящего класса и подробное описание его алгоритма:

– _ready(). Выполняется при создании экземпляра. Указателю g_notifiers у Glb присваивается адрес настоящего экземпляра.

g_persistent.

Файл с исходным кодом: «./groups/g_persistent.gd» [9].

Метод настоящего класса и подробное описание его алгоритма:

– _ready(). Выполняется при создании экземпляра. Указателю g_persistent у Glb присваивается адрес настоящего экземпляра.

g_world.

Файл с исходным кодом: «./groups/g_world.gd» [9].

Метод настоящего класса и подробное описание его алгоритма:

– _ready(). Выполняется при создании экземпляра. Указателю g_world у Glb присваивается адрес настоящего экземпляра.

fetchserver.

Файл с исходным кодом: «./auxilliary/fetchserver.gd».

Конфигурационный файл класса: «./auxilliary/fetchserver.tscn» [9].

Объявляет сигнал got_something с параметром addressnport.

Методы настоящего класса и подробное описание их алгоритмов:

– _ready(). Выполняется при создании экземпляра. Соединяет сигнал request_completed экземпляра-наследника типа HTTPRequest внутреннему методу _on_request_completed;

– get_main_address(link : String). Совершает HTTP запрос с помощью HTTPRequest по URL с параметра link;

– _on_request_completed(result, response_code, headers, body). Объявляет переменную magic строкового типа, присваивает ей значение "0az-610dwe0d234880". Объявляет переменную bodystr строкового типа и присваивает ей значение, возвращаемое вызовом body.get_string_from_ascii(). Этот метод возвращает строку на основе массива ASCII значений. На основе значения bodystr и магического слова в переменной magic начинает искать адрес. Если адрес и порт найдены, то они декодируется. Подразумевается, что по ссылке адрес после магического слова закодирован смещением ASCII значений у символов. Декодировав адрес и порт они передаются как параметр один строковый параметр в вызове сигнала got_something, эталон строки: "127.0.0.1:27015".

3.2.4 Текстовые уведомления (notifier)

Файл с исходным кодом: «./groups/env.gd». Конфигурационный файл класса: «./groups/env.tscn» [9].

Объявляет следующие переменные:

– cont. Динамический тип. Инициализируется значением «$HboxContainer/VboxContainer». Указатель на экземпляр класса «VBoxContainer», объявленный в конфигурации;

– timer. Динамический тип. Инициализируется значением «$Ticktock».

Указатель на экземпляр класса «Timer», объявленный в конфигурации;

– notifier_lbl. Динамический тип. Подгружает во время компиляции игры класс по пути "res://notifier/notifier_lbl.tscn";

– can_start. Логический тип. Используется для возобновления счетчика.

Объявляет следующие константы:

– maxnotes. Целочисленный тип. Инициализируется значением 6. Максимальное количество отображаемых и хранящихся текстовых уведомлений;

– timetoflush. Целочисленный тип. Инициализируется значением 4. Сколько времени в секундах необходимо для автоматического удаления самого старого текстового уведомления.

Методы настоящего класса и подробное описание их алгоритмов:

– _ready(). Выполняется при создании экземпляра настоящего класса. Присваивает настоящий экземпляр к указателю Glb.notifier. Соединяет сигнал timeout у экземпляра переменной timer с внутренним методом remove_first. Указывает timer, что нужно выполняться только один раз, без автоматических возобновлений, вызовом timer.set_one_shot(true). Указывает время длительности таймера равное константе timetoflush вызовом timer.set_wait_time(timetoflush);

– _process(_delta). Выполняется постоянно после создания экземпляра. Если timer закончил работу, can_remove() возвращает положительный логический тип и can_start равен true, то счетчик запускается снова вызовом timer.start();

– remove_first(). Переменной can_start присваивается значение false. Счетчик останавливается вызовом timer.stop(). Объявляется переменная n, указывающая на самый первый по счету экземпляр-наследник у переменной cont. Этот экземпляр посредством обращения к переменной удаляется из перечня экземпляров-наследников и из памяти игры. Переменной can_start присваивается значение true;

– _remove_first(). Объявляется переменная n, указывающая на самый первый по счету экземпляр-наследник у переменной cont. Этот экземпляр посредством обращения к переменной удаляется из перечня экземпляров-наследников и из памяти игры;

– push_note(string). Переменной can_start присваивается значение false. Счетчик останавливается вызовом timer.stop(). Создает экземпляр класса подгруженного в переменной notifier_lbl и передает указатель на него объявляемой переменной label. Присваивает экземпляру по указателю label значение параметра string переменной text у label. Если количекство экземпляров-наследников у cont равно или превышает maxnotes, то вызывается метод _remove_first(). Созданный экземпляр добавляется в перечень экземпляров-наследников переменной cont. Переменной can_start присваивается значение true;

– can_remove(). Возвращает логический тип. Делает сравнение, больше ли нуля количество экземпляров-наследников у переменной cont.

3.2.5 Классы главного и игрового меню

Элемент интерфейса tmpl_label.

Файл с исходным кодом: «.menus/templates/tmpl_label.gd». Конфигурационный файл класса: «.menus/templates/tmpl_label.tscn» [9].

Наследует от абстрактного класса Label.

Метод настоящего класса и подробное описание его алгоритма:

– _ready(). Выполняется при создании экземпляра настоящего класса. Проходится по унаследованному параметру строкового типа text начиная от самого первого символа и заканчивая последним, при этом ведет счетчик количества рассмотренных символов. Счетчик сбрасывается если попадается символ '\n'. Когда счетчик становится равным 30 на место в строке, где остановился алгоритм, добавляется символ '\n'. После этого продолжается

рассмотрение строки по такому же принципу пока не дойдет алгоритм до конца строки.

Главное меню (menu_main).

Файл с исходным кодом: «./menus/menu_main.gd». Конфигурационный файл класса: «./menus/menu_main.tscn» [9].

Наследует от абстрактного класса CanvasLayer.

Объявляет следующие переменные:

– maincontainer. Динамический тип. Указывает на экземпляр абстрактного класса VBoxContainer, объявленного в конфигурации. Инициализируется значением $HboxContainer/VboxContainer;

– item_default. Динамический тип. Указывает на экземпляр абстрактного класса VBoxContainer, объявленного в конфигурации. Инициализируется значением $HBoxContainer/VBoxContainer/item_default;

– item_back. Динамический тип. Указывает на экземпляр абстрактного класса VBoxContainer, объявленного в конфигурации. Инициализируется значением $HBoxContainer/VBoxContainer/item_back;

– item_connect. Динамический тип. Указывает на экземпляр абстрактного класса VBoxContainer, объявленного в конфигурации. Инициализируется значением $HBoxContainer/VBoxContainer/item_connect;

– item_connect_title. Динамический тип. Указывает на экземпляр класса tmpl_label_noscript, объявленного в конфигурации. Инициализируется значением $HBoxContainer/VBoxContainer/item_connect/LblTitle;

– item_connect_desc. Динамический тип. Указывает на экземпляр класса tmpl_label, объявленного в конфигурации. Инициализируется значением $HBoxContainer/VBoxContainer/item_connect/LblDesc;

– item_connect_butt. Динамический тип. Указывает на экземпляр класса tmpl_button, объявленного в конфигурации. Инициализируется значением $HBoxContainer/VBoxContainer/item_connect/ButtCancel;

– item_edit_plyname. Динамический тип. Указывает на экземпляр

класса tmpl_edit, объявленного в конфигурации. Инициализируется значением $HBoxContainer/VBoxContainer/item_plysettings/EditPlyName;

– fetchserver. Динамический тип. Указывает на экземпляр класса fetchserver, объявленного в конфигурации. Инициализируется значением $fetchserver.

Методы настоящего класса и подробное описание их алгоритмов:

– _ready(). Вызывает метод showdefault(). Связывает сигнал server_created экземпляра класса NetServer с внутренним методом _on_server_created(). Связывает сигналы game_session_received и client_failed экземпляра класса NetClient с внутренними методами _game_session_received() и _on_client_failed() соответственно. Связывает сигнал got_something переменной fetchserver с внутренним методом _got_something;

– showitem(name : String). Ищет среди экземпляров-наследников переменной maincontainer экземпляр с названием "item_" + name. Если он был найден то все экземпляры-наследники переменной maincontainer с префиксом "item_" в названии скрываются, найденный экземпляр показывается вызовом maincontainer.get_node("item_" + name).show(), экземпляр по указателю переменной item_back показывается вызовом item_back.show();

– showdefault(). Скрывает все экземпляры-наследники переменной maincontainer с префиксом "item_" в названии кроме "item_default";

– showconnect(title : String, desc : String, butt : String). Скрывает Скрывает все экземпляры-наследники переменной maincontainer с префиксом "item_" в названии кроме "item_connect", параметрам text переменных item_connect_title, item_connect_desc и item_connect_butt присваиваются параметры настоящего метода title, desc и butt соответственно;

– _on_ButtBack_pressed(). Вызывает метод showdefault();

– _on_ButtHost_pressed(). Вызывает метод showitem() с параметром

"host";

– _on_ButtJoin_pressed(). Вызывает метод showitem() с параметром "join";

– _on_ButtAbout_pressed(). Вызывает метод showitem() с параметром "about";

– _on_ButtQuit_pressed(). Вызывает метод get_tree().quit(), завершающий работу игры;

– _on_ButtHostServer_pressed(). Проверяет параметр text у экземпляров полей ввода по путям "item_host/EditHostAddr", "item_host/EditHostPort" и "item_host/EditMaxPly". Если длина у каждого больше нуля то вызывается метод showconnect() с тремя строковыми параметрами "CREATING", «"Creating " + EditAddress.text + ":" + EditPort.text» и "Cancel", после чего вызывается метод Create() экземпляра класса NetServer, куда передаются параметры text рассмотренных полей ввода. Как последний параметр Listen у функции передается false;

– _on_server_created(). Вызывает метод Init_GameState() у Glb;

– _on_ButtJoinServer_pressed(). Проверяет параметр text у экземпляров полей ввода по путям "item_join/EditJoinAddr" и "item_join/EditJoinPort". Если длина у каждого больше нуля то вызывается метод showconnect() с тремя строковыми параметрами "CONNECTING", «"Establishing connection with " + EditAddress.text + ":" + EditPort.text» и "Cancel", после чего вызывается метод Create() экземпляра класса NetClient, куда передаются параметры text рассмотренных полей ввода;

– _game_session_received(). Вызывает метод Init_GameState() у Glb;

– _on_client_failed(). Вызывает метод showdefault();

– _on_ButtCancel_pressed(). Вызывает метод Destroy() у экземпляров классов NetServer и NetClient. Два раза вызывает метод gclr() у Glb с параметрами «g_world» и «g_persistent». Вызывает метод GameScenesUnload() у Glb. Вызывает внутренний метод showdefault().

– _on_ButtPlySettings_pressed(). Вызывает метод showitem() с параметром "plysettings";

– _on_ButtSave_pressed(). Если у переменной item_edit_plyname параметр текста имеет длину больше нуля и меньше десяти то значение этого параметра присваивается переменной PlyName у Glb и вызывается метод Note у Glb с параметром "Player Settings Saved!". В противном случае параметру text у переменной item_edit_plyname присваивается пустая строка как значение и вызывается метод Note у Glb с параметром "Incorrect name, try again";

– _on_ButtHostQuick_pressed(). Вызывает метод showconnect() с параметрами "CREATING DEFAULT SERVER", "Creating defult dedicated setup" и "Cancel". Вызывает метод Create у экземпляра класса NetServer с параметрами "192.168.0.100", 27015, 5 и false;

– _on_ButtJoinQuick_pressed(). Вызывает метод showconnect() с параметрами "CREATING DEFAULT SERVER", "Creating defult dedicated setup" и "Cancel". Вызывает метод get_main_address() у переменной fetchserver с параметром "https://vk.com/tofetchanaddress";

– _got_something(addressnport : String). Если item_connect виден (имеет параметр visible равным true), то переданная строка разделяется на две вызовом строковой функции split(‘:’) и если получилось две строки, то они передаются как параметры методу Create у NetClient. В противном случае вызывается метод Glb.Note с параметром "Failed fetching Home PC" и внутренний метод showdefault();

– _on_ButtHostServerListen_pressed(). То же самое, что и _on_ButtHostServer_pressed(), только как последний параметр Listen у функции создания сервера передается true.

Игровое меню (menu_ingame).

Файл с исходным кодом: «./menus/menu_ingame.gd». Конфигурационный файл класса: «./menus/menu_ingame.tscn» [9].

Наследует от абстрактного класса CanvasLayer.

Объявляет следующую переменную:

– control. Динамический тип. Указывает на экземпляр абстрактного класса Control, объявленного в конфигурации. Инициализируется значением $Control.

Методы настоящего класса и подробное описание их алгоритмов:

– _ready(). Скрывает экземпляр по переменной control и всех его наследников вызвав control.hide();

– _process(delta). Если была нажата клавиша «Escape» то показать или спрятать экземпляр по переменной control, представляющий из себя элементы меню;

– _on_ButtLeave_pressed(). Вызывает метод Init_MainMenu() у экземпляра класса Glb.

3.2.6 Игровая логика и игровой интерфейс

Игрок (player_full).

Файл с исходным кодом: «./actors/player_full.gd». Конфигурационный файл класса: «./actors/player_full.tscn» [9].

Наследует от абстрактного класса KinematicBody2D.

Объявляет следующие переменные:

– _bullet. Динамический тип. Подгружает во время компиляции игры класс по пути "res://actors/player_bullet.tscn";

– _exp. Динамический тип. Подгружает во время компиляции игры класс по пути "res://actors/effects/explosion.tscn";

– _exp_snd. Динамический тип. Подгружает во время компиляции игры класс по пути "res://actors/effects/explosion_sound.tscn";

– _body. Динамический тип. Указывает на экземпляр абстрактного класса Sprite, объявленного в конфигурации. Инициализируется значением

$sprite_body. Это тело танка-игрока;

– _turret. Динамический тип. Указывает на экземпляр абстрактного класса Sprite, объявленного в конфигурации. Инициализируется значением $sprite_turret. Это турель танка-игрока;

– _gunpos. Динамический тип. Указывает на экземпляр абстрактного класса Position2D, объявленного в конфигурации. Инициализируется значением $sprite_turret/Position2D;

– _prevpos. Тип двухмерного вектора. Инициализируется двухмерным вектором с нулевыми координатами;

– _curcolor. Тип Color, трехмерный вектор. Инициализируется значением Color(1, 1, 1);

– _lastshoot. Целочисленный тип. Инициализируется значением 0. Последнее зафиксированное время выстрела в миллисекундах;

– _shootdelay. Целочисленный тип. Инициализируется значением 60. Длительность задержки после выстрела перед следующим выстрелом в миллисекундах;

– _lastreg. Целочисленный тип. Инициализируется значением 0. Последнее зафиксированное время когда попадали по игроку. Используется для визуального отображения попадания по игроку;

– _hitduration. Целочисленный тип. Инициализируется значением 300. Длительность отображения эффекта попадания по игроку в миллисекундах;

– _defhealth. Динамический тип. Инициализируется значением 100. Здоровье игрока по умолчанию;

– _lastdmgdealer. Целочисленный тип. Инициализируется значением -1. Последний игрок, нанесший настоящему экземпляру класса игрока урон;

– _alive. Логический тип. Инициализируется значением true. Вспомогательная переменная, показывающая жив ли игрок, нужна для обработки смерти;

– health. Динамический тип. Инициализируется значением _defhealth.

Это текущее здоровье игрока.

Объявляет следующие константы:

– _movespeed. Целочисленный тип. Инициализируется значением 600. Определяет скорость передвижение игрока;

– _initvec. Тип двухмерного вектора. Инициализируется вектором с координатами 0 по x и -1 по y. Направление этого вектора будет считаться изначальным положением графического отображения игрока, тела и турели;

– _allycolor. Тип Color, трехмерный вектор. Инициализируется значением Color(0.3, 0.3, 1). Определяет в какой цвет должен окраситься экземпляр класса при его создании если его владельцем является игрок, от лица которого запущена игра;

– _enemycolor. Тип Color, трехмерный вектор. Инициализируется значением Color(1, 0.3, 0.3). Определяет в какой цвет должен окраситься экземпляр класса при его создании если его владельцем не является игрок, от лица которого запущена игра.

Методы настоящего класса и подробное описание их алгоритмов:

– _ready(). Вызывается при создании экземпляра класса. Присваивает переменной _prevpos значение параметра global_position, представляющего собой положение экземпляра игрока в игровом мире. Если параметр network_master равен идентификатору игрока, от лица которого рассматривается игра, то значению переменной _curcolor присваивается значение переменной _allycolor, в противном случае присваивается значение _enemycolor. Значение _curcolor применяется в отношении тела и турели экземпляра класса игрока, оно передается встроенному параметру modulate у тела и игрока;

– _process(delta). Если время с момента запуска игры в миллисекундах (метод OS.get_ticks_msec()) меньше _lastreg + _hitduration и больше _lastreg то экземпляр класса игрока окрашивается в белый цвет. Для этого объявляется переменная mult типа числа с плавающей точкой, которая

является разницой между возвращаемым значением OS.get_ticks_msec() и _lastreg, поделенная на _hitduration. Каждому из параметров modulate, а именно r, g и b присваивается результат выполнения max(mult, 1). Если условие не выполняется, то параметрам modulate тела и турели присваивается _curcolor. Если здоровье равно или меньше 0 и при этом _alive возвращает положительный логический тип то вызывается функция die();

_physics_process(delta). Если _is_in_control() возвращает true, то вызывается методы _handle_movement() с параметром delta и _handle_aim(). Если нажата левая кнопка мыши и _lastshoot меньше OS.get_ticks_msec() то вызывается метод shoot_bullet(). При этом если Glb.CurMode равен Mode.CLIENT, то совершается RPC-запрос в отношении сервера и метода player_action экземпляра класса NetShared, передается название экземпляра и строка "shoot_bullet". Если Glb.CurMode равен Mode.SERVER то просто вызывается метод player_action экземпляра класса NetShared, передается название экземпляра и строка "shoot_bullet". В конце вне условия вызывается внутренний метод _handle_body_rotation();

– take_damage(var dmg : int, var ownerid : int). Если Glb.CurMode равен Mode.SERVER то переменной health присваивается либо разница health – dmg, либо 0, в зависимости от того, что больше. Переменной _lastdmgdealer присваивается параметр метода ownerid. Если health меньше либо равно 0 и _alive положительный, то вызывается метод die(). При этом если Glb.CurMode равен Mode.SERVER то вызывается метод player_action экземпляра класса NetShared с параметрами названия экземпляра и строки "die". В конце метода вне условий присваивается переменной _lastreg значение OS.get_ticks_msec();

– die(). Если _alive положительный, _alive присваивается значение false. Переменной health присваивается значение 0. Создаются экземпляры классов из переменных _exp и _exp_snd, их положение в игровом мире устанавливается в соответствии с положением настоящего экземпляра

класса. Эти экземпляры добавляются как экземпляры-наследователи относительно экземпляра, на который указывает переменная Glb.g_world. Если Glb.CurMode равен Mode.SERVER, то вызывается метод queue_spawn() у экземпляра класса NetServer, передается значение параметра network_master, вызывается метод game_session_kill у экземпляра класса NetShared, передается переменная _lastdmgdealer и параметр настоящего экземпляра класса network_master. В конце метода вне условий настоящий экземпляр класса удаляется из перечня экземпляров-наследователей относительно наследуемого экземпляра и из памяти игры;

– _is_in_control(). Возвращает логический тип, являющийся результатом вызова встроенного метода is_network_master(), проверяющего, соответствует ли параметр network_master настоящего экземпляра класса идентификатору игрока, от лица которого рассматривается запущенная игра;

– shoot_bullet(). Если разница _lastshoot – 0.1 больше или равна OS.get_ticks_msec() то выполнение функции на этом прерывается. В противном случае переменной _lastshoot присваивается сумма OS.get_ticks_msec() + _shootdelay. Создается экземпляр класса снаряда на основе подгруженного класса в переменной _bullet, этому экземпляру задается положение в игровом мире соответствующее переменной _gunpos, задается вращение равное результату разницы _turret.global_rotation - (PI / 2). Параметру созданного экземпляра ownerid присваивается значение параметра network_master. Параметру экземпляра снаряда shooternode присваивается название настоящего экземпляра класса игрока. Этот экземпляр добавляется как наследник относительно экземпляра, на который указывает указатель Glb.g_world;

_handle_movement(delta). Объявляет переменную movevec типа двухмерного вектора. Координате x переменной movevec приравнивается разница между логическими методами, проверяющими, нажата ли клавиша D и нажата ли клавиша A, перед вычислением результата операции результаты

вызова этих логических методов конвертируются в целочисленный тип встроенным методом int(). То же самое с координатой y, но уже с клавишами S и W соответственно. Этот вектор нормализируется и умножается на _movespeed. Вызывается наследуемый метод move_and_slide() и ему передается как параметр вектор movevec;

_get_angle_btwn(vec1 : Vector2, vec2 : Vector2). Возвращает значение типа числа с плавающей точкой. Вычисляет разницу вектора vec1 и vec2 в радианах;

– _handle_aim(). Объявляет переменную mouse, ей присваивает разницу положения курсора мыши в игровом мире и положения экземпляра игрока. Объявляет переменную angle и присваивает ей значение, возвращаемое вызовом _get_angle_btwn() с параметрами mouse и _initvec. Если angle > 0, то вращению турели _turret присваивается значение angle, но перед этим проверяется mouse.x < 0. Если да, то angle умножается на -1;

– _handle_body_rotation(). Объявляет переменную difference типа двухмерный вектор, являющийся разницой между положением экземпляра класса игрока и переменной _prevpos. Если модули координат вектора difference больше нуля, то объявляется переменная angle со значением _get_angle_btwn(difference, _initvec), и если координата x вектора difference меньше нуля, то angle умножается на -1. Параметру вращения тела игрока _body присваивается значение angle. Вне условий в конце метода положение экземпляра игрока в игровом мире присваивается переменной _prevpos.

Смерть игрока, визуальное сопровождение (explosion).

Файл с исходным кодом: «./actors/effects/explosion.gd». Конфигурационный файл класса: «./actors/effects/explosion.tscn» [9].

Наследует от абстрактного класса AnimatedSprite.

Методы настоящего класса и подробное описание их алгоритмов:

– _ready(). Присваивает наследуемому параметру вращения rotation случайное значение, определяющееся вызовом fmod(randf(), PI * 2).

Вызывается наследуемый метод play();

– _on_AnimatedSprite_animation_finished(). Удаляет настоящий экземпляр класса из перечня наследуемых относительно наследуемого экземпляра и из памяти игры.

Смерть игрока, звуковое сопровождение (explosion_sound).

Файл с исходным кодом: «./actors/effects/explosion_sound.gd». Конфигурационный файл класса: «./actors/effects/explosion_sound.tscn» [9].

Наследует от абстрактного класса AudioStreamPlayer2D.

Методы настоящего класса и подробное описание их алгоритмов:

– _ready(). Вызывается наследуемый метод play();

– _on_explosion_sound_finished(). Удаляет настоящий экземпляр класса из перечня наследуемых относительно наследуемого экземпляра и из памяти игры.

Снаряд (player_bullet).

Файл с исходным кодом: «./actors/player_bullet.gd». Конфигурационный файл класса: «./actors/player_bullet.tscn» [9].

Наследует от абстрактного класса Sprite.

Объявляет следующие переменные:

– _hit. Динамический тип. Подгружает во время компиляции игры класс по пути "res://actors/effects/hit.tscn";

– _hit_snd. Динамический тип. Подгружает во время компиляции игры класс по пути "res://actors/effects/hit_sound.tscn";

– lifecycles. Инициализируется значением 1000. Время жизни;

– dmg. Инициализируется значением 5. Это урон наносимый снарядом;

– ownerid. Инициализируется значением 0. Идентификатор игрока, выстрелившего настоящим снарядом;

– shooternode. Строковый тип. Инициализируется пустой строкой. Название экземпляра класса, выстрелившего этим снарядом.

Объявляет следующую константу:

– speed. Инициализируется значением 25.

Методы настоящего класса и подробное описание их алгоритмов:

– _physics_process(delta). 60 раз в секунду прибавляет двухмерный вектор, сформированный вызовом функций косинуса и синуса, которым передаются радианы вращения экземпляра класса. Вектор умножается на speed. Если lifecycles <= 0 то экземпляр удаляется из памяти, иначе от lifecycles отнимается единица.

– produce_effects(). Создает экземпляры классов из переменных _hit и _hit_snd, устанавливает положение в мире этих экземпляров в соответствии с положением экземпляра настоящего класса и добавляет эти экземпляры как наследников относительно экземпляра, на который указывает Glb.g_world;

– _on_Area2D_body_entered(body). Если параметр name у body равен переменной shooternode то работа функции останавливается. В противном случае вызывается метод produce_effects() и экземпляр настоящего класса удаляется из памяти, но перед этим проверяется, имеется ли у body метод take_damage() и неравен ли параметр network_master у body переменной ownerid. Если проверка положительная, то вызывается метод take_damage, ему передаются переменные dmg и ownerid.

Попадание, визуальное сопровождение (hit).

Файл с исходным кодом: «./actors/effects/hit.gd». Конфигурационный файл класса: «./actors/effects/hit.tscn» [9].

Наследует от абстрактного класса AnimatedSprite. Методы описаны в explosion.

Попадание, звуковое сопровождение (hit_sound).

Файл с исходным кодом: «./actors/effects/explosion_sound.gd». Конфигурационный файл класса: «./actors/effects/hit_sound.tscn» [9].

Наследует от абстрактного класса AudioStreamPlayer2D. Методы описаны в explosion_sound.

Камера (devout_camera).

Файл с исходным кодом: «./actors/devout_camera.gd». Конфигурационный файл класса: «./actors/devout_camera.tscn» [9].

Наследует от абстрактного класса Camera2D.

Объявляет следующую переменную:

– _movespeed. Целочисленный тип. Инициализируется значением 10.

Метод настоящего класса и подробное описание его алгоритма:

– _process(_delta). Если Glb.CurMode равен Mode.NONE то выполнение функции прекращается. В противном случае объявляется переменная id равная идентификатору пользователя в сети (get_tree().get_network_unique_id()). Если id есть как ключ в NetShared.game_session, то камере назначается позиция в игровом мире, соответствующая игроку с некоторым отдалением в сторону курсора мыши. Отдаление курсора мыши равно примерно 160 игровых единиц. Если игрок мертв, то ему позволяется двигать карту камеры клавишами W, A, S и D по такому же принципу, как двигался и игрок, только без вызова метода move_and_slide. Вектор просто добавляется к вектору позиции камеры в игровом мире.

Отображение здоровья (hp_points).

Файл с исходным кодом: «./hud/hp_points.gd». Конфигурационный файл класса: «./hud/hp_points.tscn» [9].

Наследует от абстрактного класса HBoxContainer.

Объявляет следующую переменную:

– label. Динамический тип. Указывает на экземпляр класса tmpl_label_noscript, объявленного в конфигурации. Инициализируется значением $VBoxContainer/tmpl_label_noscript;

Объявляет следующие константы:

– placeholder. Строковый тип. Инициализируется значением "PENDING SPAWN";

– placeholder_dedicated. Строковый тип. Инициализируется

значением"SPECTATING".

Метод настоящего класса и подробное описание его алгоритма:

– _process(delta). Если есть экземпляр класса игрока с названием, соответствующим идентификатору игрока get_tree().get_network_unique_id(), в перечне наследователей экземпляра, на который указывает Glb.g_persistent, то параметру text переменной label присваивается значение переменной health экземпляра игрока. В противном случае, если идентификатор есть в массиве game_session у экземпляра класса NetShared как ключ, то параметру text присваивается значение константы placeholder. Если идентификатора там нет, то присваивается значение константы placeholder_dedicated.

Отображение таблицы счета (hud_scoreboard).

Файл с исходным кодом: «./hud/hud_scoreboard.gd». Конфигурационный файл класса: «./hud/hud_scoreboard.tscn» [9].

Наследует от абстрактного класса Control.

Объявляет следующие переменные:

– container. Динамический тип. Инициализируется значением $CanvasLayer/VboxContainer;

– playercount. Динамический тип. Инициализируется значением $CanvasLayer/VboxContainer/playercount;

– titles. Динамический тип. Инициализируется значением $CanvasLayer/VboxContainer/titles;

– players. Динамический тип. Инициализируется значением $CanvasLayer/VboxContainer/players;

– always_on. Логический тип. Инициализируется значением false.

Объявляет следующую константу:

– margins. Целочисленный тип. Инициализируется значением 20.

Методы настоящего класса и подробное описание их алгоритмов:

– _ready(). Всем наследникам экземпляра titles устанавливаются прибавляются значения margins относительно их параметров rect_size и

rect_size_min по координате x. Соединяется сигнал game_lock от экземпляра класса NetShared с внутренним методом toggle. Вызывается метод _calculate_margins(). Экземпляр container скрывается вызовом container.visible = false. Вызывается метод interpret_session() с параметром NetShared.game_session;

– _process(delta). Если always_on равен true, то экземпляр в переменной container показывается. В противном случае, если нажата клавиша «Tab», то container скрывается, если был показан, или показывается, если был скрыт;

– toggle(). Присваивает переменной always_on противоположное значение относительно того, которое имеется на момент вызова функции. Если false, то true. Если true, то false;

– _plycount(var session : Dictionary). Параметру text переменной playercount присваивается значение str(session.size()) + "/" + str(maxply) + ", score " + str(NetShared.MaxKills) + " to win!";

– _push(var id : int, var dic : Dictionary). Создается экземпляр класса в переменной titles. Его наследнику id присваивается значение ключа «id» из параметра dic. То же самое с «name», «kills», «deaths» и «wins». Этот экземпляр показывается и добавляется в перечень наследников экземпляра players;

– _clear_players(). Удаляет всех наследников экземпляра players из перечня наследников и из памяти игры;

– interpret_session(var session : Dictionary). Вызывает _clear_players(). Вызывает метод _push() с параметрами ключа и значения ключа относительно каждого ключа в ассоциативном массиве session. Вызывает методы _calculate_margins() и _plycount();

– _calculate_margins(). Считает и присваивает границы на основе константы margins. Для этого использует параметры rect_size и rect_min_size у экземпляра titles и наследников экземпляра players, по которым он проходится, присваивая им значения границ.

3.2.7 Игровая арена

Конфигурационный файл класса: «./maps/map_simple.tscn» [9]. Содержание конфигурационного файла доступно на рисунках 11, 12 и 13.

```
[gd_scene load_steps=9 format=2]

[ext_resource path="res://assets/square.png" type="Texture" id=1]
[ext_resource path="res://assets/sqr_new.png" type="Texture" id=3]

[sub_resource type="OccluderPolygon2D" id=2]
polygon = PoolVector2Array( 32, 32, 0, 32, 0, 0, 32, 0 )

[sub_resource type="ConvexPolygonShape2D" id=3]
points = PoolVector2Array( 32, 32, 0, 32, 0, 0, 32, 0 )

[sub_resource type="TileSet" id=1]
0/name = "square.png 0"
0/texture = ExtResource( 1 )
0/tex_offset = Vector2( 0, 0 )
0/modulate = Color( 1, 1, 1, 1 )
0/region = Rect2( 0, 0, 32, 32 )
0/tile_mode = 0
0/occluder_offset = Vector2( 0, 0 )
0/navigation_offset = Vector2( 0, 0 )
0/shape_offset = Vector2( 0, 0 )
0/shape_transform = Transform2D( 1, 0, 0, 1, 0, 0 )
0/shape_one_way = false
0/shape_one_way_margin = 0.0
0/shapes = [  ]
0/z_index = 0
1/name = "square.png 1"
1/texture = ExtResource( 1 )
1/tex_offset = Vector2( 0, 0 )
1/modulate = Color( 1, 1, 1, 1 )
1/region = Rect2( 0, 0, 32, 32 )
1/tile_mode = 0
1/occluder_offset = Vector2( 0, 0 )
1/occluder = SubResource( 2 )
1/navigation_offset = Vector2( 0, 0 )
1/shape_offset = Vector2( 0, 0 )
1/shape_transform = Transform2D( 1, 0, 0, 1, 0, 0 )
1/shape = SubResource( 3 )
1/shape_one_way = false
1/shape_one_way_margin = 1.0
1/shapes = [ {
"autotile_coord": Vector2( 0, 0 ),
"one_way": false,
"one_way_margin": 1.0,
"shape": SubResource( 3 ),
"shape_transform": Transform2D( 1, 0, 0, 1, 0, 0 )
} ]
1/z_index = 0
```

Рисунок 11 — Конфигурация map_simple, часть 1

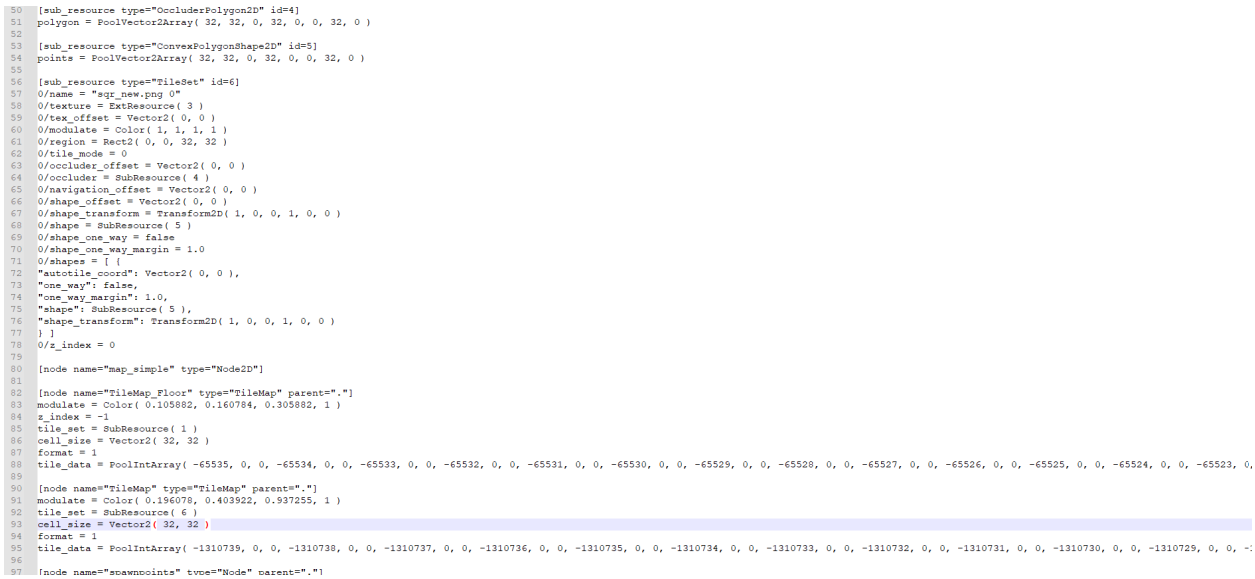

Рисунок 12 — Конфигурация map_simple, часть 2

```
[node name="Position2D" type="Position2D" parent="spawnpoints"]
position = Vector2( 208, 152 )

[node name="Position2D4" type="Position2D" parent="spawnpoints"]
position = Vector2( 1688, 1016 )

[node name="Position2D5" type="Position2D" parent="spawnpoints"]
position = Vector2( 936, 584 )

[node name="Position2D2" type="Position2D" parent="spawnpoints"]
position = Vector2( 200, 1032 )

[node name="Position2D3" type="Position2D" parent="spawnpoints"]
position = Vector2( 1704, 128 )
```

Рисунок 13 — Конфигурация map_simple, часть 3

3.3 Инструкция по сборке

Необходимо учесть, что при разработке использовалась следующая версия Godot: v3.4.4.stable.official.419e713a2 (64-bit Standard Version). Движок молодой и находится в стадии активного развития, такие, что ожидаются крупные изменения в следующих версиях (v4+), а предыдущие версии, то есть версии меньше третьей, по моим наблюдениям, имели абсолютно

отличное видение от используемой мной версии в плане кода.

Для начала необходимо загрузить сам движок Godot, после чего обязательно потребуется загрузить пакеты, осуществляющие сборку проектов Godot под Windows.

Для загрузки последней версии движка можно перейти на https://godotengine.org/download, оттуда уже можно перейти к каталогу всех опубликованных официальных версий движка, прямая ссылка на момент: https://downloads.tuxfamily.org/godotengine/. Скачать можно и на GitHub странице этого движка: https://github.com/godotengine/godot/releases/tag/3.4.4-stable.

Была использована следующая прямая ссылка по загрузке необходимой версии движка: https://downloads.tuxfamily.org/godotengine/3.4.4/Godot_v3.4.4-stable_win64.exe.zip.

После загрузки самого Godot Engine необходимо еще загрузить пакеты для сборки проектов Godot под различные платформы. Они доступны в каталоге всех опубликованных версий и на GitHub, их название заканчивается на _export_templates.tpz. Можно скачать по такой прямой ссылке: https://downloads.tuxfamily.org/godotengine/3.4.4/Godot_v3.4.4-stable_export_templates.tpz. Содержимое этого архива это и есть нужные пакеты для сборки. Для настройки окружения пакетов сборки необходимо открыть или создать любой проект так как необходимые далее по инструкции пункты меню не доступны из окна выбора проекта.

Устанавливаем пакеты сборки с помощью функции «Install from File» в Editor → Manage Export Templates.

В этом же окне имеется альтернативный, полностью автоматизированный способ загрузить и сразу же установить пакеты для сборки, при этом делая это все прямиком из среды редактирования проекта. Для этого опять же нужно перейти в то же окно по пути «Editor → Manage Export Templates», но уже в этом окне необходимо нажать кнопку «Download

and Install» и дождаться завершения загрузки и установки, прогресс загрузки и установки виден в этом же окне. Слева от этой кнопки можно настроить используемое зеркало при загрузке, но это делается по желанию, можно значения оставить такие, какие есть по умолчанию. Снимок вида этого окна в настоящей версии движка предоставлен на рисунке 14.

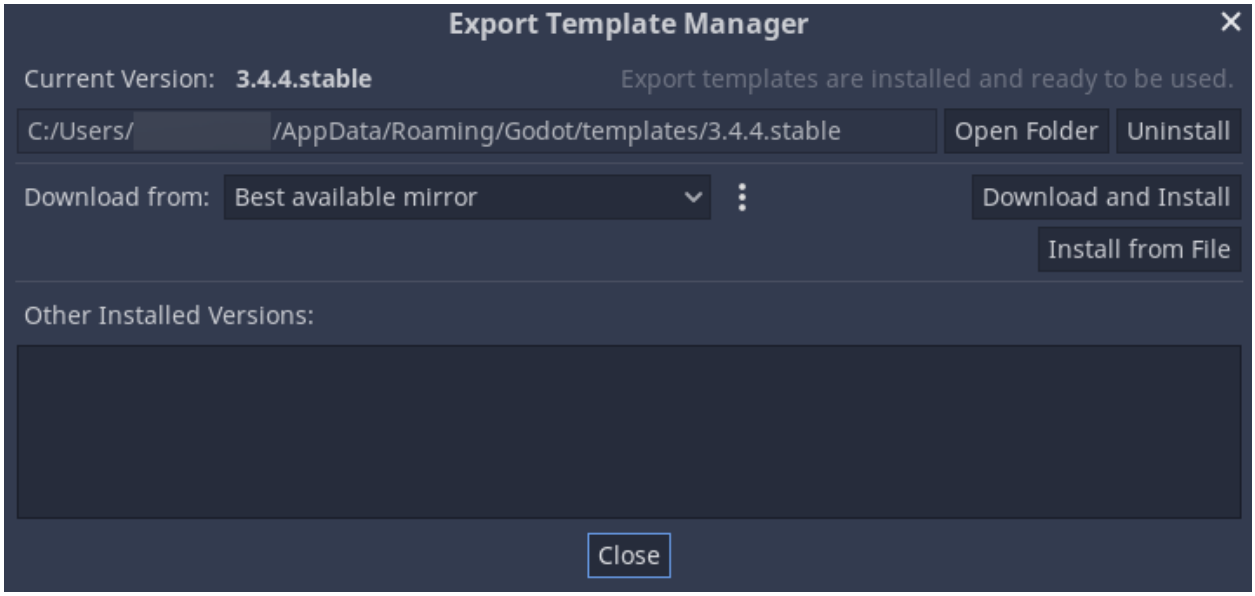

Рисунок 14 — Окно «Export Template Manager»

После того, как был загружен сам движок Godot Engine и были скачаны и установлены все необходимые пакеты для сборки проектов Godot мы имеем среду Godot Engine, настроенную для экспорта проектов на все доступные для этой версии платформы.

Для того, чтобы собрать проект Godot, откроем нужный проект и перейдем по пунктам меню в «Project → Export». В всплывшем после окне справа от текста «Presets» должна быть кнопка «Add...», нажимаем ее, в выпавшем после нажатия меню выбираем «Windows Desktop» и видим, что в списке пресетов экспорта появился пункт «Windows Desktop», необходимо выбрать его, если это не было сделано автоматически. Снимок этого окна на данной версии движка предоставлен на рисунке 15.

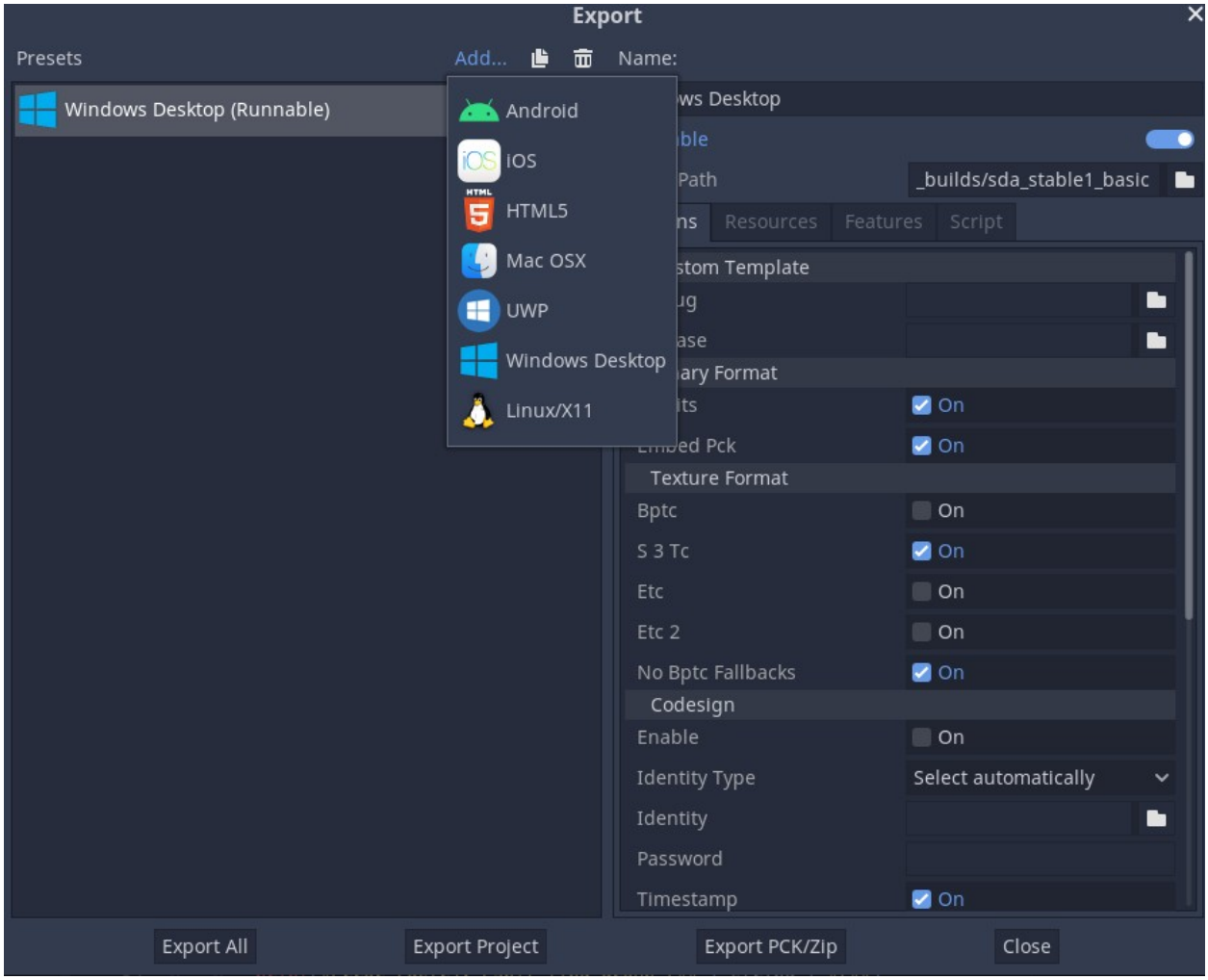

Рисунок 15 — Снимок окна «Project → Export»

Теперь остается лишь настроить параметры сборки проекта под Windows Desktop. Выбирать их следует просто по желанию, значения всех параметров в окне «Export» были оставлены такие, как были по умолчанию, кроме пути сохранения собранного проекта и одного лишь параметра, связанного с конкретно способом сборки запускаемого файла игры. Этот параметр доступен для изменения при нажатии «Export Project» все в этом же окне, называется он «Export with Debug» и представлен как чекбокс, он включен по умолчанию. Как следует из названия, включения этого параметра позволяет собрать запускаемый файл со встроенным отладчиком, это далеко не обязательно и, как и любая отладка, необходимо только в течении

разработки программного обеспечения. Рисунок этого окна доступен на рисунке 16.

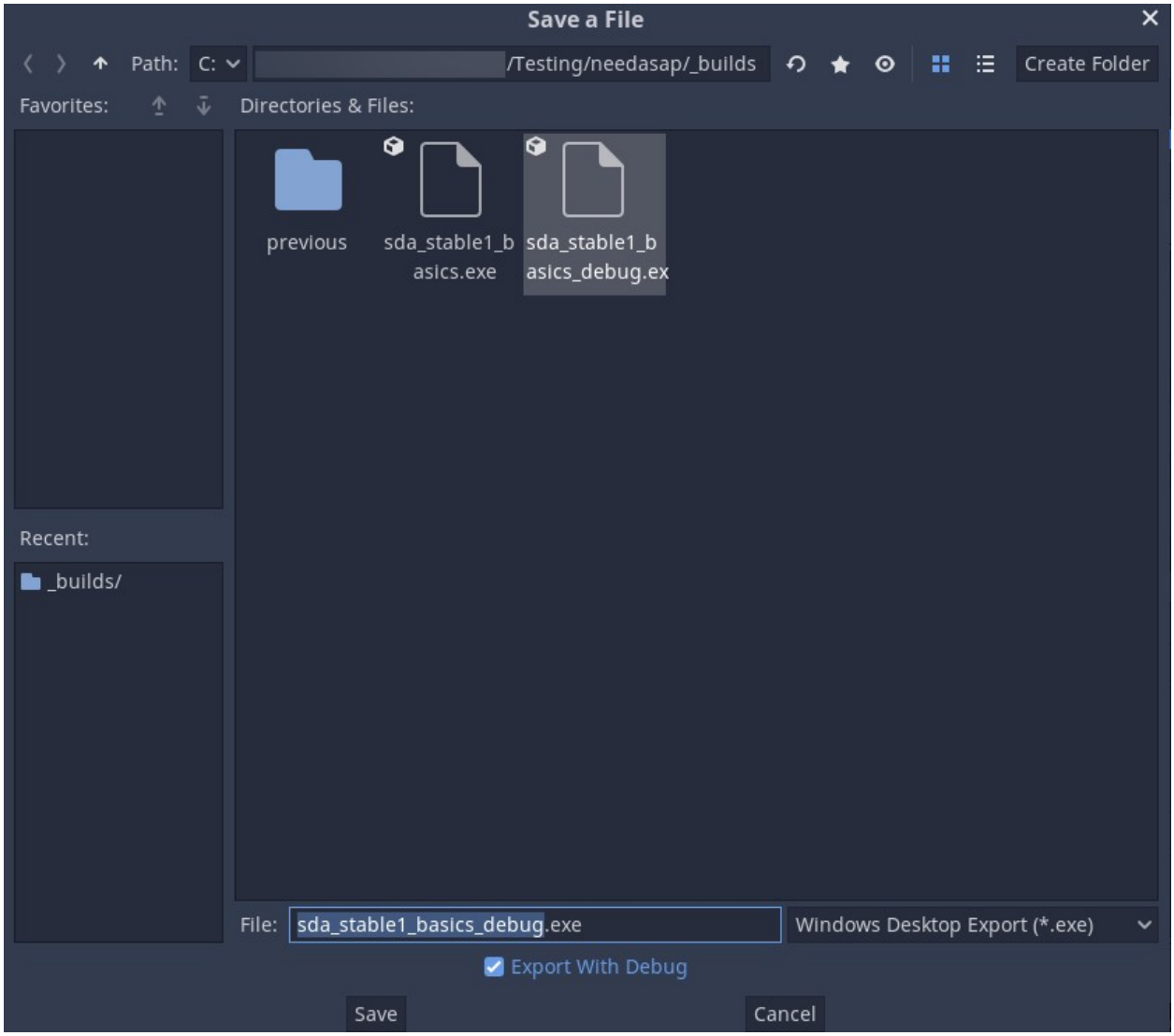

Рисунок 16 — Окно сохранения запускаемого файла

На сохранении запускаемого файла сборка проекта заканчивается.

3.4 Демонстрация разработанной игры

Демонстрация проводилась на компьютере с операционной системой Windows 10. Для тестирования игры обратимся к главе 2 и на основе спроектированной игры рассмотрим реализованную игру.

На рисунке 17 изображено первое, что видит игрок по запуску игры.

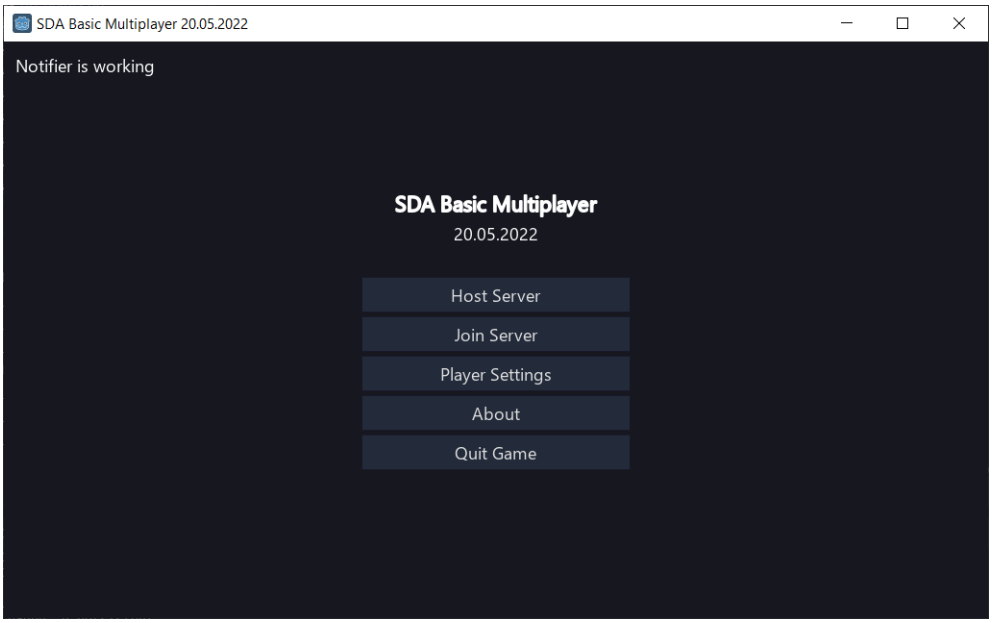

Рисунок 17 — Только что запущенная игра, главное меню

На рисунке 18 изображены все подменю, их дизайн полностью соответствует разработанному макету меню. Они соответственно вызывались нажатиями на кнопки из рисунка 17.

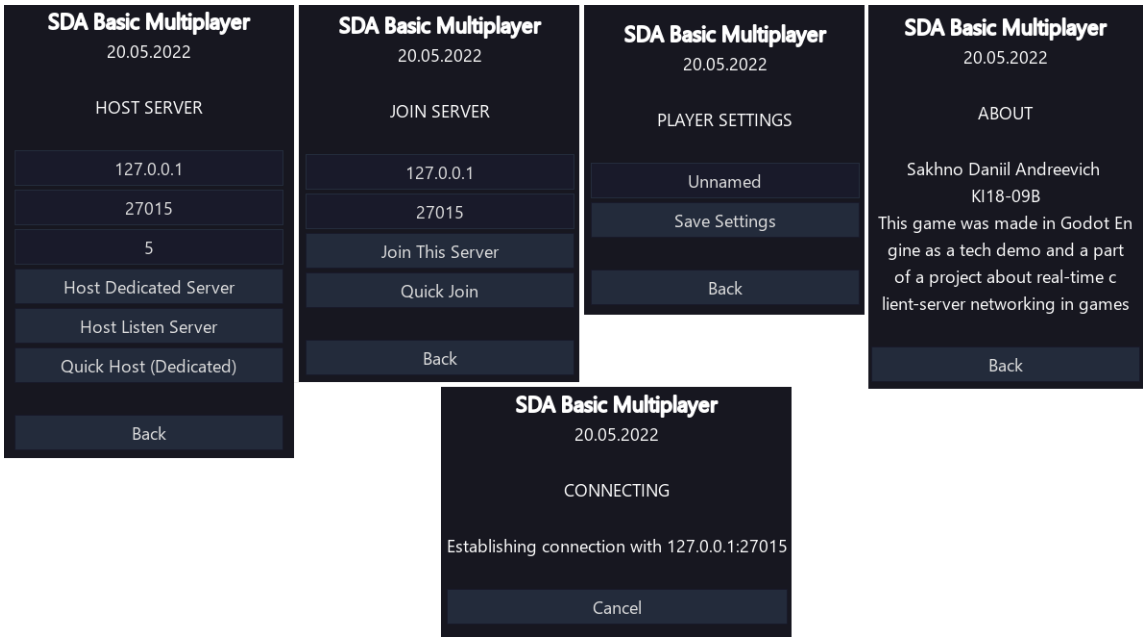

Рисунок 18 — Все подменю главного меню

На рисунке 19 приведена игровая сессия, соответствующая спроектированным правилам игры. На нем изображены танки-игроки, снаряды, отображение здоровья и игровая арена.

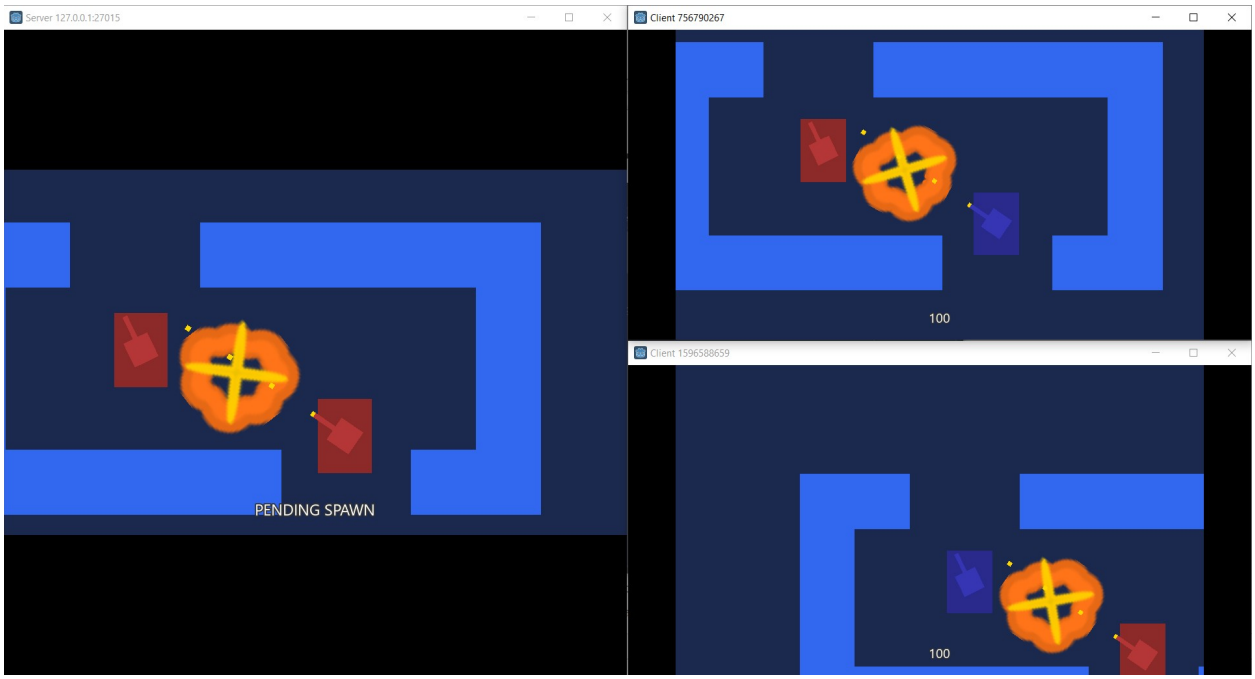

Рисунок 19 — Активная игровая сессия

На рисунке 20 приведена таблица счета, на рисунке 21 приведено игровое меню. Они соответствуют всем спроектированным критериям.

Рисунок 20 — Таблица счета

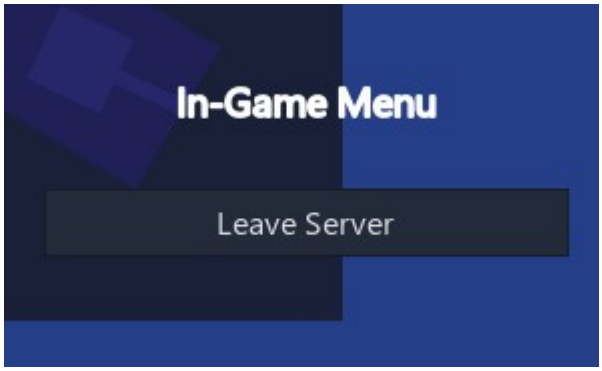

Рисунок 21 — Игровое меню

3.5 Вывод

Была разработана игра в согласии с спроектированной игрой.

Составлена структура проекта и разработанных классов. Были реализованы и подробно задокументированы следующие классы:

– Основной класс Glb;

– Сетевые классы NetShared, NetServer и NetClient;

– Вспомогательные классы env, g_world, g_persistent, g_hud, g_menus, g_notifiers;

– Класс текстовых уведомлений notifier;

– Классы главного меню menu_main и игрового меню menu_ingame;

– Классы игровая логики, состоят из таких классов, как player_full, player_bullet, explosion, explosion_sound, hit, hit_sound;

– Классы игрового интерфейса, состоят из таких классов, как hp_points, hud_scoreboard и devout_camera;

– Игровая арена map_simple.

Была продемонстрирована работа разработанной игры.

ЗАКЛЮЧЕНИЕ

В результате выполнения выпускной квалификационной работы была реализована сетевая компьютерная игра типа «шутер», цель достигнута, задачи решены.

Перечень решенных задач:

– Анализ предметной области, постановка задачи и выбор инструментов для разработки;

– Проектирование игры, ее игровой части ее сетевого модуля;

– Разработка игры.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. СТУ 7.5-07-2021 – Стандарт университета «Общие требования к построению, изложению и оформлению документов учебной деятельности».

2. Шутер - Википедия [Электронный ресурс] – Режим доступа: https://ru.wikipedia.org/w/index.php?title=%D0%A8%D1%83%D1%82%D0%B5%D1%80&oldid=123081644

3. Шутер + 2D [Электронный ресурс] – Режим доступа: https://store.steampowered.com/tags/ru/%D0%A8%D1%83%D1%82%D0%B5%D1%80/3871/

4. OpenGL – The Industry Standard for High Performance Graphics [Электронный ресурс] – Режим доступа: https://www.opengl.org

5. Simple DirectMedia Layer – Homepage [Электронный ресурс] – Режим доступа: https://www.libsdl.org

6. DirectX – Wikipedia [Электронный ресурс] – Режим доступа: https://en.wikipedia.org/w/index.php?title=DirectX&oldid=1089393222

7. GitHub – ValveSoftware/GameNetworkingSockets: Reliable & unreliable messages over UDP. Robust message fragmentation & reassembly. P2P networking / NAT traversal. Encryption. [Электронный ресурс] – Режим доступа: https://github.com/ValveSoftware/GameNetworkingSockets

8. ENet: ENet [Электронный ресурс] – Режим доступа: http://enet.bespin.org.

9. Сахно Д.А. Сетевая компьютерная игра типа шутер (пояснительная записка). Приложение 2. Исходный код [электронный ресурс] / СФУ, ИКИТ, Кафедра вычислительной техники. - Красноярск, 2022. - CD-ROM. - Заголовок компакт-диска

Федеральное государственное автономное
образовательное учреждение
высшего образования
«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт космических и информационных технологий

институт

Вычислительная техника

кафедра

УТВЕРЖДАЮ
Заведующий кафедрой

_____ О.В. Непомнящий

подпись инициалы, фамилия

«20» июня 2022 г.

БАКАЛАВРСКАЯ РАБОТА

Сетевая компьютерная игра типа шутер

тема

09.03.01 «Информатика и вычислительная техника»

код и наименование направления

Руководитель _____ Доцент, канд. физ.-мат. наук К. В. Коршун

подпись, дата должность, ученая степень инициалы, фамилия

Выпускник _____ Д. А. Сахно

подпись, дата инициалы, фамилия

Нормоконтролер _____ Доцент, канд. физ.-мат. наук К. В. Коршун

подпись, дата должность, ученая степень инициалы, фамилия

Красноярск 2022