

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ
Федеральное государственное бюджетное образовательное учреждение
высшего образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет
вычислительной техники

Кафедра
МОиПЭВМ

Направление подготовки 09.03.04 «Программная инженерия»

БАКАЛАВРСКАЯ РАБОТА

на тему

Гейм-дизайн игрового программного приложения

Студент Азаров Владимир Евгеньевич
(подпись, дата) (ФИО полностью)

Руководитель Князев В.Н.
(подпись, дата) (фамилия, инициалы)

Нормоконтролер Такташкин Д.В.
(подпись, дата) (фамилия, инициалы)

Работа допущена к защите (протокол заседания кафедры от _____ № _____)

Заведующий кафедрой Макарычев П.П.
(подпись) (фамилия, инициалы)

Работа защищена с отметкой 5 (протокол заседания ГЭК от 25.06.21 № 12)

Секретарь ГЭК Попова Н.А.
(подпись) (фамилия, инициалы)

Пенза, 2021

Федеральное государственное бюджетное образовательное учреждение
высшего образования
ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет
Вычислительной техники

Кафедра
МО и ПЭВМ

«Утверждаю»
Заведующий кафедрой
П.П. Макарычев
«21» января 2021 г

**ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ
БАКАЛАВРА**

1. Студент Азаров Владимир Евгеньевич гр. 17ВП1 факультета ВТ
<Фамилия, имя, отчество полностью>
2. Тема работы Гейм-дизайн игрового программного приложения

Тема утверждена приказом ПГУ № 0027а/о от "21" января 2021

3. Руководитель работы к.т.н., доцент кафедры «МОиПЭВМ» Князев В.Н.

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

1) *Разработать гейм-дизайн игрового программного приложения для эскиз игры*
Реализовать искусственный интеллект противников

2) *Среда выполнения: операционная система Windows*

Языковые средства разработки: C#, UML.

Инструментальные средства: Unity, Microsoft Visual Studio, Draw.io,

Microsoft Project, MagicaVoxel, Adobe Photoshop, Blender, Figma.

3) *Технологии разработки RUP, ООП, ECS.*

Основные функции приложения:

а) регистрация новых пользователей;

б) вход пользователей в систему;

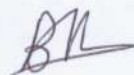
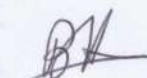
в) однопользовательский режим игры;

г) многопользовательский режим игры со случайными напарниками.

. Перечень подлежащих разработке вопросов:

1) Анализ предметной области и постановка задачи;
2) Проектирование;
3) Реализация;
4) Контроль качества программного обеспечения;
5) Оформление пояснительной записки.

6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1	Анализ предметной области и постановка задачи	15%	14.04.2021–22.04.2021	
2	Проектирование	25%	23.04.2021–07.05.2021	
3	Реализация	30%	08.05.2021–26.05.2021	
4	Контроль качества программного обеспечения	20%	27.05.2021–07.06.2021	
5	Оформление пояснительной записки	10%	08.06.2021–14.06.2021	

Дата выдачи задания " 28 " декабря 2020

Руководитель бакалаврской работы

 28.12.2021
(подпись, дата)

Задание к исполнению принял студент

 28.12.20
(подпись, дата)

Работу к защите допустить
Декан факультета

 Л.Р. Фишова
24.06.2021 (подпись, дата)

Реферат

Пояснительная записка содержит 116 листов, 36 рисунков, 8 таблиц, 32 использованных источника и 4 приложения.

ГЕЙМ-ДИЗАЙН, ИГРОВОЕ ПРИЛОЖЕНИЕ, СЕТЕВЫЕ ИГРОВЫЕ ПРИЛОЖЕНИЯ, UNITY, C#, RUP, UML, NUNIT, ГЕНЕРАЦИЯ МУЗЫКИ, НЕЙРОННЫЕ СЕТИ, МАШИНА СОСТОЯНИЙ, КОНЕЧНЫЙ АВТОМАТ, ИСКУСТВЕННЫЙ ИНТЕЛЛЕКТ, ООП, ECS.

Объектом разработки является игровое приложение с возможностью многопользовательского режима игры.

Цель работы – разработка гейм-дизайна игрового программного приложения.

Результатом разработки является игровое приложение, позволяющее пользователю играть как в одиночном, так и в многопользовательском режиме игры.

Разработка проводилась на языке программирования C#. В качестве среды разработки использовалась IDE Visual Studio 2019. Игровое приложение было реализовано при помощи средств игрового фреймворка Unity. Реализованное приложение было отлажено и протестировано.

					ПГУ 09.03.04– 8BP171.02 ПЗ						
Изм.	Лист	№ докум.	Подпись	Дата	Гейм-дизайн игрового программного приложения. Пояснительная записка	Лит.		Лист		Листов	
Разраб.		Азаров В.Е.		13.06.21							
Пров.		Князев В.Н.		16.06.21					3		116
Н. Контр.		Такташкин Д.В.		22.06.21		Группа 17ВП1					
Утв.		Макарычев П.П.									

Содержание

Введение	6
1 Анализ предметной области	9
1.2 Анализ аналогичных решений.....	10
1.2.1 Анализ игры «Wizard of Legend».....	11
1.2.2 Анализ игры «Furi».....	13
1.2.3 Анализ игры «The Binding of Isaac: Rebirth».....	14
1.2.4 Результат анализа программных аналогов.....	15
1.3 Анализ целевой аудитории	16
1.4 Анализ требований	18
1.4.1 Анализ функциональных требований	18
1.4.2 Нефункциональные требования	22
1.5 Планирование разработки и оценка бюджета.....	22
2 Проектирование игрового приложения.....	28
2.1 Выбор архитектурного шаблона.....	28
2.2 Выбор средства генерации музыки	31
2.2.1 Анализ сервиса «AI Duet»	31
2.2.2 Анализ сервиса «MubertStock»	33
2.2.3 Анализ сервиса «AmperScore».....	34
2.2.4 Результат анализа	35
2.3 Проектирование интерфейса.....	36
2.4 Проектирование искусственного интеллекта противников	39
2.4.1 Нейронная сеть в качестве контроллера противника	39
2.4.2 Конечный автомат в качестве контроллера противника	42
2.4.3 Проектирование боевой системы	44
3 Реализация игрового приложения	48
3.1 Реализация графической составляющей игрового приложения	48
3.2 Реализация игровых механик.....	49
3.2.1 Реализация системы перемещения	50

3.2.2	Реализация боевой системы	52
4	Контроль качества программного обеспечения.....	55
4.1	Функциональное тестирование клиентского приложения	55
4.2	Модульное тестирование клиентского приложения	56
4.3	Исследовательское тестирование	58
4.4	Метрики кода	59
4.5	Цикломатическая метрика Мак-Кейба	61
4.6	Метрика Холстеда	62
	Заключение	65
	Список использованных источников	66
	Приложение А. Глоссарий.....	70
	Приложение Б. Расчет критического пути проекта.....	73
	Приложение В. Исходный код приложения	81
	Приложение Г. Результаты функционального тестирования	104
	Приложение Д. Результаты исследовательского тестирования	113

Введение

Видеоигры возникли достаточно давно. И в момент своего появления они не были доступны широким массам. В то время и компьютер не был доступен обычному человеку, что говорить о видеоиграх. В наше время ситуация поменялась кардинально. Сейчас в видеоигры играют люди всех возрастов, рас, социального достатка и пола. И все благодаря тому, что рынок видеоигр разросся до невероятных масштабов. Если в 70-х поиграть в видеоигру себе могли позволить пара сотен человек, то сейчас таких людей миллионы, если не миллиарды. И если в тех же 70-х в год могло выйти 2 – 3 игры для аркадных автоматов, то сейчас столько игр может быть выпущено в продажу за несколько дней в одном только steam. И с каждым годом рынок все более разрастается и рост этот не прекращается [1 – 3]. Сами по себе игры стали более разнообразные. Если рассматривать опять же 70-е, то в них выпускались лишь аркады, игры с простыми игровыми механиками, чаще всего даже с одной игровой механикой. В настоящий момент существует огромное количество игровых жанров и в итоге каждый человек может найти игру для себя. Любитель гонок может сыграть в гоночные симуляторы или аркадные гонки. Любитель истории может посмотреть в сторону исторических стратегий или же в сторону RPG, которые чаще всего берут как основу для мира именно какие-либо исторические эпохи. Если же человеку нравится фантастика, то поиграть он может практически в любой жанр, начиная от тех же самых гонок и заканчивая играми боевиками. При этом очевидно, что каждый такой жанр содержит в себе совершенно разные механики, иначе смысла в делении на жанры как такового и не было бы. Но при этом некоторые разработчики все же пытаются расширять уже приевшиеся игроку игровые механики жанра игровыми механиками других жанров. Например, добавлением в RPG элементов жанра стратегии [4], или усложнение боевой системы, из-за которой она смотрится как симуляторная [5]. Или же, например, добавление многопользовательского режима. Благодаря нему игра

может оставаться на пике популярности даже спустя несколько лет после ее официального выхода [6]. И если посмотреть статистику, такая тенденция сохраняется для многих игр с хорошим мультиплеером, в который интересно играть с другими людьми [7 – 8]. То есть в топе игр на апрель 2021 находятся в основном именно проекты с многопользовательским режимом, либо вышедшие не так давно игры. Но почему вообще существует такая тенденция и почему не все многопользовательские игры остаются популярными? Все дело в первоначальной задумке, на которой и строится многопользовательский режим. Основа «Dota 2» и «Counter Strike» – соревновательность. Люди заходят в них каждый день лишь для того, чтобы начать играть лучше и все больше побеждать других людей, именно по этой причине под данным играм начали устраивать киберспортивные соревнования. Если посмотреть на ту же «GTA 5», то она все еще популярна благодаря тому, что разработчики дали игрокам возможность создавать новый контент для игры самостоятельно. То есть создавать новые гонки, новые соревновательные режимы и так далее. Таким образом, создав такой полигон для творческой мысли, разработчики привлекли как тех, кому интересно создание такого типа вещей и людей, которым интересно проходить пользовательские карты и режимы. И пока люди будут создавать в «GTA 5» что-то новое, игра будет жить.

Можно также рассмотреть пример плохого многопользовательского режима. Для примера возьмем «Hood: Outlaws & Legends» [9]. Проблема ее сетевого режима состоит в том, что он не интересен и каждый раз дает практически одинаковый опыт. Если рассматривать уже названные «Counter Strike» и «Dota 2», то интерес и желание становиться лучше в игре в них появляется именно из-за того, что каждая новая игра, каждый новый раунд дают свой почти уникальный опыт. Тоже самое и с «GTA 5». Пользовательские постройки практически всегда уникальны. Так почему же мультиплеер в «Hood» плох, а в «Counter Strike» хорош? И как получилось так, что у «Counter Strike» получается создавать новый игровой опыт, а у «Hood»

нет? Все дело в игровом дизайне. Игровой дизайн это, по сути своей, основа создания игры. И, по сути, любой человек, который причастен к созданию игры, ее мира и правил – игровой дизайнер. Но тогда в чем же проблема? В любом случае у «Hood» были программисты, были дизайнеры, у многопользовательских режимов были свои правила. Но все не так просто. Любую работу можно сделать как на низкую оценку, так и на высокую, при этом потратив одинаковые ресурсы. Вся проблема «Hood» состоит именно в плохом игровом дизайне. Ведь основная часть гейм-дизайн, помимо остальных его составляющих, именно проектирование игровых правил и условностей. Но некоторые игровые правила и условности могут хорошо выглядеть на бумаге, а в совокупности с другими правилами и условностями делать игре хуже. И такая проблема не характерна только для многопользовательских игр. Любая хорошая игра строится на грамотном игровом дизайне [10]. Именно из-за него серия игр «Марио» стала популярна. Именно игровой дизайн, опять же, сделал «Counter Strike» популярным, и именно из-за игрового дизайна игры компании «Telltale», практически лишенные каких-либо игровых механик, заработали международную любовь. Потому игровой дизайн, или же гейм-дизайн, – основа не только конкретной игры, но и всего бурно развивающегося рынка в целом.

Таким образом разработка видеоигр – актуальная и перспективная область сферы ИТ, как и ее основа гейм-дизайн.

1 Анализ предметной области

Рынок видеоигр растет с высокими темпами. По данным Интерфакса [3] на конец 2020 года индустрия видеоигр получала больше выручки чем сферы кино и спорта одновременно. На рисунке 1 указана прибыль индустрии видеоигр, сфер кино и спорта в миллиардах долларов.

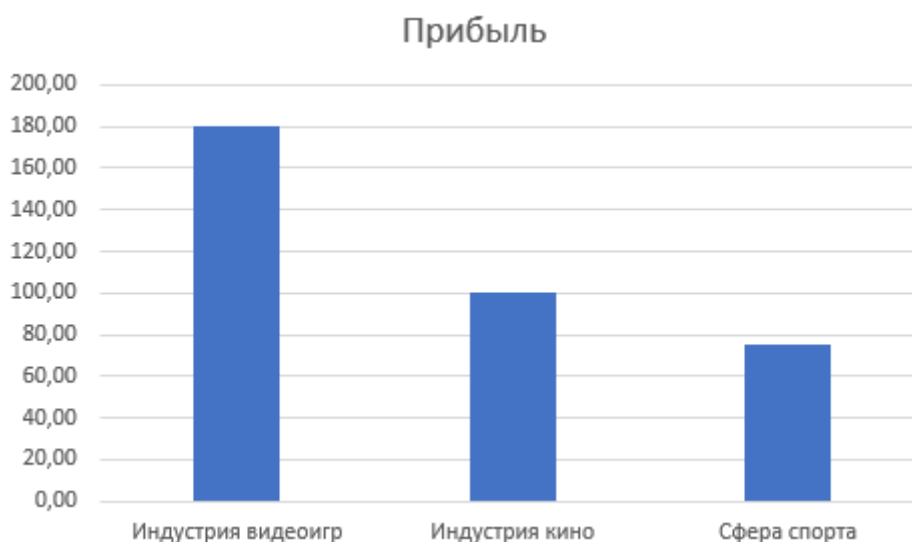


Рисунок 1 – Доходы индустрии видеоигр и сфер кино и спорта за 2020 год

Как видно по графикам на конец 2020 года индустрия видеоигр получала практически в 2 раза больше денег чем индустрия кино и больше чем в 2 раза больше, чем сфера спорта. При этом эксперты прогнозируют продолжение увеличения роста рынка видеоигр. Такая ситуация наблюдается во всем мире, в частности в России, как сообщает РБК [1], за последние несколько лет рынок видеоигр вырос до 2 млрд. долларов и к 2023 может увеличиться до 2,5 млрд. Таким образом на данный момент времени разработка видеоигр перспективна.

При этом, если посмотреть в топы игр по количеству игроков в день, то можно заметить тенденцию, что большее количество игроков играет именно в многопользовательские игры. И соответственно большее количество игроков покупает многопользовательские игры. Также нередки случаи, когда игроки платят вполне реальные деньги за то, чтобы купить в игре какие-либо вещи, что также увеличивает прибыль именно с многопользовательских игр.

Одним из подвидов многопользовательской игры является кооперативная игра. В ней несколько игроков исполняют роль одной команды при этом сражаясь с искусственным интеллектом. Данный вид видеоигр выпускается достаточно редко из-за сложности реализации именно со стороны гейм-дизайна. Однако хорошо реализованные кооперативные игры могут быстро стать популярными. Примерами таких игр можно назвать «It takes two» [11], «A way out» [12].

В рамках выпускной квалификационной работы необходимо разработать гейм-дизайн для кооперативной игры. Под гейм-дизайном подразумевается процесс создания формы и содержания игры. То есть того, как будет выглядеть игра (создание визуальной составляющей игры), как игра будет звучать (создание звуковой составляющей игры), как пользователь будет взаимодействовать с самой игрой (проектирование и реализация интерфейса), с чем пользователь сможет взаимодействовать в игре (проектирование и реализация игровых механик). Таким образом, гейм-дизайн охватывает все составляющие игры и игрового процесса в целом. Конечная игра должна соответствовать следующим требованиям. Пользователь должен иметь возможность:

- а) играть в одиночную игру;
- б) играть со случайной командой;
- в) зарегистрировать аккаунт;
- г) войти в аккаунт;
- д) выйти из игрового матча.

1.2 Анализ аналогичных решений

На данный момент времени многопользовательские игры являются наиболее популярными, и, соответственно, существует большое количество многопользовательских игр. Также существуют и кооперативные игры. Но наибольшее внимание в данном анализе будет отнесено именно к играм геймплейно схожими с разрабатываемой игрой.

1.2.1 Анализ игры «Wizard of Legend»

Игра «Wizard of Legend» – проект, разработанный независимой студией Contingent99. представляет собой экшн-рогалик про мага, который должен пройти все подземелья, победить всех боссов и стать легендарным магом [13]. Данная игра поддерживает кооперативный режим из 2 человек. То есть имеется возможность прохождения «сюжета» вдвоем. Скриншот игры представлен на рисунке 2.

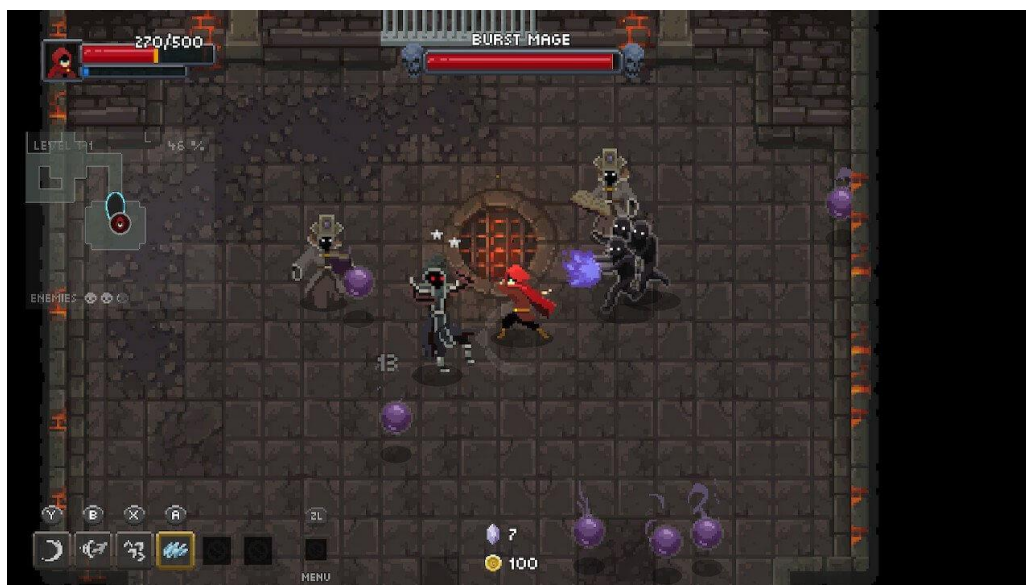


Рисунок 2 – Скриншот из игры Wizard of Legend

Игра предлагает игроку исследовать случайно генерируемые локации подземелий, наполненных врагами. В каждом таком подземелье есть несколько торговцев, у которых можно покупать различные товары, тем самым усиливать некоторые способности своего персонажа. При этом существует только 2 выхода с подземелья: победа над боссом и прохождение в другое подземелье или смерть и потеря всего, что было заработано в предыдущих подземельях.

Основная часть геймплея основана на сражениях. Основной частью сражений же являются заклинания. Во время боя, если игрок хочет применить заклинание, он должен направить прицел (круг с треугольником, очерченный вокруг персонажа) в сторону врага и использовать заклинание. Также

поддерживается система уклонений. А так как в игре бои чаще всего происходят на высоких скоростях, то использовать уклонение игроку придется часто. И если в других играх в битвах требуется скорее грамотная тактика, то здесь помимо нее, нужна еще и отличная реакция, так как иногда необходимо уклоняться от толпы противников, пытающейся атаковать персонажа.

Бои с боссами в данной игре выполнены хорошо. Боссы разнообразные, с различными атаками. К каждому боссу нужна тщательная подготовка, желательно иметь различное оружие, так как босса стихии огня будет достаточно трудно победить магом, у которого есть только огненные заклинания.

Игра имеет достаточно хорошо проработанную боевую систему, и интересный графический стиль. Но у нее есть ряд серьезных минусов. Первым минусом можно назвать локальный кооперативный режим. То есть двум игрокам нужно находиться за одним компьютером, если они хотят играть вместе. Так как подземелья генерируются случайным образом, разнообразием они не отличаются и потому локации в этой игре не запоминаются. И главным минусом в данном случае являются битвы с боссами. Боссы разнообразны, это хорошо, но у всех них есть одна слабость, которая делает их предельно скучными для прохождения. После 2 – 3 атак они останавливаются, чем дают возможность ударить игроку. И такое замедление босса используется практически везде, но в данном случае это всегда 2 – 3 атаки. Потому основная тактика в этой игре сводится к:

- а) переждать атаки;
- б) атаковать во время замедления.

Из-за этого игра теряет свою динамику, которая задается волнами врагов в подземельях.

1.2.2 Анализ игры «Furi»

Игра «Furi» разработана независимой студией The Game Bakers. С точки зрения жанра игру можно отнести к экшен играм. На рисунке 3 представлен скриншот, демонстрирующий битву с одним из боссов игры.



Рисунок 3 – Скриншот из игры «Furi»

Геймплейно «Furi» предлагает игрокам последовательно сразиться с несколькими боссами. Каждый босс имеет свои уникальные приемы, особенности и множество фаз. Фазы могут значительно отличаться друг от друга, заставляя игрока постоянно менять тактику и методы сражения. После победы над боссом прогресс игрока сохраняется, и игрок может приступить к сражению со следующим боссом не опасаясь оказаться в самом начале игры после поражения. Игра полностью сосредотачивается на боевой системе и сражениях с боссами и не предлагает игроку других игровых активностей [14].

С точки зрения геймплея игра соответствует поставленным требованиям, однако она имеет существенный недостаток. Единственный доступный режим игры – одиночный, следовательно, играть в кооперативе с другими игроками не представляется возможным. Кроме того, игра отличается высокой сложностью.

1.2.3 Анализ игры «The Binding of Isaac: Rebirth»

Игра «The Binding of Isaac» – экшн-RPG с элементами жанра roguelike. Основная цель данной игры побеждать противников на случайно сгенерированных уровнях. В конце каждого уровня присутствует босс. Перейти на следующий уровень можно только победив его. Также на уровнях можно найти различное снаряжение, которое может помочь в прохождении. Особенностью данной игры также является специфичная система сердец. Количество сердец — это количество жизней. Однако в отличие от обычной системы жизней, основанной на количестве сердец, здесь каждое сердце может иметь, во-первых, разный цвет, а во-вторых, различный эффект при потере сердца. Так, например, при потере черного сердца произойдет нанесение урона всем противникам, находящимся в данной локации. Также данная игра предоставляет большое количество разнообразных врагов, а также костюмов для главного героя, которые принципиально меняют стиль прохождения игры [15 – 16]. Также к плюсам игры можно отнести отзывчивое управление. Рисунок 4 содержит скриншот из игры.



Рисунок 4 – Скриншот из игры «The Binding of Isaac: Rebirth»

Основных минусов здесь 2. Первым из них является отсутствие мультиплеера, даже локального. Таким образом эту игру можно проходить только одному. Вторым минусом здесь выступает именно случайная генерация. При этом минусом она здесь является достаточно весомым, так как влияет не только на «красоту локаций», но и непосредственно на возможность прохождения игры. В данной игре присутствует режим испытаний, за прохождение которого можно получать новые предметы. Однако минус данного получения предметов заключается в том, что из-за него в списке предметов, которые могут появиться на карте во время прохождения могут появиться и бесполезные, собственно, полученные за испытания. Таким образом прохождение этого режима уменьшает шанс выпадения полезной экипировки. А она нужна, так как некоторых боссов победить можно лишь с определенной. А из-за случайного построения локаций и выпадения экипировки может выйти так, что игра станет попросту непроходимой.

1.2.4 Результат анализа программных аналогов

На основании приведенного анализа аналогов можно прийти к выводу, что разработка собственного решения остается актуальной.

У приведенных игр либо нет возможности мультиплеера, либо он строго локальный. Но даже локальный мультиплеер затрудняет совместное прохождение игры, ведь для этого нужно находиться за одним компьютером, что не всегда возможно.

Пусть у «Wizard of Legend» и есть интересные, отличающиеся друг от друга боссы, но они имеют один и тот же медленный и плавный алгоритм прохождения. В то же время, «The Binding of Isaac» имеет разнообразных боссов, для которых нет единого паттерна прохождения, но из-за случайного выпадения снаряжения некоторых боссов просто невозможно победить и, соответственно, пройти игру.

Таким образом было принято решение разработать свою кооперативную игру, основной целью которой будет битва с боссами.

1.3 Анализ целевой аудитории

Основными игровыми платформами на сегодняшний день являются персональные компьютеры, консоли и смартфоны. По данным Statista [17] на март 2019 года данные платформы являются наиболее перспективными, с точки зрения дальнейшего роста (рисунок 5).

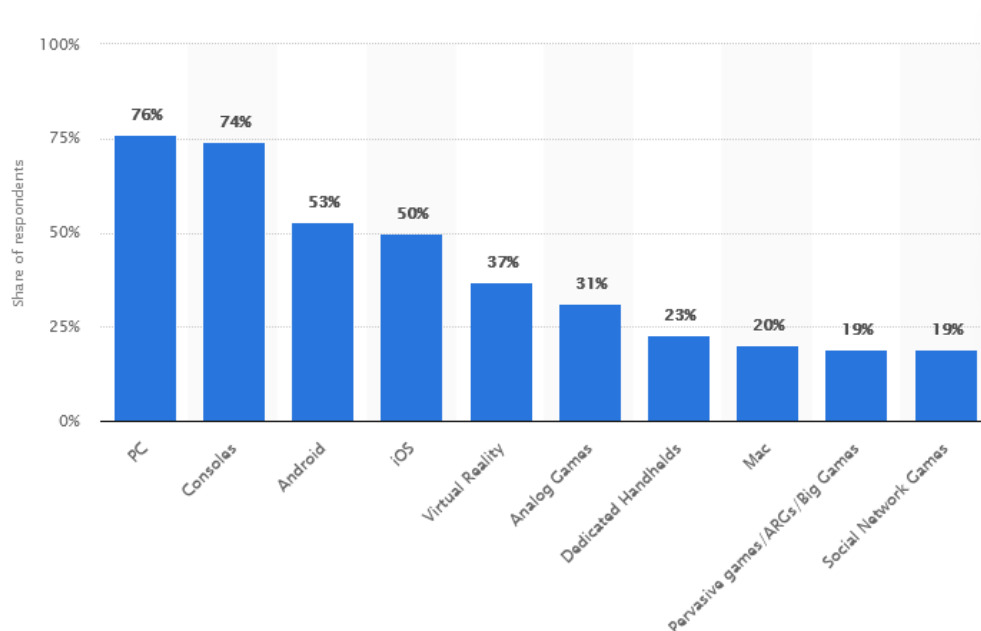


Рисунок 5 – Наиболее перспективные игровые платформы с точки зрения их дальнейшего роста для разработчиков игр

Следовательно, разрабатывая игровой проект, целесообразно ориентироваться именно на эти платформы. Однако консоли и портативные компьютеры не имеют минусов, которые имеются у мобильных устройств. Первый из них это относительно низкая вычислительная способность смартфонов, что создает достаточно узкие рамки производительности для разработчика игры. Второй – возможные проблемы с интернет-соединением, что усложняет создание сетевых игр на мобильной платформе.

На основании представленных данных было принято решение выбрать в качестве целевой платформы персональные компьютеры, так как данная платформа является одной из самых популярных и не накладывает жестких технических ограничений на разработку.

На рисунке 6 представлена диаграмма, демонстрирующая распределение пользователей по разным ОС согласно данным сервиса цифрового распространения компьютерных игр Steam [18 – 19].

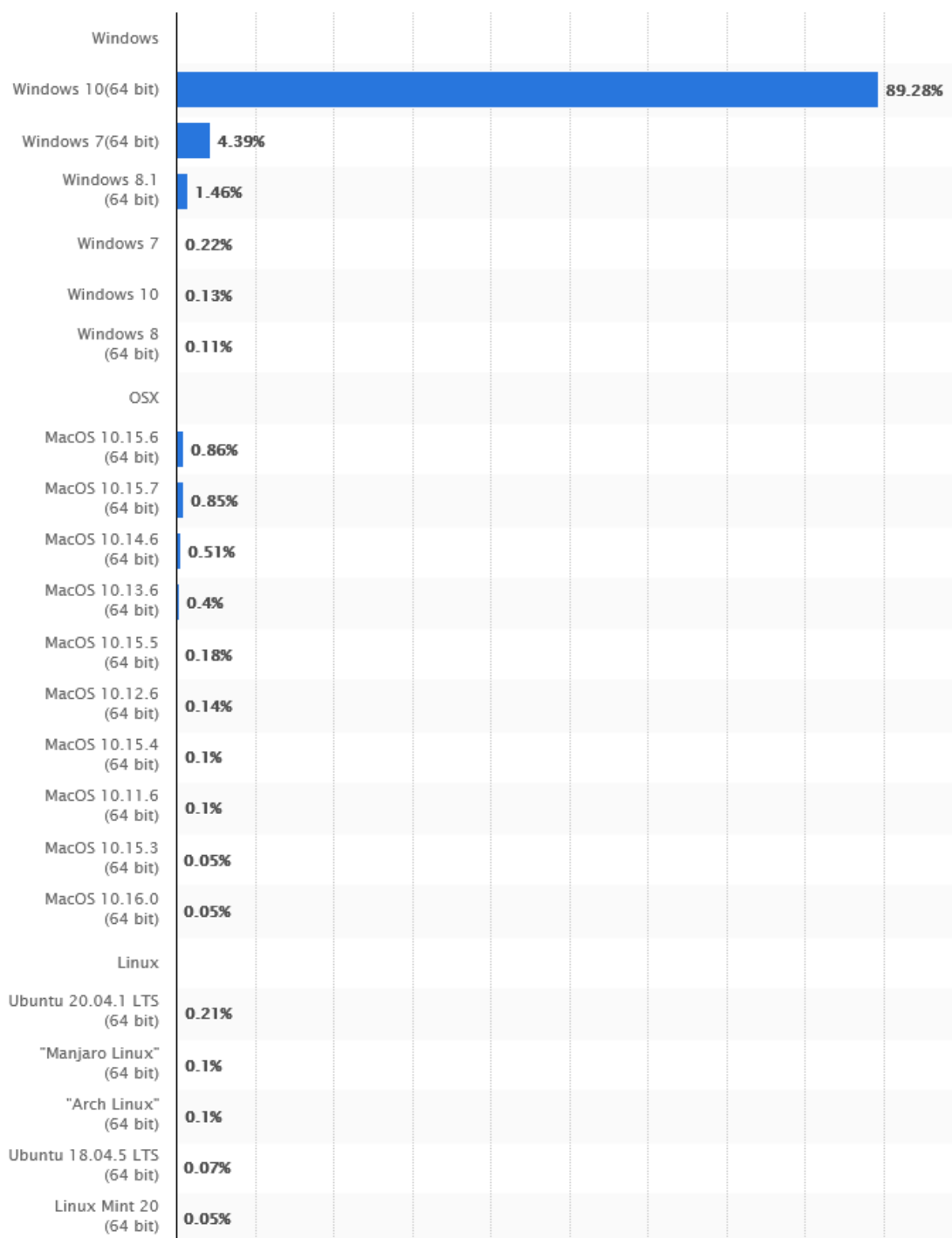


Рисунок 6 – Распределение пользователей сервиса Steam по разным ОС

По данной диаграмме можно сделать вывод, что ОС Windows 10 значительно опережает другие ОС по популярности. Таким образом, было

принято решение разрабатывать игровое приложение для персональных компьютеров, работающих под управлением ОС Windows 10.

1.4 Анализ требований

1.4.1 Анализ функциональных требований

В результате анализа предметной области были построены диаграммы прецедентов, описывающие функциональные требования, предъявляемые к разрабатываемому приложению.

На рисунке 7 представлена прецедентов, описывающая все приложение в целом.



Рисунок 7 – Диаграмма прецедентов системы

Приведем описание актеров, представленных на диаграмме.

Актер «Игрок» представляет собой игрока, использующего игровое приложение.

Актер «Сервер» представляет собой серверную часть игрового приложения, ответственную за организацию игровых матчей.

Формально, данное игровое приложение можно разделить на игровую подсистему, ответственную за гейм-дизайн, и серверную подсистему, которая ответственна за реализацию сетевого кода.

На рисунке 8 представлена диаграмма прецедентов игровой подсистемы разрабатываемого приложения.

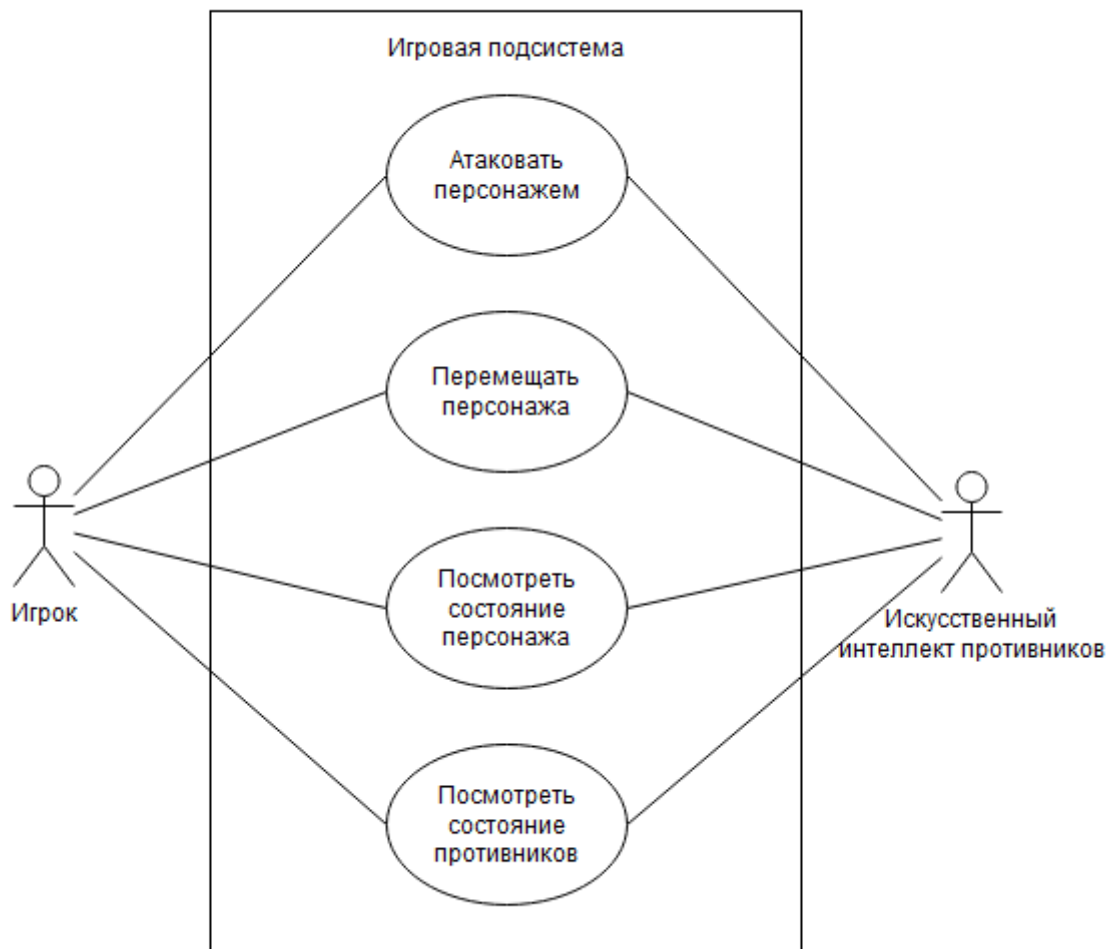


Рисунок 8 – Диаграмма прецедентов игровой подсистемы

На данной диаграмме в дополнение к уже существующим актерам добавляется «Искусственный интеллект противников». Данный актер ответственен за поведение игровых противников, реагирующих на действия игрока.

Спецификации прецедентов, ответственных за перемещение персонажа и атаку, представлены на рисунках 9 – 10.

Атаковать персонажем
ID: 1
Краткое описание: Клиент атакует
Главные актеры: Клиент, игровая система
Предусловия: Клиент вошел в игру
Основной поток: <ol style="list-style-type: none"> 1. Клиент инициирует функцию атаки 2. Если персонаж не в стадии атаки и персонаж жив <ol style="list-style-type: none"> 2.1. Персонаж выполняет атаку
Постусловия: Система обработала функцию атаки

Рисунок 9 – Спецификация прецедента «Атаковать персонажем»

Перемещать персонажа
ID: 2
Краткое описание: Клиент перемещает игрового персонажа по локации
Главные актеры: Клиент, игровая система
Предусловия: Клиент вошел в игру
Основной поток: <ol style="list-style-type: none"> 1. Клиент инициирует функцию перемещения и указывает его направление 2. Если персонаж жив и направление не указывает на препятствие <ol style="list-style-type: none"> 2.1. Персонаж выполняет перемещение
Постусловия: Система обработала функцию перемещения

Рисунок 10 – Спецификация прецедента «Перемещать персонажа»

Спецификации прецедентов, ответственных за просмотр состояний персонажей и противников, представлены на рисунках 11 – 12.

Посмотреть состояние персонажа
ID: 3
Краткое описание:
Главные актеры: Клиент, игровая система
Предусловия: Клиент вошел в игру
Основной поток: <ol style="list-style-type: none"> 1. Клиент инициировал просмотр информации о состоянии персонажа 2. Игровая система выводит на экран информацию о состоянии персонажа
Постусловия: Клиент получил информацию о состоянии персонажа

Рисунок 11 – Спецификация прецедента «Посмотреть состояние персонажа»

Посмотреть состояние противников
ID: 4
Краткое описание:
Главные актеры: Клиент, игровая система
Предусловия: Клиент вошел в игру
Основной поток: <ol style="list-style-type: none"> 1. Клиент инициировал просмотр информации о состоянии противников 2. Игровая система выводит на экран информацию о состоянии противников
Постусловия: Клиент получил информацию о состоянии противников

Рисунок 12 – Спецификация прецедента «Посмотреть состояние противников»

Диаграмма последовательности, поясняющая прецедент «Играть в случайной команде» представлена на рисунке 13.

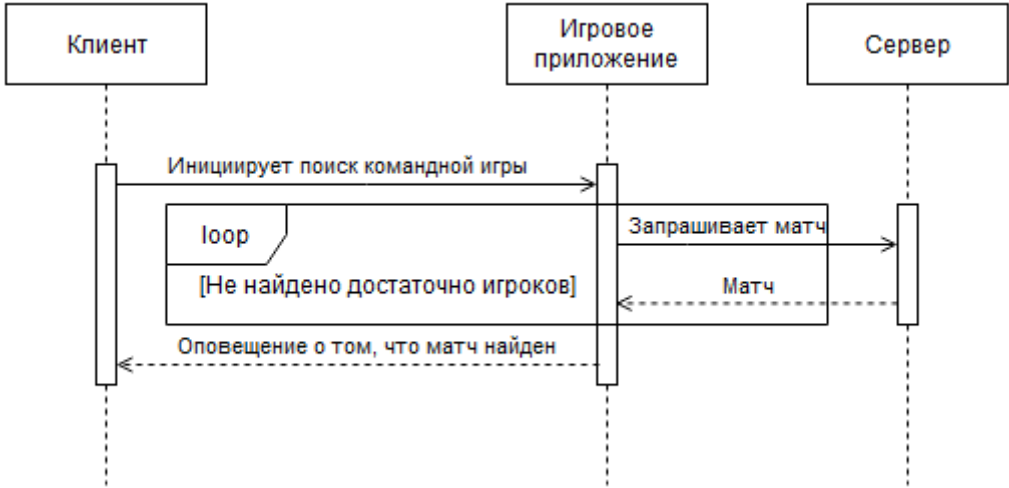


Рисунок 13 – Диаграмма последовательности для прецедента «Играть в случайной команде»

Данная система демонстрирует механизм получения клиентом данных для подключения к кооперативному матчу.

1.4.2 Нефункциональные требования

Игровая система должна быть реализована на языке C# при помощи средств игрового фреймворка Unity 2019. Игра должна корректно работать на персональных компьютерах под управлением ОС Windows 10.

При разработке необходимо использовать IDE Visual Studio.

Для работы с графикой необходимо использовать программы MagicaVoxel, Blender 2.9, Adobe Photoshop 2020.

1.5 Планирование разработки и оценка бюджета

Для определения стоимости и трудоемкости проекта произведем структурное планирование. Таблица 1 содержит перечень задач проекта.

Таблица 1 – Перечень задач проекта

№	Название задачи	Длительность	Предшественники
1	Начало проекта	-	-
2	Анализ требований	-	-
3	Анализ аналогичных решений	3 дня	1
4	Анализ подходов к реализации игровых приложений	3 дня	3

Продолжение таблицы 1

№	Название задачи	Длительность	Предшественники
5	Анализ подходов к реализации игровых интерфейсов	3 дня	3
6	Выбор оптимального подхода к реализации игрового приложения	1 день	4
7	Выбор оптимального подхода к реализации игрового интерфейса	1 день	5
8	Анализ требований завершен	-	6,7
9	Выбор средств реализации	-	-
10	Анализ возможных программных средств	2 дня	1
11	Выбор оптимальных программных средств	1 день	10
12	Выбор средств завершен	-	11
13	Реализация проекта	-	-
14	Проектирование дизайна персонажей	4 дня	8, 12
15	Проектирование игрового окружения	4 дня	8, 12
16	Проектирование интерфейса	4 дня	8, 12
17	Моделирование персонажей	7 дней	14
18	Моделирование деталей игрового окружения	7 дней	15
19	Реализация игрового интерфейса	4 дня	16
20	Анимация персонажей	7 дней	17
21	Реализация алгоритмов управления персонажем	7 дней	19
22	Реализация алгоритмов боевой системы	7 дней	21
23	Реализация искусственного интеллекта противников	7 дней	21
24	Реализация завершенна	-	18, 20, 22, 23
25	Отладка	-	-
26	Тестирование интерфейса приложения	3 дней	24
27	Тестирование системы управления персонажем	7 дней	24
28	Тестирование боевой системы	7 дней	27
29	Исправление ошибок	3 дня	26, 28
30	Отладка завершенна	-	29
31	Конец проекта	-	30

В таблице 2 приведено описание ресурсов.

Таблица 2 – Список ресурсов проекта

Название ресурса	Тип	Стандартная ставка	Ставка сверхурочных
Программист 1	Трудовой	500 р/ч	750 р/ч
Программист 2	Трудовой	500 р/ч	750 р/ч
Дизайнер интерфейсов	Трудовой	450 р/ч	675 р/ч
Дизайнер персонажей	Трудовой	200 р/ч	300 р/ч
Дизайнер игрового окружения	Трудовой	450 р/ч	675 р/ч
3D-аниматор	Трудовой	300 р/ч	450 р/ч
Интернет	Затраты		

Назначение ресурсов на задачи представлено в таблице 3.

Таблица 3 – Назначение ресурсов

№	Название задачи	Ресурсы	Единицы
1	Начало проекта		
2	Анализ требований		
3	Анализ аналогичных решений	Программист 1 Интернет	100%
4	Анализ подходов к реализации игровых приложений	Программист 1 Интернет	100%
5	Анализ подходов к реализации игровых интерфейсов	Программист 2 Интернет	100%
6	Выбор оптимального подхода к реализации игрового приложения	Программист 1 Интернет Интернет	100%
7	Выбор оптимального подхода к реализации игрового интерфейса	Программист 2 Интернет	100%
8	Анализ требований завершен		
9	Выбор средств реализации		
10	Анализ возможных программных средств	Программист 1 Программист 2 Интернет	50% 50%
11	Выбор оптимальных программных средств	Программист 1 Программист 2 Интернет	50% 50%
12	Выбор средств завершен		
13	Реализация проекта		
14	Проектирование дизайна персонажей	Дизайнер персонажей Интернет	100%
15	Проектирование игрового окружения	Дизайнер игрового окружения Интернет	100%
16	Проектирование интерфейса	Дизайнер интерфейсов Интернет	100%
17	Моделирование персонажей	Дизайнер персонажей Интернет	100%
18	Моделирование деталей игрового окружения	Дизайнер игрового окружения Интернет	100%
19	Анимация персонажей	3D-аниматор Интернет	100%
20	Реализация игрового интерфейса	Программист 2 Интернет	100%
21	Реализация алгоритмов управления персонажем	Программист 1 Программист 2 Интернет	50% 50%
22	Реализация алгоритмов боевой системы	Программист 1 Программист 2 Интернет	50% 50%
23	Реализация искусственного интеллекта противников	Программист 2 Интернет	100%
24	Реализация завершена		
25	Отладка		
26	Тестирование интерфейса приложения	Программист 2 Интернет	100%
27	Тестирование системы управления персонажем	Программист 1 Программист 2 Интернет	50% 50%
28	Тестирование боевой системы	Программист 1 Программист 2 Интернет	50% 50%

Продолжение таблицы 3

№	Название задачи	Ресурсы	Единицы
29	Исправление ошибок	Программист 1 Программист 2 Интернет	50% 50%
30	Отладка завершена		
31	Конец проекта		

Полный расчет позднего и раннего времени начала проекта представлено в приложении Б.

На рисунке 14 представлен список задач из таблицы 1 внесенный в Microsoft Project.

	Реж. зада	Название задачи	Длительность	Начало	Окончание	Предшес	Названия ресурсов
1		Начало проекта	0 дней	Пн 12.04.21	Пн 12.04.21		
2		Анализ требований	7 дней	Пн 12.04.21	Вт 20.04.21		
3		Анализ аналогичных решений	3 дней	Пн 12.04.21	Ср 14.04.21	1	Программист 1;Интернет
4		Анализ подходов к реализации игровых приложений	3 дней	Чт 15.04.21	Пн 19.04.21	3	Программист 1;Интернет
5		Анализ подходов к реализации игровых интерфейсов	3 дней	Чт 15.04.21	Пн 19.04.21	3	Программист 2;Интернет
6		Выбор оптимального подхода к реализации игрового приложения	1 день	Вт 20.04.21	Вт 20.04.21	4	Программист 1;Интернет
7		Выбор оптимального подхода к реализации игрового интерфейса	1 день	Вт 20.04.21	Вт 20.04.21	5	Программист 2;Интернет
8		Анализ требований завершен	0 дней	Вт 20.04.21	Вт 20.04.21	6;7	
9		Выбор средств реализации	3 дней	Пн 12.04.21	Ср 14.04.21		
10		Анализ возможных программных средств	2 дней	Пн 12.04.21	Вт 13.04.21	1	Программист 1[50%];Программист 2[50%];Интернет
11		Выбор оптимальных программных средств	1 день	Ср 14.04.21	Ср 14.04.21	10	Программист 1[50%];Программист 2[50%];Интернет
12		Выбор средств завершен	0 дней	Ср 14.04.21	Ср 14.04.21	11	
13		Реализация проекта	22 дней	Ср 21.04.21	Чт 20.05.21		
14		Проектирование дизайна персонажей	4 дней	Ср 21.04.21	Пн 26.04.21	8;12	Дизайнер персонажей;Интернет
15		Проектирование игрового окружения	4 дней	Ср 21.04.21	Пн 26.04.21	12;8	Дизайнер игрового окружения;Интернет
16		Проектирование интерфейса	4 дней	Ср 21.04.21	Пн 26.04.21	8;12	Дизайнер интерфейсов;Интернет
17		Моделирование персонажей	7 дней	Вт 27.04.21	Ср 05.05.21	14	Дизайнер персонажей;Интернет
18		Моделирование деталей игрового окружения	7 дней	Вт 27.04.21	Ср 05.05.21	15	Дизайнер игрового окружения;Интернет
19		Анимация персонажей	7 дней	Чт 06.05.21	Пт 14.05.21	17	3D-аниматор;Интернет
20		Реализация игрового интерфейса	4 дней	Вт 27.04.21	Пт 30.04.21	16	Интернет;Программист 2
21		Реализация алгоритмов управления персонажем	7 дней	Пн 03.05.21	Вт 11.05.21	20	Интернет;Программист 1[50%];Программист 2[50%];Интернет
22		Реализация алгоритмов боевой системы	7 дней	Ср 12.05.21	Чт 20.05.21	21	Интернет;Программист 1[50%];Программист 2[50%];Интернет
23		Реализация искусственного интеллекта противников	7 дней	Ср 12.05.21	Чт 20.05.21	21	Интернет;Программист 2
24		Реализация завершена	0 дней	Чт 20.05.21	Чт 20.05.21	18;22;23;19	
25		Отладка	17 дней	Пт 21.05.21	Пн 14.06.21		
26		Тестирование интерфейса приложения	3 дней	Пт 21.05.21	Вт 25.05.21	24	Интернет;Программист 2
27		Тестирование системы управления персонажем	7 дней	Пт 21.05.21	Пн 31.05.21	24	Интернет;Программист 1[50%];Программист 2[50%];Интернет
28		Тестирование боевой системы	7 дней	Вт 01.06.21	Ср 09.06.21	27	Интернет;Программист 1[50%];Программист 2[50%];Интернет
29		Исправление ошибок	3 дней	Чт 10.06.21	Пн 14.06.21	26;28	Интернет;Программист 1;Программист 2
30		Отладка завершена	0 дней	Пн 14.06.21	Пн 14.06.21	29	
31		Конец проекта	0 дней	Пн 14.06.21	Пн 14.06.21	30	

Рисунок 14 – Задачи проекта в среде Microsoft Project

На рисунке 15 представлена диаграмма Ганта, построенная в среде Microsoft Project на основе введенных данных.

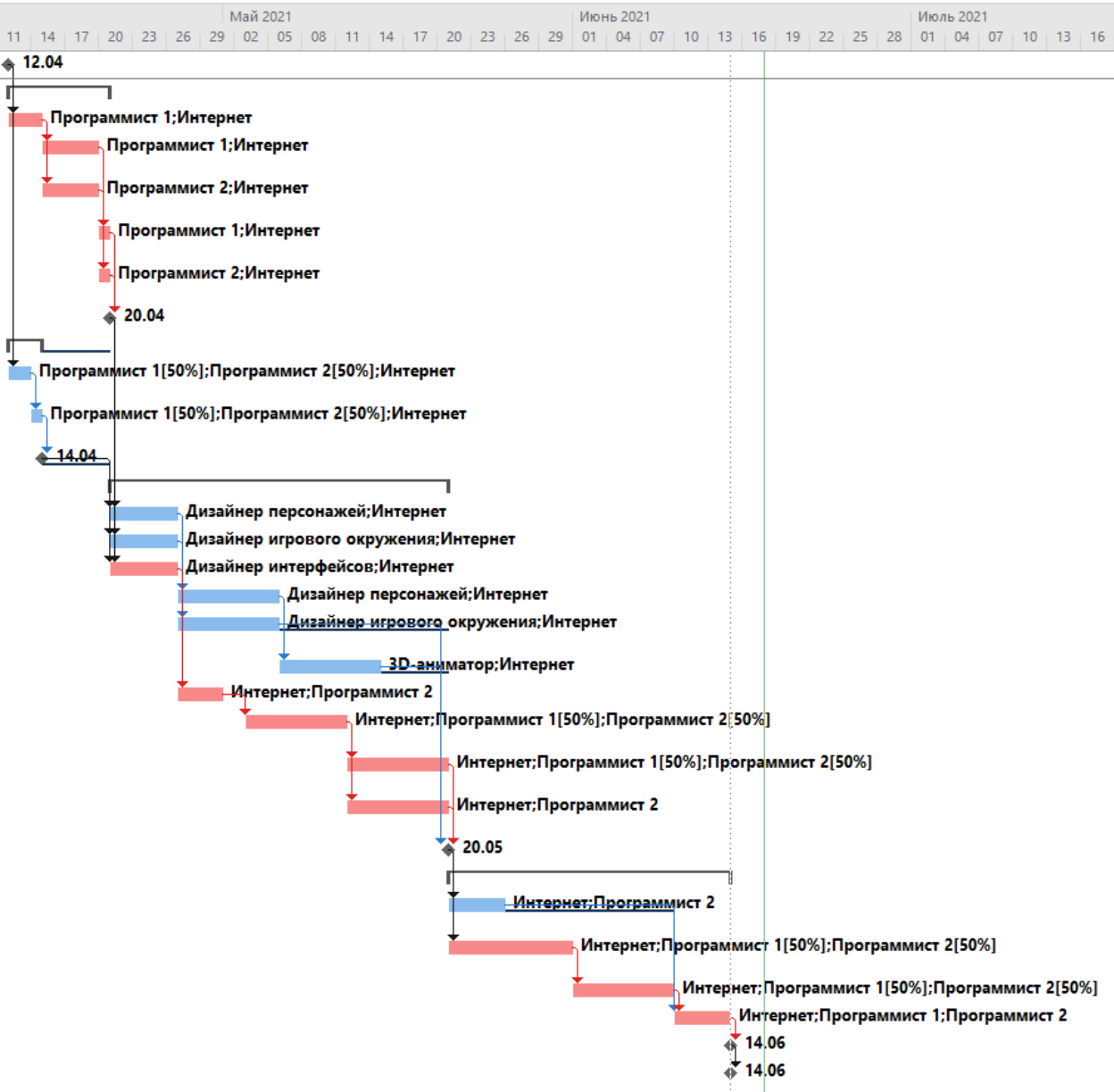


Рисунок 15 – Диаграмма Ганта

На рисунке 16 представлен обзор проекта.

Начало	Окончание	Длительность	Затраты	Трудозатраты
Пн 12.04.21	Пн 14.06.21	46 дней	336 400,00 Р	760 ч

Рисунок 16 – Обзорный отчет проекта

Из рисунка видно, что общая длительность проекта составляет 46 дней, а предполагаемые затраты – 336 400 руб. Таким образом, анализ показал экономическую целесообразность разработки данного игрового приложения.

2 Проектирование игрового приложения

2.1 Выбор архитектурного шаблона

В Unity на данный момент времени существует 2 известных архитектурных подхода к построению приложения.

Классический подход подразумевает создание скриптов, описывающих поведение объектов и использование их в качестве компонентов к сущностям сцены. Данного подхода сообщество Unity придерживалось примерно до 2017 года. Он позволяет в короткие сроки и с малым количеством усилий реализовать передвижение объектов, искусственный интеллект и в целом все базовые вещи, которые нужны однопользовательской игре. И это очевидно, ведь долгое время разработчики Unity развивали именно этот подход к проектированию, потому для него создано больше инфраструктуры. Однако такой подход удобен лишь для приложений малых размеров. При росте приложения, увеличивается и сложность добавления новых механик, код становится более запутанным, бизнес логика же разносится по огромному количеству файлов, прикрепленных к различным сущностям сцены. И все это контролируется игровым циклом Unity (рисунок 17) [20].

Но Unity уже давно стал фреймворком, который используется не только для малоразмерных игр [21]. Таким образом нужен был другой подход, который поддержат и разработчики, чтобы под него разрабатывались свой функционал, и сообщество, чтобы этот функционал использовался. Этим подходом стал ECS.

Подход ECS существует уже давно, был придуман он значительно задолго до Unity и за время своего существования зарекомендовал себя с хорошей стороны. К его плюсам можно отнести гибкость, расширяемость, а также малоразмерность его игрового цикла [22]. Достигается все это за счет использования композиции вместо наследования. Вся проблема наследования в том, что при неудачном проектировании дерево классов может разрастись настолько, что поддержка приложения станет практически невозможной.

Соответственно сильно пострадает расширяемость, а чтобы добиться гибкости придется прикладывать большое количество усилий.

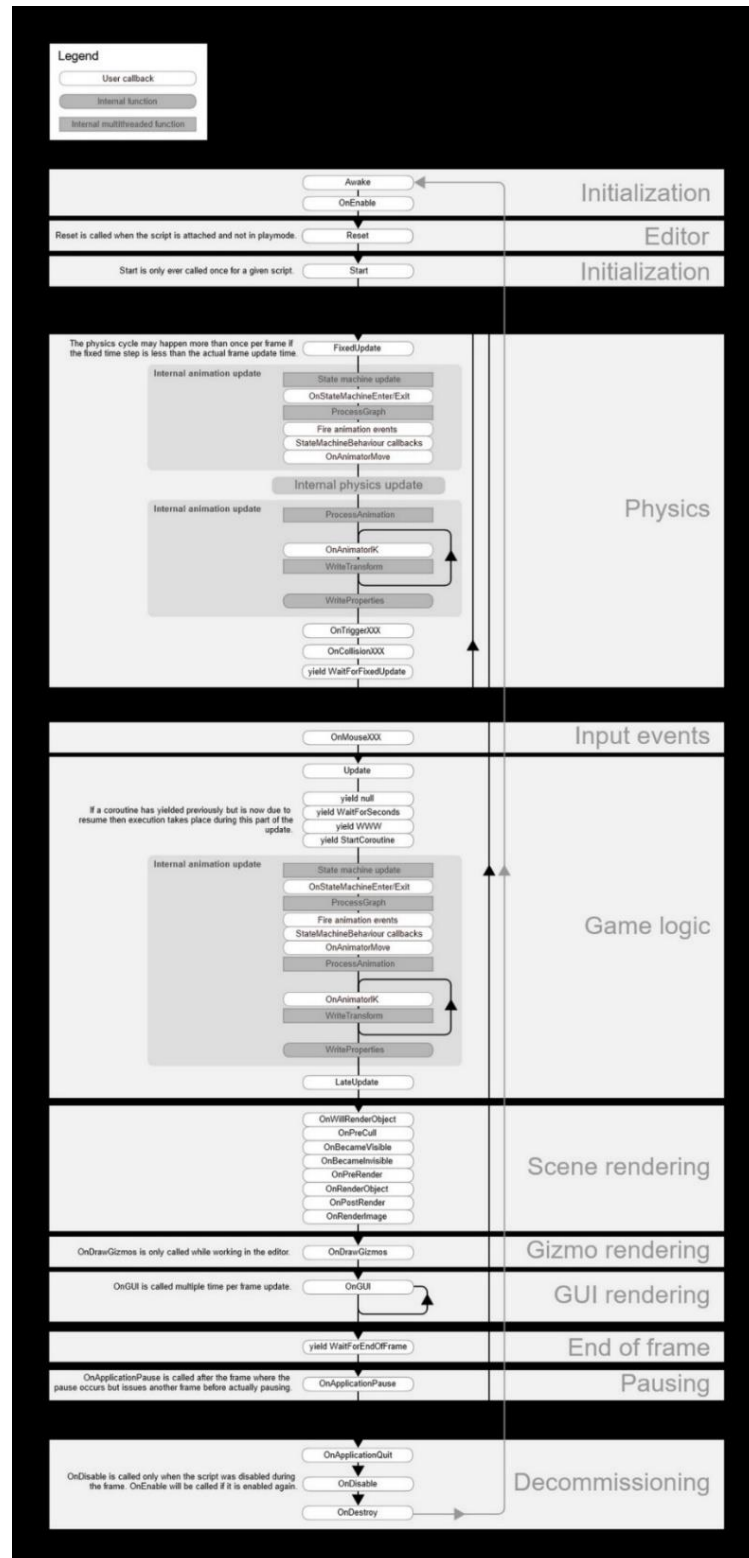


Рисунок 17 – Игровой цикл Unity

Такая же проблема и с игровым циклом. Каким бы хорошо продуманным не был код, велика вероятность того, что однажды игровой цикл разрастется и станет трудно поддерживаемым. ECS же используется чтобы избежать этого.

В основе ECS лежит разделение всего приложения на 3 основные части [23]. Компоненты содержат в себе атомарные блоки данных. Они не содержат в себе никакой функциональности. Сущности содержат в себе компоненты. Сущность в целом определяется тем, какие компоненты она содержит. Если производить аналогии с ООП, то компонент для сущности это как интерфейс для наследуемого класса. Последняя часть ECS – системы. Они уже содержат бизнес-логику. Система четко указывает какой набор компонентов должна содержать сущность, чтобы быть обработанной. Так, например, система принимает только сущности, у которых есть компонент скорости и вращения. Если у сущности есть компонент скорости и, например, визуальный компонент, то она не будет обработана этой системой.

В игровом же цикле просто происходит последовательный запуск этих систем. Таким образом игровой цикл растет лишь от добавления новых систем, но не более того. Представим игровой цикл работы ECS в виде диаграммы деятельности (рисунок 18).

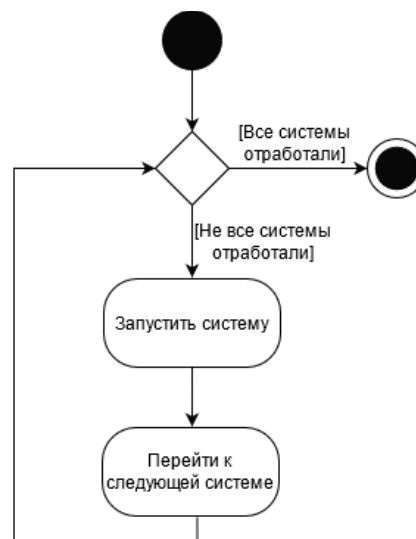


Рисунок 18 – Диаграмма деятельности игрового цикла ECS

Однако в данном случае стоит сделать уточнение, что пользователь обязан указать то, в какой последовательности будут запущены системы. Это нужно для того, чтобы порядок обработки данных не нарушался. Например, одна из систем нужна для того, чтобы обнулять данные. Если запустить ее сразу после получения данных, то в систему обработки этих данных всегда будут попадать нули. Данную особенность, в целом, можно отнести к минусу данного подхода, также как и, например, появление большого количества файлов, но эти минусы, в сравнении с названными плюсами, в действительности незначительны.

2.2 Выбор средства генерации музыки

В игре, как и в кино, музыка необходима, так как она настраивает игрока на нужное настроение, а также задает нужный темп происходящему. Так, например, если на фоне к сцене серьезной битвы будет играть веселая музыка или же излишне спокойная музыка, это будет выглядеть глупо и скорее всего будет сбивать игрока. Разрабатываемая игра также должна иметь музыкальное сопровождение, а так как на данный момент времени уже существуют нейронные сети, которые могут сгенерировать музыку, то удобнее всего использовать именно их для получения уникальной фоновой музыки.

Сервисов, которые предоставляют возможность генерации музыки пока что не так много, но они все же есть. Они отличаются по функционалу, предоставляемому пользователю, а также по тому по каким данным генерируется музыка.

2.2.1 Анализ сервиса «AI Duet»

Сервис созданный Google, позволяет дополнять сыгранную на пианино музыку [24]. Данный сервис не акцентируется полностью именно на генерации музыки, а скорее на воссоздании «дуэта» из двух пианистов, где первый это человек, который вводит данные, а второй это нейронная сеть. Интерфейс сервиса представлен на рисунке 19.

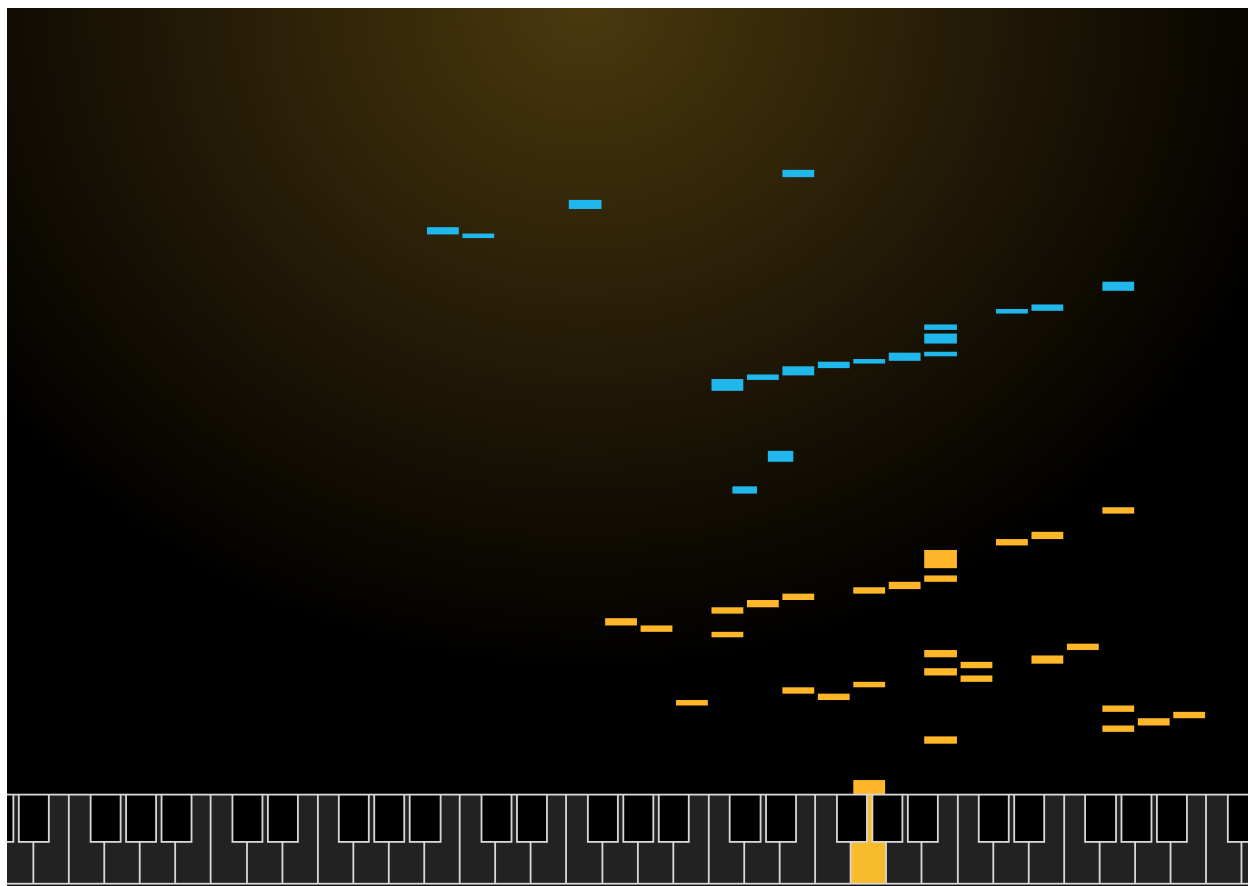


Рисунок 19 – Интерфейс сервиса «AI Duet»

Для использования данным сервисом не нужно знать ни основ построения, ни информации об обучении нейронных сетей, даже не нужно устанавливать ничего на свой компьютер. Однако, так как данный сервис акцентирует свое внимание на «дополнении» сыгранной композиции, то очевидно, что композиции уже должна быть, а пользователь должен уметь играть на пианино. Это может быть как минусом, так и плюсом. Ведь очевидно, что человек, который не умеет играть и сочинять музыку, не сможет создать с помощью данного сервиса ничего стоящего, но, если же пользователь обладает достаточными способностями, сервис предоставит ему достаточно гибкий функционал, с помощью которого можно сгенерировать именно такую музыку, которая ему необходима.

2.2.2 Анализ сервиса «MubertStock»

Данный сервис ставит своей задачей генерировать музыку без авторских прав, которую творцы смогут использовать в своих работах [25]. Данный сервис, в отличие от предыдущего, не является бесплатным, но при этом предоставляет более широкий функционал именно к генерации композиций. За покупку подписки пользователь получит право использование музыки, возможность скачивания композиции с сайта в неограниченном виде. Интерфейс сайта представлен на рисунке 20.

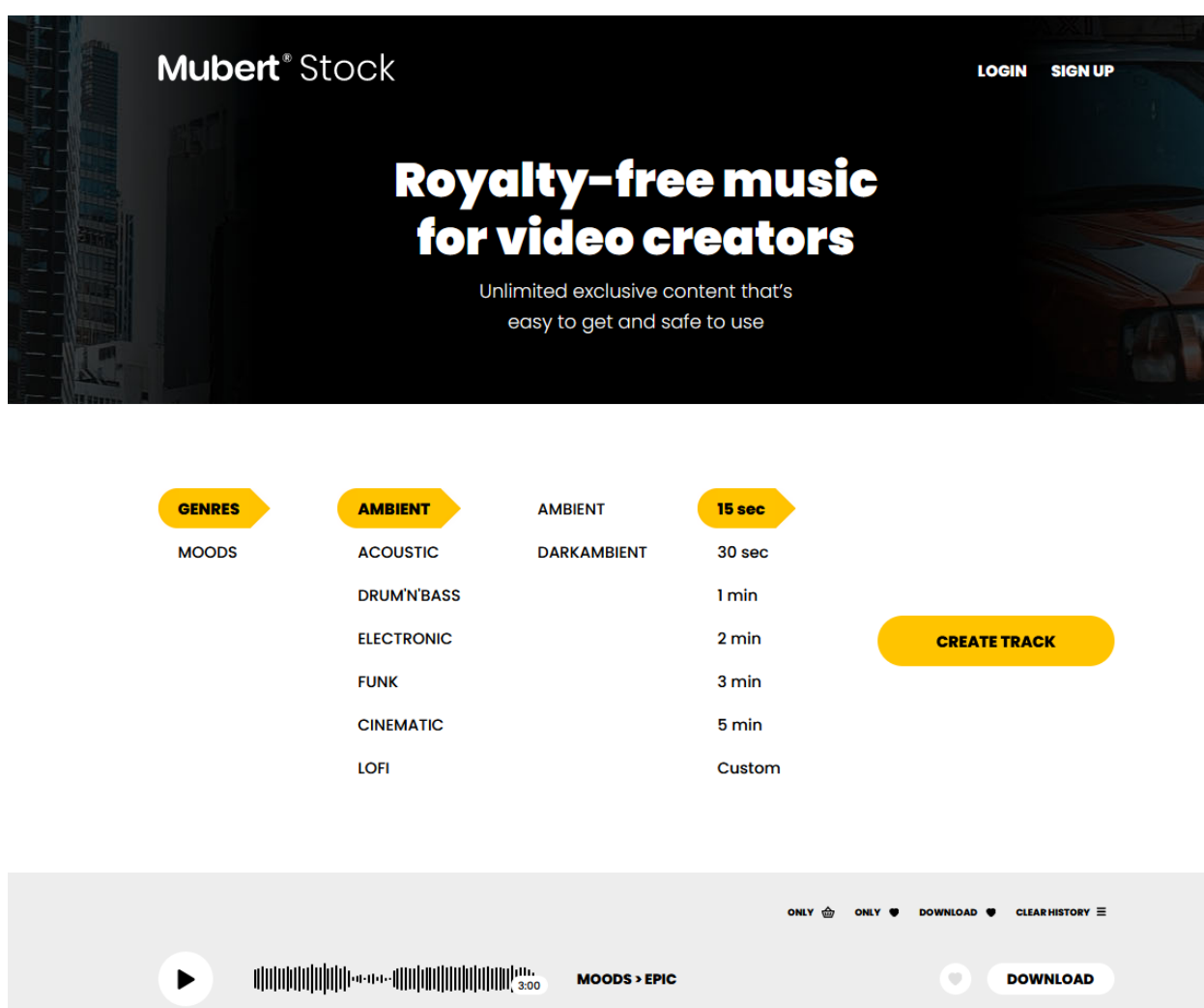


Рисунок 20 – Интерфейс сайта «MubertStock»"

Сервис предоставляет возможность генерировать музыку разной длины по разным настроениям и жанрам. Пользователь сам выбирает то, что ему нужно. Например, радостная музыка длиной 5 минут 15 секунд. И нейронная

сеть генерирует музыку по данным параметрам. Генерировать можно неограниченное число раз, при этом генерация происходит достаточно быстро даже для таких длинных композиций как приведенная в примере. Но данный сервис не предоставляет больше никаких возможностей задания дополнительных конфигураций композиции. Например, нельзя указать инструменты и ритм. Таким образом, с помощью данного сервиса можно сгенерировать интересную ни на что не похожую музыку, но для того, чтобы сгенерировать именно то, что нужно пользователю, может понадобиться достаточно длительное время.

2.2.3 Анализ сервиса «AmperScore»

С помощью данного сервиса можно генерировать музыку по различным, задаваемым пользователем, параметрам [26]. Также присутствует подписка, которая позволяет скачивать и использовать сгенерированные композиции. Интерфейс данного сайта представлен на рисунке 21.

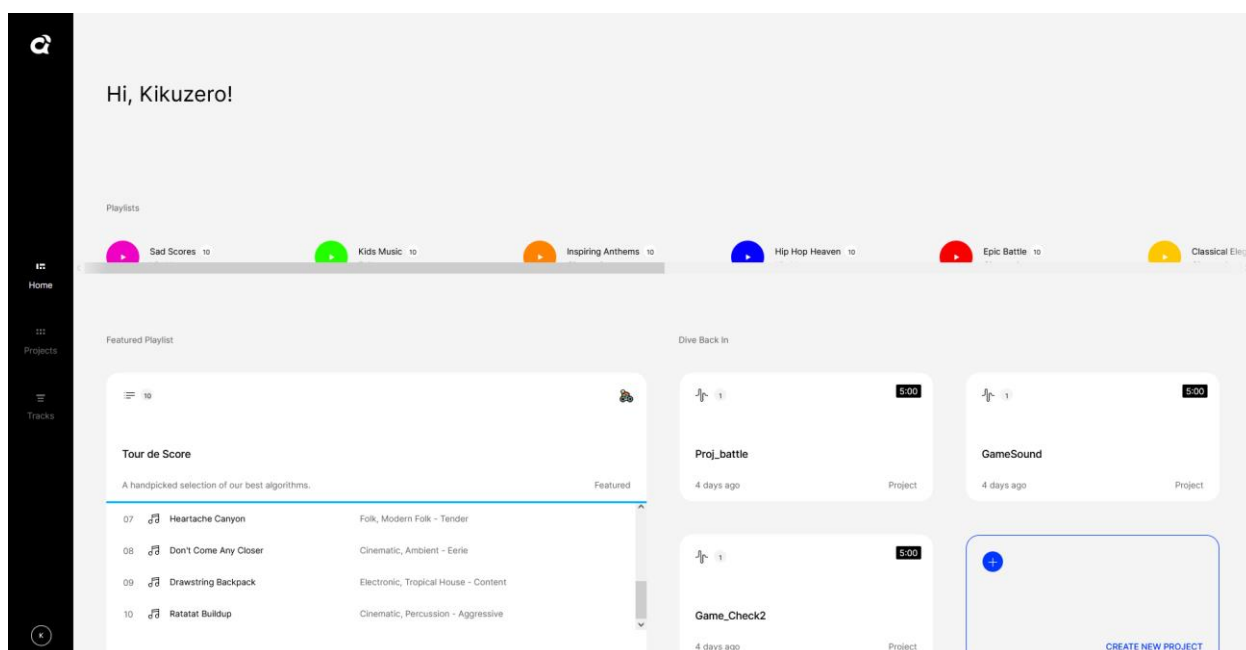


Рисунок 21 – Интерфейс сайта «AmperScore»

Данный сайт предоставляет возможность создать композицию определенного жанра и настроения. Также после генерации музыки по выбранному критерию, пользователь может изменить инструменты, которые

используются в музыке и поменять темп музыки. Главным минусом в данном плане является ограниченность инструментов по жанрам. То есть человек, выбрав первоначально определенный жанр ограничен им. Если на сайте после генерации музыки из инструментов можно выбрать только различные виды барабанов, то пользователь не сможет выбрать, например, пианино и добавить его звучание композиции. Таким образом сайт все же достаточно сильно ограничивает пользователя в выборе инструментов, пусть и дает его. Однако даже имеющийся функционал дает пользователю возможность генерировать уникальную музыку, по достаточно большому количеству критериев.

2.2.4 Результат анализа

В таблице 4 приведено сравнение описанных сервисов генерации музыки по критериям.

Таблица 4 – Сравнение сервисов по критериям

Характеристика	AI Duet	MubertStock	AmperScore
Высокие требования к навыкам игры на музыкальных инструментах	+	-	-
Возможность генерирования музыки определенного жанра	-	+	+
Возможность генерирования музыки определенного настроения	-	+	+
Возможность выбора инструментов в генерируемой музыке	-	-	+
Возможность выбора темпа звучания генерируемой музыки	-	-	+

Если при генерации необходима высокая гибкость, при этом пользователь умеет играть на пианино, лучше использовать «AI Duet», в других же случаях лучшим средством для генерации музыки из перечисленных является «AmperScore».

2.3 Проектирование интерфейса

На основании функциональных требований к системе был спроектирован интерфейс пользователя, состоящий из следующих страниц:

- а) страница главного меню;
- б) страница регистрации;
- в) страница входа в аккаунт;
- г) страница игры.

На рисунке 22 представлена страница главного меню.

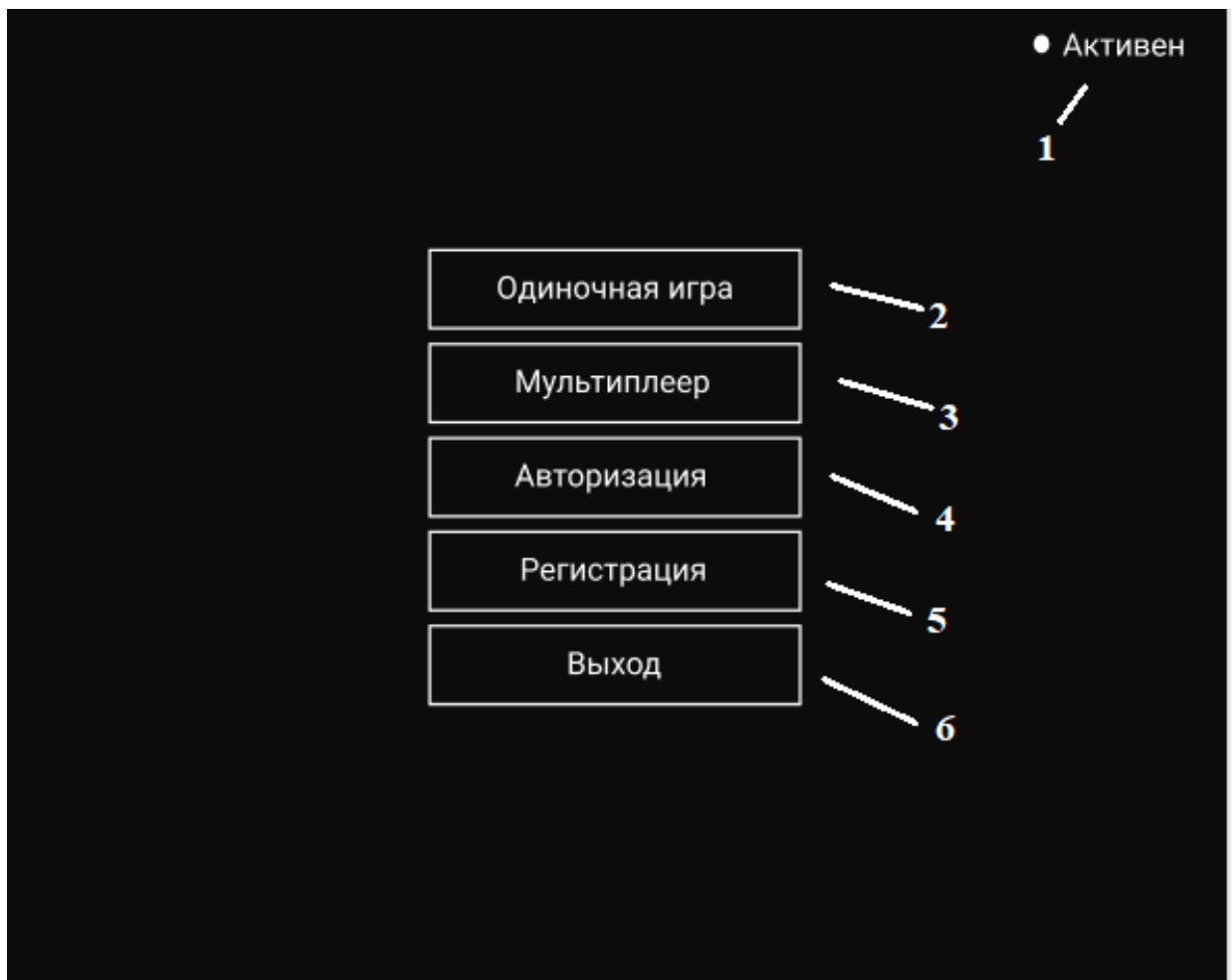


Рисунок 22 – Главное меню игры

Данная страница содержит следующие компоненты:

- 1 – идентификатор авторизован пользователь или нет;
- 2 – кнопка для создания одиночной сессии игры;

- 3 – кнопка для поиска матча кооперативной игры;
- 4 – кнопка для перехода на страницу авторизации;
- 5 – кнопка для перехода на страницу регистрации;
- 6 – кнопка для выхода из приложения.

На рисунке 23 представлена страница регистрации.

The image shows a registration form on a dark background. It includes two input fields for 'Логин:' (Login) and 'Пароль:' (Password), each with a placeholder 'Введите текст...' (Enter text...). Below these are two buttons: 'Назад' (Back) and 'Ок' (OK). A note at the bottom states: 'Логин и пароль могут содержать только символы латинского алфавита, цифры и символ "-".' (Login and password may contain only symbols of the Latin alphabet, digits, and the hyphen symbol). Numbered callouts point to specific elements: 1 points to the login input field, 2 points to the password input field, 3 points to the 'Ок' button, and 4 points to the 'Назад' button.

Рисунок 23 – Страница регистрации нового пользователя

Данная страница содержит следующие компоненты:

- 1 – поле ввода логина пользователя;
- 2 – поле ввода пароля пользователя;
- 3 – кнопка для инициации регистрации;
- 4 – кнопка для перехода на страницу главного меню.

На рисунке 24 представлена страница регистрации.

Данная страница содержит следующие компоненты:

- 1 – поле ввода логина пользователя;
- 2 – поле ввода пароля пользователя;

3 – кнопка для инициации входа в аккаунт;

4 – кнопка для перехода на страницу главного меню.



Рисунок 24 – Страница входа в аккаунт

На рисунке 25 представлена страница игры. На ней и происходит весь игровой процесс. Данная страница в целом достаточно динамична и потому в определенные моменты времени может не содержать некоторых элементов, представленных на изображении. Однако в среднем данная страница содержит следующие компоненты:

1 – поле игры;

2 – слайдер, отображающий количество здоровья противника;

3 – спрайт противника;

4 – оружие противника;

5 – слайдер, отображающий количество здоровья игрока;

6 – оружие игрока;

7 – спрайт игрока.

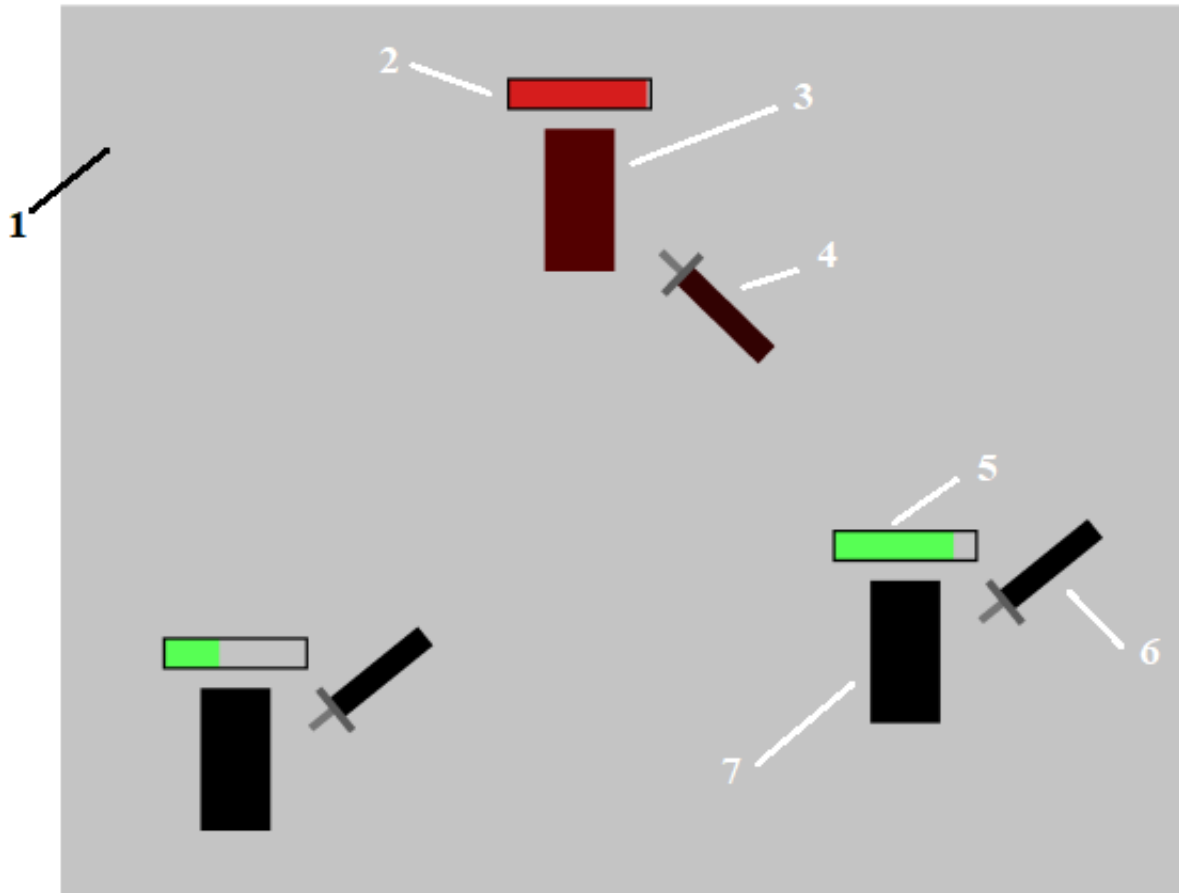


Рисунок 25 – Страница игры

В списке представлены элементы интерфейса, относящиеся к игроку, однако по рисунку видно, что элементы интерфейса, относящиеся к другому игроку, находящемуся в команде, выглядят идентично.

2.4 Проектирование искусственного интеллекта противников

2.4.1 Нейронная сеть в качестве контроллера противника

На данный момент времени, когда говорят «искусственный интеллект» обычно подразумевают нейронные сети. Однако нейронные сети не используется при проектировании искусственного интеллекта противников в играх. Обычное явление в данном случае – использование имитации

искусственного интеллекта, обычного конечного автомата. Причин для этого достаточно много, рассмотрим несколько из них.

Противника, контролируемого компьютером, в игре можно рассматривать как еще одного игрока. Таким образом, теоретически, если использовать нейронную сеть как контроллер для противника, можно воспользоваться и топологиями, и техниками обучения нейронных сетей, которые используются для имитации действий игрока. Но в таком случае появляется проблема с данными для обучения сети. Если рассматривать данные, которые используются для обучения имитации действий игрока, то они получаются при записи игры, которую ведет реальный игрок. То есть противник во время такой записи должен функционировать в полной степени. В данном же случае игра должна производиться как за игрока, так и за противника. То есть в данном случае будет необходимо реализовать функционал управления противником, а также создать возможность многопользовательской игры, что достаточно трудно. Но предположим, что данные все же получены. Наступает этап обработки этих данных, а затем непосредственно обучения. На данный момент времени правильно построенная и обученная нейронная сеть может, например, пройти уровень Марио или же несколько уровней Арканоида. Но обе эти игры, во-первых, двумерные, а во-вторых, персонажи в них обладают достаточно малым количеством состояний. Например, Марио может только ходить по горизонтали и прыгать. Но противник в игре, пусть и такой же двумерной, в идеале, должен уметь делать различные атаки, уже нет таких игр, где все противники просто идут на персонажа, как в том же Марио. Таким образом появляется проблема в обучении из-за большого количества состояний системы.

В итоге на всем пути такого обучения появляется проблема в небрежном использовании ресурсов. Даже на первом этапе для сбора данных уже нужна реализация многопользовательского режима, который в целом может и не понадобиться игре. А если в игре уже есть многопользовательский режим, и

пользователи могут сражаться друг с другом, то зачем такой игре в целом противники с искусственным интеллектом, если можно ограничиться и этой механикой. Также и на последнем этапе. Обучение такой системы может быть либо дешевым, но очень долгим, либо дорогим, но относительно долгим. В любом случае будут потрачены большие денежные и временные ресурсы, даже если будет выбран 1 вариант, ведь деньги идут не только на покупку видеокарт, но и на оплату работы программистов. Таким образом, использование нейронных сетей в качестве реализации искусственного интеллекта противников в игре, как минимум, экономически невыгодно.

К другим минусам данного подхода можно отнести то, что, во-первых, после обучения полученную модель уже практически невозможно изменить. То есть предположим, что в результате обучения получилось, что персонаж может ходить, использовать один из двух ударов. Но модель должна была обучиться так, чтобы она могла использовать оба удара. И в итоге даже если попытаться дообучить модель, велика вероятность неудачи [27]. И скорее всего модель придется учить заново. Второй проблемой является то, что чтобы понять, что модель обучилась удачно, нужно провести большое количество тестов, а так как сама по себе модель содержит трудно анализируемые данные, то оптимизировать этот процесс затруднительно. И третья проблема — это как раз трудно анализируемые данные. Одна из самых больших трудностей гейм-дизайна — создания баланса в игре. То есть, когда любое испытание, которое встречает игрок представляет для него некоторую сложность, но при этом является все еще преодолимым. Этот баланс задается грамотным дизайном уровня, а также изменением сложности противников. Из-за того, что данные, содержащиеся в модели трудно анализировать и в целом поведение такого противника трудно детерминировать на какие-либо фазы, усложняется сам гейм-дизайн и подход к созданию баланса в игре.

2.4.2 Конечный автомат в качестве контроллера противника

Всю деятельность противника можно разделить на различные состояния и на этой основе построить машину состояний [28], которая представляет собой обычный конечный автомат. В зависимости от окружающей среды противник будет менять свое состояние от одного к другому. При этом в данном случае наличие большого количества состояний не затруднит процесс разработки.

В данном случае на локации будет находиться всего 1 противник и он должен сразу же находиться в состоянии битвы. В данном случае у противника доступно 2 удара:

- а) удар по площади (круговая атака);
- б) локальный удар (по одному персонажу).

Таким образом для противника можно спроектировать несколько состояний в зависимости от местоположения игроков.

Первоначально, противник должен найти цель, за которой он будет первоначально следовать, чтобы нанести удар по ней. Очевидно, что самым простым решением в данном случае будет зафиксироваться на ближайшей цели. Таким образом состояние 1 – поиск цели.

После фиксации на цели должно производиться следование за целью. Таким образом состояние 2 – следование.

Результатом следования обязано быть либо переключение на другого игрока, либо удар. И для того и другого необходимо введения понятия «зоны поражения», то есть зоны, на которой противник уже может нанести удар. И в данном случае, состояния противника, следующие за «следованием», зависят именно от состояния этой зоны.

Предположим, в зону поражения попал только целевой противник, тогда противник должен нанести локальный удар. Состояние 3 – локальный удар. При этом цель все также остается на данном игроке.

Теперь предположим, что в зону поражения попали сразу же несколько игроков, в том числе и целевой, тогда противник должен нанести удар по площади. Состояние 4 – удар по площади. Цель также остается на том же игроке, на котором и была.

И тут появляется проблема. Если существуют только эти базовые состояния, то противник всегда будет следовать только за 1 игроком, на котором зафиксировался с самого начала игры. Но при этом очевидно, что из-за постоянного движения как противника, так и игроков, велика вероятность того, что в определенный момент времени может произойти ситуация, когда более успешным будет переключение цели на другого игрока, нежели следование за текущим. Нужно обработать такие ситуации.

Самым простым решением в данном случае будет ответная реакция противника на излишне агрессивные действия игрока. То есть, например, если во время преследования на пути противника появится игрок, который расположен в зоне поражения, будет нерационально продолжать преследование цели. Или же если по противнику, также во время преследования, был произведен удар игроком, то очевидно, что игрок опять же находится слишком близко, и целью можно назначить именно его. В остальных же случаях противник должен продолжать преследовать уже определенную цель. Также, очевидно, нужно производить смену цели, если преследуемый игрок был убит после удара.

Таким образом получается интеллект противника, который может ходить, бить игроков и переключаться между ними при необходимости. И в целом это необходимый базис, который должен уметь делать любой боевой противник. Но в данном случае появляется еще одна проблема – баланс. С точки зрения гейм-дизайна противник должен быть сложен, иногда даже слишком сложен, когда дело касается кооперативных игр с упором на боевую составляющую. Однако он все равно должен быть победим. И в описанных состояниях есть одна проблема. И раскрывается она во время непосредственно боевого столкновения. Если у противника будет возможность производить

атаку постоянно, он будет производить ее постоянно. Но проблема состоит в том, что если позволить противнику такую же возможность, то он может стать практически непобедимым. У противника в данном случае больше здоровья чем у игрока и больше урон. Соответственно при прямом столкновении игрока и противника, игрок проиграет в любом случае. Таким образом нужно создать задержку между ударами противника, чтобы дать игроку возможность совершить какое-либо действие. Например, отойти в сторону и не дать противнику попасть по себе. Таким образом можно построить диаграмму состояний, представленную на рисунке 26.

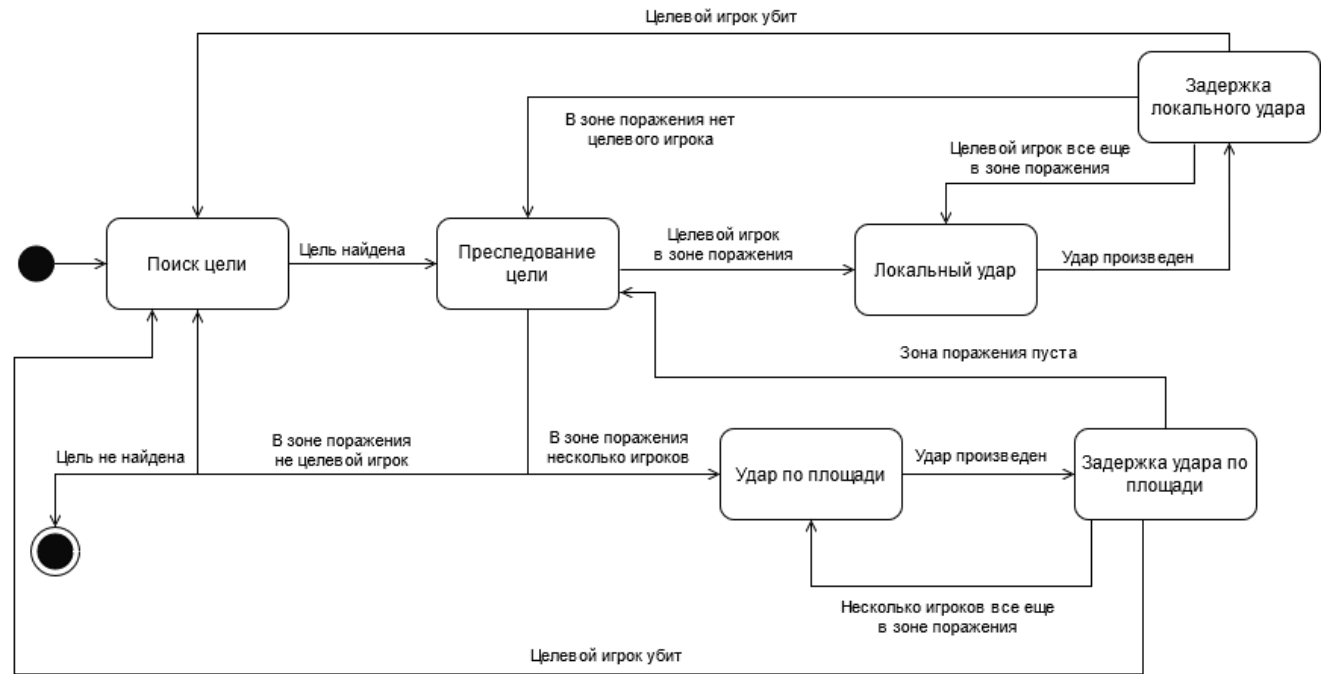


Рисунок 26 – Диаграмма состояний искусственного интеллекта противника

Данная диаграмма в полной мере описывает действия противника в момент игры в зависимости от окружающей среды.

2.4.3 Проектирование боевой системы

Обычно боевая система контролируется нажатием каких-либо кнопок. В компьютерных видеоиграх это обычно правая и левая кнопка мыши и в зависимости от того, какая из кнопок нажата, происходят разные удары. В играх с видом сверху (или изометрических) часто бывает, что у персонажа есть

своего рода прицел, по которому можно понять в какую сторону персонаж будет производить удар. И очевидно, направление этого прицела меняется в зависимости от поворота героя, либо же от положения мыши на экране. В последнем случае этот прицел будет направлен на то место, куда указывает мышь.

В случае данной игры было принято решение достаточно сильно видоизменить классическую боевую систему. Если ранее мышью пользователь мог изменять лишь положение прицела, и чтобы нанести удар ему также приходилось нажимать какие-либо кнопки, то в данном случае ему достаточно переместить мышь и оружие переместится за «прицелом». Соответственно, таким образом будет нанесен удар.

Данное видоизменение имеет несколько преимуществ. Первое из них это гибкость. Так как в классическом случае все комбо удары жестко вписаны в боевую систему, пользователю не остается места для придумывания чего-то своего. В данном же случае пользователь может сам придумывать свои комбо. Вторым преимуществом является уменьшение количества анимаций. Так как данная игра ориентирована на сетевое взаимодействие, то ограничение количества синхронизируемых данных, значительно упрощает сетевую разработку. Большое количество комбо и соответственно анимаций к ним, напротив, усложнят разработку, так как анимации будут нуждаться в синхронизации на различных клиентах.

На игровом поле персонаж и его оружие представляют собой 2 сущности. Оружие во время удара осуществляет вращение вокруг персонажа, как если бы это был прицел.

В данной игре, помимо игроков, есть и искусственный интеллект, потому нужно спроектировать боевую систему и для него. Визуально она не должна отличаться от боевой системы обычного игрока. Потому самым логичным решением в данном случае является написание своего эмулятора мыши, или же генератора данных мыши.

В данном случае мышь можно рассматривать как стик на геймпаде, который движется по окружности определенного радиуса. Геймпад это еще один вид устройства ввода данных. Однако в отличие от мыши и клавиатуры, данный тип устройства используется только в играх. Стик на геймпаде – рычаг, который может принимать различные положения в пределах определенной окружности. Так как мышь рассматривается как стик, и движение происходит только по окружности, в данном случае угол поворота является наиболее полным описанием системы.

Диаграмма деятельности алгоритма атаки представлена на рисунке 27.



Рисунок 27 – Диаграмма деятельности алгоритма атаки

Алгоритм атак в данном случае достаточно прост. На вход поступает угол между осью X и вектором от положений босса и его цели, а также угол удара оружия. Затем происходит нахождение углов начала движения и конца движения. Для нахождения конца и начала движения как раз нужен угол удара оружия и угол, на котором находится игрок по отношению к противнику. Угол удара обозначим как УУ, а угол, на котором находит игрок по отношению к противнику как УИ. Тогда конечный угол (КУ) движения оружия это:

$$КУ = УИ + \frac{УУ}{2}.$$

Тогда как начальный угол (НУ) это:

$$НУ = УИ - \frac{УУ}{2}.$$

Соответственно, если конечный угол больше 360, то он будет обновлен на значение: $КУ \bmod 360$. Если же начальный угол меньше 0, то он будет обновлен до $360 + (НУ \bmod 360)$.

Производится перемещение виртуальной мыши в положение угла начала движения. Затем с каждой итерацией происходит обновление угла виртуальной мыши до того момента, пока угол не выйдет за пределы дуги, полученной из углов начала и конца движения.

Таким образом в зависимости от угла оружия можно менять и тип ударов, так, например, угол 60 может быть рассмотрен как локальный удар, а угол 340, как удар по площади.

3 Реализация игрового приложения

3.1 Реализация графической составляющей игрового приложения

Для данного игрового приложения нужно было реализовать несколько визуальных составляющих:

- а) уровень;
- б) персонажи.

Уже на этапе анализа требований было принято решение, что для 3D моделей будет использована программа MagicaVoxel. Данная программа имеет широкий набор функций для работы с воксельной графикой. Таким образом был получен уровень в воксельном его представлении (рисунок 28).

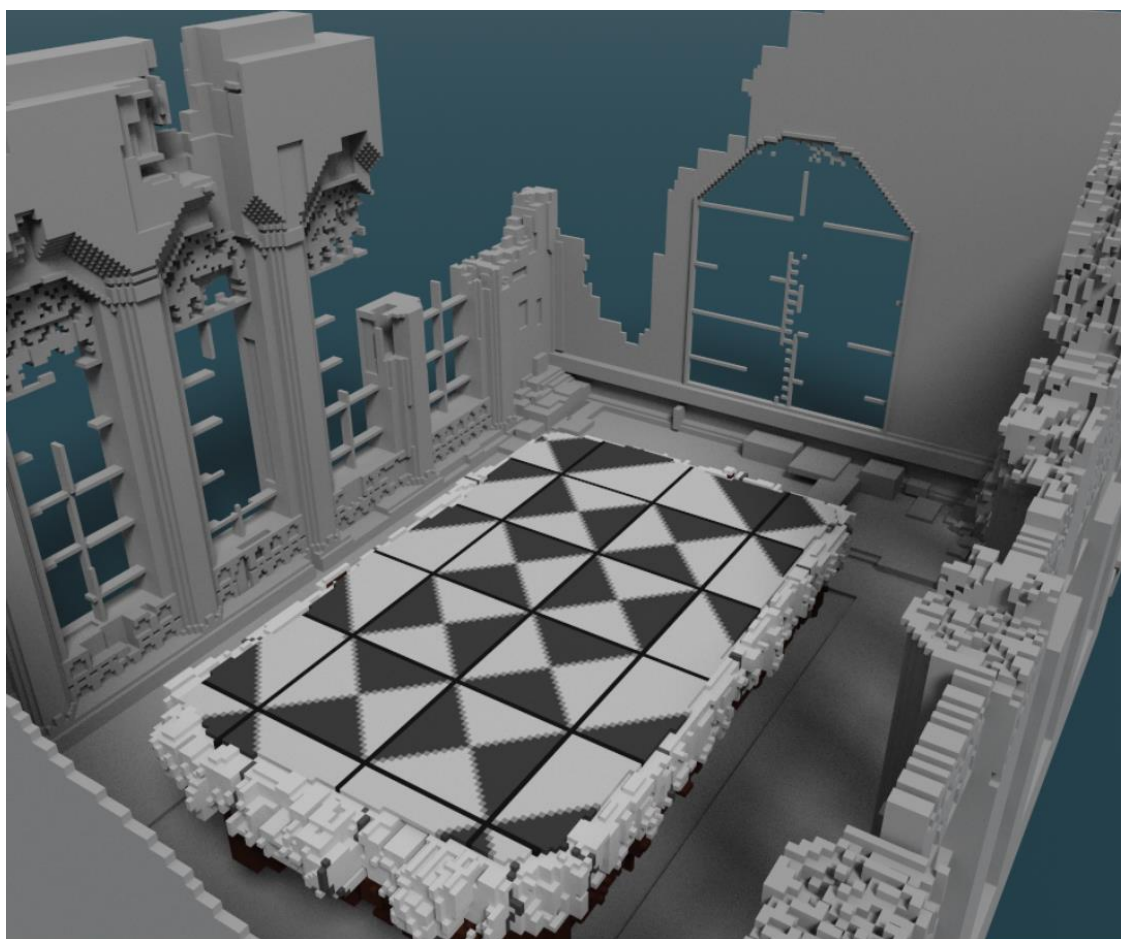


Рисунок 28 – Уровень игры

Также основным решением в графическом плане при реализации визуальной составляющей игры было смешение 3D мира и 2D персонажей. Так как мир в данном плане представляет из себя 3D объекты в воксельном

виде, то очевидным решением реализации 2D персонажей был пикселизированный вид. Размер каждого спрайта персонажей 32 на 32 пикселя. Спрайты босса и персонажа представлены на рисунках 29, 30.



Рисунок 29 – Спрайт босса



Рисунок 30 – Спрайт персонажа

Основная особенность босса в данном случае состоит в том, что он должен визуально максимально отличаться от окружения. А так как цветовая составляющая как окружения, так и персонажей представляет собой в основном серый и его оттенки, то для босса был выбран контрастный красный оттенок.

3.2 Реализация игровых механик

Для реализации приложения был выбран архитектурный подход ECS. Также, для более удобного его использования был написан ECS-фреймворк, реализующий управление системами.

Главная особенность данного фреймворка – фильтрация сущностей через узлы. Узел содержит в себе компоненты. Системы же в данном случае используют список узлов. Формируется этот список через фильтрацию компонентов сущностей. Если множество компонентов узла является подмножеством компонентов сущности, то в список узлов заносится узел с данными компонентов сущности.

Полный код приложения находится в приложении Б.

3.2.1 Реализация системы перемещения

В рамках реализации игровой системы перемещения игрока было реализовано несколько систем и компонентов в составе ECS (рисунок 31).

Опишем несколько классов, входящий в данную игровую систему.

`ResetDirectionSystem` – система нужная для обнуления значения `DirectionComponent`, указывающий на актуальный вектор движения объекта.

`CalculateDirectionSystem` – система по `KeysComponent` считывает какие клавиши нажаты, генерирует вектор направления, который будет положен в `DirectionComponent`.

`CalculateVelocitySystem` – на основе `DirectionComponent` высчитывает вектор скорости, который будет положен в `VelocityComponent`.

`MoveCharacterSystem` – применяет полученный `VelocityComponent` к игровому объекту на сцене.

`ResetInputKeysSystem` – обнуляет значения `KeysComponent`.

Важно также отметить компонент `IsMoveEnableComponent`. Данный компонент не несет в себе никакой информации в прямом смысле этого слова. Он пуст. Однако, если он не будет прикреплен к обрабатываемой системой `MoveCharacterSystem` сущности, то эта сущность не будет перемещена. То есть, пусть в данном классе и нет никакой информации, но сам факт присоединения его как компонента уже несет в себе информацию.

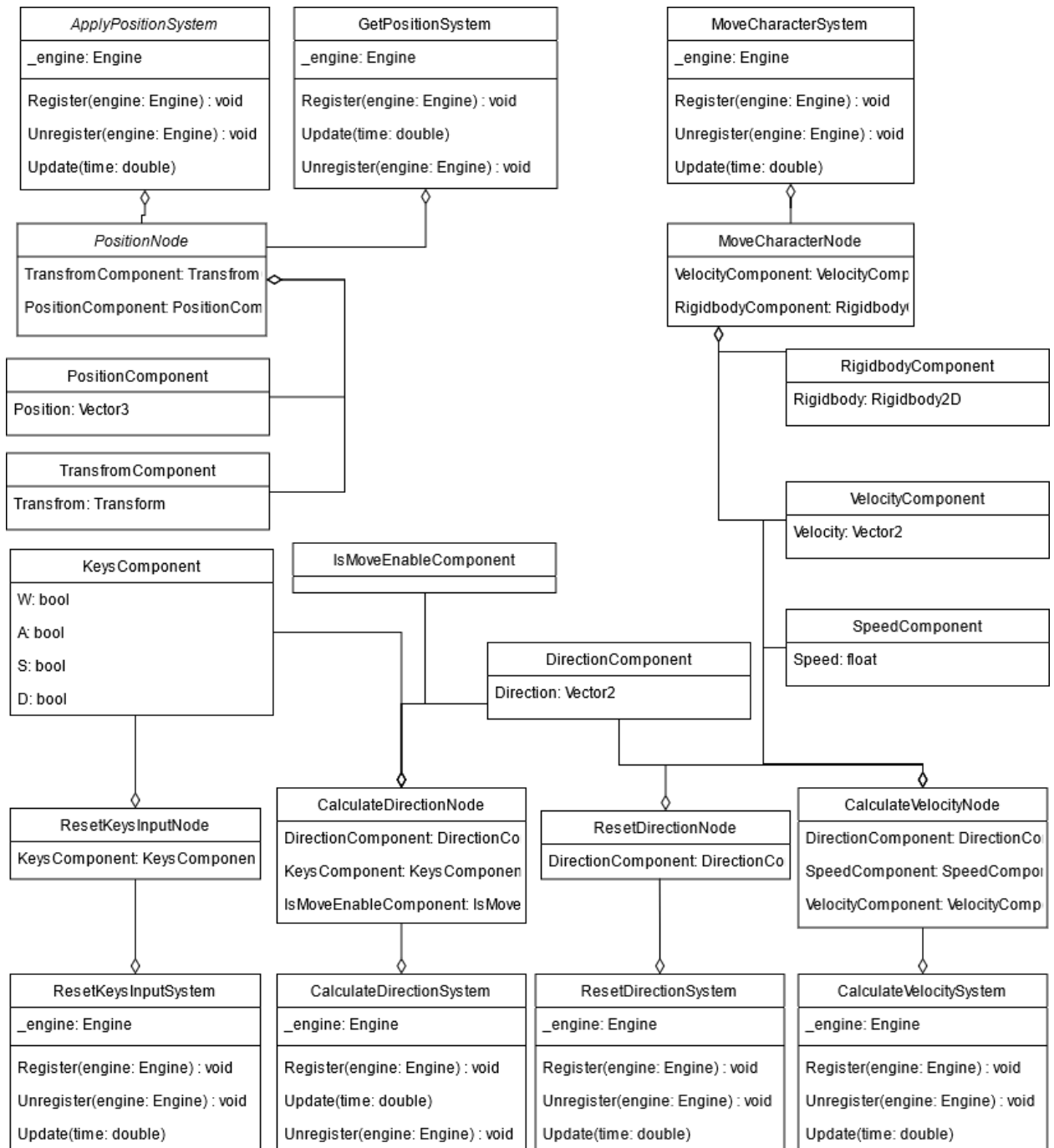


Рисунок 31 – Диаграмма классов игровой системы перемещения

В рамках реализации игровой системы перемещения противников было реализовано несколько систем и компонентов в составе ECS. Диаграмма классов, показывающая связь данных систем и компонентов, реализованного искусственного интеллекта для перемещения, представлена на рисунке 32.

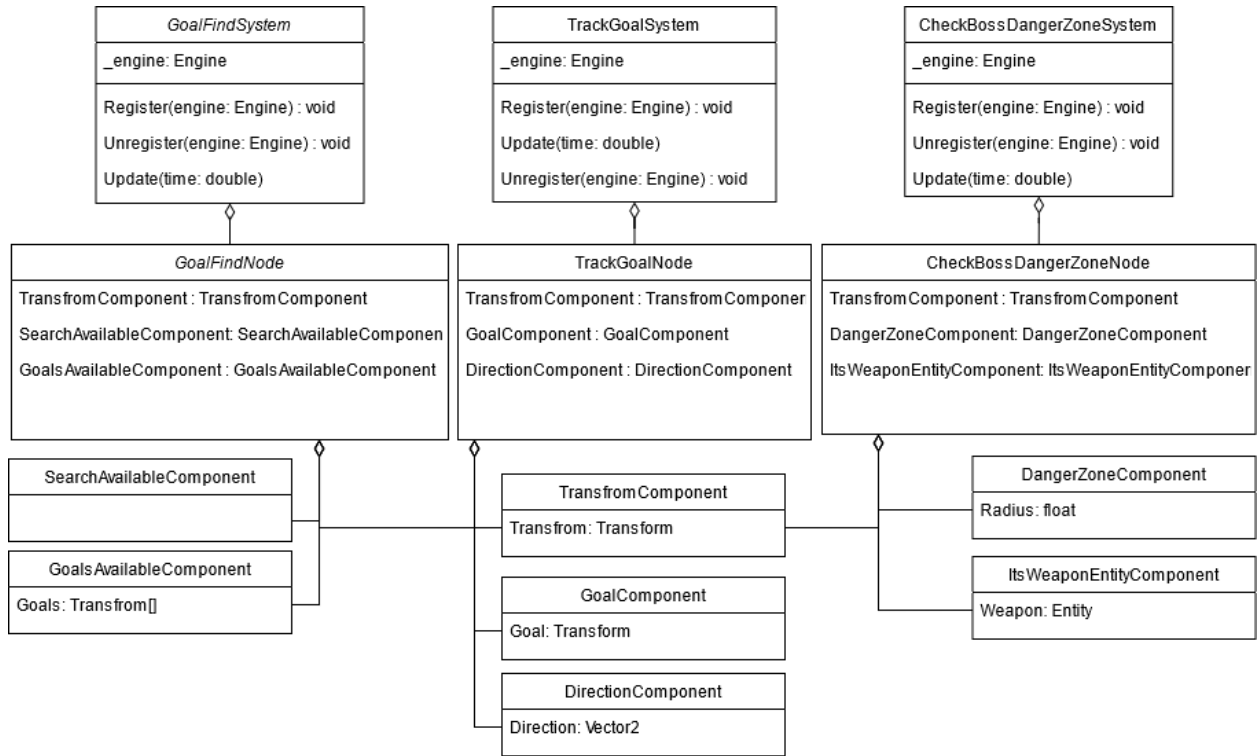


Рисунок 32 – Диаграмма классов игровой системы перемещения

Опишем несколько классов, входящий в данную игровую систему.

GoalFindSystem – система для поиска цели противника, с помощью компонента GoalsAvailableComponent находит ближайшую цель, перемещает ее Transfrom в GoalComponent.

TrackGoalSystem – система следования за целью, создает направление на цель, заполняет компонент DirectComponent.

CheckDangerZone – система, из-за которой может быть изменена цель. Если какой-либо объект войдет в зону поражения, он автоматически становится целью.

3.2.2 Реализация боевой системы

Для реализации боевой системы игровых персонажей было реализовано несколько систем и компонентов в составе ECS (рисунок 33).

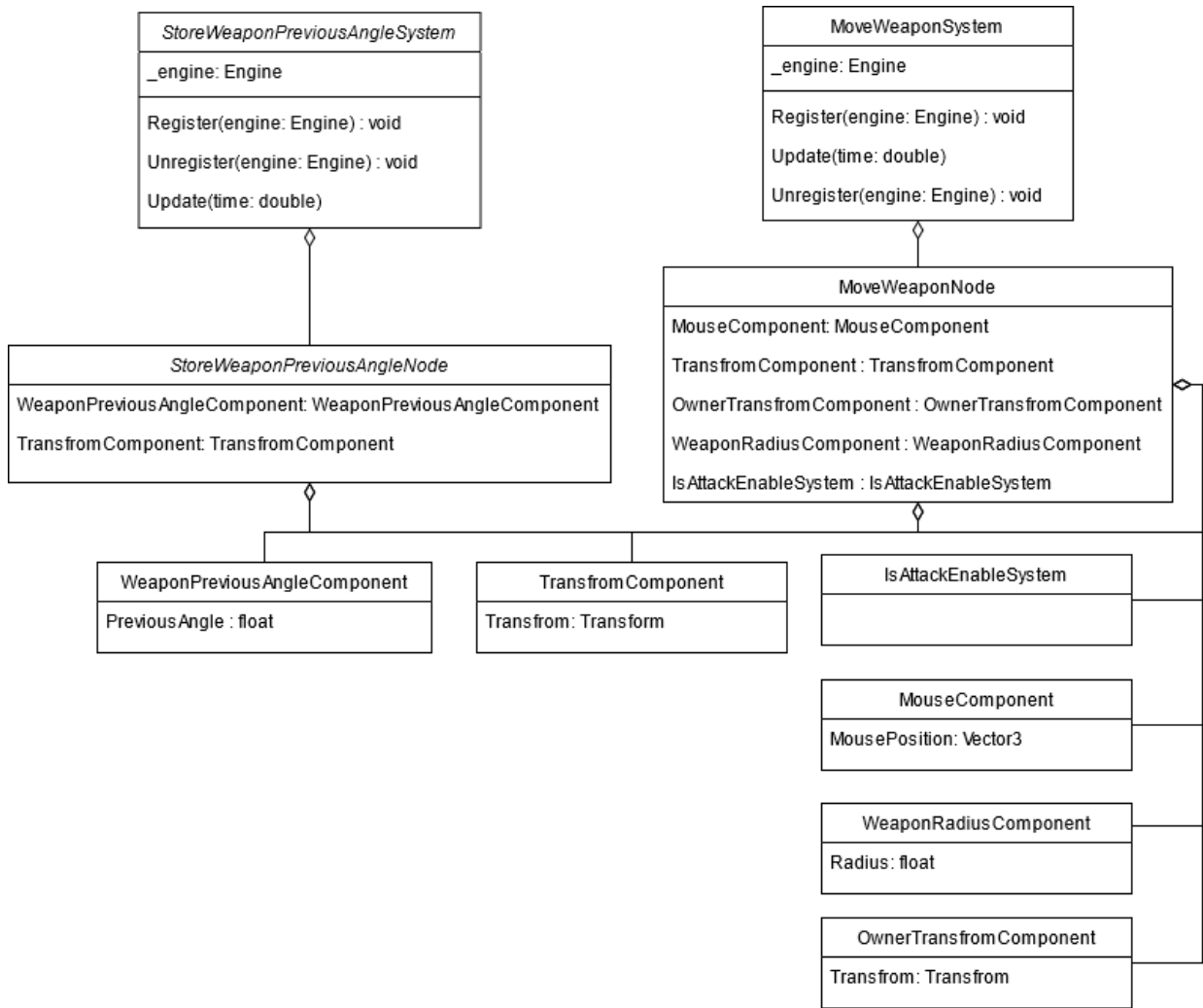


Рисунок 33 – Диаграмма классов боевой системы игровых персонажей

Опишем несколько классов, входящий в данную игровую систему.

StoreWeaponPreviousAngleSystem – система, с помощью которой высчитывается насколько изменился угол поворота оружия. Если угол больше определенного, то урон будет нанесен, если нет, то урон по объекту нанес не будет.

MoveWeaponSystem – перемещает оружие объекта относительно него по кругу, в зависимости от местоположения мыши.

По своему смыслу боевая система противников ничем не отличается от боевой системы игровых персонажей. Основное отличие здесь именно в способе ее реализации. Очевидно, что перемещением оружия у игрового

персонажа занят сам игрок. В данном же случае перемещать оружие должен искусственный интеллект, когда противник находится в стадии атаки.

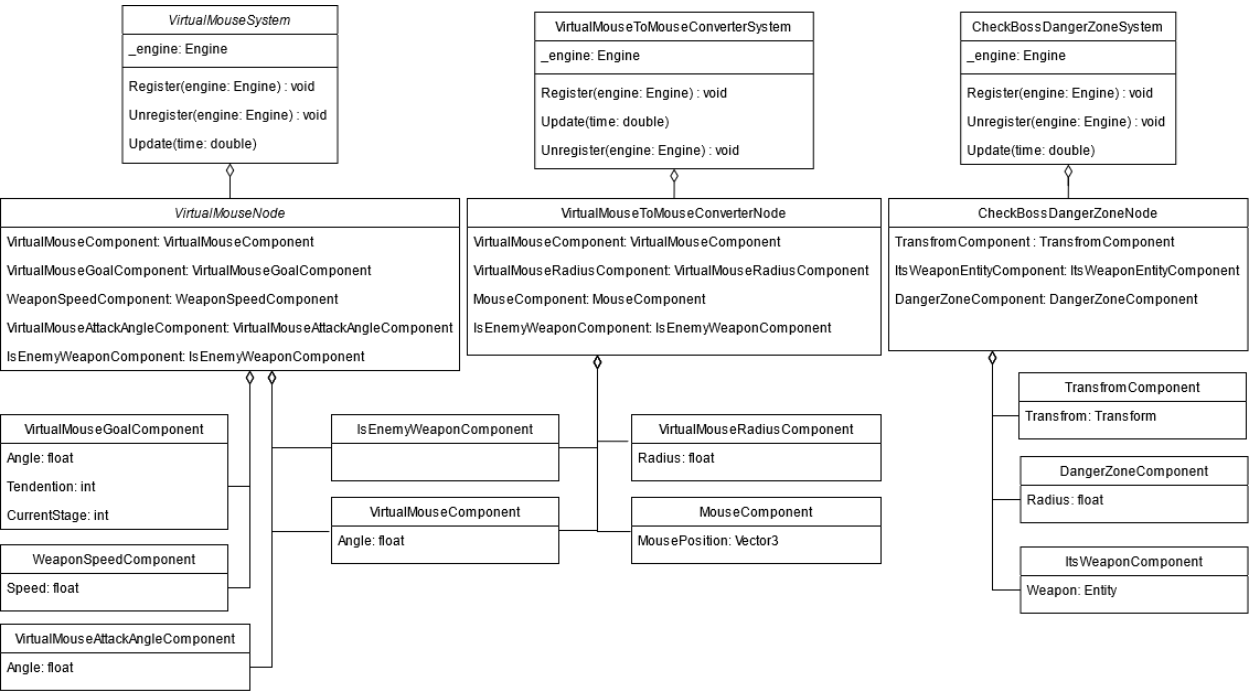


Рисунок 34 – Диаграмма классов боевой системы противников

Для реализации данной игровой системы можно воспользоваться некоторыми ECS системами, реализованными для игровых персонажей. Но в данном случае управлять «мышью» должен уже искусственный интеллект. Таким образом, в рамках данной игровой системы был написан «эмулятор» мыши. Он и другие компоненты и системы представлены на рисунке 34.

Опишем несколько классов, входящий в данную игровую систему.

CheckDangerZoneSystem – система, с помощью которой противник узнает, что время атаковать. Также в этой системе выбирается тип атаки.

VirtualMouseSystem – система определяет насколько и как нужно повернуть виртуальную мышь.

VirtualMouseToMouseConverterSystem – система нужная для конвертации данных. Так как виртуальная мышь работает лишь с углами и ничего не знает о координатах, а компонент мыши работает напротив только с координатами, нужен конвертер одних данных в другие.

4 Контроль качества программного обеспечения

4.1 Функциональное тестирование клиентского приложения

Функциональное тестирование – тестирование, проводимое с целью проверки способности разрабатываемого ПО удовлетворять поставленным функциональным требованиям. То есть проверка того, может ли ПО в различных условиях, описанных в тестах, выполнять задачи, необходимые пользователю.

Произведем функциональное тестирование клиентского приложения. План функционального тестирования клиентского приложения представлен в таблице 5.

Таблица 5 – План функционального тестирования клиентского приложения

№	Вариант использования	Тест	Результат
1	Зарегистрироваться в системе	Произвести регистрацию пользователя с корректным, не занятым логином и корректным паролем.	Тест пройден. Систем оповестила об успешной регистрации (рисунок Г.1).
2		Попытаться произвести регистрацию пользователя с некорректным логином.	Тест пройден. Система оповестила о некорректных символах в логине (рисунок Г.2).
3		Запустить сервер. Попытаться произвести регистрацию пользователя с некорректным паролем.	Тест пройден. Система оповестила о некорректных символах в пароле (рисунок Г.3).
4		Запустить сервер. Попытаться произвести регистрацию пользователя с занятым логином.	Тест пройден. Система оповестило о том, что логин занят (рисунок Г.4).
5		Попытаться произвести регистрацию пользователя без логина.	Тест пройден. Система оповестила о том, что не указан логин (рисунок Г.5).
6		Попытаться произвести регистрацию пользователя без пароля	Тест пройден. Система оповестила о том, что не указан пароль (рисунок Г.6).
7		Не запускать сервер. Попытаться зарегистрировать пользователя с корректным логином и паролем.	Тест пройден. Система оповестила о том, что не удалось установить соединение с сервером (рисунок Г.7).
8	Войти в систему	Произвести вход в систему с использованием логина и пароля зарегистрированного пользователя.	Тест пройден. Система оповестила об успешном входе в систему (рисунок Г.8).

Продолжение таблицы 5

№	Вариант использования	Тест	Результат
9	Войти в систему	Попытаться войти в систему под неверным логином.	Тест пройден. Система оповестила о том, что использован неверный логин или пароль (рисунок Г.9).
10		Попытаться войти в систему под неверным паролем.	Тест пройден. Система оповестила о том, что использован неверный логин или пароль (рисунок Г.10).
11		Не запускать сервер. Попытаться войти в систему.	Тест пройден. Система оповестила о том, что не удалось установить соединение с сервером (рисунок Г.11).
12	Найти матч	Войти в систему за двух пользователей. Инициировать пользователями поиск матча.	Тест пройден. Оба пользователя попало в один матч (рисунок Г.12).
13		Войти в систему. Инициировать поиск матча. Прервать поиск матча.	Тест пройден. Произошел переход в главное меню (рисунок Г.13).
14		Войти в систему за двух пользователей. Инициировать пользователями поиск матча. Дождаться этапа соединения. Отменить поиск матча одним из пользователей.	Тест пройден. Система оповестила о разрыве соединения (рисунок Г.14).
15		Дважды войти в систему за одного пользователя. Инициировать поиск матча.	Тест пройден. Система оповестила о том, что пользователь уже находится в очереди (рисунок Г.15).
16		Не запускать сервер. Инициировать поиск матча.	Тест пройден. Система оповестила о том, что не удалось установить соединение с сервером (рисунок Г.16).

Функциональное тестирование приложения показало корректность его работы.

4.2 Модульное тестирование клиентского приложения

Модульные тесты, или юнит тесты, – тесты юнитов кода. Под юнитом подразумевается отдельно взятая законченная функциональность, это может быть, например, функция или метод.

Внутри Unity есть инструменты по поддержке модульных тестов, использующие библиотеку NUnit. Таким образом не нужно использовать

каких-либо сторонних библиотек. Функционал Unity Testing позволяет как проверять работоспособность модулей, для которых не нужен запуск игры, так и тех, для которых запуск необходим. Примером таких модулей, могут быть функции или методы, взаимодействующие с физикой Unity.

В данном случае юнит тестирование очень удобно для проверки работоспособности ECS-систем. Каждая система в ECS выполняет какую-либо законченную функциональность. И для того, чтобы вся программа работала правильно, достаточно, чтобы правильно работали все ее системы. Таким образом нужно написать по одному модульному тесту на каждую систему. Результат модульного тестирования представлен на рисунке 35, описание проводимых тестов представлено в таблице 6.

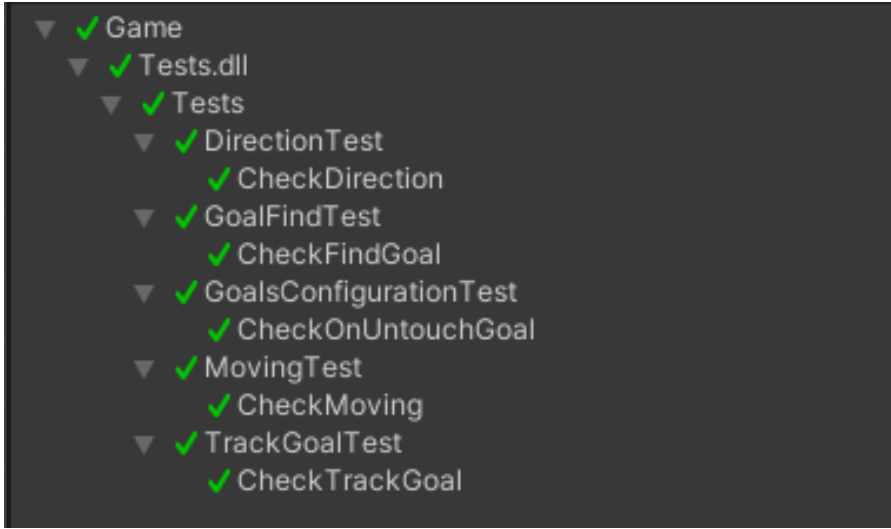


Рисунок 35 – Результаты модульного тестирования

Таблица 6 – Описание проводимых модульных тестов

Тестовый класс	Проверяемый класс	Тестовый метод	Описание
DirectionTest	CalculateDirection System	CheckDirection	Проверяет корректность расчета направления движения
GoalFindTest	GoalFindSystem	CheckFindGoal	Проверяет корректность нахождения цели
GoalConfiguration Test	AvailableGoals ConfiguratorSystem	CheckOnUntouchGoal	Проверяет удаление цели из списка целей

Продолжение таблицы 6

Тестовый класс	Проверяемый класс	Тестовый метод	Описание
MovingTest	MoveCharacter System	CheckMoving	Проверяет корректность перемещения персонажа в пространстве
TrackGoalTest	TrackGoalSystem	CheckTrackGoal	Проверяет корректность определения направления движения к цели

Данная таблица не содержит наименований проверяемых методов, так как из-за специфики ECS-систем наименование всех проверяемых методов в нем – Update.

Таким образом с помощью модульных тестов были проверены основные методы приложения.

4.3 Исследовательское тестирование

При тестировании игр часто используется исследовательское тестирование. Оно подразумевает под собой тестирование основных критических сценариев приложения, имитируя действия пользователя. Основная причина использования такого вида тестирования состоит в том, что исследовательские тесты проводятся значительно быстрее. Также они не требуют полного описания всех возможных состояний тестируемой системы, необходимых для тест-кейсов. А так как у игры количество состояний велико, то и количество тест-кейсов будет велико. В исследовательском тестировании количество тест-кейсов будет значительно меньше, из-за ориентации лишь на основные критические точки приложения.

В рамках данного тестирования были выделены основные тестовые случаи. В рамках данных игровых случаев, покрывающих основные игровые механики, были выделены более конкретные тесты. Результат тестирования представлен в таблице 7.

Таблица 7 – Результаты исследовательского тестирования

Игровая механика	Тест	Результат
Перемещение персонажа	Перемещение персонажа в различные стороны	Тест пройден
	Попытка перемещения персонажа за пределы карты	Тест пройден
Урон	Нанести урон другому персонажу	Тест пройден
	Получить урон	Тест пройден
Боевая система	Перемещение оружия на различные угла	Тест пройден
Конец игры	Победа	Тест пройден
	Поражение	Тест пройден

4.4 Метрики кода

|Произведем расчет метрик кода для клиентской части приложения средствами Visual Studio (рисунок 36).

Core (Debug)	89	692	8	221	3 556	892
AnimatorComponent	100	1	2	2	4	0
CameraComponent	100	1	2	2	10	0
ColliderComponent	100	1	2	2	10	0
DamageComponent	100	1	2	1	10	0
DangerZoneComponent	100	1	2	1	4	0
DirectionComponent	100	1	2	4	11	2
DirectionComponentConverter	90	2	1	7	11	3
EndGameComponent	100	1	2	2	5	0
EnemyFlagComponent	100	1	2	1	3	0
GoalComponent	100	1	2	1	4	0
GoalsAvailableComponent	100	1	2	2	4	0
HealthBarComponent	100	1	2	2	4	0
HealthComponent	100	1	2	3	15	2
HealthComponentConverter	82	2	1	5	16	4
IdComponent	100	1	2	3	14	2
IdComponentConverter	90	2	1	5	11	3
IsAliveComponent	100	1	2	1	3	0
IsAttackEnableComponent	100	1	2	1	4	0
IsDeadComponent	100	1	2	1	3	0
IsEnemyWeaponComponent	100	1	2	1	3	0

Рисунок 36 – Результаты расчета метрик с помощью Visual Studio

Средний индекс удобства поддержки (Maintainability Index) может находиться на отрезке от 0 до 100, где значения от 20 до 100 означают высокое удобство поддержки. Для данного решения метрика составила решения составил 85, что говорит о высокой простоте поддержки кода.

Сложность организации циклов (Cyclomatic Complexity) позволяет оценить структурную сложность программы. Чем больше значение, тем сложнее структура и тем больше требуется тестов для покрытия всей

программы. Так как на рисунке 36 представлены метрики на уровне целых проектов, полученные результаты приемлемы.

Глубина наследования (Depth of Inheritance) указывает количество классов, наследуемых друг от друга вплоть до базового. Высокое значение недопустимо, так как это будет означать, что внести изменения в проект будет практически невозможно. Для проекта в решении глубина наследования равна 8, что является хорошим показателем.

Взаимосвязь классов (Class Coupling) позволяет оценить зависимость классов друг от друга. Высокое значение данного параметра указывает на то, что вносить изменения в данный проект будет затруднительно. Представленные значения взаимосвязи являются приемлемыми [32].

Таким образом, по данным оценкам можно определить, что разработанный код имеет достаточно высокий уровень качества.

Произведем оценку качества метода Update ECS-системы CheckBossDangerZone, которая имеет ключевое значение в искусственном интеллекте противника. Метод представлен в листинге 1.

Листинг 1 – Метод CheckBossDangerZone

```
public override void Update(double time)
{
    foreach (var node in _engine.GetNodes<Node>().ToList())
    {
        if
(node.ItsWeaponEntityComponent.Weapon.Contain<VirtualMouseGoalComponent>()) continue;

        var colliders =
Physics2D.OverlapCircleAll(node.TransformComponent.Transform.position,
node.DangerZoneComponent.Radius, LayerMask.GetMask("Characters"));

        if (colliders.Length == 1) continue;

        var goal = GetGoals(colliders);
        if (goal.Count == 0)
        {
            SetFindActive(node);
            continue;
        }

        var angle = GetAngleBetween(goal[0].gameObject.transform,
node.TransformComponent.Transform);

        node.ItsWeaponEntityComponent.Weapon.Add(new VirtualMouseGoalComponent()
{ Tendency = 1, Angle = angle });
        node.ItsWeaponEntityComponent.Weapon.Add(new IsPrepareStateComponent());
    }
}
```

```

        if (colliders.Length != 2)
            node.ItsWeaponEntityComponent.Weapon.Add(new
            IsWideAttackComponent());
        else
            node.ItsWeaponEntityComponent.Weapon.Add(new
            IsShortAttackComponent());
    }
}

```

Произведем для него расчет цикломатической метрики Мак-Кейба и метрики Холстеда.

4.5 Цикломатическая метрика Мак-Кейба

Цикломатическая метрика Мак-Кейба (цикломатическая сложность) позволяет оценить структурную сложность программы. Для того, чтобы оценить цикломатическую сложность метода CheckBossDangerZone построим граф потока управления данного метода (рисунок 36).

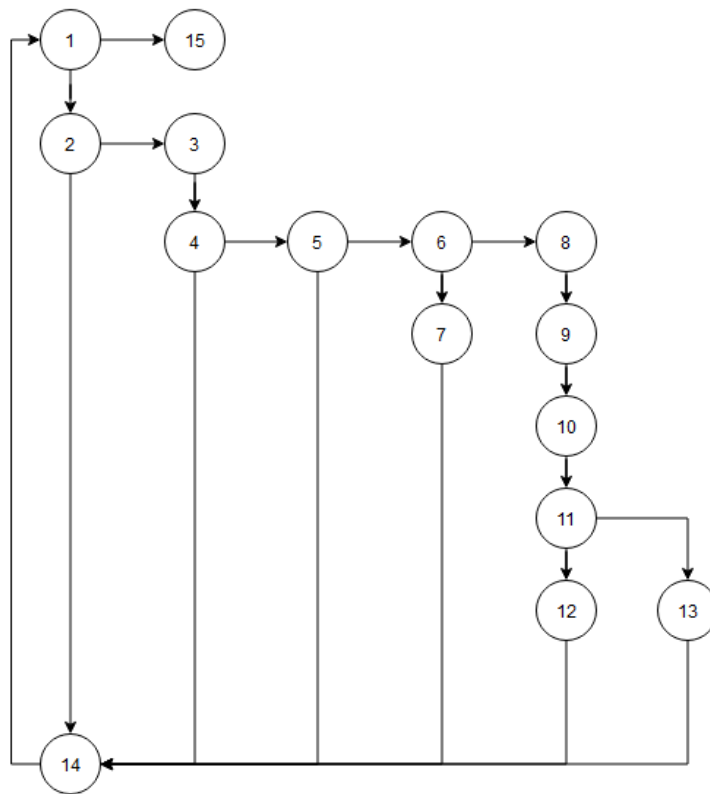


Рисунок 36 – Граф потока управления метода Update

Цикломатическая сложность вычисляется по следующей формуле:

$$V(G)=e-n+2p,$$

где e – количество дуг;
 n – количество вершин;
 p – число компонент связности.

Граф потока управления метода CheckBossDangerZone содержит 11 вершин и 15 дуг. Подставив полученные значения: $V(G)=20-15+2=10$.

Полученное значение цикломатической сложности не более 10, что говорит о структурной простоте метода CheckBossDangerZone. Однако, так как это значение находится на границе «нормальности», то имеет смысл упростить данный метод.

4.6 Метрика Холстеда

Для расчета метрик Холстеда необходимо измерить следующие характеристики программы:

- $n1$ – число уникальных операторов программы, включая символы-разделители, имена процедур и знаки операций (словарь операторов);
- $n2$ – число уникальных операндов программы, включая константы и имена переменных (словарь операндов);
- $N1$ – общее число операторов в программе;
- $N2$ – общее число операндов в программе.

Таблица 8 содержит информацию об операторах и операндах исходного кода

Таблица 8 – Информация об операторах и операндах

Операторы			Операнды		
№	Оператор	Количество	№	Операнд	Количество
1	foreach	1	1	node	2
2	var	4	2	_engine	1
3	in	1	3	Node	1
4	()	21	4	node.ItsWeaponEntity Component.Weapon	5
5	{}	2	5	VirtualMouseGoalComponent	1
6	GetNodes	1	6	Physics2D	1

Продолжение таблицы 8

Операторы			Операнды		
№	Оператор	Количество	№	Операнд	Количество
7	ToList	2	7	colliders	1
8	If	4	8	node.TransformComponent. Transform.position	1
9	Contain	1	9	node.DangerZoneComponent. Radius	1
10	<>	2	10	LayerMask	1
11	;	10	11	colliders.Length	2
12	continue	3	12	1	2
13	=	5	13	goal	2
14	GetOverlapCircleAll	1	14	goal.Count	1
15	GetMask	1	15	goal[0].gameObject.transform	1
16	==	1	16	node.TransformComponent. Transform	1
17	[]	1	17	Tendention	1
18	SetFindActive	1	18	2	1
19	GetAngleBetween	1	19	Angle	1
20	Add	4	20	angle	2
21	VirtualMouseGoal Component	1			
22	IsPrepareState Component	1			
23	IsWideAttack Component	1			
24	else	1			
25	IsShortAttack Component	1			
26	!=	1			
27	new	4			
28	GetGoals	1			
29	.	30			
30	,	3			
Всего:	n1 = 30	N1 = 111	Всего:	n2 = 20	N2 = 29

Словарь программы рассчитывается по формуле:

$$n = n1 + n2 = 30 + 20 = 50.$$

Длина программы рассчитывается по формуле:

$$N = N1 + N2 = 111 + 29 = 140.$$

Объем программы рассчитывается по формуле:

$$NV = N * \log_n 2 = 14 * 5.584 = 781.76.$$

Сложность программы рассчитывается по формуле:

$$HD = \frac{n1}{2} * \frac{N2}{n2} = \frac{30}{2} * \frac{29}{20} = 21.75$$

Полученные метрики говорят о том, что код метода CheckBossDangerZone лаконичен и прост.

Заключение

В рамках выпускной квалификационной работы было разработано игровое приложение с многопользовательским режимом игры. Для игрового приложения был реализован гейм-дизайн.

Был произведен анализ предметной области, в ходе которого было произведено сравнение игровых приложений, аналогичных разрабатываемому. Для рассмотренных аналогов были выделены их ключевые особенности и существующие недостатки. В результате анализа аналогов было принято решение разработать собственное игровое приложение с многопользовательским режимом игры. Для разрабатываемого игрового приложения в целом и гейм-дизайна в частности были определены функциональные и нефункциональные требования.

Было произведено планирование разработки и оценка бюджета проекта. Длительность реализации проекта составила 46 дней, а предполагаемые затраты – 336 400 руб. Таким образом планирование показало экономическую целесообразность разработки.

На этапе проектирования программных средств был применен язык моделирования UML. В качестве среды для построения UML диаграмм была использована программа «Draw.io»

Для реализации программных средств использовался язык программирования C#. В качестве среды разработки была использована IDE Visual Studio.

Было проведено функциональное, модульное и исследовательское тестирование разработанного приложения. На основе результатов тестирования можно сделать вывод, что разработанное приложение соответствует поставленным требованиям и работает корректно.

Таким образом, в результате выполнения выпускной квалификационной работы были разработаны программные средства, отвечающие всем требованиям технического задания.

Список использованных источников

1. 2020 в игре: как поменяется индустрия гейминга и киберспорта// РБК [Электронный ресурс] – URL: <https://quote.rbc.ru/news/article/5e183c589a794703ef1f5781> (дата обращения: 18.04.2021)
2. Медиаиндустрия в 2019 году: Видеоигры // PwC | Россия [Электронный ресурс] – URL: <https://www.pwc.ru/ru/publications/mediaindustriya-v-2019/videoigry.html> (дата обращения: 18.04.2021)
3. Выручка мировой индустрии видеоигр превзошла доходы кино и спорта // Интерфакс [Электронный ресурс] – URL: <https://www.interfax.ru/world/742705> (дата обращения: 18.04.2021)
4. Darkest Dungeon // Steam [Электронный ресурс] – URL: https://store.steampowered.com/app/262060/Darkest_Dungeon/ (дата обращения: 18.04.2021)
5. Kingdom Come Deliverance // Steam [Электронный ресурс] – URL: https://store.steampowered.com/app/379430/Kingdom_Come_Deliverance/ (дата обращения: 18.04.2021)
6. 100 лучших игр года по размеру валовой выручки // Steam [Электронный ресурс] – URL: <https://store.steampowered.com/sale/BestOf2020> (дата обращения: 18.04.2021)
7. Top games // Steam Charts [Электронный ресурс] – URL: <https://steamcharts.com/top> (дата обращения: 18.04.2021)
8. Most played games on Steam // steamdb [Электронный ресурс] – URL: <https://steamdb.info/graph/> (дата обращения: 18.04.2021)
9. Hood: Outlaws and Legends // Steam [Электронный ресурс] – URL: https://store.steampowered.com/app/927350/Hood_Outlaws__Legends/ (дата обращения: 18.04.2021)

10. Шелл, Д. Геймдизайн. Как создать игру, в которую будут играть все / Д. Шелл – Москва: Альпина Паблишер, 2020 – 640 с.
11. It takes two // Steam Charts [Электронный ресурс] – URL: <https://steamcharts.com/app/1426210> (дата обращения: 18.04.2021)
12. The way out // Steam Charts [Электронный ресурс] – URL: <https://steamcharts.com/app/1222700> (дата обращения: 18.04.2021)
13. Wizard of Legend // Steam [Электронный ресурс] – URL: https://store.steampowered.com/app/445980/Wizard_of_Legend/ (дата обращения: 18.04.2021)
14. Furi // Steam [Электронный ресурс] – URL: <https://store.steampowered.com/app/423230/Furi/> (дата обращения: 18.04.2021)
15. The Binding of Isaac // Steam [Электронный ресурс] – URL: https://store.steampowered.com/app/113200/The_Binding_of_Isaac/ (дата обращения: 18.04.2021)
16. The Binding of Isaac: Rebirth // Steam [Электронный ресурс] – URL: https://store.steampowered.com/app/250900/The_Binding_of_Isaac_Rebirth/ (дата обращения: 18.04.2021)
17. Most important gaming platforms for future growth according to game developers worldwide as of March 2019 // Statista [Электронный ресурс] – URL: <https://www.statista.com/statistics/1128100/platforms-game-developers-future-growth/> (дата обращения: 18.04.2021)
18. Share of Steam (gaming platform) users in October 2020, by operating system used // Statista [Электронный ресурс] – URL: <https://www.statista.com/statistics/265033/proportion-of-operating-systems-used-on-the-online-gaming-platform-steam> (дата обращения: 18.04.2021)
19. Данные об оборудовании и ПО пользователей: April 2021 // Steam [Электронный ресурс] – URL: <https://store.steampowered.com/hwsurvey/Steam-Hardware-Software-Survey-Welcome-to-Steam> (дата обращения: 18.04.2021)

20. Порядок исполнения // Unity: Documentation [Электронный ресурс] – URL: <https://docs.unity3d.com/Manual/ExecutionOrder.html> (дата обращения: 17.05.21)
21. Воплощенные в жизнь сюжеты // Unity [Электронный ресурс] – URL: <https://unity.com/ru/madewith> (дата обращения: 18.05.21)
22. Что такое Entity System Framework и зачем он нужен в геймдеве? // Хабр [Электронный ресурс] – URL: <https://habr.com/ru/post/197920/> (дата обращения: 18.05.21)
23. Как и почему мы написали свой ECS // Хабр [Электронный ресурс] – URL: <https://habr.com/ru/company/pixonix/blog/413729/> (дата обращения: 20.05.21)
24. AI Duet// Google: Experiments [Электронный ресурс] – URL: <https://experiments.withgoogle.com/ai/ai-duet/view/> (дата обращения: 20.05.21)
25. Mubert Stock // Mubert Stock [Электронный ресурс] – URL: <https://stock.mubert.com/> (дата обращения: 22.05.21)
26. Amper Music // Amper Music [Электронный ресурс] – URL: <https://www.ampermusic.com/> (дата обращения: 22.05.21)
27. Шампандар, А. Искусственный интеллект в компьютерных играх: как обучить виртуальные персонажи реагировать на внешние воздействия/ А. Шампандар; пер. с англ. под ред. К.А. Птицына – М: ООО «И.Д. Вильямс», 2007. – 768 с.
28. Millington, I. Artificial Intelligence for Games/ I. Millington – SF: Morgan Kaufmann Publishers, 2006. – 895 с.
29. Documentation // NUnit [Электронный ресурс] – URL: <https://docs.nunit.org/> (дата обращения: 30.05.21)
30. Unit testing // Unity: Documentation [Электронный ресурс] – URL: <https://docs.unity3d.com/Manual/testing-editor-test-runner.html> (дата обращения: 30.05.21)

31. Тестируйте игровой код с помощью Unity Test Framework // Unity [Электронный ресурс] – URL: <https://unity.com/ru/how-to/unity-test-framework-video-game-development> (дата обращения: 30.05.21)

32. Значения метрик кода // Microsoft [Электронный ресурс] – URL: <https://docs.microsoft.com/ru-ru/visualstudio/code-quality/code-metrics-values?view=vs-2019> (дата обращения: 01.06.21)

Приложение А
ГЛОССАРИЙ
(справочное)

Гейм-дизайн – процесс принятия решения о том, какой будет игра, а также процесс создания какого-либо наполнения игры, будь то программные коды или же дизайн персонажей.

Steam – онлайн центр распространения компьютерных игр.

Аркада – жанр компьютерных игр, характеризующийся коротким по времени, но интенсивным игровым процессом.

Геймплей – процесс игры.

Аркадный автомат – стойка с аппаратурой для запуска игр, обычно аркадного жанра.

Игровая механика – некоторая часть игрового процесса.

Симулятор – жанр игр, направленный на доскональное воссоздание какой-либо деятельности.

Аркадные гонки – поджанр гонок, характеризуемый специфической физикой управления автомобилем, упрощающей езду.

Стратегия – жанр видео игр, в рамках которых игрок должен использовать стратегическое и тактическое мышление для достижения каких-либо целей.

RPG – жанр видеоигр, характеризуемый тем, что под управление игроку дается единственный персонаж, который в последствии может быть развит.

Экшн – игровой жанр, характеризуемый быстрым темпом игры.

Экшн-рогалик – игра рогалик с элементами экшна.

Босс – особо сильный противник в игре.

Фаза – весь бой с боссом можно разделить на фазы, в каждой фазе босс может менять используемые атаки и, соответственно, стиль борьбы с ним.

Воксель – мельчайший элемент трёхмерного дискретного пространства, где все элементы имеют одинаковый размер.

Пиксельная графика – вид графики, характерный использованием малого количества пикселей для изображения чего-либо. Частые размеры изображений 16x16, 32x32, 64x64.

Спрайт – двумерный графический объект фиксированного размера в компьютерной графике.

Приложение Б
РАСЧЕТ КРИТИЧЕСКОГО ПУТИ ПРОЕКТА
(рекомендуемое)

Для того, чтобы определить критические работы проекта необходимо произвести расчет раннего и позднего времени начала каждой работы.

Начальное время работы – время, раньше которого работа не может быть начата.

Позднее время работы – время, позже которого работа не может быть начата без увеличения общего времени выполнения всего проекта.

Так как критические работы не имеют резерва времени, раннее и позднее время начала для таких работ будут совпадать.

Пусть t_i – время выполнения работы i , $T_p(i)$ – раннее время начала работы i , $T_n(i)$ – позднее время начала работы i , тогда формула для расчета раннего времени будет иметь следующий вид:

$$T_p(i) = \max(T_p(j) + t_j), j \in G,$$

где G – множество работ, непосредственно предшествующих работе i . Раннее время начала проекта принимается равным нулю.

Формула для расчета позднего времени имеет следующий вид:

$$T_n(i) = \min(T_n(j) - t_j), j \in H,$$

где H – множество работ, непосредственно следующих за работой i . Так как последняя работа – это веха с нулевой длительностью, раннее время ее начала совпадает с длительностью всего проекта. Эту величину принимают за позднее время начала последней работы.

Определим критические работы и критические пути. Задачи, которые необходимо выполнить в рамках проекта, представлены в таблице Б.1.

Таблица Б.1 – Перечень задач проекта

№	Название задачи	Длительность	Предшественники
1	Начало проекта	-	-
2	Анализ требований	-	-
3	Анализ аналогичных решений	3 дня	1
4	Анализ подходов к реализации игровых приложений	3 дня	3
5	Анализ подходов к реализации игровых интерфейсов	3 дня	3
6	Выбор оптимального подхода к реализации игрового приложения	1 день	4

Продолжение таблицы Б.1

№	Название задачи	Длительность	Предшественники
7	Выбор оптимального подхода к реализации игрового интерфейса	1 день	5
8	Анализ требований завершен	-	6,7
9	Выбор средств реализации	-	-
10	Анализ возможных программных средств	2 дня	1
11	Выбор оптимальных программных средств	1 день	10
12	Выбор средств завершен	-	11
13	Реализация проекта	-	-
14	Проектирование дизайна персонажей	4 дня	8, 12
15	Проектирование игрового окружения	4 дня	8, 12
16	Проектирование интерфейса	4 дня	8, 12
17	Моделирование персонажей	7 дней	14
18	Моделирование деталей игрового окружения	7 дней	15
19	Реализация игрового интерфейса	4 дня	16
20	Анимация персонажей	7 дней	17
21	Реализация алгоритмов управления персонажем	7 дней	19
22	Реализация алгоритмов боевой системы	7 дней	21
23	Реализация искусственного интеллекта противников	7 дней	21
24	Реализация завершена	-	18, 20, 22, 23
25	Отладка	-	-
26	Тестирование интерфейса приложения	3 дней	24
27	Тестирование системы управления персонажем	7 дней	24
28	Тестирование боевой системы	7 дней	27
29	Исправление ошибок	3 дня	26, 28
30	Отладка завершена	-	29
31	Конец проекта	-	30

В таблице Б.2 приведено описание ресурсов.

Таблица Б.2 – Список ресурсов проекта

Название ресурса	Тип	Стандартная ставка	Ставка сверхурочных
Программист 1	Трудовой	500 р/ч	750 р/ч
Программист 2	Трудовой	500 р/ч	750 р/ч
Дизайнер интерфейсов	Трудовой	450 р/ч	675 р/ч
Дизайнер персонажей	Трудовой	200 р/ч	300 р/ч
Дизайнер игрового окружения	Трудовой	450 р/ч	675 р/ч
3D-аниматор	Трудовой	300 р/ч	450 р/ч
Интернет	Затраты		

Назначение ресурсов на задачи представлено в таблице Б.3.

Таблица Б.3 – Назначение ресурсов

№	Название задачи	Ресурсы	Единицы
1	Начало проекта		
2	Анализ требований		

Продолжение таблицы Б.3

№	Название задачи	Ресурсы	Единицы
3	Анализ аналогичных решений	Программист 1 Интернет	100%
4	Анализ подходов к реализации игровых приложений	Программист 1 Интернет	100%
5	Анализ подходов к реализации игровых интерфейсов	Программист 2 Интернет	100%
6	Выбор оптимального подхода к реализации игрового приложения	Программист 1 Интернет Интернет	100%
7	Выбор оптимального подхода к реализации игрового интерфейса	Программист 2 Интернет	100%
8	Анализ требований завершен		
9	Выбор средств реализации		
10	Анализ возможных программных средств	Программист 1 Программист 2 Интернет	50% 50%
11	Выбор оптимальных программных средств	Программист 1 Программист 2 Интернет	50% 50%
12	Выбор средств завершен		
13	Реализация проекта		
14	Проектирование дизайна персонажей	Дизайнер персонажей Интернет	100%
15	Проектирование игрового окружения	Дизайнер игрового окружения Интернет	100%
16	Проектирование интерфейса	Дизайнер интерфейсов Интернет	100%
17	Моделирование персонажей	Дизайнер персонажей Интернет	100%
18	Моделирование деталей игрового окружения	Дизайнер игрового окружения Интернет	100%
19	Анимация персонажей	3D-аниматор Интернет	100%
20	Реализация игрового интерфейса	Программист 2 Интернет	100%
21	Реализация алгоритмов управления персонажем	Программист 1 Программист 2 Интернет	50% 50%
22	Реализация алгоритмов боевой системы	Программист 1 Программист 2 Интернет	50% 50%
23	Реализация искусственного интеллекта противников	Программист 2 Интернет	100%
24	Реализация завершена		
25	Отладка		
26	Тестирование интерфейса приложения	Программист 2 Интернет	100%
27	Тестирование системы управления персонажем	Программист 1 Программист 2 Интернет	50% 50%
28	Тестирование боевой системы	Программист 1 Программист 2 Интернет	50% 50%
29	Исправление ошибок	Программист 1 Программист 2 Интернет	50% 50%
30	Отладка завершена		

Окончание таблицы Б.3

№	Название задачи	Ресурсы	Единицы
31	Конец проекта		

Найдем критический путь проекта. Произведем вычисление, используя формулы, приведенные выше.

В таблице Б.4 представлен расчет раннего времени начала работ по проекту.

Таблица Б.4 – Расчеты раннего времени начала работ

Работа	Предшественники			Вычисление	Раннее время начала
	№	Раннее время начала	Длительность		
1	-	-	-	-	0
3	1	0	0	$0 + 0$	0
4	3	0	3	$0 + 3$	3
5	3	0	3	$0 + 3$	3
6	4	3	3	$3 + 3$	6
7	5	3	3	$3 + 3$	6
8	6,7	6 6	1 1	$\text{Max}(6 + 1, 6 + 1)$	7
10	1	0	0	$0 + 0$	0
11	10	0	2	$0 + 2$	2
12	11	2	1	$2 + 1$	3
14	8, 12	7 3	0 0	$\text{Max}(7 + 0, 3 + 0)$	7
15	8, 12	7 3	0 0	$\text{Max}(7 + 0, 3 + 0)$	7
16	8, 12	7 3	0 0	$\text{Max}(7 + 0, 3 + 0)$	7
17	14	7	4	$7 + 4$	11
18	15	7	4	$7 + 4$	11
19	16	7	4	$7 + 4$	11
20	17	11	7	$11 + 7$	18
21	19	11	4	$11 + 4$	15
22	21	15	7	$15 + 7$	22
23	21	15	7	$15 + 7$	22
24	18, 20, 22, 23	11 18 22 22	7 7 7 7	$\text{Max}(11 + 7, 18 + 7, 22 + 7, 22 + 7)$	29
26	24	29	0	$29 + 0$	29
27	24	29	0	$29 + 0$	29
28	27	31	7	$29 + 7$	36
29	26, 28	29 36	3 7	$\text{Max}(29 + 3, 36 + 7)$	43
30	29	43	3	$43 + 3$	46
31	30	46	0	$46 + 0$	46

В таблице Б.5 представлен расчет позднего времени начала работ по проекту.

Таблица Б.5 – Расчеты позднего времени начала работ

Работа	Последователи			Вычисление	Позднее время начала
	№	Позднее время начала	Длительность		
31	-	-	-	-	46
30	31	46	0	46 - 0	46
29	30	46	3	46 - 3	43
28	29	43	7	43 - 7	36
27	28	36	7	36 - 7	29
26	29	43	3	43 - 3	40
24	26, 27	29 40	0	Min(29 - 0, 40 - 0)	29
23	24	29	7	29 - 7	22
22	24	29	7	29 - 7	22
21	22, 23	22 22	7	Min(22 - 7, 22 - 7)	15
20	24	29	7	29 - 7	22
19	21	15	4	15 - 4	11
18	24	29	7	29 - 7	22
17	20	22	7	22 - 7	15
16	19	11	4	11 - 4	7
15	18	22	4	22 - 4	18
14	17	15	4	15 - 4	11
12	14, 15, 16	11 18 7	0	Min(11 - 0, 18 - 0, 7 - 0)	7
11	12	7	1	7 - 1	6
10	11	6	2	6 - 2	4
8	14, 15, 16	11 18 7	0	Min(11 - 0, 18 - 0, 7 - 0)	7
7	8	7	1	7 - 1	6
6	8	7	1	7 - 1	6
5	7	6	3	6 - 3	3
4	6	6	3	6 - 3	3
3	4, 5	3 3	3	Min(3 - 3, 3 - 3)	0
1	3, 10	4 0	0	Min(4 - 0, 0 - 0)	0

В таблице Б.6 представлен расчет резерва времени для работ.

Таблица Б.6 – Расчет резерва времени работ

№	Раннее время начала	Позднее время начала	Резерв времени
1	0	0	0
3	0	0	0
4	3	3	0
5	3	3	0

Продолжение таблицы Б.6

№	Раннее время начала	Позднее время начала	Резерв времени
6	6	6	0
7	6	6	0
8	7	7	0
10	0	4	4
11	2	6	4
12	3	7	4
14	7	11	4
15	7	18	11
16	7	7	0
17	11	15	4
18	11	22	11
19	11	11	0
20	18	22	4
21	15	15	0
22	22	22	0
23	22	22	0
24	29	29	0
26	29	40	11
27	29	29	0
28	36	36	0
29	43	43	0
30	46	46	0
31	46	46	0

На основе полученных результатов можно сделать вывод, что критический путь проходит через вершины 1, 3, 4, 5, 6, 7, 8, 16, 19, 21, 22, 23, 24, 27, 28, 29, 30, 31 а общая длительность проекта составляет 46 дней.

На рисунке Б.1 представлен сетевой график проекта с выделенным критическим путем. Критический путь в данном случае выделен с помощью пунктирных линий. В свою очередь линии, не отмеченные пунктиром, являются

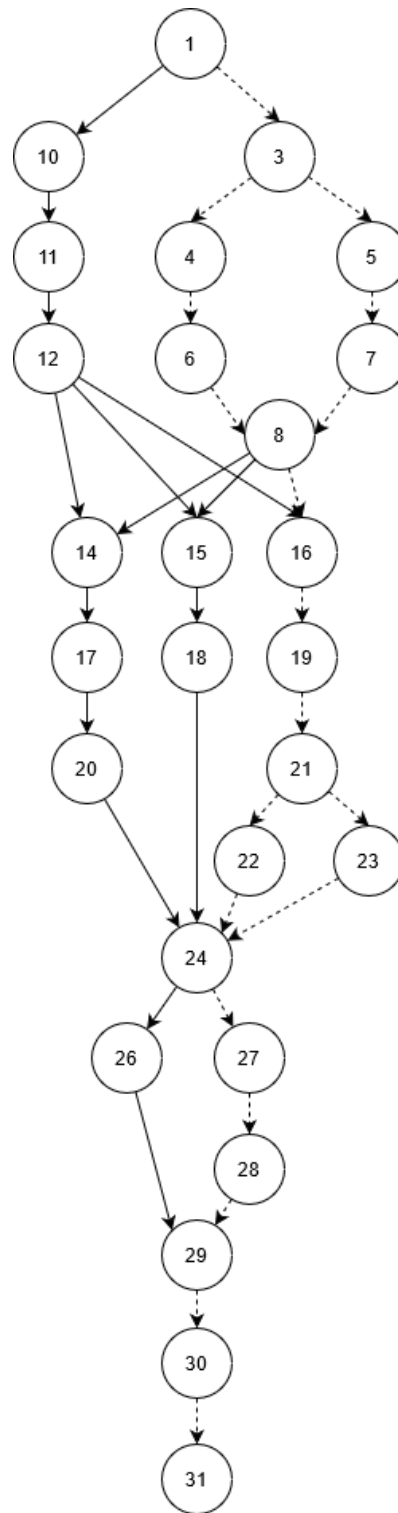


Рисунок Б.1 – Критический путь проекта

Таким образом был вычислен критический путь данного проекта.

Приложение В
ИСХОДНЫЙ КОД ПРИЛОЖЕНИЯ
(рекомендуемое)

Листинг В.1 – StartupSingle.cs

```

using Assets.Scripts.ECS.Systems.Fixed;
using Assets.Scripts.ECS.Systems.Late;
using Assets.Scripts.ECS.Systems.Unfixed;
using ECS.Core;
using UnityEngine;

using Assets.Scripts.ECS.Components;

namespace Assets.Scripts.Startup
{
    public class StartupSingle : StartupBase
    {
        public GameObject CameraPrefab;
        public GameObject PlayerPrefab;
        public GameObject SwordPrefab;
        public GameObject BossPrefab;

        private GameObject _cameraGo;
        private GameObject _playerGo;
        private GameObject _swordGo;
        private GameObject _bossGo;
        private GameObject _swordBossGO;

        private Entity _cameraEntity;
        private Entity _playerEntity;
        private Entity _swordEntity;
        private Entity _bossEntity;
        private Entity _swordBossEntity;

        protected override void Start()
        {
            base.Start();

            AddUnfixedSystems(
                new CollectKeyInputsSystem(),
                new CollectMouseInputsSystem());

            AddFixedSystems(
                new ResetDirectionSystem(),
                new GoalFindSystem(),
                new TrackGoalSystem(),
                new CheckBossDangerZoneSystem(),
                new VirtualMouseSystem(),
                new VirtualMouseToMouseConverterSystem(),
                new CalculateDirectionSystem(),
                new CalculateVelocitySystem(),
                new MoveCharacterSystem(),
                new StoreWeaponPreviousAngleSystem(),
                new MoveWeaponSystem(),
                new DamageByWeaponSystem(),
                new KillCharacterSystem(),
                new KillWeaponSystem(),
                new ResetKeysInputsSystem());

            AddLateSystem(
                new MoveCameraSystem(),
                new UpdateHealthBarSystem());

            InstantiateGameObjects();
            CreateEntities();
            RegisterEntities();
        }
    }
}

```

```

        private void InstantiateGameObjects()
        {
            _cameraGo = Instantiate(CameraPrefab, new Vector3(0, 0, -10),
Quaternion.identity);

            _playerGo = Instantiate(PlayerPrefab, new Vector3(0, 0, 0),
Quaternion.identity);
            _swordGo = Instantiate(SwordPrefab, new Vector3(0, 0, 0),
Quaternion.identity);
            _swordGo.transform.parent = _playerGo.transform;

            _bossGo = Instantiate(BossPrefab, new Vector3(0, 13, 0),
Quaternion.identity);

            _swordBossGO = Instantiate(SwordPrefab, new Vector3(0, 13, 0),
Quaternion.identity);
            _swordBossGO.transform.parent = _bossGo.transform;
        }

        private void CreateEntities()
        {
            _cameraEntity = EntityHelper.GetCameraEntity(_cameraGo, _playerGo);

            _playerEntity = EntityHelper.GetCharacterEntity(_playerGo, 100, 3)
                .KeyInputsReceiver()
                .PlayerIdentity();

            _swordEntity = EntityHelper.GetWeaponEntity(_swordGo, _playerGo,
_playerEntity, 1.2f, 1, 1)
                .MouseInputReceiver(_cameraGo);

            _bossEntity = EntityHelper.GetCharacterEntity(_bossGo, 100, 2.5f)
                .EnemyIdentity()
                .GoalIndigentIdentity(new Transform[] { _playerGo.transform })
                .Add(new DangerZoneComponent() { Radius = 2 });

            _swordBossEntity = EntityHelper.GetWeaponEntity(_swordBossGO, _bossGo,
_bossEntity, 2f, 1, 1)
                .Add(new MouseComponent() { MousePosition = new Vector3(0, 0, -9.7f) })
                .Add(new IsEnemyWeaponComponent())
                .Add(new WeaponSpeedComponent() { Speed = .2f })
                .Add(new VirtualMouseComponent() { Angle = 0 })
                .Add(new VirtualMouseRadiusComponent() { Radius = 100 })
                .Add(new VirtualMouseAttackAngleComponent() { Angle = 30 });

            _bossEntity.Add(new ItsWeaponEntityComponent() { Weapon = _swordBossEntity
});
        }

        private void RegisterEntities()
        {
            Unfixed.AddEntity(_playerEntity);
            Unfixed.AddEntity(_swordEntity);

            Fixed.AddEntity(_playerEntity);
            Fixed.AddEntity(_swordEntity);
            Fixed.AddEntity(_bossEntity);
            Fixed.AddEntity(_swordBossEntity);

            Late.AddEntity(_cameraEntity);
            Late.AddEntity(_playerEntity);
            Late.AddEntity(_bossEntity);
        }

```

```
    }
}
```

Листинг В.2 – StartupBase.cs

```
using System.Linq;
using ECS.Core;
using UnityEngine;

namespace Assets.Scripts.Startup
{
    public class StartupBase : MonoBehaviour
    {
        protected Engine Unfixed { get; private set; }
        protected Engine Fixed { get; private set; }
        protected Engine Late { get; private set; }

        protected virtual void Start()
        {
            Unfixed = new Engine();
            Fixed = new Engine();
            Late = new Engine();
        }

        protected virtual void Update()
        {
            Unfixed.Update(Time.deltaTime);
        }

        protected virtual void FixedUpdate()
        {
            Fixed.Update(Time.fixedDeltaTime);
        }

        protected virtual void LateUpdate()
        {
            Late.Update(Time.deltaTime);
        }

        protected void AddUnfixedSystems(params SystemBase[] systems) =>
        AddSystems(Unfixed, systems);

        protected void AddFixedSystems(params SystemBase[] systems) => AddSystems(Fixed,
        systems);

        protected void AddLateSystem(params SystemBase[] systems) => AddSystems(Late,
        systems);

        private void AddSystems(Engine engine, params SystemBase[] systems)
        {
            uint priority = 0;

            if (engine.GetSystems().Any())
            {
                priority = engine.GetSystems().Max(s => s.Priority);
            }

            foreach (var system in systems)
            {
                engine.AddSystem(system, priority);
                priority++;
            }
        }
    }
}
```

```

    }
}
}

```

Листинг В.3 – EntityHelper.cs

```

using Assets.Scripts.ECS.Components;
using Assets.Scripts.UI;
using Assets.Scripts.UI.Game;
using Assets.Scripts.Util;
using ECS.Core;
using UnityEngine;

namespace Assets.Scripts.Startup
{
    public static class EntityHelper
    {
        #region Player.

        public static Entity GetCharacterEntity(GameObject characterGo, int health, float
speed)
        => GetCharacterEntityBase(characterGo, health)
            .Add(new IsMoveEnableComponent())
            .Add(new DirectionComponent())
            .Add(new ColliderComponent {Collider =
characterGo.GetComponent<Collider2D>()})
            .Add(new SpeedComponent {Speed = speed});

        public static Entity GetHostCharacterEntity(GameObject characterGo, uint id, int
health, float speed)
        => GetCharacterEntity(characterGo, health, speed)
            .Add(new PositionComponent())
            .Id(id);

        public static Entity GetClientCharacterEntity(GameObject characterGo, int health,
uint id)
        => GetCharacterEntityBase(characterGo, health)
            .Add(new PositionComponent())
            .Id(id);

        private static Entity GetCharacterEntityBase(GameObject characterGo, int health)
        => new Entity()
            .Add(new VelocityComponent())
            .Add(new HealthComponent { MaxHealth = health, CurrentHealth = health })
            .Add(new TransformComponent {Transform =
characterGo.GetComponent<Transform>()})
            .Add(new RigidbodyComponent {Rigidbody =
characterGo.GetComponent<Rigidbody2D>()})
            .Add(new HealthBarComponent {HealthBar =
characterGo.GetComponentInChildren<HealthBar>()});

        #endregion

        #region Sword.

        public static Entity GetWeaponEntity(
            GameObject swordGo,
            GameObject ownerGo,
            Entity ownerEntity,
            float r,
            int damage,
            float delta)

```

```

=> GetWeaponEntityBase(swordGo)
    .Add(new IsAttackEnableComponent())
    .Add(new WeaponPreviousAngleComponent())
    .Add(new OwnerTransformComponent {Transform =
ownerGo.GetComponent<Transform>()})
    .Add(new OwnerHealthComponentComponent {HealthComponent =
ownerEntity.Get<HealthComponent>()})
    .Add(new ColliderComponent {Collider =
swordGo.GetComponent<Collider2D>()})
    .Add(new WeaponRadiusComponent { R = r })
    .Add(new WeaponEffectiveDeltaComponent {Delta = delta})
    .Add(new DamageComponent {Damage = damage});

public static Entity GetHostWeaponEntity(
    GameObject swordGo,
    GameObject ownerGo,
    Entity ownerEntity,
    uint id,
    float r,
    int damage,
    float delta)
=> GetWeaponEntity(swordGo, ownerGo, ownerEntity, r, damage, delta)
    .Add(new PositionComponent())
    .Add(new RotationComponent())
    .Id(id);

public static Entity GetClientWeaponEntity(GameObject swordGo, uint id)
=> GetWeaponEntityBase(swordGo)
    .Add(new PositionComponent())
    .Add(new RotationComponent())
    .Id(id);

private static Entity GetWeaponEntityBase(GameObject swordGo)
=> new Entity()
    .Add(new TransformComponent {Transform =
swordGo.GetComponent<Transform>()});

#endregion

#region Camera.

public static Entity GetCameraEntity(GameObject cameraGo, GameObject targetGo)
=> new Entity()
    .Add(new IsFollowComponent())
    .Add(new TransformComponent { Transform =
cameraGo.GetComponent<Transform>() })
    .Add(new OwnerTransformComponent { Transform =
targetGo.GetComponent<Transform>() });

#endregion

public static Entity KeyInputSource(this Entity entity)
=> entity.Add(new KeysComponent());

public static Entity KeyInputsReceiver(this Entity entity)
=> entity.KeyInputSource()
    .Add(new IsInputReceiverComponent());

public static Entity MouseInputSource(this Entity entity)
=> entity.Add(new MouseComponent());

public static Entity MouseInputReceiver(this Entity entity, GameObject cameraGo)
=> entity.MouseInputSource()
    .Add(new IsInputReceiverComponent())

```

```

        .Add(new CameraComponent {Camera = cameraGo.GetComponent<Camera>()}));

    public static Entity Serializable(this Entity entity)
        => entity.Add(new IsSerializableComponent());

    public static Entity Id(this Entity entity, uint id)
        => entity.Add(new IdComponent {Id = id});

    public static Entity PlayerIdentity(this Entity entity)
        => entity.Add(new PlayerFlagComponent());

    public static Entity EnemyIdentity(this Entity entity)
        => entity.Add(new EnemyFlagComponent());

    public static Entity GoalIndigentIdentity(this Entity entity, Transform[] goals)
        => entity.Add(new TEST_GoalsAvailableComponent() { Goals = goals})
            .Add(new SearchAvailableComponent());
    }
}

```

Листинг В.4 – CollectKeyInputsSystem.cs

```

using Assets.Scripts.ECS.Components;
using ECS.Core;
using UnityEngine;

namespace Assets.Scripts.ECS.Systems.Unfixed
{
    public class CollectKeyInputsSystem : SystemBase
    {
        public class Node : NodeBase
        {
            public IsInputReceiverComponent IsInputReceiverComponent { get; private set; }

            public KeysComponent KeysComponent { get; private set; }

            protected override void OnEntityChanged()
            {
                IsInputReceiverComponent = Entity.Get<IsInputReceiverComponent>();
                KeysComponent = Entity.Get<KeysComponent>();
            }
        }

        private Engine _engine;

        public override void Update(double time)
        {
            foreach (var node in _engine.GetNodes<Node>())
            {
                node.KeysComponent.W |= Input.GetKey(KeyCode.W);
                node.KeysComponent.A |= Input.GetKey(KeyCode.A);
                node.KeysComponent.S |= Input.GetKey(KeyCode.S);
                node.KeysComponent.D |= Input.GetKey(KeyCode.D);
            }
        }

        public override void Register(Engine engine) => _engine = engine;

        public override void Unregister(Engine engine) => _engine = null;
    }
}

```

Листинг В.5 – CollectMouseInputsSystem.cs

```

using Assets.Scripts.ECS.Components;
using ECS.Core;
using UnityEngine;

namespace Assets.Scripts.ECS.Systems.Unfixed
{
    class CollectMouseInputsSystem : SystemBase
    {
        public class Node : NodeBase
        {
            public IsInputReceiverComponent IsInputReceiverComponent { get; private set; }

            public MouseComponent MouseComponent { get; private set; }
            public CameraComponent CameraComponent { get; private set; }

            protected override void OnEntityChanged()
            {
                MouseComponent = Entity.Get<MouseComponent>();
                IsInputReceiverComponent = Entity.Get<IsInputReceiverComponent>();
                CameraComponent = Entity.Get<CameraComponent>();
            }
        }

        private Engine _engine;

        public override void Update(double time)
        {
            foreach (var node in _engine.GetNodes<Node>())
            {
                var screenMousePosition = Input.mousePosition;
                var camera = node.CameraComponent.Camera;
                node.MouseComponent.MousePosition = camera.ScreenToWorldPoint(new
Vector3(
                    screenMousePosition.x,
                    screenMousePosition.y,
                    node.CameraComponent.Camera.nearClipPlane));
            }
        }

        public override void Register(Engine engine) => _engine = engine;

        public override void Unregister(Engine engine) => _engine = null;
    }
}

```

Листинг В.6 – ResetDirectionSystem.cs

```

using Assets.Scripts.ECS.Components;
using ECS.Core;
using UnityEngine;

namespace Assets.Scripts.ECS.Systems.Fixed
{
    public class ResetDirectionSystem : SystemBase
    {
        public class Node : NodeBase
        {
            public DirectionComponent DirectionComponent { get; private set; }

```

```

        protected override void OnEntityChanged()
        {
            DirectionComponent = Entity.Get<DirectionComponent>();
        }
    }

    private Engine _engine;

    public override void Update(double time)
    {
        foreach (var node in _engine.GetNodes<Node>())
        {
            node.DirectionComponent.Direction = Vector2.zero;
        }
    }

    public override void Register(Engine engine) => _engine = engine;

    public override void Unregister(Engine engine) => _engine = null;
}

```

Листинг В.7 – GoalFindSystem.cs

```

using System.Linq;
using UnityEngine;
using ECS.Core;
using Assets.Scripts.ECS.Components;

namespace Assets.Scripts.ECS.Systems.Fixed
{
    public class Node : NodeBase
    {
        public SearchAvailableComponent SearchAvailableComponent { get; private set; }
        public TransformComponent TransformComponent { get; private set; }
        public TEST_GoalsAvailableComponent GoalsAvailableComponent { get; private set; }

        protected override void OnEntityChanged()
        {
            TransformComponent = Entity.Get<TransformComponent>();
            GoalsAvailableComponent = Entity.Get<TEST_GoalsAvailableComponent>();
        }
    }

    class GoalFindSystem : SystemBase
    {
        private Engine _engine;

        public override void Update(double time)
        {
            foreach (var node in _engine.GetNodes<Node>().ToList())
            {
                int i = FindGoal(node.TransformComponent.Transform,
                                node.GoalsAvailableComponent.Goals);

                node.Entity.Add(new GoalComponent() { Goal =
                                node.GoalsAvailableComponent.Goals[i]});
                node.Entity.Remove<TEST_GoalsAvailableComponent>();
                node.Entity.Remove<SearchAvailableComponent>();
            }
        }
    }
}

```

```

public override void Register(Engine engine) => _engine = engine;

public override void Unregister(Engine engine) => _engine = null;

private int FindGoal(Transform seeker, Transform[] goals)
{
    float minDist = float.PositiveInfinity;
    float dist;

    int goalIndex = -1;
    for (int i = 0; i < goals.Length; i++)
    {
        dist = Vector3.Distance(seeker.position, goals[i].position);

        if (dist < minDist)
        {
            minDist = dist;
            goalIndex = i;
        }
    }

    return goalIndex;
}
}
}

```

Листинг В.8 – TrackGoalSystem.cs

```

using System.Linq;
using UnityEngine;
using ECS.Core;
using Assets.Scripts.ECS.Components;

namespace Assets.Scripts.ECS.Systems.Fixed
{
    class TrackGoalSystem : SystemBase
    {
        Engine _engine;

        public class Node: NodeBase
        {
            public TransformComponent TransformComponent { get; private set; }
            public GoalComponent GoalComponent { get; private set; }
            public DirectionComponent DirectionComponent { get; private set; }

            protected override void OnEntityChanged()
            {
                TransformComponent = Entity.Get<TransformComponent>();
                GoalComponent = Entity.Get<GoalComponent>();
                DirectionComponent = Entity.Get<DirectionComponent>();
            }
        }

        public override void Update(double time)
        {
            foreach (var node in _engine.GetNodes<Node>().ToList())
            {
                var direction = GetGoalDirection(
                    node.TransformComponent.Transform,
                    node.GoalComponent.Goal);
            }
        }
    }
}

```

```

        node.DirectionComponent.Direction = direction;
    }
}

public override void Register(Engine engine) => _engine = engine;

public override void Unregister(Engine engine) => _engine = null;

private Vector2 GetGoalDirection(Transform seeker, Transform goal)
{
    var heading = goal.position - seeker.position;
    var distance = heading.magnitude;
    return heading / distance;
}
}
}

```

Листинг В.9 – CheckBossDangerZone.cs

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using UnityEngine;
using ECS.Core;
using Assets.Scripts.ECS.Components;

namespace Assets.Scripts.ECS.Systems.Fixed
{
    class CheckBossDangerZoneSystem : SystemBase
    {
        public class Node : NodeBase
        {
            public TransformComponent TransformComponent { get; private set; }
            public DangerZoneComponent DangerZoneComponent { get; private set; }
            public ItsWeaponEntityComponent ItsWeaponEntityComponent { get; private set; }
        }

        protected override void OnEntityChanged()
        {
            TransformComponent = Entity.Get<TransformComponent>();
            DangerZoneComponent = Entity.Get<DangerZoneComponent>();
            ItsWeaponEntityComponent = Entity.Get<ItsWeaponEntityComponent>();
        }

        private Engine _engine;

        public override void Update(double time)
        {
            foreach (var node in _engine.GetNodes<Node>().ToList())
            {
                if
                (node.ItsWeaponEntityComponent.Weapon.Contain<VirtualMouseGoalComponent>()) continue;

                var colliders =
                Physics2D.OverlapCircleAll(node.TransformComponent.Transform.position,
                    node.DangerZoneComponent.Radius, LayerMask.GetMask("Characters"));
            }
        }
    }
}

```

```

        if (colliders.Length == 1) continue;

        if (colliders.Length == 2)
        {
            var player = colliders.First(e1 => e1.gameObject !=
node.TransformComponent.Transform.gameObject);
            var angle = Mathf.Atan2(player.gameObject.transform.position.y -
node.TransformComponent.Transform.position.y,
            player.gameObject.transform.position.x -
node.TransformComponent.Transform.position.x) * 180 / Mathf.PI;
            if (angle < 0) angle += 360;
            node.ItsWeaponEntityComponent.Weapon.Add(new
VirtualMouseGoalComponent() { Tendency = 1, Angle = angle });
        }
        else
        {
            node.ItsWeaponEntityComponent.Weapon.Add(new
VirtualMouseGoalComponent() { Tendency = 1 });
        }
    }
}

public override void Register(Engine engine) => _engine = engine;

public override void Unregister(Engine engine) => _engine = null;
}
}

```

Листинг B.10 – VirtualMouseSystem.cs

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using ECS.Core;
using Assets.Scripts.ECS.Components;
using UnityEngine;

namespace Assets.Scripts.ECS.Systems.Fixed
{
    class VirtualMouseSystem : SystemBase
    {
        public class Node : NodeBase
        {
            public VirtualMouseComponent VirtualMouseComponent { get; private set; }
            public VirtualMouseGoalComponent VirtualMouseGoalComponent { get; private
set; }
            public WeaponSpeedComponent WeaponSpeedComponent { get; private set; }
            public IsEnemyWeaponComponent IsEnemyWeaponComponent { get; private set; }
            public VirtualMouseAttackAngleComponent VirtualMouseAttackAngleComponent {
get; private set; }

            protected override void OnEntityChanged()
            {
                VirtualMouseComponent = Entity.Get<VirtualMouseComponent>();
                VirtualMouseGoalComponent = Entity.Get<VirtualMouseGoalComponent>();
                WeaponSpeedComponent = Entity.Get<WeaponSpeedComponent>();
                IsEnemyWeaponComponent = Entity.Get<IsEnemyWeaponComponent>();
            }
        }
    }
}

```

```

        VirtualMouseAttackAngleComponent =
Entity.Get<VirtualMouseAttackAngleComponent>();
    }
}

Engine _engine;

public override void Update(double time)
{
    foreach (var node in _engine.GetNodes<Node>().ToList())
    {
        var difference = node.VirtualMouseAttackAngleComponent.Angle / 2;
        var angle1 = GetNextAngle(node.VirtualMouseGoalComponent.Angle, -
difference);
        var angle2 = GetNextAngle(node.VirtualMouseGoalComponent.Angle,
difference);

        var currentAngle = node.VirtualMouseComponent.Angle;

        if(node.VirtualMouseGoalComponent.CurrentStage == 0)
        {
            node.VirtualMouseComponent.Angle = angle1;
            node.VirtualMouseGoalComponent.CurrentStage = 1;
        }

        if (currentAngle > angle2)
        {
            node.Entity.Remove<VirtualMouseGoalComponent>();
            continue;
        }

        node.VirtualMouseComponent.Angle =
GetNextAngle(node.VirtualMouseComponent.Angle,
            node.VirtualMouseGoalComponent.Tendention *
node.WeaponSpeedComponent.Speed);
    }
}

public override void Register(Engine engine) {
    _engine = engine;
}

public override void Unregister(Engine engine) => _engine = null;

private float GetNextAngle(float current, float addition)
{
    var nextAngle = current + addition;

    if (nextAngle > 360) nextAngle -= 360;
    else if (nextAngle < 0) nextAngle = 360 - nextAngle;

    return nextAngle;
}

private bool IsInside(float current, float angle1, float angle2)
{
    if(IsOnRange(angle1, angle2))
    {
        return current < angle1 || current > angle2;
    }

    return current > angle1 || current < angle2;
}

```

```

        private bool IsOnRange(float angle1, float angle2)
        {
            return angle1 > angle2;
        }
    }
}

```

Листинг В.11 – VirtualMouseToMouseConverterSystem.cs

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using UnityEngine;
using ECS.Core;

using Assets.Scripts.ECS.Components;

namespace Assets.Scripts.ECS.Systems.Fixed
{
    class VirtualMouseToMouseConverterSystem : SystemBase
    {
        public class Node : NodeBase
        {
            public VirtualMouseComponent VirtualMouseComponent { get; private set; }
            public VirtualMouseRadiusComponent VirtualMouseRadiusComponent { get; private
set; }

            public MouseComponent MouseComponent { get; private set; }
            public IsEnemyWeaponComponent IsEnemyWeaponComponent { get; private set; }

            protected override void OnEntityChanged()
            {
                VirtualMouseComponent = Entity.Get<VirtualMouseComponent>();
                VirtualMouseRadiusComponent = Entity.Get<VirtualMouseRadiusComponent>();
                MouseComponent = Entity.Get<MouseComponent>();
                IsEnemyWeaponComponent = Entity.Get<IsEnemyWeaponComponent>();
            }
        }

        Engine _engine;

        public override void Update(double time)
        {
            foreach (var node in _engine.GetNodes<Node>().ToList())
            {
                var radius = node.VirtualMouseRadiusComponent.Radius;
                var mouseCoords = new Vector2(radius *
(float)Math.Cos(node.VirtualMouseComponent.Angle),
                radius * (float)Math.Sin(node.VirtualMouseComponent.Angle));

                node.MouseComponent.MousePosition = mouseCoords;
            }
        }

        public override void Register(Engine engine) => _engine = engine;

        public override void Unregister(Engine engine) => _engine = null;
    }
}

```

Листинг B.12 – CalculateDirectionSystem.cs

```

using Assets.Scripts.ECS.Components;
using ECS.Core;
using UnityEngine;

namespace Assets.Scripts.ECS.Systems.Fixed
{
    public class CalculateDirectionSystem : SystemBase
    {
        public class Node : NodeBase
        {
            public DirectionComponent DirectionComponent { get; set; }
            public KeysComponent KeysComponent { get; set; }
            public IsMoveEnableComponent IsMoveEnableComponent { get; set; }

            protected override void OnEntityChanged()
            {
                DirectionComponent = Entity.Get<DirectionComponent>();
                KeysComponent = Entity.Get<KeysComponent>();
                IsMoveEnableComponent = Entity.Get<IsMoveEnableComponent>();
            }
        }

        private Engine _engine;

        public override void Update(double time)
        {
            foreach (var node in _engine.GetNodes<Node>())
            {
                node.DirectionComponent.Direction = new Vector2(
                    -BoolToInt(node.KeysComponent.A) + BoolToInt(node.KeysComponent.D),
                    -BoolToInt(node.KeysComponent.S) + BoolToInt(node.KeysComponent.W));
            }
        }

        public override void Register(Engine engine) => _engine = engine;

        public override void Unregister(Engine engine) => _engine = null;

        private static int BoolToInt(bool b) => b ? 1 : 0;
    }
}

```

Листинг B.13 – CalculateVelocitySystem.cs

```

using Assets.Scripts.ECS.Components;
using ECS.Core;

using System.Linq;
using UnityEngine;

namespace Assets.Scripts.ECS.Systems.Fixed
{
    public class CalculateVelocitySystem : SystemBase
    {
        public class Node : NodeBase
        {
            public DirectionComponent DirectionComponent { get; private set; }
            public SpeedComponent SpeedComponent { get; private set; }
            public VelocityComponent VelocityComponent { get; private set; }
        }
    }
}

```

```

        protected override void OnEntityChanged()
        {
            DirectionComponent = Entity.Get<DirectionComponent>();
            SpeedComponent = Entity.Get<SpeedComponent>();
            VelocityComponent = Entity.Get<VelocityComponent>();
        }
    }

    private Engine _engine;

    public override void Update(double time)
    {
        foreach (var node in _engine.GetNodes<Node>())
        {
            node.VelocityComponent.Velocity = node.DirectionComponent.Direction *
node.SpeedComponent.Speed;
        }
    }

    public override void Register(Engine engine) => _engine = engine;

    public override void Unregister(Engine engine) => _engine = null;
}

```

Листинг В.14 – MoveCharacterSystem

```

using Assets.Scripts.ECS.Components;
using ECS.Core;

using UnityEngine;

namespace Assets.Scripts.ECS.Systems.Fixed
{
    public class MoveCharacterSystem : SystemBase
    {
        public class Node : NodeBase
        {
            public VelocityComponent VelocityComponent { get; private set; }
            public RigidbodyComponent RigidbodyComponent { get; private set; }

            protected override void OnEntityChanged()
            {
                VelocityComponent = Entity.Get<VelocityComponent>();
                RigidbodyComponent = Entity.Get<RigidbodyComponent>();
            }
        }

        private Engine _engine;

        public override void Update(double time)
        {
            foreach (var node in _engine.GetNodes<Node>())
            {
                node.RigidbodyComponent.Rigidbody.velocity =
node.VelocityComponent.Velocity;
            }
        }

        public override void Register(Engine engine) => _engine = engine;
    }
}

```

```

        public override void Unregister(Engine engine) => _engine = null;
    }
}

```

Листинг В.15 – StoreWeaponPreviousAngleSystem

```

using Assets.Scripts.ECS.Components;
using ECS.Core;

namespace Assets.Scripts.ECS.Systems.Fixed
{
    public class StoreWeaponPreviousAngleSystem : SystemBase
    {
        public class Node : NodeBase
        {
            public WeaponPreviousAngleComponent WeaponPreviousAngleComponent { get;
private set; }
            public TransformComponent TransformComponent { get; private set; }

            protected override void OnEntityChanged()
            {
                WeaponPreviousAngleComponent =
Entity.Get<WeaponPreviousAngleComponent>();
                TransformComponent = Entity.Get<TransformComponent>();
            }
        }

        private Engine _engine;

        public override void Update(double time)
        {
            foreach (var node in _engine.GetNodes<Node>())
            {
                node.WeaponPreviousAngleComponent.PreviousAngle =
node.TransformComponent.Transform.rotation.eulerAngles.z;
            }
        }

        public override void Register(Engine engine)
        {
            _engine = engine;
        }

        public override void Unregister(Engine engine)
        {
            _engine = null;
        }
    }
}

```

Листинг В.16 – MoveWeaponSystem

```

using Assets.Scripts.ECS.Components;
using ECS.Core;
using UnityEngine;

using System.Linq;

namespace Assets.Scripts.ECS.Systems.Fixed
{

```

```

public class MoveWeaponSystem : SystemBase
{
    public class Node : NodeBase
    {
        public MouseComponent MouseComponent { get; private set; }
        public TransformComponent TransformComponent { get; private set; }
        public OwnerTransformComponent OwnerTransformComponent { get; private set; }
        public WeaponRadiusComponent WeaponRadiusComponent { get; private set; }
        public IsAttackEnableComponent IsAttackEnableComponent { get; private set; }

        protected override void OnEntityChanged()
        {
            MouseComponent = Entity.Get<MouseComponent>();
            TransformComponent = Entity.Get<TransformComponent>();
            OwnerTransformComponent = Entity.Get<OwnerTransformComponent>();
            WeaponRadiusComponent = Entity.Get<WeaponRadiusComponent>();
            IsAttackEnableComponent = Entity.Get<IsAttackEnableComponent>();
        }
    }

    private Engine _engine;

    public override void Update(double time)
    {
        foreach (var node in _engine.GetNodes<Node>())
        {
            var relativeToOwner = node.MouseComponent.MousePosition -
node.OwnerTransformComponent.Transform.position;
            var direction = new Vector2(relativeToOwner.x,
relativeToOwner.y).normalized;
            var r = node.WeaponRadiusComponent.R;
            node.TransformComponent.Transform.localPosition = new
Vector3(direction.x, direction.y, 0) * r;
            node.TransformComponent.Transform.rotation = Quaternion.Euler(new
Vector3(0, 0, GetRotationAngle(relativeToOwner)));
        }
    }

    public override void Register(Engine engine) => _engine = engine;

    public override void Unregister(Engine engine) => _engine = null;

    private float GetRotationAngle(Vector2 v)
    {
        var angle = Mathf.Atan2(v.y, v.x) * Mathf.Rad2Deg;
        if (angle < 0) angle += 360;
        return angle;
    }
}
}

```

Листинг В.17 – DamageByWeaponSystem

```

using Assets.Scripts.ECS.Components;
using Assets.Scripts.ECS.Nodes;
using ECS.Core;
using UnityEngine;

namespace Assets.Scripts.ECS.Systems.Fixed
{
    public class DamageByWeaponSystem : SystemBase
    {

```

```

public class DamageNode : NodeBase
{
    public ColliderComponent ColliderComponent { get; private set; }
    public DamageComponent DamageComponent { get; private set; }
    public WeaponPreviousAngleComponent PreviousWeaponAngleComponent { get;
private set; }
    public WeaponEffectiveDeltaComponent WeaponEffectiveDeltaComponent { get;
private set; }
    public TransformComponent TransformComponent { get; private set; }

    protected override void OnEntityChanged()
    {
        ColliderComponent = Entity.Get<ColliderComponent>();
        DamageComponent = Entity.Get<DamageComponent>();
        PreviousWeaponAngleComponent =
Entity.Get<WeaponPreviousAngleComponent>();
        WeaponEffectiveDeltaComponent =
Entity.Get<WeaponEffectiveDeltaComponent>();
        TransformComponent = Entity.Get<TransformComponent>();
    }
}

private Engine _engine;

public override void Update(double time)
{
    foreach (var weaponNode in _engine.GetNodes<DamageNode>())
    {
        if (IsAttack(weaponNode))
        {
            foreach (var damagableNode in _engine.GetNodes<DamagableNode>())
            {
                if (IsIntersects(weaponNode.ColliderComponent,
damagableNode.ColliderComponent))
                {
                    damagableNode.HealthComponent.CurrentHealth -=
weaponNode.DamageComponent.Damage;
                    damagableNode.HealthComponent.CurrentHealth =
Mathf.Max(damagableNode.HealthComponent.CurrentHealth, 0);
                }
            }
        }
    }
}

public override void Register(Engine engine)
{
    _engine = engine;
}

public override void Unregister(Engine engine)
{
    _engine = null;
}

private bool IsAttack(DamageNode weaponNode)
{
    var delta = Mathf.Abs(weaponNode.PreviousWeaponAngleComponent.PreviousAngle -
        weaponNode.TransformComponent.Transform.rotation.eulerAngles.z);
    return delta >= weaponNode.WeaponEffectiveDeltaComponent.Delta;
}

private bool IsIntersects(ColliderComponent a, ColliderComponent b)
    => Physics2D.IsTouching(a.Collider, b.Collider);

```

```
    }
}
```

Листинг В.18 – KillCharacterSystem

```
using System.Linq;
using Assets.Scripts.ECS.Components;
using ECS.Core;
using Debug = UnityEngine.Debug;

namespace Assets.Scripts.ECS.Systems.Fixed
{
    public class KillCharacterSystem : SystemBase
    {
        public class Node : NodeBase
        {
            public HealthComponent HealthComponent { get; private set; }
            public IsMoveEnableComponent IsMoveEnableComponent { get; private set; }

            protected override void OnEntityChanged()
            {
                HealthComponent = Entity.Get<HealthComponent>();
                IsMoveEnableComponent = Entity.Get<IsMoveEnableComponent>();
            }
        }

        private Engine _engine;

        public override void Update(double time)
        {
            foreach (var node in _engine.GetNodes<Node>().ToList())
            {
                if (node.HealthComponent.CurrentHealth <= 0)
                {
                    node.Entity.Remove<IsMoveEnableComponent>();
                    Debug.Log("kill");
                }
            }
        }

        public override void Register(Engine engine) => _engine = engine;

        public override void Unregister(Engine engine) => _engine = null;
    }
}
```

Листинг В.19 – KillWeaponSystem

```
using System.Linq;
using Assets.Scripts.ECS.Components;
using ECS.Core;

namespace Assets.Scripts.ECS.Systems.Fixed
{
    public class KillWeaponSystem : SystemBase
    {
        public class Node : NodeBase
        {
            public OwnerHealthComponentComponent OwnerHealthComponentComponent { get;
private set; }
            public IsAttackEnableComponent IsAttackEnableComponent { get; private set; }
        }
    }
}
```

```

        protected override void OnEntityChanged()
        {
            OwnerHealthComponentComponent =
Entity.Get<OwnerHealthComponentComponent>();
            IsAttackEnableComponent = Entity.Get<IsAttackEnableComponent>();
        }
    }

    private Engine _engine;

    public override void Update(double time)
    {
        foreach (var node in _engine.GetNodes<Node>().ToList())
        {
            if (node.OwnerHealthComponentComponent.HealthComponent.CurrentHealth <=
0)
            {
                {
                    node.Entity.Remove<IsAttackEnableComponent>();
                }
            }
        }

        public override void Register(Engine engine) => _engine = engine;

        public override void Unregister(Engine engine) => _engine = null;
    }
}

```

Листинг B.20 – ResetKeysInputsSystem

```

using Assets.Scripts.ECS.Components;
using ECS.Core;

namespace Assets.Scripts.ECS.Systems.Fixed
{
    public class ResetKeysInputsSystem : SystemBase
    {
        public class Node : NodeBase
        {
            public KeysComponent KeysComponent { get; private set; }

            protected override void OnEntityChanged()
            {
                KeysComponent = Entity.Get<KeysComponent>();
            }
        }

        private Engine _engine;

        public override void Update(double time)
        {
            foreach (var node in _engine.GetNodes<Node>())
            {
                {
                    node.KeysComponent.W = false;
                    node.KeysComponent.A = false;
                    node.KeysComponent.S = false;
                    node.KeysComponent.D = false;
                }
            }

            public override void Register(Engine engine) => _engine = engine;
        }
    }
}

```

```

        public override void Unregister(Engine engine) => _engine = null;
    }
}

```

Листинг В.21 – MoveCameraSystem

```

using Assets.Scripts.ECS.Components;
using Assets.Scripts.ECS.Nodes;
using ECS.Core;
using UnityEngine;

namespace Assets.Scripts.ECS.Systems.Late
{
    public class MoveCameraSystem : SystemBase
    {
        public class Node : NodeBase
        {
            public IsFollowComponent IsFollowComponent { get; private set; }
            public TransformComponent TransformComponent { get; private set; }
            public OwnerTransformComponent OwnerTransformComponent { get; private set; }

            protected override void OnEntityChanged()
            {
                IsFollowComponent = Entity.Get<IsFollowComponent>();
                TransformComponent = Entity.Get<TransformComponent>();
                OwnerTransformComponent = Entity.Get<OwnerTransformComponent>();
            }
        }

        private Engine _engine;

        public override void Update(double time)
        {
            foreach (var node in _engine.GetNodes<Node>())
            {
                node.TransformComponent.Transform.position = new Vector3(
                    node.OwnerTransformComponent.Transform.position.x,
                    node.OwnerTransformComponent.Transform.position.y,
                    node.TransformComponent.Transform.position.z);
            }
        }

        public override void Register(Engine engine) => _engine = engine;

        public override void Unregister(Engine engine) => _engine = null;
    }
}

```

Листинг В.22 – UpdateHealthBarSystem

```

using Assets.Scripts.ECS.Components;
using ECS.Core;

namespace Assets.Scripts.ECS.Systems.Late
{
    public class UpdateHealthBarSystem : SystemBase
    {
        public class Node : NodeBase
        {
            public HealthComponent HealthComponent { get; private set; }

```

```

        public HealthBarComponent HealthBarComponent { get; private set; }

        protected override void OnEntityChanged()
        {
            HealthComponent = Entity.Get<HealthComponent>();
            HealthBarComponent = Entity.Get<HealthBarComponent>();
        }
    }

    private Engine _engine;

    public override void Update(double time)
    {
        foreach (var node in _engine.GetNodes<Node>())
        {
            node.HealthBarComponent.HealthBar.MaxHealth =
node.HealthComponent.MaxHealth;
            node.HealthBarComponent.HealthBar.CurrentHealth =
node.HealthComponent.CurrentHealth;
        }
    }

    public override void Register(Engine engine)
    {
        _engine = engine;
    }

    public override void Unregister(Engine engine)
    {
        _engine = null;
    }
}

```

Приложение Г
РЕЗУЛЬТАТЫ ФУНКЦИОНАЛЬНОГО ТЕСТИРОВАНИЯ
(рекомендуемое)

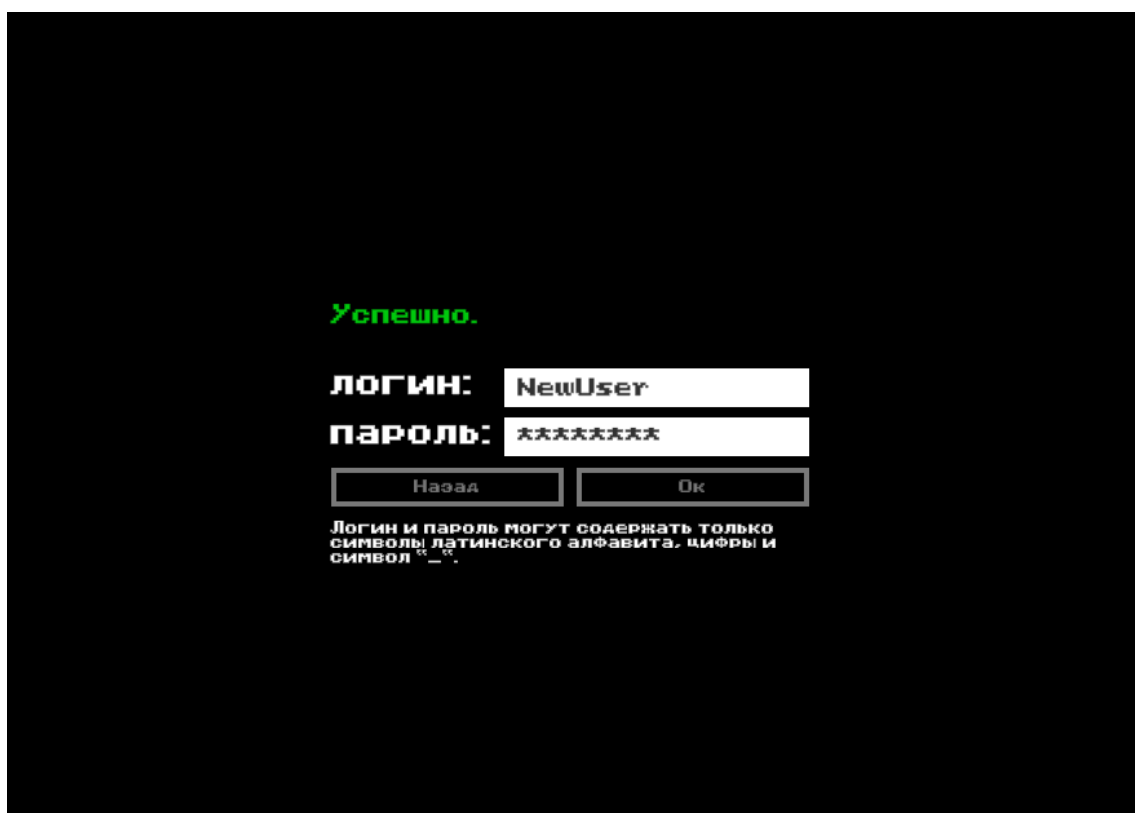


Рисунок Г.1 – Регистрация пользователя с корректным, не занятым логином и корректным паролем



Рисунок Г.2 – Регистрация пользователя с некорректным логином

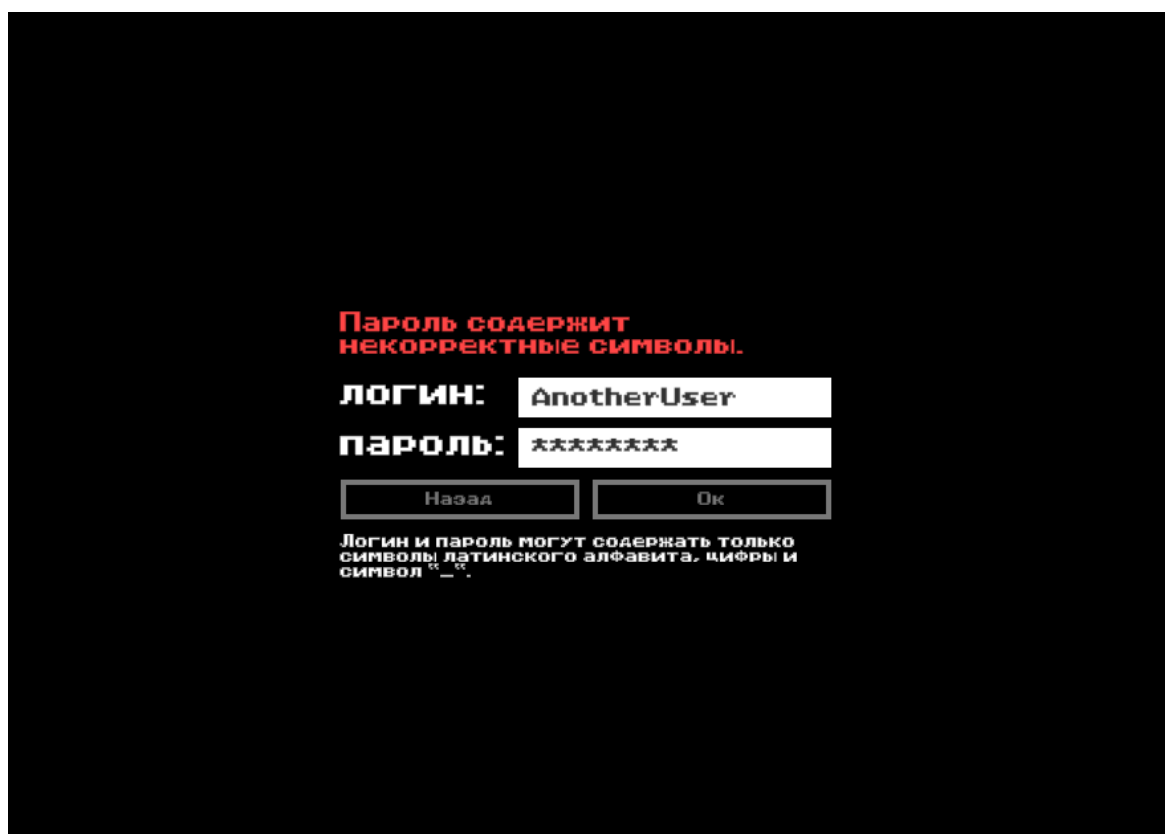


Рисунок Г.3 – Регистрация пользователя с некорректным паролем

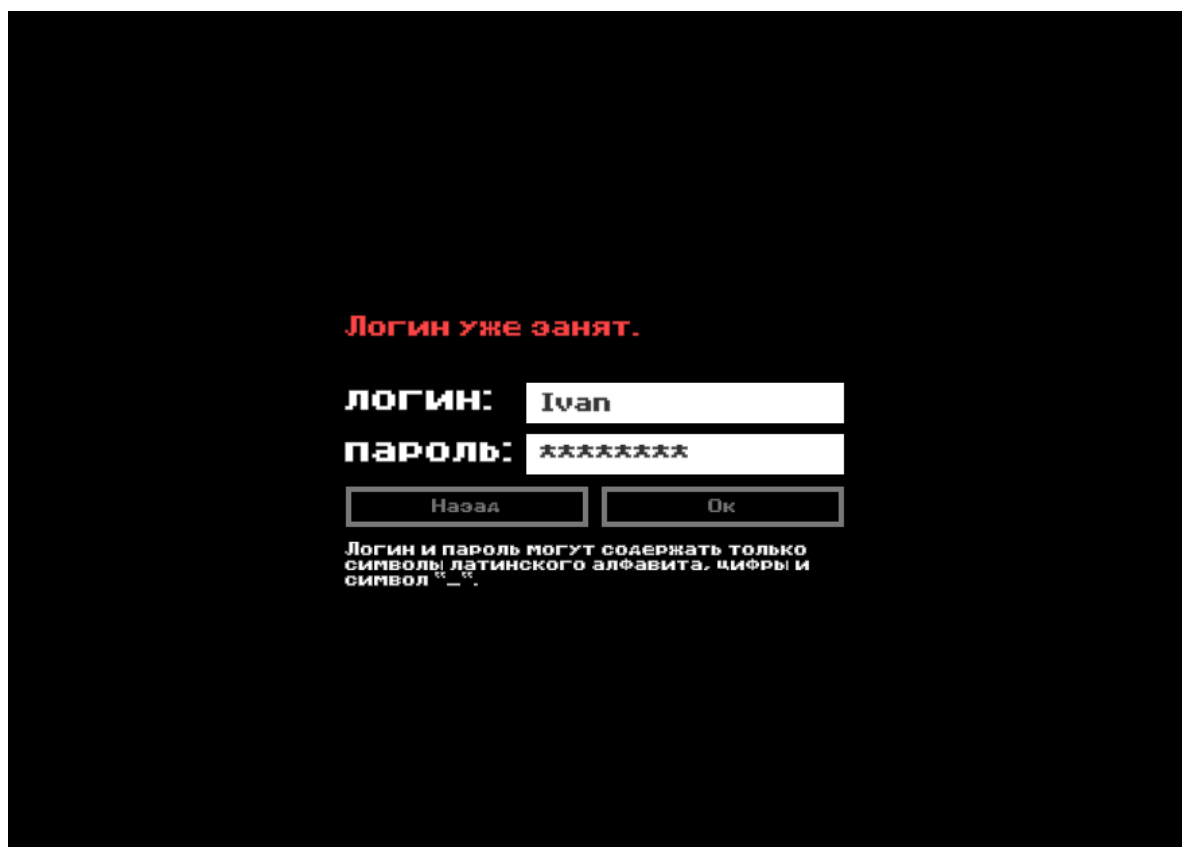
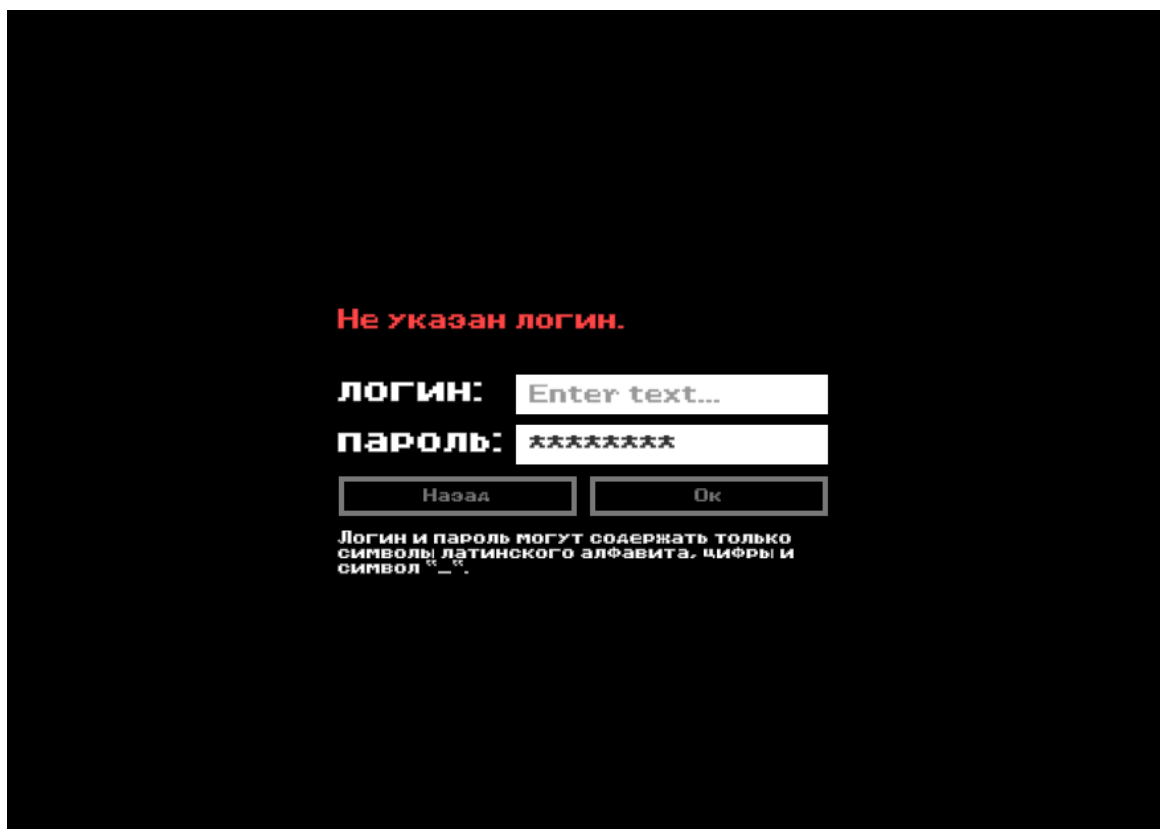


Рисунок Г.4 – Регистрация пользователя с занятым логином



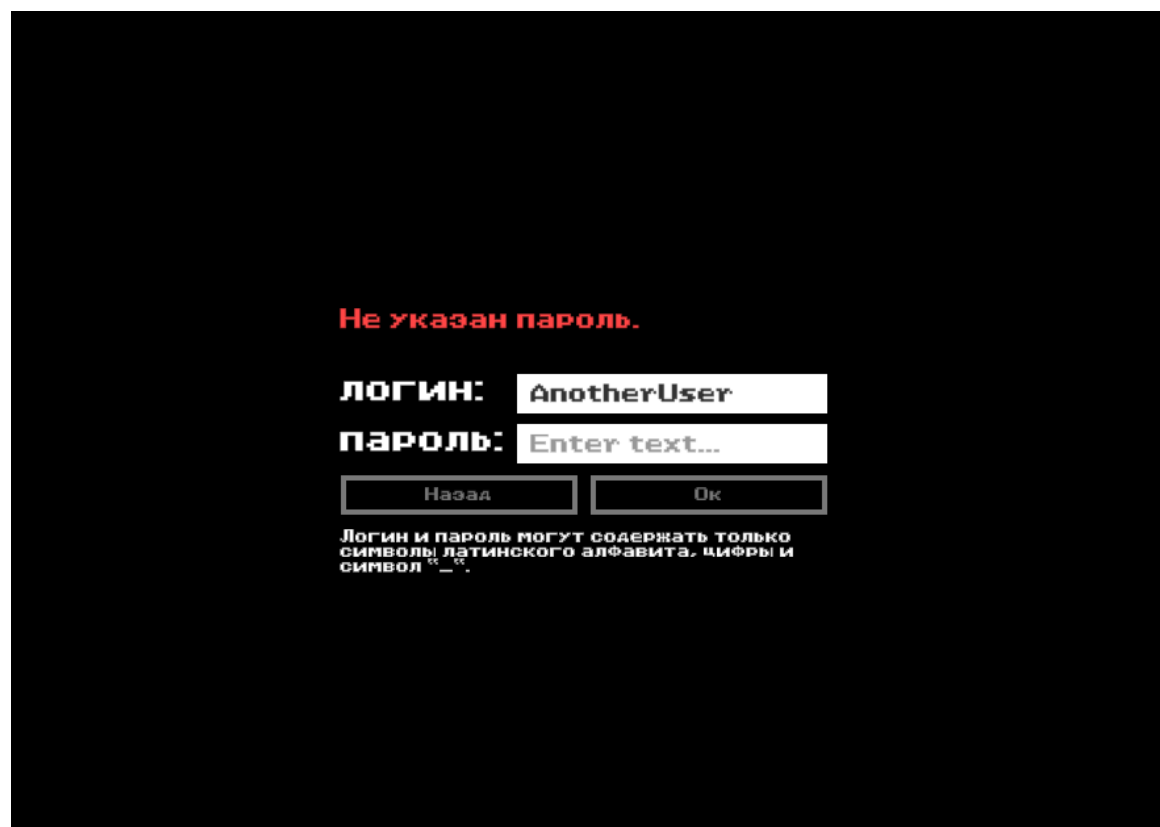
Не указан логин.

ЛОГИН:

ПАРОЛЬ:

Логин и пароль могут содержать только символы латинского алфавита, цифры и символ `_`.

Рисунок Г.5 – Регистрация пользователя без логина



Не указан пароль.

ЛОГИН:

ПАРОЛЬ:

Логин и пароль могут содержать только символы латинского алфавита, цифры и символ `_`.

Рисунок Г.6 – Регистрация пользователя без пароля

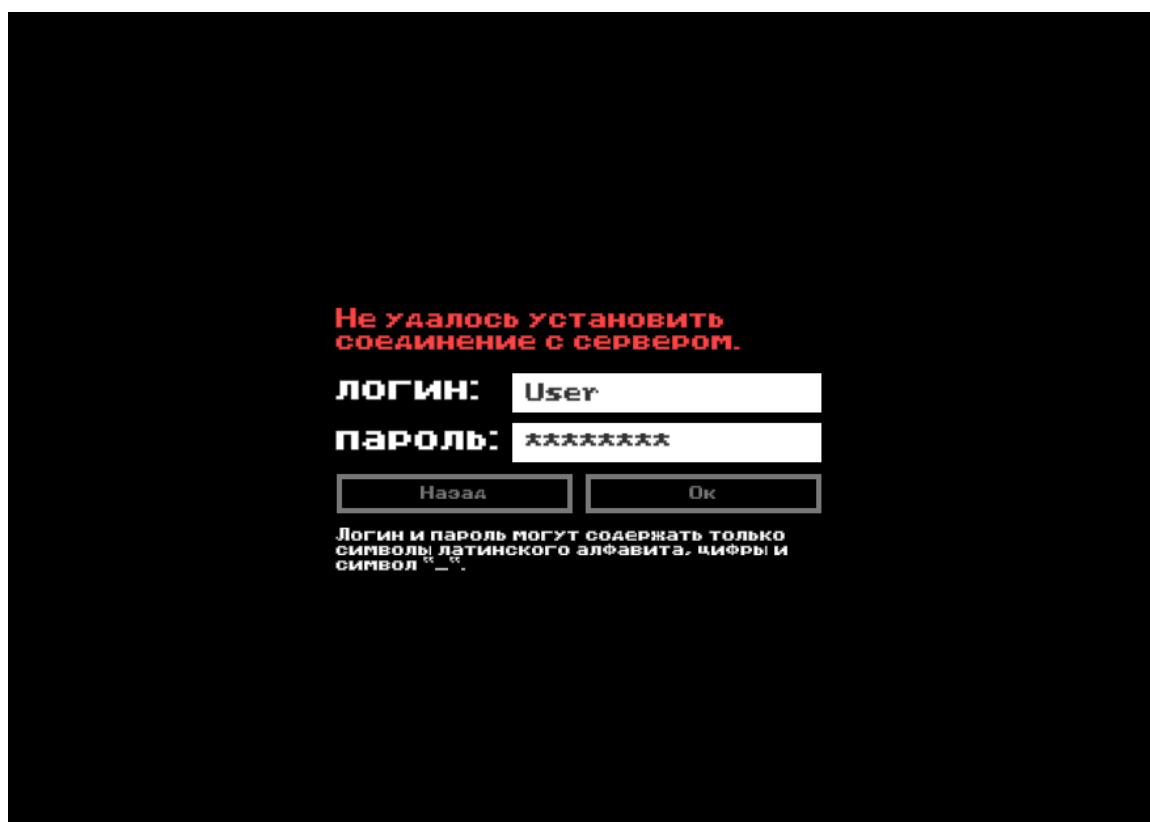


Рисунок Г.7 – Попытка регистрации при незапущенном сервере



Рисунок Г.8 – Вход в систему с использованием логина и пароля
существующего пользователя



Рисунок Г.9 – Вход в систему под несуществующим логином

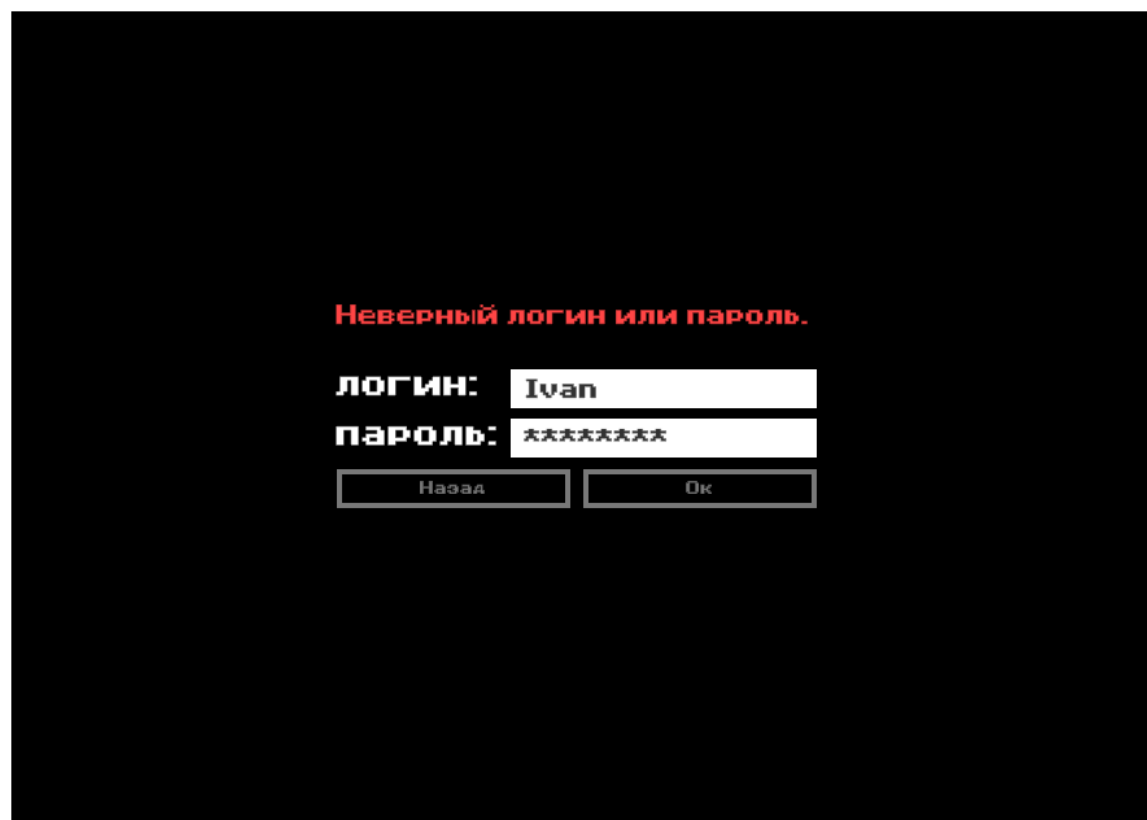


Рисунок Г.10 – Вход в систему с неверным паролем

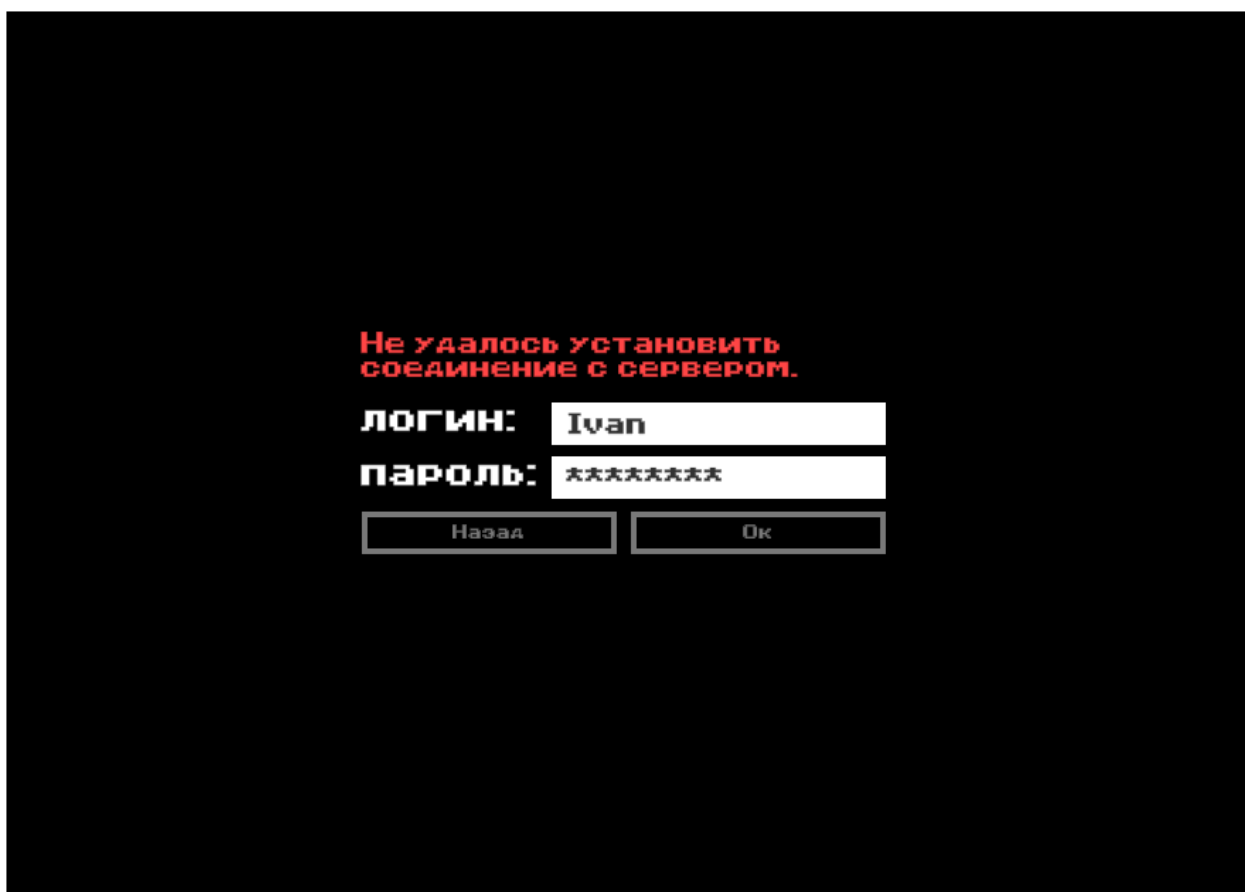


Рисунок Г.11 – Вход в систему с незапущенным сервером



Рисунок Г.12 – Два пользователя в одном матче

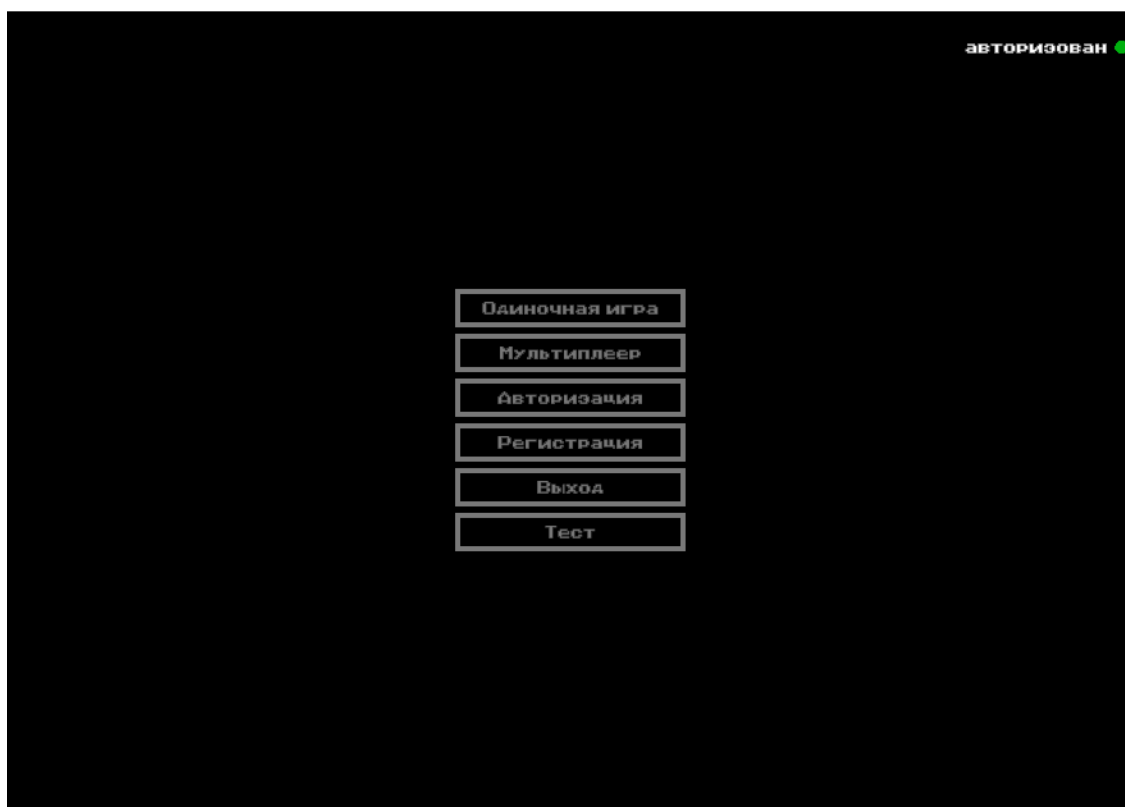


Рисунок Г.13 – Возврат в главное меню после остановки поиска матча

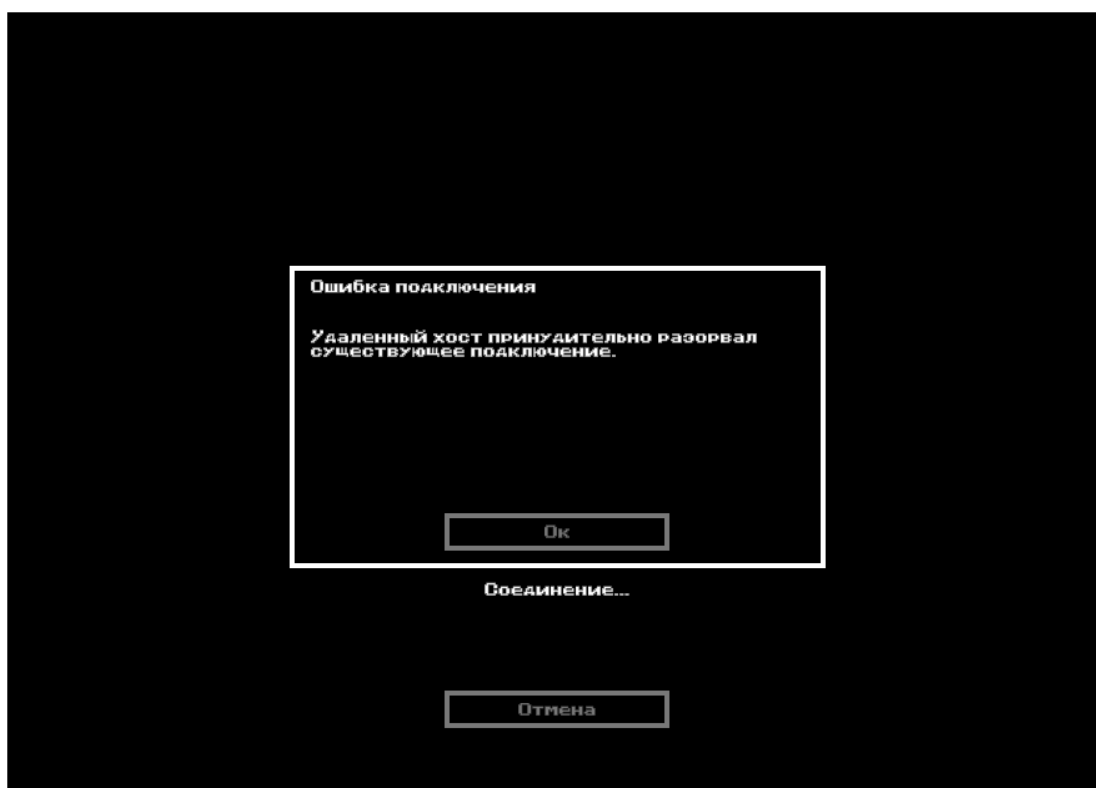


Рисунок Г.14 – Один из пользователей прервал соединение



Рисунок Г.15 – Попытка дважды инициировать поиск матча одним и тем же пользователем

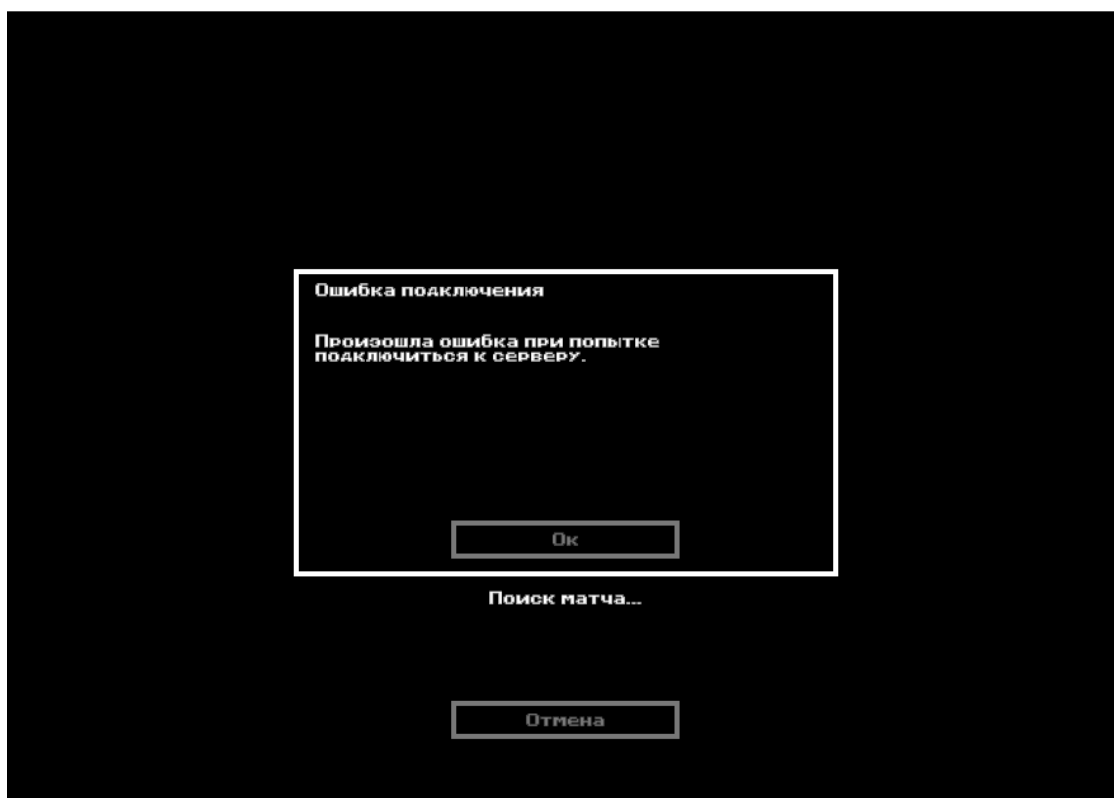


Рисунок Г.16 – Попытка поиска матча с незапущенным сервер

Приложение Д

РЕЗУЛЬТАТЫ ИССЛЕДОВАТЕЛЬСКОГО ТЕСТИРОВАНИЯ

(рекомендуемое)

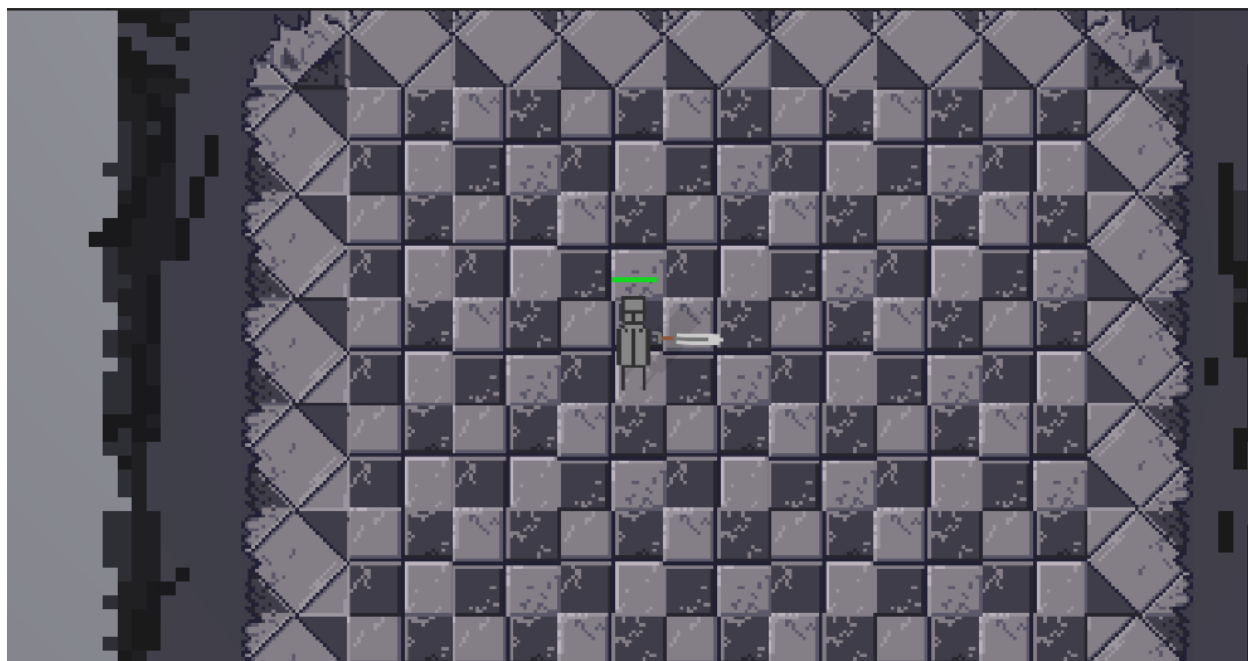


Рисунок Д.1 – Персонаж до перемещения

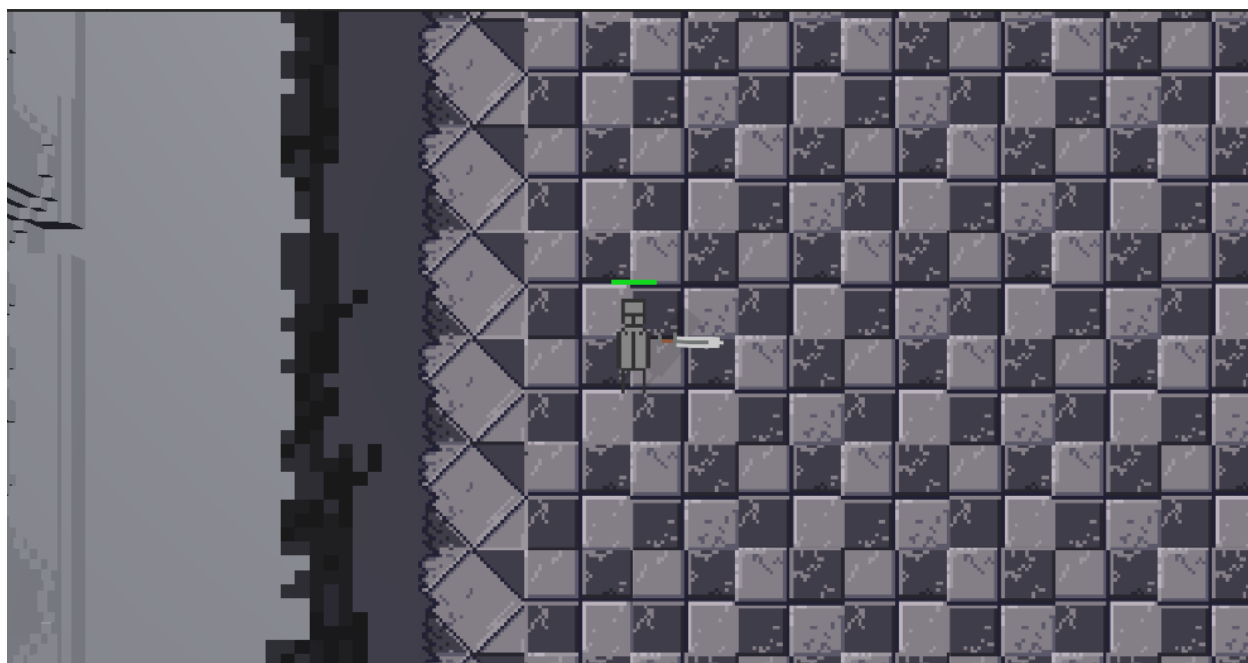


Рисунок Д.2 Персонаж после перемещения

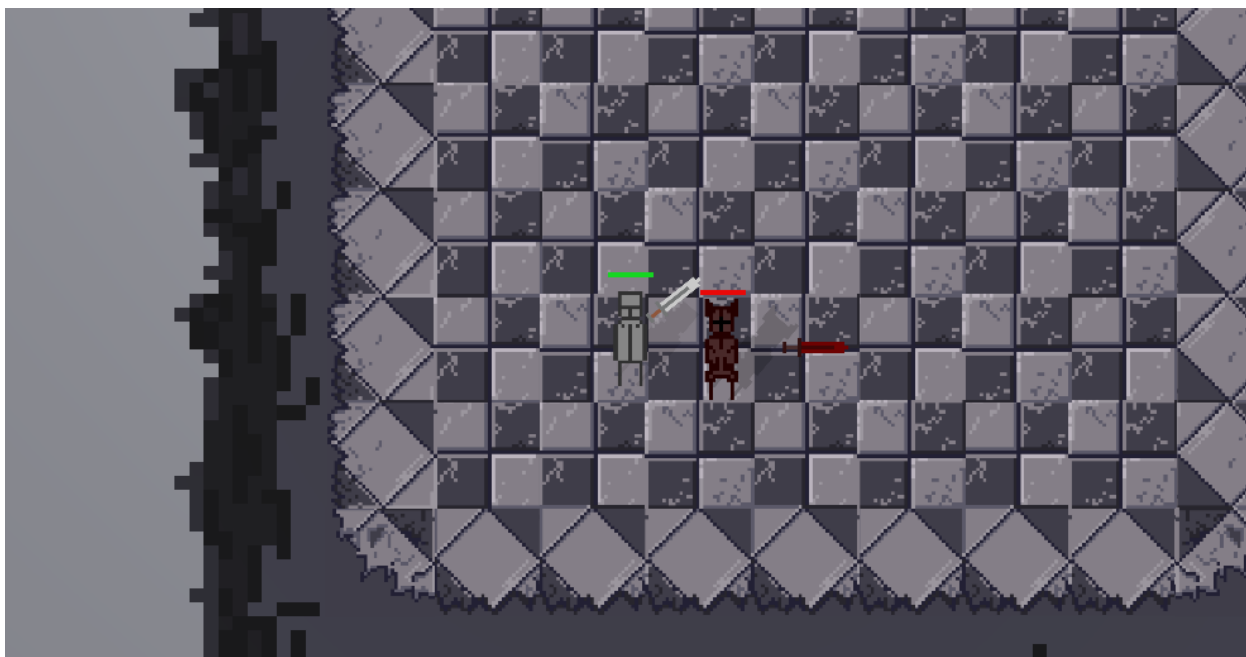


Рисунок Д.3 – Здоровье противника до нанесения урона

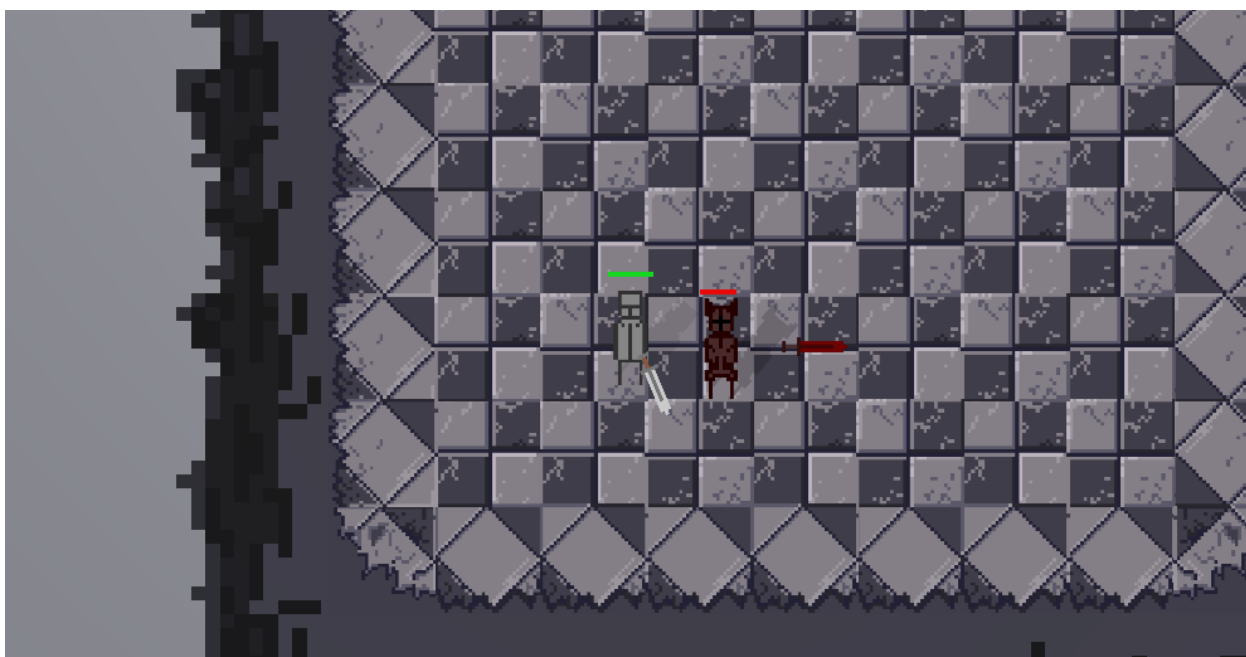


Рисунок Д.4 – Здоровье противника после нанесения урона



Рисунок Д.5 – Победа



Рисунок Д.6 - Поражение