

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

**Роловая компьютерная игра с элементами
случайной генерации локаций**

Студент	_____	Мызгаев Денис Александрович
	(подпись, дата)	(ФИО полностью)
Руководитель	_____	Прошкина Е.Н.
	(подпись, дата)	(фамилия, инициалы)
Нормоконтролёр	_____	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 _____ (протокол заседания ГЭК от _____ № _____)

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

Пенза, 2020

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
«30» декабря 2019 г

**ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ
БАКАЛАВРА**

1. Студент Мызгаев Денис Александрович гр. 16ВП1 факультета ВТ
<Фамилия, имя, отчество полностью>

2. Тема работы Ролевая компьютерная игра с элементами случайной генерации локаций

Тема утверждена приказом ПГУ № 1488/о от "30" декабря 2019

3. Руководитель работы к.т.н., доцент кафедры «МОиПЭВМ» Прошкина Е.Н.

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

1) Разработать ролевую компьютерную игру с элементами случайной генерации локаций под операционную систему Windows 10

2) Среда выполнения: операционная система Windows 10

Языковые средства разработки: C#, UML

Инструментальные средства разработки: Microsoft Visual Studio 2019, Unity

3) Технологии разработки: RUP, ООП

Основные функции компьютерной игры:

а) обеспечить игроку возможность исследовать мир;

б) возможность участвовать в виртуальных сражениях;

в) предоставлять возможность для взаимодействия с виртуальными предметами;

г) возможность взаимодействия с игровыми персонажами;

д) возможность загрузки и сохранения игрового прогресса.

5. Перечень подлежащих разработке вопросов

1) <i>Анализ предметной области и постановка задачи</i>
2) <i>Проектирование</i>
3) <i>Реализация</i>
4) <i>Контроль качества программного обеспечения</i>
5) <i>Оформление пояснительной записки</i>

6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1	<i>Анализ предметной области и постановка задачи</i>	10%	15.03.2020 – 20.03. 2020	
2	<i>Проектирование</i>	25%	20.03. 2020– 30.03.2020	
3	<i>Реализация</i>	30%	30.03. 2020– 30.04. 2020	
4	<i>Контроль качества программного обеспечения</i>	5%	30.04. 2020– 10.05. 2020	
5	<i>Оформление пояснительной записки</i>	10%	10.05. 2020– 20.05. 2020	

Дата выдачи задания " 30 " декабря 2019

Руководитель бакалаврской работы _____
(подпись, дата)

Задание к исполнению принял студент _____
(подпись, дата)

Работу к защите допустить
Декан факультета _____
(подпись, дата)

Реферат

Пояснительная записка содержит 98 листов, 59 рисунков, 6 таблиц, 11 использованных источников, 4 приложения.

КОМПЬЮТЕРНАЯ РОЛЕВАЯ ИГРА С ЭЛЕМЕНТАМИ СЛУЧАЙНОЙ ГЕНЕРАЦИИ ЛОКАЦИЙ, UNITY, C#, UML 2.0.

Объектом разработки является ролевая компьютерная игра с элементами случайной генерации локаций.

Цель работы – разработка компьютерной ролевой игры с элементами случайной генерации локаций.

В результате проведенной работы создана компьютерная ролевая игра с элементами случайной генерации локаций, в которую можно играть и сохранять свой игровой прогресс.

Разработка проводилась на языке программирования C#, в качестве сред разработки использовались среда программирования Visual Studio 2019 Community и игровой движок Unity, операционная система Windows 10. Среда исполнения – операционная система Windows 10.

					ПГУ 09.03.04 – 08БР161.21 ПЗ						
Изм.	Лист	№ докум.	Подп.	Дата	Ролевая компьютерная игра с элементами случайной генерации локаций.	Лит.		Лист	Листов		
Разраб.		Мызгаев Д.А.						3	98		
Пров.		Прошкина Е.Н.									
Реценз.						Гр. 16ВП1					
Н. контр.		Такташкин Д.В.									
Утв.											

Содержание

Введение.....	5
1 Анализ предметной области и постановка задачи.....	6
1.1 Анализ предметной области	6
1.2 Анализ программных продуктов аналогичного назначения	8
1.3 Постановка задачи.....	10
1.4 Анализ требований.....	12
1.4.1 Функциональные требования.....	12
1.4.2 Нефункциональные требования	16
2 Проектирование ролевой компьютерной игры с элементами случайной генерации локаций	17
2.1 Выбор технологий и инструментальных средств разработки.....	17
2.2 Проектирование ролевой системы	19
2.3 Проектирование интерфейсов.....	22
2.4 Проектирование архитектуры.....	26
2.5 Проектирование алгоритмов.....	31
2.6 Проектирование игрового управления	34
2.7 Планирование разработки и оценка бюджета	35
3 Реализация компьютерной игры.....	42
4 Тестирование компьютерной игры	49
4.1 Функциональное тестирование.....	49
Заключение	52
Список использованных источников	53
Приложение А. Глоссарий	55
Приложение Б. Исходный код	57
Приложение В. Игровая графика.....	78
Приложение Г. Результаты тестирования	81

Введение

Компьютерные игры уже давно вошли в жизнь пользователей ПК и являются одним из способов проведения свободного времени. Индустрия игр на подъеме и не имеет значения какое устройство используется для игр: компьютер, смартфон, планшет, игровая приставка.

Существует множество жанров игр от примитивных аркад до игр в виртуальной реальности. Однако жанр ролевых компьютерных игры традиционно остается одним из самых популярных в индустрии.

Компьютерная ролевая игра (Computer Role-Playing Game) – жанр компьютерных игр, основанный на элементах игрового процесса традиционных настольных ролевых игр. В ролевой игре игрок управляет одним или несколькими персонажами, каждый из которых описан набором численных характеристик, списком способностей и умений [1].

Впервые жанр ролевых компьютерных игр появился в 1970-е годы. Но популярность его с тех пор только росла. Существует множество концепций, представлений, реализаций данного жанра. Однако характерные особенности даже по прошествии столь длительного промежутка времени остаются неизменными.

В данном жанре традиционно делается уклон на ролевую систему, сражения и элемент случайности происходящего, что в многом делает его популярным среди множества людей. Этим и обуславливается актуальность создания данного программного проекта.

1 Анализ предметной области и постановка задачи

1.1 Анализ предметной области

Анализ предметной области – это процесс, который позволяет выделить сущности, определить первоначальные требования к функциональности и определить границы проекта [2].

Ролевая компьютерная игра – один из наиболее развитых и успешных жанров игровой индустрии. История развития данного жанра начинается с 1970-х годов. Первые представители данного жанра представляли собой примитивные текстовые игры, но со временем популярность ролевых игр росла, как и росли аппаратные возможности компьютеров, что позволяло ролевым играм эволюционировать в качественном и в количественном плане. Улучшалась графика, игровой процесс, звуковое оформление, росла их продолжительность. Однако основные элементы не менялись с момента зарождения жанра.

Определим основные элементы присущие любой компьютерной ролевой игре: знаки препинания.

- ролевая система;
- исследование;
- боевая система.

Ролевая система – это описание способов создания, изменения и развития персонажей, повышающие их эффективность в игре. Включает в себя множество внутриигровых параметров, на которые игрок влиять прямо или косвенно, что позволяет прогрессировать в игре. Зачастую ролевая система основана на системе классов. Класс – это игровая роль, которая в зависимости от конкретной реализации определяет возможности и ограничения персонажа. Класс представляет собой совокупность навыков. Навык – количественный параметр, определяющий эффективность выполнения того или иного игрового действия. Увеличение навыков происходит по двум основным схемам: навыки увеличиваются при получении внутриигрового опыта, связанного с победой

над противниками и выполнением заданий, или же с частотой использования навыков, чем чаще навык используется, тем эффективнее он становится.

Исследование – это возможность взаимодействия игрока с внутриигровыми персонажами, событиями, локациями и предметами. Перемещение по игровому миру является одной из наиболее важных частей исследования. Игровой мир – внутриигровая область, где происходит место действия игры.

Реализация игрового мира бывает нескольких типов:

- бесшовный мир, где все игровое пространство доступно для перемещения игроком в любой момент;
- глобальная карта мира, возможностью игрока перемещаться по отдельным игровым локациям;
- отдельные игровые локации, сменяющие друг друга, в заранее определенном порядке, игрок перемещается по ним линейно.

Боевая система – это возможности игрока участвовать в виртуальных сражениях. Боевая система всегда основывается на ролевой системе, где виртуальные параметры напрямую влияют на эффективность игрока в сражениях. Боевые системы подразделяются на несколько основных видов:

- пошаговые сражения, где все действия проходят поочередно между игроком и игровыми противниками;
- сражения реального времени, где игрок и противник имеют возможность атаки в любой момент времени;
- сражения реального времени с активной паузой, где игрок имеет возможность остановить сражение в любой момент времени и раздать команды игровому персонажу, которые будут исполнены после снятия паузы.

Таким образом, именно указанные выше критерии, будут основными при разборе аналогичных программных продуктов.

1.2 Анализ программных продуктов аналогичного назначения

Рассмотрим системы подобные, разрабатываемой системе (игре), в рамках дипломной работы.

Final Fantasy – серия ролевых игр, выходявших на многих устройства от ранних игровых консолей до современных компьютеров. Эта серия состоит из 15 основных номерных частей, первая из которых вышла еще в 1987 году. Последняя на данный момент игра из серии вышла в 2020 году. Все игры были выпущены кампанией Square Enix. Все номерные части Final Fantasy совершенно независимы друг от друга, действие игр происходит в разных вымышленных мирах и описывает приключения разных героев, при том, что некоторые элементы, такие как игровая механика и характерные музыкальные темы, часто остаются неизменными.

Основные особенности игр данной серии:

- ролевая система, построенная на системе классов, когда параметры персонажа не зависят от игрока, повышаются самостоятельно при получении игрового опыта;
- исследование основано на перемещении по глобальной карте мира с возможностью входа в игровые локации, на которых возможно взаимодействовать с игровыми персонажами и предметами;
- боевая система первых игр серии пошаговая, в последних частях реального времени с активной паузой;
- часть игрового окружения является сгенерированной псевдослучайно включая следующие элементы: расположение некоторых игровых противников на игровой локации, а также предметы, выпадающие из поверженных противников или сундуков с сокровищами.

Elder Scrolls – серия компьютерных ролевых игр, состоящая из пяти основных номерных частей, разработанная кампанией Bethesda Softworks. Первая часть серии вышла 1994 году, последняя в 2011. Игры серии объединены общей, детально проработанной вселенной и предоставляют игроку большую свободу, позволяя по собственному усмотрению посещать

различные области и города вымышленного фэнтезийного мира и самостоятельно искать интересные места и задания.

Основные особенности включают в себя:

- ролевая система, построенная на классах и навыках, которые увеличиваются в зависимости от частоты их использования;
- игровой мир является бесшовным и не ограничен для исследования;
- часть игрового окружения является псевдослучайно сгенерированной, включая противников и игровые предметы;
- боевая система реального времени;

Fallout – серия постапокалиптических компьютерных ролевых игр. Однако для сравнения было решено взять лишь две игры серии: Fallout 1 и Fallout 2, разработанные студией Black Isle Studios. Действие происходит после ядерной войны на территории США, которая превратилась в радиоактивную пустошь.

Особенности данной игровой серии:

- ролевая система, основанная на навыках, которые игрок может увеличивать, по своему усмотрению, при получении достаточного количества игрового опыта;
- игровой мир представлен, глобальной картой, с возможностью выбора игроком нужной ему локации;
- боевая система является пошаговой, однако основное исследование мира происходит в реальном времени;
- помимо созданный разработчиками игровых локаций, существуют так называемые случайные встречи, которые происходят при перемещении игрока по глобальной карте, от одной локации к другой, во время данных случайных встреч игрок перемещается на случайную локацию со случайными возможными противниками и игровыми предметами.

Исходя из выше представленных программных аналогов была построена сводная таблица 1 для сравнения.

Таблица 1 – Таблица сравнения аналогов

Критерии для сравнения	Final Fantasy	Elder Scrolls	Fallout (1- 2)	Разрабатываемая игра
Боевая система	Пошаговая боевая система	Боевая система реального времени	Боевая система реального времени с активной паузой	Боевая система реального времени
Игровой мир	Глобальная карта мира	Бесшовный мир/бесшовный мир	Глобальная карта мира	Глобальная карта мира
Ролевая система	Основанная на классах	Основанная на навыках и характеристиках	Основанная на навыках и характеристиках	Основанная на характеристиках бесклассовая система
Элемент случайности	Случайное расположение противников и их количества	Случайное расположение противников и их характеристик	Случайное расположение противников и их характеристик	Случайное расположение противников, их количества и характеристик

Исходя из вышеперечисленных примеров можно сделать вывод, что жанр ролевых компьютерных игр чрезвычайно развит. Поэтому довольно сложно определить, какие представители этого жанра лучше в сравниваемых выше критериях, так как они определяют общие обязательные концептуальные особенности жанра. Однако выбор разновидности их для реализации не является показателем качества продуктов, поскольку качество будет определяться уже предпочтения конкретными пользователями.

1.3 Постановка задачи

В рамках дипломного проектирования должна быть разработана ролевая компьютерная игра с элементами случайной генерации локаций.

Исходя из того, что приложение представляет собой игру, следует разделить его возможности на стартового меню, меню загрузки, меню настроек игры, меню создания персонажа и непосредственно игрового процесса.

Игра должна начинаться со стартового меню, которое должно включать следующие возможности:

- начать новую игру;
- загрузить игру;
- настроить игру;
- выйти из игры.

С учетом того, что зачастую ролевые игры отличаются высокой продолжительностью прохождения невозможно сделать такую без системы сохранений. Поэтому меню загрузки и сохранения игры должно предоставить следующие возможности:

- загрузить сохраненную игру;
- сохранить текущую игру.

Меню настроек игры должно быть использовано пользователем для того, чтобы настроить техническую часть игр, для этого оно должно иметь следующие возможности:

- настроить разрешение экрана;
- изменить громкость музыки;
- включить или отключить полноэкранный режим.

С учетом того, что проект представляет собой компьютерную игру, следует выделить в отдельный пункт возможности игрового процесса, потому что они имеют определяющее для пользователя значение. Поэтому для удобного игрового процесса должны присутствовать следующие возможности:

- перемещение персонажа;
- возможность собирать игровые предметы: зелья и золото;
- использование собранных предметов;

- возможность сражений с врагами, при наличие таковых на игровой локации;
- возможность взаимодействия с неигровыми персонажами, что включает в себя возможность: повышения характеристик и покупки зелий;
- возможность перехода по локациям;
- возможность выхода на глобальную карту мира;
- возможность повышать уровень персонажа.

Не следует также забывать, что проект должен быть основан на случайной генерации игровых локаций, что ставит перед игрой требования:

- расстановка противников по локациям должна иметь псевдослучайный порядок, с заранее определенными возможными зонами появления и количественными разбросом;
- характеристики врагов должны генерировать случайно, но с учетом характеристик персонажа игрока;
- предметы, получаемые с врагов, также определяются случайно.

Разработка данного приложения будет проводиться в среде программирования Visual Studio 2019 с интеграцией игрового движка Unity на языке программирования C#.

1.4 Анализ требований

1.4.1 Функциональные требования

Для того, чтобы сформировать требования для программного продукта, было решено отразить в форме диаграммы вариантов использования. Исходя из поставленных ранее задач, была построена следующая диаграмма вариантов использования, представленная на рисунке 1 [3], на которой описаны основные функциональные особенности, которые должны быть в реализуемой системе, для того чтобы ее реализацию можно было бы считать успешной с точки зрения поставленных задач на этапе анализа предметной области реализуемого программного проекта и в целом выполнение выпускной квалификационной работы.

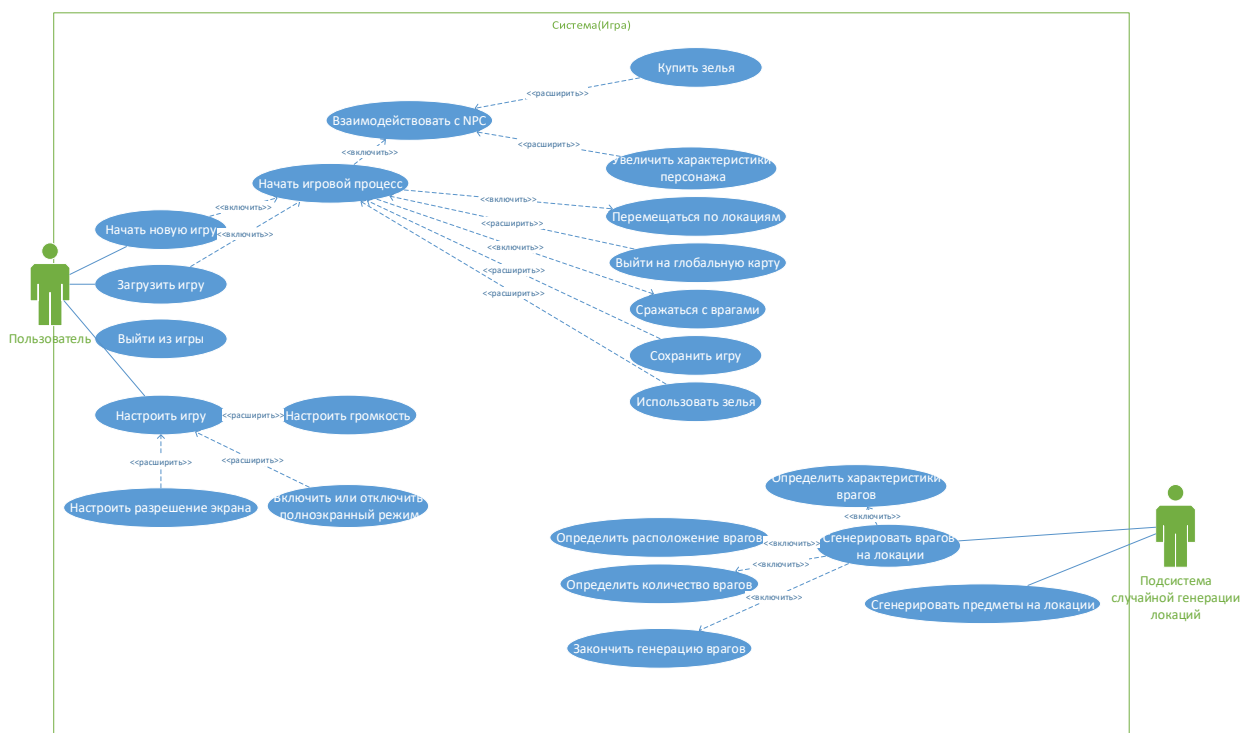


Рисунок 1 – Диаграмма вариантов использования

Исходя из диаграммы, представленной выше, можно сделать о присутствии двух основных акторов: пользователя(игрока) и подсистема случайной генерации локаций, чья роль заключает в генерации части окружения игровых уровней. Данная подсистема генерирует только часть от общего количества игровых уровней, только те, где могут присутствовать противники.

В ходе планирования разработки данного программного продукта было решено отдельно выделить следующие части диаграммы вариантов использования для создания их спецификации: Настроить игру, Загрузить игру, Сгенерировать врагов на локации. Данные прецеденты были выбраны, в качестве основных по причине того, что они являются принципиальными для удобства пользователей.

Опишем спецификацию прецедентов. Рассмотрение было решено начать с варианта использования «Настроить игру». Спецификация данного прецедента изображена на рисунке 2.

Прецедент: Настроить игру
ID: 1
Краткое описание: Пользователь настраивает игру под свои нужды
Действующие лица: Пользователь, система
Предусловия: Пользователь запустил игру
Основной поток: <ol style="list-style-type: none"> 1. Пользователь выбирает в меню кнопку «Настройки» 2. Система отображает настройки игры 3. Пользователь изменяет нужные настройки <ol style="list-style-type: none"> a. Изменить разрешение экрана b. Включить или отключить полноэкранный режим c. Изменить громкость музыки 4. Система сохраняет настройки игры
Постусловия: Настройки сохранены

Рисунок 2 – Спецификация прецедента «Настроить игру»

Вышеприведенная спецификация прецедента «Настроить игру» описывает возможные настройки доступные для игрока в меню настроек в их числе управление и разрешение экрана. Рассмотрим спецификацию прецедента «Загрузить игру» (рисунок 3).

Прецедент: Загрузить игру
ID: 2
Краткое описание: Пользователь загружает игровое сохранение
Действующие лица: Пользователь, система
Предусловия: Пользователь запустил игру
Основной поток: <ol style="list-style-type: none"> 1. Пользователь выбирает в меню кнопку «Загрузить игру» 2. Система отображает список игровых сохранений 3. Пользователь выбирает нужное сохранение и нажимает кнопку «Загрузить» 4. Система запускает игровой процесс с учетом загруженного сохранения 5. include(Начать игровой процесс)
Постусловия: Пользователю стал доступен игровой процесс

Рисунок 3 – Спецификация прецедента «Загрузить игру»

Выше рассмотренная спецификация прецедента варианта использования «Загрузить игру» описывает последовательность действий при загрузке ранее сохраненного пользователем состояния игрового прогресса.

Рассмотрим спецификацию прецедента «Сгенерировать врагов на локации» (рисунок 4).

Прецедент: Сгенерировать врагов на локации
ID: 3
Краткое описание: Подсистема случайной генерации локаций создает врагов на текущей локации
Действующие лица: Подсистема случайной генерации локаций, система
Предусловия: Пользователь посетил игровую локацию
Основной поток: <ol style="list-style-type: none"> 1. Подсистема случайной генерации локаций начинает выбор противников для создания 2. Include(Определить характеристики врагов) 3. Include(Определить расположение врагов) 4. Include(Определить количество врагов) 5. Include(Закончить генерацию врагов) 6. Система создает врагов на локации в зависимости подсистемы случайной генерации локаций
Постусловия: -

Рисунок 4 – Спецификация прецедента «Сгенерировать врагов на локации»

Выше представленная спецификация прецедента «Сгенерировать врагов на локации», описывает функционал подсистемы случайной генерации локаций. Из данного потока событий становится понятным, что процесс генерации врагов на локации представляет из себя многоступенчатую структуру с несколькими последовательными действиями, которые генерируют врагов, в зависимости от многих параметров.

Следует отметить, что для данного проекта подсистема случайной генерации локаций является чрезвычайно важной и позволяет разнообразить игровой опыт за счет многочисленных вариантов сгенерированных локаций.

1.4.2 Нефункциональные требования

Помимо обозначенных ранее функциональных требований, к разрабатываемым программным средствам было предъявлено ряд нефункциональных.

Нефункциональные требования – требования, определяющие свойства, которые система должна демонстрировать, или ограничения, которые она должна соблюдать, не относящиеся к поведению системы [4].

Программное средство должно работать под управлением операционной системы Windows 10. Это требование является ключевым фактором разработки.

Программное средство должно быть разработано с использованием игрового движка Unity Engine версии 2019.3.12f1.

Программное средство должно быть разработано на языке программирования C# в среде Visual Studio 2019, что обусловлено высокой скоростью работы данного языка, а также наличием компонентов, пригодных для использования при разработке данных программных средств.

Формат хранения игровых данных должен быть JSON.

Необходимо обеспечить высокую производительность данной игры.

Данный проект должен представлять собой двухмерную игру с сверху с боевой системой реального времени. Где игрок и противники могут действовать независимо друг от друга. Каждая локация представляет собой огороженное пространство с возможностью перехода между ними.

2 Проектирование ролевой компьютерной игры с элементами случайной генерации локаций

2.1 Выбор технологий и инструментальных средств разработки

Выбор технологии разработки осуществлялся между каскадной, RAD и RUP моделями.

Каскадную модель следует использовать при следующих условиях:

- требования известны, понятны и зафиксированы противоречивых требований не имеется;
- нет проблем с доступностью программистов нужной квалификации;
- в относительно небольших проектах.

Для данного проекта требования могут изменяться по ходу выполнения работы, поэтому использование этой модели нежелательно.

RAD-модель может использоваться только при наличии высококвалифицированных и узкоспециализированных архитекторов. Бюджет проекта большой, чтобы оплатить этих специалистов вместе со стоимостью готовых инструментов автоматизированной сборки. RAD-модель может быть выбрана при уверенном знании целевого бизнеса и необходимости срочного производства системы в течение 2 – 3 месяцев. Для выбранного проекта это не требуется [5].

RUP — итеративный подход, имеющий следующие принципы, обусловившие выбор данной методологии:

- итеративная модель разработки – устранение рисков на каждой стадии проекта позволяет лучше понять проблему и вносить необходимые изменения, пока не будет найдено приемлемое решение;
- управление требованиями – RUP описывает процесс организации и отслеживания функциональных требований, документации и выбора оптимальных решений (как в процессе разработки, так и при ведении бизнеса);

- компонентная архитектура – архитектура системы разбивается на компоненты, которые можно использовать как в текущем, так и в будущих проектах;
- визуальное моделирование ПО – RUP методология разработки показывает, как создать визуальную модель программного обеспечения, чтобы понять структуру и поведение архитектуры и его компонентов;
- проверка качества ПО – в процессе разработки программного обеспечения контролируется качество всех действий команды;
- контроль внесённых изменений – отслеживание изменений позволяет выстроить непрерывный процесс разработки. Создаётся благоприятная обстановка, в рамках которой команда будет защищена от изменений в рабочем процессе [6].

Данный проект будет реализоваться на базе игрового движка Unity Engine. Оно включает в себя такие возможности как: создавать игры, моделировать различные объекты и их поведение в реальном времени, широко используемое как программистами и разработчиками игр, так и дизайнерами, художниками, аниматорами и режиссерами [7, 8]. Одной из важных задач, решаемых игровыми движками, является моделирование различных физических процессов в режиме реального времени. В Unity для этого предоставляется удобный и гибкий интерфейс, при грамотном подходе позволяющий не просто моделировать поведение многих объектов, но и контролировать его правильность и изменять ее, если это будет необходимо разработчику в его приложении.

Также важным аспектом этого движка является его кроссплатформенность, т.к. она позволяет при одинаковом подходе к разработке приложений свободно и без труда компилировать их на любую из доступных в широком списке платформ.

Функциональность игровых объектов, отображаемых на экране движком Unity, реализуется на языке программирования C#. Это позволяет программисту использовать преимущества объектно-

ориентированного программирования, что позволяет упростить реализацию его идей. Для написания скриптов на данном языке будет использоваться среда программирования Visual Studio Community 2019, потому что она имеет встроенные средства для интеграции с Unity.

2.2 Проектирование ролевой системы

Поскольку разрабатываемый проект представляет собой ролевую компьютерную игру, необходимо четко обозначить правила, по которым будет она работать.

Игрок – игровой объект, который управляется пользователем. Ему присущ ряд характеристик, который описан ниже. Внешний вид игрок представлен в виде спрайта в приложении В.1.

Ролевая система данной игры на основывается на 3 главных параметрах: здоровье, выносливость, магия.

Здоровье – характеристика, которая обозначает количество возможного для получения урона перед окончанием игры, в случае игрока, или уничтожением в случае игровых противников.

Выносливость – характеристика, которая обозначает количество физических ударов, которые способен нанести персонаж.

Магия – характеристика, которая обозначает количество магических ударов, которые способен нанести персонаж.

Поскольку речь идет о ролевой системе, не мало важной частью является развитие вышеназванных характеристик, которое зависит от уровня персонажа. Уровень – характеристика, которая обозначает общий уровень прогресса в игре. Для игрока повышение уровня напрямую завязано на победах над противниками, а точнее получении очков опыта.

Очки опыта – характеристика, которая обозначает прогресс персонажа в получении нового уровня. Данная характеристика увеличивается при победе над соперниками, и что важно, чем выше уровень побежденного соперника, тем больше получаемых очков опыта. Еще одной важной особенностью

является постепенный рост требований в количестве очков опыта, которые нужно получить перед получением нового уровня.

После повышения игрового уровня игрок получает очки уровня. Очки уровня – характеристика, которая отражает количество очков, которые могут быть потрачены на увеличение трех главных характеристик.

С учетом частого упоминания необходимо систематизировать значение термина противник. Противник (враг) – игровой объект, который обладает четырьмя главными характеристиками: здоровьем, выносливостью, магией, а также уровнем. Подобно игроку использует боевую систему. Шаблоны внешнего вида противников созданы заранее, однако их параметры генерируются случайно. Примеры противников представлены в виде спрайтов в приложении В.2.

Боевая система основана на трех базовых атаках: ближний удар, дальний удар, магический удар.

Ближний удар – действие, которое наносит урон здоровью, потерпевшей стороне на ближней дистанции. Ближний удар наносится прямо перед персонажем на небольшом расстоянии, и при каждом использовании расходует выносливость.

Дальний удар – действие, которое наносит урон здоровью, потерпевшей стороне на дальней дистанции. Представляет собой запущенный снаряд имеющей, определенный запас расстояния полета. Расходует выносливость.

Магический удар – действие, которое наносит урон здоровью, потерпевшей стороне на дальней дистанции. Состоит из двух этапов: полет снаряд до цели и дальнейшего взрыва на небольшом радиусе, поражающем рядом расположенных персонажей. Расходует магию.

С учетом того факта, что каждый из вышеназванных типов атак расходует тот или иной главный параметр было решено сделать их восстановление автоматическим со временем, но довольно медленным. Однако учитывая тот факт, что боевая система проходит в реальном времени было решено ввести концепцию так называемых «зелий».

Зелье – предмет, который может быть использован для восстановления одного из трех параметров. Существует три вида зелий: зелье здоровья, зелье выносливости, зелье магии. Каждый вид предназначен для своей характеристики, что следует из названия. Зелье мгновенно восстанавливает значение параметра на треть от максимального значения.

Еще одним полезным видом предметов являются монеты. Монеты – предмет, который собирается с побежденных врагов и используется в качестве платы за зелья и увеличение параметров с помощью очков уровня.

С точки зрения общей игровой концепции было решено разделить все игровые уровни на три основные категории: город, глобальная карта, подземелья.

Город – игровая локация на которой возможно взаимодействие с NPC в виде диалога, который позволяет покупать зелья или повышать главные параметры. На данной локации нет противников. Город представлен в приложении В.3.

Глобальная карта – игровая локация, на которой игрок перемещается между игровыми локациями, такими как город или подземелья. На данной локации также нет противников. Глобальная карта представлена в приложении В.4.

Подземелья – игровые локации, которые предназначены для сражений игрока с противниками. Каждое подземелье сделано заранее, однако на них генерируются враги только при посещении их игроком в зависимости от его уровня и некоторых других характеристик. Пример подземелья представлен в приложении В.5.

Главная цель в игре – это повышение уровня персонажа, которое по мере продвижения становится только сложнее из-за увеличения уровней противников, их количества, а также повышения требований в очках опыта.

2.3 Проектирование интерфейсов

На основе функциональных требований, предъявленных к игре ранее, были спроектированы несколько видов меню:

- главное меню;
- меню паузы;
- меню настроек игры;
- меню загрузки и сохранения;
- меню игрового процесса (HUD).

Первое, что видит пользователь при запуске приложения – это экран главного меню. Данный экран представлен на рисунке 4.



Рисунок 4 – Главное меню

Экран главного меню состоит из следующих элементов, отмеченных цифрами:

- 1 – надпись;
- 2 – кнопка «начала игры»;
- 3 – кнопка «загрузки игры»;
- 4 – кнопка «настройки»;
- 5 – кнопка «выхода из игры».

При выборе пользователем кнопки настройки, он попадает в меню внутриигровых настроек, представленное на рисунке 5.



Рисунок 5 – Меню настроек

Меню настроек состоит из следующий компонентов:

- 1 – надпись;
- 2 – флажок для выбора полноэкранная режима;
- 3 – ползунок громкости;
- 4 – выпадающий список разрешений экрана;
- 5 – кнопка возврата в главное меню.

В случае выбора пользователем «Загрузить игру», он попадает на экран загрузки сохраненных игр, представленный на рисунке 6.



Рисунок 6 – Меню загрузки

Меню загрузки состоит из следующих элементов:

- 1 – надпись;
- 2 – кнопка выбора 1 слота для сохранения;
- 3 – кнопка выбора 2 слота для сохранения;
- 4 – кнопка выбора 3 слота для сохранения;
- 5 – кнопка возврата в главное меню.

С учетом специфики работы с сохранениями меню загрузки, также выполняет функцию меню сохранения, отсутствие разницы во внешнем виде нивелируется, разнице в функционале. Если выбор слота в меню загрузки загружает сохраненную игру, то тоже действие в меню сохранений сохранение текущего состояния игры в выбранный слот.

Важной частью игрового процесса является возможность вызвать меню паузы, которое представлено на рисунке 7.

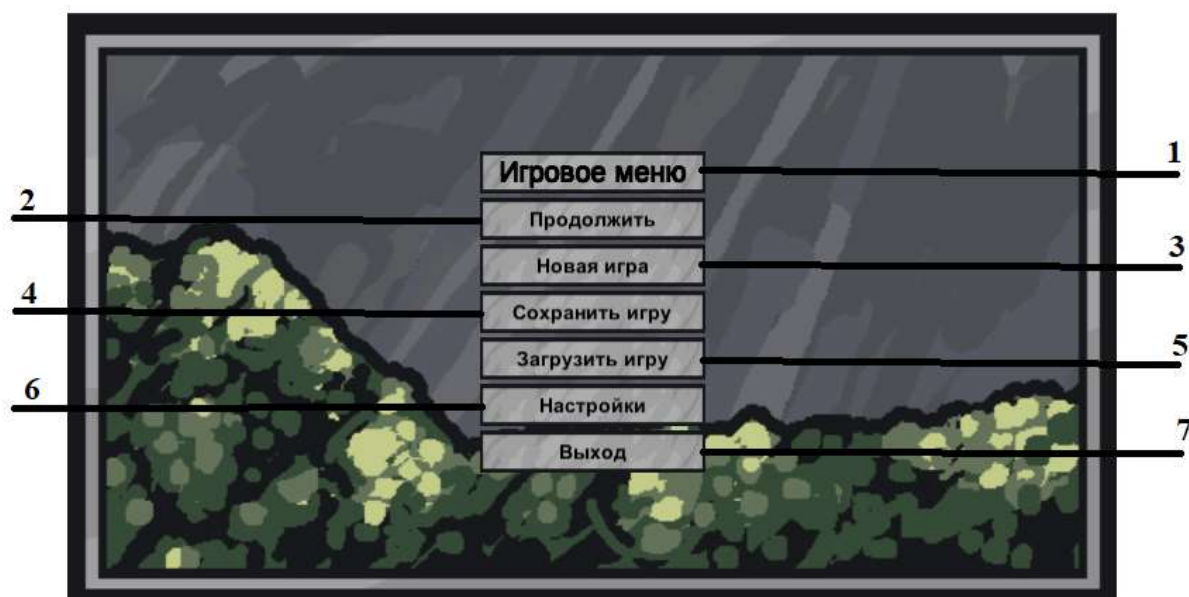


Рисунок 7 – Меню паузы

Меню паузы состоит из следующих элементов:

- 1 – надпись;
- 2 – кнопка «продолжить»;
- 3 – кнопка «новая игра»;
- 4 – кнопка «сохранить игру»;

- 5 – кнопка «загрузить игру»;
- 6 – кнопка «настройки»;
- 7 – кнопка «выхода из игры».

Важнейшей частью непосредственно игрового процесса является так называемый HUD. HUD – часть игрового интерфейса игрока, отображающаяся на фоне происходящего в игре, несет полезную для пользователя информацию об игровой ситуации. Для его демонстрации представлен на рисунке 8, являющийся скриншотом игрового момента.



Рисунок 8 – HUD

HUD состоит из следующих компонентов:

- 1 – показатель здоровья игрока;
- 2 – показатель выносливости игрока;
- 3 – показатель магии игрока;
- 4 – показатель текущего уровня игрока;
- 5 – показатель текущего опыта игрока;
- 6 – показатель текущих очков уровня;
- 7 – показатель количества зелий выносливости;
- 8 – показатель количества зелий здоровья;
- 9 – показатель количества зелий магии;

10 – показатель количества монет.

Таким образом данный HUD отображает все текущие показатели персонажа игрока.

2.4 Проектирование архитектуры

Логика игрового движка Unity построена на так называемых игровых объектах (GameObject). Каждый такой объект представляет собой совокупность компонентов (Component), которые реализуют функционал объектов. Каждый компонент представляет собой скрипт написанный на C#.

Также важной особенностью игрового движка Unity является то, что все игровые объекты находятся на так называемой сцене (Scene). Каждая сцена представляет собой изолированное пространство, однако Unity поддерживает возможность переключения между ними прямо во время игрового процесса, с возможностью сохранения. Что особенно важно Unity может держать несколько сцен одновременно или, загружая новую сцену, сохранять часть объектов предыдущей в памяти посредством команды DontDestroyOnLoad().

Вся игровая логика строиться на компонентах-скриптах, которые являются наследниками единого класса «MonoBehaviour». Исходя из этого каждый из них обладает рядом специфичных методов, из которых напрямую строится логика движка путем задания им нужной функциональности. В их список входят:

- Update(), который выполняется каждую смену кадра;
- Awake(), который выполняется однажды после создания объекта на сцене;
- Start(), которая вызывается однажды, но после Awake().

Исходя из вышесказанного, вся планируемая логика игры будет представлять собой совокупность объектов с прикрепленными к ним игровыми скриптами [9].

С учетом компонентной модели движка Unity построить диаграмму классов в привычном понимании довольно сложно, поэтому на диаграмме

представленной ниже обозначены базовые игровые скрипты, отвечающие за непосредственно игровой процесс. Однако учитывая большое количество подобных классов, было решено разделить их на два функциональных блока: классы, ответственные за игровой процесс, и классы, ответственные за интерфейс и систему сохранений. Далее на рисунке 9 изображена диаграмма классов, ответственных за игровой процесс.

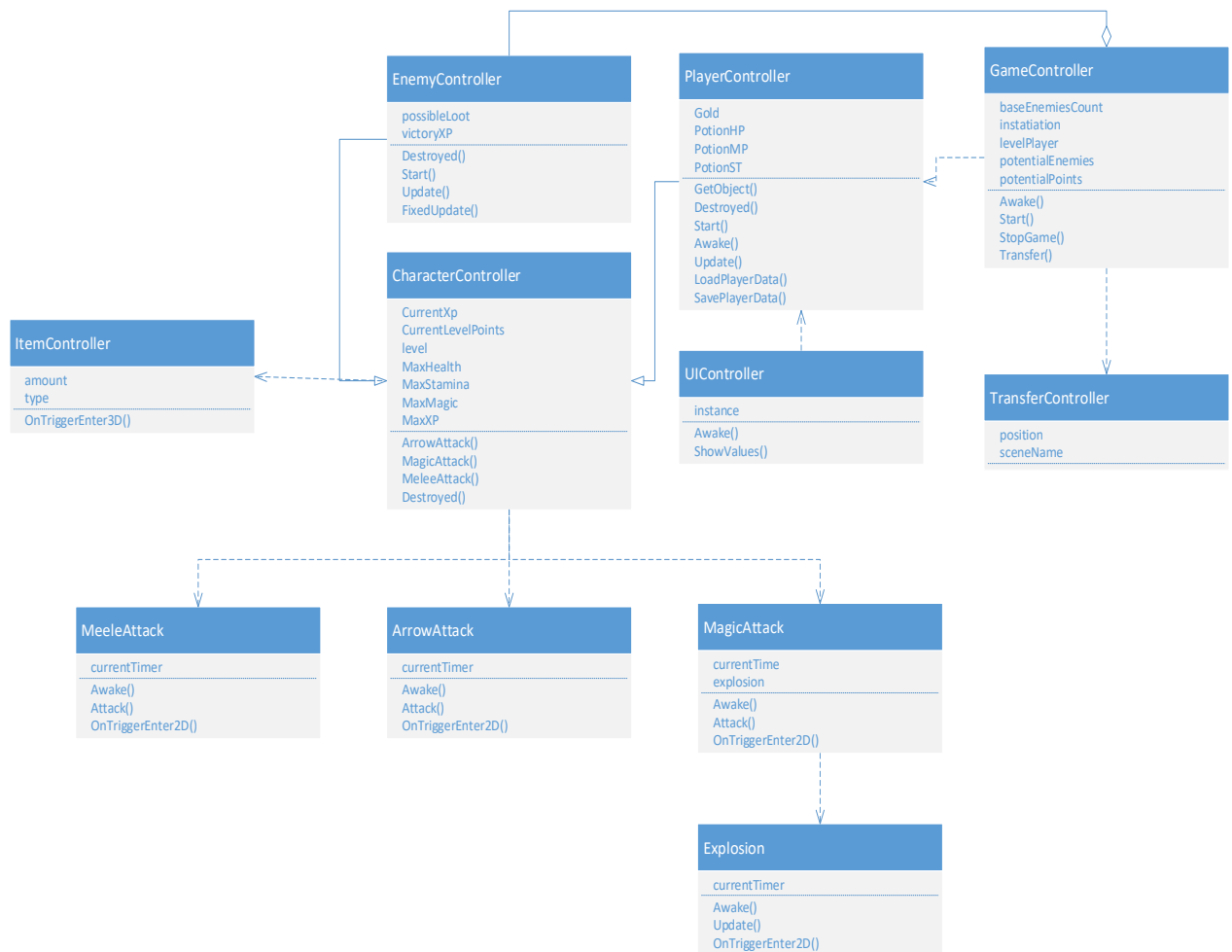


Рисунок 9 – Диаграмма классов игрового процесса

Далее необходимо привести описание всех представленных выше игровых скриптов и некоторых особенно важных методов и полей.

Класс **CharacterController** является родительским классом для **EnemyController** и **PlayerController**. Содержит в себе основные поля для персонажей и врагов, так же методы для боевой системы такие как: **MagicAttack**, **ArrowAttack**, **MeeleAttack**. Немаловажным также является поле `lookDirection` которое представляет из себя двухмерный вектор, хранящий

информацию о повороте объекта, что необходимо для боевой системы и прочего взаимодействия. Метод `Destroyed` вызывается при уменьшении здоровья до 0.

Класс `PlayerController` определяет все возможности игрока, в их числе перемещение и боевая, отслеживаемые в методе `Update()`, а также возможность подбора предметов, реализуемая в методе `GetObject()`. Методы `LoadPlayerData()` и `SavePlayerData()` принимают и передают основные характеристики игрока, что используется для загрузки и сохранения игры. Свойства `Gold`, `PotionHP`, `PotionST`, `PotionMP`, хранят значения предметов игрока: монет, зелий здоровья, выносливости и магии. Метод `Destroyed()` возвращает пользователя в главное меню в случае проигрыша, из-за потери здоровья.

Класс `EnemyController` определяет функциональные возможности игровых противников. В методе `Update()` реализована возможность атаки игрока при его приближении на достаточное расстояние. Поле `TimeBetweenAtack` определяет периодичность вражеских атак. `Destroyed()` определяет, выпавший предмет.

Класс `GameController` – основной класс управления игрой. Создает объект игрока в `Awake()`, генерирует врагов на локации в `GenerateEnemies()`, отвечает за переход персонажа между сценами Unity в методе `Transfer()`.

Класс `ItemController` является обязательным компонентом любого подбираемого предмета. Поле `Type` определяет тип предмета, `amount` его количество. Метод `OnEnterTrigger2D()` является стандартным методом Unity при столкновении объектов, в данном случае его функционал вызывает метод `GetObject()` и передает данные классу `PlayerController`.

Класс `UIController` отвечает за прорисовку HUD.

Класс `TransferController` является вспомогательным компонентом для объектов, позволяющих персонажу перемещаться между локациями. Содержит поля `position`, которое определяет позицию персонажу при загрузке новой

сцены, а также `sceneName`, который содержит название сцены, которая будет загружена.

Класс `MeeleAttack` реализует функционал для ближней атаки. Поле `animator` хранит анимацию объекта. Поле `assaulter` указывает на предпринявшего атаку. Поле `time` хранит время существования объекта. Метод `Attack()` запускает объект исходя из поворота персонажа, осуществившего атаку. Метод `OnTriggerEnter2d()` срабатывает при встрече объекта с персонажем и наносит урон.

Класс `ArrowAttack` реализует функционал для дальней атаки. Поле `animator` хранит анимацию объекта. Поле `assaulter` указывает на предпринявшего атаку. Поле `time` хранит время существования объекта. Метод `Attack()` запускает объект исходя из поворота персонажа, осуществившего атаку. Метод `OnTriggerEnter2d()` срабатывает при встрече объекта с персонажем и наносит урон.

Класс `MagicAttack` реализует функционал для магической атаки. Поле `animator` хранит анимацию объекта. Поле `assaulter` указывает на предпринявшего атаку. Поле `time` хранит время существования объекта. Метод `Attack()` запускает объект исходя из поворота персонажа, осуществившего атаку. Метод `OnTriggerEnter2d()` срабатывает при встрече объекта с персонажем и наносит урон, при этом порождая новый объект.

Класс `Explosion` реализует функционал для взрыва, возникающего после встречи `MagicAttack` с персонажем или истечением времени существования объекта. Метод `OnTriggerEnter2d()` наносит урон все персонажам область поражения. Поле `animator` хранит анимацию взрыва.

Далее представим диаграмму классов для управления интерфейсом и системой сохранений на рисунке 10.

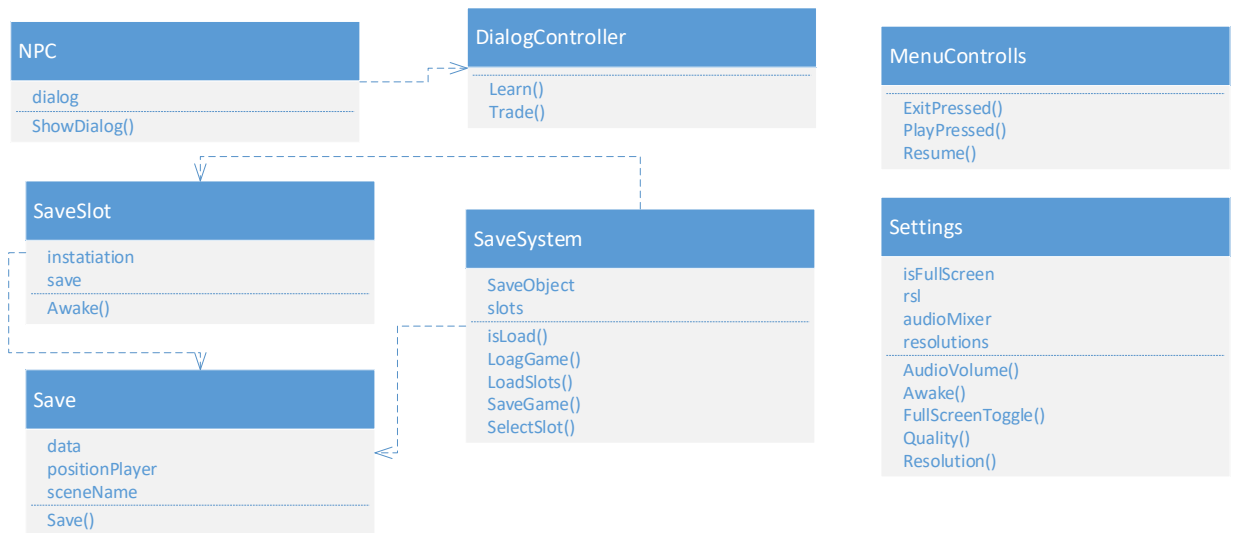


Рисунок 10 – Диаграмма классов управления меню и системой сохранений

Класс `SaveSystem` отвечает за создание игровых сохранений, их загрузку, а также частично за меню загрузки и сохранений. Метод `LoadSlots()` просматривает папку с игрой на наличие игровых сохранений и отображает их в меню загрузки и сохранений. Свойство `isLoad()` отвечает за то, используется ли меню сохранений или меню. `SelectSlot()` метод определяет нажатый пользователем в меню слот сохранения и в зависимости от значения `isLoad()` вызывает методы `LoadGame()` или `SaveGame()`, которые отвечают за загрузку и запись данных соответственно.

Класс `DialogController` отвечает за обработку выборов игрока в диалогах. Метод `Learn()` используется в случае если игрок хочет увеличить свои характеристики. Метод `Trade()` обрабатывает покупку зелий игроком.

Класс `MenuControlls` обрабатывает нажатия клавиш в главном меню и в меню паузы. `Resume()` обрабатывает кнопку «Продолжить» и закрывает меню паузы и возвращает к игре. `ExitPressed()` обрабатывает нажатие кнопки «Выход» и закрывает игру. `PlayPressed()` обрабатывает нажатие кнопки «Новая Игра» и загружает начальную сцену.

`Save` – сериализуемый класс, используемый в качестве структуры для хранения игровых. Используется при загрузке и сохранении. Поле `data` хранит данные о характеристиках игрока, `sceneName` хранит название сцены на

которой находился игрок в момент сохранения, position хранит позицию игрока на момент сохранения.

Класс Settings отвечает за обработку меню настроек. Поле AudioManager хранит компонент для воспроизведения звука. Поле isFullScreen хранит данные полно размерности экрана. Поле Resolution хранит данные о текущем разрешении экрана. Поле rs1 хранит все доступные разрешения экрана для текущего монитора. Метод AudioVolume() вызывается при изменении значения ползунка громкости и меняет громкость музыки в игре. Метод FullscreenToggle() меняет полно размерность экрана. Метод Resolution() меняет разрешение экрана.

Листинг перечисленных выше классов находится в приложении Б.

2.5 Проектирование алгоритмов

Важнейшим алгоритмом данной разработки является алгоритм случайной генерации локаций. Данный алгоритм представлен в виде диаграммы последовательность на рисунке 11.



Рисунок 11 – Диаграмма деятельности для генерации врагов

Вышеприведенная схема требует ряда пояснений. Решение «Проверить точки спавна» используется для определения наличия на игровой локации точек спавна. Точки спавна – особый вид игровых объектов, которые используются для установка заранее определенных зон генерации врагов. Такие точки присутствуют только в локациях подземельях и отсутствуют во все прочих. Именно наличие таковых точек определяет локацию как подземелье.

Решение «Определить количество врагов» используется для определения заранее количества потенциальных врагов на локации. Данное количество рассчитывается по следующей формуле:

$$Enemies = \log_2 Level + base,$$

где Level – уровень игрока;

base – базовое количество врагов на локации, определяется для каждой индивидуально, для того чтобы некоторые локации упростить для прохождения, а другие усложнить;

Enemies – количество, врагов, которое будет создано на текущей локации.

Решение «Выбрать шаблон враг» основывается на концепции Prefab в Unity. Prefab – это шаблон игрового объекта, который может быть использован неоднократно. В данном случае эти Prefab врагов определяют их внешний вид и анимации.

Решение «Выбрать место» определяет случайно на какой точке спавна будет создан враг.

Решение «Выбрать уровень врага» определяет уровень игрока случайно по схеме: от 1 до текущего уровня игрока.

Решение «Распределить характеристики врага» определяет случайно три основных параметра для врагов.

Решение «Определить выпадающие предметы с врага» определяет предметы, которые появятся на месте сраженного врага. Реализуется следующим образом: 30% вероятность на выпадение любого зелья и 50%

вероятность выпадения монет. Количество монет определяется по следующей формуле:

$$Gold = Level * Rnd,$$

где Level – уровень игрока;

Rnd – случайное число от 20 до 100;

Gold – выпавшие монеты.

Решение «Определить получаемый от врага опыт» определяет опыт, который игрок получит при уничтожения врага. Реализуется по формуле:

$$XP = BaseXP + BaseXP * 0.3 * Level,$$

где BaseXP – базовое значение опыта, получаемого с врага;

Level – уровень игрока;

XP – опыт получаемый с врага.

Решение «Разместить врага на локации» перемещает созданного врага в точку спавна.

Отдельно также стоит выделить процесса загрузки сохранений представленный в виде диаграммы последовательности на рисунке 12.

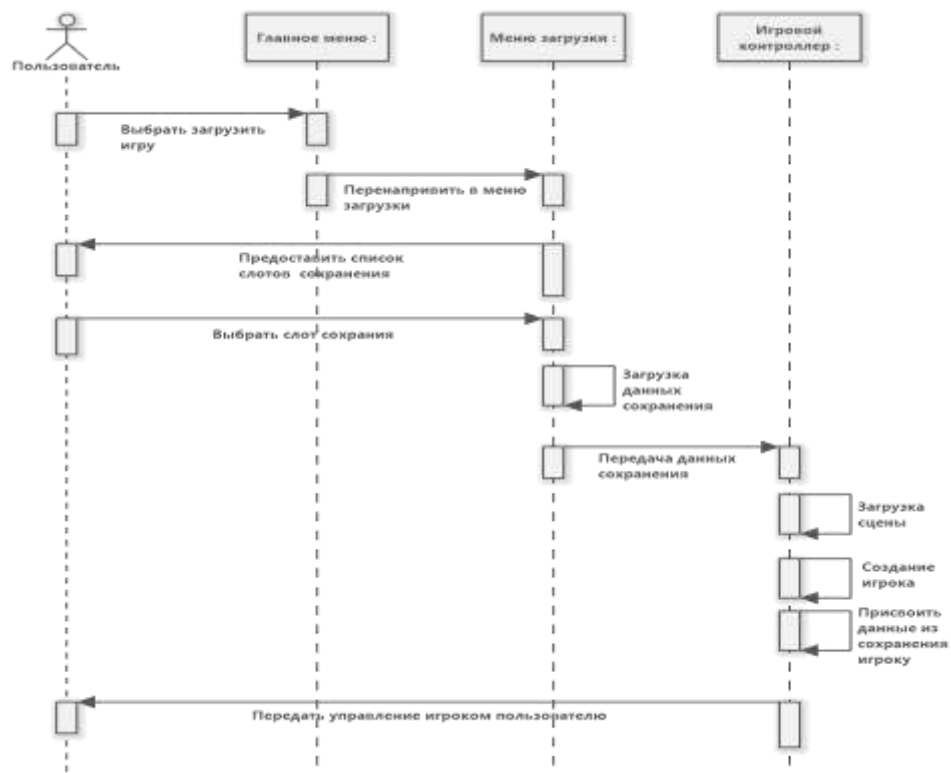


Рисунок 12 – Диаграмма последовательности для загрузки игры

Отдельно необходимо описать некоторые моменты процесса загрузки.

Игровой контроллер – это объект, который оперирует большей частью игровой логики, в том числе: создает объекты на сцене, генерирует врагов, переносит игрока с одной сцены на другую, а также отвечает за загрузку игровых данных игрока.

Игровое сохранения – файл, содержащий сведения о характеристиках персонажа, его местоположении на сцене, а также о самой сцене, которая была активной в момент сохранения игровых данных.

2.6 Проектирование игрового управления

Управление персонажем неотъемлемая часть игрового процесса, поэтому для было решено разделить игровое управление на две функциональные части: управление в игровых меню и управление непосредственно в игре. Для управления в игровых меню и диалогах с игровыми персонажами была выбрана мышь. Для управления перемещением игрока и боевой системой была выбрана клавиатуры. Ниже в таблице 2 указаны назначения клавиш, используемых в процессе управления персонажем.

Таблица 2 – Игровое управление

Клавиши клавиатуры	Выполняемое действие
WASD	Перемещение по игровой локации
E	Взаимодействие с игровыми персонажами и другими предметами
X	Атака ближнего боя
C	Атака дальнего боя
V	Магическая атака
R	Зелье здоровья
T	Зелье выносливости
Y	Зелье магии
ESC	Вызов меню паузы

Данное игровое управление построено с учетом возможности для игрока играть одной рукой на клавиатуре, благодаря близкому расположению клавиш, а другой использовать мышь в меню и диалогах.

2.7 Планирование разработки и оценка бюджета

Управление программными проектами – это особый вид управления проектами, смысл которого заключается в планировании, отслеживании и контроле над проектами по разработке ПО [10].

Для эффективного планирования разработки данного проекта требуется выделить основные задачи. Список задач представлен в таблице 3.

Таблица 3 – Список задач проекта

№	Название	Предшественники	Длительность
1	Начало проекта		0 дней
2	Анализ требований		11 дней
3	Сбор информации о предметной области	1	3 дней
4	Изучение и анализ информации о предметной области	3	3 дней
5	Изучение и анализ программных аналогов	3	2 дней
6	Формирование требований к проекту	4	3 дней
7	Уточнение требований к проекту	5;6	2 дней
8	Анализ требований завершен	7	0 дней
9	Проектирование ролевой компьютерной игры с элементами случайной генерации локаций		15 дней
10	Проектирование общей архитектуры проекта	8	5 дней
11	Анализ возможных технологий	10	3 дней
12	Выбор технологий для реализации	11	2 дней
13	Анализ возможного инструментария разработки	10	3 дней
14	Выбор инструментария разработки	13	3 дней
15	Проектирование архитектуры компонентов	14;12	4 дней
16	Проектирование проекта завершенно	15	0 дней
17	Реализация ролевой компьютерной игры с элементами случайной генерации локаций		24 дней
18	Реализация системы генерации игровых уровней	16	9 дней
19	Реализация геймплейных элементов	16	15 дней

Продолжение таблицы 3

№	Название	Предшественники	Длительность
20	Реализация пользовательского интерфейса	16	12 дней
21	Реализация готовой игровой сборки	18;19;20	9 дней
22	Реализация проекта завершена	21	0 дней
23	Отладка и тестирование		28 дней
24	Тестирование системы генерации игровых уровней	18	3 дней
25	Тестирование геймплейных элементов	19	3 дней
26	Тестирование пользовательского интерфейса	20	3 дней
27	Отладка работы системы	24;25;26;22	5 дней
28	Исправление ошибок	27	4 дней
29	Тестирование системы в целом	28	4 дней
30	Отладка и тестирование завершены	29	0 дней
31	Конец проекта	30	0 дней

Чтобы планирование разработки было эффективным присутствует необходимость в оценке имеющихся в распоряжении проекта ресурсов. Перечень ресурсов и затрат представлен в таблице 4.

Таблица 4 – Перечень ресурсов и затрат

Название ресурса	Тип	Затраты	
		Станд. ставка	Ставка сверхур.
Программист 1	Т	70,00 Р/ч	100,00 Р/ч
Программист 2	Т	80,00 Р/ч	110,00 Р/ч
Тестировщик	Т	60,00 Р/ч	90,00 Р/ч

Ресурсы, используемые в проекте, были добавлены в Microsoft Project и представлены на рисунке 13.

	И	Название ресурса	Тип	Единицы измерения материалов	Краткое название	Группа	Макс. единиц	Стандартная ставка	Ставка сверхурочных
1		Программист 1	Трудовой		ПР1	Программи	100%	70,00 Р/ч	100,00 Р/ч
2		Программист 2	Трудовой		ПР2	Программи	100%	80,00 Р/ч	110,00 Р/ч
3		Тестировщик	Трудовой		ТСТ	Тестировщ	100%	60,00 Р/ч	90,00 Р/ч

Рисунок 13 – Список ресурсов в Microsoft Project

После выделения требуемых ресурсов для проекта требует провести их распределении по производственным задачам. Процесс распределения ресурсов в соответствии с задачами представлен в таблице 5.

Таблица 5 – Назначение ресурсов

Название	Ресурсы	Единицы(Затраты)
Сбор информации о предметной области	Программист 2	100
	Программист 1	100
Изучение и анализ информации о предметной области	Программист 1	100
Изучение и анализ программных аналогов	Программист 2	100
Формирование требований к проекту	Программист 2	100
Уточнение требований к проекту	Программист 1	100
	Программист 2	
Проектирование общей архитектуры проекта	Программист 1	100
	Программист 2	100
Анализ возможных технологий	Программист 1	100
Выбор технологий для реализации	Программист 1	100
Анализ возможного инструментария разработки	Программист 2	100
Выбор инструментария разработки	Программист 2	100
Проектирование архитектуры компонентов	Программист 1	100
	Программист 2	100
Реализация системы генерации игровых уровней	Программист 1	50
Реализация геймплейных элементов	Программист 1	50
	Программист 2	50
Реализация пользовательского интерфейса	Программист 2	50
Реализация готовой игровой сборки	Программист 2	
Тестирование системы генерации игровых уровней	Тестировщик	50
Тестирование геймплейных элементов	Тестировщик	100
Тестирование пользовательского интерфейса	Тестировщик	50
Отладка работы системы	Программист 1	100
	Программист 2	100
	Тестировщик	100
Исправление ошибок	Программист 1	100
	Программист 2	100
Тестирование системы в целом	Программист 1	100
	Программист 2	100
	Тестировщик	100

В качестве наглядного представления составленного плана можно использовать сетевой график. Самое важное, что Microsoft Project предоставляет возможность отобразить на сетевом графике критический путь [11]. Созданный в Microsoft Project критический путь представлен на рисунке 14.

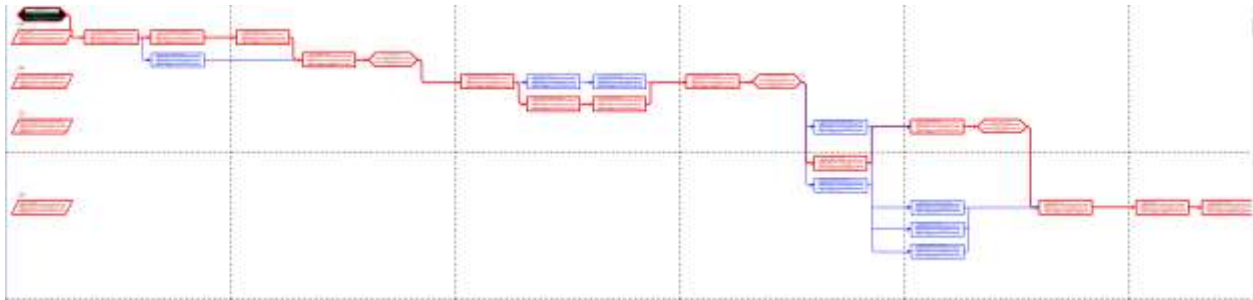


Рисунок 14 – Сетевой график

Наиболее наглядно задачи и распределенные на них ресурсы можно отразить с помощью диаграммы Ганта, которая представлена на рисунке 15.

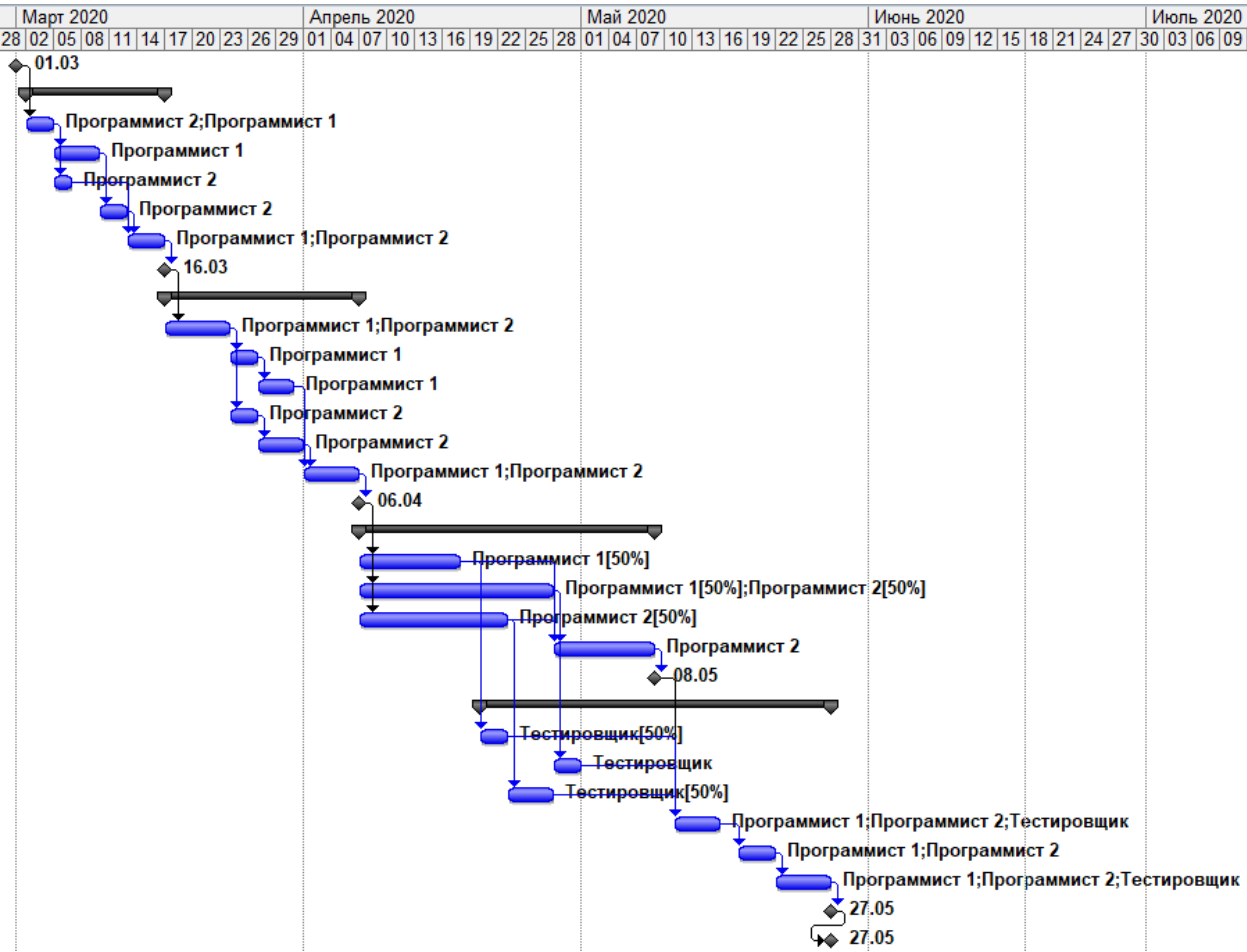


Рисунок 15 – Диаграмма Ганта

В случае трудовых ресурсов важным моментом станет определение их загрузки. Графики загрузки программиста 1, программиста 2 и тестировщика представлены ниже соответственно. Под каждый из них присутствует вывод по изображенному графику. На рисунке 16 график «Программист 1».

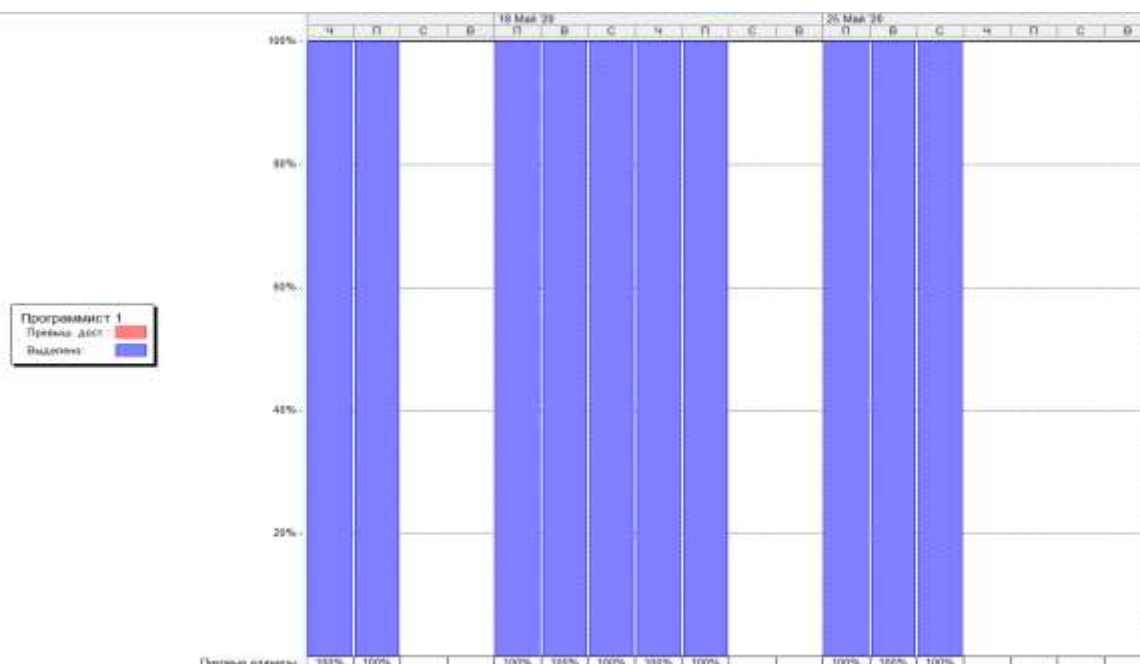


Рисунок 16 – Загруженность ресурс «Программист 1»

Исходя из изображенного графика загрузки «Программиста 1» можно сделать вывод, об отсутствии перегруженности ресурса, из чего следует, равномерная его нагрузка. Далее на рисунке 17 представлен «Программист 2».

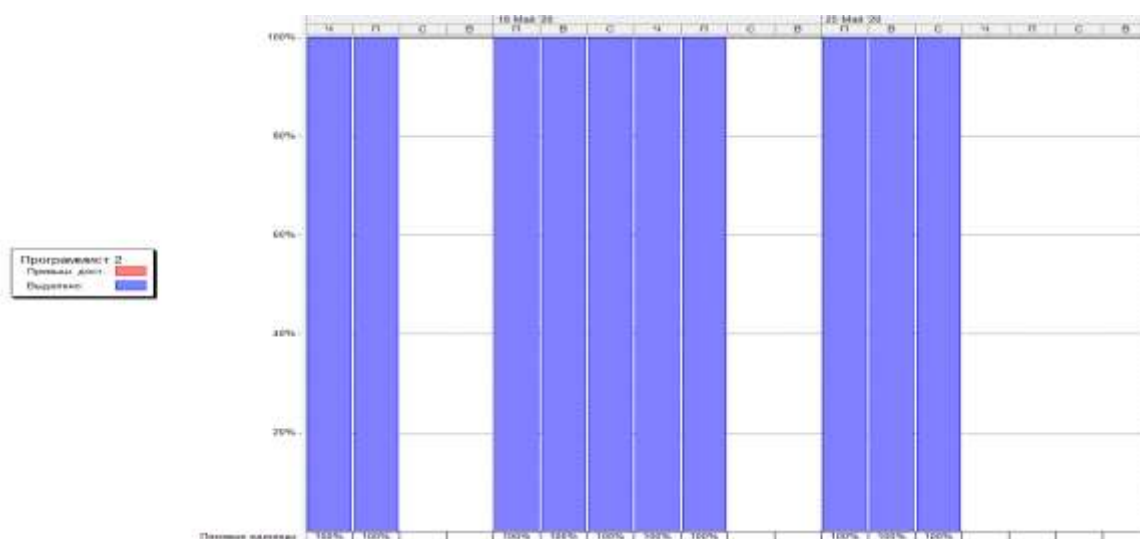


Рисунок 17 – Загруженность ресурс «Программист 2»

На основе графика загрузки ресурса «Программист 2», делается вывод о его равномерной загрузке на протяжении всего проекта. Ниже представлен ресурс «Тестировщик» на рисунке 18.

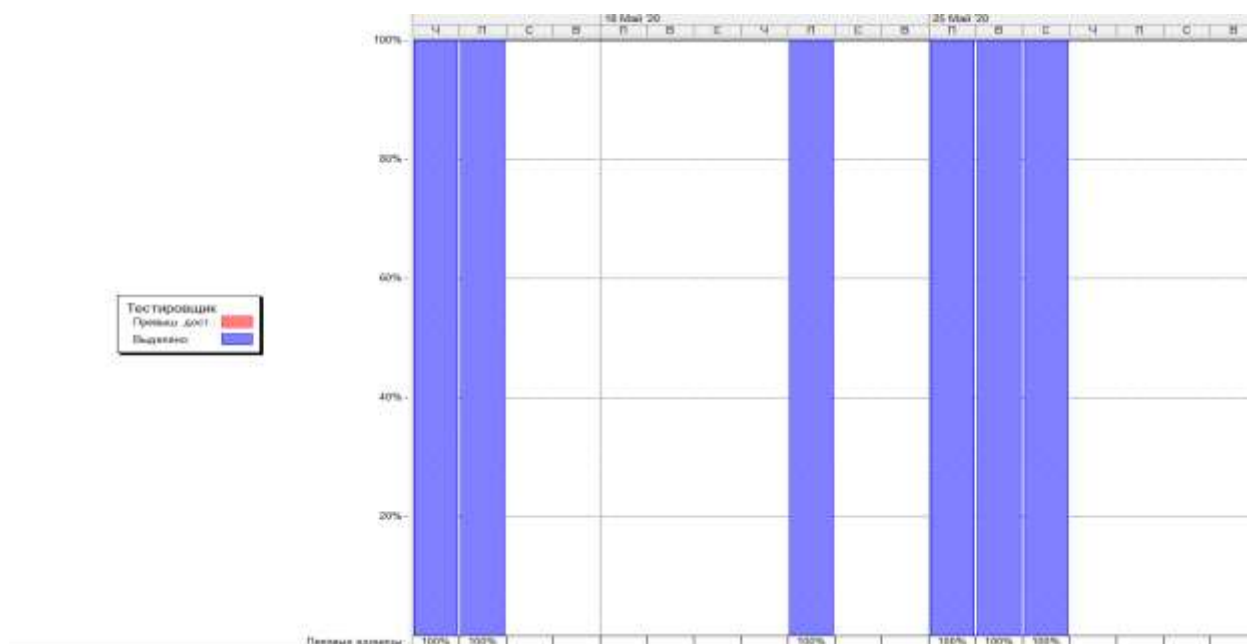


Рисунок 18 – Загруженность ресурс «Тестировщик»

Как видно из графика, ресурс «Тестировщик» использовался в начале и в конце разработки, перегрузки при этом не наблюдается.

Не мало важно особенностью Microsoft Project является возможность создания различных отчетов. Вызываемыми наибольший интерес являются отчеты по срокам выполнения и бюджету проекта. Отчет по проекту предназначен для того, чтобы отображать прогресс и общую длительность проекта. Отчеты по срокам выполнения и бюджету проекта приведены на рисунке 19.

Разработка игры			
на дату Чт 18.06.20			
Даты			
Начало:	Вт 01.03.20	Окончание:	Ср 27.06.20
Баз. начало:	НД	Баз. окончание:	НД
Факт. начало:	НД	Факт. окончание:	НД
Отклон. начало:	0 дней	Отклон. окончание:	0 дней
Длительность			
Текущий план:	63 дней	Оставшаяся:	63 дней
Базовый план:	0 дней?	Фактическая:	0 дней
Отклонение:	63 дней	Процент завершения:	0%
Трудозатраты			
Текущий план:	980 ч	Оставшиеся:	980 ч
Базовый план:	0 ч	Фактические:	0 ч
Отклонение:	980 ч	Процент завершения:	0%
Затраты			
Текущий план:	72 240,00 Р	Оставшиеся:	72 240,00 Р
Базовый план:	0,00 Р	Фактические:	0,00 Р
Отклонение:	72 240,00 Р		
Состояние задач		Состояние ресурса	
Еще не начатые задачи:	31	Трудовые ресурсы:	3
Выполняющиеся задачи:	0	Трудовые ресурсы с прев. доп:	0
Завершённые задачи:	0	Материальные ресурсы:	0
Итого задач:	31	Итого ресурсов:	3

Рисунок 19 – Отчет по срокам выполнения и бюджету проекта

Делая выводы из представленного отчета по затратам выходит, что стоимость разработки ролевой компьютерной игры с элементами случайной генерации локаций составляет 72 240,00 рублей. Так же из данного отчета следует, что длительность проекта составила 52 дня, выполненных задач 31.

3 Реализация компьютерной игры

Приложение было написано на языке C# с использованием среды разработки Visual Studio 2019 Community совместно с игровым движком Unity версии 2019.3.12f1.

На этапе реализации было решено подробнее рассмотреть реализацию одного из важнейших классов игры класса GameController, который отвечает за множество аспектов игрового процесса, таких как: генерация врагов, переход игрока между уровнями и перенос игровых данных из сохранения в класс PlayerController. Данный класс изображен на рисунке 20.

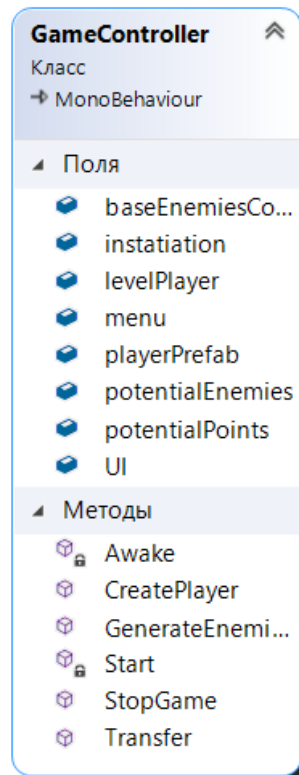


Рисунок 20 – Класс GameController

Рассмотрим подробнее поля данного класса:

- baseEnemiesCount – базовое количество врагов для текущей сцены;
- instatiation – указатель на сам объект GameController;
- levelPlayer – хранит уровень игрока;
- menu – указатель на объект меню паузы;
- playerPrefab – указатель на шаблон объекта игрока;


```

        potentialPoints.Length)].GetComponent<Transform>(
        ).position, Quaternion.identity);
EnemyControllerAI enemyController =
    enemy.GetComponent<EnemyControllerAI>();
enemyController.target =
    PlayerController.instatiation.transform;
//Определение уровня врага
int enemyLevel = Random.Range(1, levelPlayer);
for (int x = 0; x < enemyLevel; x++)
{
    //Случайное распределение характеристик врага
    switch (Random.Range(1, 4))
    {
    case 1:
        enemyController.maxHealth += 10;
        break;
    case 2:
        enemyController.maxStamina += 10;
        break;
    case 3:
        enemyController.maxMagic += 10;
        break;
    }
}
//Определение опыта получаемого от врагов
enemyController.victoryXP =
    (int)Mathf.Round(enemyController.victoryXP * 0.2f
    * enemyLevel);
    }
}
}
}
}

```

Если рассмотреть код представленный выше можно увидеть частое использование метода `Random.Range()`, который используется в данном случае для определения вероятности события.

Далее рассмотрим еще одну совокупность классов, которая реализует систему сохранений в игре. Они представлены на рисунке 21.

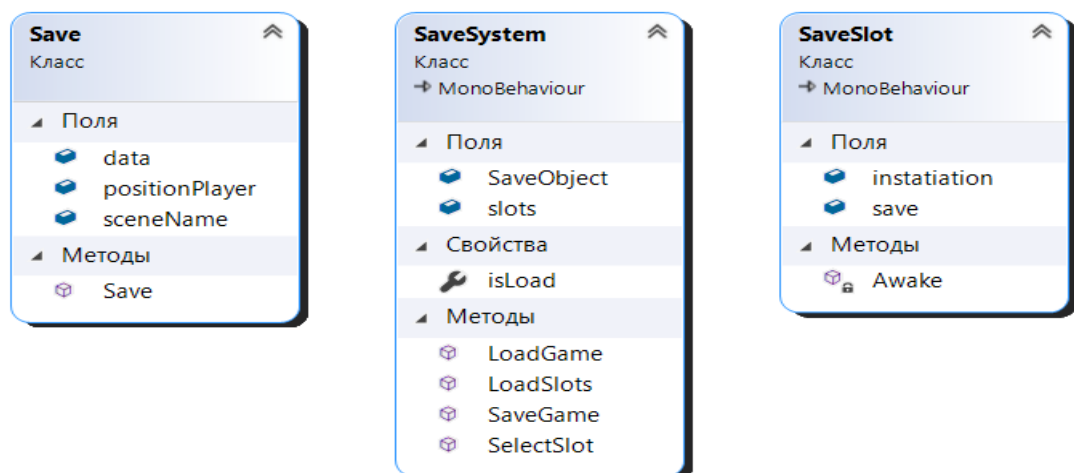


Рисунок 21 – Диаграмма классов системы сохранений

Рассмотрим подробнее каждый из классов, начав с класса `Save`. Для начала рассмотрим его поля и метод:

- `data` – массив хранящий все числовые характеристики игрока, включая текущий уровень, здоровье, магию, выносливость, предметы и т.д.;
- `position` – поле, хранящее значение типа `Vector3`, которое является координатами положения игрока на сцене;
- `sceneName` – строковое поле, которое хранит название сцены, которая была активна на момент сохранения;
- `Save(Vector3 position, params int[] _data)` – конструктор класса, который принимает на вход параметры для полей класса, где `position` – местоположение игрока, `_data` – параметры игрока.

Класс является с сериализуемым, что позволяет использовать его для хранения в файле

Класс `SaveSlot` используется для создания объекта, который может быть использован классом `GameController` для передачи данных игроку. Рассмотрим подробнее поля и методы класса:

- `Instatiation` – статичное поле, указывающее на объект, которое будет использовано `GameController` для обнаружении существования данного объект;
- `save` – поле, хранящее объект типа `Save`;
- `Awake()` – метод, который вызывается при появлении объекта на сцене и присваивает значение полю `instatiation`;

Далее рассмотрим класс `SaveSystem`, который использует классы `Save` и `SaveSlot` для хранения и загрузки данных, кроме того, данный класс отвечает за меню загрузки и сохранений. Рассмотрим поля и методы данного класса:

- `SaveObject` – поле, которое хранит в себе шаблон объекта с компонентом `SaveSlot`;
- `slots` – массив объектов типа `GameObject`, который хранит в себе все объекты, которые представлены в виде слотов в меню загрузки и сохранения;

- `isLoad()` – свойств, которое при значении `true` включает загрузки сохранений, при `false` – создание сохранения;
- `LoadSlots()` – метод, который просматривает директорию игры на предмет наличия сохраненных игр и видоизменяет слоты сохранения при наличии сохранения;
- `SelectSlot(int slot)` – метод, который вызывается при нажатии на слот сохранения в меню загрузки и сохранения, где `slot` определяет номер слота сохранения, кроме этого в зависимости от свойства `isLoad()` вызывает методы `SaveGame()` и `LoadGame()`;
- `SaveGame(int slot)` – метод, который сохраняет данные игрока, название в файл, с использованием преобразования данных в формат `json`;
- `LoadGame(int slot)` – метод, который загружает данные игрока из файла, посредством обратного преобразования из формата `json` в класс `Save` и далее передает его в созданный объект с компонентом `SaveSlot`.

Далее разберем подробнее метод `SaveGame()`, реализация которого описана в листинге 2.

Листинг 2 – Метод `SaveGame()`

```
public void SaveGame(int slot)
{
    //Получение данных игрока
    Save save = PlayerController.instatiation.SavePlayerData();
    //Получение имени сцены
    save.sceneName = SceneManager.GetActiveScene().name;
    //Перевод данных в строку
    string obj = JsonUtility.ToJson(save);
    //Запись данных в файл
    File.WriteAllText(Application.dataPath +
        "/SaveGame"+slot.ToString()+".json", obj);
}
```

В листинге видно, что для преобразования объекта класса `Save` используется метод `JsonUtility.ToJson()`, который создает строку из публичных полей сериализуемого объекта.

Далее разберем подробнее метод `LoadGame()`, реализация которого описана в листинге 3.

Листинг 3 – Метод LoadGame()

```
public void LoadGame(int slot)
{
    //Чтение из файла данных сохранения и запись в объект Save
    Save save =
        JsonUtility.FromJson<Save>(File.ReadAllText(Application.dataPath +
            "/SaveGame" + slot.ToString() + ".json"));
    //Создание игрового объекта для хранения данных
    Instantiate(SaveObject);
    //Передача данных созданному объекту
    SaveSlot.instatiation.save = save;
    //Загрузка сцены
    SceneManager.LoadScene(save.sceneName, LoadSceneMode.Single);
    //Запрет на уничтожение нового объекта
    DontDestroyOnLoad(SaveSlot.instatiation.gameObject);
}
```

В ходе работы методы LoadGame происходит чтение из файла и обратное преобразование из формата json в объект save, далее создается игровой объект для хранения прочитанных данных и из использования после загрузки сцены.

Сам файл сохранения представляет из себя следующую структуру, представленную в листинге 4.

Листинг 4 – Содержимое файл сохранения

```
{
    "data": [100, 100, 100, 0, 0, 0, 0, 0, 400],
    "positionPlayer": { "x": -0.9862051010131836,
        "y": -4.087704181671143, "z": 0.0 },
    "sceneName": "Town"
}
```

Перечислим содержимое файла:

- «data» – параметры игрока;
- «positionPlayer» – координаты игрока на игровой сцене, на которой было сделано сохранение;
- «sceneName» – название сцены, которая была активной на момент создания сохранения.

Далее необходима описать для данного проекта его компонентную составляющую. На рисунке 22 приведена диаграмма компонентов, отражающая общую структуру исходного кода программы.

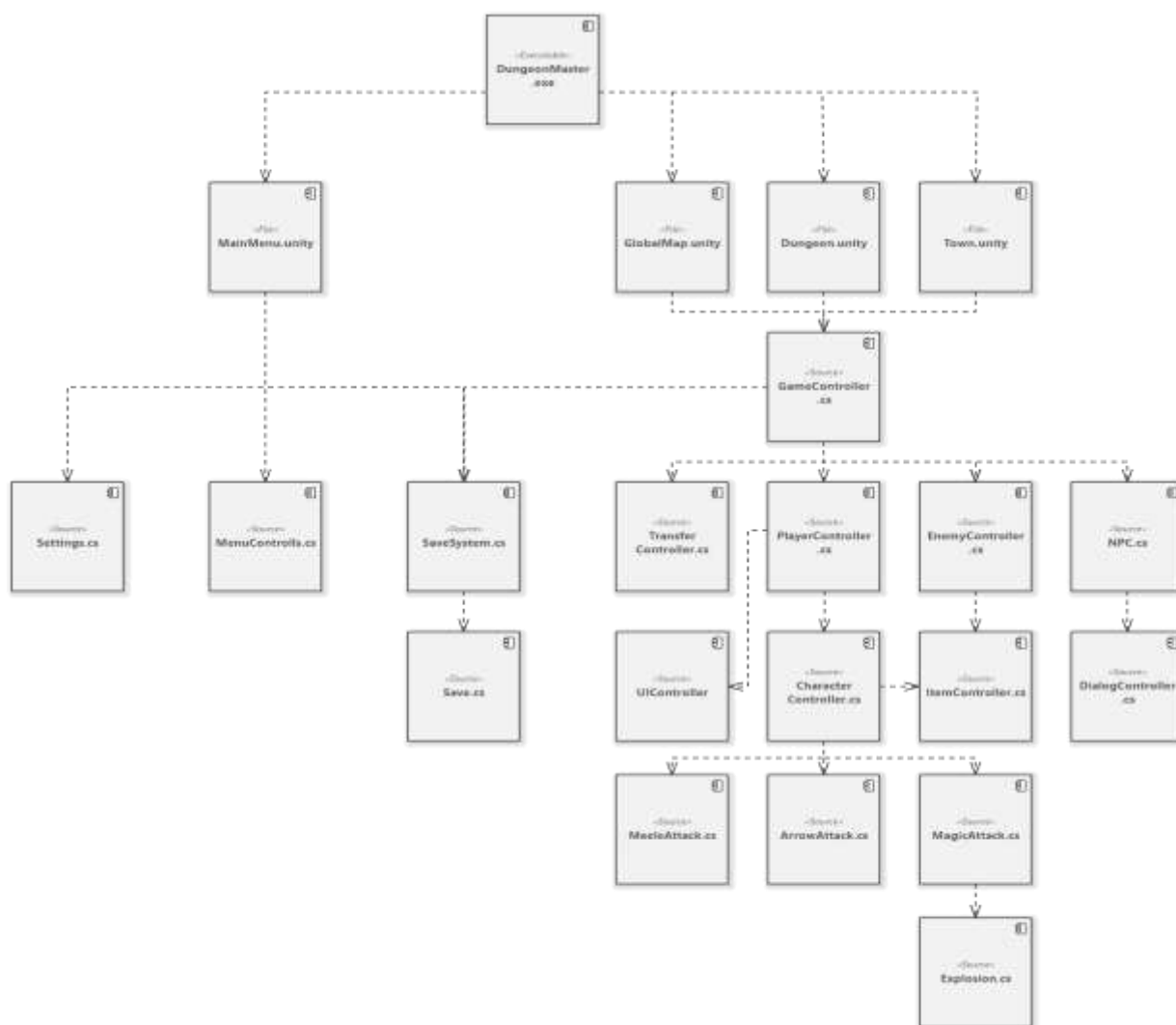


Рисунок 22 – Диаграмма компонентов

В случае данной диаграммы компонентами приложения являются файлы с исходным кодом для используемых в игре скриптов. Исходя из диаграммы можно сделать вывод, что все компоненты являются частями игровых сцен: главное меню «MainMenu», глобальной карты «GlobalMap», города «Town» и подземелья «Dungeon».

4 Тестирование компьютерной игры

Для каждого программного обеспечения важной частью является тестирование ПО, важным оно является не только для поиска возможных ошибок, но и для того, чтобы установить соответствует ли продукт заявленным требованиям.

4.1 Функциональное тестирование

Функциональное тестирование – это тестирование ПО необходимое для проверки того, что оно реализует функционал, заявленный на этапе формирования функциональных требований. Для этого необходимо провести ряд тестов, напрямую связанных с заявленным функционалом системы.

На таблице 6 отражено функциональное тестирование компьютерной ролевой игры.

Таблица 6 – Функциональные тесты

Название	Описание	Результат	Успешность
Новая игра	Игрок выбирает в главном меню начать новую игру	Запуск игрового уровня с персонажем с характеристиками по умолчанию	Успех (рисунок Г.1, рисунок Г.2)
Загрузить игру	Игрок выбирает в главном меню начать новую игру, а затем выбирает слот сохранения	Игра загружена, данные персонажа соответствуют игровому слоту	Успех (рисунок Г.3, рисунок Г.4)
Сохранить игру	Игрок выбирает в меню паузы сохранить игру в выбранный слот	Игра сохранена, данные записаны	Успех (рисунок Г.5, рисунок Г.6)
Включение оконного режима игры	Игрок выбирает в настройках оконный режим	Игра переключилась в оконный режим	Успех (рисунок Г.7)
Изменение громкости музыки	Игрок передвигает ползунок громкости музыки в настройках на центральное значение	Громкость музыки уменьшилась до половинного значения	Успех (рисунок Г.8)
Перемещение по игровой локации	Игрок перемещает персонажа с использованием клавиш WASD	Персонаж перемещен	Успех (рисунок Г.9, рисунок Г.10)

Продолжение таблицы 6

Название	Описание	Результат	Успешность
Использование атаки ближнего боя	Игрок использует атаку ближнего боя по нажатию клавиши X	Атака ближнего боя отрисована перед персонажем	Успех (рисунок Г.11)
Использование атаки дальнего боя	Игрок использует атаку ближнего боя по нажатию клавиши C	Атака дальнего боя отрисована перед персонажем	Успех (рисунок Г.12)
Использование магической атаки	Игрок использует атаку ближнего боя по нажатию клавиши V	Магическая атака отрисована в двух фазах	Успех (рисунок Г.13, рисунок Г.14)
Получение урона от атаки противника	Игрок находится рядом с противником и получает от него урон в ближнем бою.	Атака отрисована, количество здоровья игрока уменьшилось	Успех (рисунок Г.15, рисунок Г.16)
Генерация противников	При входе на игровую локацию создается несколько врагов	На игровой локации создано несколько противников	Успех (рисунок Г.17)
Игрок подбирает монеты	После победы над врагом игрок подбирает выпавшие монеты	Количество монет у игрока возросло	Успех (рисунок Г.18, рисунок Г.19)
Игрок взаимодействует с игровым персонажем	Игрок начинает диалог с персонажем при нажатии рядом с ним E	Диалог успешно начался	Успех (рисунок Г.20, рисунок Г.21, рисунок Г.22, рисунок Г.23)
Игрок покупает зелье здоровья	В ходе диалога с персонажем игрок покупает зелье здоровья за 100 монет	Количество монет уменьшилось на 100, количество зелий здоровья возросло на 1.	Успех (рисунок Г.24, рисунок Г.25)
Игрок улучшает параметр здоровья	В ходе диалога с игровым персонажем, игрок выбирает увеличить характеристику здоровья на 10 за 100 монет и 10 очков уровня	Количество монет уменьшилось на 100, количество очков уровня уменьшилось на 10, показатель здоровья увеличился на 10	Успех (рисунок Г.26, рисунок Г.27)
Игрок перемещается между локациями	Игрок выходит из локации города на локацию глобальной карты	Изменилась игровая сцена параметры игрока остались в тех же значениях.	Успех (рисунок Г.28, рисунок Г.29)
Вызов паузы	Игрок вызывает меню паузы нажатием клавиши ESC	На экране появилось меню паузы, игровой процесс приостановлен	Успех (рисунок Г.30)
Выпадение предметов из врагов	Игрок уничтожает врагов, из которых выпадают предметы	Из побежденных врагов выпали предметы.	Успех (рисунок Г.31, рисунок Г.32)

После проведения функционального тестирования можно констатировать, что приложение выполняет все заявленные функции корректно.

Заключение

В результате выполнения выпускной квалификационной работы была разработана компьютерная ролевая игра с элементами случайной генерации локаций.

На этапе анализа предметной области выявлены функциональные и нефункциональные требования к игре, а также рассмотрены ее налоги на рынке.

На этапе проектирования был произведен выбор технологии инструментальных средств. Кроме этого, были спроектированы ролевая система, интерфейсы, архитектура приложения, а также алгоритмы его работы. Далее был составлен план разработки программного обеспечения, сформирована последовательность работ, рассчитан бюджет проекта и его продолжительность. Для реализации необходимо 72 240 руб., а разработка заняла 52 дня.

На этапе реализации было создано приложение, написанное на языке C# в среде программирования Visual Studio 2019 Community с использованием движка Unity 2019.03.12f1.

Было проведена проверка качества разработанного программного обеспечения. Корректность и отказоустойчивость проверена с помощью функционального тестирования.

Список использованных источников

1. Компьютерная ролевая игра [Электронный ресурс] URL:https://ru.wikipedia.org/wiki/Компьютерная_ролевая_игра (дата обращения 26.04.2020).
2. Анализ предметной области и постановка задачи[Электронный ресурс] URL: <https://studfile.net/preview/4673149/page:2/> (дата обращения 26.04.2020).
3. М. Фаулер, С. А. UML Основы. / Пер. с англ. – М.: издание Питер, 2005. - 180 с.
4. К. Вигерс и Дж. Битти. Разработка требований к программному обеспечению / Пер. с англ. – М.: издание Русская редакция, 2014. - 730 с.
5. Гагарина, Л.Г. Технология разработки программного обеспечения / Л.Г. Гагарина, Е.В. Кокорева, Б.Д. Виснадул. – М.:Форум, ИНФРА-М, 2008. – 400 с.
6. Статья о методологии RUP // Информационный ресурс [Электронный ресурс]. URL: <http://www.informicus.ru/default.aspx?SECTION=6&id=73&subdivisionid=7> (дата обращения: 18.05.2020).
7. Материалы с сайта «Unity» // Официальный сайт «Unity» [Электронный ресурс]. URL: <https://unity3d.com/ru> (дата обращения 26.04.2020).
8. Статья о методологии RUP // Информационный ресурс [Электронный ресурс]. URL: <http://www.informicus.ru/default.aspx?SECTION=6&id=73&subdivisionid=7> (дата обращения: 18.05.2020).
9. Общее описание движка "Unity" на сайте Wikipedia // Электронная энциклопедия [Электронный ресурс]. URL: [https://ru.wikipedia.org/wiki/Unity_\(игровой_движок\)](https://ru.wikipedia.org/wiki/Unity_(игровой_движок)) (дата обращения 26.04.2020).
10. Абдрафиков, М.А. Управление программными проектами: теория и практика: учебное пособие / М.А. Абдрафиков, В.Е. Гвоздев, Р.Ф. Маликов [и др.]. – Электрон. дан. – БГПУ имени М. Акмуллы (Башкирский

государственный педагогический университет им.М. Акмуллы), 2015. — 128 с.

11. Джонсон Т., Четфилд К. Microsoft Project 2016. Шаг за шагом. М., Изд-во КТК Галактика, 2018. – 566 с.

Приложение А
Глоссарий
(справочный)

RUP (Rational Unified Process) – методология разработки программного обеспечения, основанная на использовании итеративной модели разработки ПО.

UML (Unified Modeling Language) – язык графического описания, используемый для объектного моделирования в области разработки ПО, проектировании систем и отображения организационных структур.

Windows – операционная система для настольных компьютеров, разработанная компанией Windows.

Скрипт – класс-наследник от MonoBehaviour, который является частью игровых компонентов игровых объектов в Unity для описания логики их поведения.

Сцена – пространство в движке Unity на котором расположены игровые объекты.

Префаб – шаблон объектов в Unity, который позволяет многократное использование данных объектов во многих сценах с заранее определенными компонентами и их параметрами.

Спрайт – графический объект, применяемый в компьютерной графике. Представляет собой совокупность нескольких картинок в одном файле.

Приложение Б
Исходный код
(рекомендуемое)

Листинг Б.1 – Файл CharacterController.cs

```

using System.Collections;
using System.Collections.Generic;
using UnityEngine;
public abstract class CharacterController : MonoBehaviour
{
    //Характеристики
    public int maxHealth = 100;
    public int maxStamina = 100;
    public int maxMagic = 100;
    public int baseAttack = 20;
    public int level = 1;
    public int maxXP = 100;
    public int currentXP = 0;
    public int currentLevelPoints = 0;
    public int levelPointsPerLevel = 10;
    //Физические свойства
    public float speed = 4.0f;
    //Интерфейс
    public GameObject meleeAttackPrefab;
    public GameObject arrowAttackPrefab;
    public GameObject magicAttackPrefab;
    public Vector2 lookDirection = new Vector2(0, -1);
    public Rigidbody2D rigidbody2d;
    //Свойства для характеристик
    public int health { get; protected set; }
    public int magic { get; protected set; }
    public int stamina { get; protected set; }
    public void ChangeHealth(int amount)
    {
        health = Mathf.Clamp(health + amount, 0, maxHealth);
        if (health == 0)
        {
            Destroyed();
        }
    }
    public void ChangeStamina(int amount)
    {
        health = Mathf.Clamp(stamina + amount, 0, maxHealth);
    }
    public void ChangeMagic(int amount)
    {
        health = Mathf.Clamp(magic + amount, 0, maxHealth);
    }
    public void ChangeXP(int amount)
    {
        currentXP += amount;
        if (currentXP >= maxXP)
        {
            currentXP -= maxXP;
        }
    }
}

```

```

        maxXP = maxXP * 2;
        level += 1;
        currentLevelPoints += levelPointsPerLevel;
    }
}

public void MeleeAttack()
{
    GameObject meleeAttackObject =
Instantiate(meleeAttackPrefab, rigidbody2d.position +
lookDirection * 0.8f, Quaternion.identity);
    MeleeAttack meleeAttack =
meleeAttackObject.GetComponent<MeleeAttack>();
    meleeAttack.Attack(lookDirection, 200f, this);

}

public void ArrowAttack()
{
    GameObject arrowAttackObject =
Instantiate(arrowAttackPrefab, rigidbody2d.position +
lookDirection * 0.8f, Quaternion.identity);
    ArrowAttack meleeAttack =
arrowAttackObject.GetComponent<ArrowAttack>();
    meleeAttack.Attack(lookDirection, 200f, this);
}

public void MagicAttack()
{
    GameObject magicAttackObject =
Instantiate(magicAttackPrefab, rigidbody2d.position +
lookDirection * 0.8f, Quaternion.identity);
    MagicAttack magicAttack =
magicAttackObject.GetComponent<MagicAttack>();
    magicAttack.Attack(lookDirection, 200f, this);
}

public virtual void Destroyed()
{
    Destroy(this.gameObject);
}
}

```

Листинг Б.2 – Файл PlayerController.cs

```

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

public class PlayerController : CharacterController
{
    float horizontal;
    float timer = 1f;

```

```

float vertical;
AudioSource audioSource;
Animator animator;
public static PlayerController instatiation;
public AudioClip ProjectileAudioClip;
public ParticleSystem healtheffect;
public ParticleSystem attackeffect;
public int potionHP { get; protected set; }
public int potionST { get; protected set; }
public int potionMP { get; protected set; }
public int gold { get; protected set; }
private void Awake()
{
    instatiation = this;
}
void Start()
{
    healtheffect.Stop();
    attackeffect.Stop();
    rigidbody2d = GetComponent<Rigidbody2D>();
    animator = GetComponent<Animator>();
    gold = 400;
    potionHP = 0;
    potionST = 0;
    potionMP = 0;
    health = maxHealth;
    stamina = maxStamina;
    magic = maxMagic;
    audioSource = GetComponent<AudioSource>();
    UIController.instance.ShowValues(
        new List<int> { health, stamina, magic, currentXP,
maxXP, maxMagic, maxStamina, maxHealth },
        new List<int> { potionHP, potionST, potionMP, gold,
level, currentLevelPoints });
}
public void PlaySound(AudioClip clip)
{
    audioSource.PlayOneShot(clip);
}
// Update is called once per frame
void Update()
{
    timer -= Time.deltaTime;
    if(timer <0)
    {
        health = Mathf.Clamp(health +
(int)Math.Round(maxHealth * 0.02), 0, maxHealth);
        stamina = Mathf.Clamp(stamina +
(int)Math.Round(maxStamina * 0.05), 0, maxStamina);
        magic = Mathf.Clamp(magic + (int)Math.Round(maxMagic
* 0.04), 0, maxMagic);
        timer = 1f;
    }
}

```

```

        UIController.instance.ShowValues(
            new List<int> { health, stamina, magic, currentXP,
maxXP, maxMagic, maxStamina, maxHealth },
            new List<int> { potionHP, potionST, potionMP, gold,
level,currentLevelPoints });

        horizontal = Input.GetAxis("Horizontal");
        vertical = Input.GetAxis("Vertical");

        Vector2 move = new Vector2(horizontal, vertical);

        if (!Mathf.Approximately(move.x, 0.0f) ||
!Mathf.Approximately(move.y, 0.0f))
        {
            lookDirection.Set(move.x, move.y);
            lookDirection.Normalize();
        }
        animator.SetFloat("Move X", lookDirection.x);
        animator.SetFloat("Move Y", lookDirection.y);
        animator.SetFloat("Speed", move.magnitude);
        if (Input.GetKeyDown(KeyCode.X))
        {
            if (stamina>=10)
            {
                MeleeAttack();
                stamina -= 10;
            }
        }
        if (Input.GetKeyDown(KeyCode.C))
        {
            if (stamina >= 10)
            {
                ArrowAttack();
                stamina -= 10;
            }
        }
        if (Input.GetKeyDown(KeyCode.V))
        {
            if (magic >= 10)
            {
                MagicAttack();
                magic -= 10;
            }
        }
        if (Input.GetKeyDown(KeyCode.R))
        {
            if (potionHP>0) {
                health =Mathf.Clamp(health +
(int)Math.Round(maxHealth * 0.3), 0, maxHealth);
                potionHP -= 1;
            }
        }
        if (Input.GetKeyDown(KeyCode.T))

```

```

        {
            if (potionMP > 0) {
                stamina = Mathf.Clamp(stamina +
(int)Math.Round(maxStamina * 0.3), 0, maxStamina);
                potionMP -= 1;
            }
        }
        if (Input.GetKeyDown(KeyCode.Y))
        {
            if (potionST > 0) {
                magic = Mathf.Clamp(magic +
(int)Math.Round(maxMagic * 0.3), 0, magic);
                potionST -= 1;
            }
        }
        if (Input.GetKeyDown(KeyCode.E))
        {
            RaycastHit2D hit =
Physics2D.Raycast(rigidbody2d.position + Vector2.up * 0.2f,
lookDirection,
1.5f, LayerMask.GetMask("Exit"));
            if (hit.collider != null)
            {

GameController.instatiation.Transfer(this, hit.collider.gameObjec
t.GetComponent<TransferController>());
            }
            hit = Physics2D.Raycast(rigidbody2d.position +
Vector2.up * 0.2f, lookDirection,
1.5f, LayerMask.GetMask("NPC"));
            if (hit.collider != null)
            {

hit.collider.gameObject.GetComponent<NPC>().ShowDialog();
            }
        }
        if (Input.GetKeyDown(KeyCode.Escape))
        {
            GameController.instatiation.StopGame();
        }
    }
    void FixedUpdate()
    {
        Vector2 position = rigidbody2d.position;
        position.x = position.x + speed * horizontal *
Time.deltaTime;
        position.y = position.y + speed * vertical *
Time.deltaTime;
        rigidbody2d.MovePosition(position);
    }

    public void GetObject(int amount, int type)
    {

```

```

switch (type)
{
    case 0:
        potionHP += amount;
        break;
    case 1:
        potionST += amount;
        break;
    case 2:
        potionMP += amount;
        break;
    case 3:
        gold += amount;
        break;
}
}
public override void Destroyed()
{
    Debug.Log("Мертв");
}
public void LoadPlayerData(Save data)
{
    maxHealth = data.data[0];
    maxStamina = data.data[1];
    maxMagic = data.data[2];
    currentXP = data.data[3];
    currentLevelPoints = data.data[4];
    potionHP = data.data[5];
    potionST = data.data[6];
    potionMP = data.data[7];
    gold = data.data[8];
    gameObject.GetComponent<Transform>().position =
data.positionPlayer;
}
public Save SavePlayerData()
{
    return new Save(
gameObject.GetComponent<Transform>().position,maxHealth,maxStamina,maxMagic,currentXP,currentLevelPoints,potionHP,potionST,potionMP,gold);
}
}

```

Листинг Б.3 – Файл EnemyControllerAI.cs

```

using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using Pathfinding;
using UnityEngine.SocialPlatforms;

public class EnemyControllerAI : CharacterController

```

```

{
    float timeBetweenAttackNominal = 1.0f;
    float timeBetweenAttack;
    bool canAttack = true;
    public Transform target;
    public float acceleration = 200f;
    public float nextWayPointDistance = 3f;
    public List<GameObject> possibleLoot;
    Path path;
    int currentWayPoint;
    bool reachEndOfPath = false;
    Seeker seeker;
    Animator animator;
    public int victoryXP=20;
    void Start()
    {
        seeker = GetComponent<Seeker>();
        //target =
GameObject.Find("Player").GetComponent<Transform>();
        rigidbody2d = GetComponent<Rigidbody2D>();
        animator = GetComponent<Animator>();
        health = maxHealth;
        stamina = maxStamina;
        magic = maxMagic;
        InvokeRepeating("UpdatePath", 0f, 0.5f);
    }
    void UpdatePath()
    {
        if(seeker.IsDone())
            seeker.StartPath(rigidbody2d.position,
target.position, OnPathComplete);
    }
    void OnPathComplete(Path p)
    {
        if (!p.error)
        {
            path = p;
            currentWayPoint = 0;
        }
    }
    private void Update()
    {
        if (Vector2.Distance(target.transform.position,
transform.position) < 2)
        {
            if (canAttack)
            {
                MeleeAttack();
                timeBetweenAttack = timeBetweenAttackNominal;
                canAttack = false;
            }
            else
            {

```

```

        timeBetweenAttack -= Time.deltaTime;
        if (timeBetweenAttack < 0)
        {
            canAttack = true;
        }
    }
}

void FixedUpdate()
{
    if (path == null)
        return;
    if (currentWayPoint >= path.vectorPath.Count)
    {
        reachEndOfPath = true;
        return;
    }
    else
    {
        reachEndOfPath = false;
    }
    Vector2 direction =
    ((Vector2)path.vectorPath[currentWayPoint] -
    rigidbody2d.position).normalized;
    if (!Mathf.Approximately(direction.x, 0.0f) ||
    !Mathf.Approximately(direction.y, 0.0f))
    {
        lookDirection.Set(direction.x, direction.y);
        lookDirection.Normalize();
    }
    animator.SetFloat("Move X", lookDirection.x);
    animator.SetFloat("Move Y", lookDirection.y);
    animator.SetFloat("Speed", direction.magnitude);
    Vector2 force = direction * acceleration *
    Time.deltaTime;
    rigidbody2d.AddForce(force);
    float distance = Vector2.Distance(rigidbody2d.position,
    path.vectorPath[currentWayPoint]);
    if (distance < nextWayPointDistance)
    {
        currentWayPoint++;
    }
}

public override void Destroyed()
{
    target.GetComponent<PlayerController>().ChangeXP(victoryXP);
    {
        for (int i = 0; i < 2; i++)
        {
            if(Random.Range(1, 11) <= 2)
            {

```

```

        GameObject loot =
Instantiate(possibleLoot[i], rigidbody2d.position,
Quaternion.identity);
        loot.GetComponent<ItemController>().type =
i;
        loot.GetComponent<ItemController>().amount =
1;

    }
}
if (Random.Range(1, 11) <= 5)
{
    GameObject loot = Instantiate(possibleLoot[3],
rigidbody2d.position, Quaternion.identity);
    loot.GetComponent<ItemController>().type = 3;
    loot.GetComponent<ItemController>().amount =
Random.Range(20, 101)*level;
}
}
Destroy(this.gameObject);
}
}

```

Листинг Б.4 – Файл SaveSystem.cs

```

using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using System.IO;
using System;
using UnityEngine.UIElements;
using UnityEngine.UI;
using UnityEngine.SceneManagement;

public class SaveSystem : MonoBehaviour
{
    public GameObject[] slots;
    public GameObject SaveObject;
    public bool isLoad { get; set; }
    public void LoadSlots()
    {
        for (int i = 0; i < slots.Length; i++)
        {
            if (File.Exists(Application.dataPath + "/SaveGame" +
i.ToString()+".json"))
            {
                slots[i].GetComponentInChildren<Text>().text =
"Save "+i.ToString();
            }
        }
    }
    public void SelectSlot(int slot)

```

```

    {
        if (isLoad)
        {
            if (File.Exists(Application.dataPath + "/SaveGame" +
slot.ToString() + ".json"))
            {
                LoadGame(slot);
            }
        }
        else
        {
            SaveGame(slot);
            LoadSlots();
        }
    }
    public void SaveGame(int slot)
    {
        Save save =
PlayerController.instatiation.SavePlayerData();
        save.sceneName = SceneManager.GetActiveScene().name;
        string obj = JsonUtility.ToJson(save);
        File.WriteAllText(Application.dataPath +
            "/SaveGame"+slot.ToString()+".json", obj);
    }
    public void LoadGame(int slot)
    {
        Save save =
JsonUtility.FromJson<Save>(File.ReadAllText(Application.dataPath
+ "/SaveGame" + slot.ToString() + ".json"));
        Time.timeScale = 1f;
        Instantiate(SaveObject);
        SaveSlot.instatiation.save = save;
        SceneManager.LoadScene(save.sceneName,
LoadSceneMode.Single);
        DontDestroyOnLoad(SaveSlot.instatiation.gameObject);
        //PlayerController.instatiation.LoadPlayerData(load);
    }
}
[Serializable]
public class Save
{
    public int[] data;
    public Vector3 positionPlayer;
    public string sceneName;
    public Save( Vector3 position, params int[] _data )
    {
        data = _data;
        positionPlayer = position;
    }
}

```

Листинг Б.5 – Файл SaveSlot.cs

```

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

public class SaveSlot : MonoBehaviour
{
    public Save save;
    public static SaveSlot instatiation;
    private void Awake()
    {
        instatiation = this;
    }
}

```

Листинг Б.6 – Файл GameController.cs

```

using Cinemachine;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;
public class GameController : MonoBehaviour
{
    public List<GameObject> potentialEnemies;
    public int baseEnemiesCount;
    public GameObject[] potentialPoints;
    public int levelPlayer;
    public GameObject UI;
    public GameObject playerPrefab;
    public static GameController instatiation;
    private bool gameIsStop;
    public GameObject menu;
    private void Awake()
    {
        instatiation = this;
        Instantiate(UI);
        menu = Instantiate(menu);
    }
    private void Start()
    {
        CreatePlayer();
        potentialPoints =
GameObject.FindGameObjectsWithTag("Respawn");
        levelPlayer = PlayerController.instatiation.level;
        GenerateEnemies();
    }
    public void GenerateEnemies()
    {
        for (int i = 0; i < baseEnemiesCount +
Mathf.Log(levelPlayer,2); i++)
        {
            if (Random.Range(1,3)==1) {

```

```

        GameObject enemy =
Instantiate(potentialEnemies[Random.Range(0,
potentialEnemies.Count)], potentialPoints[Random.Range(0,
potentialPoints.Length)].GetComponent<Transform>().position,
Quaternion.identity);
        EnemyControllerAI enemyController =
enemy.GetComponent<EnemyControllerAI>();
        enemyController.target =
PlayerController.instatiation.transform;
        int enemyLevel = Random.Range(1, levelPlayer);
        for (int x = 0; x < enemyLevel; x++)
        {
            switch (Random.Range(1, 4))
            {
                case 1:
                    enemyController.maxHealth += 10;
                    break;
                case 2:
                    enemyController.maxStamina += 10;
                    break;
                case 3:
                    enemyController.maxMagic += 10;
                    break;
            }
        }
        enemyController.victoryXP =
(int)Mathf.Round(enemyController.victoryXP * 0.2f*enemyLevel);
    }
}

public void Transfer(PlayerController
player, TransferController trans)
{
    player.gameObject.GetComponent<Transform>().position =
trans.position.position;
    SceneManager.LoadScene(trans.sceneName,
LoadSceneMode.Single);
    DontDestroyOnLoad(player.gameObject);
}

public void CreatePlayer()
{
    if (PlayerController.instatiation==null)
    {
        Instantiate(playerPrefab);
    }
    if (SaveSlot.instatiation != null)
    {
        Debug.Log("Объект есть");
    }

    PlayerController.instatiation.LoadPlayerData(SaveSlot.instatiation.save);
    Destroy(SaveSlot.instatiation.gameObject);
}

```

```

        else
        {
            Debug.Log("Объекта нет");
        }

GameObject.Find("CameraController").GetComponent<CinemachineVirtualCameraBase>().Follow =
PlayerController.instatiation.gameObject.GetComponent<Transform>().transform;
    }
    public void StopGame()
    {
        menu.SetActive(true);
        Time.timeScale = 0f;
    }
}

```

Листинг Б.7 – Файл MenuControlls.cs

```

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

public class MenuControlls : MonoBehaviour
{
    public void PlayPressed()
    {
        SceneManager.LoadScene("Town");
    }
    public void ExitPressed()
    {
        Application.Quit();
    }
    public void Resume()
    {
        Time.timeScale = 1f;
        transform.parent.gameObject.SetActive(false);
    }
}

using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.Audio;
using UnityEngine.UI;

public class Settings : MonoBehaviour
{
    public AudioManager audioMixer;
    public bool isFullScreen;
}

```

```

Resolution[] rsl;
List<string> resolutions;
public Dropdown resolutionDropDown;
public void Awake()
{
    resolutions = new List<string>();
    rsl = Screen.resolutions;
    foreach (var i in rsl)
    {
        resolutions.Add(i.width + "x" + i.height);
    }
    resolutionDropDown.ClearOptions();
    resolutionDropDown.AddOptions(resolutions);
}
public void FullScreenToggle()
{
    isFullScreen = !isFullScreen;
    Screen.fullScreen = isFullScreen;
}
public void AudioVolume(float sliderValue)
{
    audioMixer.SetFloat("masterVolume",Mathf.Log10(sliderValue)*20);
}
public void Quality(int q)
{
    QualitySettings.SetQualityLevel(q);
}
public void Resolution(int r)
{
    Screen.SetResolution(rsl[r].width, rsl[r].height,
isFullScreen);
}
}

```

Листинг Б.7 – Файл ArrowAttack.cs

```

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

public class ArrowAttack : MonoBehaviour
{
    Rigidbody2D rigidbody2d;
    float currentTimer;
    public float timer = 0.1f;
    private Animator animator;
    public CharacterController assaulter;
    void Awake()
    {
        rigidbody2d = GetComponent<Rigidbody2D>();
        animator = GetComponent<Animator>();
        currentTimer = timer;
    }
}

```

```

    }

    void Update()
    {
        currentTimer -= Time.deltaTime;
        if (currentTimer <= 0)
        {
            Destroy(gameObject);
        }
    }

    public void Attack(Vector2 direction, float force,
        CharacterController _assaulter)
    {
        assaulter = _assaulter;
        rigidbody2d.AddForce(direction * force);
        rigidbody2d.SetRotation(Vector2.SignedAngle(new
        Vector2(0f, -1f), direction));
    }

    private void OnTriggerEnter2D(Collider2D other)
    {
        CharacterController e =
        other.GetComponent<CharacterController>();
        if (e != null && e != assaulter)
        {
            e.ChangeHealth(-15);
            Destroy(gameObject);
        }
    }
}

```

Листинг Б.8 – Файл MeleeAttack.cs

```

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

public class MeleeAttack : MonoBehaviour
{
    Rigidbody2D rigidbody2d;
    float currentTimer;
    public float timer = 0.1f;
    private Animator animator;
    public CharacterController assaulter;
    void Awake()
    {
        rigidbody2d = GetComponent<Rigidbody2D>();
        animator = GetComponent<Animator>();
        currentTimer = timer;
    }

    void Update()

```

```

    {
        currentTimer -= Time.deltaTime;
        if (currentTimer <= 0)
        {
            Destroy(gameObject);
        }
    }

    public void Attack(Vector2 direction, float force,
CharacterController _assaulter)
    {
        assaulter = _assaulter;
        rigidbody2d.AddForce(direction * force);
        animator.SetFloat("Move X", direction.x);
        animator.SetFloat("Move Y", direction.y);
    }

    private void OnTriggerEnter2D(Collider2D other)
    {
        CharacterController e =
other.GetComponent<CharacterController>();
        if (e != null && e != assaulter)
        {
            e.ChangeHealth(-20);
            Destroy(gameObject);
        }
    }
}

```

Листинг Б.9 – Файл MagicAttack.cs

```

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

public class MagicAttack : MonoBehaviour
{
    Rigidbody2D rigidbody2d;
    float currentTimer;
    public float timer;
    private Animator animator;
    public GameObject explosionPrefab;
    public CharacterController assaulter;
    void Awake()
    {
        rigidbody2d = GetComponent<Rigidbody2D>();
        currentTimer = timer;
    }

    void Update()
    {
        currentTimer -= Time.deltaTime;
        if (currentTimer <= 0)
        {

```

```

        Instantiate(explosionPrefab, rigidbody2d.position,
Quaternion.identity);
        Destroy(gameObject);
    }
}
public void Attack(Vector2 direction, float force,
CharacterController _assaulter)
{
    assaulter = _assaulter;
    rigidbody2d.AddForce(direction * force);
    rigidbody2d.SetRotation(Vector2.SignedAngle(new
Vector2(0f, -1f), direction));
}
private void OnTriggerEnter2D(Collider2D other)
{
    CharacterController e =
other.GetComponent<CharacterController>();
    if (e != null && e != assaulter)
    {
        e.ChangeHealth(-15);
        Instantiate(explosionPrefab, rigidbody2d.position,
Quaternion.identity);
        Destroy(gameObject);
    }
}
}
}

```

Листинг Б.10 – Файл NPC.cs

```

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

public class NPC : MonoBehaviour
{
    public GameObject dialog;
    public void ShowDialog()
    {
        if (dialog.activeSelf)
            dialog.SetActive(false);
        else
        {
            dialog.SetActive(true);
        }
    }
}

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

```

Листинг Б.11 – Файл Explosion.cs

```
public class Explosion : MonoBehaviour
{
    float currentTimer;
    public float timer = 0.5f;
    private Animator animator;
    void Awake()
    {
        currentTimer = timer;
    }

    void Update()
    {
        currentTimer -= Time.deltaTime;
        if (currentTimer <= 0)
        {
            Destroy(gameObject);
        }
    }
    private void OnTriggerEnter2D(Collider2D other)
    {
        CharacterController e =
other.GetComponent<CharacterController>();
        if (e != null)
        {
            e.ChangeHealth(-15);
        }
    }
}
```

Листинг Б.12 – Файл DialogController.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
public class DialogController : MonoBehaviour
{
    public void Trade(int opt)
    {
        if (PlayerController.instatiation.gold >= 100)
        {
            PlayerController.instatiation.GetObject(1, opt);
            PlayerController.instatiation.GetObject(-100, 3);
        }
    }
    public void Learn(int opt)
    {
        if (PlayerController.instatiation.gold >= 100 &&
PlayerController.instatiation.currentLevelPoints >=10)
```

```

        {
            switch (opt)
            {
                case 0:
                    PlayerController.instatiation.maxHealth +=
10;
                    break;
                case 1:
                    PlayerController.instatiation.maxStamina +=
10;
                    break;
                case 2:
                    PlayerController.instatiation.maxMagic +=
10;
                    break;
            }
            PlayerController.instatiation.currentLevelPoints -=
10;
            PlayerController.instatiation.GetObject(-100, 3);
        }
    }
}

```

```

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

```

```

public class TransferController : MonoBehaviour
{
    public Transform position;
    public string sceneName;
}

```

Листинг Б.13 – Файл DialogController.cs

```

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

```

```

public class ItemController : MonoBehaviour
{
    public int type;
    public int amount;
    private void OnTriggerEnter2D(Collider2D collision)
    {
        if (collision.GetComponent<PlayerController>() != null)
        {
            collision.GetComponent<PlayerController>().GetObject(amount,
            type);
            Destroy(gameObject);
        }
    }
}

```

```

    }
}

```

Листинг Б.14 – Файл UIController.cs

```

using System.Collections;
using System.Collections.Generic;
using TMPro;
using UnityEngine;
using UnityEngine.UI;

public class UIController : MonoBehaviour
{
    // Start is called before the first frame update
    public static UIController instance { get; private set; }
    public List<Image> images;
    public List<TextMeshProUGUI> texts;
    float originalSize;
    private void Awake()
    {
        instance = this;
        originalSize = images[0].rectTransform.rect.width;
    }
    public void ShowValues(List<int> characteristics, List<int>
Items)
    {
        for (int i = 0; i < 4; i++)
        {
            images[i].rectTransform.SetSizeWithCurrentAnchors(RectTransform.
Axis.Horizontal, originalSize *
(characteristics[i]/(float)characteristics[characteristics.Count
-1-i]));
            texts[i].text = characteristics[i] + "/" +
characteristics[characteristics.Count-1 - i];
        }
        for (int i = 4; i < 8; i++)
        {
            texts[i].text = Items[i-4].ToString();
        }
        texts[8].text = "Уровень:"+Items[4].ToString();
        texts[9].text = "Очки уровня:"+Items[5].ToString();
    }
}

```

Приложение В
Игровая графика
(рекомендуемое)



Рисунок В.1 – Спрайт игрока



Рисунок В.2 – Спрайты игровых противников



Рисунок В.3 – Внешний вид города



Рисунок В.4 – Глобальная карта



Рисунок В.5 – Подземелье

Приложение Г
Результаты тестирования
(рекомендуемое)

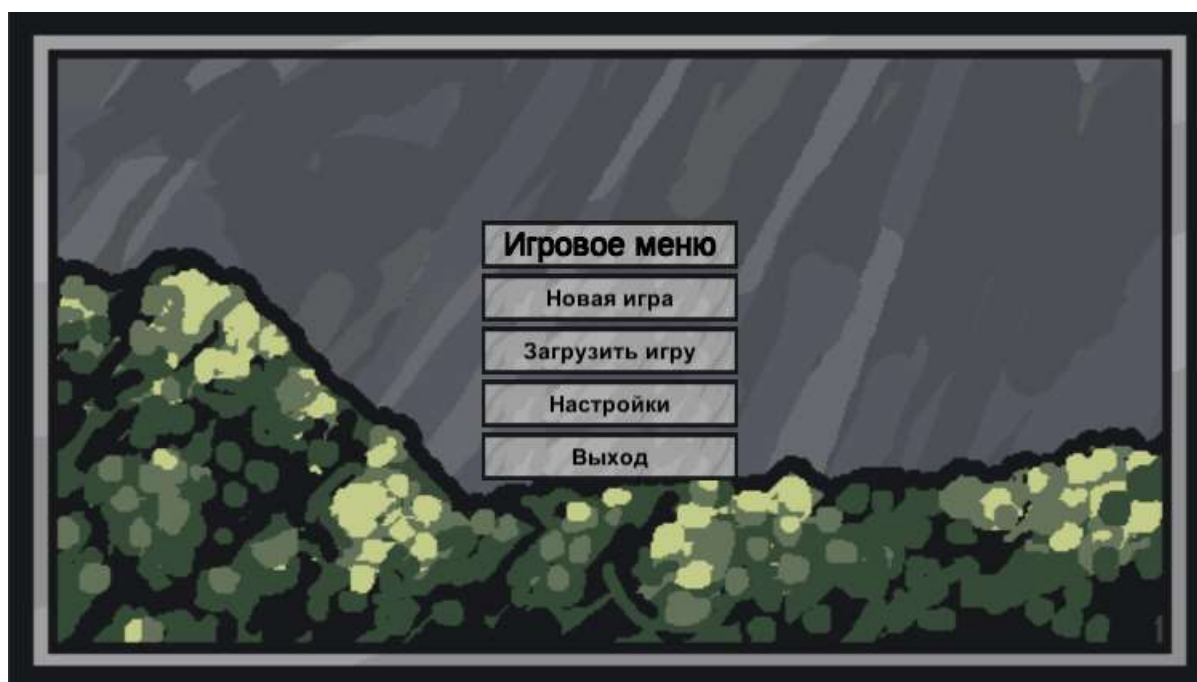


Рисунок Г.1 – Новая игра



Рисунок Г.2 – Новая игра успешно



Рисунок Г.3 – Загрузить игру



Рисунок Г.4 – Загрузить игру успешно



Рисунок Г.5 – Сохранить игру



Рисунок Г.6 – Сохранение игру успешно



Рисунок Г.7 – Оконный режим

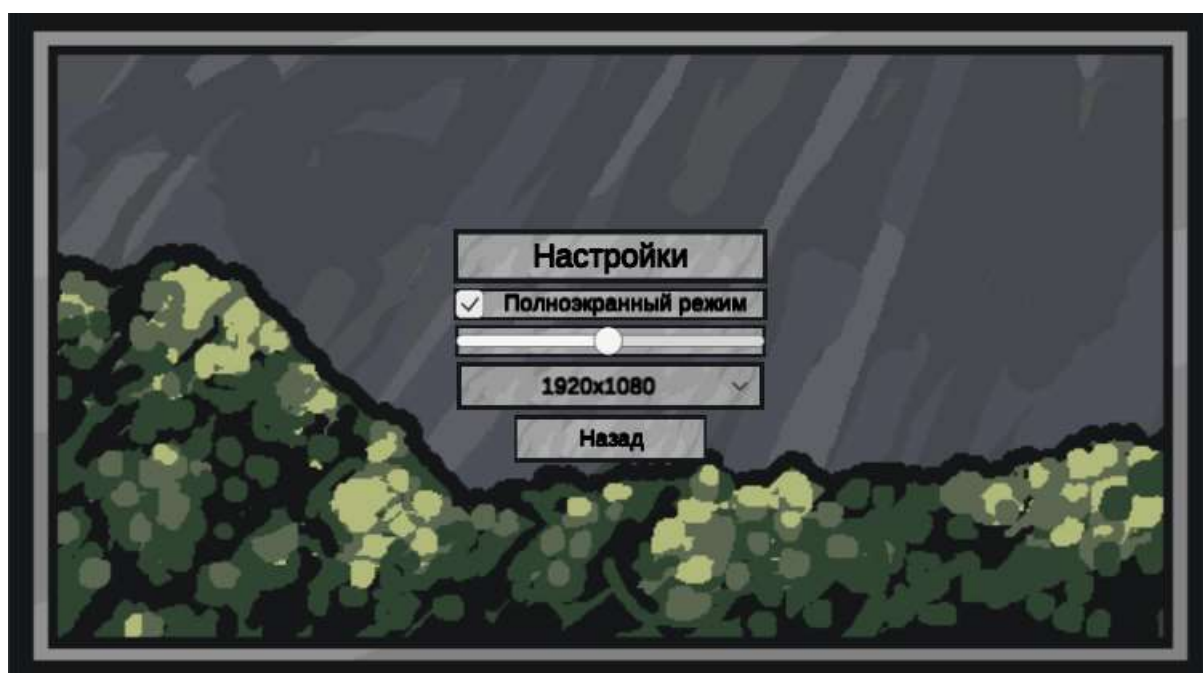


Рисунок Г.8 – Изменение громкости



Рисунок Г.9– Перемещение по игровой локации



Рисунок Г.10 – Перемещение по игровой локации успешно



Рисунок Г.11 – Использование атаки ближнего боя



Рисунок Г.12 – Использование атаки дальнего боя



Рисунок Г.13 – Использование магической атаки



Рисунок Г.14 – Использование магической атаки успешно



Рисунок Г.15 – Получение урона от атак противника



Рисунок Г.16 – Получение урона от атак противников



Рисунок Г.17 – Генерация противников



Рисунок Г.18 – Игрок подбирает монеты



Рисунок Г.19 – Игрок подбирает монеты успешно



Рисунок Г.20 – Игрок взаимодействует с игровым персонажем



Рисунок Г.21 – Игрок взаимодействует с игровым персонажем

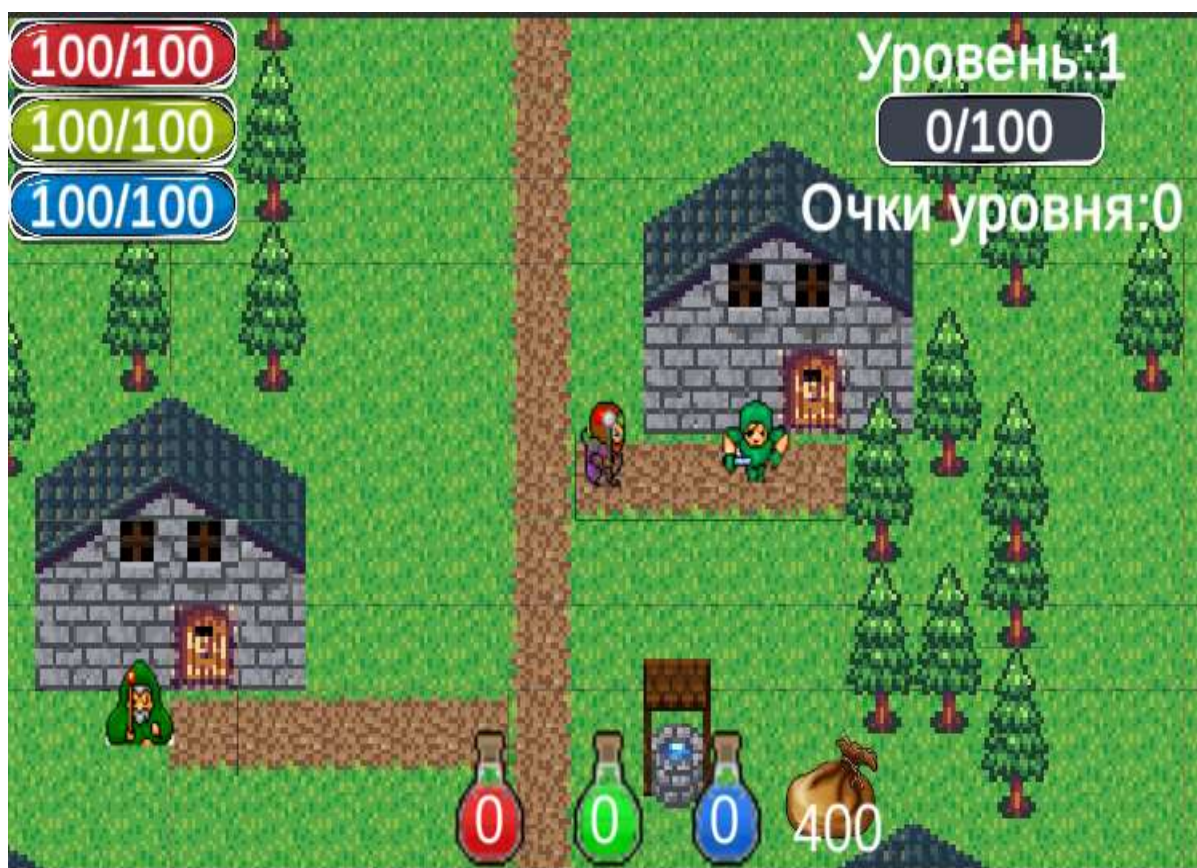


Рисунок Г.22 – Игрок взаимодействует с игровым персонажем

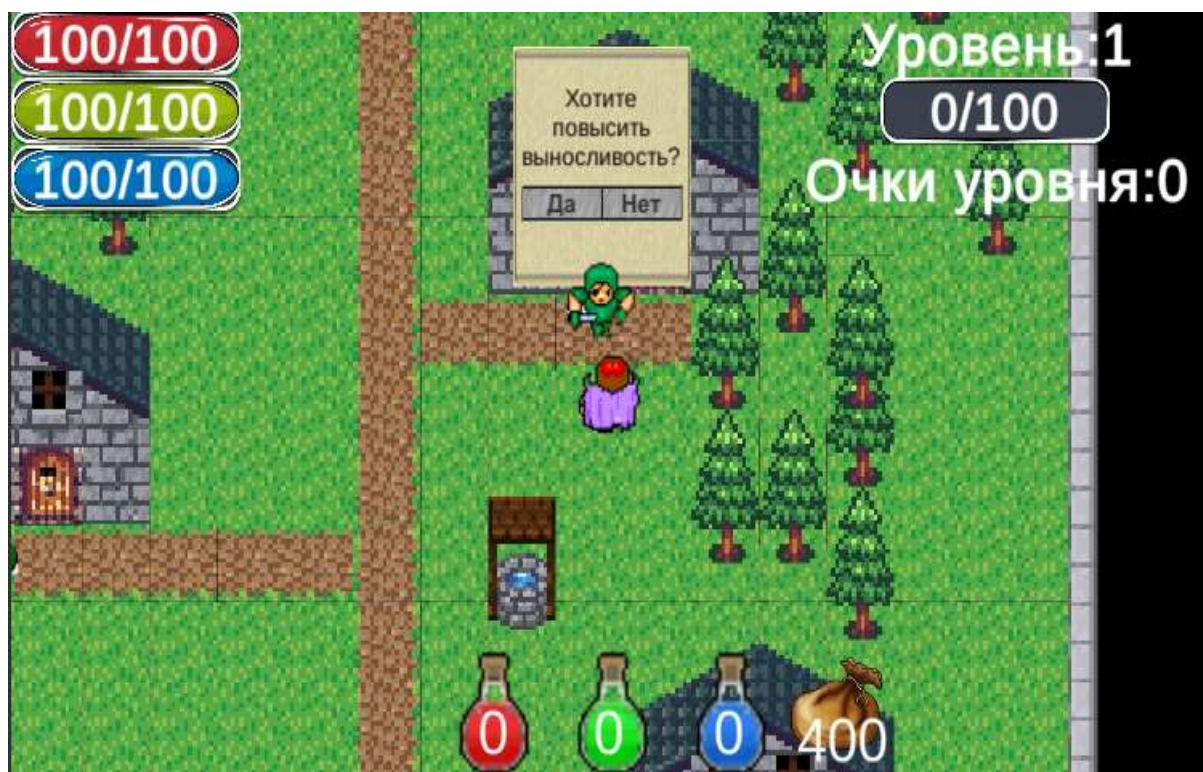


Рисунок Г.23 – Игрок взаимодействует с игровым персонажем



Рисунок Г.24 – Игрок покупает зелье здоровья



Рисунок Г.25 – Игрок покупает зелье здоровья

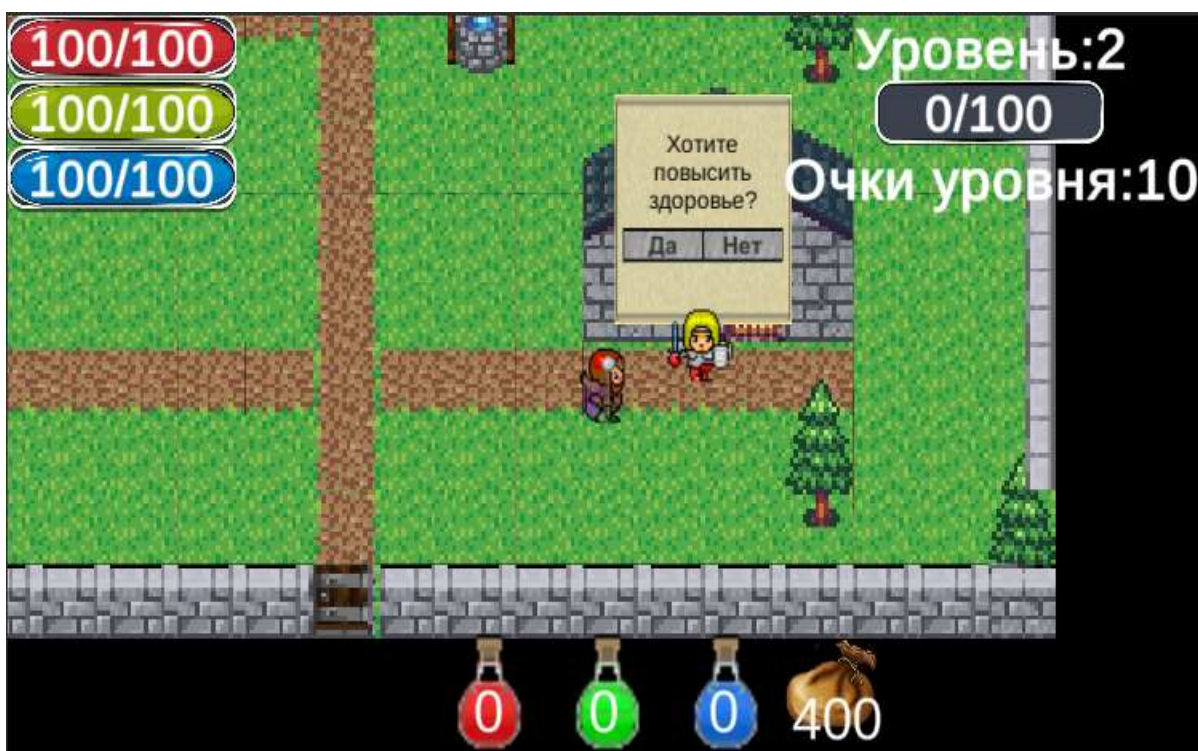


Рисунок Г.26 – Игрок улучшает параметр здоровья

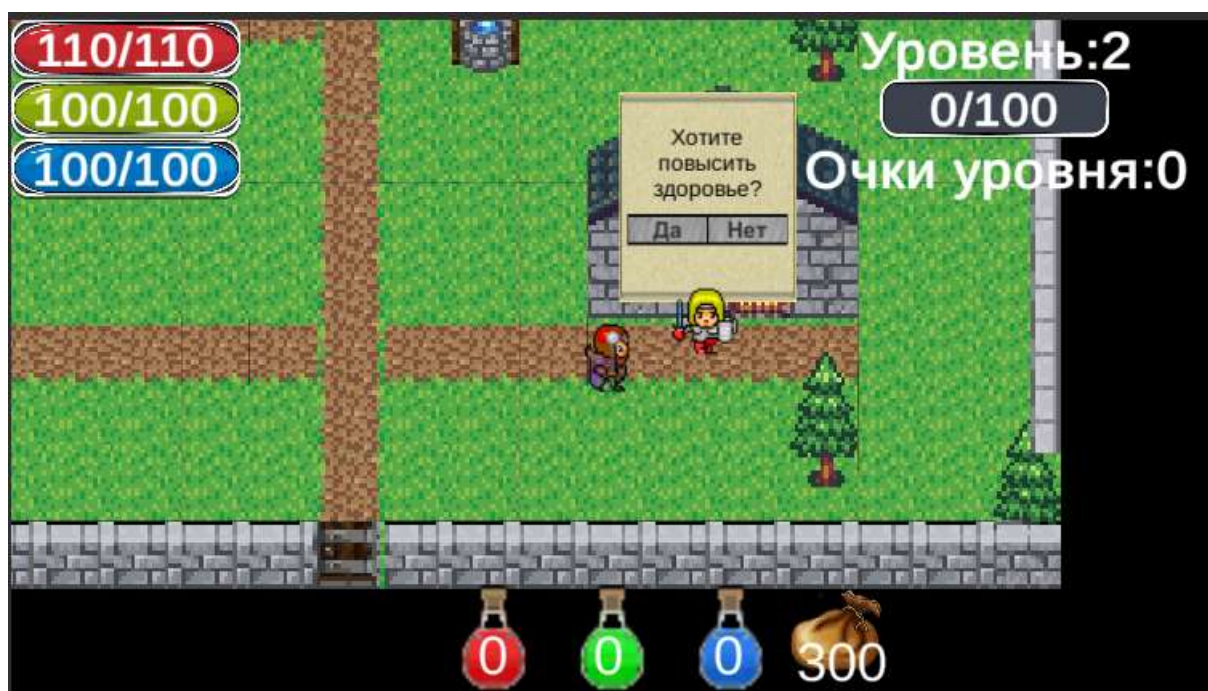


Рисунок Г.27 – Игрок улучшает параметр здоровья



Рисунок Г.28 – Игрок перемещается между локациями



Рисунок Г.29 – Игрок перемещается между локациями



Рисунок Г.30 – Вызов паузы



Рисунок Г.31 – Выпадение предметов из врагов



Рисунок Г.32 – Выпадение предметов из врагов