

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего образования

**«Южно-Уральский государственный университет
(национальный исследовательский университет)»**

Высшая школа электроники и компьютерных наук

Кафедра системного программирования

ДОПУСТИТЬ К ЗАЩИТЕ

Заведующий кафедрой, д.ф.-м.н.,
профессор

_____ Л.Б. Соколинский

«___»_____ 2022 г.

**Разработка игры в жанре «Top-Down Shooter» с видом сверху
на платформе Unity**

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
ЮУрГУ – 09.03.04.2022.308-156.ВКР

Научный руководитель,
доцент кафедры СП, к.ф.-м.н.,
доцент

_____ И.И. Клебанов

Автор работы,
студент группы КЭ-403

_____ Д.В. Кутюшкин

Ученый секретарь
(нормоконтролер)

_____ И.Д. Володченко

«___»_____ 2022 г.

Челябинск, 2022 г.

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего образования
**«Южно-Уральский государственный университет
(национальный исследовательский университет)»**
Высшая школа электроники и компьютерных наук
Кафедра системного программирования

УТВЕРЖДАЮ

Зав. кафедрой СП

_____ Л.Б. Соколинский

07.02.2022 г.

ЗАДАНИЕ

на выполнение выпускной квалификационной работы бакалавра
студенту группы КЭ-403
Кутюшкину Дмитрию Владимировичу,
обучающемуся по направлению
09.03.04 «Программная инженерия»

- 1. Тема работы** (утверждена приказом ректора от 25.04.2022 г. № 697-13/12)
Разработка игры в жанре «Top-Down Shooter» с видом сверху на платформе Unity.
- 2. Срок сдачи студентом законченной работы:** 06.06.2022 г.
- 3. Исходные данные к работе**
 - 3.1. Среда разработки Unity. [Электронный ресурс] URL: <https://unity.com/ru> (дата обращения: 11.05.2022 г.).
 - 3.2. Руководство Unity. [Электронный ресурс] URL: <https://docs.unity3d.com/ru/530/Manual> (дата обращения: 25.05.2022 г.).
 - 3.3. Роллингз Э., Моррис Д. Проектирование и архитектура игр. Вильямс, 2006. 1035 с.
- 4. Перечень подлежащих разработке вопросов**
 - 4.1. Провести анализ предметной области.
 - 4.2. Спроектировать игровое приложение.
 - 4.3. Реализовать игровое приложение.
 - 4.4. Провести тестирование игрового приложения.
- 5. Дата выдачи задания:** 07.02.2022 г.

Научный руководитель,
доцент кафедры СП, к.ф.-м.н., доцент

И.И. Клебанов

Задание принял к исполнению

Д.В. Кутюшкин

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ.....	4
1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ И ОБЗОР АНАЛОГОВ.....	6
1.1. Описание предметной области	6
1.2. Анализ аналогичных проектов	6
2. ПРОЕКТИРОВАНИЕ СИСТЕМЫ	9
2.1. Функциональные и нефункциональные требования.....	9
2.2. Концепция игры	10
2.3. Диаграмма вариантов использования	12
3. АРХИТЕКТУРА СИСТЕМЫ.....	14
3.1. Общее описание архитектуры системы.....	14
3.2. Диаграмма компонентов	15
3.3. Макеты пользовательских интерфейсов.....	16
4. РЕАЛИЗАЦИЯ СИСТЕМЫ	21
4.1. Средства реализации	21
4.2. Реализация компонента персонажа.....	21
4.3. Реализация компонента меню	26
4.4. Реализация компонента отображаемой информации.....	31
4.5. Реализация компонента врагов.....	32
4.6. Реализация компонента комнат.....	38
5. ТЕСТИРОВАНИЕ	42
5.1. Функциональное тестирование	42
5.2. Юзабилити тестирование	43
ЗАКЛЮЧЕНИЕ	45
ЛИТЕРАТУРА.....	46
ПРИЛОЖЕНИЯ.....	48
Приложение А. Спецификация вариантов использования.....	48
Приложение Б. Листинги классов	50
Приложение В. Скриншоты итоговой версии игры	66

ВВЕДЕНИЕ

Актуальность

Зарождение компьютерных игр началось еще в далеком 1952 году, когда Александром Дугласом была разработана простая игра в крестики-нолики под названием «ОХО» в рамках докторской диссертации [1]. Затем в 1962 году появилась на свет первая игра для двух игроков под названием «Spacewar!» [2].

С тех пор прошло немало времени и компьютерные игры развились в отдельную индустрию досуга, сравнимую с индустриями музыки и кино [3]. Игровая индустрия по-прежнему стремительно развивается и приносит существенный вклад в экономику. По итогам 2021 года объем российского рынка видеоигр достиг 177,4 млрд. рублей [4].

Компьютерные игры поражают разнообразием жанров, многие из которых способны развивать разные качества человека [5]. Например, в жанре «шутер» человеку необходимо быстро принимать правильные решения и предельно точно управлять персонажем, нельзя не отметить и жанр «стратегия», развивающий стратегическое мышление.

Также, для создания крупных проектов помимо программистов требуются представители многих творческих профессий, например, художники, композиторы [6]. В результате чего среди них уменьшается безработица.

Постановка задачи

Целью выпускной квалификационной работы является разработка компьютерной 2D игры в жанре «Top-Down Shooter» на платформе Unity. Для достижения данной цели необходимо решить следующие задачи:

- 1) провести анализ предметной области;
- 2) спроектировать игровое приложение;
- 3) реализовать игровое приложение;
- 4) провести тестирование игрового приложения.

Структура и содержание работы

Работа состоит из введения, пяти разделов, заключения и списка литературы. Объем работы составляет 69 страниц, объем списка литературы – 15 источников.

В первом разделе «Анализ предметной области и обзор аналогов» описываются особенности игрового жанра, а также рассматриваются аналогичные проекты.

Второй раздел «Проектирование системы» посвящен выявлению требований к системе, формированию концепции, а также определению и описанию вариантов использования.

В третьем разделе «Архитектура системы» описываются все компоненты системы, а также составляются макеты пользовательских интерфейсов.

В четвертом разделе «Реализация системы» производится обзор платформы Unity и подробно описывается реализация каждого компонента.

В пятом разделе «Тестирование» представляются результаты функционального и юзабилити тестирования.

В приложении А содержатся спецификации ключевых вариантов использования.

В приложении Б содержатся листинги некоторых классов.

В приложении В содержатся скриншоты итоговой версии игры.

1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ И ОБЗОР АНАЛОГОВ

1.1. Описание предметной области

Основными отличительными особенностями всех игр жанра «Top-Down Shooter» являются камера, закреплённая в формате «вид сверху», и стрельба как инструмент битвы с противниками. Как правило цель игр данного жанра – победить всех противников [7]. Часто игры данного жанра совмещают с другими жанрами, особенно часто с жанром «roguelike», его особенностями являются случайная генерация, необратимость смерти персонажа и повторное прохождение как часть игры [8].

1.2. Анализ аналогичных проектов

Для анализа были выбраны следующие представители жанра: «The Binding of Isaac: Rebirth» и «Nuclear Throne».

The Binding of Isaac: Rebirth

Скриншот из игры представлен на рисунке 1.



Рисунок 1 – Скриншот из игры «The Binding of Isaac»

Action RPG шутер с видом сверху, с сильным использованием элементов Roguelike [9].

Основными особенностями геймплея данной игры можно выделить:

- 1) случайную генерацию уровней;
- 2) наличие предметов, меняющих свойства «оружия» и стиль прохождения;
- 3) суммирование друг с другом эффектов от предметов, что дает множество комбинаций;
- 4) возможность стрелять лишь в 4 стороны по прямой, по этой причине стрельба управляется не мышкой, а клавишами-стрелками.

Nuclear Throne

Скриншот из игры представлен на рисунке 2.



Рисунок 2 – Скриншот из игры «Nuclear Throne»

Это постапокалиптический шутер с видом сверху в стиле roguelike [10].

Основные особенности геймплея:

- 1) минимизирована случайность, характерная для игр жанра roguelike;
- 2) огромное разнообразие противников, к каждому из которых приходится подбирать тактику;
- 3) стрельба производится наведением мыши и нажатием на ПКМ, на ЛКМ активируется специальный прием.

Вывод по первому разделу

Были проанализированы аналоги разрабатываемого приложения. Из полученных данных был сделан вывод касательно необходимых геймплейных механик и цели игры. Основные механики, встречаемые в данном типе игр:

- 1) возможность стрелять;
- 2) улучшение оружия;
- 3) специальные приемы;
- 4) разнообразные типы врагов с различным поведением, к каждому из которых нужно подбирать тактику;
- 5) также данный жанр часто содержит механики жанра roguelike, в особенности случайность.

2. ПРОЕКТИРОВАНИЕ СИСТЕМЫ

2.1. Функциональные и нефункциональные требования

Функциональные требования к проектируемой системе:

- 1) система должна отображать главное меню при запуске;
- 2) система должна закрываться при нажатии на кнопку главного меню «Выход»;
- 3) система должна запускать игровой процесс при нажатии на кнопку главного меню «Начать игру»;
- 4) система должна отображать меню настроек при нажатии на кнопку главного меню «Настройки»;
- 5) система должна отображать главное меню при нажатии на кнопку меню настроек «Назад»;
- 6) система должна менять соответствующие параметры при изменениях в меню настроек;
- 7) система должна загружать главное меню, когда здоровье персонажа опускается до нуля;
- 8) система должна открывать меню улучшений при нажатии на клавишу «Р» во время игры;
- 9) система должна отображать карту при нажатии на клавишу «М» во время игры;
- 10) система должна перемещать персонажа при нажатии на клавиши «W», «A», «S», «D» во время игры;
- 11) система должна создавать снаряд возле персонажа и двигать его по направлению курсора при нажатии на ПКМ во время игры;
- 12) система должна уничтожать врагов, когда их здоровье опускается до нуля;
- 13) система должна случайно генерировать игровое поле при запуске игрового процесса;
- 14) система должна начислять очки за уничтожение врагов.

Нефункциональные требования к проектируемой системе:

- 1) система должна работать на операционной системе Windows 7 и выше;
- 2) система должна быть разработана на платформе Unity;
- 3) система должна быть написана на языке программирования C#.

2.2. Концепция игры

Цель игры

Целью игры является убийство босса, который находится в одной из крайних комнат.

Игровые механики

Игра имеет следующие механики:

- 1) стрельба – возможность персонажа стрелять, посредством нажатия на ПКМ, это инструмент сражения с врагами;
- 2) перемещение – возможность персонажа перемещаться по комнате и между комнатами посредством нажатия клавиш «W», «A», «S», «D»;
- 3) улучшение – возможность улучшения характеристик персонажа за внутриигровую валюту, получаемую за убийство врагов.

Игровой мир

Игровой мир состоит из:

- 1) случайно сгенерированного лабиринта комнат с врагами, которые появляются в количестве от 1 до 7 при первом входе персонажа в комнату, типы врагов выбираются случайно;
- 2) главной комнаты, в которой в начале игры появляется персонаж;
- 3) комнаты с боссом.

Характеристики персонажа

Персонаж имеет следующие характеристики:

- 1) текущий запас здоровья – определяет суммарный урон, который должны нанести враги для убийства персонажа в данный момент;

- 2) максимальный запас здоровья – определяет число, выше которого не может подняться текущий запас здоровья;
- 3) скорость – определяет с какой скоростью персонаж перемещается по комнате;
- 4) количество денег – выступает в роли очков, начисляемых за убийство врагов, их можно тратить на улучшение характеристик;
- 5) урон – определяет какой урон нанесет врагу один снаряд при падании;
- 6) шанс критического удара – определяет шанс, с которым один снаряд нанесет больше урона;
- 7) критический урон – определяет во сколько раз будет увеличен урон при критическом ударе;
- 8) скорострельность – промежуток времени между текущим и следующим выстрелом;
- 9) скорость снаряда – скорость передвижения снаряда по направлению выстрела;
- 10) количество снарядов – число снарядов, вылетающих при однократном нажатии на ПКМ.

Враги

Типы врагов отличаются значением характеристик и внешним видом, некоторые имеют свой уникальный механизм атаки, всего в игре представлено 5 типов врагов, визуально отличающихся по цвету и дизайну:

- 1) зеленые – наносят персонажу среднее количество урона, когда тот попадает в радиус атаки, имеют средний запас здоровья, передвигаются со средней скоростью, частота атак также средняя;
- 2) синие – наносят персонажу малое количество урона, но с большой частотой, быстро передвигаются, имеют малый запас здоровья и отступают, когда персонаж приближается на определенное расстояние;

3) красные – наносят персонажу большое количество урона, но с низкой частотой, передвигаются со средней скоростью, имеют много здоровья и большой радиус атаки;

4) желтые – быстро приближаются к персонажу и уничтожаются, нанося очень много урона;

5) пурпурные – находятся на месте, имеют средний запас здоровья, осуществляют атаку посредством стрельбы по персонажу.

Босс

Имеет очень много здоровья, урона и 3 типа атаки, между которыми случайно переключается раз в 10 секунд, бой с ним проходит в 2 фазы, вторая фаза активируется, когда здоровье босса падает до 50%, типы атаки в ней получают усиление. Типы атак босса:

1) перемещается в центр комнаты и стреляет в 8 сторон лучами, во второй фазе выстрелы происходят с вращением;

2) телепортируется вокруг персонажа с постоянной частотой и наносит урон, во второй фазе частота постепенно нарастает, пока не достигнет пиковой;

3) телепортируется в центр комнаты и создает раз в 4 секунды трех миньонов, которые стреляют по персонажу со случайной погрешностью от 0 до 45 градусов, при прикосновении к боссу персонаж очень быстро теряет здоровье, во второй фазе создается 7 миньонов, а босс медленно расширяется.

2.3. Диаграмма вариантов использования

Для проектирования системы воспользуемся языком графического описания UML. С системой взаимодействует один актер – игрок. Игрок – это человек, играющий в игру. На рисунке 3 представлена диаграмма вариантов использования.

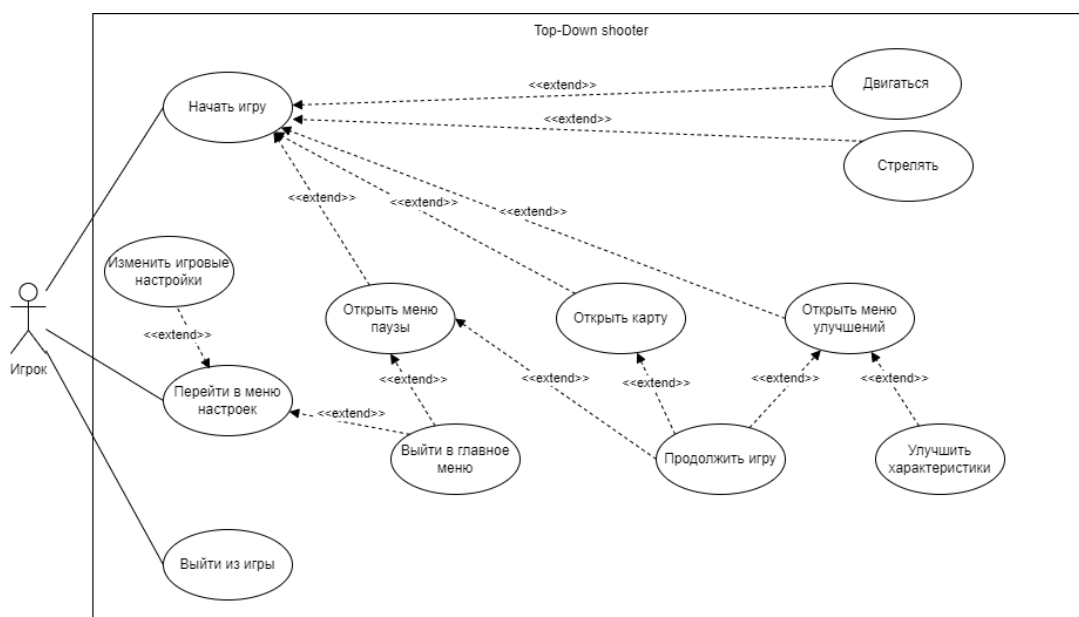


Рисунок 3 – Диаграмма вариантов использования

Первое, что видно игроку при запуске системы– главное меню, где можно начать игру, выйти из игры, перейти в меню настроек.

В меню настроек игрок может поменять настройки и вернуться в главное меню.

Когда игрок начинает игру, он может передвигаться и стрелять, а также открывать 3 меню – меню паузы, меню улучшений и карту.

Из меню паузы игрок может выйти в главное меню и вернуться к игре.

В меню улучшений он может улучшить характеристики и вернуться к игре, стоит отметить, что меню улучшений невозможно открыть если в комнате есть враги.

Открыв карту, игрок может лишь вернуться к игре.

Спецификации ключевых вариантов использования приведены в приложении А.

Вывод по второму разделу

В данном разделе была описана концепция игры, выделены функциональные и нефункциональные требования к системе, а также определены и описаны варианты использования.

3. АРХИТЕКТУРА СИСТЕМЫ

3.1. Общее описание архитектуры системы

Игра состоит из двух сцен – Game и Menu.

Сцена «Menu» открывается при запуске игры и предлагает начать игру, перейти в настройки или выйти из игры.

Сцена «Game» загружается если в сцене «Menu» игрок кликнул на кнопку «Начать игру». При запуске сцены «Game» система случайно генерирует лабиринт из комнат, и одну из крайних комнат делает комнатой с боссом. После этого игрок получает управление и может играть.

Игроку в сцене «Game» доступно несколько пользовательских интерфейсов и окно игрового поля, между которыми он переключается посредством клавиш на клавиатуре.

1. Игровое поле – окно, открытое по умолчанию, в нем игрок может передвигать персонажа, стрелять и перемещаться между комнатами.

2. Меню паузы – пользовательский интерфейс, открывающийся на клавишу «Esc», позволяет игроку остановить игровой процесс и выйти в главное меню.

3. Меню карты – пользовательский интерфейс, открывающийся на клавишу «M», позволяет игроку посмотреть расположение комнат, свое местоположение и местоположение комнаты с боссом, а также какие комнаты были зачищены.

4. Меню улучшений – пользовательский интерфейс, открывающийся на клавишу «P», позволяет игроку улучшать характеристики персонажа за внутриигровую валюту.

Вернуться к окну игрового поля из интерфейсов можно нажатием соответствующей кнопки внутри интерфейса.

3.2. Диаграмма компонентов

На рисунке 4 изображена диаграмма компонентов приложения.

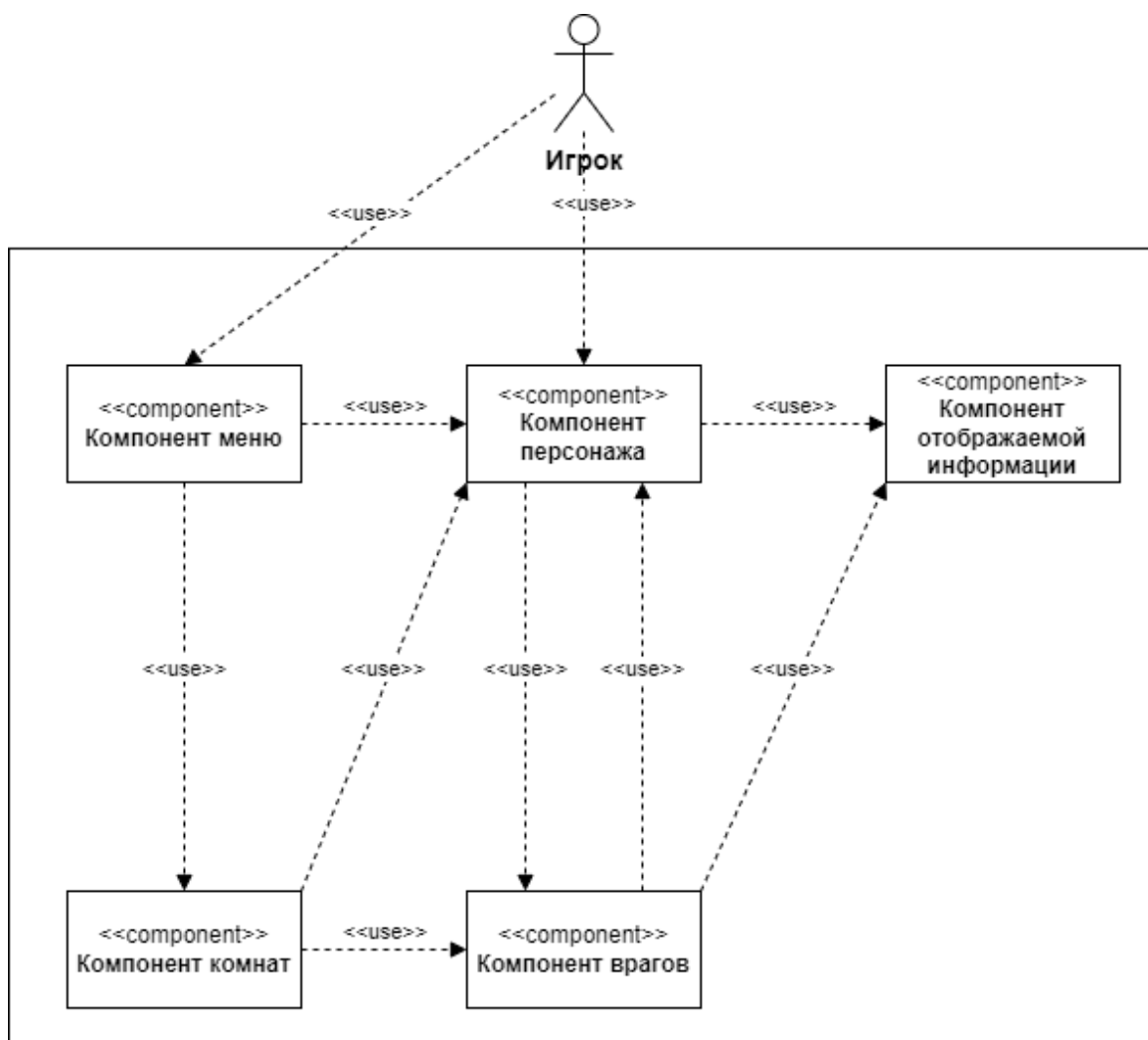


Рисунок 4 – Диаграмма компонентов

Компонент персонажа – содержит набор классов, реализующих передвижение персонажа, стрельбу, поведение снаряда после выстрела, характеристики персонажа, получение урона от врагов.

Компонент меню – содержит набор классов, реализующих главное меню, меню настроек, меню паузы, меню улучшений и карту, а также переключение между этими меню.

Компонент отображаемой информации – содержит набор классов, реализующих отображение количества монет персонажа и отображение шкалы здоровья персонажа и врагов.

Компонент комнат – содержит набор классов, реализующих случайную генерацию лабиринта комнат, создание врагов при появлении персонажа в комнате, открытие дверей после зачистки комнаты, перемещение персонажа между комнатами.

Компонент врагов – содержит набор классов, реализующих поведение врагов и босса.

3.3. Макеты пользовательских интерфейсов

Главное меню

В главном меню игроку будут доступны 3 кнопки:

- 1) начать игру – запускает сцену «Game»;
- 2) настройки – открывает меню настроек;
- 3) выйти из игры – закрывает приложение.

На рисунке 5 представлен макет интерфейса.

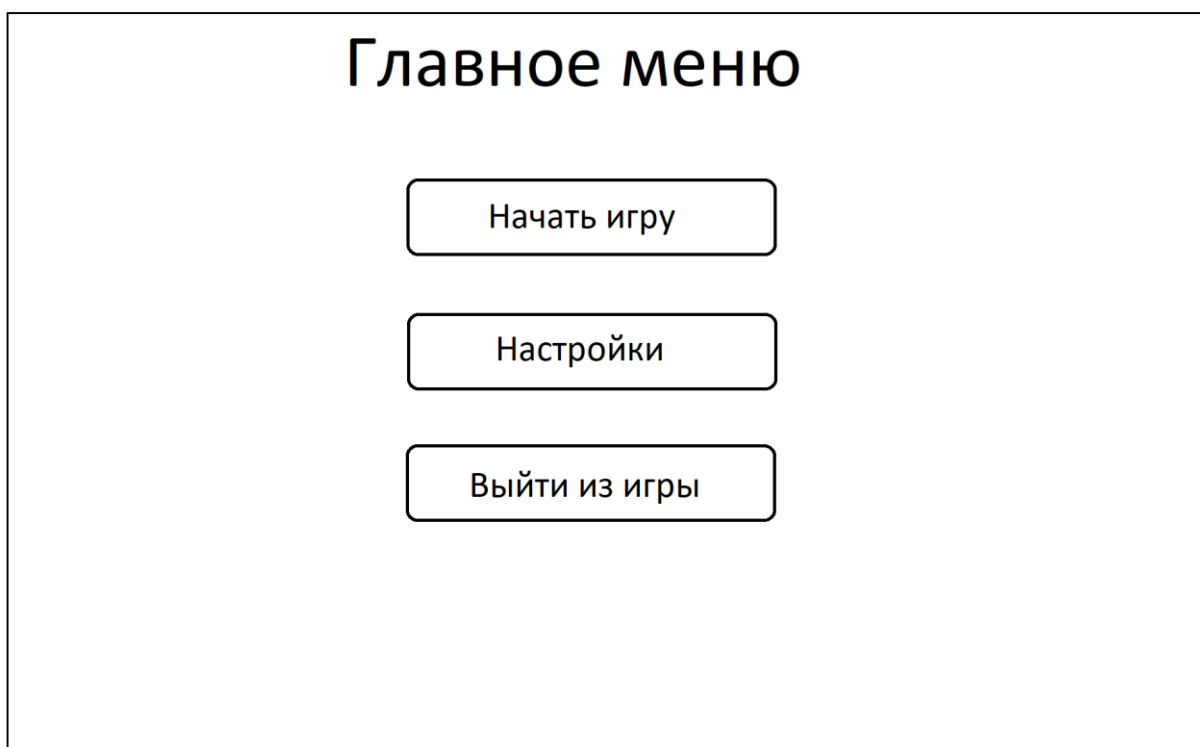


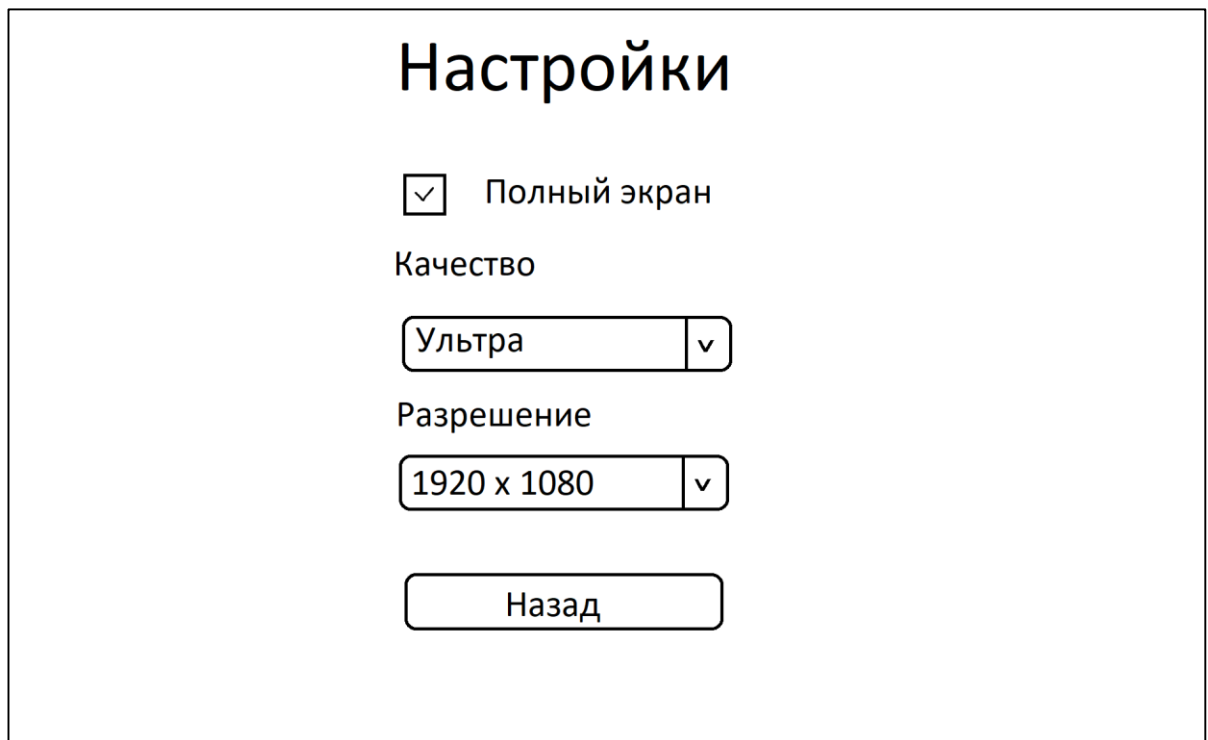
Рисунок 5 – Макет интерфейса «Главное меню»

Меню настроек

В меню настроек игроку будут доступны следующие интерактивные элементы:

- 1) чекбокс «Полный экран» – переводит игру в полноэкранный режим, когда стоит галочка;
- 2) выпадающий список «Качество» – позволяет изменить настройки качества;
- 3) выпадающий список «Разрешение» – позволяет изменить настройки разрешения;
- 4) кнопка «Назад» – закрывает меню настроек и возвращает игрока в главное меню.

На рисунке 6 представлен макет интерфейса.



Макет интерфейса меню настроек. В центре экрана находится заголовок «Настройки». Ниже него расположены три элемента управления: чекбокс «Полный экран» (с галочкой), выпадающий список «Качество» (с текущим значением «Ультра») и выпадающий список «Разрешение» (с текущим значением «1920 x 1080»). В самом низу находится кнопка «Назад».

Рисунок 6 – Макет интерфейса «Меню настроек»

Меню паузы

В меню паузы игроку будут доступны 2 кнопки:

- 1) продолжить – закрывает меню и возобновляет игровой процесс;

2) главное меню – загружает сцену «Menu».

На рисунке 7 представлен макет интерфейса.



Рисунок 7 – Макет интерфейса «Меню паузы»

Меню улучшений

Меню улучшений содержит текстовые дисплеи, а также некоторое количество кнопок:

- 1) дисплей «Количество монет» – выводит количество монет, имеющихся у персонажа;
- 2) кнопка «Продолжить» – закрывает меню и возобновляет игровой процесс;
- 3) кнопки «Характеристика» с прикрепленным дисплеем «Цена» – при нажатии на кнопку происходит увеличение соответствующей характеристики, вычитание из числа монет цены, а также отображение новой, повышенной цены в дисплее «Цена».

На рисунке 8 представлен макет интерфейса.



Рисунок 8 – Макет интерфейса «Меню улучшений»

Карта

Карта содержит:

- 1) кнопку «Продолжить», при нажатии на которую происходит закрытие карты и возобновление игрового процесса;
- 2) панель, на которую выводится информация о расположении комнат, местоположении персонажа и комнаты босса;
- 3) картинки, отображаемые на панели, каждая из которых обозначает одну комнату, в зависимости от того, обычная это комната, комната с боссом, комната с игроком или защищенная комната, выводится соответствующая картинка.

На рисунке 9 представлен макет интерфейса.

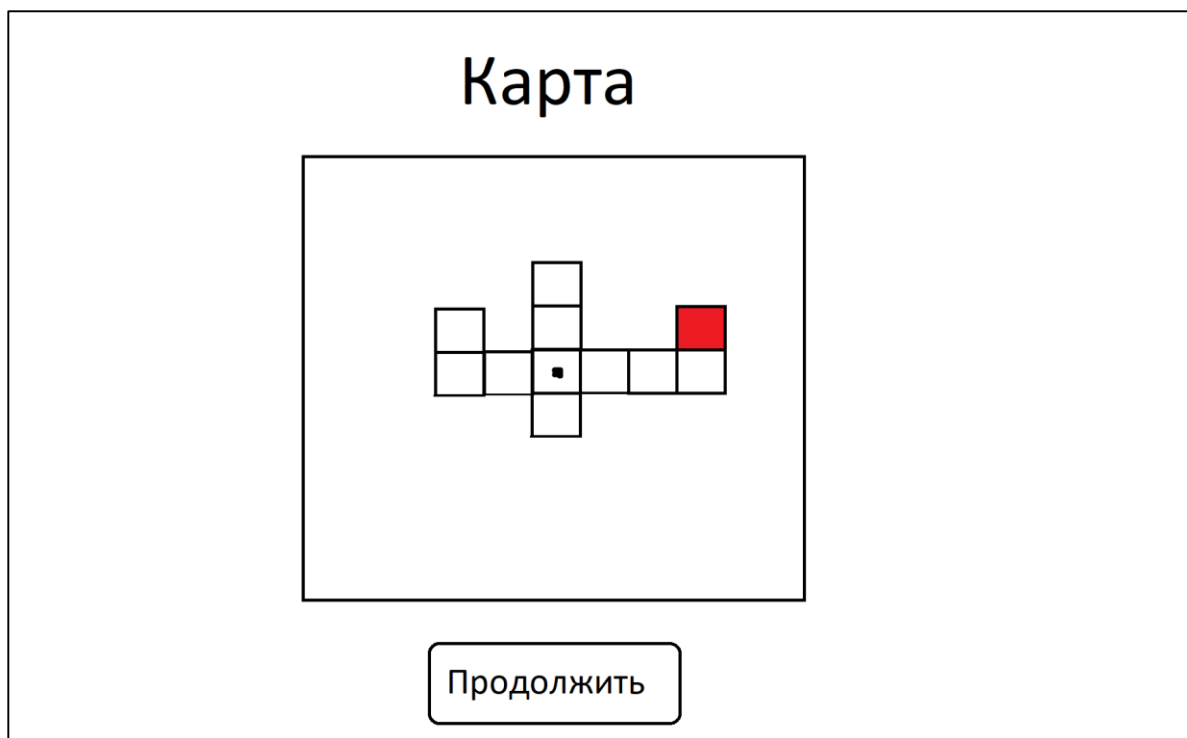


Рисунок 9 – Макет интерфейса «Карта»

Вывод по третьему разделу

В данном разделе была определена архитектура приложения, составлена диаграмма компонентов, также были спроектированы макеты интерфейсов с описанием функционала каждого элемента.

Архитектура игры представляет собой 2 сцены, собранные из игровых объектов, к которым прикреплены компоненты. Игрок может взаимодействовать с игрой, переключаясь между окном игрового поля и интерфейсами, а также выполняя внутри них некоторые действия, для более полного представления о системе, окно и каждый интерфейс были описаны.

4. РЕАЛИЗАЦИЯ СИСТЕМЫ

4.1. Средства реализации

В качестве средств реализации были выбраны игровой движок Unity Engine 2021, среда разработки Visual Studio 2019 и язык программирования C#.

В Unity архитектура проекта основана на шаблоне EntityComponentSystem, подразумевающим следующее – приложение состоит из базовых сущностей, функционал которых может быть расширен с помощью компонентов [11, 12].

Сцена состоит из игровых объектов (GameObject), каждый GameObject содержит некоторое количество компонентов, они могут быть сохранены как готовые объекты, или префабы (Prefab) [13].

Префаб – тип файла, который позволяет хранить GameObject со всеми прикрепленными к нему компонентами и значениями свойств компонентов. Он является своего рода шаблоном, для создания экземпляров хранимого объекта.

Базовый компонент любого GameObject – Transform. Он определяет положение объекта на сцене и его поворот.

Суть разработки на Unity – написание классов, которые подключаются к GameObject в роли компонентов. Любой из таких классов должен наследоваться от класса MonoBehaviour [14].

4.2. Реализация компонента персонажа

Компонент персонажа состоит из 2 классов – Player и Bullet.

Реализация передвижения персонажа

За передвижение персонажа отвечает метод `move()` класса Player, он представлен на рисунке 10.

```
private void move()
{
    move_vector = new Vector2(Input.GetAxis("Horizontal"),
        Input.GetAxis("Vertical"));
    if (move_vector.x == 0 && move_vector.y == 0)
        anim.SetBool("isRunning", false);
    else
        anim.SetBool("isRunning", true);
    move_velocity = new Vector2(move_vector.x * speed,
        move_vector.y * speed);
    rb.velocity = move_velocity;
}
```

Рисунок 10 – Код метода move()

При нажатии на любую из клавиш «W», «A», «S», «D» считывается вектор движения, затем рассчитывается скорость в этом направлении и данный вектор присваивается свойству `velocity` компонента `Rigidbody2D`. Также, в зависимости от считанного вектора проигрывается анимация простоя или бега. Метод `move()` вызывается в методе `Update()`, который вызывается каждый кадр [15].

Реализация стрельбы

За стрельбу отвечают методы `shot()` и `rotate()` класса `Player`.

Метод `rotate()` рассчитывает разницу между позицией курсора и позицией точки создания снаряда, далее выражает эту разницу в виде угла и поворачивает точку создания на этот угол, дополнительно добавив 90 градусов для корректировки. На рисунке 11 представлена его реализация.

```
private void rotate()
{
    Vector3 difference =
        Camera.main.ScreenToWorldPoint(Input.mousePosition)
        - shot_point.position;
    float rotZ = Mathf.Atan2(difference.y, difference.x)
        * Mathf.Rad2Deg;
    shot_point.rotation = Quaternion.Euler(0f, 0f, rotZ - 90);
}
```

Рисунок 11 – Код метода rotate()

Метод `shot()` просматривает значение переменной `timer_firerate`, которая выступает в роли таймера, отслеживающего возможность стрелять,

затем, если стрельба возможна, создает экземпляр снаряда в точке создания и с поворотом этой точки, который изменяется в методе `rotate()`. В случае если в результате выстрела создается больше одного снаряда, дополнительно меняется позиция, в которой создается каждый снаряд так, чтобы они находились друг от друга на равном расстоянии. На рисунке 12 представлена его реализация.

```
private void shot(int count_bullets)
{
    if (timer_firerate <= 0){
        if (Input.GetMouseButton(0)){
            if (count_bullets == 1)
                Instantiate(bullet, shot_point.position,
                    shot_point.rotation);
            if (count_bullets > 1 && count_bullets % 2 == 1){
                Instantiate(bullet, shot_point.position,
                    shot_point.rotation);
                for (int i = 1; i <= count_bullets / 2; i++)
                {
                    Vector3 increment = new Vector3(i * 0.4f, 0);
                    Instantiate(bullet,
                        shot_point.position + increment,
                        shot_point.rotation);
                }
            }
            for (int i = 1; i <= count_bullets / 2; i++)
            {
                Vector3 increment = new Vector3(-i * 0.4f, 0);
                Instantiate(bullet,
                    shot_point.position + increment,
                    shot_point.rotation);
            }
        }
        if (count_bullets > 1 && count_bullets % 2 == 0)
        {
            for (int i = 1; i <= count_bullets / 2; i++)
            {
                Vector3 increment = new Vector3(i * 0.4f, 0);
                Instantiate(bullet,
                    shot_point.position + increment,
                    shot_point.rotation);
            }
            for (int i = 1; i <= count_bullets / 2; i++)
            {
                Vector3 increment = new Vector3(-i * 0.4f, 0);
                Instantiate(bullet,
                    shot_point.position + increment,
                    shot_point.rotation);
            }
        }
        timer_firerate = firerate;
    }
    else timer_firerate -= Time.deltaTime; }
```

Рисунок 12 – Код метода `shot()`

Оба этих метода вызываются в методе `Update()` в том порядке, в котором были описаны выше.

Реализация поведения снаряда

Поведение снаряда описывается в классе `Bullet`.

Сначала в методе `Start()` снаряду присваивается вектор движения и скорость по этому вектору. Метод `Start()` вызывается при создании объекта до всех методов `Update()` [15]. Также снаряд имеет булеву переменную `isHit`, хранящую информацию о том, было ли попадание по врагу.

Затем метод `OnTriggerEnter2D()` прослушивает попадания в коллайдер-триггер снаряда других коллайдеров. Метод `OnTriggerEnter2D()` вызывается, когда в коллайдер-триггер попадает другой коллайдер [15]. Далее в зависимости от класса объекта, имеющего попавший коллайдер, выполняются действия:

1) если попал враг, босс или миньон босса, то уничтожаем снаряд и с определенным шансом наносим ему критический урон или обычный, в случае если попал рядовой враг, его объект сохраняется и ему наносится урон в методе `Update()`, это необходимо для того, чтобы урон прошел только по первому врагу, попавшему в триггер снаряда;

2) если попал другой триггер, то не выполняется никаких действий и пуля летит дальше, таким образом пуля игнорирует вражеские пули и радиусы атаки врагов;

3) если попал персонаж, не выполняется никаких действий, то есть снаряд пролетает насквозь;

4) если попало что-то, не описанное выше, то пуля уничтожается, например, такое происходит при столкновении со стеной.

Реализация представлена на рисунке 13.


```

private void Start()
{
    player = FindObjectOfType<Player>();
    rb = GetComponent<Rigidbody2D>();
    rb.velocity = transform.up * player.bullet_speed;
    isHit = false;
}

private void Update()
{
    if (isHit){
        Destroy(gameObject);
        int rand = Random.Range(1, 101);
        if (rand <= player.crit_chance)
            enemy.takeDamage(player.damage * player.crit_damage);
        else
            enemy.takeDamage(player.damage);
    }
}

private void OnTriggerEnter2D(Collider2D hit_info)
{
    if (hit_info != null)
    {
        if (hit_info.CompareTag("Enemy") && !hit_info.isTrigger)
        {
            isHit = true;
            enemy = hit_info.GetComponent<Enemy>();
        }
        else if (hit_info.CompareTag("Boss") && !hit_info.isTrigger)
        {
            boss = hit_info.GetComponent<Boss>();
            Destroy(gameObject);
            int rand = Random.Range(1, 101);
            if (rand <= player.crit_chance)
                boss.takeDamage(player.damage * player.crit_damage);
            else
                boss.takeDamage(player.damage);
        }
        else if (hit_info.CompareTag("Minion") && !hit_info.isTrigger)
        {
            minion = hit_info.GetComponent<Minion>();
            Destroy(gameObject);
            int rand = Random.Range(1, 101);
            if (rand <= player.crit_chance)
                minion.takeDamage(player.damage * player.crit_damage);
            else
                minion.takeDamage(player.damage);
        }
        else if (hit_info.isTrigger)
            return;
        else if (!(hit_info.CompareTag("Player")))
            Destroy(gameObject);
    }
}

```

Рисунок 13 – Реализация поведения снаряда

Другие методы

Рассмотрим кратко остальные методы, содержащиеся в классе `Player`:

- 1) `flip()` – разворот персонажа;
- 2) `checkFlip()` – проверка и исполнение разворота, когда персонаж смотрит в одну сторону, а движется в противоположную;
- 3) `changeHealth(float health_value)` – изменение текущего здоровья на величину `health_value`, а также его отображение и проверка смерти персонажа;
- 4) `changeMoney(float money_value)` – изменение числа монет на величину `money_value` и его отображение.

4.3. Реализация компонента меню

Компонент меню состоит из 5 классов – `MainMenuControl`, `RuntimeMenuControl`, `Upgrade`, `Settings` и `Map`.

Реализация главного меню

Главное меню находится на отдельной сцене и является UI элементом `Panel` с несколькими кнопками, к каждой кнопке прикреплен соответствующий метод из класса `MainMenuControl`, прикрепленные методы вызываются при нажатии на кнопку.

Исключением является кнопка перехода в меню настроек, при нажатии на нее вызывается стандартный метод `SetActive()` меню настроек, которое изначально неактивно, в результате чего после активации меню настроек показывается вместо главного меню. Меню настроек также находится на сцене с главным меню.

На рисунке 14 можно увидеть, как в инспекторе крепятся методы к кнопкам.

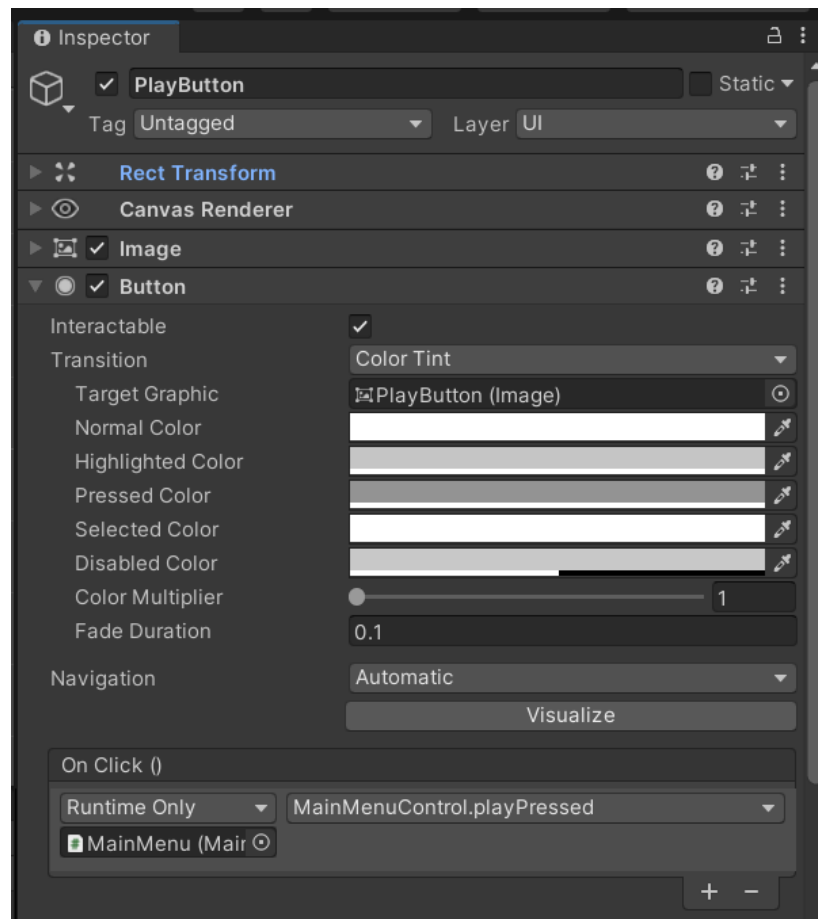


Рисунок 14 – Пример прикрепления метода к кнопке

В методе `Start()` класса `MainMenuControl` системный курсор заменяется на игровой курсор для интерфейса, а также выгружается сцена с игровым процессом, если она запущена вместе со сценой главного меню. Код метода `Start()` представлен на рисунке 15.

```
private void Start()
{
    Cursor.SetCursor(cursor_interface, Vector2.zero, CursorMode.Auto);
    if (SceneManager.sceneCount > 1)
        SceneManager.UnloadSceneAsync("Game");
}
```

Рисунок 15 – Код метода `Start()`

Нажатие на кнопки начала игры и выхода из игры обрабатывают методы `playPressed()` и `exitPressed()` класса `MainMenuControl` соответственно.

Метод `playPressed()` устанавливает курсор в виде прицела и загружает сцену с игровым процессом.

Метод `exitPressed()` закрывает приложение.

Их код представлен на рисунке 16.

```
public void playPressed()
{
    Cursor.SetCursor(cursor_game, game_hot_spot, CursorMode.Auto);
    SceneManager.LoadScene("Game");
}

public void exitPressed()
{
    Application.Quit();
}
```

Рисунок 16 – Код методов `playPressed()` и `exitPressed()`

Реализация меню настроек

Меню настроек является UI элементом `Panel` и содержит несколько UI элементов, к каждому из которых прикреплен соответствующий метод класса `Settings`, исключением является кнопка «Назад», при нажатии на нее вызывается метод `SetActive()` меню настроек, который делает меню неактивным, в результате чего на экране снова показывается главное меню. Изначально меню настроек неактивно, и активировать его можно нажатием на соответствующую кнопку в главном меню.

Рассмотрим методы класса `Settings`.

1. Метод `Awake()` – вызывается до всех методов `Start()` независимо от того включен ли объект [15]. Здесь в методе `Awake()` создается список всех возможных разрешений экрана, а затем выпадающий список, отвечающий за настройку разрешения, заполняется данными значениями. Выпадающий список с настройками качества был заполнен вручную в инспекторе.

2. Метод `quality()` – устанавливается качество с `id` выбранного в выпадающем списке пункта.

3. Метод `fullScreenToggle()` – свойству `fullScreen` класса `Screen` присваивается булево значение состояния чекбокса.

4. Метод `resolution()` – устанавливается разрешение с `id` выбранного в выпадающем списке пункта.

Код рассмотренных методов представлен на рисунке 17.

```
public void Awake()
{
    resolutions = new List<string>();
    rsl = Screen.resolutions;
    foreach (Resolution i in rsl)
    {
        resolutions.Add(i.width + "x" + i.height);
    }
    dropdown.ClearOptions();
    dropdown.AddOptions(resolutions);
}

public void quality(int qual)
{
    QualitySettings.SetQualityLevel(qual);
}

public void fullScreenToggle()
{
    Screen.fullScreen = fullscreen_toggle.isOn;
}

public void resolution(int r)
{
    Screen.SetResolution(rsl[r].width, rsl[r].height,
        fullscreen_toggle.isOn);
}
```

Рисунок 17 – Код методов класса `Settings`

Реализация игровых меню

Меню паузы, улучшений и карта находятся на сцене с игровым процессом и изначально неактивны, они активируются при нажатии на соответствующие клавиши. Стоит отметить, что меню улучшений невозможно открыть, когда в комнате есть враги.

При активации любого из меню курсор в виде прицела заменяется на курсор интерфейса, игровой процесс останавливается, а остальные меню,

если были активны, становятся неактивными, вышеперечисленные операции выполняются в методах `checkPause()`, `checkUpgrade()` и `checkMap()` класса `RuntimeMenuControl`, они вызываются в методе `Update()` данного класса. Объект этого класса всегда активен и его функция – отслеживание вызовов меню.

На каждом из игровых меню присутствует кнопка «Продолжить», к каждой из которых прикреплен соответствующий меню метод, при нажатии на кнопку устанавливается курсор в виде прицела, меню становится неактивным и игровой процесс возобновляется. Методы, прикрепленные к кнопкам «Продолжить», также находятся в классе `RuntimeMenuControl`.

Меню паузы дополнительно содержит кнопку «Главное меню», к ней прикреплен метод `mainMenuPressed()` класса `RuntimeMenuControl`, при нажатии на нее загружается сцена главного меню.

Логикой меню улучшений управляет класс `Upgrade`, меню улучшений состоит из кнопки продолжить, текстового дисплея, показывающего количество имеющихся монет, множества кнопок, отвечающих за улучшение характеристик, к каждой из которых прикреплен текстовый дисплей, отображающий текущую цену на данное улучшение, и метод, осуществляющий данное улучшение. Эти методы работают следующим образом – получается цена на улучшение, в случае если у персонажа хватает монет улучшается характеристика, затем у персонажа убавляются монеты, а цена на следующее улучшение возрастает.

Логикой карты управляет класс `Map`, карта состоит из кнопки продолжить и панели, на которой будет рисоваться карта. Сначала карта получает данные о сгенерированном лабиринте комнат из класса `RoomGenerator`, который будет описан далее, и с помощью этих данных и подготовленных изображений обычных комнат, зачищенных комнат, обычных комнат с персонажем, комнаты с боссом, комнаты с боссом и персонажем, рисует карту каждый раз, когда меню карты становится активным.

Листинги классов `RuntimeMenuControl`, `Upgrade` и `Map` приведены в приложении Б.

4.4. Реализация компонента отображаемой информации

Компонент отображаемой информации состоит из 2 классов – `HealthBar` и `MoneyCount`.

Реализация отображения здоровья

Класс `HealthBar` манипулирует картинкой `bar`, в методе `Start()` заливка картинки указывается как полная, далее в методе `Update()` заливка шкалы изменяется в зависимости от значения `fill`. Величина `fill` является отношением текущего здоровья сущности к `full_health`, эти 2 величины изменяются в классах `Player` и `Enemy` в методах `changeHealth()` и `takeDamage()` соответственно. Также есть метод `flip()` для разворота шкалы, он используется у врагов для обратного разворота шкалы при повороте врага. Код класса `HealthBar` представлен на рисунке 18.

```
public class HealthBar : MonoBehaviour
{
    public Image bar;
    public float fill;
    public float full_health;

    void Start()
    {
        fill = 1f;
    }

    void Update()
    {
        bar.fillAmount = fill;
    }

    public void flip()
    {
        if (bar.fillOrigin == 0) bar.fillOrigin = 1;
        else bar.fillOrigin = 0;
    }
}
```

Рисунок 18 – Класс `HealthBar`

Реализация отображения количества монет

Отображение числа монет реализовано методом `displayMoney()`, этот метод отображает на дисплеях `money_displays` число `money`. Код класса `MoneyCount` представлен на рисунке 19.

```
public class MoneyCount : MonoBehaviour
{
    public Text[] money_displays;

    public void displayMoney(float money)
    {
        foreach (Text display in money_displays)
        {
            display.text = System.Convert.ToString(money);
        }
    }
}
```

Рисунок 19 – Класс `MoneyCount`

4.5. Реализация компонента врагов

Компонент врагов состоит из 8 классов: `Boss`, `Enemy`, `EnemyGreenBlueRed`, `EnemyYellow`, `EnemyPurple`, `Minion`, `GunEnemy`, `BulletEnemy`.

Реализация передвижения рядовых врагов

Передвижением врагов управляет метод `move()` класса `Enemy`, метод работает следующим образом:

- 1) если дистанция до персонажа больше чем `stop_dist`, то враг преследует персонажа;
- 2) если дистанция до персонажа меньше, чем `stop_dist` и больше, чем `retr_dist`, то враг стоит на месте, у большинства врагов дистанция остановки совпадает с радиусом атаки;
- 3) если дистанция до персонажа меньше, чем `retr_dist`, то враг отступает.

Код метода `move()` представлен на рисунке 20.


```
protected void move()
{
    if (Vector2.Distance(transform.position,
        player.transform.position) > stop_dist)
        transform.position =
            Vector2.MoveTowards(transform.position,
                player.transform.position,
                current_speed * Time.deltaTime);
    else if (Vector2.Distance(transform.position,
        player.transform.position) <= stop_dist
        && Vector2.Distance(transform.position,
        player.transform.position) > retr_dist)
        return;
    else if (Vector2.Distance(transform.position,
        player.transform.position) <= retr_dist)
        transform.position =
            Vector2.MoveTowards(transform.position,
                player.transform.position,
                -current_speed * Time.deltaTime);
}
```

Рисунок 20 – Метод move ()

Реализация атаки рядовых врагов

Классы EnemyGreenBlueRed, EnemyYellow и EnemyPurple унаследованы от класса Enemy и реализуют атаку врагов, указанных в названии класса, за исключением класса EnemyPurple.

Зеленый, синий и красный враги атакуют по одному принципу и отличаются лишь характеристиками. Когда срабатывает метод OnTriggerStay2D() и в триггере находится персонаж, враг начинает отсчет до начала атаки, когда отсчет будет завершен, персонажу нанесется урон. В случае если сработает метод OnTriggerExit2D() и триггер покинет персонаж, отсчет откатится до изначального значения, это имитация уворота. Метод OnTriggerStay2D() срабатывает, когда в коллайдере-триггере находится другой коллайдер, а метод OnTriggerExit2D() срабатывает, когда коллайдер-триггер покидает другой коллайдер [15]. Реализация этих методов для класса EnemyGreenBlueRed представлена на рисунке 21.

```

public void OnTriggerStay2D(Collider2D hit_info)
{
    if (hit_info.CompareTag("Player"))
    {
        if (timer_attack_rate <= 0)
        {
            hit_info.GetComponent<Player>().changeHealth(-damage);
            timer_attack_rate = attack_rate;
        }
        else
        {
            anim.SetBool("isAttack", true);
            rb.WakeUp();
            timer_attack_rate -= Time.deltaTime;
        }
    }
}

private void OnTriggerExit2D(Collider2D hit_info)
{
    if (hit_info.CompareTag("Player"))
    {
        anim.SetBool("isAttack", false);
        timer_attack_rate = attack_rate;
    }
}

```

Рисунок 21 – Реализация атаки врагов класса EnemyGreenBlueRed

Желтый враг атакует по тому же принципу, но уничтожается после доведенной до конца атаки и у него не реализован метод OnTriggerExit2D(). Реализация атаки для врагов класса EnemyYellow представлена на рисунке 22.

```

public void OnTriggerStay2D(Collider2D hit_info)
{
    if (hit_info.CompareTag("Player"))
    {
        if (timer_attack_rate <= 0)
        {
            hit_info.GetComponent<Player>().changeHealth(-damage);
            Destroy(gameObject);
        }
        else
        {
            rb.WakeUp();
            timer_attack_rate -= Time.deltaTime;
        }
    }
}

```

Рисунок 22 – Реализация атаки врагов класса EnemyYellow

Пурпурный враг является стрелком, класс `EnemyPurple` содержит лишь переопределение метода `flip()` для того, чтобы он не вертелся, потому что является турелью по аналогии с реальным миром. Его атака реализована в классах `GunEnemy` и `BulletEnemy`.

Методы `rotate()` и `shot()` класса `GunEnemy` реализованы аналогичным образом с одноименными методами класса `Player`. За тем исключением, что метод `rotate()` поворачивает пушку не за курсором, а за персонажем, и метод `shot()` создает лишь одну пулю. На рисунке 23 представлена реализация класса `GunEnemy`.

```
public class GunEnemy : MonoBehaviour
{
    private float timer_firerate;
    public GameObject bullet;
    public Transform shot_point;
    public float firerate;
    private Player player;

    private void Start()
    {
        player = FindObjectOfType<Player>();
    }
    void Update()
    {
        rotate();
        shot();
    }
    private void shot()
    {
        if (timer_firerate <= 0)
        {
            Instantiate(bullet, shot_point.position, transform.rotation);
            timer_firerate = firerate;
        }
        else timer_firerate -= Time.deltaTime;
    }
    private void rotate()
    {
        Vector3 difference = player.transform.position - transform.position;
        float rotZ = Mathf.Atan2(difference.y, difference.x) *
            Mathf.Rad2Deg;
        transform.rotation = Quaternion.Euler(0f, 0f, rotZ - 90);
    }
}
```

Рисунок 23 – Реализация класса `GunEnemy`

Класс `BulletEnemy` также аналогичен классу `Bullet`, управляющему пулями персонажа, за тем исключением, что попадание регистрируется по персонажу, а через врагов пули пролетают насквозь. На рисунке 24 представлена реализация класса `BulletEnemy`.

```
public class BulletEnemy : MonoBehaviour
{
    private Rigidbody2D rb;
    public float speed;
    public float damage;
    private Player player;

    void Start()
    {
        rb = GetComponent<Rigidbody2D>();
        rb.velocity = transform.up * speed;
    }
    private void OnTriggerEnter2D(Collider2D hit_info)
    {
        if (hit_info != null)
        {
            if (hit_info.CompareTag("Player"))
            {
                Destroy(gameObject);
                hit_info.GetComponent<Player>().changeHealth(-damage);
            }
            else if (hit_info.isTrigger)
            {
                return;
            }
            else if (!(hit_info.CompareTag("Enemy"))
                && !(hit_info.CompareTag("Bullet"))
                && !(hit_info.CompareTag("Boss"))
                && !(hit_info.CompareTag("Minion")))
            {
                Destroy(gameObject);
            }
        }
    }
}
```

Рисунок 24 – Реализация класса `BulletEnemy`

Реализация босса и его миньонов

Поведение босса реализовано в классе `Boss` следующим образом: в методе `Update()` случайно генерируется одно из трех состояний раз в 10 секунд, босс телепортируется в центр, затем в зависимости от количества здоровья и состояния вызывается один из методов – `distant()`, `melee()` и

`minionAndAoe()` первой или второй фазы. Фаза определяется булевым аргументом метода. Рассмотрим подробнее каждый из них.

Метод `melee()` является состоянием ближнего боя, сначала методом `generateIncrement()` случайно генерируется приращение от позиции персонажа таким образом, что при сложении всех возможных приращений и позиции персонажа множество возможных итоговых точек будет лежать на окружности вокруг персонажа, если персонаж стоит близко к стене, точки, телепортирующие босса в стену и за нее отсекаются, далее босс телепортируется на сгенерированное приращение от персонажа и наносит удар, в 1 фазе данная операция происходит циклично с постоянной частотой. В случае, если метод был вызван во 2 фазе, частота нарастает с каждым ударом, пока не достигнет пиковой.

Метод `distant()` является состоянием дальнего боя, в нем босс стреляет лучами в 8 сторон с равным углом между ними. В 1 фазе каждый луч летит в постоянном направлении. В случае, если метод был вызван во 2 фазе, направление полета каждого луча сдвигается на пару градусов после каждого выстрела, т.е. стрельба происходит с вращением.

Метод `minionAndAoe()` реализован следующим образом – каждые 4 секунды босс создает в случайных точках комнаты несколько миньонов, эти точки генерируются методом `generateMinionPosition()`, также при приближении персонажа вплотную к боссу в методе `OnTriggerStay2D()` персонажу каждый кадр наносится немного урона, можно провести аналогию с эффектом горения. В 1 фазе создается 3 миньона, а босс бездействует. Во 2 фазе создается 7 миньонов, а босс каждый кадр немного расширяется.

Поведение миньонов описано в классе `Minion`, миньоны не двигаются и стреляют по персонажу лучами босса с погрешностью от 0 до 45 градусов.

Листинги классов `Boss` и `Minion` приведены в приложении Б.

Другие методы

Рассмотрим кратко другие методы, имеющиеся в описанных классах:

- 1) метод `takeDamage()` – получение врагом урона и его смерть;
- 2) методы `flip()` и `checkFlip()` – разворот врага в сторону персонажа и проверка необходимости разворота, имеются только в классе `Enemy` и унаследованных от него классах;
- 3) метод `checkStun()` – короткая приостановка врага при попадании по нему снаряда, имеется только в классе `Enemy` и унаследованных от него классах.

4.6. Реализация компонента комнат

Компонент комнат состоит из 5 классов: `RoomGenerator`, `Spawner`, `OpenDoor`, `DoorDeleter`, `ChangeRoom`.

Реализация генерации комнат

За генерацию комнат отвечает метод класса `RoomGenerator`, в частности метод `spawnRooms()`. Первоначально на карте уже имеется 5 комнат: центровая с 4 дверьми, к каждой из этих дверей подведена комната с 2 дверьми, лежащими на противоположных сторонах. Далее в методе `Start()` генерируется количество комнат, которые будут созданы, на основе этого количества рассчитывается величина `size`, создается двумерный массив из комнат размера `size`, в центральную и соприкасающиеся с ней ячейки записываются изначально существующие комнаты. Далее в методе `spawnRooms()` мы проходим по всему полю, находя комнату, мы случайно выбираем одно из доступных направлений и если текущая комната будет единственным соседом будущей, создаем в этом направлении комнату, во время генерации последней комнаты мы создаем комнату босса. Когда цикл достигнет конца, мы заново его запускаем до тех пор, пока не будет создана комната с боссом, т.е. не будут сгенерированы все комнаты. По итогу лабиринт комнат не будет замыкаться и иметь ветвистую структуру.

Прочие методы:

1) метод `activateOpenDoor()` – заполняет массив дверей в классе `OpenDoor`, без этой операции этот класс работать не сможет, он вызывается спустя 20мс работы класса `RoomGenerator`;

2) метод `haveOnlyOneNeighboor()` – проверяет, имеет ли комната с указанными индексами только одного соседа;

3) метод `deleteAllExcessDoors()` – удаляет у всех комнат двери, которые не соединяют с другими комнатами.

Листинг класса `RoomGenerator` приведен в приложении Б.

Реализация создания врагов

За создание врагов при входе персонажа в комнату отвечает класс `Spawner`. При входе персонажа в триггер выполняется случайное количество раз с указанным верхним пределом метод `spawnOne()`, который работает следующим образом – случайно выбирается одна из точек спавна и в ней создается враг случайного типа, затем данная точка спавна удаляется. Благодаря булевой переменной `spawned` для каждой комнаты спавн происходит только один раз. На рисунке 25 представлен код класса `Spawner`.

```
public class Spawner : MonoBehaviour
{
    public List<GameObject> enemy_types, spawnpoints;
    private GameObject spawnpoint;
    private bool spawned = false;
    public int max_count;

    private void OnTriggerEnter2D(Collider2D collision) {
        if (collision.CompareTag("Player")) {
            if (!spawned)
                for (int count = 1; count <= Random.Range(1,
max_count); count++)
                    spawnOne();
            spawned = true;
        }
    }
    private void spawnOne() {
        int index_spawnpoint = Random.Range(0, spawnpoints.Count);
        Instantiate(enemy_types[Random.Range(0, enemy_types.Count)],
            spawnpoints[index_spawnpoint].transform.position,
            transform.rotation);
        spawnpoints.RemoveAt(index_spawnpoint);
    }
}
```

Рисунок 25 – Класс `Spawner`

Реализация открытия дверей

За открытие дверей сразу после зачистки комнаты отвечает класс `OpenDoor`.

Сначала из класса `GenerateDoors` вызывается метод класса `OpenDoor` `getGeneratedDoors()`, который возвращает массив всех дверей, затем в зависимости от того, имеются ли враги, метод `Update()` проходит по каждой двери и активирует ее, либо делает неактивной.

На рисунке 26 представлена реализация класса `OpenDoor`.

```
public class OpenDoor : MonoBehaviour
{
    public GameObject[] doors = null;
    private Enemy enemy;
    private Boss boss;
    private Minion minion;

    private void Update()
    {
        if (doors != null)
        {
            enemy = FindObjectOfType<Enemy>();
            boss = FindObjectOfType<Boss>();
            minion = FindObjectOfType<Minion>();
            if (enemy == null && boss == null && minion == null)
            {
                foreach (GameObject door in doors)
                {
                    door.SetActive(false);
                }
            }
            if (enemy != null || boss != null || minion != null)
            {
                foreach (GameObject door in doors)
                {
                    door.SetActive(true);
                }
            }
        }
    }

    public GameObject[] getGeneratedDoors()
    {
        GameObject[] doors;
        doors = GameObject.FindGameObjectsWithTag("Door");
        return doors;
    }
}
```

Рисунок 26 – Класс `OpenDoor`

Реализация удаления дверей

Класс `DoorDeleter` отвечает за удаление указанных дверей у комнаты и хранение информации о том какие двери есть у данной комнаты, находится ли в ней персонаж и была ли она зачищена.

Листинг класса `DoorDeleter` приведен в приложении Б.

Реализация перемещения между комнатами

За перемещение между комнатами отвечает класс `ChangeRoom`, когда персонаж касается определенной зоны у двери, камера перемещается на комнату, соответствующую двери, персонаж перемещается ко входу этой комнаты. Реализация класса `ChangeRoom` представлена на рисунке 27.

```
public class ChangeRoom : MonoBehaviour
{
    private Camera cam;
    public Vector3 camera_position_to_change;
    public Vector3 player_position_to_change;

    void Start()
    {
        cam = Camera.main.GetComponent<Camera>();
    }

    private void OnTriggerEnter2D(Collider2D other)
    {
        if (other.CompareTag("Player"))
        {
            other.transform.position += player_position_to_change;
            cam.transform.position += camera_position_to_change;
        }
    }
}
```

Рисунок 27 – Класс `ChangeRoom`

Вывод по четвертому разделу

В соответствии с требованиями были реализованы компоненты приложения, были созданы некоторые объекты, к ним добавлены некоторые стандартные компоненты, а также реализованные, реализованные компоненты были подробно описаны. Разработанное приложение полностью соответствует сформированным в предыдущих разделах требованиям. Для приложения было написано 22 класса, общая численность строк кода – 1478.

5. ТЕСТИРОВАНИЕ

5.1. Функциональное тестирование

В ходе данного тестирования проверялось соответствие приложения функциональным требованиям. Все тесты были успешно пройдены. В таблице 1 представлены результаты тестирования.

Таблица 1 – Функциональное тестирование игры

№	Название теста	Действия	Результат
1	Работоспособность меню настроек	1. В главном меню нажать кнопку «Настройки». 2. Поменять все возможные настройки.	Из главного меню был осуществлен переход в меню настроек, измененные настройки успешно применились.
2	Работоспособность управления	1. В главном меню нажать на кнопку «Начать игру». 2. Нажать поочередно клавиши «W», «A», «S», «D». 3. Поводить мышкой по экрану время от времени нажимать ЛКМ.	Персонаж двигается вверх, влево, вниз, вправо, при нажатии на ЛКМ снаряды летят точно по направлению курсора.
3	Переход между комнатами	1. Пройти через все 4 прохода в главной комнате.	Персонаж, камера и дисплеи каждый раз перемещаются в соответствующую комнату.
4	Уничтожение врагов	1. Пройти в любой проход. 2. Навестись на любого врага и зажать ЛКМ.	Полоска здоровья выбранного врага уменьшается после каждого попадания, когда она уменьшается полностью, враг исчезает, число отображаемых на дисплее монет увеличилось.
5	Смерть персонажа	1. Зайти в любую комнату. 3. Подождать некоторое время.	Враги подбегают к персонажу и начинают атаковать, полоска здоровья уменьшается, когда она уменьшается полностью загружается главное меню.
6	Правильность открытия дверей	1. Перейти в любую комнату. 2. Зачистить ее.	Изначально проходы открыты, при переходе в комнату появляются враги и закрываются проходы, после зачистки проходы открываются.

№	Название теста	Действия	Результат
7	Работоспособность меню улучшений	1. Накопить монеты. 2. Во время боя нажать клавишу «Р». 3. Нажать клавишу «Р» после зачистки комнаты. 4. Нажать несколько раз на улучшение скорострельности. 5. Выйти из меню нажав на кнопку «Продолжить». 6. Пострелять.	Во время боя меню улучшений не открывается, после боя меню открывается, при однократном нажатии цена возрастает, а число монет уменьшается, если цена больше, чем имеется монет, ничего не происходит, при стрельбе заметно увеличение скорострельности.
8	Правильность генерации	1. Нажать клавишу «М». 2. Нажать кнопку продолжить. 3. Походить по нескольким комнатам.	На карте отображается один белый квадрат с точкой, один красный квадрат и много пустых белых квадратов, нет слияний 4х маленьких квадратов в один большой, расположение комнат на карте совпадает с реальным.
9	Правильность создания врагов	1. Перейти в любую комнату. 2. Зачистить ее. 3. Выйти в предыдущую комнату. 4. Зайти в ту же комнату.	При входе в комнату появляется один или несколько врагов, при повторном заходе в комнату враги не появляются.
10	Работоспособность меню паузы	1. Нажать клавишу «Esc». 2. Нажать кнопку «Продолжить». 3. Нажать клавишу «Esc». 4. Нажать кнопку «Главное меню».	При нажатии на «Esc» открывается меню паузы, при нажатии на кнопку «Продолжить» в меню паузы игровой процесс возобновляется, при нажатии на кнопку «Главное меню» в меню паузы загружается главное меню.

5.2. Юзабилити тестирование

Данное тестирование производилось при участии друзей, в его ходе были выявлены и исправлены некоторые недочеты:

1) теряющийся на фоне игры системный курсор был заменен на 2 новых – для интерфейса и для геймплея;

2) была скорректирована сложность игры путем изменения характеристик врагов и персонажа;

3) был переработан дизайн для дальнейшего развития продукта.

На рисунке 28 представлены новые курсоры.

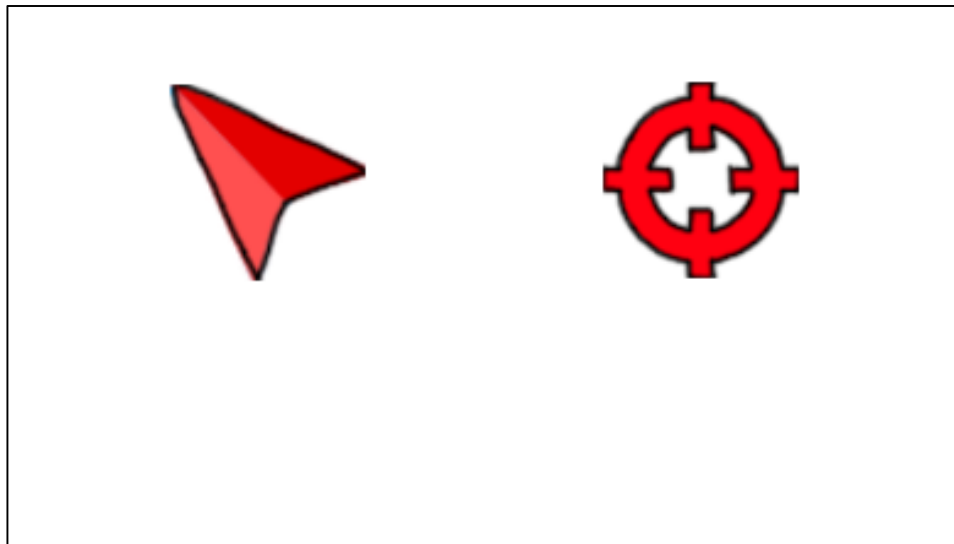


Рисунок 28 – Новые курсоры

В приложении В представлены скриншоты итоговой версии игры после реализации и проведения тестирования.

Вывод по пятому разделу

На основании функциональных требований было проведено функциональное тестирование, в целях корректировки сложности и выявления некоторых недочетов было проведено юзабилити тестирование.

Все функциональные тесты были успешно пройдены, а после юзабилити тестирования были немного изменены характеристики, дизайн и прочие маленькие недочеты.

ЗАКЛЮЧЕНИЕ

В ходе выполнения выпускной квалификационной работы была разработана компьютерная игра в жанре «Top-Down Shooter» на платформе Unity.

Основные результаты работы.

1. Проведен анализ предметной области. Были рассмотрены механики аналогичных проектов, для разрабатываемой игры были выбраны некоторые из них.

2. Выполнено проектирование игры. А именно, были выделены функциональные и нефункциональные требования, сформирована концепция игры и составлена диаграмма вариантов использования.

3. Спроектирована архитектура игры и описаны все компоненты приложения. Также были спроектированы макеты пользовательских интерфейсов.

4. Реализована компьютерная игра. Реализация каждого компонента игры была подробно описана.

5. Проведено функциональное и юзабилити тестирование. Все функциональные тесты были успешно пройдены, а в результате юзабилити тестирования была поправлена сложность и мелкие недочеты.

ЛИТЕРАТУРА

1. Fish C. The History of Video Games. – USA: White Owl, 2021. – 120 с.
2. Steven L. Kent. The Ultimate History of Video Games. – USA: Crown, 2001. – 624 с.
3. Тристан Донован. Играй! История видеоигр. – Москва: Белое яблоко, 2014. – 648 с.
4. TADVISER. [Электронный ресурс] URL: <https://www.tadviser.ru/> (дата обращения: 11.05.2022 г.).
5. Knowable Magazine. [Электронный ресурс] URL: <https://knowablemagazine.org/article/mind/2019/video-games-educational-benefits/> (дата обращения: 11.05.2022 г.).
6. Morgan McGuire, Odest Chadwicke Jenkins. Creating Games: Mechanics, Content, and Technology. – USA: A K Peters/CRC Press, 2008. – 500 с.
7. Brian Ashcraft. Arcade Mania! The Turbo-Charged World of Japan's Game Centers. – USA: Kodansha International, 2008. – 192 с.
8. Alexander Smith. They Create Worlds: The Story of the People and Companies That Shaped the Video Game Industry, Vol. I. – USA: CRC Press, 2019. – 606 с.
9. Страница The Binding of Isaac: Rebirth в steam. [Электронный ресурс] URL: https://store.steampowered.com/app/250900/The_Binding_of_Isaac_Rebirth/ (дата обращения: 12.05.2022 г.).
10. Страница Nuclear Throne в steam. [Электронный ресурс] URL: https://store.steampowered.com/app/242680/Nuclear_Throne/ (дата обращения: 12.05.2022 г.).
11. Gold J. Object-Oriented Game Development. – UK: Pearson Education Limited, 2004. – 440 с.

12. Gregory J. Game Engine Architecture. – USA: A K Peters/CRC Press, 2009. – 864 с.
13. Д. Бонд. Unity и C# геймдев от идеи до реализации. 2-е изд. – СПб.: Питер, 2019. – 928 с.
14. Д. Хокинг. Unity в действии. Мультиплатформенная разработка на C#. – СПб.: Питер, 2016. – 336 с.
15. Руководство Unity. [Электронный ресурс] URL: <https://docs.unity3d.com/ru/530/Manual/> (дата обращения: 25.05.2022 г.).

ПРИЛОЖЕНИЯ

Приложение А. Спецификация вариантов использования

В таблицах 1-4 приведены спецификации основных вариантов использования.

Таблица 1 – Спецификация ВИ «Начать игру»

Вариант использования: начать игру
ID: 1
Аннотация: запуск игрового процесса
Главные актеры: игрок
Второстепенные актеры: нет
Предусловия: отображается главное меню
Основной поток: 1. Вариант использования начинается, когда игрок нажимает на кнопку «Начать игру». 2. Система загружает сцену с игровым процессом.
Постусловия: отображается игровой процесс
Альтернативные потоки: нет

Таблица 2 – Спецификация ВИ «Двигаться»

Вариант использования: двигаться
ID: 2
Аннотация: передвижение персонажа
Главные актеры: игрок
Второстепенные актеры: нет
Предусловия: отображается игровой процесс
Основной поток: 1. Вариант использования начинается, когда игрок нажимает на одну из клавиш «W», «A», «S», «D». 2. Система передвигает персонажа в соответствии с нажатой клавишей. 3. Если персонаж входит в дверь. 3.1. Система перемещает камеру и персонажа в комнату за дверью.
Постусловия: нет
Альтернативные потоки: нет

Таблица 3 – Спецификация ВИ «Стрелять»

Вариант использования: стрелять
ID: 3
Аннотация: стрельба в направлении курсора
Главные актеры: игрок
Второстепенные актеры: нет
Предусловия: отображается игровой процесс
Основной поток: 1. Вариант использования начинается, когда игрок нажимает на ПКМ. 2. Система создает снаряд возле персонажа. 3. Пока снаряд не столкнется с другим объектом. 3.1. Система передвигает снаряд в направлении позиции курсора на момент нажатия на ПКМ. 3.2. Если снаряд столкнулся с другим объектом. 3.2.1. Если этим объектом был враг. 3.2.1.1. Система уничтожает снаряд. 3.2.1.2. Система наносит врагу урон. 3.2.2. Иначе. 3.2.2.1. Система уничтожает снаряд.
Постусловия: нет
Альтернативные потоки: нет

Таблица 4 – Спецификация ВИ «Открыть меню улучшений»

Вариант использования: открыть меню улучшений
ID: 4
Аннотация: открытие меню улучшения характеристик
Главные актеры: игрок
Второстепенные актеры: нет
Предусловия: отображается игровой процесс
Основной поток: 1. Вариант использования начинается, когда игрок нажимает на клавишу «Р». 2. Если в комнате с персонажем нет врагов. 2.1. Система делает активным меню улучшений.
Постусловия: нет
Альтернативные потоки: нет

Приложение Б. Листинги классов

В листингах 1-7 представлен код некоторых классов.

Листинг 1 – Класс RuntimeMenuControl

```
public class RuntimeMenuControl : MonoBehaviour
{
    public Texture2D cursor_game;
    public Texture2D cursor_interface;
    private Vector2 game_hot_spot = new Vector2(32, 32);
    public GameObject pause_menu;
    public GameObject upgrade_menu;
    public GameObject map_menu;
    public Map map;

    private void Update()
    {
        checkPause();
        checkUpgrade();
        checkMap();
    }
    private void checkPause()
    {
        if (Input.GetKeyDown("escape"))
        {
            Cursor.SetCursor(cursor_interface, Vector2.zero,
                             CursorMode.Auto);
            pause_menu.SetActive(true);
            upgrade_menu.SetActive(false);
            map_menu.SetActive(false);
            Time.timeScale = 0f;
        }
    }
    private void checkUpgrade()
    {
        if (Input.GetKeyDown("p"))
        {
            Cursor.SetCursor(cursor_interface, Vector2.zero,
                             CursorMode.Auto);
            if (FindObjectOfType<Enemy>() == null
                && FindObjectOfType<Boss>() == null
                && FindObjectOfType<Minion>() == null)
            {
                upgrade_menu.SetActive(true);
                pause_menu.SetActive(false);
                map_menu.SetActive(false);
                Time.timeScale = 0f;
            }
        }
    }
    private void checkMap()
    {
        if (Input.GetKeyDown("m"))
        {
            Cursor.SetCursor(cursor_interface, Vector2.zero,
                             CursorMode.Auto);
            map_menu.SetActive(true);
            upgrade_menu.SetActive(false);
            pause_menu.SetActive(false);
            map.drawMap();
            Time.timeScale = 0f;
        }
    }
}
```

```

    }
}
public void continueInPausePressed()
{
    Cursor.SetCursor(cursor_game, game_hot_spot, CursorMode.Auto);
    pause_menu.SetActive(false);
    Time.timeScale = 1f;
}
public void continueInUpgradePressed()
{
    Cursor.SetCursor(cursor_game, game_hot_spot, CursorMode.Auto);
    upgrade_menu.SetActive(false);
    Time.timeScale = 1f;
}
public void continueInMapPressed()
{
    Cursor.SetCursor(cursor_game, game_hot_spot, CursorMode.Auto);
    map_menu.SetActive(false);
    Time.timeScale = 1f;
}
public void mainMenuPressed()
{
    SceneManager.LoadScene("Menu");
    Time.timeScale = 1f;
}
}

```

Листинг 2 – Класс Upgrade

```

public class Upgrade : MonoBehaviour
{
    public Player player;
    public Text damage_cost;
    public Text max_health_cost;
    public Text heal_cost;
    public Text firerate_cost;
    public Text count_bullets_cost;
    public HealthBar health_bar;

    public void upDamage()
    {
        float cost;
        cost = Convert.ToSingle(damage_cost.text);
        if (player.money >= cost)
        {
            player.damage *= 1.1f;
            player.changeMoney(-cost);
            cost *= 1.2f;
            damage_cost.text = Convert.ToString(cost);
        }
    }
    public void upFirerate()
    {
        float cost;
        cost = Convert.ToSingle(firerate_cost.text);
        if (player.money >= cost && player.firerate > 0.05)
        {
            player.firerate *= 0.9f;
            player.changeMoney(-cost);
        }
    }
}

```

```

        cost *= 1.2f;
        firerate_cost.text = Convert.ToString(cost);
    }
}
public void upCountBullets()
{
    float cost;
    cost = Convert.ToSingle(count_bullets_cost.text);
    if (player.money >= cost)
    {
        player.count_bullets += 1;
        player.changeMoney(-cost);
        cost *= 2f;
        count_bullets_cost.text = Convert.ToString(cost);
    }
}
public void upMaxHealth()
{
    float cost;
    cost = Convert.ToSingle(max_health_cost.text);
    if (player.money >= cost)
    {
        float health_difference =
            health_bar.full_health * 1.1f - health_bar.full_health;
        health_bar.full_health *= 1.1f;
        player.changeHealth(health_difference);
        player.changeMoney(-cost);
        cost *= 1.2f;
        max_health_cost.text = Convert.ToString(cost);
    }
}
public void heal()
{
    float cost;
    cost = Convert.ToSingle(heal_cost.text);
    if (player.money >= cost && player.health < health_bar.full_health)
    {
        player.changeHealth(health_bar.full_health * 0.05f);
        if (player.health > health_bar.full_health)
            player.health = health_bar.full_health;
        player.changeMoney(-cost);
    }
}
}

```

Листинг 3 – Класс Map

```

public class Map : MonoBehaviour
{
    private float left_panel_coord = 700;
    private float right_panel_coord = 1200;
    private float down_panel_coord = 300;
    private float up_panel_coord = 800;
    private float panel_size_horizontal;
    private float panel_size_vertical;
    public RoomGenerator room_generator;
    private DoorDeleter[,] field;
    private int field_size;
    private float image_size_vertical;
}

```

Продолжение листинга 3 приложения Б

```
private float image_size_horizontal;
public GameObject room;
public GameObject boss_room;
public GameObject room_player;
public GameObject boss_room_player;
public Transform panel;
private Vector3 position_to_spawn;

private void Start()
{
    panel_size_horizontal = left_panel_coord - right_panel_coord;
    panel_size_vertical = up_panel_coord - down_panel_coord;
    field = room_generator.field;
    field_size = room_generator.size;
    image_size_horizontal = panel_size_horizontal / (field_size - 6);
    image_size_vertical = panel_size_vertical / field_size;
    room.transform.localScale =
        new Vector3(image_size_horizontal/100,
            image_size_vertical/100);
    boss_room.transform.localScale =
        new Vector3(image_size_horizontal / 100,
            image_size_vertical / 100);
    room_player.transform.localScale =
        new Vector3(image_size_horizontal / 100,
            image_size_vertical / 100);
    boss_room_player.transform.localScale =
        new Vector3(image_size_horizontal / 100,
            image_size_vertical / 100);
    boss_room_cleared.transform.localScale =
        new Vector3(image_size_horizontal / 100,
            image_size_vertical / 100);
    room_cleared.transform.localScale =
        new Vector3(image_size_horizontal / 100,
            image_size_vertical / 100);
    drawMap();
}

public void drawMap()
{
    for (int col = 3; col < field_size-3; col++)
    {
        for (int line = 3; line < field_size-3; line++)
        {
            if (field[line, col] != null)
            {
                if (field[line, col].isBossRoom
                    && field[line, col].player_in_this_room)
                {
                    position_to_spawn =
                        new Vector3((- (col - 3) * image_size_horizontal
                            + left_panel_coord
                            - image_size_horizontal / 2),
                            ((line) * image_size_vertical
                            + down_panel_coord
                            + image_size_vertical / 2));
                    Instantiate(boss_room_player, position_to_spawn,
                        transform.rotation, panel);
                }
                else if (field[line, col].isBossRoom
                    && !field[line, col].player_in_this_room)
                {

```

Окончание листинга 3 приложения Б

```
        position_to_spawn =
            new Vector3((- (col - 3) * image_size_horizontal
            + left_panel_coord
            - image_size_horizontal / 2),
            ((line) * image_size_vertical
            + down_panel_coord
            + image_size_vertical / 2));
        Instantiate(boss_room, position_to_spawn,
            transform.rotation, panel);
    }
    else if (!field[line, col].isBossRoom
        && field[line, col].player_in_this_room)
    {
        position_to_spawn =
            new Vector3((- (col - 3) * image_size_horizontal
            + left_panel_coord
            - image_size_horizontal / 2),
            ((line) * image_size_vertical
            + down_panel_coord
            + image_size_vertical / 2));
        Instantiate(room_player, position_to_spawn,
            transform.rotation, panel);
    }
    else if (!field[line, col].isBossRoom
        && !field[line, col].player_in_this_room)
    {
        position_to_spawn =
            new Vector3((- (col - 3) * image_size_horizontal
            + left_panel_coord
            - image_size_horizontal / 2),
            ((line) * image_size_vertical
            + down_panel_coord
            + image_size_vertical / 2));
        Instantiate(room, position_to_spawn,
            transform.rotation, panel);
    }
}
}
```

Листинг 4 – Класс Boss

```
public class Boss : MonoBehaviour
{
    private float timer_attack_rate;
    private HealthBar health_bar;
    private Player player;
    public float attack_rate_melee;
    public float super_attack_rate_melee;
    public float minion_spawn_rate;
    public float extension_rate;
    public float extention_percent;
    private float timer_minion_spawn_rate;
    private float changes_attack_rate_melee;
    public float attack_rate_distant;
    public float time_in_condition;
    private float timer_condition;
```

```

public float health;
public float damage;
public float cost;
private int next_condition;
private Vector3 start_position;
public GameObject bullet;
public Transform[] shot_points;
public Minion minion;
private Vector3 room_position;
private Vector3 start_scale;

private void Start()
{
    player = FindObjectOfType<Player>();
    health_bar = GetComponent<HealthBar>();
    health_bar.full_health = health;
    timer_condition = 0;
    changes_attack_rate_melee = attack_rate_melee;
    start_position = transform.position;
    timer_attack_rate = 0.5f;
    timer_minion_spawn_rate = 0.1f;
    start_scale = transform.localScale;
}
private void Update()
{
    generateCondition();
    if (health / health_bar.full_health > 0.5)
    {
        switch (next_condition)
        {
            case 1:
                distant(false);
                break;
            case 2:
                melee(false);
                break;
            case 3:
                minionAndAoe(3, false);
                break;
        }
    }
    else if (health / health_bar.full_health <= 0.5)
    {
        switch (next_condition)
        {
            case 1:
                distant(true);
                break;
            case 2:
                melee(true);
                break;
            case 3:
                minionAndAoe(7, true);
                break;
        }
    }
}
private void minionAndAoe(int count_minions, bool aoe)
{
    if (timer_minion_spawn_rate <= 0)

```

Продолжение листинга 4 приложения Б

```

{
    for (int i = 1; i <= count_minions; i++)
    {
        Instantiate(minion, generateMinionPosition(),
            transform.rotation);
    }
    timer_minion_spawn_rate = minion_spawn_rate;
}
else
    imer_minion_spawn_rate -= Time.deltaTime;
if (aoe)
{
    if (timer_attack_rate <= 0)
    {
        if (transform.localScale.x < 2.5)
            transform.localScale =
                new Vector3(transform.localScale.x
                    + extention_percent,
                    transform.localScale.y + extention_percent, 1f);
        timer_attack_rate = extension_rate;
    }
    else
        timer_attack_rate -= Time.deltaTime;
}
}
private void distant(bool rotate)
{
    if (!rotate)
        if (timer_attack_rate <= 0)
        {
            foreach (Transform shot_point in shot_points)
            {
                Instantiate(bullet, shot_point.position,
                    shot_point.rotation);
            }
            timer_attack_rate = attack_rate_distant;
        }
        else
            timer_attack_rate -= Time.deltaTime;
    else
    {
        if (timer_attack_rate <= 0)
        {
            foreach (Transform shot_point in shot_points)
            {
                Instantiate(bullet, shot_point.position,
                    shot_point.rotation);
                shot_point.rotation *= Quaternion.Euler(0f, 0f, 10f);
            }
            timer_attack_rate = attack_rate_distant;
        }
        else
            timer_attack_rate -= Time.deltaTime;
    }
}
}
private void melee(bool extension)
{
    if (extension)
    {
        if (timer_attack_rate <= 0)

```


Продолжение листинга 4 приложения Б

```
{
    transform.position =
        player.transform.position + generateIncrement();
    player.changeHealth(-damage);
    timer_attack_rate = changes_attack_rate_melee;
    if (changes_attack_rate_melee > super_attack_rate_melee)
        changes_attack_rate_melee -= 0.1f;
}
else
    timer_attack_rate -= Time.deltaTime;
}
else {
    if (timer_attack_rate <= 0)
    {
        transform.position =
            player.transform.position + generateIncrement();
        player.changeHealth(-damage);
        timer_attack_rate = attack_rate_melee;
    }
    else
        timer_attack_rate -= Time.deltaTime;
}
}
private Vector3 generateIncrement()
{
    Vector3 tp_position;
    double grad, x, y;
    int rand;
    bool can_left, can_right, can_down, can_up;
    Vector3 difference_player_position_and_center;

    can_left = true;
    can_right = true;
    can_up = true;
    can_down = true;

    difference_player_position_and_center =
        player.transform.position - room_position;
    if (difference_player_position_and_center.x < -6)
        can_left = false;
    if (difference_player_position_and_center.x > 6)
        can_right = false;
    if (difference_player_position_and_center.y > 2)
        can_up = false;
    if (difference_player_position_and_center.y < -2)
        can_down = false;

    if (!can_left && !can_up)
        grad = UnityEngine.Random.Range(300, 330);
    else if (!can_left && !can_down)
        grad = UnityEngine.Random.Range(30, 60);
    else if (!can_right && !can_up)
        grad = UnityEngine.Random.Range(210, 240);
    else if (!can_right && !can_down)
        grad = UnityEngine.Random.Range(120, 150);
    else if (!can_right)
        grad = UnityEngine.Random.Range(120, 240);
    else if (!can_left)
    {
        rand = UnityEngine.Random.Range(0, 2);
```

Продолжение листинга 4 приложения Б

```
        if (rand == 1)
            grad = UnityEngine.Random.Range(0, 60);
        else
            grad = UnityEngine.Random.Range(300, 360);
    }
    else if (!can_up)
        grad = UnityEngine.Random.Range(210, 330);
    else if (!can_down)
    {
        grad = UnityEngine.Random.Range(30, 150);
    }
    else
    {
        grad = UnityEngine.Random.Range(0, 360);
    }

    grad /= 57.2958;
    x = Math.Cos(grad);
    y = Math.Sin(grad);
    tp_position.x = Convert.ToSingle(x) * 1.7f;
    tp_position.y = Convert.ToSingle(y) * 1.7f;
    tp_position.z = 0f;
    return tp_position;
}
private void generateCondition()
{
    if (timer_condition <= 0)
    {
        timer_condition = time_in_condition;
        next_condition = UnityEngine.Random.Range(1, 4);
        changes_attack_rate_melee = attack_rate_melee;
        transform.position = start_position;
        transform.localScale = start_scale;
    }
    else
    {
        timer_condition -= Time.deltaTime;
    }
}
public void takeDamage(float damage)
{
    health -= damage;
    health_bar.fill = health / health_bar.full_health;
    if (health <= 0)
    {
        Destroy(gameObject);
        player.changeMoney(cost);
    }
}
private void OnTriggerEnter2D(Collider2D collision)
{
    if (collision.CompareTag("Room"))
    {
        room_position = collision.transform.position;
    }
}
private void OnTriggerStay2D(Collider2D collision)
{
    if (collision.CompareTag("Player") && next_condition == 3)
    {

```

```

        player.changeHealth(-damage * 0.1f);
        GetComponent<Rigidbody2D>().WakeUp();
    }
}
private Vector3 generateMinionPosition()
{
    Vector3 minion_position;
    minion_position.x = UnityEngine.Random.Range(-8, 9);
    minion_position.y = UnityEngine.Random.Range(-4, 5);
    minion_position.z = 0;
    minion_position += room_position;
    return minion_position;
}
}

```

Листинг 5 – Класс Minion

```

public class Minion : MonoBehaviour
{
    private Player player;
    public BulletEnemy bullet;
    public Transform shot_point;
    public float health;
    public float firerate;
    private float timer_firate;

    private void Start()
    {
        player = FindObjectOfType<Player>();
    }
    private void Update()
    {
        Vector3 difference =
            player.transform.position - transform.position;
        float angle =
            Mathf.Atan2(difference.y, difference.x) * Mathf.Rad2Deg - 90;
        angle = Random.Range(angle - 45, angle + 45);
        shot_point.rotation = Quaternion.Euler(0f, 0f, angle);
        shot();
    }
    private void shot()
    {
        if (timer_firate <= 0)
        {
            Instantiate(bullet, shot_point.position, shot_point.rotation);
            timer_firate = firerate;
        }
        else timer_firate -= Time.deltaTime;
    }
    public void takeDamage(float damage)
    {
        health -= damage;
        if (health <= 0)
            Destroy(gameObject);
    }
}

```

Листинг 6 – Класс RoomGenerator

```

public class RoomGenerator : MonoBehaviour
{
    public DoorDeleter[] rooms_types_normal;
    public DoorDeleter boss_room;
    public DoorDeleter main_room;
    public DoorDeleter start_room_L;
    public DoorDeleter start_room_R;
    public DoorDeleter start_room_U;
    public DoorDeleter start_room_D;
    private float main_y;
    private float main_x;
    public DoorDeleter[,] field;
    public int size;
    private int count_rooms;
    private OpenDoor open_door;
    List<string> directions_to_del = new List<string>();

    private void Start()
    {
        count_rooms = UnityEngine.Random.Range(30, 70);
        size = calculateSize();
        main_y = main_room.transform.position.y;
        main_x = main_room.transform.position.x;
        field = new DoorDeleter[size, size];
        field[size / 2, size / 2] = main_room;
        field[(size / 2) - 1, (size / 2)] = start_room_D;
        field[(size / 2) + 1, (size / 2)] = start_room_U;
        field[(size / 2), (size / 2) + 1] = start_room_R;
        field[(size / 2), (size / 2) - 1] = start_room_L;
        spawnRooms(count_rooms);
        deleteAllExcessDoors();
        open_door = GetComponent<OpenDoor>();
        Invoke("activateOpenDoor", 0.01f);
    }

    private void activateOpenDoor()
    {
        open_door.doors = open_door.getGeneratedDoors();
    }

    private void spawnRooms(int count_rooms)
    {
        int counter = 0;
        while (counter < count_rooms)
        {
            for (int col = 0; col < size; col++)
            {
                for (int line = 0; line < size; line++)
                {
                    if (field[line, col] != null)
                    {
                        List<int> available_directions = new List<int>();
                        if (field[line, col].have_down_door)
                            available_directions.Add(1);
                        if (field[line, col].have_up_door)
                            available_directions.Add(2);
                        if (field[line, col].have_left_door)
                            available_directions.Add(3);
                        if (field[line, col].have_right_door)

```

Продолжение листинга 6 приложения Б

```
available_directions.Add(4);
int rand = available_directions[Random.Range(0,
available_directions.Count)];

if (rand == 1 && field[line, col].have_down_door
&& field[line - 1, col] == null
&& haveOnlyOneNeighbor(line - 1, col))
{
    if (line >= 3)
    {
        if (counter < count_rooms - 1)
        {
            field[line - 1, col] =
                Instantiate(
                    rooms_types_normal[Random.
                        Range(0,
                            rooms_types_normal.Length)],
                    transform.position,
                    transform.rotation);
            field[line - 1, col].transform.
                position =
                    calculatePosition(line - 1,
                        col);
            counter += 1;
            continue;
        }
        else
        {
            field[line - 1, col] = Instantiate(
                boss_room,
                transform.position,
                transform.rotation);
            field[line - 1, col].transform.
                position =
                    calculatePosition(line - 1,
                        col);
        }
    }
}
if (rand == 2 && field[line, col].have_up_door
&& field[line + 1, col] == null
&& haveOnlyOneNeighbor(line + 1, col))
{
    if (line < size - 3)
    {
        if (counter < count_rooms - 1)
        {
            field[line + 1, col] =
                Instantiate(
                    rooms_types_normal[Random.
                        Range(0,
                            rooms_types_normal.Length)],
                    transform.position,
                    transform.rotation);
            field[line + 1, col].transform.
                position =
                    calculatePosition(line + 1,
                        col);
        }
    }
}
```

Продолжение листинга 6 приложения Б

```
        counter += 1;
        continue;
    }
    else
    {

        field[line + 1, col] = Instantiate(
            boss_room,
            transform.position,
            transform.rotation);
        field[line + 1, col].transform.
            position =
                calculatePosition(line + 1,
                    col);
        return;
    }
}

if (rand == 3 && field[line, col].have_left_door
    && field[line, col - 1] == null
    && haveOnlyOneNeighbor(line, col - 1))
{
    if (col >= 3)
    {
        if (counter < count_rooms - 1)
        {

            field[line, col - 1] =
                Instantiate(
                    rooms_types_normal[Random.
                        Range(0,
                            rooms_types_normal.Length)],
                    transform.position,
                    transform.rotation);
            field[line, col - 1].transform.
                position =
                    calculatePosition(line,
                        col - 1);
            counter += 1;
            continue;
        }
        else
        {

            field[line, col - 1] = Instantiate(
                boss_room,
                transform.position,
                transform.rotation);
            field[line, col - 1].transform.
                position =
                    calculatePosition(line,
                        col - 1);
        }
    }
}

if (rand == 4 && field[line, col].have_right_door
    && field[line, col + 1] == null
    && haveOnlyOneNeighbor(line, col + 1))
{
```

Продолжение листинга 6 приложения Б

```

        if (col < size - 3)
        {
            if (counter < count_rooms - 1)
            {

                field[line, col + 1] = Instantiate(
                    rooms_types_normal[Random.Range(0,
                        rooms_types_normal.Length)],
                    transform.position,
                    transform.rotation);
                field[line, col + 1].transform.
                    position =
                        calculatePosition(line,
                            col + 1);
                counter += 1;
                continue;
            }
            else
            {

                field[line, col + 1] = Instantiate(
                    boss_room,
                    transform.position,
                    transform.rotation);
                field[line, col + 1].transform.
                    position =
                        calculatePosition(line,
                            col + 1);

                return;
            }
        }
    }
}

private Vector3 calculatePosition(int line, int col)
{
    Vector3 position;
    position.x = main_x + (col - size / 2) * 18;
    position.y = main_y + (line - size / 2) * 10;
    position.z = 0;
    return position;
}

private bool haveOnlyOneNeighbor(int line, int col)
{
    int count_neighbor = 0;
    if (field[line, col + 1] != null) count_neighbor += 1;
    if (field[line, col - 1] != null) count_neighbor += 1;
    if (field[line + 1, col] != null) count_neighbor += 1;
    if (field[line - 1, col] != null) count_neighbor += 1;
    if (count_neighbor == 1) return true;
    else return false;
}

private void deleteAllExcessDoors()
{
    for (int col = 0; col < size; col++)
    {
        for (int line = 0; line < size; line++)

```

```

        {
            if (field[line, col] != null)
            {
                if (field[line, col].have_down_door
                    && field[line - 1, col] == null)
                    directions_to_del.Add("down");
                if (field[line, col].have_up_door
                    && field[line + 1, col] == null)
                    directions_to_del.Add("up");
                if (field[line, col].have_right_door
                    && field[line, col + 1] == null)
                    directions_to_del.Add("right");
                if (field[line, col].have_left_door
                    && field[line, col - 1] == null)
                    directions_to_del.Add("left");
                field[line, col].deleteDoors(directions_to_del);
                directions_to_del = new List<string>();
            }
        }
    }
}

```

Листинг 7 – Класс DoorDeleter

```

public class DoorDeleter : MonoBehaviour
{
    public bool isBossRoom = false;
    public bool player_in_this_room = false;
    public bool isRoomCleared = false;
    public GameObject left_door;
    public GameObject right_door;
    public GameObject up_door;
    public GameObject down_door;
    public GameObject left_mover;
    public GameObject right_mover;
    public GameObject up_mover;
    public GameObject down_mover;
    public GameObject wall;
    public bool have_left_door = true;
    public bool have_right_door = true;
    public bool have_down_door = true;
    public bool have_up_door = true;

    public void deleteDoors(List<string> directions)
    {
        foreach (string direction in directions)
        {
            switch (direction)
            {
                case "left":
                    Instantiate(wall,
                        left_door.transform.position,
                        left_door.transform.rotation);
                    Destroy(left_door);
                    Destroy(left_mover);
                    have_left_door = false;
                    break;
                case "up":

```



```

        Instantiate(wall,
            up_door.transform.position,
            up_door.transform.rotation);
        Destroy(up_door);
        Destroy(up_mover);
        have_up_door = false;
        break;
    case "right":
        Instantiate(wall,
            right_door.transform.position,
            right_door.transform.rotation);
        Destroy(right_door);
        Destroy(right_mover);
        have_right_door = false;
        break;
    case "down":
        Instantiate(wall,
            down_door.transform.position,
            down_door.transform.rotation);
        Destroy(down_door);
        Destroy(down_mover);
        have_down_door = false;
        break;
    }
}
}
private void OnTriggerStay2D(Collider2D collision)
{
    if (collision.CompareTag("Player"))
    {
        player_in_this_room = true;
    }
}
private void OnTriggerExit2D(Collider2D collision)
{
    if (collision.CompareTag("Player"))
    {
        isRoomCleared = true;
        player_in_this_room = false;
    }
}
}

```

Приложение В. Скриншоты итоговой версии игры

На рисунках 1-7 представлены скриншоты итоговой версии игры.

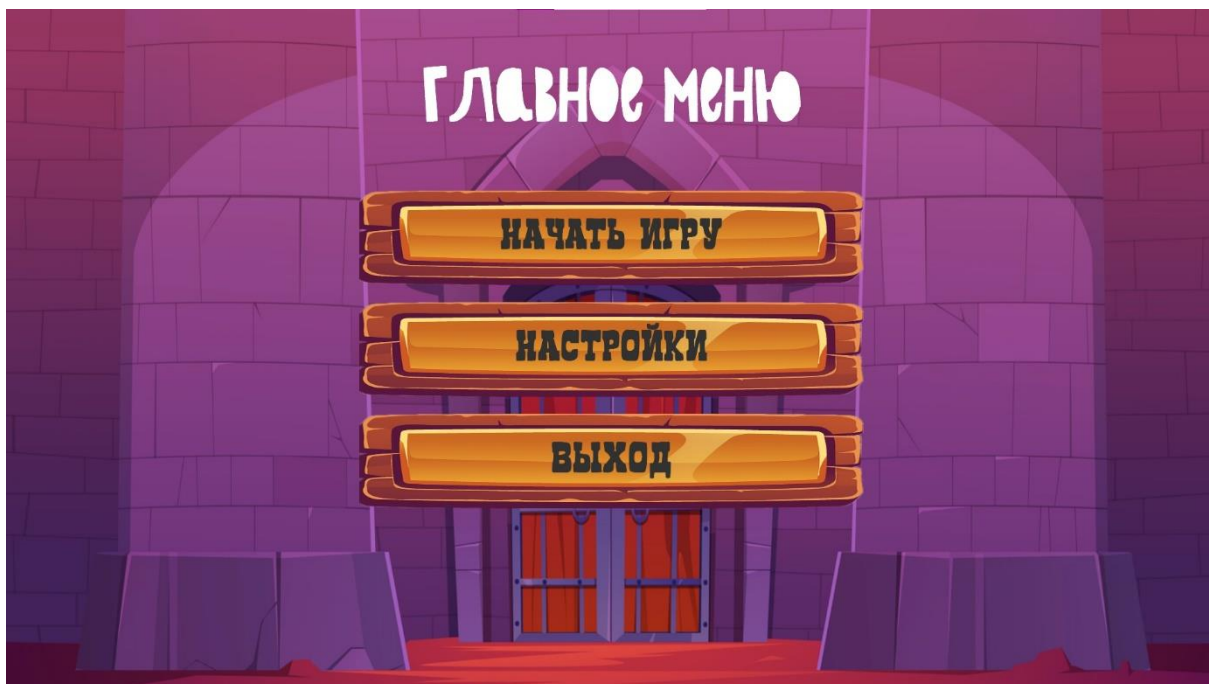


Рисунок 1 – Главное меню

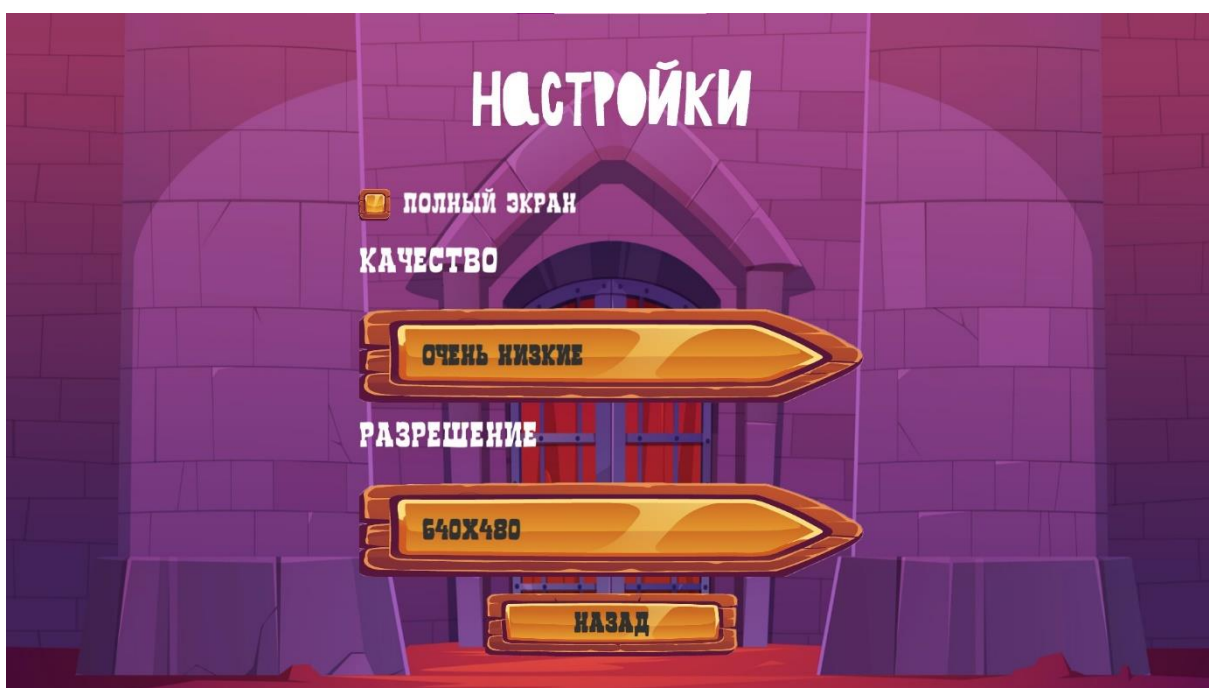


Рисунок 2 – Меню настроек



Рисунок 3 – Карта

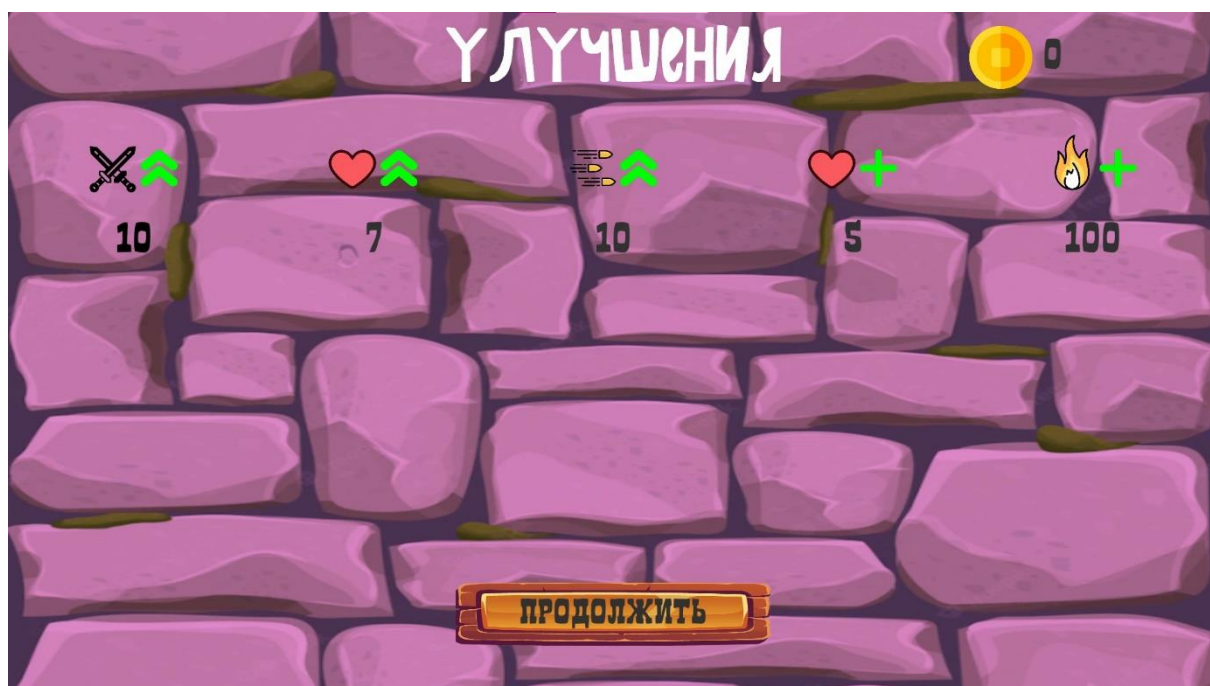


Рисунок 4 – Меню улучшений

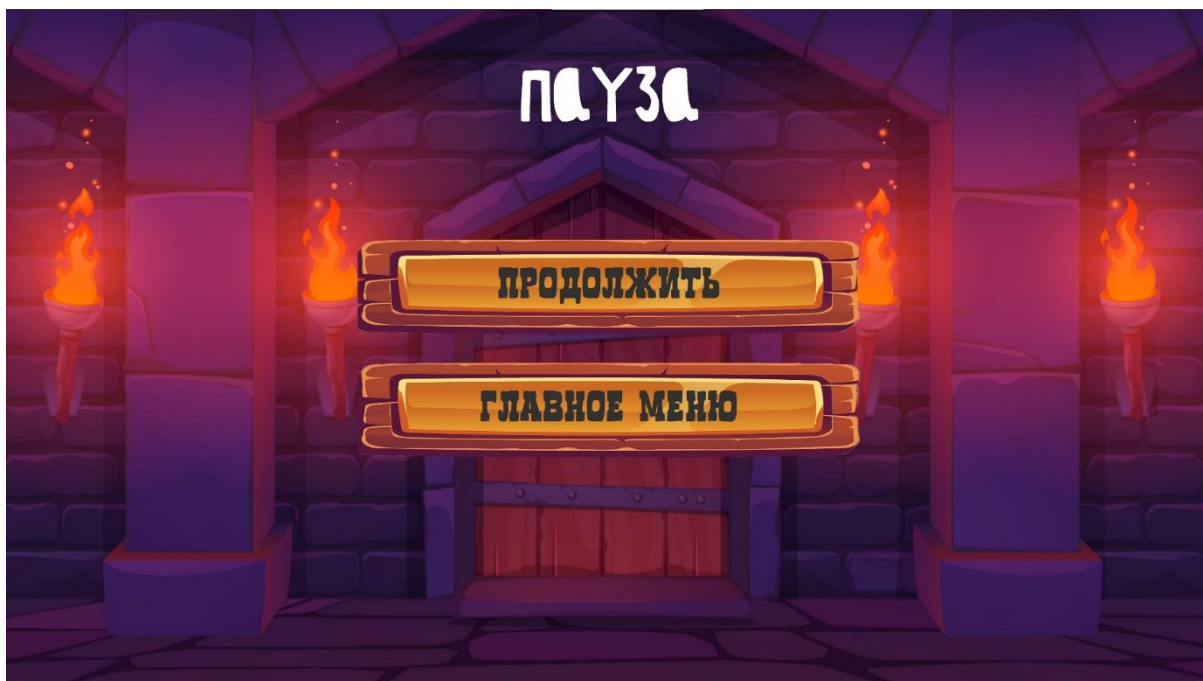


Рисунок 5 – Меню паузы

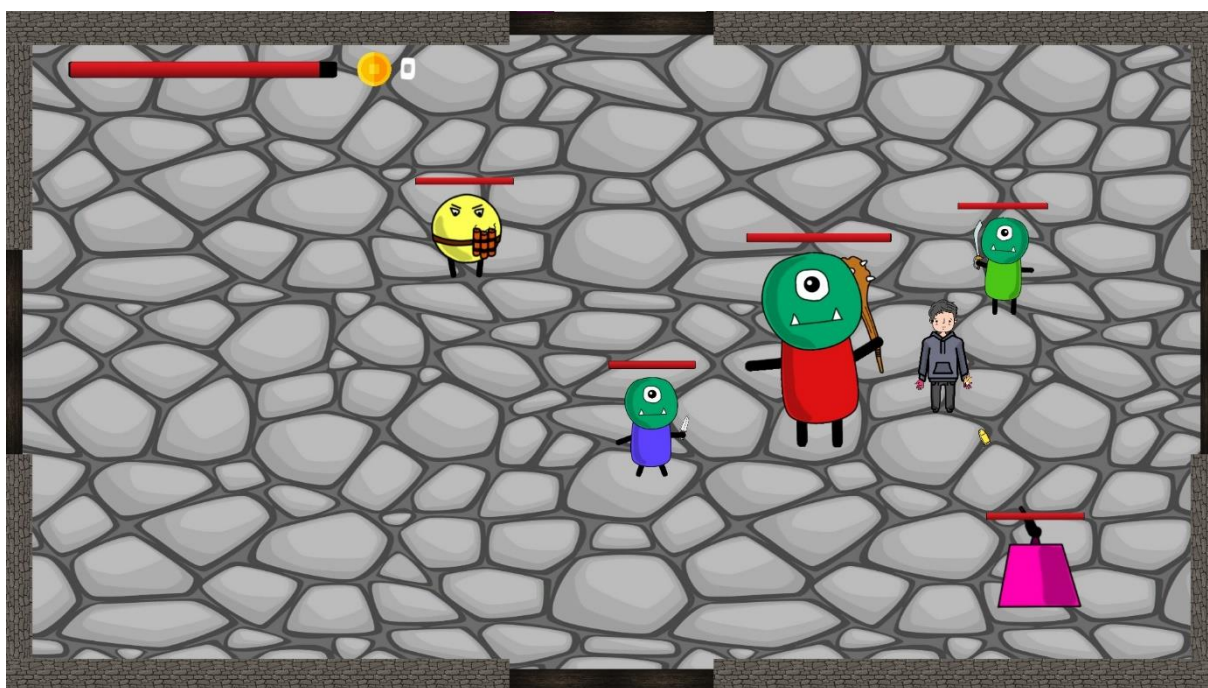


Рисунок 6 – Обычная комната



Рисунок 7 – Комната босса