МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ

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

Факультет	*Кафедра*
вычислительной техники	МОиПЭВМ

Направление подготовки 09.03.04 «Программная инженерия»

БАКАЛАВРСКАЯ РАБОТА

на тему

Программное игровое приложение с использованием мультиплатформенной инструментальной среды Unity

Студент _____ Костерев Григорий Андреевич
(подпись, дата) (ФИО полностью)

Руководитель _____ Князев В.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, 2022

Федеральное государственное бюджетное образовательное учреждение высшего образования
ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕР СИТЕТ
ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет	*Кафедра*
Вычислительной техники	МО и ПЭВМ

«Утверждаю»
Заведующий кафедрой
_____П.П. Макарычев
«__»_____202 г

ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ
БАКАЛАВРА

1. Студент Костерев Григорий Андреевич гр. 18ВП2 факультета ВТ
<Фамилия, имя, отчество полностью>

2.Тема работы Программное игровое приложение с использованием мультиплатформенной инструментальной среды Unity

Тема утверждена приказом ПГУ № № 320/О от “ 18 ” апреля 2021

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

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

1) Разработать игровое приложение в жанре логических игр puzzle с использованием средств мультиплатформенной разработки Unity.
2) Среда выполнения приложения: операционная система Windows 10 и выше
Языковые средства разработки: C#, UML
Инструментальные средства разработки: Microsoft Visual Studio 2019, Unity 2020.3f1, Microsoft Project, ArgoUML
Технологии разработки: RUP, ООП
3) Основные функции игрового приложения:
а) выбор уровня и начало игры;
б) завершение уровня с разными результатами;
в) создание собственного уровня пользователем;
г) просмотр статистики завершенных игр.

5. Перечень подлежащих разработке вопросов

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

6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1	*Анализ предметной области постановка задачи*	15%	13.04.2022–20.04.2022	
2	*Проектирование*	25%	21.04.2022–05.05.2022	
3	*Реализация*	40%	06.05.2022–30.05.2022	
4	*Контроль качества программного обеспечения*	5%	31.05.2022–02.06.2022	
5	*Оформление пояснительной записки*	15%	03.06.2022–17.06.2022	

Дата выдачи задания "22" декабря 2021

Руководитель бакалаврской работы _____
(подпись, дата)

Задание к исполнению принял студент _____
(подпись, дата)

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

Реферат

Пояснительная записка содержит 109 листов, 51 рисунка, 18 таблиц, 33 использованных источников, 3 приложения.

ИГРОВОЕ ПРИЛОЖЕНИЕ, ЛОГИЧЕСКАЯ ИГРА В ЖАНРЕ PUZZLE, WINDOWS 10, UNITY, C#, RUP, UML

Объектом разработки является игровое приложение с использованием средств мультиплатформенной разработки Unity.

Цель работы – разработка игрового приложения в жанре логических игр puzzle с использованием средств мультиплатформенной разрабоки Unity. Игровое приложение позволяет выбрать уровень и начать игру, завершить уровень, создать новый уровень, а также просмотривать статистику предыдущих завершенных игр.

Разработка приложения производилась на языке программирования C#. В качестве среды разработки использовалась среда программирования Microsoft Visual Studio, операционная система Windows 10. Приложение предназначено для использования под управлением ОС Windows 10.

Содержание

Введение 6
1 Анализ предметной области и требований 8
1.1 Анализ предметной области и постановка задачи 8
1.2 Анализ программных продуктов аналогичного назначения 11
1.2.1 Игровое приложение в жанре логических игр «Boulder Dash – 30th Anniversary» 11
1.2.2 Игровое приложение в жанре логических игр «Boulder Dash – Deluxe» 12
1.2.3 Игровое приложение в жанре логических игр «Boulder Dash – XL» 13
1.2.4 Сравнение аналогов 14
1.3 Анализ целевой аудитории 18
1.4 Анализ функциональных требований 20
1.5 Анализ нефункциональных требований 24
1.6 Планирование разработки и оценка бюджета 25
2 Проектирование игрового приложения 33
2.1 Выбор технологий и инструментальных средств разработки 33
2.2 Проектирование архитектуры программного обеспечения 34
2.3 Проектирование интерфейса пользователя 36
3 Реализация игрового приложения 40
3.1 Реализация клиентской части игрового приложения 40
4 Тестирование программного продукта 42
4.1 Тестирование черным ящиком 42
4.1.1 Классы эквивалентности 42
4.1.2 Граничные условия 43
4.1.3 Тест-кейсы 43
4.2 Функциональное тестирование белым ящиком алгоритма передвижения 44
4.2.1 Алгоритм 44
4.2.2 Покрытие операторов 46
4.2.3 Покрытие условий 46
4.2.4 Покрытие маршрутов 47

4.3 Метрики кода 47
4.3.1 Цикломатическая сложность 48
4.3.2 Метрики размера программы 49
4.3.3 Метрики Холстеда 49
Заключение 53
Список использованных источников 54
Приложение А. Глоссарий 58
Приложение Б. Исходный код игрового приложения 60
Приложение Е. Результаты тестирования игрового приложения 97

Введение

В наше время информационные технологии все шире распространяются на разнообразные сферы человеческой деятельности. С каждым годом по всему миру компьютеры все активнее используют в различных отраслях экономики, и это кардинально меняет жизнь людей.

Не остается в стороне от этой тенденции и наша страна. В ней также взят курс на максимальное внедрение информационных технологий во всех сферах деятельности, который поддерживается на государственном уровне. 9 мая 2017 года была утверждена «Стратегия развития информационного общества в Российской Федерации на 2017 – 2030 годы». В ней были поставлены основные цели и задачи политики Российской Федерации в сфере информационных технологий, а также объявлен курс на формирование национальной цифровой экономики [1]. С этой же целью в том же 2017 году была принята государственная программа «Цифровая экономика». Она призвана обеспечить «развитие цифровой экономики Российской Федерации, в которой данные в цифровой форме являются ключевым фактором производства во всех сферах экономической деятельности, что повышает конкурентоспособность страны, качество жизни граждан, обеспечивает экономический рост и экономический суверенитет» [2].

На сегодняшний день, индустрия компьютерных игр (или интерактивных развлечений) является одной из наиболее активно развивающихся и перспективных направлений в медиаиндустрии. С финансовой точки зрения, она уже вплотную подходит к сборам от реализации продукции киноиндустрии. Технологии разработки доступны для всех и позволяют даже одному разработчику создать свою игру, не прибегая к чужой помощи. Сейчас большинство проектов разрабатываются при помощи игровых движков – специальных инструментов для разработки, которые значительно упрощают создание программной части. Однако, несмотря на большое количество многофункциональных инструментов, от разработчика всё равно требуется очень большое количество труда, чтобы создать действительно хороший продукт.

На данный момент одно из самых популярных и эффективных решений предоставляет компания Unity Technologies [3], которая в июне 2005 года выпустила платформу разработки для создания 2D и 3D игр – Unity. Платформа Unity обладает широким спектром возможностей, удобным и интуитивно понятны интерфейсом. Большим преимуществом Unity является кроссплатформенная разработка, которая делает легким и быстрым портирование игр под такие платформы как Windows, iOS, Android, Windows Phone 8, а также разрабатывать игры для Xbox, PS, Nintendo, веб-браузеров и даже VR.

Таким образом, на основании приведенных выше факторов можно сделать вывод о том, что разработка игрового приложения весьма актуальна и перспективна в условиях современного общества.

1 Анализ предметной области и требований

1.1 Анализ предметной области и постановка задачи

Видеоигра – это компьютерная программа, главная суть которой во взаимодействии человека и компьютерного устройства с помощью средств ввода и вывода информации, например посредством клавиатуры и мыши происходит управление игровым персонажем, а посредством монитора или телевизора игровое приложение сообщает текущее состояние игры. Игровой процесс или геймплей (англ. gameplay) определяет, как игроки взаимодействуют с игровым миром, и как мир игры отвечает на совершенные игроками действия.

Компьютерные игры могут быть классифицированы по нескольким признакам:

– жанр;

– количество игроков и способ их взаимодействия;

– визуальное представление;

– платформа.

Нет единой классификации игр по жанрам, ввиду огромной комплексности и разнообразии вариантов взаимодействия элементов между друг другом. Тем не менее множество представителей индустрии видеоигр выделяют следующие жанры компьютерных игр [4]:

– экшн (action) – в игровом процессе основной акцент ставят на битвы и соревновании игрока с неигровыми персонажами, либо соревновании игрока с другими игроками;

– ролевые (role playing game, RPG) – позволяют вжиться в роль и отыгрывать игрового персонажа, как если бы сам игрок непосредственно находился внутри игрового мира;

– стратегии (strategy) – требуют от игрока выработки стратегии и достижения определенной цели, а игровой процесс данного типа игр обычно строится на сборе необходимых ресурсов, захват территорий и уничтожении базы соперника;

– симуляторы (simulator) – симулируют один или несколько процессов реального мира;

– приключенческие игры (adventure) – характеризуют полноценный литературный сюжет, на котором построен мир игры, а игровой процесс предлагает игроку решать множество задач и головоломок;

– логические игры – требуют от игрока изучение неких правил, поиск закономерностей и комбинаций с целью достижения результата.

Альтернативную классификацию по жанрам на основании собственных исследований предлагает А. Киризлеев [5]. Он выделил три группы игр по критерию выполняемых действий:

– игры общения: получение информации, общение, изучение мира;

– игры действий: перемещение в пространстве, использование оружия и техники;

– игры контроля: командование, управление, распределение материальных средств.

Внутри групп задаются границы, определяющие дальнейшую детализацию с помощью уточняющих критериев и с использованием устоявшихся названий категорий. Такой подход кажется более рациональным и системным, хотя и более сложным. Схема представлена на рисунке 1.

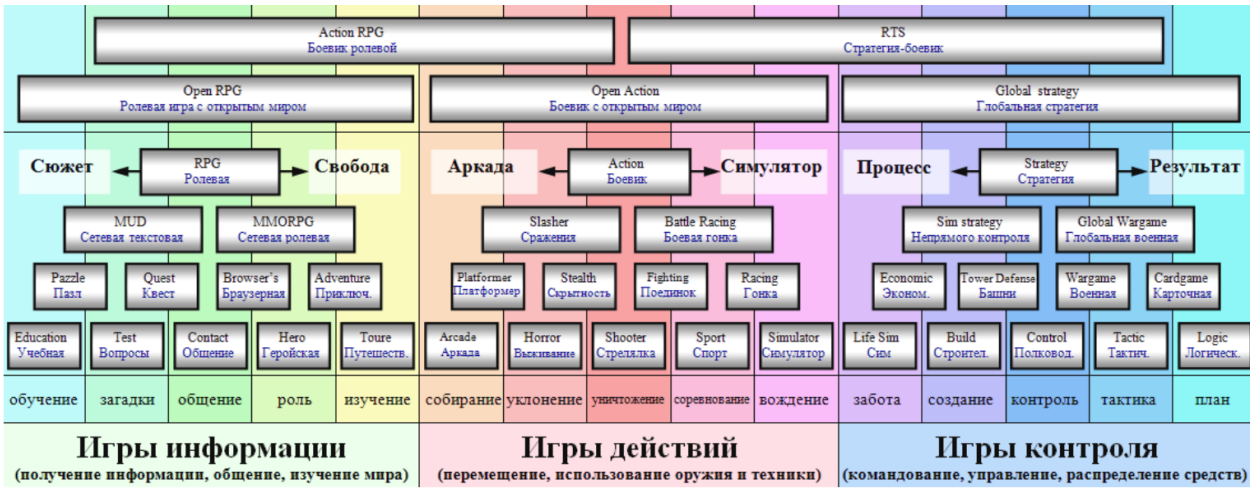

Рисунок 1 – Классификация игр по жанрам Александра Киризлеева

Классификация по количеству игроков:

– одиночные (синглплеер, англ. singleplayer) – рассчитаны на игру в одиночку, против компьютера;

– многопользовательские (англ. multiplayer) – рассчитаны на игру нескольких человек по локальной сети, модему или Интернету.

В классификации по игровым платформам выделяют:

– игровые консоли и приставки;

– персональные компьютеры;

– мобильные телефоны и КПК.

В качестве визуального оформления игра может либо использовать графические материалы, либо быть текстовой. Игра также может быть двухмерной или трёхмерной. Кроме того, существуют примеры звуковых игр, в которых вместо визуального представления используются звуки.

На основе проведенных исследований было принято решение разработать одиночную игру в жанре логических игр с двухмерным стилем визуального оформления. Основной платформой является Windows 10. На данный момент Windows 10 – самая популярная операционная система для персональных компьютеров, а логические игры являются достаточно популярным жанром и позволяют не только приятно, но и полезно провести время. Двухмерный визуальный стиль позволит сократить время разработки, обеспечить большую скорость работы игры и увеличить диапазон поддерживающих устройств.

Приложение должно давать пользователю возможность выбора игрового уровня. Пользователь должен иметь возможность начать и завершить уровень, а также выйти из приложения. Также было принято решение добавить дополнительные функциональные возможности: просмотр результатов предыдущих игр и возможности создании пользовательского уровня.

Таким образом, приложение имеет следующие основные функции:

– выбор уровня для начала игры;

– возможность завершить уровень;

– создание пользовательского уровня;

– просмотр сведений своих предыдущих игр.

1.2 Анализ программных продуктов аналогичного назначения

В настоящее время спроектировано и разработано целый ряд программных игровых приложений в жанре «Головоломка». Рассмотрим наиболее известные приложения такого рода и произведем для них сравнительный анализ.

1.2.1 Игровое приложение в жанре логических игр «Boulder Dash – 30th Anniversary»

Boulder Dash – 30th Anniversary [6]. Данная игра является самым популярным игровым приложением серии Boulder Dash, разработанной BBG Entertainment GmbH в 2016 году.

Особенностью данного приложение является большой список доступных уровней и встроенный редактор карт. В игре есть множество игровых механик, напрямую не связанных с оригинальной серией, как например выбор игрового персонажа.

В игре есть несколько видов неигровых персонажей под управлением компьютера, которые могут убить игрового персонажа, из-за чего игроку придётся начинать уровень заново.

Справа вверху пользователь может посмотреть количество очков, снизу по центру игроку доступен набор навыков, а слева вверху – количество собранных сундуков и бриллиантов. Таймер сверху ведет подсчет секунд, затраченных на прохождения уровня. Иконки занимают много места на экране, а потому мешают погружению в игровой процесс.

Во время игрового процесса идёт подсчёт очков, которые пользователь может получить, совершив определённое действие в игре: уничтожив соперника в игре, собрав ценный предмет.

Скриншот игрового процесса представлен на рисунке 2.

Рисунок 2 – Скриншот игрового процесса Boulder Dash – 30th Anniversary

Проект оказался успешный, однако, вскоре после выхода игры в продажу, разработчики прекратили поддержку обновлениями и проект быстро потерял аудиторию.

1.2.2 Игровое приложение в жанре логических игр «Boulder Dash – Deluxe»

Boulder Dash – Deluxe [7]. Разработанная также BBG Entertainment GmbH в 2021 году, по игровому процессу и интерфейсу является копией Boulder Dash – 30th Anniversary.

Однако, несмотря на более позднюю дату выхода, по сравнению с аналогом не имеет встроенного редактора карт, в своём распоряжении у нее меньше уровней. Из-за данных факторов уменьшается вовлечённость игрока по мере прохождения игры.

Интерфейс во много схож с предыдущей игрой серии, за исключением месторасположения умений, теперь они располагаются с левой стороны экрана. Справа вверху пользователь может посмотреть количество очков, слева вверху – количество собранных сундуков и бриллиантов. Таймер сверху ведет подсчет секунд, затраченных на прохождение уровня. Иконки

занимают много места на экране, а потому мешают погружению в игровой процесс. Скриншот игрового процесса представлен на рисунке 3.

Рисунок 3 – Скриншот игрового процесса Boulder Dash – Deluxe

Проект опробовали всего несколько тысяч человек. Отсутствие редактора карт, посредственно реализованные 3d модели объектов в игре и быстрое прекращение поддержки обновлениями сыграли важную роль.

1.2.3 Игровое приложение в жанре логических игр «Boulder Dash – XL»

Boulder Dash – XL [8] – логическая игра от студии Catnip Games вышедшая в 2013 года, как и ранее рассмотренные аналоги является частью серии Boulder Dash.

В отличие от своих аналогов имеет более минималистичный интерфейс – игровой счёт, время, количество собранных алмазов расположены справа вверху и никак не мешают игровому процессу. Сам игровой процесс немного проще. За отведенное время в игре герою предстоит собрать все алмазы, которые разбросаны по уровню. Скриншот игрового процесса представлен на рисунке 4.

Рисунок 4 – Скриншот игрового процесса Boulder Dash – XL

Игровое приложения не снискало популярности, так как оно является эксклюзивом платформы Nintendo 3DS [9], кроме того, в нём нет редактора карт.

1.2.4 Сравнение аналогов

Рассмотрев подробно каждый из аналогов, были выявлены основные функции, которые должны присутствовать в каждом разрабатываемом игровом приложении в жанре головоломка.

Главными элементами, которые должны находиться в каждом приложении, являются:

– игровой персонаж, которым управляет игрок;

– элементы интерфейса позволяющие узнать текущие достижения пользователя;

– неигровые персонажи, способные при заданных условиях завершить прохождение уровня досрочно;

– один или несколько вариантов прохождения уровня;

– выбор уровня из списка.

Игра даёт пользователю персонажа, которым тот может управлять и взаимодействовать с объектами на карте согласно игровому дизайну конкретного проекта. Интерфейс, который позволяется узнать текущие состояние игры. Текущей счёт, время, пройденное с момента начала уровня, количество собранных ценных ресурсов.

Неигровые персонажи под управлением компьютера. Данные персонажи должны влиять на игрового персонажа, а также на игровой процесс, с возможностью досрочного окончания уровня.

Один или несколько вариантов прохождения уровня. Игра должна давать возможность успешно завершить уровень.

Выбор уровня из списка. Интерфейс игры предусматривает выбор карты для прохождения из списка карт.

Среди критериев, по которым была проведена сравнительная характеристика аналогов, были выделены следующие:

– удобство интерфейса;
– вовлечённость пользователя;
– производительность;
– функциональность.

Рассмотрим подробно каждый из приведенных выше критериев.

Критерий удобства интерфейса показывает, насколько эффективно конечный пользователь будет пользоваться основными функциями, предоставляемыми игровым приложением, и насколько быстро пользователь будет ориентироваться в программе. Оценка критерия от 1 до 5. Расшифровка оценок критерия представлена в таблице 1.

Таблица 1 – Расшифровка оценок критерия «Удобство интерфейса»

Оценка	Описание
1	Интерфейс максимально неудобен для использования, его элементы очень трудно воспринимать, и они мешают игровому процессу.
2	Удобство интерфейса на низком уровне, элементы расположены в порядке, затрудняющем взаимодействие с ними, а также элементы мешают восприятию игрового процесса.
3	Удобство интерфейса на удовлетворительном уровне, имеются небольшие затруднения при взаимодействии с его элементами, и они немного мешают восприятию игрового процесса.

Продолжение таблицы 1.

Оценка	Описание
4	Удобство интерфейса на хорошо уровне, имеются незначительные затруднения при взаимодействии с его элементами, но в целом интерфейс не мешает восприятию игрового процесса.
5	Интерфейс максимально удобен для использования, нет затруднений с восприятием его элементов и их взаимодействием.

Критерий вовлечённости пользователя показывает, насколько сильно проект способен захватить внимание игрока. Оценка критерия от 1 до 5. Расшифровка оценок критерия приведена в таблице 2.

Таблица 2 – Расшифровка оценок критерия «Вовлечённость пользователя»

Оценка	Описание
1	Игровое приложение не способно захватить внимание игрока.
2	Отдельные игровые события вы игре способны захватить внимание игрока, однако по мере использования приложения пользователь быстро теряет интерес.
3	Игровое приложение в целом справляется со своей задачей, однако далеко не все игровые события способны удержать внимание игрока.
4	Игровое приложение успешно захватывает внимания игрока, но некоторые игровые события не способно его удерживать.
5	Игровое приложение крайне успешно удерживает внимания игрока.

Критерий производительности показывает, будет ли возникать проблемы с восприятием игрового процесса на всём спектре доступных устройств. Оценка критерия от 1 до 5. Расшифровка оценок представлена в таблице 3.

Таблица 3 – Расшифровка оценок критерия «Производительность»

Оценка	Описание
1	Достигнуть приемлемого уровня восприятия игрового процесса невозможно на всём доступном спектре доступных устройств.
2	Приемлемый уровень восприятия игрового процесса достигается только на новых мощных устройствах.
3	Приемлемый уровень восприятия игрового процесса достижим на устройствах среднего класса.
4	Проблемы возникают только на самых старых устройствах.
5	Комфортный уровень восприятия игрового процесса возможен на всём спектре доступных устройств.

Критерий функциональности показывает набор возможностей (функций), которые предоставляет данное игровое приложение. Оценка критерия от 1 до 5. Расшифровка оценок приведена в таблице 4.

Таблица 4 – Расшифровка оценок критерия «Функциональность»

Оценка	Описание
1	Приложение не предоставляет минимальный набор необходимых функций.
2	Основные функции представлены в приложении, однако их использование затруднительно или они плохо взаимодействуют друг с другом.
3	Основные функции представлены в приложении.
4	Приложение предоставляет весь основной функционал, а также предлагает свой набор дополнительных возможностей.
5	Приложение предоставляет весь основной функционал, набор его дополнительных возможностей предоставляет отличную гибкость в использовании приложения.

Результаты сравнения программ-аналогов по представленным выше характеристикам приведены в таблице 5.

Таблица 5 – Сравнительная характеристика продуктов-аналогов

Критерий	Boulder Dash – 30th Anniversary	Boulder Dash – Deluxe	Boulder Dash – XL	Разрабатываемое приложение
Удобство интерфейса	3	3	4	4
Вовлечённость пользователя	3	2	2	4
Производительность	4	4	5	4
Функциональность	4	3	3	4

В результате сравнения программ-аналогов можно сказать, что ни одна из рассмотренных программ не обладает всем характеристиками, необходимыми для успешной реализации продукта на рынке. Удобство интерфейса находится на хорошем уровне среди всех представленных программ, однако интерфейс «Boulder Dash – XL» отличился отсутствием негативного влияния на погружение в игровой процесс. Лучше всех в игровой процесс вовлекает «Boulder Dash – 30th Anniversary», однако данное приложение показывает лишь средний результат по данному критерию. Все

приложения показали хорошую производительность на всем спектре доступных устройств, но стоит выделить «Boulder Dash – XL», которая хоть и является эксклюзивом Nintendo 3DS, но все же показывает отличную производительность. Больше всех функций предоставляет «Boulder Dash – 30th Anniversary», данное приложение предлагает собственный редактор карт, а также количество уровней и механик больше, чем у аналогов. Худшим приложением по критерию функциональности является «Boulder Dash – XL», у которого реализованы исключительно основные функции игрового приложения. Каждое из представленных игровых приложений не может хорошо удерживать внимание игрока.

Исходя из вышеизложенного анализа, было принято решение создать своё игровое приложение в жанре логических игр (головоломка), лишенного недостатков своих аналогов.

1.3 Анализ целевой аудитории

Целевой аудиторией проекта было принято решение сделать пользователей, играющие в игры на персональном компьютере. График доходности игровой индустрии [10], сортированный по игровой платформе, вы можете увидеть на рисунке 5.

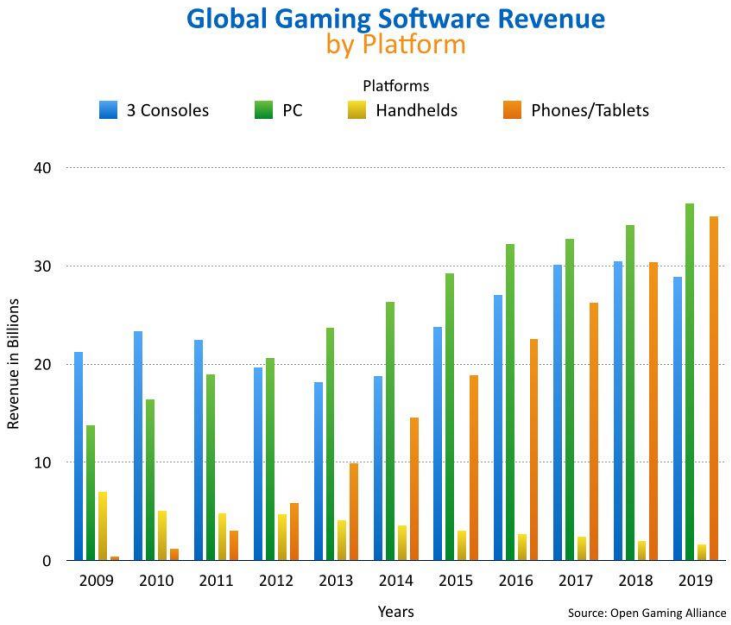

Рисунок 5 – График доходности игровой индустрии по игровой платформе

Как мы можем видеть на графике, рынок персональных компьютеров находится на первом месте по прибыльности среди игровых платформ и имеет постоянный рост. А также разработка под персональные компьютеры быстрее и проще, чем под другие системы, из-за огромного места под игровой интерфейс, большого удобства управления клавиатурой и мыши, а также большой мощности систем. Также режим редактирования уровня, который планируется в игровом приложении, лучше всего реализуется именно под управлением компьютера.

Предоставляемая игровым ресурсов Steam [11] статистика распространения операционных систем для персональных компьютеров представлена на рисунке 6.

САМЫЙ ПОПУЛЯРНЫЙ	ДОЛЯ	ДИНАМИКА
Windows	96.68%	+0.37%
Windows 10 64 bit	73.89%	+0.34%
Windows 11 64 bit	19.59%	+0.65%
Windows 7 64 bit	2.41%	-0.52%
Windows 8.1 64 bit	0.53%	-0.04%
Windows 7	0.12%	-0.05%
Windows 10	0.05%	-0.02%
OSX	2.20%	-0.35%
MacOS 12.3.1 64 bit	0.85%	+0.39%
MacOS 12.2.1 64 bit	0.18%	-0.26%
MacOS 10.15.7 64 bit	0.18%	-0.04%
MacOS 10.13.6 64 bit	0.10%	-0.02%
MacOS 12.1.0 64 bit	0.10%	-0.06%
MacOS 10.14.6 64 bit	0.10%	-0.01%
MacOS 12.0.1 64 bit	0.10%	-0.04%
MacOS 11.6.5 64 bit	0.08%	+0.03%
MacOS 11.6.0 64 bit	0.08%	-0.02%
MacOS 12.3.0 64 bit	0.06%	-0.15%
Linux	1.12%	-0.02%
"Arch Linux" 64 bit	0.14%	0.00%
Ubuntu 20.04.4 LTS 64 bit	0.13%	-0.03%
"Manjaro Linux" 64 bit	0.12%	-0.01%
Ubuntu 22.04 LTS 64 bit	0.09%	+0.09%
Linux Mint 20.3 64 bit	0.07%	0.00%

Рисунок 6 – Популярные ОС для персональных компьютеров

Как мы можем видеть на рисунке выше, самой популярной операционной системой для персональных компьютеров среди игроков является Windows 10, следом идет Windows 11 [12]. Вместе они занимают более 90% рынка ОС, вследствие чего было принято решение разработать игровое приложение для этих операционных систем. В будущем планируется разработать версии и для других ОС, а также для мобильных устройств под управлением Android.

1.4 Анализ функциональных требований

В результате анализа требований задания была создана диаграмма вариантов использования игрового приложения (рисунок 7).

Рисунок 7 – Диаграмма вариантов использования игрового приложения

Составим спецификацию вариантов использования «Начать игру» (рисунок 8).

Прецедент: Начать игру
ID: 1
Краткое описание: Начало игры
Главные актеры: Пользователь.
Основной поток: 1. Пользователь нажимает кнопку «Выбрать уровень» 2. Приложение предоставляет список доступных пользователю уровней 3. Пользователь выбирает из списка нужный уровень 4. Пользователь нажимает «Начать игру»

Рисунок 8 – Спецификация прецедента «Начать игру»

Составим спецификацию вариантов использования «Создать уровень» (рисунок 9).

Прецедент: Создать уровень
ID: 2
Краткое описание: Пользователь создаёт собственный уровень
Главные актеры: Пользователь
Основной поток: 1. Пользователь нажимает кнопку «Редактор карт» 2. Пользователь нажимает «Создать новый уровень» 3. Приложение показывает меню создание нового уровня

Рисунок 9 – Спецификация прецедента «Создать уровень»

Составим спецификацию вариантов использования «Посмотреть свои предыдущие результаты» (рисунок 10).

Прецедент: Посмотреть свои предыдущие результаты
ID: 3
Краткое описание: Пользователь просматривает результаты своих прошлых игр
Главные актеры: Пользователь
Предусловие: Пользователь запустил приложение
Основной поток: 1. В главном меню игрового приложения открывается список прошлых игр пользователя 2. Пользователь анализирует данных своих прошлых игр

Рисунок 10 – Спецификация прецедента «Посмотреть свои предыдущие результаты»

В результате анализа требований была также создана диаграмма вариантов использования игровой подсистемы (рисунок 11).

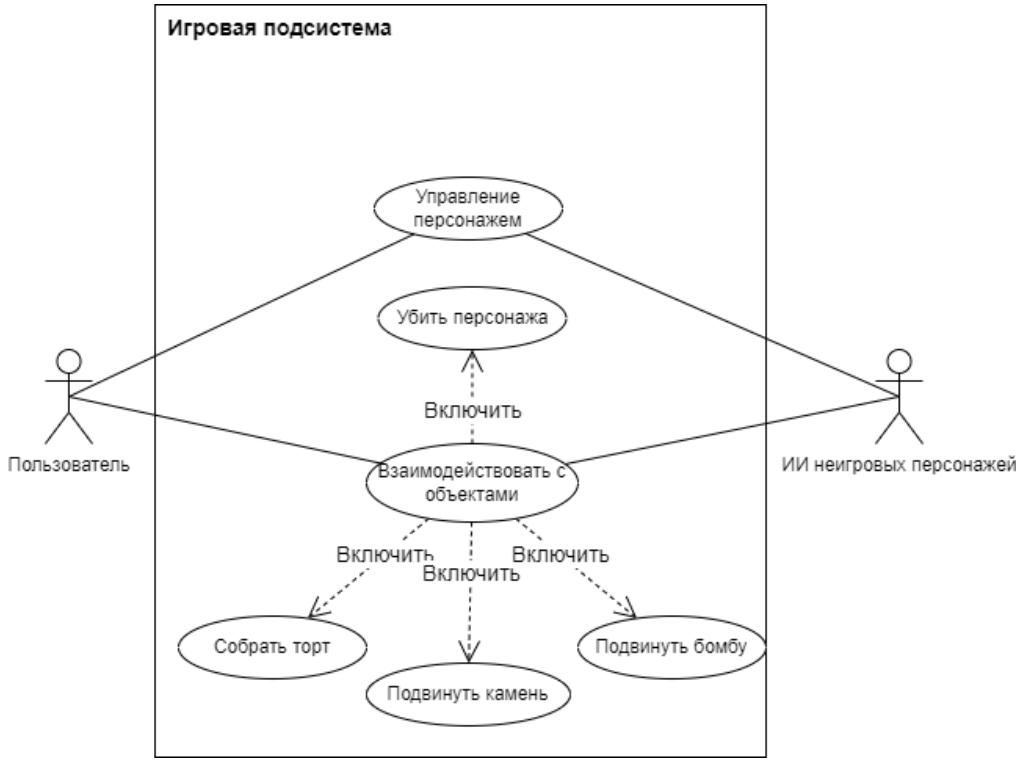

Рисунок 11 – Диаграмма вариантов использования игровой подсистемы

Составим спецификацию вариантов использования «Собрать торт» (рисунок 12).

Прецедент: Собрать торт
ID: 4
Краткое описание: Игровой персонаж под управлением пользователя собирает торт
Главные актеры: Пользователь
Предусловие: Пользователь запустил уровень
Основной поток: 1. Пользователь находит на игровом поле клетку “Торт” 2. Пользователь, используя элементы управления персонажем, размещает персонажа на нужной клетке 3. Игровая подсистема заменяет клетку “Торт” на пустую клетку, а пользователю засчитывает сбор торта и начисляет очки.

Рисунок 12 – Спецификация прецедента «Собрать торт»

Составим спецификацию вариантов использования «Управление персонажем» (рисунок 13).

Прецедент: Управление персонажем
ID: 5
Краткое описание: Пользователь управляет игровым персонажем
Главные актеры: Пользователь
Предусловие: Пользователь запустил уровень
Основной поток: 1. Пользователь нажимает кнопки управления 2. Система анализирует нажатые кнопки 3. Система меняет состояние игровой карты, в зависимости от ввода

Рисунок 13 – Спецификация прецедента «Управление персонажем»

Более подробное описание функционирования прецедентов можно представить в виде диаграммы деятельности. На рисунке 14 представлена диаграмма деятельности для варианта использования «Управление персонажем».

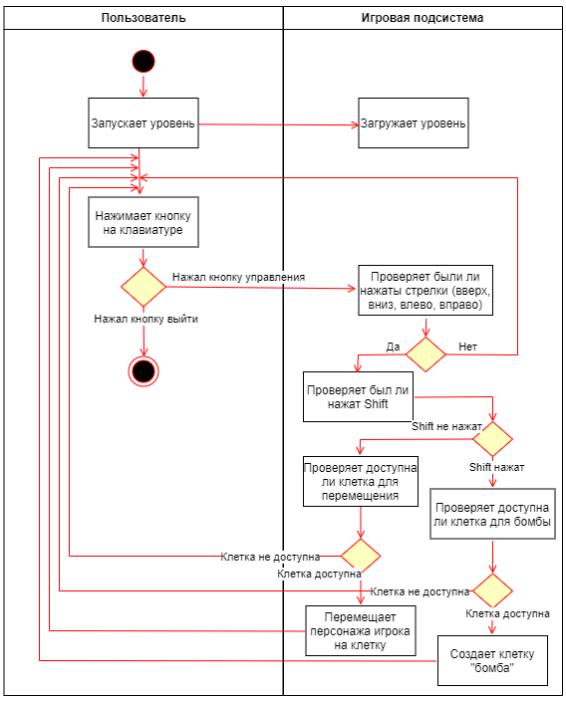

Рисунок 14 – Диаграмма деятельности для прецедента «Управление персонажем»

Для варианта использования «Съесть торт» была составлена диаграмма последовательности, представленная на рисунке 15.

Рисунок 15 – Диаграмма последовательности варианта использования «Съесть торт»

В результате анализа функциональных требований были успешно выявлены основные функции разрабатываемого приложения. При этом для разрабатываемого приложения была построена диаграмма вариантов использования, а также для некоторых выявленных вариантов использования были составлены спецификации.

1.5 Анализ нефункциональных требований

Будут использованы следующие технологии разработки: RUP [13], ООП.

В качестве инструмента создания игрового приложения был выбран межплатформенная среда разработки компьютерных игр Unity. Игровые сценарии должны быть написаны на языке программирования C#, в среде программирования Microsoft Visual Studio 2019. Визуальная часть создаётся с помощью многофункционального графического редактора Adobe Photoshop.

приложение должно быть отлажено и должно успешно работать под управлением ОС Windows 10. Для проектирования бюджета будет использован Microsoft Project.

1.6 Планирование разработки и оценка бюджета

Основной целью структурного планирования является описание состава и взаимосвязи технологических операций, которые требуются для реализации проекта. Также на этом этапе необходимо определить или предварительно оценить продолжительности работ [14 – 18].

Структурное планирование можно разделить на следующие этапы:

– разбиение проекта на совокупность отдельных работ, выполнение которых необходимо для реализации проекта;

– построение сетевого графика, описывающего последовательность выполнения работ;

– оценка временных характеристик работ и анализ сетевого графика.

Результатом структурного планирования является сетевой график – ориентированный граф, вершинами которого являются работы проекта, а дугами временные взаимосвязи.

Он должен удовлетворять следующим свойствам:

– каждой работе соответствует только одна вершина;

– ни одна работа не может быть начата, пока не закончатся все непосредственно предшествующие ей работы;

– ни одна работа, которая непосредственно следует за некоторой работой, не может начаться до момента ее окончания;

– начало и конец проекта (вехи) обозначены работами с нулевой продолжительностью.

Для оценки стоимости и трудоемкости проекта необходимо провести правильное планирование. Для этого были определены задачи и ресурсы, которые необходимы для создания игрового приложения. В таблице 6 представлены задачи, которые представляют этапы создания проекта.

Таблица 6 – Перечень задач проекта

Название задачи	Длительность	Начало	Окончание	Предшественники
Начало	0 дней	Вт 01.03.22	Вт 01.03.22	
Анализ требований	**10 дней**	**Вт 01.03.22**	**Пн 14.03.22**	
Анализ аналогичных решений	6 дней	Вт 01.03.22	Вт 08.03.22	1
Анализ правил проведения соревнований	3 дней	Ср 09.03.22	Пт 11.03.22	3
Принятие окончательное решения	1 день	Пн 14.03.22	Пн 14.03.22	4
Анализ требований завершен	0 дней	Пн 14.03.22	Пн 14.03.22	5
Выбор средств реализации проекта	**8 дней**	**Вт 01.03.22**	**Чт 10.03.22**	
Анализ возможных программных средств	6 дней	Вт 01.03.22	Вт 08.03.22	1
Выбор программного средства для реализации	2 дней	Ср 09.03.22	Чт 10.03.22	8
Выбор средств завершен	0 дней	Чт 10.03.22	Чт 10.03.22	9
Реализация проекта	**12 дней**	**Вт 15.03.22**	**Ср 30.03.22**	
Разработка структуры данных	2 дней	Вт 15.03.22	Ср 16.03.22	6;10
Разработка игрового приложения	**10 дней**	**Чт 17.03.22**	**Ср 30.03.22**	
Программирование логики приложения	3 дней	Чт 17.03.22	Пн 21.03.22	12
Разработка редактора карт	7 дней	Вт 22.03.22	Ср 30.03.22	14
Разработка пользовательского интерфейса	**10 дней**	**Чт 17.03.22**	**Ср 30.03.22**	
Разработка макета интерфейса приложения	4 дней	Чт 17.03.22	Вт 22.03.22	12
Верстка приложения	6 дней	Ср 23.03.22	Ср 30.03.22	17
Реализация проекта завершена	0 дней	Ср 30.03.22	Ср 30.03.22	15;18
Отладка	**13 дней**	**Чт 31.03.22**	**Пн 18.04.22**	
Тестирование пользовательского интерфейса	3 дней	Чт 31.03.22	Пн 04.04.22	19
Исправление ошибок	10 дней	Вт 05.04.22	Пн 18.04.22	21
Тестирование завершено	0 дней	Пн 18.04.22	Пн 18.04.22	22
Внедрение программного средства в систему проведения соревнований города	**15 дней**	**Вт 19.04.22**	**Пн 09.05.22**	
Внедрение программного средства	10 дней	Вт 19.04.22	Пн 02.05.22	23
Поиск и исправления ошибок при внедрении	5 дней	Вт 03.05.22	Пн 09.05.22	25
Внедрение завершено	0 дней	Пн 09.05.22	Пн 09.05.22	26
Конец проекта	0 дней	Пн 09.05.22	Пн 09.05.22	27

Из таблицы, представленной выше видно, что для реализации проекта необходимо выполнить 28 задач. Список ресурсов представлен на таблице 7.

Таблица 7 – Список ресурсов проекта

Название ресурса	Тип	Стандартная ставка		Ставка сверхурочных
Геймдизайнер-программист	Трудовой	300,00 ₽/ч		350,00 ₽/ч
Программист	Трудовой	200,00 ₽/ч		250,00 ₽/ч
Дизайнер	Трудовой	150,00 ₽/ч		200,00 ₽/ч
Компьютер	Материальный	400,00 ₽		
Интернет	Затраты			

Назначение ресурсов по задачам представлено в таблице 8.

Таблица 8 – Назначение ресурсов

Название задачи	Трудозатраты	Длительность
Начало	0 ч	0 дней
Анализ требований	**52 ч**	**10 дней**
Анализ аналогичных решений	24 ч	6 дней
Геймдизайнер-программист	*24 ч*	
Компьютер	*1*	
Интернет		
Анализ правил проведения соревнований	24 ч	3 дней
Геймдизайнер-программист	*12 ч*	
Программист	*12 ч*	
Компьютер	*1*	
Принятие окончательное решения	4 ч	1 день
Геймдизайнер-программист	*2 ч*	
Программист	*2 ч*	
Анализ требований завершен	0 ч	0 дней
Выбор средств реализации проекта	**64 ч**	**8 дней**
Анализ возможных программных средств	48 ч	6 дней
Геймдизайнер-программист	*24 ч*	
Программист	*24 ч*	
Компьютер	*2*	
Интернет		
Выбор программного средства для реализации	16 ч	2 дней
Геймдизайнер-программист	*8 ч*	
Программист	*8 ч*	
Компьютер	*2*	
Интернет		
Выбор средств завершен	0 ч	0 дней
Реализация проекта	**176 ч**	**12 дней**
Разработка структуры данных	16 ч	2 дней
Геймдизайнер-программист	*16 ч*	
Компьютер	*1*	
Интернет		
Разработка игрового приложения	**80 ч**	**10 дней**
Программирование логики приложения	24 ч	3 дней
Геймдизайнер-программист	*24 ч*	
Компьютер	*1*	
Интернет		
Разработка редактора карт	56 ч	7 дней

Продолжение таблицы 8

Название задачи	Трудозатраты	Длительность
Геймдизайнер-программист	*56 ч*	
Компьютер	*1*	
Интернет		
Разработка пользовательского интерфейса	**80 ч**	**10 дней**
Разработка макета интерфейса приложения	32 ч	4 дней
Дизайнер	*32 ч*	
Компьютер	*1*	
Интернет		
Верстка приложения	48 ч	6 дней
Программист	*48 ч*	
Компьютер	*1*	
Интернет		
Реализация проекта завершена	0 ч	0 дней
Отладка	**104 ч**	**13 дней**
Тестирование пользовательского интерфейса	24 ч	3 дней
Геймдизайнер-программист	*4,8 ч*	
Программист	*12 ч*	
Дизайнер	*7,2 ч*	
Компьютер	*1*	
Интернет		
Исправление ошибок	80 ч	10 дней
Геймдизайнер-программист	*32 ч*	
Программист	*48 ч*	
Компьютер	*2*	
Интернет		
Тестирование завершено	0 ч	0 дней
Внедрение программного средства в систему проведения соревнований города	**120 ч**	**15 дней**
Внедрение программного средства	80 ч	10 дней
Геймдизайнер-программист	*40 ч*	
Программист	*40 ч*	
Компьютер	*2*	
Интернет		
Поиск и исправления ошибок при внедрении	40 ч	5 дней
Геймдизайнер-программист	*20 ч*	
Программист	*20 ч*	
Компьютер	*2*	
Интернет		
Внедрение завершено	0 ч	0 дней
Конец проекта	0 ч	0 дней

Из представленных таблиц видно, что реализацией проекта будут заниматься 1 геймдизайнер-программист, программиста и 1 дизайнер, которым для работы нужны компьютеры и доступ к интернету.

Сетевой график представлен на рисунке 16.

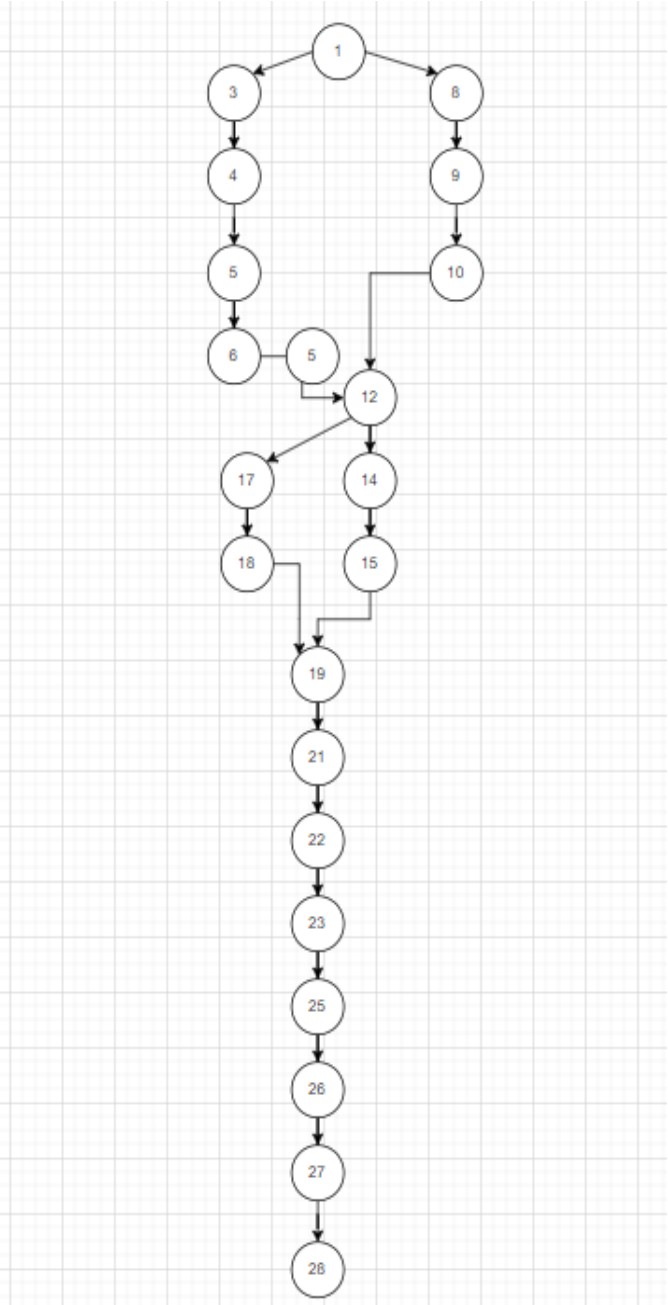

Рисунок 16 – Сетевой график

По данному графику видно задачи, что некоторые задачи выполняются параллельно.

Календарное планирование проведем с помощью системы управлениями проектами Microsoft Office Project 2007.

На рисунке 17 представлен список задач из таблицы 8, внесенный в Microsoft Project. Началом проекта была принята дата 01.03.2022.

	Название задачи	Длительность	Начало	Окончание	Предшественн	Названия
1	Начало	0 дней	Вт 01.03.22	Вт 01.03.22		
2	**Анализ требований**	**10 дней**	**Вт 01.03.22**	**Пн 14.03.22**		
3	Анализ аналогичных решений	6 дней	Вт 01.03.22	Вт 08.03.22	1	Геймдизайнер-прс
4	Анализ правил проведения сорев	3 дней	Ср 09.03.22	Пт 11.03.22	3	Геймдизайнер-прс
5	Принятие окончательное решени	1 день	Пн 14.03.22	Пн 14.03.22	4	Геймдизайнер-прс
6	Анализ требований завершен	0 дней	Пн 14.03.22	Пн 14.03.22	5	
7	**Выбор средств реализации проекта**	**8 дней**	**Вт 01.03.22**	**Чт 10.03.22**		
8	Анализ возможных программных	6 дней	Вт 01.03.22	Вт 08.03.22	1	Геймдизайнер-прс
9	Выбор программного средства дл	2 дней	Ср 09.03.22	Чт 10.03.22	8	Геймдизайнер-прс
10	Выбор средств завершен	0 дней	Чт 10.03.22	Чт 10.03.22	9	
11	**Реализация проекта**	**12 дней**	**Вт 15.03.22**	**Ср 30.03.22**		
12	Разработка структуры данных	2 дней	Вт 15.03.22	Ср 16.03.22	6;10	Геймдизайнер-прс
13	**Разработка игрового приложения**	**10 дней**	**Чт 17.03.22**	**Ср 30.03.22**		
14	Программирование логики при	3 дней	Чт 17.03.22	Пн 21.03.22	12	Геймдизайнер-прс
15	Разработка редактора карт	7 дней	Вт 22.03.22	Ср 30.03.22	14	Геймдизайнер-прс
16	**Разработка пользовательского ин**	**10 дней**	**Чт 17.03.22**	**Ср 30.03.22**		
17	Разработка макета интерфейса	4 дней	Чт 17.03.22	Вт 22.03.22	12	Дизайнер;Компью
18	Верстка приложения	6 дней	Ср 23.03.22	Ср 30.03.22	17	Программист;Комп
19	Реализация проекта завершена	0 дней	Ср 30.03.22	Ср 30.03.22	15;18	
20	**Отладка**	**13 дней**	**Чт 31.03.22**	**Пн 18.04.22**		
21	Тестирование пользовательского	3 дней	Чт 31.03.22	Пн 04.04.22	19	Геймдизайнер-прс
22	Исправление ошибок	10 дней	Вт 05.04.22	Пн 18.04.22	21	Геймдизайнер-прс
23	Тестирование завершено	0 дней	Пн 18.04.22	Пн 18.04.22	22	
24	**Внедрение программного средства**	**15 дней**	**Вт 19.04.22**	**Пн 09.05.22**		
25	Внедрение программного средст	10 дней	Вт 19.04.22	Пн 02.05.22	23	Геймдизайнер-прс
26	Поиск и исправления ошибок при	5 дней	Вт 03.05.22	Пн 09.05.22	25	Геймдизайнер-прс
27	Внедрение завершено	0 дней	Пн 09.05.22	Пн 09.05.22	26	
28	Конец проекта	0 дней	Пн 09.05.22	Пн 09.05.22	27	

Рисунок 17 – Задачи проекта в среде Microsoft Project

На рисунке 18 представлена диаграмма Ганта, построенная в среде Microsoft Project на основе введенных данных.

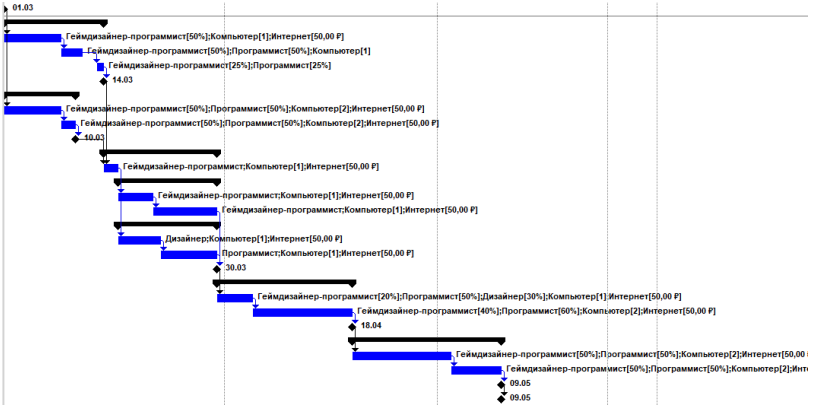

Рисунок 18 – Диаграмма Ганта

Из составленного сетевого графика можно сделать вывод, что критический путь проходит через задачи 1, 3, 4, 5, 12, 14, 17, 18, 19, 21, 22, 23, 25, 26, 27, 28.

На рисунке 19 представлен обзор проекта. Из рисунка видно, что общая длительность проекта составляет 50 дней, а предполагаемые затраты – 135 320 руб.

Даты

Начало:	Вт 01.03.22	Окончание:	Пн 09.05.22
Баз. начало:	НД	Баз. окончание:	НД
Факт. начало:	НД	Факт. окончание:	НД
Отклон. начала:	0 дней	Отклон. окончания:	0 дней

Длительность

Текущий план:	50 дней	Оставшаяся:	50 дней
Базовый план:	0 дней?	Фактическая:	0 дней
Отклонение:	50 дней	Процент завершени:	0%

Трудозатраты

Текущий план:	516 ч	Оставшиеся:	516 ч
Базовый план:	0 ч	Фактические:	0 ч
Отклонение:	516 ч	Процент завершени:	0%

Затраты

Текущий план:	135 320,00 ₽	Оставшиеся:	135 320,00 ₽
Базовый план:	0,00 ₽	Фактические:	0,00 ₽
Отклонение:	135 320,00 ₽		

Состояние задачи

Еще не начатые задачи	28
Выполняющиеся задачи	0
Завершенные задачи:	0
Итого задач:	28

Состояние ресурса

Трудовые ресурсы:	4
Трудовые ресурсы с прев. дос	0
Материальные ресурсы:	1
Итого ресурсов:	5

Рисунок 19 – Обзорный отчет проекта

Приведем графики ресурсов. На рисунках 20 – 22 представлены графики трудовых ресурсов «Геймдизайнер-программист», «Программист», «Дизайнер».

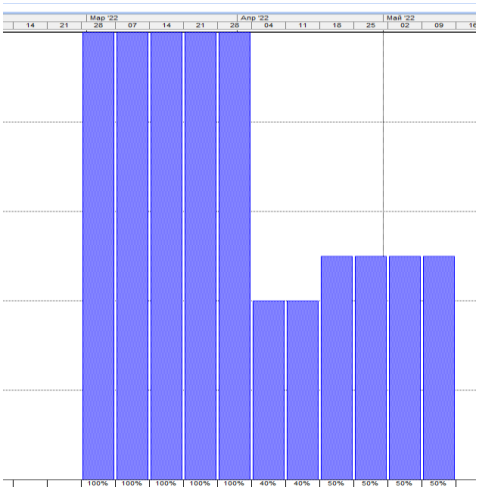

Рисунок 20 – График ресурсов для ресурса «Геймдизайнер-программист»

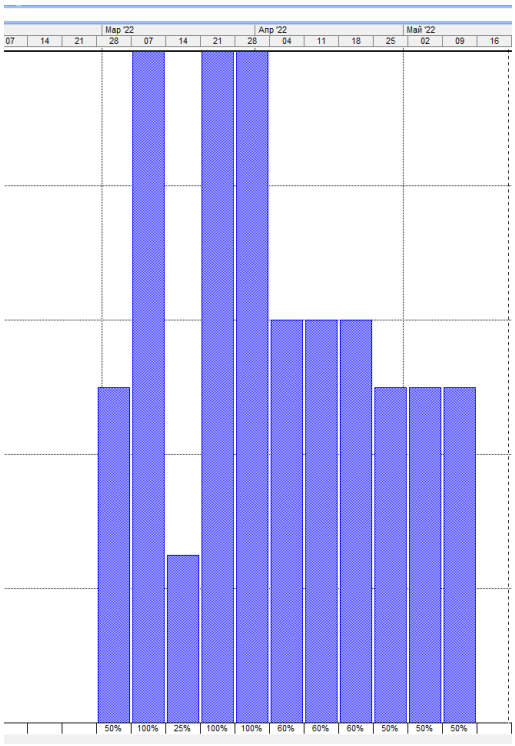

Рисунок 21 – График ресурсов для ресурса «Программист»

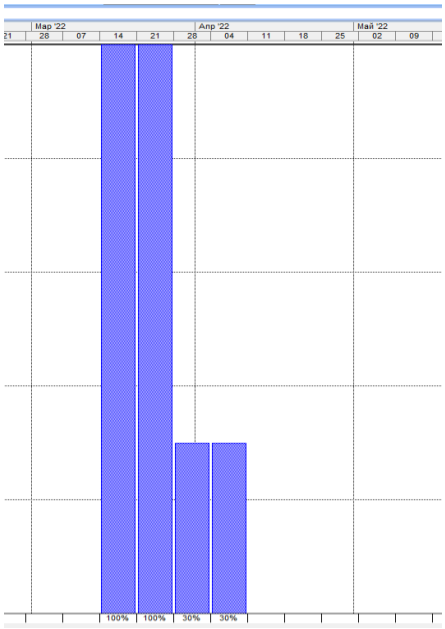

Рисунок 22 – График ресурсов для ресурса «Дизайнер»

На основе произведенных оценок стоимости и сроков проекта можно сделать вывод о его экономической целесообразности, а также ни один трудовой ресурс не превышает норму нагрузки.

2 Проектирование игрового приложения

2.1 Выбор технологий и инструментальных средств разработки

В процессе разработки приложения было решено использовать методологию RUP. Разработка приложения в соответствии с методологией RUP носит итеративный характер. На каждой итерации перед разработчиками ставятся определенные цели, которые должны быть достигнуты к ее концу. В результате в конце каждой итерации должна получиться промежуточная функциональная версия разрабатываемого программного обеспечения. В методологии RUP существуют 4 основных фазы, каждая из которых может включать в себя одну или несколько итераций [19]:

– фаза начала проекта – формируется общее видение проекта и экономическое обоснование;

– уточнение – анализ предметной области и построение архитектуры продукта;

– построение – реализация основного функционала продукта;

– внедрение – получение финальной версии продукта и его передача заказчику, фаза включает в себя бета-тестирование и устранение выявленных в его процессе ошибок, а также обучение пользователей.

Каждая фаза завершается вехой, на которой команда разработчиков оценивает выполнение поставленных целей и принимает критически важные решения о ходе дальнейшей работы над продуктом.

Выбор в пользу данной методологии продиктован рядом ее достоинств. С одной стороны, методология RUP обеспечивает гибкость процесса разработки и позволяет быстро вносить коррективы в проект в случае выявления в нем ошибок или изменения требований к разрабатываемому приложению.

Для анализа требований, а также моделирования структуры проекта в соответствии с принципами RUP было решено использовать диаграммы на

языке моделирования UML [20]. Для построения диаграмм использовалось средство UML-моделирования ArgoUML.

На этапе планирования проекта было решено использовать приложение Microsoft Project. Оно позволяет обеспечить эффективное управление проектом на всех стадиях его выполнения. Microsoft Project предоставляет возможность выполнить структуризацию проекта, составить его календарный план, а также рассчитать затраты трудовых, материальных и денежных ресурсов в процессе работы над проектом.

Для создания игрового приложения было решено использовать платформу разработки Unity, а также среду программирования Microsoft Visual Studio 2019, с использованием языка программирования C#.

2.2 Проектирование архитектуры программного обеспечения

Было принято решение создать основной абстрактный класс управления игровыми объектами (AbstractObjectController), который имеет в себе базовую логику управления и создания объекта, в нем реализованы такие методы как CanMove (проверка возможности перемещения), Move (перемещение), PushObject (толкание объекта), а также такой абстрактный метод как MovementLogic (логика перемещения объекта по игровому полю) [21]. Все классы управления игровыми объектами должны наследоваться от данного класса и реализовывать метод MovementLogic. Диаграмма классов представлена на рисунке 23.

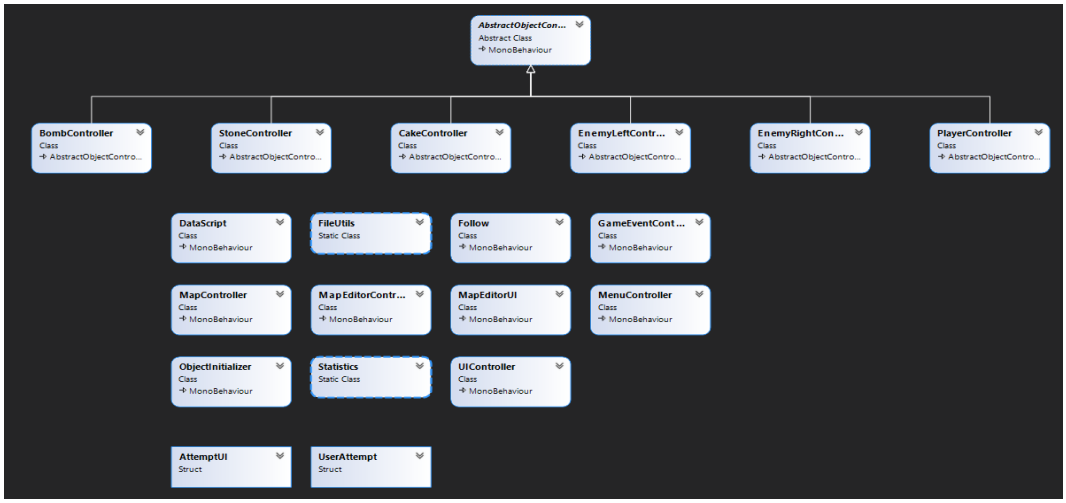

Рисунок 23 – Диаграмма классов

Описание классов игрового приложения представлены ниже.

MenuController и UIController – это классы отвечающие за интерфейс игры.

AbstarctObjectController – это абстрактный класс, от которого наследуются классы, отвечающие за игровые объекты, имеющие возможность перемещаться по игровому полю.

PlayerController – это класс управления игроком, содержит реализацию управления пользователем игрового персонажа.

EnemyLeftController – это класс управления противником левого типа, содержит реализацию логики перемещения левого типа, а также реализация логики убийства игрового персонажа.

EnemyRightController – это класс управления противником правого типа, содержит реализацию логики перемещения правого типа, а также реализация логики убийства игрового персонажа.

StoneController – это класс управления камнем, содержит реализацию логики падения, а также реализацию логики убийства игрового персонажа и некоторых других объектов сцены.

BombController – это класс управления бомбой, содержит реализацию логики падения и взрыва.

CakeController – это класс управления тортом, содержит реализацию логики падений, реализацию логики съедения.

ObjectInitializer – это класс, в котором реализовано правильное, с точки зрения игровой логики, создание и удаление игровых объектов.

MapController – это класс управление игровой картой, сдержит реализацию логики загрузки игровой карты, получения объектов на карте и перемещения объектов на карте.

FileUtils – это класс, содержащий логику распознавания файлов карт игрового приложения.

DataScript – это класс, содержащий данные для работы игровой логики: префабы объектов, коды игровых объектов, коды игровых режимов, белые списки перемещения.

MapEditorController – это класс управления логикой редактора карт, содержит реализацию разных режимов редактирования, импорта и экспорта игровой карты.

MapEditorUI – это класс, отвечающий за интерфейс сцены «Редактор карт».

GameEventController – это класс, отвечающий за игровые события и их вывод на интерфейс пользователя.

Statistics – это класс, отвечающий за сохранение информации о попытках пользователя.

UserAttempt и AttemptUI – это структуры, хранящие в себе данные о попытках пользователя и обеспечивающие правильный её вывод на интерфейс в главном меню игры.

2.3 Проектирование интерфейса пользователя

На основе выявленных функциональных требований был разработан интерфейс игрового приложения [22]. Он включает в себя главное меню игры, меню выбора уровня, а также интерфейс игрового процесса и интерфейс редактирования уровня.

Главное меню представлено на рисунке 24. На нем располагаются следующие элементы: 1 – текст главного меню, 2 – кнопка «Играть», которая переносит пользователя в меню выбора уровня, 3 – кнопка «Выйти», 4 – текстовая информация о последней попытке игрока, 5 – текстовая информация о лучшей попытке игрока. Кнопка «Выйти» закрывает приложение, тогда как при нажатии кнопки «Играть» пользователя переносит в меню выбора уровня. Также на главном меню программы пользователь может посмотреть результаты своей самой успешной попытки и своей последней попытки, если пользователь ещё не делал попыток соответствующие поля будут равны нулю.

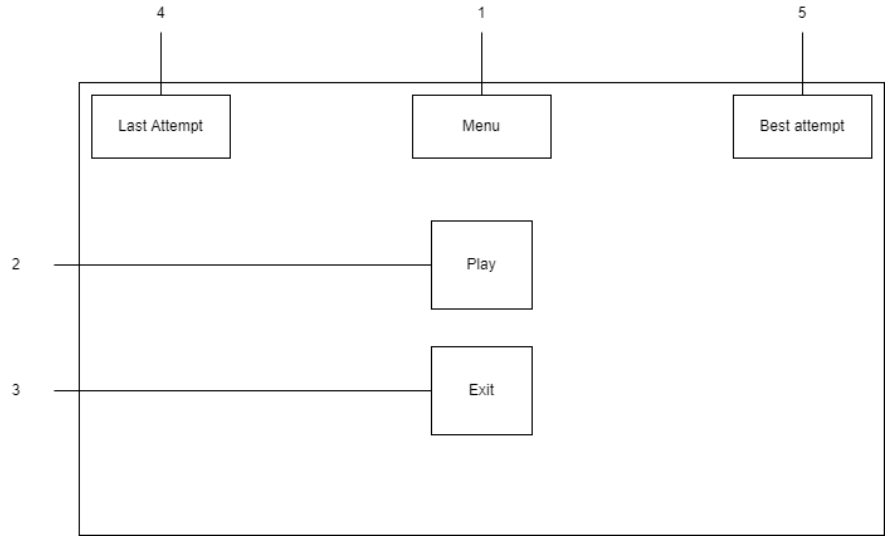

Рисунок 24 – Главное меню игрового приложения

После того как пользователь нажал кнопку «Играть», он попадает в меню выбора уровня (рисунок 25). На нем располагаются следующие элементы: 1 – кнопка «Назад», 2 – текстовый блок выбора уровня, 3 – панель, где выводятся кнопки выбора уровня, 4 – кнопка «Начать уровень», 5 – кнопка «Редактировать уровень». После нажатия кнопки «Начать уровень» пользователя переносит в выбранный на панели 3 уровень в режиме стандартной игры. После нажатия кнопки «Редактировать уровень» пользователя переносит в выбранный на панели 3 уровень в режиме редактирования уровня. Нажатие кнопки «Назад» возвращает пользователя в главное меню.

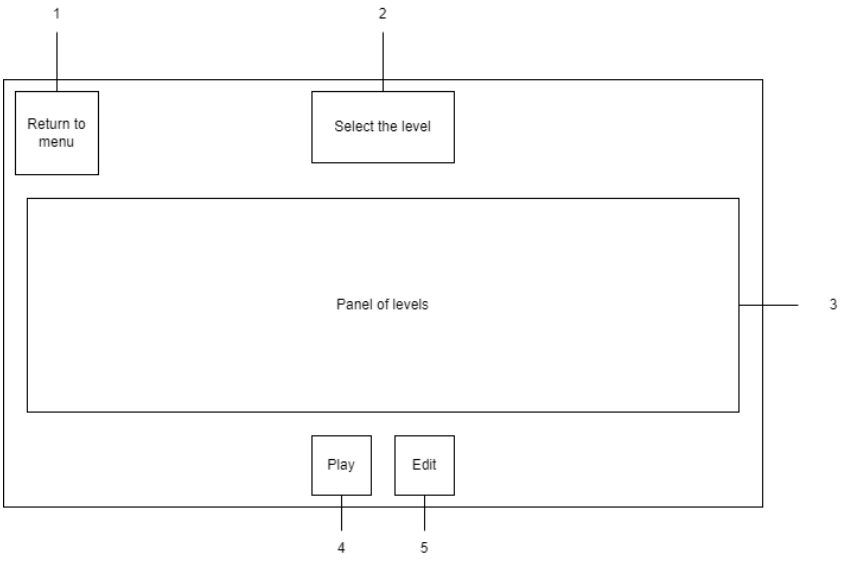

Рисунок 25 – Интерфейс главного меню во время выбора уровня

После того как пользователь нажал кнопку «Начать уровень» он попадает непосредственно на уровень, где располагается интерфейс игрового процесса (рисунок 26). На нем располагаются следующие элементы: 1 – кнопка «Меню», 2 – текстовый блок с информацией о количестве собранных тортов, 3 – текстовый блок с информацией о количестве оставшихся противников, 4 – текстовый блок с информацией о количестве оставшихся бомб, 5 – игровое поле, где непосредственно проходит игровой процесс. При нажатии кнопки «Меню» уровень заканчивается и открывается главное меню приложения. На игровом поле располагаются разные игровые объекты, с которыми может взаимодействовать игровой персонаж под управлением пользователя.

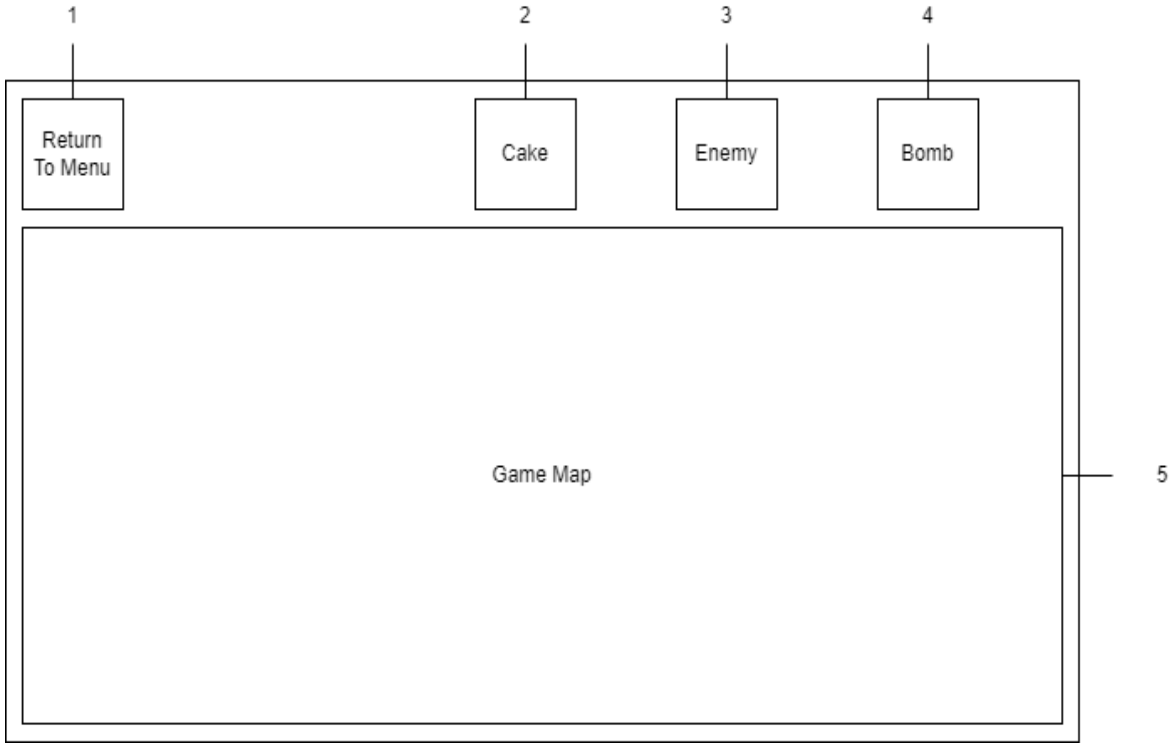

Рисунок 26 – Интерфейс во время игрового процесса

После того как пользователь нажал кнопку «Редактировать уровень» он попадает непосредственно на уровень, где располагается интерфейс игрового процесса в режиме редактирования уровня (рисунок 27). Интерфейс аналогичен таковому в режиме простой игры. На нем располагаются следующие элементы: 1 – кнопка «Меню», 2 – текстовый блок с информацией о количестве собранных тортов, 3 – текстовый блок с

информацией о количестве оставшихся противников, 4 – текстовый блок с информацией и количестве оставшихся бомб, 5 – игровое поле, где непосредственно проходит игровой процесс, 6 – выпадающий список, содержащий все игровые объекты приложения. При нажатии кнопки «Меню» уровень заканчивается и открывается главное меню приложения. На игровом поле располагаются разные игровые объекты, с которыми может взаимодействовать игровой персонаж под управлением пользователя. Также, при нажатии левой кнопки мыши, на месте расположения курсора будет появляться игровой объект, выбранный в выпадающем списке 6. Во время игрового процесса игрок может использовать бомбы, количество доступных бомб указано в текстовом блоке 4, при использовании бомбы, обновленная информация о доступном количестве сразу же высветится в данном блоке.

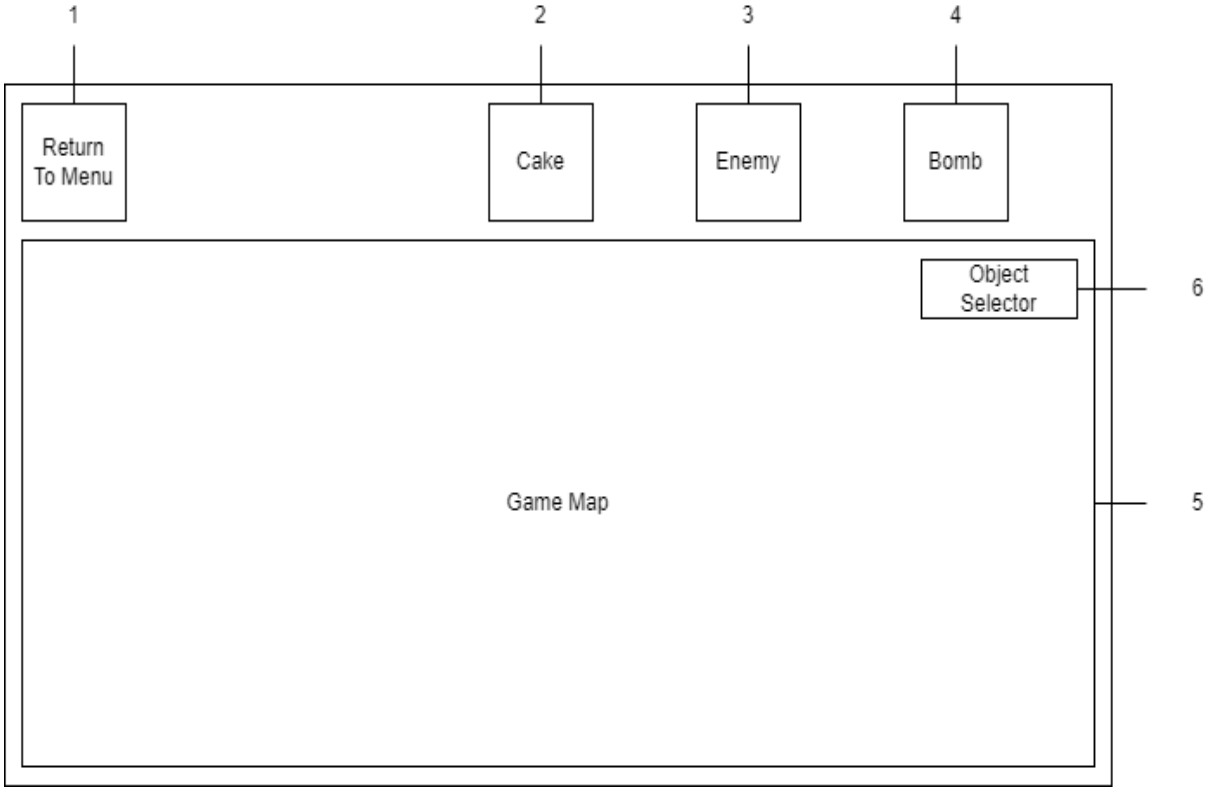

Рисунок 27 – Интерфейс во время редактирования карты

Таким образом спроектированный графический интерфейс удовлетворяет всем функциональным требованиям игрового приложения.

3 Реализация игрового приложения

3.1 Реализация клиентской части игрового приложения

Разработка игрового приложения производилась на языке C# с использованием среды программирования Microsoft Visual Studio и мультиплатформенной инструментальной среды Unity. Диаграмма компонентов, отражающая структуру программы, представлена на рисунке 28.

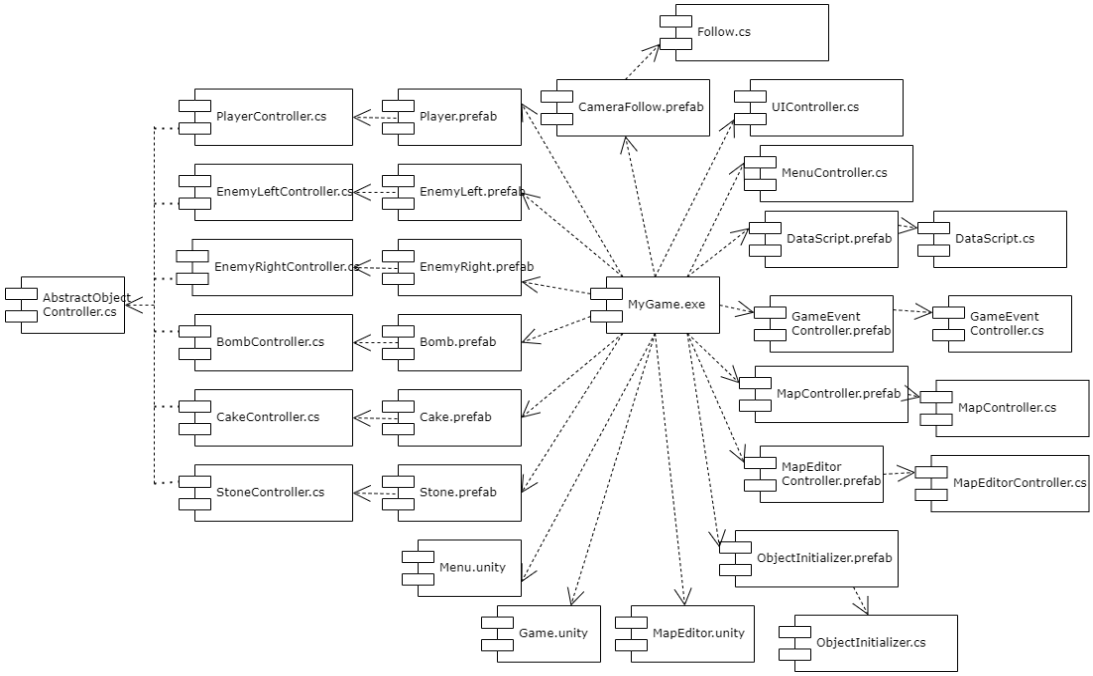

Рисунок 28 – Диаграмма компонентов игрового приложения

Описание компонентов представлено в таблице 9.

Таблица 9 – Компоненты игрового приложения

Компонент	Назначение
AbstractObjectController.cs	Файл абстрактного класса, реализующий базовое управление объектами сцены.
PlayerController.cs	Файл класса, реализующий управление игроком.
EnemyLeftController.cs	Файл класса, реализующий управление врагом левого типа.
EnemyRightController.cs	Файл класса, реализующий управление врагом правого типа.
BombController.cs	Файл класса, реализующий управление бомбами.
CakeController.cs	Файл класса, реализующий управление тортом.
StoneController.cs	Файл класса, реализующий управление камнем.
Player.prefab	Prefab-файл, содержащий объект сцены – игрока.

Продолжение таблицы 9

Компонент	Назначение
EnemyLeft.prefab	Prefab-файл, содержащий объект сцены – врага левого типа.
EnemyRight.prefab	Prefab-файл, содержащий объект сцены – врага правого типа.
Bomb.prefab	Prefab-файл, содержащий объект сцены – бомбу.
Cake.prefab	Prefab-файл, содержащий объект сцены – торт.
Stone.prefab	Prefab-файл, содержащий объект сцены – камень.
CameraFollow.prefab	Prefab-файл, содержащий объект сцены – управление камерой.
DataScript.prefab	Prefab-файл, содержащий объект сцены – игровые данные.
GameEventController.prefab	Prefab-файл, содержащий объект сцены – управление игровыми событиями.
MapController.prefab	Prefab-файл, содержащий объект сцены – управление картой.
MapEditorController.prefab	Prefab-файл, содержащий объект сцены – управление редактором карт.
ObjectInitializer.prefab	Prefab-файл, содержащий объект сцены – управление игровыми объектами.
Menu.unity	Файл сцены Unity, содержащий логику меню.
Game.unity	Файл сцены Unity, содержащий логику игры.
MapEditor.unity	Файл сцены Unity, содержащий логику редактора карт.
Follow.cs	Файл класса, реализующий управление камерой.
UIController.cs	Файл класса, реализующий управление игровым интерфейсом.
MenuController.cs	Файл класса, реализующий управление меню игры.
DataScript.cs	Файл класса, содержащий игровые данные.
GameEventController.cs	Файл класса, реализующий управление игровыми событиями.
MapController.cs	Файл класса, реализующий управление картой.
MapEditorController.cs	Файл класса, реализующий управление редактором карт.
ObjectInitializer.cs	Файл класса, реализующий управление игровыми объектами.
MyGame.exe	Исполняемый файл игрового приложения.

Все данные о стандартных и пользовательских картах хранятся в виде файлов, каждой карте соответствует свой файл определенного формата. Данные в этом формате состоят из двух блоков: блоке информации о карте и блоке самой карты. В первом блоке содержится информация об имени и размере карты, а во втором блоке – информация о расположении объектов. Стандартные карты хранятся в директории игрового приложения в maps/official, а созданные пользователем карты хранятся в maps/unofficial. Игровое приложение работает под управлением ОС Windows 10 и выше.

Листинг клиентской части программы представлен в приложении Б.

4 Тестирование программного продукта

4.1 Тестирование черным ящиком

Тестирование чёрного ящика или поведенческое тестирование [23] – стратегия (метод) тестирования функционального поведения объекта (программы, системы) с точки зрения внешнего мира, при котором не используется знание о внутреннем устройстве (коде) тестируемого объекта. Иначе говоря, тестированием чёрного ящика занимаются тестировщики, не имеющие доступ к исходному коду приложения. Под стратегией понимаются систематические методы отбора и создания тестов для тестового набора. Стратегия поведенческого теста исходит из технических требований и их спецификаций.

4.1.1 Классы эквивалентности

Классы эквивалентности [24] – это техника тестирования ПО, которая делит вводимые данные на классы эквивалентных друг другу значений, на базе которых создаются тест-кейсы. Было принято решение сформировать классы эквивалентности для функции обработки карты. Классы эквивалентности приведены на таблице 10.

Таблица 10 – Классы эквивалентности

	Правильные КЭ	Неправильные КЭ
Размер карты	3-1024x3-1024(1)	0-2x0-2(2), 0-2x1025-∞(3), 1025-∞x0-2(4), 1025-∞x1025-∞(5)
Структура	Подблок имеет id и кол-во, разделенных “x”(6), подблоки разделены “|”(7)	Подблок не имеет id и кол-во, разделенных “x” (8), подблоки не разделены “|”(9)
Содержание	Цифра (10), Прямая черта (11), Буква “x” (12)	Остальное (13)
	Хотя бы один блок 100x1 (14)	Нет блока 100x1 (15)

Таким образом были сформированы классы эквивалентности для функции обработки карты.

4.1.2 Граничные условия

Граничное условия [25] – это процесс тестирования между крайними концами или границами между разделами входных значений. Таким образом, эти крайние значения, такие как начальное-конечное, нижнее-верхнее, максимальное-минимальное, значение Just Inside-Just Outside, называются граничными значениями, а тестирование называется «граничным тестированием». Граничные условия для заданного алгоритма приведены на таблице 11.

Таблица 11 – Граничные условия

Границы	
Физические	Отрицательное число объектов (Ф1)
Логические	Отрицательный ID (Л1)
Технологические	Размер карты больше 1024x1024 (Е1)
Произвольные	Нет блока 1x100 (П1)

Таким образом были сформированы граничные условия для функции обработки карты.

4.1.3 Тест-кейсы

По ранее заданным классам эквивалентности и граничным условиям были созданы тест-кейсы для их проверки (таблица 12).

Таблица 12 – Тест-кейсы

№	КЭ/ГУ	Входные данные	Выходные данные
1	1,6,7,10,11,12,14	1x3 1x1|100x1|1x1 1x3	Успешно
2	2,6,7,10,11,12,14	1x2 1x1|100x1 1x2	Некорректный размер карты
3	3,6,7,10,11,12,14	1x2 1x1|100x1|1x1025 1x2	Некорректный размер карты
4	4,6,7,10,11,12,14	1x1025 1x1|100x1 1x2	Некорректный размер карты

Продолжение таблицы 12

№	КЭ/ГУ	Входные данные	Выходные данные
5	5,6,7,10,11,12,14	1x1025 1x1|100x1|1x1025 1x1025	Некорректный размер карты
6	1,8,7,10,11,12,14	13 11|1001|11 13	Неправильный формат карты
7	1,6,9,10,11,12,14	1x3 1x1100x11x1 1x3	Неправильный формат карты
8	11,12,13	dasx|	Неправильный формат карты
9	1,6,7,10,13,12,14	1x3 1x1100x11x1 1x3	Неправильный формат карты
10	1,6,7,10,11,13,14	13 11|1001|11 13	Неправильный формат карты
11	1,6,7,10,11,12,15	1x3 1x1|2x1|1x1 1x1	Нет блока игрока
12	Ф1	1x-3 1x1|100x1|1x1 1x3	Ошибка
13	Л1	-1x3 1x1|100x1|1x1 1x3	Ошибка
14	Т1	1x1024 1x1|100x1|1x1 1x3	Ошибка
15	П1	1x3 1x3 1x3	Ошибка

Таким образом были определены входные и выходные значения для тестирования граничных условий и классов эквивалентности.

4.2 Функциональное тестирование белым ящиком алгоритма передвижения

4.2.1 Алгоритм

В качестве тестируемого алгоритма было принято решение взять функцию перемещения игрового персонажа. Исходный код представлен в листинге 1.

Листинг 1 – Исходный код алгоритма перемещения персонажа

```
        /// <summary>
        /// Функция передвижения
        /// </summary>
        /// <param name="moveDirection">Направление движения</param>
        /// <param name="offset">Вектор-смещение относительно текущей
позиции</param>
        private void Move(string moveDirection, Vector3 offset)
        {
            if (Direction != moveDirection)
                Direction = moveDirection;

            int obj = Map.GetObj(Position + offset);

            //Если на том месте куда мы движемся есть объекты, которые мы
можем толкать, то пытаемся их толкнуть
            if (obj == IDBomb || obj == IDStone)
            {
                var go = Initializer.listOfObjects[Position + offset];

                PushObject(go, offset);
            }
            //Если на том месте куда мы движемся есть объекты, которые мы
можем съесть, то пытаемся их съесть
            else if (obj == IDCake)
            {
                Event.EatDiamond();
                Initializer.KillObject(Initializer.listOfObjects[Position +
offset]);
            }

            base.Move(offset);
        }
```

Схема алгоритма представлена на рисунке 29.

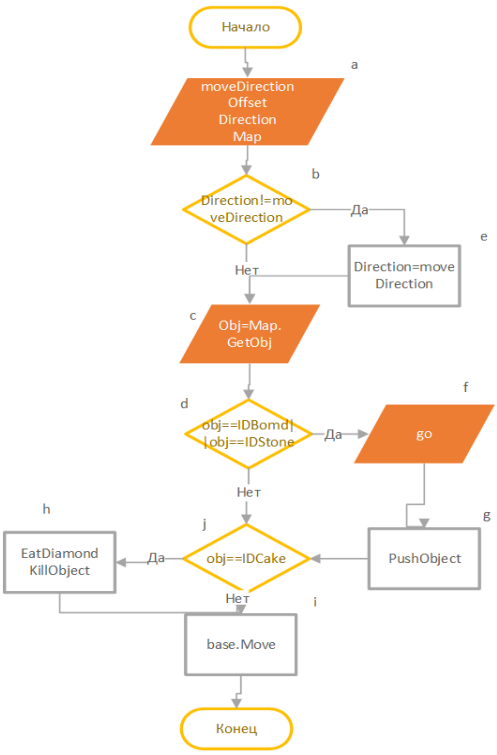

Рисунок 29 – Схема алгоритма перемещения персонажа

Таким образом было принято решение протестировать алгоритм перемещения персонажа по карте. Был приведен листинг алгоритма (листинг 1) и составлена схема алгоритма (рисунок 29).

4.2.2 Покрытие операторов

Критерии покрытия операторов [26] подразумевает выполнение каждого оператора программы по крайней мере один раз. Учитывая вышеизложенное условие, была составлена таблица покрытия операторов (таблица 13).

Таблица 13 – Покрытие операторов

Операторы	Входные данные	Выходные данные
aecfgi	moveDirection=Direction, obj=IDBomb	Успех
achi	moveDirection!=Direction, obj=IDCake	Успех

Таким образом была составлена таблица покрытия операторов, где каждый оператор был достигнут хотя бы раз.

4.2.3 Покрытие условий

Записывается число тестов достаточное для того, чтобы все возможные результаты каждого условия в решении были выполнены по крайней мере один раз [27]. Учитывая вышеизложенное условие, была составлена таблица покрытия условий (таблица 14).

Таблица 14 – Покрытие условий

Условия	Входные данные	Выходные данные
b-T,d-T	moveDirection=Direction, obj=IDBomb	Успех
d-F,d-F,j-T	moveDirection!=Direction, obj=IDCake	Успех
b-F,d-F,j-F	moveDirection!=Direction, obj=4	Передвижение не удалось

Таким образом была составлена таблица покрытия условий, где каждое условие было достигнуто хотя бы раз.

4.2.4 Покрытие маршрутов

В соответствии с этим критерием [28] необходимо составить тесты так, чтобы результаты каждого условия выполнялись хотя бы один раз, результаты каждого решения так же выполнялись хотя бы один раз, и каждый оператор должен быть выполнен хотя бы один раз. Учитывая это условие была составлена таблица покрытия маршрутов (таблица 15).

Таблица 15 – Покрытие маршрутов

Условия	Входные данные	Выходные данные
abecdfgi	moveDirection=Direction, obj=IDBomb	Успех
abecdjhi	moveDirection=Direction, obj=IDCake	Успех
abecdji	moveDirection=Direction, obj=4	Передвижение не удалось
abcdfgi	moveDirection!=Direction, obj=IDBomb	Успех
abcdjhi	moveDirection!=Direction, obj=IDCake	Успех
abcdji	moveDirection!=Direction, obj=4	Передвижение не удалось

Таким образом была составлена таблица покрытия маршрутов, где каждое условие выполнялось хотя бы один раз, каждый оператор был достигнут хотя бы раз и каждое решение выполнялось хотя бы раз.

4.3 Метрики кода

Метриками кода [29] называются некоторые количественные величины, позволяющие подсчитать, насколько код программы обладает тем или иным свойством. Существует огромное количество самых разнообразных метрик, имеющих различное назначение. Наиболее известными являются следующие группы метрик:

– метрики размера программы (SLOC-метрики);

– метрики сложности потока управления программ (метрики Холстеда и цикломатическая сложность).

Для данного метода будут вычислены три группы метрик: цикломатическая сложность, метрики размера программы и метрики Холстеда.

4.3.1 Цикломатическая сложность

Цикломатическая сложность программы [30] – структурная (или топологическая) мера сложности компьютерной программы.

При вычислении цикломатической сложности используется граф потока управления программы. Узлы графа соответствуют неделимым группам команд программы, они соединены ориентированными рёбрами, если группа команд, соответствующая второму узлу, может быть выполнена непосредственно после группы команд первого узла. Цикломатическая сложность может быть также вычислена для отдельных функций, модулей, методов или классов в пределах программы.

Для подсчёта цикломатической сложности был составлен управляющий граф программы (рисунок 30) по представленному ранее алгоритму (рисунок 29).

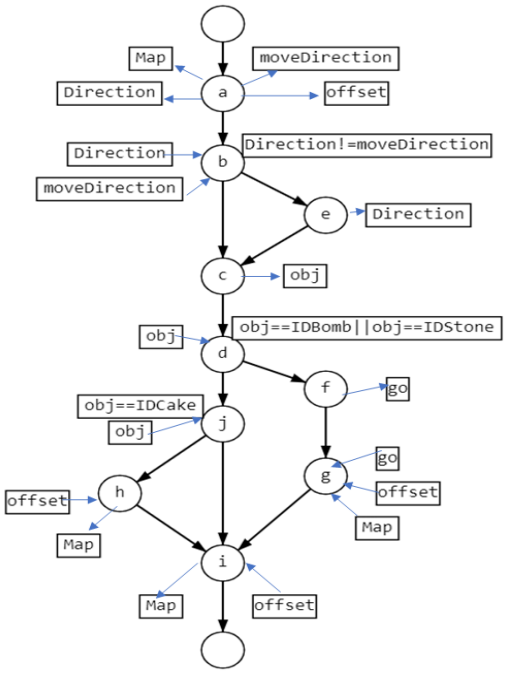

Рисунок 30 – Управляющий граф программы

На основе данного графа была вычислена цикломатическая сложность программного кода:

– Количество дуг

$$E = 13;$$

– Количество вершин

$$N = 10;$$

– Количество компонент связности

$$P = 1;$$

– Цикломатическая сложность

$$CC = E - N + 2P = 13 - 10 + 2 = 5.$$

Таким образом цикломатическая сложность алгоритма $CC = 5$.

4.3.2 Метрики размера программы

Данные метрики представляют собой подсчет количества строк программного кода, а также числа строк с комментариями. Вычислим метрики размера программы для приведенного выше программного кода. Результаты представлены в таблице 16.

Таблица 16 - Метрики размера программы

Метрика	Количество
Количество строк кода	28
Строки-комментарии	8

Также вычислим процент комментариев. Он равен $(8 / 28) * 100 = 29$. Это свидетельствует о достаточном количестве комментариев в коде для его хорошего понимания при отсутствии избыточных комментариев, переполняющих программу.

4.3.3 Метрики Холстеда

Метрики Холстеда предлагают разумный подход к решению следующих задач:

– предсказание условий, необходимых для программирования по предложенным проектам;

– определение норм первоначальных ошибок;

– количественная оценка языков программирования и эффекта модульности;

– обоснование метода измерения различий между программами, написанными специалистами разного уровня.

В основе вычисления метрик Холстеда лежит концепция, согласно которой алгоритм состоит только из операторов и операндов (проверяется рассмотрением простых вычислительных машин с форматом команд, содержащим две части: код операции и адрес операнда). Операнды – переменные или константы, используемые в данной реализации алгоритма. Операторы-комбинации символов, влияющие на значение или порядок операндов.

Для вычисления метрик Холстеда необходимо измерить 4 характеристики программы [31 – 33]:

– число уникальных операторов;

– число уникальных операндов;

– общее число операторов;

– общее число операндов.

С целью вычисления этих характеристик были составлен словарь операторов (таблица 17).

Таблица 17 – Словарь операторов

Оператор	№	Кол-во
if	1	3
.	2	6
=	3	3
==	4	3
||	5	1
!=	6	1
;	7	7
else	8	1
+	9	3
Всего	9	28
	n1	N1

А также был составлен словарь операндов (таблица 18).

Таблица 18 – Словарь операндов

Операнд	№	Кол-во
Direction	1	2
moveDirection	2	2
obj	3	4
IDBomb	4	1
IDStone	5	1
go	6	1
IDCake	7	1
Position	8	3
offset	9	3
Map	10	1
Initializer	11	3
listOfObjects	12	3
GetObj	13	1
Event	14	1
EatDiamond	15	1
Всего	15	28
	n2	N2

Таким образом, число уникальных операторов программы $n1 = 9$, число уникальных операндов программы $n2 = 15$, общее число операторов в программе $N1 = 28$, общее число операндов в программе $N2 = 28$.

На основе метрик Холстеда можно вычислить целый ряд характеристик программы. Рассмотрим некоторые из них:

– словарь программы (n):

$$n = n1 + n2;$$

– длина программы(N):

$$N = N1 + N2;$$

– объем программы (V):

$$V = N * log_2 n;$$

– сложность программы (D):

$$D = (n1 / 2) * (N2 / n2);$$

– оценка интеллектуальных усилий (E):

$$E = D * V.$$

Рассчитаем словарь программы:

$$n = 12 + 7 = 24.$$

Рассчитаем длину программы:

$$N = 36 + 12 = 56.$$

Рассчитаем объем программы:

$$V = 56 * \log_2 24 = 256{,}8.$$

Рассчитаем сложность программы:

$$D = (9 / 2) * (28 / 15) = 8{,}4.$$

Рассчитаем оценку интеллектуальных усилий:

$$E = 8{,}4 * 256{,}8 = 2157{,}96.$$

Полученные характеристики показывают, что программный код имеет достаточно высокий уровень качества и удобства для дальнейшей поддержки.

Заключение

В ходе выполнения выпускной квалификационной работы было разработано игровое приложение с использование средств мультиплатформенной разработки Unity.

На этапе анализа требований и предметной области был выполнен анализ программных продуктов аналогичного назначения, в результате которого были выявлены их основные достоинства и недостатки, определены функциональные и нефункциональные требования к создаваемому приложению и произведен анализ целевой аудитории.

Было осуществлено планирование разработки приложения и оценка бюджета. Результаты планирования показали, что общая длительность проекта составляет 50 дней, а предполагаемые затраты на реализацию – 135 320 рублей, что свидетельствует об экономической целесообразности данного программного проекта.

На этапе проектирования была составлена разработана архитектура игрового приложения. Приложение было разработано с помощью платформы разработки Unity 2020.3.8f на языке C# с использованием среды программирования Microsoft Visual Studio 2019.

Было выполнено тестирование игрового приложения. Оно показало корректность его работы. Также рассчитаны цикломатическая сложность и метрики программного кода для оценки его качества.

В дальнейшем планируется добавить в приложение новые игровые механики, а также портировать приложение под мобильные устройства и опубликовать в Steam и Google Play.

Таким образом, в ходе выполнения выпускной квалификационной работы были успешно выполнены все поставленные задачи.

Список использованных источников

1. Распоряжение Президента Российской Федерации от 9 мая 2017 г. № 203 «Стратегия развития информационного общества в Российской Федерации на 2017 – 2030 годы». [Электронный ресурс]. URL: http://zmedu.hostedu.ru/files/ykaz_7668.pdf (дата обращения: 21.04.2022).

2. Программа «Цифровая экономика Российской Федерации». Утверждена распоряжением Правительства Российской Федерации от 28 июля 2017 г. № 1632-р. [Электронный ресурс]. URL: http://static.government.ru/media/files/9gFM4FHj4PsB79I5v7yLVuPgu4bvR7M0.pdf (дата обращения: 21.04.2022).

3. Описание продукта // Официальный сайт Unity [Электронный ресурс]. URL: https://unity.com/ (дата обращения: 21.04.2022).

4. Жанры компьютерных игр // Открытое программное обеспечение [Электронный ресурс]. URL: http://www.4stud.info/oss/computergame-genres.html

5. Жанры компьютерных игр (новый и оригинальный метод разделения). Схема жанров // GamesIsArt.ru Компьютерные игры как искусство [Электронный ресурс]. URL: http://gamesisart.ru/TableJanr.html (дата обращения: 21.04.2022).

6. Boulder Dash – 30th Anniversary // Boulder Dash – 30th Anniversary Official game webpage [Электронный ресурс]. URL: https://boulder-dash.com/30th-anniversary/ (дата обращения: 21.04.2022).

7. Boulder Dash – Deluxe THE 2021 VERSION FEATURES // Boulder Dash – Deluxe game webpage [Электронный ресурс]. URL: https://boulder-dash.com/deluxe/ (дата обращения: 21.04.2022).

8. Boulder Dash-XL 3D // Nintendo 3DS [Электронный ресурс]. URL: https://www.nintendo.ru/-/Nintendo-3DS/Boulder-Dash-XL-3D-793405.html (дата обращения: 21.04.2022).

9. Портативная игровая система производства Nintendo // Nintendo Official Site: Consoles, Games, News, and More [Электронный ресурс]. URL: https://ru.wikipedia.org/wiki/Nintendo_3DS (дата обращения: 21.04.2022).

10. Игровая индустрия за 2020 год в цифрах // DTF – платформа для обсуждения игр, кино, разработки и технологий [Электронный ресурс]. URL: https://dtf.ru/gameindustry/294571-igrovaya-industriya-za-2020-god-v-cifrah-infografika-ot-gamesindustry-biz (дата обращения: 21.04.2022).

11. Данные об оборудовании и ПО пользователей // Steam – платформа для игроков и разработчиков [Электронный ресурс]. URL: https://store.steampowered.com/hwsurvey (дата обращения: 21.04.2022).

12. Explore the world of Windows 11 // Microsoft [Электронный ресурс]. URL: https://www.microsoft.com/en-us/windows/windows-11?r=1 (дата обращения: 21.04.2022).

13. Методология разработки программного обеспечения RUP. Рабочие потоки, действия и артефакты стадий [Электронный ресурс]. URL: https://skarlupka.ru/articles.php?id=38 (дата обращения: 05.05.2022).

14. Ехлаков, Ю.П. Управление программными проектами. Стандарты, модели: Учебное пособие / Ю.П. Ехлаков. – СПб.: Лань, 2019. – 244 с.

15. Иванов, П.В. Управление проектами: Учебное пособие / П.В. Иванов. – Рн/Д: Феникс, 2018. –102 с.

16. Гонтарева, И.В. Управление проектами / И.В. Гонтарева, Р.М. Нижегородцев, Д.А. Новиков. – М.: КД Либроком, 2018. – 384 с.

17. Ньютон, Р. Управление проектами от А до Я / Р. Ньютон. – М.: Альпина Паблишер, 2018. – 180 с.

18. Вольфсон, Б.Л. Гибкое управление проектами и продуктами / Б.Л. Вольфсон. - СПб.: Питер, 2019. – 208 с.

19. Унифицированный процесс Rational (RUP) // Интернет-технологии.ру [Электронный ресурс]. – URL: https://www.internet-technologies.ru/articles/newbie/unificirovannyy-process-rational-rup.html (дата обращения: 05.05.2022).

20. Арлоу, Д. UML 2 и Унифицированный процесс. Практический объектно-ориентированный анализ и проектирование, 2-е издание / Д. Арлоу, А. Нейштадт – СПб: Символ Плюс, 2007. – 624 с.

21. Бонд Д. Г. Unity и C#. Геймдев от идеи до реализации, 2-е издание / Бонд Д. Г. // Питер, 2022. – 928 с.

22. Создание архитектуры UI на Unity // DTF – платформа для обсуждения игр, кино, разработки и технологий [Электронный ресурс]. URL: https://dtf.ru/gamedev/14464-sozdanie-arhitektury-ui-na-unity (дата обращения: 05.05.2022).

23. Тестирование методом черного ящика // habr.com [Электронный ресурс]. URL: https://habr.com/ru/post/462837/ (дата обращения: 27.05.2022).

24. Классы эквивалентности // НОУ «ИНТУИТ» [Электронный ресурс]. URL: https://intuit.ru/studies/courses/1040/209/lecture/5388 (дата обращения: 27.05.2022).

25. ТЕХНИКА АНАЛИЗА ГРАНИЧНЫХ ЗНАЧЕНИЙ// QA Evolution [Электронный ресурс]. URL: https://qaevolution.ru/testovaya-dokumentaciya/test-dizajn/texnika-analiza-granichnyx-znachenij/ (дата обращения: 27.05.2022).

26. Метод покрытия операторов // Оценка качества программного обеспечения вычислительных систем tstu.ru [Электронный ресурс]. URL: https://tstu.ru/book/elib3/mm/2015/loskutov/pages/4_3_1_metod_pokrytija_operatorov.html (дата обращения: 27.05.2022).

27. Метод покрытия условий // Оценка качества программного обеспечения вычислительных систем tstu.ru [Электронный ресурс]. URL: https://tstu.ru/book/elib3/mm/2015/loskutov/pages/4_3_3_metod_pokrytija_uslovij.html (дата обращения: 27.05.2022).

28. Тестирование маршрутов. Структурное тестирование. // https://studopedia.ru/ [Электронный ресурс]. URL: https://studopedia.ru/9_40672_strukturnoe-testirovanie.html (дата обращения: 27.05.2022).

29. Метрики кода и практическая реализация по их сбору и анализу. Часть 1 – метрики // https://club.cnews.ru/ [Электронный ресурс]. URL: https://club.cnews.ru/blogs/entry/metriki_koda_i_prakticheskaya_realizatsiya_po_ih_sboru_i_analizu_chast_1_ (дата обращения: 27.05.2022).

30. Цикломатическое тестирование // уроки, учебники и другие материалы по программированию https://coderlessons.com/ [Электронный ресурс]. URL: https://coderlessons.com/tutorials/kachestvo-programmnogo-obespecheniia/ruchnoe-testirovanie/tsiklomaticheskoe-testirovanie (дата обращения: 27.05.2022).

31. Программный код и его метрики // habr.com [Электронный ресурс]. URL: https://habr.com/ru/company/intel/blog/106082/ (дата обращения: 27.05.2022).

32. Аверьянов, А.В. Применение метрик Холстеда для количественного оценивания характеристик программ ЭВМ / А.В. Аверьянов, И.Н. Кошель, В.В. Кузнецов // Известия высших учебных заведений. Приборостроение. – 2019. – №11. – С. 970 – 975.

33. Полякова, Э.Р. Исследование программного обеспечения встраиваемой вычислительной системы метриками Холстеда / Э.Р. Полякова, А.А. Поляков, А.Л.Шустов, Д.Г. Придачкин // Cloud of Science. – 2020. – №4.– С. 950 – 959.

Приложение А

ГЛОССАРИЙ

(справочное)

RUP (Rational Unified Process) – методология разработки программного обеспечения, в основе которой лежит итеративная модель разработки.

UML (Unified Modeling Language) – язык графического описания для объектного моделирования в области разработки программного обеспечения, моделирования бизнес-процессов, системного проектирования и отображения организационных структур.

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

Видеоигра – это компьютерная программа, главная суть которой во взаимодействии человека и компьютерного устройства с помощью средств ввода и вывода информации, например посредством клавиатуры и мыши происходит управление игровым персонажем, а посредством монитора или телевизора игровое приложение сообщает текущее состояние игры.

UI (User Interface) – интерфейс, обеспечивающий передачу информации между пользователем-человеком и программно-аппаратными компонентами компьютерной системы.

Приложение Б

ИСХОДНЫЙ КОД ИГРОВОГО ПРИЛОЖЕНИЯ

(рекомедуемое)

Листинг Б.1 – Файл AbstractObjectController.cs

```
using System.Linq;
using UnityEngine;

namespace Game.ObjectsControllers
{
    public abstract class AbstractObjectController : MonoBehaviour
    {
        protected int objId;

        public int ObjId
        {
            get => objId;
            protected set => objId = value;
        }

        protected int[] moveWhiteList;

        public int[] MoveWhiteList
        {
            get => moveWhiteList;
            protected set => moveWhiteList = value;
        }

        protected ObjectInitializer Initializer;
        protected MapController Map;
        protected GameEventController Event;

        //How long it will take to pass 1 block in milliseconds
        protected const int RunSpeed = 100;

        protected bool _IsDead = false;

        public virtual bool IsDead
        {
            get => _IsDead;
            protected set { _IsDead = value; }
        }

        protected int x;
        protected int y;

        public int X
        {
            get => x;
            protected set
            {
                gameObject.transform.position = new Vector3(value, Y, 0);
                x = value;
            }
        }

        public int Y
        {
            get => y;
            protected set
            {
                gameObject.transform.position = new Vector3(X, value, 0);
                y = value;
            }
        }


        public string startDirection;
```

```
        protected string direction;

        //this property needs to change sprite when we change direction
        public virtual string Direction
        {
            get { return direction; }

            set { direction = value; }
        }

        public Vector3 Position
        {
            get => gameObject.transform.position;
            protected set
            {
                gameObject.transform.position = value;
                x = (int) value.x;
                y = (int) value.y;
            }
        }

        protected void Initialize()
        {
            Initializer = FindObjectOfType<ObjectInitializer>();
            Event = FindObjectOfType<GameEventController>();
            Map = FindObjectOfType<MapController>();
            x = (int) Position.x;
            y = (int) Position.y;
            direction = startDirection;
        }

        protected abstract void MovementLogic();

        protected virtual bool CanMove(int[] whiteList, Vector3 offset)
        {
            return whiteList.Contains(Map.GetObj(Position + offset));
        }

        protected virtual void Move(Vector3 offset)
        {
            Map.MoveObjToPosition(this, Position + offset);
        }

        protected virtual void PushObject(AbstractObjectController obj,
Vector3 offset)
        {
            Map.MoveObjToPosition(obj, obj.Position + offset);
        }
    }
}
```

Листинг Б.2 – Файл BombController.cs

```
using System.Linq;
using UnityEngine;
using static DataScript;

namespace Game.ObjectsControllers
{
    public class BombController : AbstractObjectController
    {
        private int _time = 0;

        private bool _speed = false;
```

```
        void Start()
        {
            Initialize();
            objId = IDBomb;
            MoveWhiteList = WhiteListBomb;
        }

        void Update()
        {
            if (!Event.isGameStopped)
            {
                MovementLogic();
            }
        }


        protected override void MovementLogic()
        {
            if (CanMove(MoveWhiteList, Vector3.down) ||
                _speed && CanMove(WhiteListBombWhenFall, Vector3.down))
            {
                _time += (int) (Time.deltaTime * 1000);
                _speed = true;
                if (_time >= RunSpeed)
                {
                    _time = 0;
                    Move(Vector3.down);
                }
            }
            else if (_speed)
            {
                Initializer.KillObject(this);
            }
        }

        private void Explode()
        {
            var a = GetMatrix();

            foreach (var di in new[] {-1, 0, 1})
            {
                foreach (var dj in new[] {-1, 0, 1})
                {
                    if (!a[2 + di, 2 + dj])
                    {
                        foreach (var dx in new[] {-1, 0, 1})
                        {
                            foreach (var dy in new[] {-1, 0, 1})
                            {
                                if ((di + dx) * (di + dx) + (dj + dy) * (dj +
dy) > 2 * di * di + 2 * dj * dj)
                                {
                                    a[2 + di + dx, 2 + dj + dy] = false;
                                }
                            }
                        }
                    }
                }
            }

            for (int i = 0; i < 5; i++)
            {
                for (int j = 0; j < 5; j++)
                {
                    if (a[i, j])
```

```
                    {
                        Vector3 position = Position + Vector3.down * (i - 2)
+ Vector3.right * (j - 2);
                        Map.CleanCell(position);
                    }
                }
            }
        }

        private bool[,] GetMatrix()
        {
            bool[,] toReturn = new bool[5, 5];

            for (int i = 0; i < 5; i++)
            {
                for (int j = 0; j < 5; j++)
                {
                    Vector3 position = Position + Vector3.down * (i - 2) +
Vector3.right * (j - 2);
                    int obj = Map.GetObj(position);

                    if (BlackListForExplosion.Contains(obj))
                    {
                        toReturn[i, j] = false;
                    }
                    else
                    {
                        toReturn[i, j] = true;
                    }
                }
            }

            return toReturn;
        }

        private void OnDestroy()
        {
            if (!Event.isGameStopped)
                Explode();
        }

        protected override bool CanMove(int[] whiteList, Vector3 offset)
        {
            bool playerHold = !_speed && Map.GetObj(Position + offset) ==
IDPlayer;

            return !playerHold && base.CanMove(whiteList, offset);
        }
    }
}
```

Листинг В.3 – Файл CakeController.cs

```
using UnityEngine;
using static DataScript;

namespace Game.ObjectsControllers
{
    public class CakeController : AbstractObjectController
    {
        private int _time = 0;
        private bool _speed = false;

        void Start()
        {
```

```
            Initialize();
            objId = IDCake;
            MoveWhiteList = WhiteListDiamond;
        }

        private void Update()
        {
            if (!Event.isGameStopped)
            {
                MovementLogic();
            }
        }

        protected override void MovementLogic()
        {
            if (CanMove(MoveWhiteList, Vector3.down) ||
                _speed && CanMove(WhiteListDiamondWhenFall, Vector3.down))
            {
                _time += (int) (Time.deltaTime * 1000);
                _speed = true;
                if (_time >= RunSpeed)
                {
                    _time = 0;
                    Move(Vector3.down);
                }
            }
            else
            {
                _speed = false;
            }
        }

        protected override bool CanMove(int[] whiteList, Vector3 offset)
        {
            bool playerHold = !_speed && Map.GetObj(Position + offset) ==
IDPlayer;

            return !playerHold && base.CanMove(whiteList, offset);
        }
    }
}
```

Листинг Б.4 – Файл EnemyLeftController.cs

```
using UnityEngine;
using static DataScript;

namespace Game.ObjectsControllers
{
    public class EnemyLeftController : AbstractObjectController
    {
        private int _time = 0;

        void Start()
        {
            Initialize();
            objId = IDEnemyLeft;
            MoveWhiteList = WhiteListEnemy;
        }

        void Update()
        {
            if (!IsDead && !Event.isGameStopped)
            {
                DeathLogic();
```

```
            MovementLogic();
        }
    }

    protected override void MovementLogic()
    {
        _time += (int) (Time.deltaTime * 1000);
        if (_time >= RunSpeed)
        {
            _time = 0;
            switch (Direction)
            {
                case "up":
                    if (CanMove(MoveWhiteList, Vector3.up))
                    {
                        Move(Vector3.up);
                    }
                    else
                    {
                        Direction = "left";
                    }

                    break;

                case "down":
                    if (CanMove(MoveWhiteList, Vector3.down))
                    {
                        Move(Vector3.down);
                    }
                    else
                    {
                        Direction = "right";
                    }

                    break;

                case "right":
                    if (CanMove(MoveWhiteList, Vector3.right))
                    {
                        Move(Vector3.right);
                    }
                    else
                    {
                        Direction = "up";
                    }

                    break;

                case "left":
                    if (CanMove(MoveWhiteList, Vector3.left))
                    {
                        Move(Vector3.left);
                    }
                    else
                    {
                        Direction = "down";
                    }

                    break;
            }
        }
    }

    private void DeathLogic()
    {
        // TileBase tile = Map.GetTile(Map.WorldToCell(Position));
```

```
            //
            // if (CanDieBy.Contains(tile.name))
            // {
            //     IsDead = true;
            // }
        }
    }
}
```

Листинг Б.5 – Файл EnemyRightController.cs

```
using UnityEngine;
using static DataScript;

namespace Game.ObjectsControllers
{
    public class EnemyRightController : AbstractObjectController
    {
        private int _time = 0;

        private void Start()
        {
            Initialize();
            objId = IDEnemyRight;
            MoveWhiteList = WhiteListEnemy;
        }

        private void Update()
        {
            if (!IsDead && !Event.isGameStopped)
            {
                DeathLogic();
                MovementLogic();
            }
        }

        protected override void MovementLogic()
        {
            _time += (int) (Time.deltaTime * 1000);
            if (_time >= RunSpeed)
            {
                _time = 0;
                switch (Direction)
                {
                    case "up":
                        if (CanMove(MoveWhiteList, Vector3.up))
                        {
                            Move(Vector3.up);
                        }
                        else
                        {
                            Direction = "right";
                        }

                        break;

                    case "down":
                        if (CanMove(MoveWhiteList, Vector3.down))
                        {
                            Move(Vector3.down);
                        }
                        else
                        {
                            Direction = "left";
                        }
```

```
                        break;

                    case "right":
                        if (CanMove(MoveWhiteList, Vector3.right))
                        {
                            Move(Vector3.right);
                        }
                        else
                        {
                            Direction = "down";
                        }

                        break;

                    case "left":
                        if (CanMove(MoveWhiteList, Vector3.left))
                        {
                            Move(Vector3.left);
                        }
                        else
                        {
                            Direction = "up";
                        }

                        break;
                }
            }
        }

        private void DeathLogic()
        {
            // TileBase tile = Map.GetTile(Map.WorldToCell(Position));
            //
            // if (CanDieBy.Contains(tile.name))
            // {
            //     IsDead = true;
            // }
        }
    }
}
```

Листинг Б.6 – Файл PlayerController.cs

```
using System.Linq;
using UnityEngine;
using static DataScript;

namespace Game.ObjectsControllers
{
    public class PlayerController : AbstractObjectController
    {
        //These two integers means milliseconds
        private int _time = 0;
        private const int WaitBeforeRun = 200;

        private int _bombCount;


        public override string Direction
        {
            get { return direction; }

            set { direction = value; }
        }
```

```
        void Start()
        {
            Initialize();
            objId = IDPlayer;
            MoveWhiteList = WhiteListPlayer;
            _bombCount = 9;
            Event.bombCount.text = _bombCount.ToString();
        }

        void Update()
        {
            if (!IsDead && !Event.isGameStopped)
            {
                MovementLogic();
                DeathLogic();
            }
        }

        protected override void MovementLogic()
        {
            // bool up = Input.GetKey(KeyCode.UpArrow) ||
Input.GetKey(KeyCode.W);
            // bool down = Input.GetKey(KeyCode.DownArrow) ||
Input.GetKey(KeyCode.S);
            // bool right = Input.GetKey(KeyCode.UpArrow) ||
Input.GetKey(KeyCode.D);
            // bool left = Input.GetKey(KeyCode.UpArrow) ||
Input.GetKey(KeyCode.A);
            bool shift = Input.GetKey(KeyCode.LeftShift);

            if (shift)
            {
                if (Input.GetKeyDown(KeyCode.UpArrow) &&
CanMove(WhiteListBomb, Vector3.up))
                {
                    SpawnBomb(Position + Vector3.up);
                }
                else if (Input.GetKeyDown(KeyCode.DownArrow) &&
CanMove(WhiteListBomb, Vector3.down))
                {
                    SpawnBomb(Position + Vector3.down);
                }
                else if (Input.GetKeyDown(KeyCode.RightArrow) &&
CanMove(WhiteListBomb, Vector3.right))
                {
                    SpawnBomb(Position + Vector3.right);
                }
                else if (Input.GetKeyDown(KeyCode.LeftArrow) &&
CanMove(WhiteListBomb, Vector3.left))
                {
                    SpawnBomb(Position + Vector3.left);
                }
            }

            else if (Input.GetKeyDown(KeyCode.UpArrow) &&
CanMove(MoveWhiteList, Vector3.up))
            {
                Move("up", Vector3.up);
            }

            else if (Input.GetKey(KeyCode.UpArrow) && CanMove(MoveWhiteList,
Vector3.up))
            {
                if (_time >= WaitBeforeRun)
                {
```

```
                    if (_time - WaitBeforeRun >= RunSpeed)
                    {
                        Move("up", Vector3.up);
                        _time = WaitBeforeRun;
                    }
                }

                _time += (int) (Time.deltaTime * 1000);
            }

            else if (Input.GetKeyUp(KeyCode.UpArrow))
            {
                _time = 0;
            }


            else if (Input.GetKeyDown(KeyCode.DownArrow) &&
CanMove(MoveWhiteList, Vector3.down))
            {
                Move("down", Vector3.down);
            }

            else if (Input.GetKey(KeyCode.DownArrow) &&
CanMove(MoveWhiteList, Vector3.down))
            {
                if (_time >= WaitBeforeRun)
                {
                    if (_time - WaitBeforeRun >= RunSpeed)
                    {
                        Move("down", Vector3.down);
                        _time = WaitBeforeRun;
                    }
                }

                _time += (int) (Time.deltaTime * 1000);
            }

            else if (Input.GetKeyUp(KeyCode.DownArrow))
            {
                _time = 0;
            }


            else if (Input.GetKeyDown(KeyCode.RightArrow) &&
CanMove(MoveWhiteList, Vector3.right))
            {
                Move("right", Vector3.right);
            }

            else if (Input.GetKey(KeyCode.RightArrow) &&
CanMove(MoveWhiteList, Vector3.right))
            {
                if (_time >= WaitBeforeRun)
                {
                    if (_time - WaitBeforeRun >= RunSpeed)
                    {
                        Move("right", Vector3.right);
                        _time = WaitBeforeRun;
                    }
                }

                _time += (int) (Time.deltaTime * 1000);
            }

            else if (Input.GetKeyUp(KeyCode.RightArrow))
            {
```

```
                _time = 0;
            }


            else if (Input.GetKeyDown(KeyCode.LeftArrow) &&
CanMove(MoveWhiteList, Vector3.left))
            {
                Move("left", Vector3.left);
            }

            else if (Input.GetKey(KeyCode.LeftArrow) &&
CanMove(MoveWhiteList, Vector3.left))
            {
                if (_time >= WaitBeforeRun)
                {
                    if (_time - WaitBeforeRun >= RunSpeed)
                    {
                        Move("left", Vector3.left);
                        _time = WaitBeforeRun;
                    }
                }

                _time += (int) (Time.deltaTime * 1000);
            }

            else if (Input.GetKeyUp(KeyCode.LeftArrow))
            {
                _time = 0;
            }
        }

        private void DeathLogic()
        {
            // TileBase tile = Map.GetTile(Map.WorldToCell(Position));
            //
            // if (CanDieBy.Contains(tile.name))
            // {
            //     GEC.DiedScene();
            //     IsDead = true;
            // }
        }

        protected override bool CanMove(int[] whiteList, Vector3 offset)
        {
            int obj = Map.GetObj(Position + offset);

            bool toReturn = false;

            if (PlayerCanMove.Contains(obj))
            {
                var go = Initializer.listOfObjects[Position + offset];

                toReturn = base.CanMove(go.MoveWhiteList, offset + offset);
            }

            return toReturn || base.CanMove(whiteList, offset);
        }

        /// <summary>
        /// Функция передвижения
        /// </summary>
        /// <param name="moveDirection">Направление движения</param>
        /// <param name="offset">Вектор-смещение относительно текущей
позиции</param>
        private void Move(string moveDirection, Vector3 offset)
        {
```

```
            if (Direction != moveDirection)
                Direction = moveDirection;

            int obj = Map.GetObj(Position + offset);

            //Если на том месте куда мы движемся есть объекты, которые мы
можем толкать, то пытаемся их толкнуть
            if (obj == IDBomb || obj == IDStone)
            {
                var go = Initializer.listOfObjects[Position + offset];

                PushObject(go, offset);
            }
            //Если на том месте куда мы движемся есть объекты, которые мы
можем съесть, то пытаемся их съесть
            else if (obj == IDCake)
            {
                Event.EatDiamond();
                Initializer.KillObject(Initializer.listOfObjects[Position +
offset]);
            }

            base.Move(offset);
        }

        private void SpawnBomb(Vector3 position)
        {
            if (_bombCount > 0)
            {
                Event.SpendBomb();
                _bombCount--;
                Map.SetObj(IDBomb, position);
            }
        }
    }
}
```

Листинг Б.7– Файл StoneController.cs

```
using UnityEngine;
using static DataScript;

namespace Game.ObjectsControllers
{
    public class StoneController : AbstractObjectController
    {
        private int _time = 0;

        private bool _speed = false;

        void Start()
        {
            Initialize();
            objId = IDStone;
            MoveWhiteList = WhiteListStone;
        }

        void Update()
        {
            if (!Event.isGameStopped)
            {
                MovementLogic();
            }
        }
```

```
        protected override void MovementLogic()
        {
            if (CanMove(MoveWhiteList, Vector3.down) ||
                _speed && CanMove(WhiteListStoneWhenFall, Vector3.down))
            {
                _time += (int) (Time.deltaTime * 1000);
                _speed = true;
                if (_time >= RunSpeed)
                {
                    _time = 0;
                    Move(Vector3.down);
                }
            }
            else if (_speed && Map.GetObj(Position + Vector3.down) ==
IDStone)
            {
                if (CanMove(MoveWhiteList, Vector3.right) &&
CanMove(MoveWhiteList, Vector3.right + Vector3.down))
                {
                    Move(Vector3.right);
                }
                else if (CanMove(MoveWhiteList, Vector3.left) &&
CanMove(MoveWhiteList, Vector3.left + Vector3.down))
                {
                    Move(Vector3.left);
                }
                else
                {
                    _speed = false;
                }
            }
            else
            {
                _speed = false;
            }
        }

        protected override bool CanMove(int[] whiteList, Vector3 offset)
        {
            bool playerHold = !_speed && Map.GetObj(Position + offset) ==
IDPlayer;

            return !playerHold && base.CanMove(whiteList, offset);
        }
    }
}
```

Листинг Б.8 – Файл Follow.cs

```
using System;
using UnityEngine;

namespace Game.UI
{
    public class Follow : MonoBehaviour
    {
        public GameObject target;
        public float smooth = 0.1f;
        public bool follow = true;

        private void Start()
        {
            target = GameObject.FindGameObjectWithTag("Player");
        }
```

```
        void Update()
        {
            if (follow)
            {
                if (target == null)
                {
                    target = GameObject.FindGameObjectWithTag("Player");
                }
                else
                {
                    Vector3 position = target.transform.position;
                    Vector3 pos = new Vector3(position.x, position.y);
                    transform.position = Vector3.Lerp(transform.position,
pos, smooth);
                }
            }
        }
    }
}
```

Листинг Б.9 – Файл MapEditorController.cs

```
using System;
using System.Collections.Generic;
using Game.UI;
using UnityEngine;
using UnityEngine.EventSystems;
using UnityEngine.Tilemaps;
using UnityEngine.UI;
using UnityEngine.UIElements;
using static DataScript;

namespace MapEditor
{
    public class MapEditorController : MonoBehaviour
    {
        public Camera mainCamera;

        public Canvas MainCanvas;
        public Canvas WorldCanvas;

        //public GameObject sprite;

        public Tilemap map;

        private MapController _map;
        private ObjectInitializer _initializer;
        private GameEventController _event;
        private DataScript _data;
        private MapEditorUI _ui;
        private GlobalGameParameters GGP;

        public enum Mode
        {
            Draw,
            Select,
            Play,
            Watch
        }
        /*
                private int[,] MapArray
                {
                    get => Map.mapArray;
                    set => Map.mapArray = value;
                }
```

```
        private int[,] MapArrayBase
        {
            get => Map.mapArrayBase;
            set => Map.mapArrayBase = value;
        }*/

    private int _nBase;
    private int _mBase;

    private List<Vector3Int> _selectedCells;

    private Mode _mode;
    private int _selectedObj;

    private Vector3Int first;
    private bool firstB;
    private Vector3Int second;
    private bool secondB;

    private void Start()
    {
        GGP = GlobalGameParameters.GetInstance();
        _event = FindObjectOfType<GameEventController>();
        _event.isGameStopped = true;

        _map = FindObjectOfType<MapController>();
        _initializer = FindObjectOfType<ObjectInitializer>();
        _ui = FindObjectOfType<MapEditorUI>();
        _data = FindObjectOfType<DataScript>();
        //_lr = WorldCanvas.GetComponent<LineRenderer>();

        _selectedCells = new List<Vector3Int>();
        firstB = false;
        secondB = false;
        _selectedObj = IDEmpty;
        _mode = MDraw;

        _map.LoadLevel(GGP.CurrentLevelPath);

        _map.ExpandMapArray(1024, 1024);

        _event.isGameStopped = false;
    }

    private void Update()
    {
        switch (_mode)
        {
            case Mode.Draw:
                DrawUpdate();
                break;

            case Mode.Select:
                SelectUpdate();
                break;

            case Mode.Play:
                PlayUpdate();
                break;

            case Mode.Watch:
                WatchUpdate();
                break;
        }
    }
```

```
        private void DrawUpdate()
        {
            if (Input.GetMouseButtonDown(0))
            {
                if (!EventSystem.current.IsPointerOverGameObject())
                {
                    var mousePosition = Input.mousePosition;

                    // Debug.Log(mousePosition.ToString());

                    Vector3 clickPosition =
mainCamera.ScreenToWorldPoint(mousePosition);

                    Vector3 clickedCellPosition = new Vector3((float)
Math.Truncate(clickPosition.x),
                        (float) Math.Truncate(clickPosition.y));

                    _selectedObj = _ui.selectedObj;

                    if (_selectedObj != _map.GetObj(clickedCellPosition))
                    {
                        _map.SetObj(_selectedObj, clickedCellPosition);
                    }
                }
            }
        }

        private void SelectUpdate()
        {
            if (Input.GetMouseButtonDown(0))
            {
                if (!EventSystem.current.IsPointerOverGameObject())
                {
                    Vector3Int clickedCellPosition =
GetClickedCellPos(Input.mousePosition);

                    if (Input.GetKey(KeyCode.LeftControl))
                    {
                        SelectCell(clickedCellPosition);
                    }

                    /*
                    if (!firstB)
                    {
                        first = clickedCellPosition;
                        firstB = true;
                    }
                    else
                    {
                        second = clickedCellPosition;
                        secondB = true;

                        SelectCells(first, second);
                    }*/
                } // if (!EventSystem.current.IsPointerOverGameObject())
            } // if (Input.GetMouseButtonDown(0))
        }

        private void PlayUpdate()
        {
        }

        private void WatchUpdate()
        {
        }
```

```
private void SelectCell(Vector3Int position1)
{
    LineRenderer lr = new LineRenderer();
    lr.positionCount = 5;
    Vector3Int position2 = position1;

    position1 += Vector3Int.right;
    position1 += Vector3Int.up;
    /*
    _lr.SetPosition(0, position1);
    _lr.SetPosition(1, new Vector3Int(position2.x, position1.y, 0));
    _lr.SetPosition(2, position2);
    _lr.SetPosition(3, new Vector3Int(position1.x, position2.y, 0));
    _lr.SetPosition(4, position1);*/

    _selectedCells.Add(position1);
}

private void SelectCells(Vector3Int position1, Vector3Int position2)
{


    int maxX = Math.Max(position1.x, position2.x);
    int maxY = Math.Max(position1.y, position2.y);
    int minX = Math.Min(position1.x, position2.x);
    int minY = Math.Min(position1.y, position2.y);

    //rework this
    _selectedCells.Clear();

    for (int x = minX; x < maxX; x++)
    {
        for (int y = minY; y < maxY; y++)
        {
            _selectedCells.Add(new Vector3Int(x, y, 0));
        }
    }

    //rework this
    foreach (var selectedCell in _selectedCells)
    {
        _map.SetObj(_selectedObj, selectedCell);
    }
}

public void SwitchMode(Mode mode)
{
    switch (mode)
    {
        case Mode.Draw:
            _event.isGameStopped = true;
            break;

        case Mode.Select:
            _event.isGameStopped = true;
            break;

        case Mode.Play:
            _event.isGameStopped = false;
            break;

        case Mode.Watch:
            _event.isGameStopped = true;
            break;
    }
```

```
            _mode = mode;
        }

        private Vector3Int GetClickedCellPos(Vector3 pos)
        {
            Vector3 clickPosition = mainCamera.ScreenToWorldPoint(pos);
            Vector3Int clickedCellPosition = new Vector3Int((int)
Math.Truncate(clickPosition.x),
                (int) Math.Truncate(clickPosition.y),
                0);
            return clickedCellPosition;
        }
    }
}
```

Листинг Б.10 – Файл MapEditorUI.cs

```
using System;
using System.Collections.Generic;
using UnityEngine;
using static DataScript;

namespace MapEditor
{
    public class MapEditorUI : MonoBehaviour
    {
        public int selectedObj;

        [SerializeField]
        private Canvas playerCanvas;
        [SerializeField]
        private Camera playerCamera;

        [NonSerialized]
        public int cameraHeight;
        [NonSerialized]
        public int cameraWidth;

        private float _kX;
        private float _kY;

        [SerializeField]
        private Canvas _mapEditorCanvas;

        private List<LineRenderer> _lrs;

        private void Start()
        {
            selectedObj = IDEmpty;

            cameraHeight = playerCamera.pixelHeight;
            cameraWidth = playerCamera.pixelWidth;

            _kX = cameraWidth / 1920f;
            _kY = cameraHeight / 1080f;
        }

        private void Update()
        {
            // cameraHeight = playerCamera.pixelHeight;
            // cameraWidth = playerCamera.pixelWidth;
            //
            // _kX = cameraWidth / 1920;
            // _kY = cameraHeight / 1920;
```

```
        }

        public bool InBlindSpot(Vector3 mousePosition)
        {
            // if (mousePosition.x > )


            return false;
        }

        public void DropBoxFunc(int value)
        {
            switch (value)
            {
                case 0:
                    selectedObj = IDEmpty;
                    break;

                case 1:
                    selectedObj = IDDirt;
                    break;

                case 2:
                    selectedObj = IDRock;
                    break;

                case 3:
                    selectedObj = IDPlayer;
                    break;

                case 4:
                    selectedObj = IDEnemyLeft;
                    break;

                case 5:
                    selectedObj = IDEnemyRight;
                    break;

                case 6:
                    selectedObj = IDStone;
                    break;

                case 7:
                    selectedObj = IDBomb;
                    break;

                case 8:
                    selectedObj = IDCake;
                    break;
            }
        }

        #region Highlight
        private void HighlightRectangle(Vector3Int position1, Vector3Int
position2)
        {
            LineRenderer lr =
_mapEditorCanvas.gameObject.AddComponent<LineRenderer>();
            lr.positionCount = 5;

            int deltaX = position2.x - position1.x;
            int deltaY = position2.y - position1.y;

            if (deltaX <= 0)
            {
                position1 += Vector3Int.right;
```

```
            }
            else
            {
                position2 += Vector3Int.right;
            }

            if (deltaY <= 0)
            {
                position1 += Vector3Int.up;
            }
            else
            {
                position2 += Vector3Int.up;
            }

            lr.SetPosition(0, position1);
            lr.SetPosition(1, new Vector3Int(position2.x, position1.y, 0));
            lr.SetPosition(2, position2);
            lr.SetPosition(3, new Vector3Int(position1.x, position2.y, 0));
            lr.SetPosition(4, position1);
        }

        private void HighlightRectangle(Vector3Int position1)
        {
            Vector3Int position2 = position1;
            position1 += Vector3Int.right;
            position1 += Vector3Int.up;
            HighlightRectangle(position1, position2);
        }
        #endregion
    }
}
```

Листинг Б.11 – Файл DataScript.cs

```
using System.Collections;
using System.Collections.Generic;
using Game.ObjectsControllers;
using UnityEngine;
using UnityEngine.Tilemaps;

public class DataScript : MonoBehaviour
{
    public const int IDEmpty = 0;
    public const int IDDirt = 1;
    public const int IDRock = 2;
    public const int IDPlayer = 100;
    public const int IDEnemyLeft = 110;
    public const int IDEnemyRight = 120;
    public const int IDStone = 200;
    public const int IDBomb = 210;
    public const int IDCake = 220;

    public const byte MDraw = 0;
    public const byte MSelect = 1;
    public const byte MPlay = 2;
    public const byte MWatch = 3;

    #region BlackLists

    public static readonly int[] BlackListForPlayer = {IDRock, IDStone,
IDEnemyLeft, IDEnemyRight, IDBomb};
    public static readonly int[] BlackListForStoneWhenFall = {IDRock, IDDirt,
IDStone, IDCake};
```

```
    public static readonly int[] BlackListForDiamondWhenFall = {IDRock,
IDDirt, IDStone, IDBomb, IDCake};
    public static readonly int[] BlackListForStoneWhenMove =
        {IDRock, IDDirt, IDStone, IDEnemyLeft, IDEnemyRight, IDBomb, IDCake};
    public static readonly int[] BlackListForEnemy =
        {IDRock, IDDirt, IDStone, IDEnemyLeft, IDEnemyRight, IDBomb, IDCake};
    public static readonly int[] BlackListForBomb =
        {IDRock, IDDirt, IDStone, IDEnemyLeft, IDEnemyRight, IDBomb, IDCake,
IDPlayer};
    public static readonly int[] EnemyCanDieBy = {IDStone, IDCake};
    public static readonly int[] PlayerCanDieBy = {IDStone, IDEnemyLeft,
IDEnemyRight, IDCake};

    #endregion

    public static readonly int[] BlackListForExplosion = {IDRock};

    public static readonly int[] PlayerCanMove = {IDStone, IDBomb};

    public static readonly int[] WhiteListPlayer = {IDEmpty, IDDirt, IDCake};
    public static readonly int[] WhiteListEnemy = {IDEmpty, IDPlayer};
    public static readonly int[] WhiteListDiamondWhenFall = {IDEmpty,
IDPlayer, IDEnemyLeft, IDEnemyRight};
    public static readonly int[] WhiteListDiamond = {IDEmpty};
    public static readonly int[] WhiteListStoneWhenFall = {IDEmpty, IDPlayer,
IDEnemyLeft, IDEnemyRight};
    public static readonly int[] WhiteListStone = {IDEmpty};
    public static readonly int[] WhiteListBombWhenFall = {IDEmpty};
    public static readonly int[] WhiteListBomb = {IDEmpty};

    public AbstractObjectController prefabStone;
    public AbstractObjectController prefabEnemyRight;
    public AbstractObjectController prefabEnemyLeft;
    public AbstractObjectController prefabPlayer;
    public AbstractObjectController prefabBomb;
    public AbstractObjectController prefabCake;

    public TileBase emptyTile;
    public TileBase dirtTile;
    public TileBase rockTile;

    public TileBase GetTileBase(int id)
    {
        TileBase tileBase = emptyTile;

        switch (id)
        {
            case IDEmpty:
            {
                tileBase = emptyTile;
                break;
            }
            case IDDirt:
            {
                tileBase = dirtTile;
                break;
            }
            case IDRock:
            {
                tileBase = rockTile;
                break;
            }
        }

        return tileBase;
    }
```

```
    public AbstractObjectController GetPrefab(int id)
    {
        var prefab = prefabPlayer;

        switch (id / 10 * 10)
        {
            case IDPlayer:
            {
                prefab = prefabPlayer;
                break;
            }
            case IDEnemyLeft:
            {
                prefab = prefabEnemyLeft;
                break;
            }
            case IDEnemyRight:
            {
                prefab = prefabEnemyRight;
                break;
            }
            case IDStone:
            {
                prefab = prefabStone;
                break;
            }
            case IDBomb:
            {
                prefab = prefabBomb;
                break;
            }
            case IDCake:
            {
                prefab = prefabCake;
                break;
            }
        }

        switch (id % 10)
        {
            case 0:
            {
                prefab.startDirection = "up";
                break;
            }
            case 1:
            {
                prefab.startDirection = "down";
                break;
            }
            case 2:
            {
                prefab.startDirection = "right";
                break;
            }
            case 3:
            {
                prefab.startDirection = "left";
                break;
            }
        }

        return prefab;
    }
}
```

Листинг Б.12 – Файл MapController.cs

```
using System;
using Game.ObjectsControllers;
using System.Collections.Generic;
using System.IO;
using UnityEngine;
using UnityEngine.Serialization;
using UnityEngine.Tilemaps;
using static DataScript;
using static FileUtils;

public class MapController : MonoBehaviour
{
    [SerializeField]
    private Tilemap _tilemap;
    private int[,] _mapArray;

    private ObjectInitializer Initializer;
    private DataScript Data;
    private GlobalGameParameters GGP;

    private void Start()
    {
        GGP = GlobalGameParameters.GetInstance();
        Initializer = FindObjectOfType<ObjectInitializer>();
        Data = FindObjectOfType<DataScript>();
    }

    #region MapControl

    public int GetObj(Vector3 position)
    {
        return _mapArray[(int) position.x, (int) position.y];
    }

    public int GetObj(int x, int y)
    {
        return _mapArray[x, y];
    }

    /*
    public string GetObjName(Vector3 position)
    {
        string objName;

        int id = mapArray[(int) position.x, (int) position.y];

        if (id == -1)
        {
            objName = "";
        }
        else
        {
            objName = Data.idNameDict[id];
        }

        return objName;
    }*/

    #region SetObject

    /*
    public void SetObj(string objName, Vector3 position)
    {
```

```
        CleanCell(position);
        int id = Data.nameIdDict[objName];
        mapArray[(int) position.x, (int) position.y] = id;
        if (id < 100)
        {
            tilemap.SetTile(tilemap.WorldToCell(position),
Data.GetTileBase(id));
        }
        else
        {
            Initializer.SpawnObject(Data.GetPrefab(id), position);
        }
    }

    public void SetObj(string objName, int x, int y)
    {
        CleanCell(x, y);
        int id = Data.nameIdDict[objName];
        mapArray[x, y] = id;
        if (id < 100)
        {
            tilemap.SetTile(new Vector3Int(x, y, 0), Data.GetTileBase(id));
        }
        else
        {
            Initializer.SpawnObject(Data.GetPrefab(id), new Vector3(x, y,
0));
        }
    }*/

    public void SetObj(int id, Vector3 position)
    {
        CleanCell(position);
        _mapArray[(int) position.x, (int) position.y] = id;
        if (id < IDPlayer)
        {
            _tilemap.SetTile(_tilemap.WorldToCell(position),
Data.GetTileBase(id));
        }
        else
        {
            Initializer.SpawnObject(Data.GetPrefab(id), position);
        }
    }

    public void SetObj(int id, int x, int y)
    {
        CleanCell(x, y);
        _mapArray[x, y] = id;
        if (id < 100)
        {
            _tilemap.SetTile(new Vector3Int(x, y, 0), Data.GetTileBase(id));
        }
        else
        {
            Initializer.SpawnObject(Data.GetPrefab(id), new Vector3(x, y,
0));
        }
    }

    #endregion

    public void MoveObjToPosition(AbstractObjectController obj, Vector3
newPosition)
    {
        var oldPosition = obj.transform.position;
```

```
        int x = (int) oldPosition.x;
        int y = (int) oldPosition.y;
        int newX = (int) newPosition.x;
        int newY = (int) newPosition.y;
        var list = Initializer.listOfObjects;
        if (list.ContainsKey(newPosition))
            Initializer.KillObject(newPosition);
        obj.transform.position = newPosition;
        list.Remove(oldPosition);
        _mapArray[newX, newY] = obj.ObjId;
        list.Add(newPosition, obj);
        CleanTile(oldPosition);
    }

    public void MoveObjToPosition(AbstractObjectController obj, int newX, int
newY)
    {
        Vector3 newPosition = new Vector3(newX, newY, 0);
        var oldPosition = obj.transform.position;
        int x = (int) oldPosition.x;
        int y = (int) oldPosition.y;
        var list = Initializer.listOfObjects;
        if (list.ContainsKey(newPosition))
            Initializer.KillObject(newPosition);
        obj.transform.position = newPosition;
        list.Remove(oldPosition);
        _mapArray[newX, newY] = obj.ObjId;
        list.Add(newPosition, obj);
        CleanTile(oldPosition);
    }

    //Probably must be reworked to add pressure plate
    public void CleanTile(Vector3 position)
    {
        int x = (int) position.x;
        int y = (int) position.y;
        _tilemap.SetTile(_tilemap.WorldToCell(position), Data.emptyTile);
        _mapArray[x, y] = IDEmpty;
    }

    public void CleanTile(int x, int y)
    {
        Vector3 position = new Vector3(x, y, 0);
        _tilemap.SetTile(_tilemap.WorldToCell(position), Data.emptyTile);
        _mapArray[x, y] = IDEmpty;
    }

    public void CleanCell(Vector3 position)
    {
        int x = (int) position.x;
        int y = (int) position.y;
        if (Initializer.listOfObjects.ContainsKey(position))
        {
            Initializer.KillObject(Initializer.listOfObjects[position]);
        }

        _tilemap.SetTile(_tilemap.WorldToCell(position), Data.emptyTile);
        _mapArray[x, y] = IDEmpty;
    }

    public void CleanCell(int x, int y)
    {
        Vector3 position = new Vector3(x, y, 0);
        if (Initializer.listOfObjects.ContainsKey(position))
        {
            Initializer.KillObject(Initializer.listOfObjects[position]);
```

```
        }

        _tilemap.SetTile(_tilemap.WorldToCell(position), Data.emptyTile);
        _mapArray[x, y] = IDEmpty;
    }

    #endregion

    #region MapLoad

    private void ClearTileMap()
    {
        _tilemap.ClearAllTiles();
        Initializer.ClearObjList();
    }

    public void ExpandMapArray(int nMax, int mMax)
    {
        int[,] mapArrayBase = (int[,]) _mapArray.Clone();
        int nBase = mapArrayBase.GetLength(0);
        int mBase = mapArrayBase.GetLength(1);
        _mapArray = new int[nMax, mMax];

        int iBase = 0;
        int jBase = 0;
        bool incI = false;

        for (int i = 0; i < nMax; i++)
        {
            for (int j = 0; j < mMax; j++)
            {
                if ((i >= nMax / 2 - nBase / 2 && i < nMax / 2 + nBase / 2)
&&
                    (j >= mMax / 2 - mBase / 2 && j < mMax / 2 + mBase / 2))
                {
                    _mapArray[i, j] = mapArrayBase[iBase, jBase];

                    incI = true;
                    jBase++;
                }
                else
                {
                    _mapArray[i, j] = -1;
                }
            }

            if (incI)
            {
                incI = false;
                iBase++;
                jBase = 0;
            }
        }

        LoadMap(_mapArray);
    }

    public void LoadMap(int[,] arr)
    {
        ClearTileMap();
        int n = arr.GetLength(0);
        int m = arr.GetLength(1);

        for (int i = 0; i < n; i++)
        {
            for (int j = 0; j < m; j++)
```

```
            {
                int id = arr[i, j];
                if (id != -1)
                {
                    SetObj(id, i, j);
                }
            }
        }
    }

    public void LoadLevel(string path)
    {
        string mapName;

        int n;
        int m;

        StreamReader file = new StreamReader(Application.dataPath + path);
        file = GetMetaInf(file, out n, out m, out mapName);

        _mapArray = FileToArray(file, n, m);

        file.Close();
        file.Dispose();

        LoadMap(_mapArray);
    }

    #endregion
}
```

Листинг Б.13 – Файл GameEventController.cs

```
using System.Collections;
using Game.UI;
using UnityEngine;
using UnityEngine.SceneManagement;
using UnityEngine.UI;

public class GameEventController : MonoBehaviour
{
    private GlobalGameParameters GGP;

    public Canvas canvas;

    public Follow mainCamera;

    public bool isGameStopped;

    public Text bombCount;
    public Text enemyCount;
    public Text diamondCount;

    public Image diedImage;

    public Button deadButton;
    public Text deadButtonText;

    public GameObject youDiedScene;

    public MapController mapController;

    private void Start()
    {
        GGP = GlobalGameParameters.GetInstance();
```

```
        canvas = FindObjectOfType<Canvas>();

        foreach (var text in canvas.GetComponentsInChildren<Text>())
        {
            switch (text.name)
            {
                case "BombCount":
                    bombCount = text;
                    break;

                case "DiamondCount":
                    diamondCount = text;
                    break;

                case "EnemyCount":
                    enemyCount = text;
                    break;
            }
        }

        foreach (var image in canvas.GetComponentsInChildren<Image>())
        {
            switch (image.name)
            {
                case "YouDied":
                    diedImage = image;
                    break;
            }
        }

        foreach (var button in canvas.GetComponentsInChildren<Button>())
        {
            switch (button.name)
            {
                case "DeadButton":
                    deadButton = button;
                    deadButtonText =
deadButton.GetComponentInChildren<Text>();
                    break;
            }
        }

        mapController = FindObjectOfType<MapController>();

        diedImage.color = new Color(1f, 1f, 1f, 0f);
        diedImage.enabled = false;

        deadButton.image.color = new Color(1f, 1f, 1f, 0f);
        deadButtonText.enabled = false;
        deadButton.enabled = false;

        youDiedScene.SetActive(false);
        isGameStopped = false;

        if (SceneManager.GetActiveScene().name != "MapEditor")
        {
            mapController.LoadLevel(GGP.CurrentLevelPath);
        }
    }

    public void DiedScene()
    {
        if (SceneManager.GetActiveScene().name != "MapEditor")
        {
            youDiedScene.SetActive(true);
            mainCamera.follow = false;
```

```
            isGameStopped = true;
            diedImage.enabled = true;

            StartCoroutine(ShowYouDied());

            Statistics.AddAttempt(int.Parse(bombCount.text),
int.Parse(enemyCount.text),
                int.Parse(diamondCount.text));
        }
    }

    public void EnemyDied()
    {
        enemyCount.text = (int.Parse(enemyCount.text) - 1).ToString();
    }

    public void EnemySpawn()
    {
        enemyCount.text = (int.Parse(enemyCount.text) + 1).ToString();
    }

    public void SpendBomb()
    {
        bombCount.text = (int.Parse(bombCount.text) - 1).ToString();
    }

    public void EatDiamond()
    {
        diamondCount.text = (int.Parse(diamondCount.text) + 1).ToString();
    }

    IEnumerator ShowYouDied()
    {
        for (float i = 0f; i <= 1f; i += 0.01f)
        {
            diedImage.color = new Color(1f, 1f, 1f, i);
            yield return new WaitForSeconds(.02f);
        }

        deadButton.enabled = true;
        deadButtonText.enabled = true;
    }

    public void ReturnToMenu()
    {
        isGameStopped = true;
        Statistics.AddAttempt(int.Parse(bombCount.text),
int.Parse(enemyCount.text), int.Parse(diamondCount.text));
        SceneManager.LoadScene(0);
    }
}
```

Листинг Б.14 – Файл ObjectInitializer.cs

```
using System.Collections.Generic;
using Game;
using Game.ObjectsControllers;
using UnityEngine;
using UnityEngine.Serialization;
using UnityEngine.Tilemaps;

public class ObjectInitializer : MonoBehaviour
{
    public AbstractObjectController prefabStone;
    public AbstractObjectController prefabEnemyRight;
```

```
    public AbstractObjectController prefabEnemyLeft;
    public AbstractObjectController prefabPlayer;
    public AbstractObjectController prefabBomb;
    public AbstractObjectController prefabCake;

    private GameEventController Event;
    private DataScript Data;
    private MapController Map;

    /*public readonly string[] BlackListForPlayer = {"rock", "stone",
"enemyLeft", "enemyRight", "bomb"};
    public readonly string[] BlackListForStoneWhenFall = {"rock", "dirt",
"stone", "cake"};
    public readonly string[] BlackListForDiamondWhenFall = {"rock", "dirt",
"stone", "bomb", "cake"};

    public readonly string[] BlackListForStoneWhenMove =
        {"rock", "dirt", "stone", "enemyLeft", "enemyRight", "bomb", "cake"};

    public readonly string[] BlackListForEnemy = {"rock", "dirt", "stone",
"enemyLeft", "enemyRight", "bomb", "cake"};
    public readonly string[] BlackListForBomb = {"rock", "dirt", "stone",
"enemyLeft", "enemyRight", "bomb", "cake", "player"};
    public readonly string[] BlackListForExplosion = {"rock"};

    public readonly string[] EnemyCanDieBy = {"stone", "cake"};
    public readonly string[] PlayerCanDieBy = {"stone", "enemyLeft",
"enemyRight", "cake"};

    public readonly string[] PlayerCanMove = {"stone", "bomb"};

    public Tilemap map;

    public TileBase emptyTile;
    public TileBase dirtTile;
    public TileBase rockTile;*/

    public Dictionary<Vector3, AbstractObjectController> listOfObjects;


    private void Start()
    {
        Application.targetFrameRate = 144;

        Map = FindObjectOfType<MapController>();
        Event = FindObjectOfType<GameEventController>();
        Data = FindObjectOfType<DataScript>();

        listOfObjects = new Dictionary<Vector3, AbstractObjectController>();
    }

    public void SpawnObject(AbstractObjectController prefab, Vector3
position, string direction)
    {
        AbstractObjectController go = Instantiate(prefab, position,
Quaternion.identity);
        go.startDirection = direction;

        if (prefab.CompareTag("Enemy"))
        {
            Event.EnemySpawn();
        }

        listOfObjects.Add(go.transform.position, go);
    }
```

```
    public void SpawnObject(AbstractObjectController prefab, Vector3
position)
    {
        AbstractObjectController go = Instantiate(prefab, position,
Quaternion.identity);
        go.startDirection = "up";

        if (go.CompareTag("Enemy"))
        {
            Event.EnemySpawn();
        }

        listOfObjects.Add(go.transform.position, go);
    }

    // public void CreateObject(GameObject prefab, Vector3 position)
    // {
    //     GameObject go = Instantiate(prefab, position,
Quaternion.identity);
    //
    //     listOfObjects.Add(go.transform.position, go);
    // }

    public void ClearObjList()
    {
        foreach (var el in listOfObjects)
        {
            Destroy(el.Value.gameObject);
        }

        listOfObjects.Clear();
    }

    public void DeleteObject(AbstractObjectController go)
    {
        Event.isGameStopped = true;

        Destroy(go.gameObject);
        listOfObjects.Remove(go.transform.position);
        if (go.CompareTag("Enemy"))
        {
            Event.EnemyDied();
        }

        Event.isGameStopped = false;
    }

    public void DeleteObject(Vector3 position)
    {
        Event.isGameStopped = true;

        var go = listOfObjects[position].gameObject;
        Destroy(go.gameObject);
        listOfObjects.Remove(position);
        if (go.CompareTag("Enemy"))
        {
            Event.EnemyDied();
        }

        Event.isGameStopped = false;
    }

    public void KillObject(AbstractObjectController go)
    {
        if (go.CompareTag("Enemy"))
        {
```

```
            Event.EnemyDied();
        }
        else if (go.CompareTag("Player"))
        {
            Event.DiedScene();
        }

        listOfObjects.Remove(go.transform.position);
        Destroy(go.gameObject);
    }

    public void KillObject(Vector3 position)
    {
        var go = listOfObjects[position];

        if (go.CompareTag("Enemy"))
        {
            Event.EnemyDied();
        }
        else if (go.CompareTag("Player"))
        {
            Event.DiedScene();
        }

        listOfObjects.Remove(position);
        Destroy(go.gameObject);
    }
}
```

Листинг Б.15 – Файл UserAttempt.cs

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

public struct UserAttempt
{
    public readonly int BombsLeft;
    public readonly int EnemiesLeft;
    public readonly int CakeCount;
    public readonly int Score;

    public UserAttempt(int bombsLeft, int enemiesLeft, int cakeCount)
    {
        BombsLeft = bombsLeft;
        EnemiesLeft = enemiesLeft;
        CakeCount = cakeCount;

        Score = bombsLeft - enemiesLeft + cakeCount;
    }
}

public static class Statistics
{
    public static List<UserAttempt> Attempts = new List<UserAttempt>();

    public static UserAttempt LastAttempt;
    public static UserAttempt BestAttempt;

    public static void AddAttempt(int bombsLeft, int enemiesLeft, int
cakeCount)
    {
        LastAttempt = new UserAttempt(bombsLeft, enemiesLeft, cakeCount);

        if (Attempts.Count == 0 || BestAttempt.Score < LastAttempt.Score)
```

```
        {
            BestAttempt = LastAttempt;
        }

        Attempts.Add(LastAttempt);
    }
}
```

Листинг Б.16 – Файл MenuController.cs

```
using System;
using System.Collections;
using System.Collections.Generic;
using System.ComponentModel.Design.Serialization;
using System.Security.Cryptography;
using UnityEngine;
using UnityEngine.SceneManagement;
using UnityEngine.UI;

struct AttemptUI
{
    public static AttemptUI operator +(AttemptUI attemptUi, UserAttempt
attempt)
    {
        attemptUi.BombsNumber.text = attempt.BombsLeft.ToString();
        attemptUi.DiamondsNumber.text = attempt.CakeCount.ToString();
        attemptUi.EnemiesNumber.text = attempt.EnemiesLeft.ToString();
        attemptUi.ScoreNumber.text = attempt.Score.ToString();
        return attemptUi;
    }

    public Component Instance;

    public Component Score;
    public Component Bombs;
    public Component Enemies;
    public Component Diamonds;

    public Text ScoreNumber;
    public Text BombsNumber;
    public Text EnemiesNumber;
    public Text DiamondsNumber;

    public AttemptUI(Component instance)
    {

        Instance = instance;

        Score = null;
        Bombs = null;
        Enemies = null;
        Diamonds = null;

        ScoreNumber = null;
        BombsNumber = null;
        EnemiesNumber = null;
        DiamondsNumber = null;

        foreach (var component in
Instance.GetComponentsInChildren<Component>())
        {
            switch (component.name)
            {
                case "Score":
                    Score = component;
```

```
                    foreach (var text in
Score.GetComponentsInChildren<Text>())
                    {
                        if (text.name == "Number")
                            ScoreNumber = text;
                    }

                    break;

                case "Bombs":
                    Bombs = component;
                    foreach (var text in
Bombs.GetComponentsInChildren<Text>())
                    {
                        if (text.name == "Number")
                            BombsNumber = text;
                    }

                    break;

                case "Enemies":
                    Enemies = component;
                    foreach (var text in
Enemies.GetComponentsInChildren<Text>())
                    {
                        if (text.name == "Number")
                            EnemiesNumber = text;
                    }

                    break;

                case "Diamonds":
                    Diamonds = component;
                    foreach (var text in
Diamonds.GetComponentsInChildren<Text>())
                    {
                        if (text.name == "Number")
                            DiamondsNumber = text;
                    }

                    break;
            }
        }
    }
}

public class MenuController : MonoBehaviour
{
    private Canvas[] _canvases;

    private Canvas _mainMenu;
    private Canvas _mapSelector;

    private AttemptUI _lastAttempt;
    private AttemptUI _bestAttempt;

    private GlobalGameParameters GGP;

    public GameObject WelcomeMessage;

    public ButtonData LvlButtonPrefab;

    public List<ButtonData> LvlButtons;
    public GameObject LevelPanel;
```

```
    private void Start()
    {
        _canvases = Resources.FindObjectsOfTypeAll<Canvas>();
        LvlButtons = new List<ButtonData>();
        GGP = GlobalGameParameters.GetInstance();

        foreach (var canvas in _canvases)
        {
            switch (canvas.name)
            {
                case "MainMenu":
                    _mainMenu = canvas;
                    break;
                case "MapSelector":
                    _mapSelector = canvas;
                    break;
            }
        }

        foreach (var component in
_mainMenu.GetComponentsInChildren<Component>())
        {
            switch (component.name)
            {
                case "LastAttempt":
                    _lastAttempt = new AttemptUI(component);
                    break;

                case "BestAttempt":
                    _bestAttempt = new AttemptUI(component);
                    break;
            }
        }

        if (Statistics.Attempts.Count != 0)
        {
            _lastAttempt += Statistics.LastAttempt;
            _bestAttempt += Statistics.BestAttempt;
        }

        for (int i = 0; i < 8; ++i)
        {
            ButtonData levelButton = Instantiate(LvlButtonPrefab,
LevelPanel.gameObject.transform);
            levelButton.GetComponentInChildren<Text>().text = (i +
1).ToString();
            levelButton.LevelPath = "/maps/official/lvl" + (i +
1).ToString();
            LvlButtons.Add(levelButton);
        }

        _mainMenu.enabled = true;
        _mapSelector.enabled = false;
    }

    public void OpenMapSelector()
    {
        _mainMenu.enabled = false;
        _mapSelector.enabled = true;
    }

    public void CloseMapSelector()
    {
        _mainMenu.enabled = true;
        _mapSelector.enabled = false;
    }
```

```
    public void LevelButtonClicked(ButtonData button)
    {
        GGP.CurrentLevelPath = button.LevelPath;
    }

    public void LoadScene(int sceneNumber)
    {
        SceneManager.LoadScene(sceneNumber);
    }

    public void HideWelcomeMessage()
    {
        // Destroy(WelcomeMessage);
        WelcomeMessage.SetActive(false);
    }

    public void CloseGame()
    {
        Application.Quit();
    }

    // public void LoadMap(int mapNumber)
    // {
    // }
    //
    // public void LoadMapSelector()
    // {
    //     _mainMenu.enabled = false;
    //     _mapSelector.enabled = true;
    // }
}
```

Приложение В

РЕЗУЛЬТАТЫ ТЕСТИРОВАНИЯ ИГРОВОГО ПРИЛОЖЕНИЯ

(рекомендуемое)

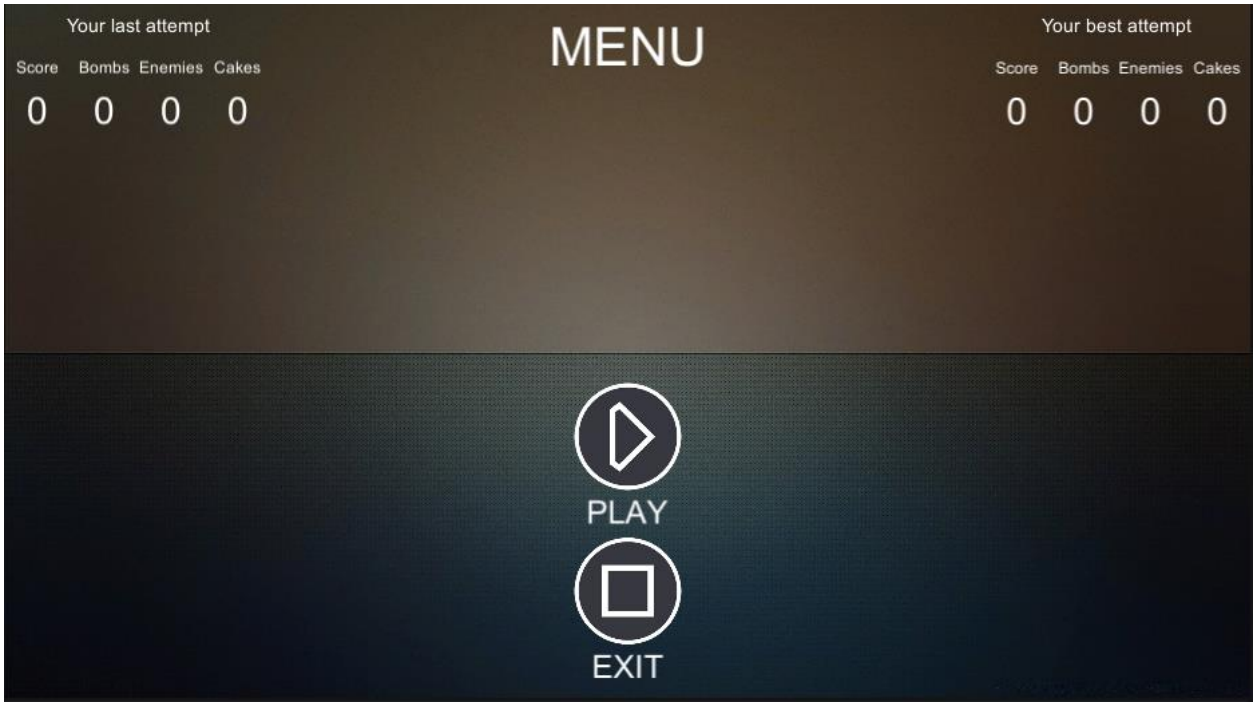

Рисунок В.1 – Запуск приложения

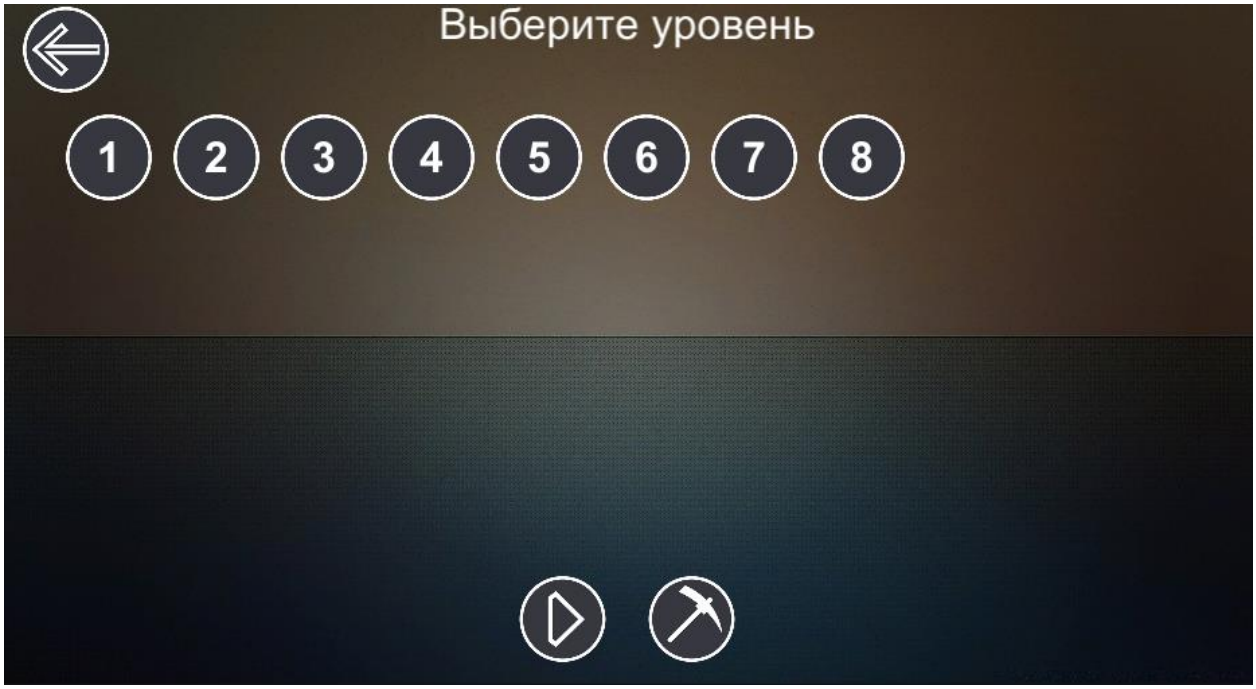

Рисунок В.2 – Нажатие кнопки «Играть»

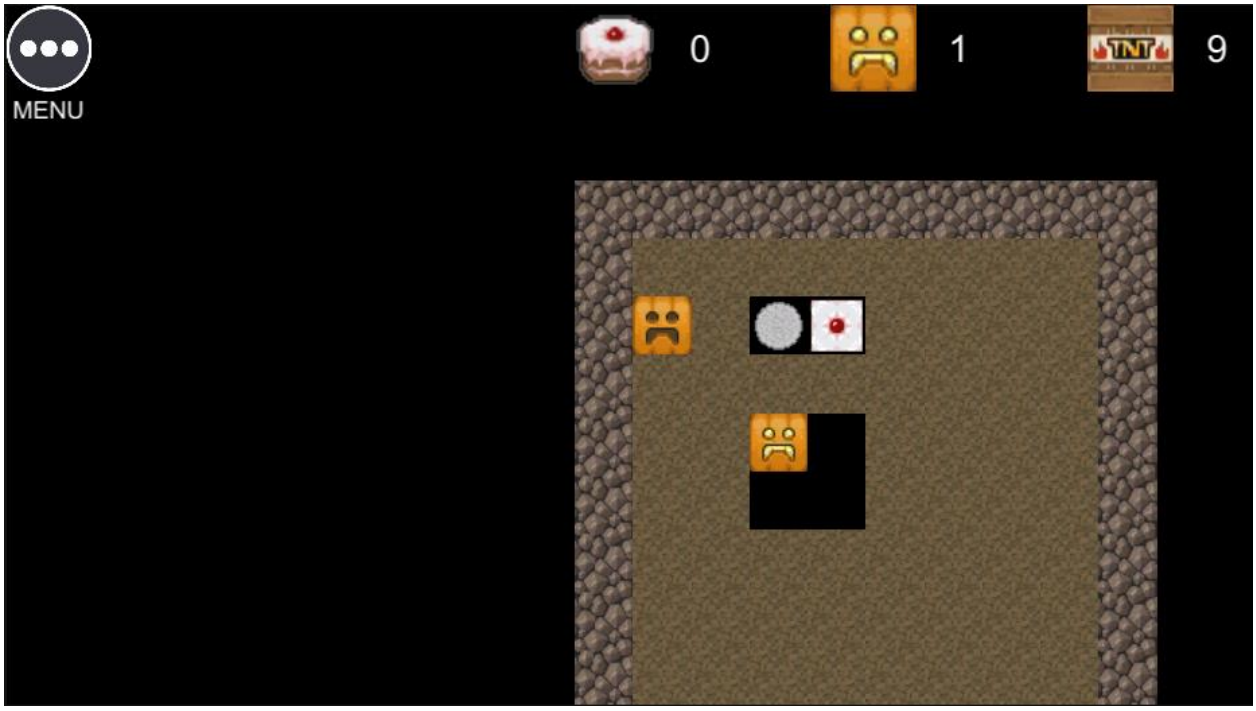

Рисунок В.3 – Нажатие кнопки «Начать уровень»

Рисунок В.4 – Нажатие кнопки редактировать уровень

Рисунок В.5 – Пойти направо

Рисунок В.6 – Пойти наверх

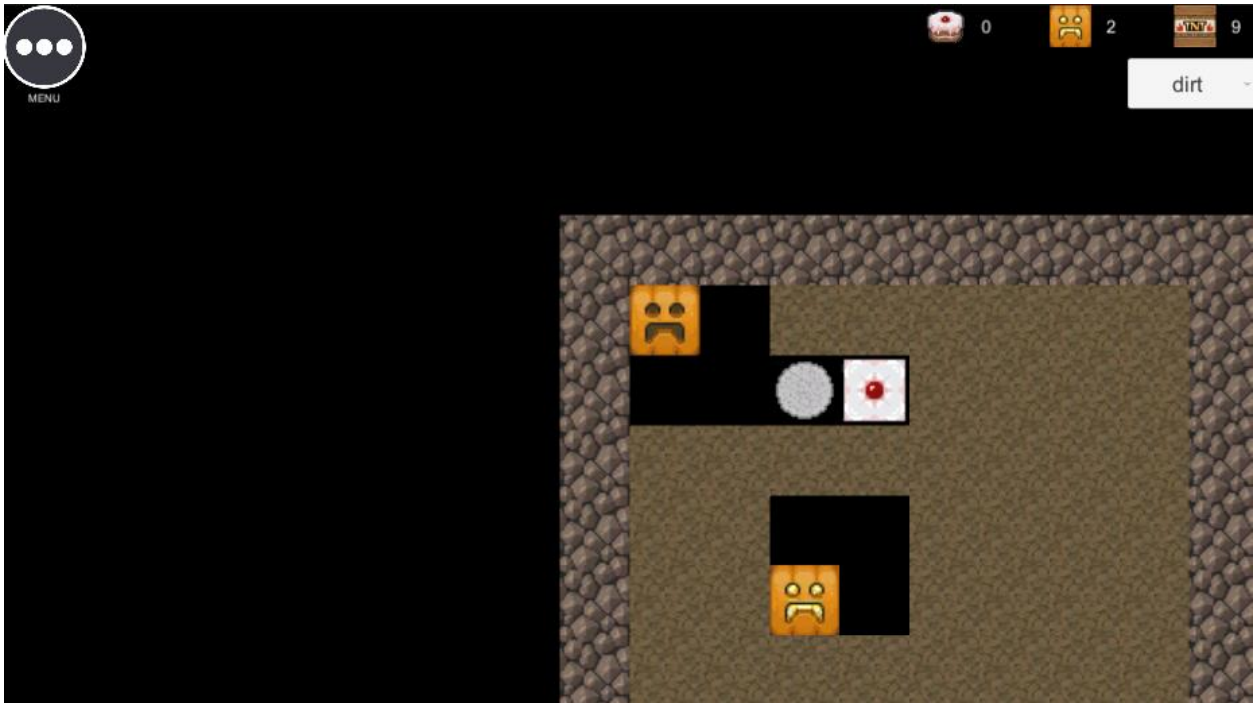

Рисунок В.7 – Пойти налево

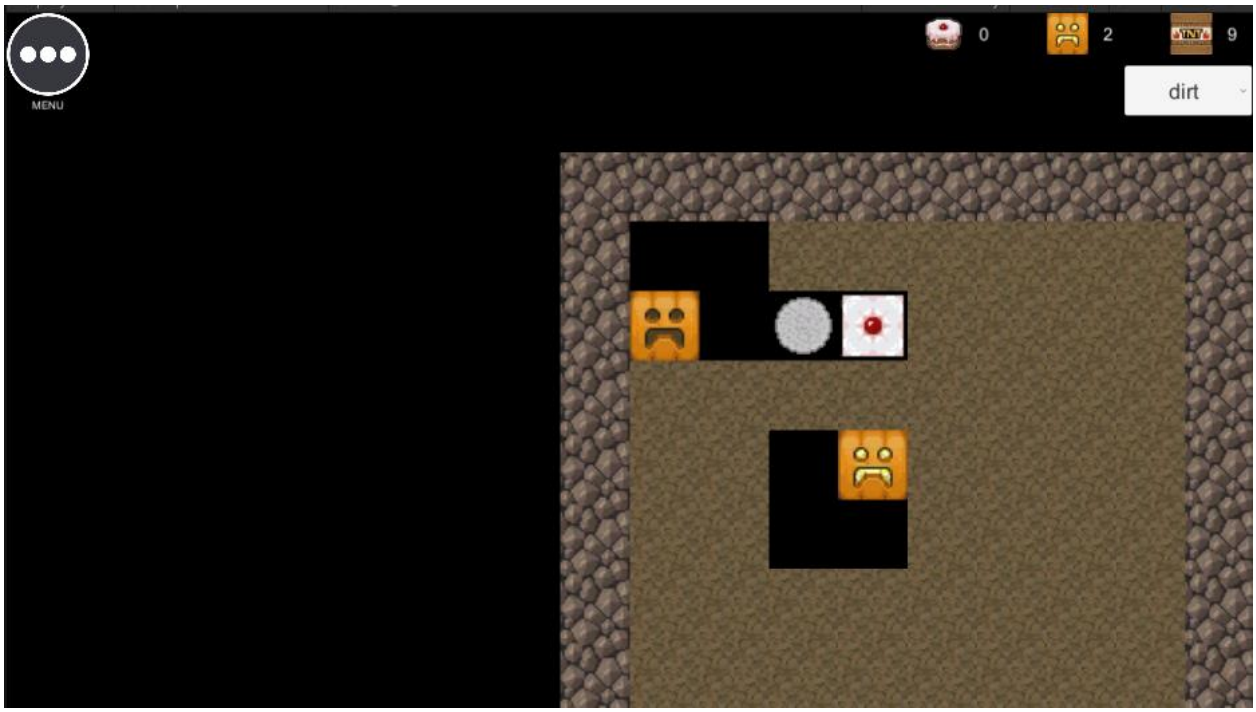

Рисунок В.8 – Пойти вниз

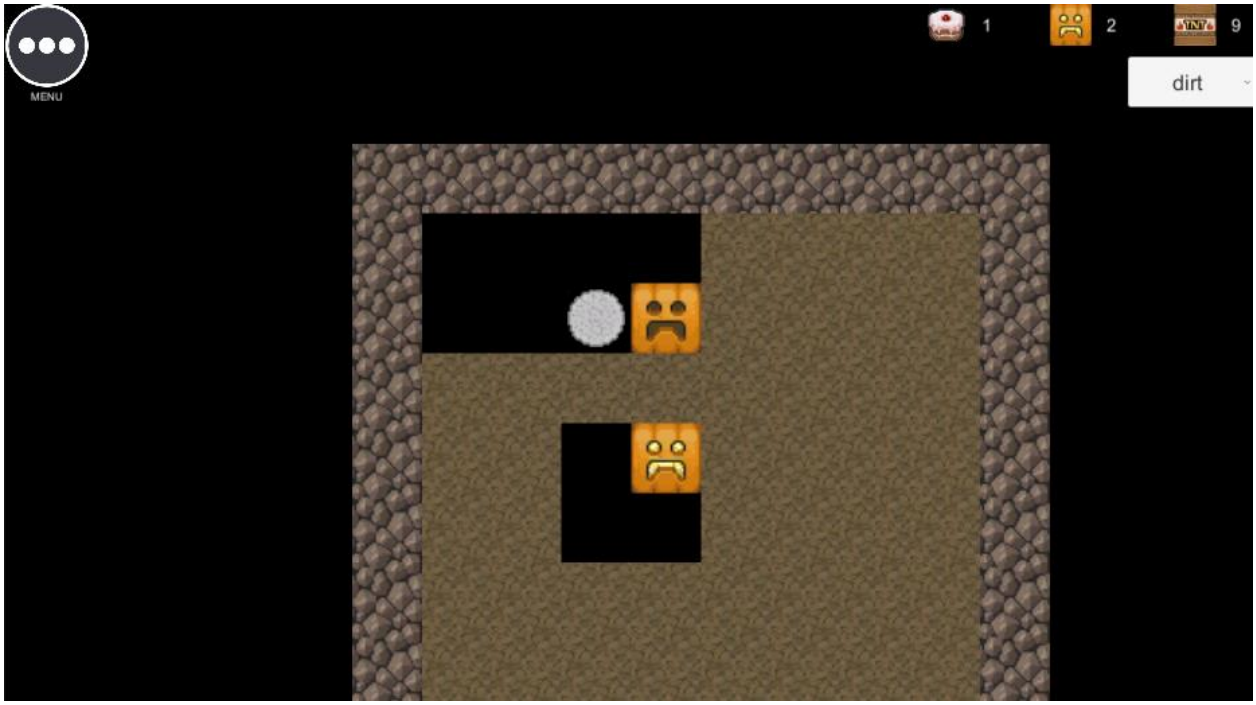

Рисунок В.9 – Съесть торт

Рисунок В.10 – Толкнуть камень

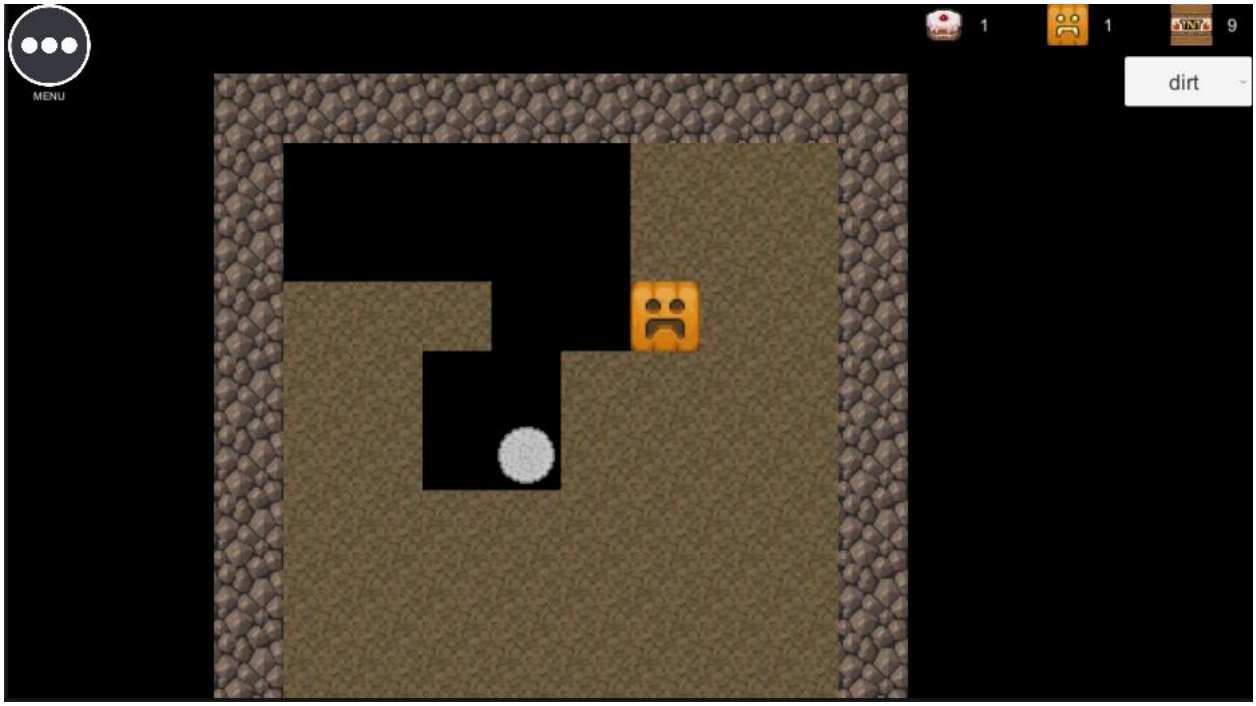

Рисунок В.11 – Сбросить камень на противника

Рисунок В.12 – Редактирование карты, пустая клетка

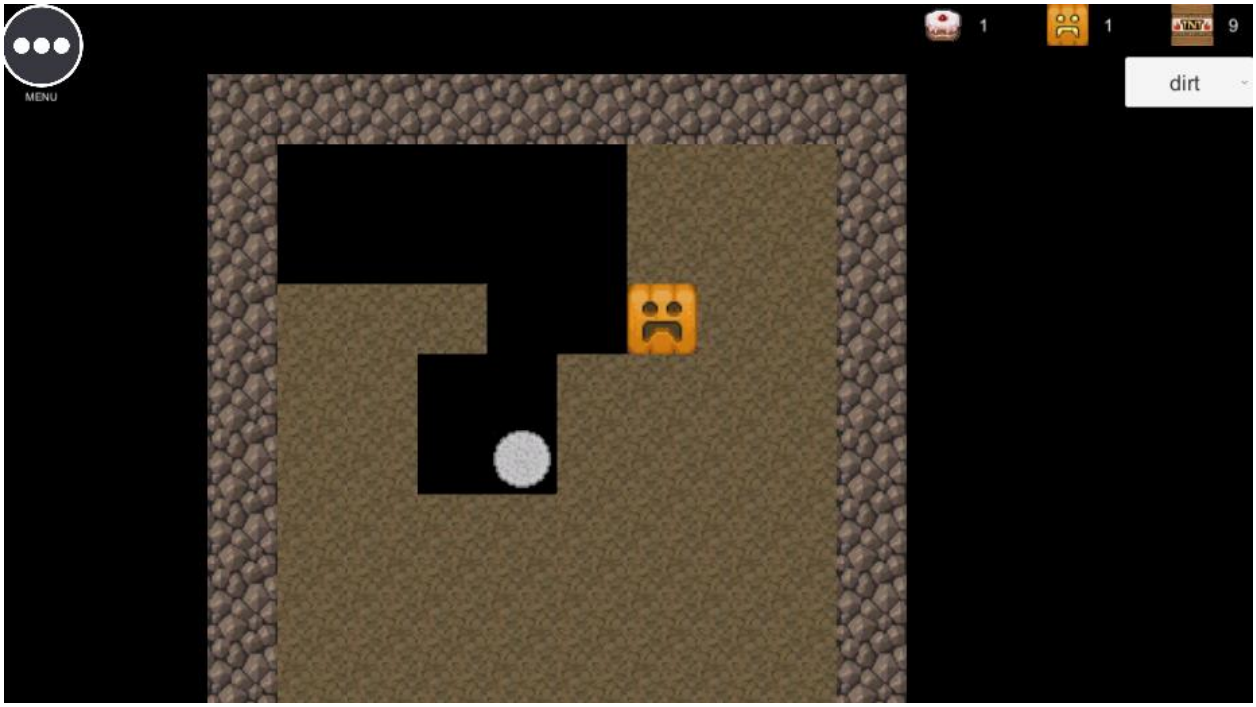

Рисунок В.13 – Редактирование карты, клетка земли

Рисунок В.14 – Редактирование карты, клетка каменной стены

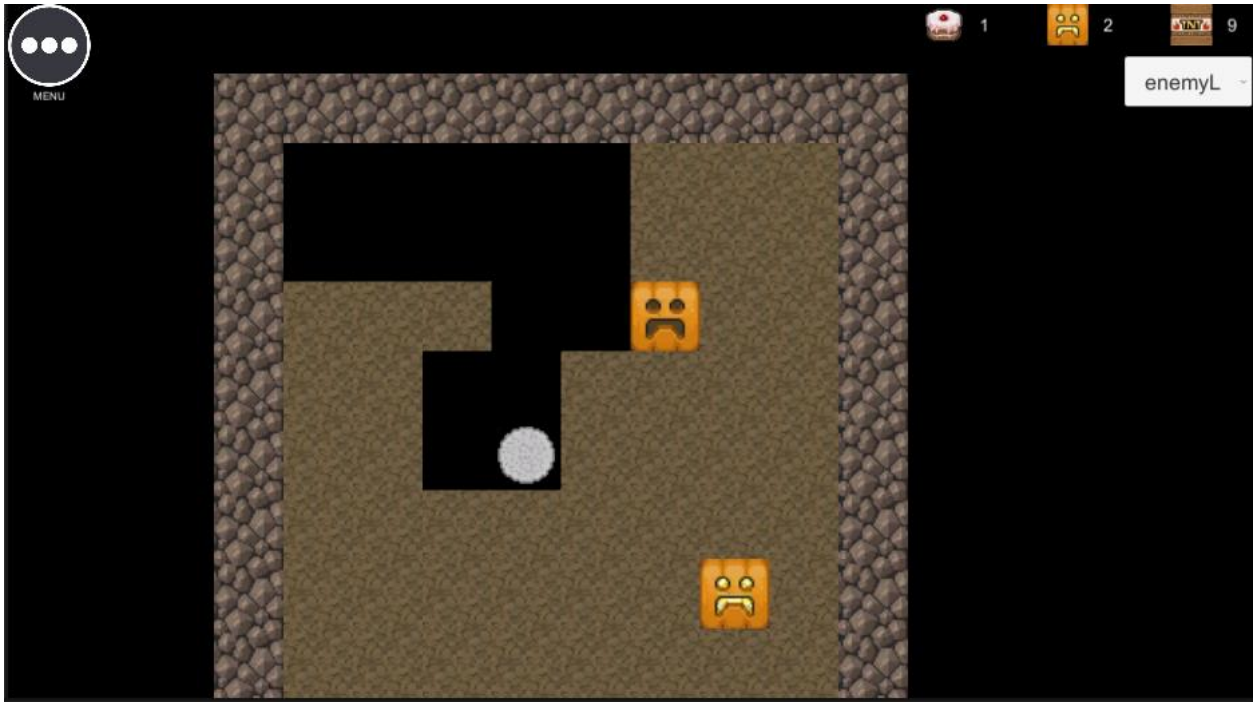

Рисунок В.15 – Редактирование карты, клетка противника

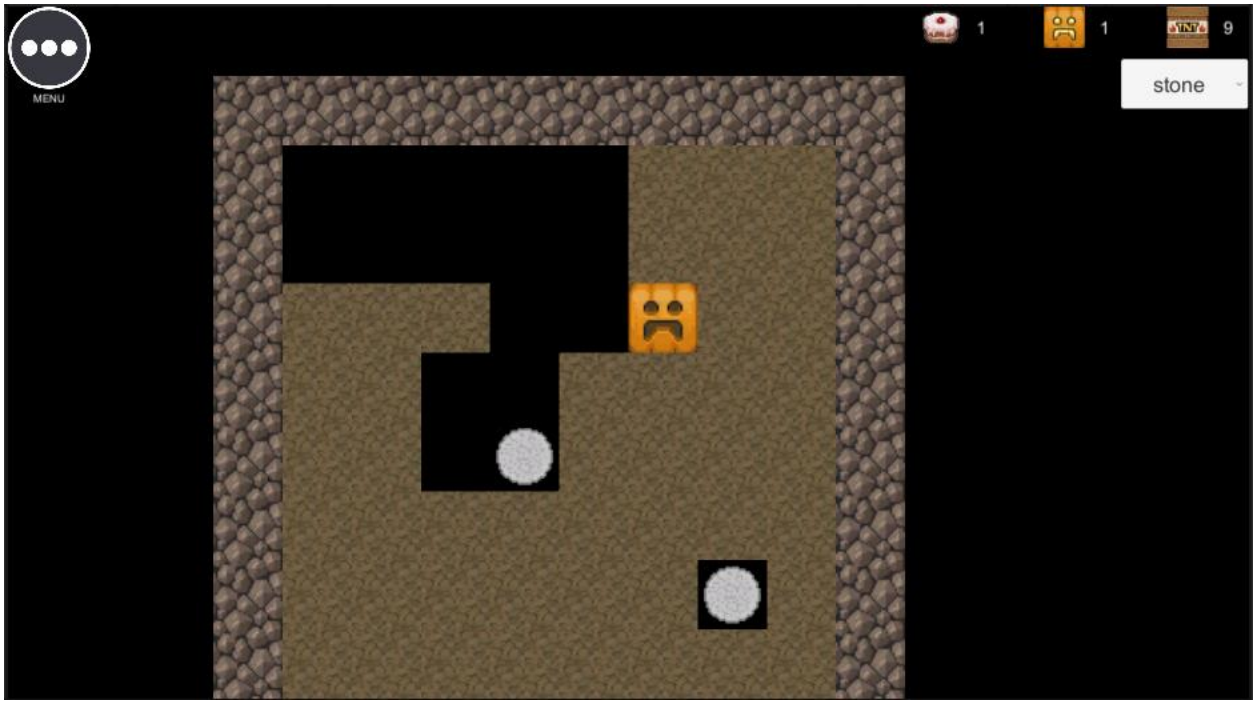

Рисунок В.16 – Редактирование карты, клетка камня

Рисунок В.17 – Редактирование карты, клетка бомбы

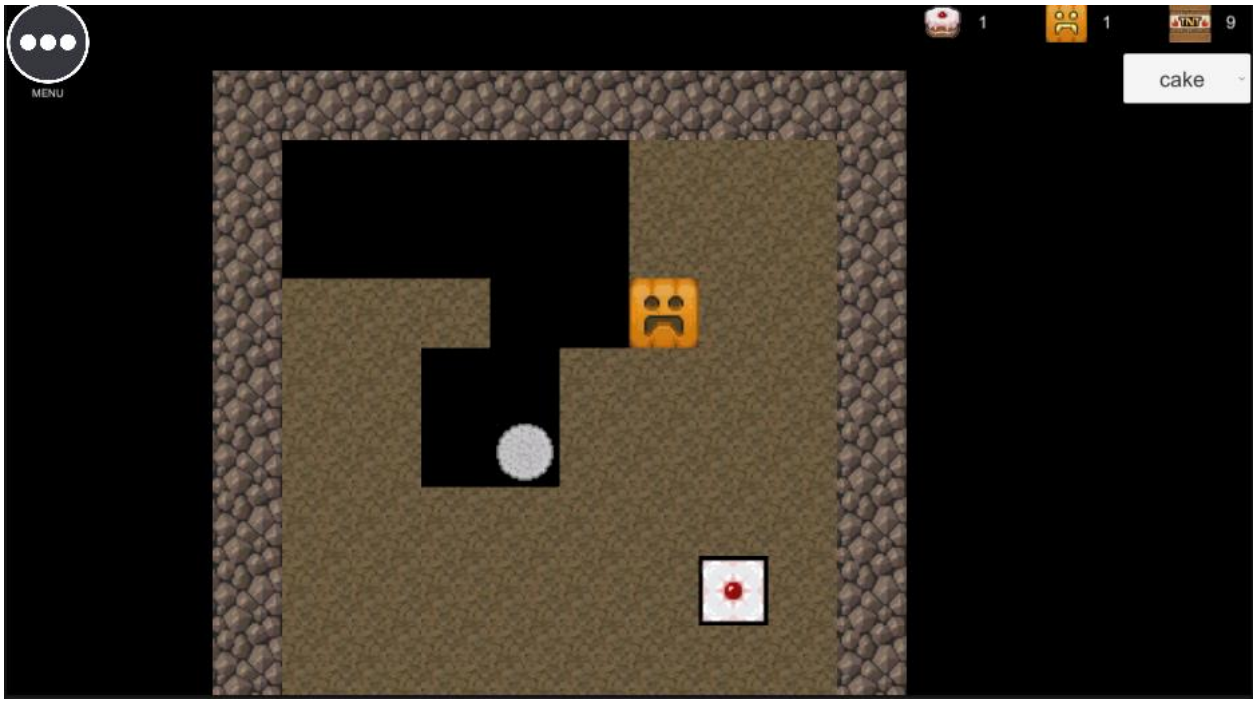

Рисунок В.18 – Редактирование карты, клетка торта

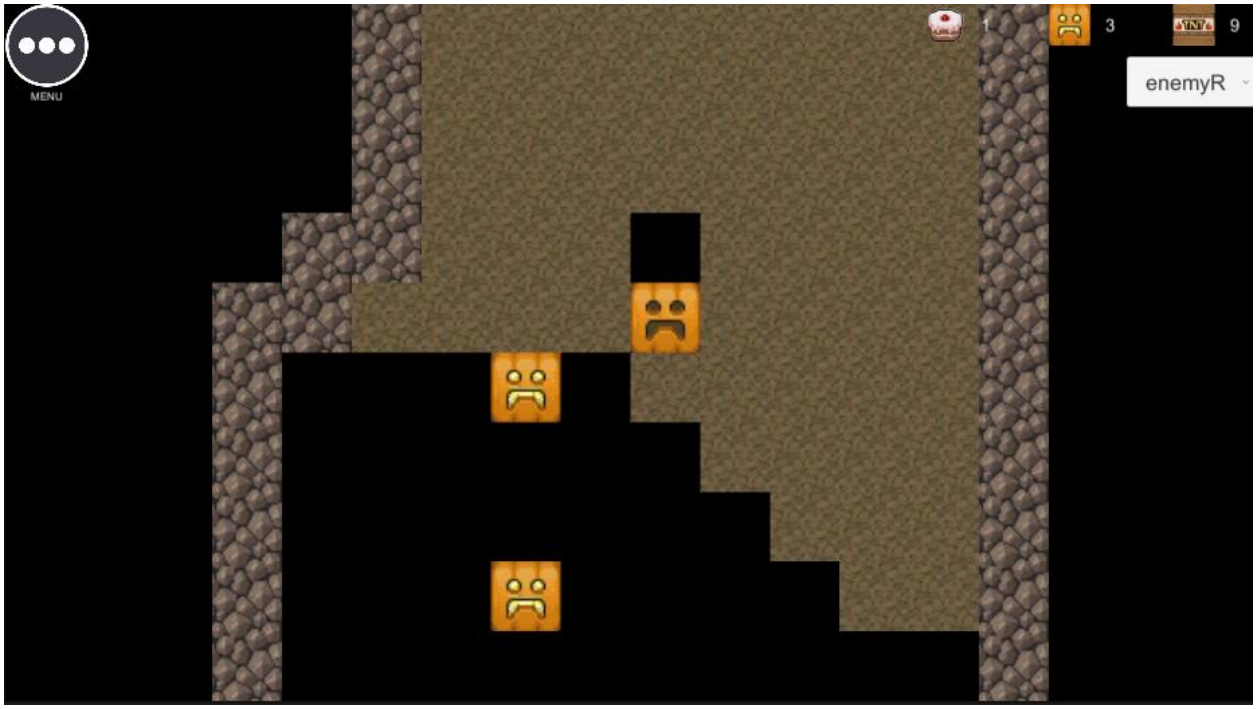

Рисунок В.19 – Редактирование карты, новый пример карты

Рисунок В.20 – Сброс бомбы на противника

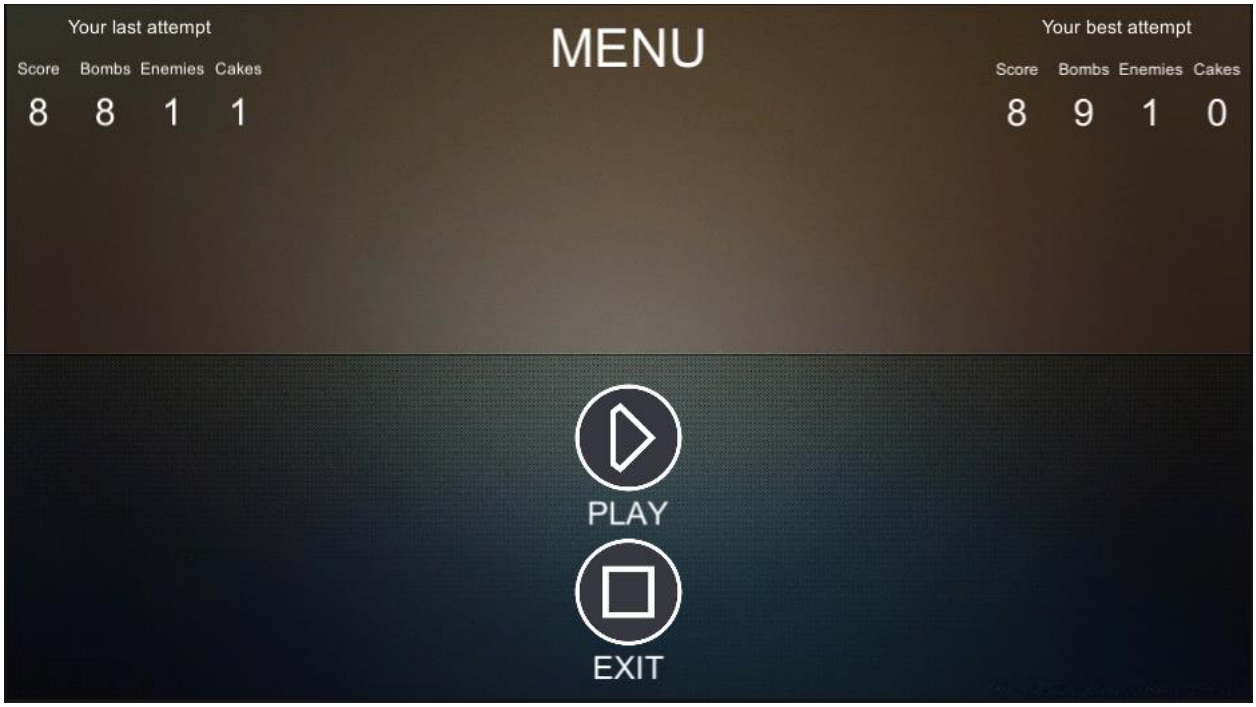

Рисунок В.21 – Возврат в главное меню и проверка результатов