

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет

вычислительной техники

Кафедра

«Вычислительная техника»

Направление подготовки: 09.03.01 «Информатика и вычислительная техника»
Профиль подготовки: «Вычислительные машины, комплексы, системы и сети»

БАКАЛАВРСКАЯ РАБОТА

на тему

**МОБИЛЬНОЕ ПРИЛОЖЕНИЕ ПОД
ОС АНДРОИД**

Студент  31.05.2022 Кривоzubов Антон Константинович
(подпись, дата)

Руководитель  31.05.2022 Юрова Ольга Викторовна
(подпись, дата)

Нормоконтролёр  31.05.2022 Бычков Андрей Станиславович
(подпись, дата)

Работа допущена к защите (протокол зас. кафедры от 02.06.2022 г. № 13)

Заведующий кафедрой  М. А. Митрохин
(подпись)

Работа защищена с отметкой отлично (протокол заседания ГЭК от
15.06.2022г. № 10)

Секретарь ГЭК  Е.И. Гурин
(подпись)

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

Факультет

вычислительной техники

Кафедра

«Вычислительная техника»

«Утверждаю»

Заведующий кафедрой

 М.А. Митрохин

«16» апреля 2022 г.

**ЗАДАНИЕ
НА БАКАЛАВРСКУЮ РАБОТУ**

1. Студент Кривозубов Антон Константинович гр. 18ВВ1 факультета вычислительной техники.

2. Тема работы «Мобильное приложение под ОС Андроид»

Тема утверждена приказом ПГУ №357/о от "26" апреля 2022 г.

3. Руководитель бакалаврской работы к.т.н., доцент кафедры ВТ ПГУ Юрова О. В.

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

Назначение разработки – мобильное приложение под ОС Андроид. Приложение должно быть спроектировано для мобильных устройств, поддерживающих актуальные версии ОС Андроид, поддерживать любые разрешения экрана, содержать интуитивно понятный интерфейс, иметь систему взаимодействия с игровыми объектами.

5. Перечень подлежащих разработке вопросов

Требуется произвести анализ предметной области, проанализировать индустрию игрового рынка, узнать причины, определяющие ее развитие, изучить виды и категории игровых приложений, изучить актуальные способы разработки игровых приложений, определить и разработать архитектуру приложения, выбрать технологии, применяемые при проектировании, выбрать языки программирования и среду разработки, разработать мобильное приложение, создать пользовательский и графический интерфейсы.

6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1.	Анализ предметной области	15 %	26.04.22 - 29.04.22	<i>фю</i>
2.	Выбор архитектуры приложения и технологий, применяемых при проектировании и разработке	10 %	30.04.22 - 04.05.22	<i>фю</i>
3.	Разработка графического интерфейса	20 %	05.05.22 - 09.05.22	<i>фю</i>
4.	Разработка функционала приложения	25 %	10.05.22 - 21.05.22	<i>фю</i>
5.	Тестирование и отладка	15 %	22.05.22 - 24.05.22	<i>фю</i>
6.	Выполнение задания по БЖД	5 %	25.05.22 - 27.05.22	<i>фю</i>
7.	Оформление пояснительной записки	10 %	28.05.22 - 31.05.22	<i>фю</i>

Дата выдачи задания "26" апреля 2022 г.

Руководитель бакалаврской работы 26.04.2022 *фю* Юрова О.В.
(подпись, дата)

Задание к исполнению принял студент 26.04.2022 *кр* Кривоzubов А. К.
(подпись, дата)

Работу к защите допустить
Декан факультета 14.06.2022 *фю* Фионова Л.Р.
(подпись, дата)

Реферат

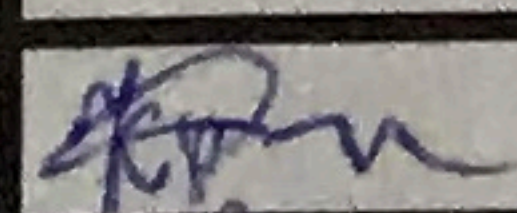
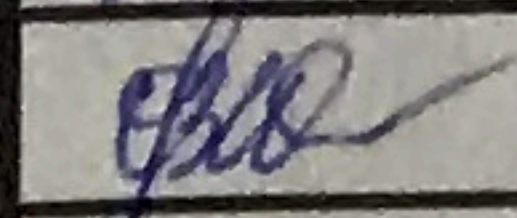
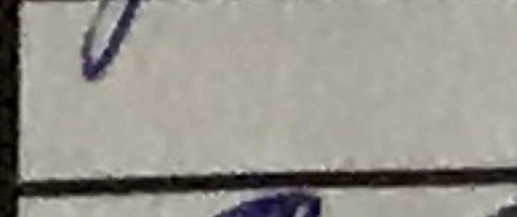
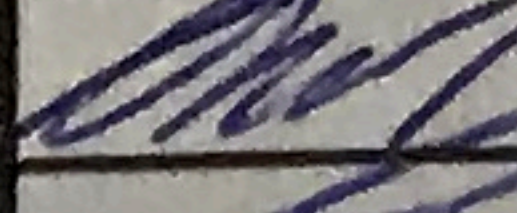
Бакалаврская работа содержит 103 листа, 47 рисунков, 18 таблиц, 10 источников, 7 приложений.

МОБИЛЬНОЕ ПРИЛОЖЕНИЕ, *TOWER DEFENSE*, *ANDROID*, *UNITY*, *VISUAL STUDIO*, *C#*, *GAME DESIGN*, *BLENDER*, *PHOTOSHOP*.

Цель работы – разработка игрового приложения для классической игры «*Tower Defense*». Данное приложение должно соответствовать классическим правилам этой игры, содержать интуитивно понятный интерфейс, выдавать информацию о своем текущем состоянии. Также система должна поддерживать добавление и удаление новых объектов, которые необходимы для визуализации сцен игры.

Результатом разработки является проект, удовлетворяющий поставленной задаче. А именно, разработано, реализовано и протестировано игровое мобильное приложение. Приложение должно быть совместимо с актуальными версиями *Android* смартфонов, а также поддерживать любые разрешения экрана.

Разработка производилась с использованием широкого стека технологий, к которым относятся: язык программирования *C#*, программное обеспечение для создания трёхмерной компьютерной графики *Blender*, интегрированная среда разработки *Visual Studio 2022* для работы со скриптами и межплатформенная среда *Unity* для разработки игрового приложения.

					ПГУ 1.090301.21.001 ПЗ					
Изм.	Лист	№ докум.	Подпись	Дата						
Разраб.		Кривошубов А. К.			Мобильное приложение под ОС Андроид Пояснительная записка			Лит.	Лист	Листов
Провер.		Юрова О. В.							2	113
Н. Контр.		Бычков А. С.						ФВТ, каф. ВТ гр. 18ВВ1		
Утверд.		Митрохин М.А.								

Содержание

Введение	5
1 Мобильное приложение под ОС Андроид	7
1.1 Анализ предметной области	7
1.1.1 Этапы разработки мобильного приложения	8
1.1.2 Виды игровых приложений	9
1.1.3 Анализ аналогов	13
1.1.4 Анализ инструментов разработки для реализации проекта	16
1.1.5 Анализ языков программирования	21
1.2 Проектирование и разработка	23
1.2.1 Структура приложения	23
1.2.2 Реализация игрового поля	24
1.2.3 Реализация противников	26
1.2.4 Реализация башен	32
1.2.5 Реализация интерфейса пользователя	42
1.3 Анализ результатов	51
1.3.1 Тестирование приложения	52
1.3.2 Преимущества и недостатки	56
2 Безопасность жизнедеятельности	57
2.1 Вредоносные факторы	57
2.2 Классификация и категорирование помещения	58
2.3 Требования к помещению	59
2.4 Требования к освещению	61
2.5 Требования по организации рабочего места	63

2.6 Требования к организации работы.....	64
2.7 Расчет освещения рабочего места оператора ПЭВМ.....	65
Заключение	70
Список использованных источников	71
Приложение А – <i>UML</i> -диаграмма классов.....	72
Приложение Б – <i>UML</i> -диаграмма прецедентов	74
Приложение В – <i>UML</i> -диаграмма развертывания.....	76
Приложение Г – <i>UML</i> -диаграмма последовательностей	78
Приложение Д – <i>UML</i> -диаграмма деятельности	80
Приложение Е – Листинг программы.....	82
Приложение Ж – Презентация	104

Введение

В XXI веке быстро развиваются различные отрасли компьютерных технологий. Компьютерная техника сегодня используется на различных предприятиях, автоматизируются производственные процессы и человек становится наблюдателем. Без компьютерных программ не обойтись сотрудникам различных научных лабораторий, нашли они своё применение на школьных уроках, в быту для общения, совершения покупок, развлечений.

Первые компьютерные игры, появившиеся в 60-ые годы XX века, не все воспринимали всерьёз, многие специалисты подвергали их серьёзной критике. Причины критики просты: электронно-вычислительные машины требовали больших площадей для размещения, квалифицированных специалистов для обслуживания. Использование дорогостоящих ресурсов лишь ради развлечения казалось нецелесообразным. Развитие научно-технического прогресса привело к совершенствованию электронно-вычислительной техники. Электронно-вычислительные машины стали более компактными, уже не требовали больших помещений для размещения. Их использование не только для решения производственных или научных задач, но и для развлечения уже не вызывало прежней критики.

Сегодня индустрия компьютерных игр (разработка, издание и продвижение игр) – одна из самых важных и быстро развивающихся отраслей компьютерных технологий и одновременно глобального сектора развлечений. Игры становятся культурными феноменами и признаются произведениями искусства. Число геймеров растёт (2,5 млрд игроков в 2020 г.), а сам гейминг становится высокооплачиваемой профессией. Формируется мощная экосистема. Вокруг ядра, состоящего из разработчиков игр и издателей, развиваются профильные СМИ, специальные финансовые и рекламные инструменты; появляются специализированные физические площадки (интернет-кафе нового поколения, локации для киберспортивных

турниров и др.); разработке игр начинают изучать и обучать в высших учебных заведениях и др.

Что касается ключевых тенденций, определяющих развитие глобальной игровой индустрии, то здесь, в первую очередь, необходимо обозначить новые технологии: мобильные, облачные, виртуальные, графические, искусственный интеллект и др. Мировой рынок компьютерных игр становится глубже и масштабней. За последние пять лет, по оценкам аналитиков и экспертов, он рос в среднем на 11% в год и достиг отметки в 152 млрд долл., опередив ряд других конкурентных контентных рынков, в частности, речь идет о кино- и музыкальной индустриях, чей суммарный объем рынков составил всего 62 млрд долл. Среди прочих причин, спровоцировавших такие изменения, можно отметить бурное развитие Интернета во всем мире и дешевизну устройств с доступом в Интернет, в частности речь идет преимущественно о смартфонах, а также упрощение инструментов разработки на фоне увеличения дистрибутивных каналов [1].

Далеко не каждый пользователь имеет материальные возможности, позволяющие приобретать актуальные комплектующие, которые будут соответствовать требованиям современных игровых приложений. Рост цен на комплектующие сильно влияет на разработку и продажу игр.

Компьютерные игры меняются под влиянием технического прогресса: совершенствуются спецэффекты, качество графики всё более приближается к реальности, расширяются возможности игровых ходов в сценариях. Отдельные игры превращаются в своеобразные реальные миры. Программные продукты, предоставляющие возможности сетевой игры, пользуются особой популярностью среди пользователей. Опыт, игровые навыки, скорость реакции, бонусы, полученные или купленные в ходе игры, определяют успех игрока.

					<i>ПГУ 1.090301.21.001 ПЗ</i>	<i>Лист</i>
<i>Изм.</i>	<i>Лист</i>	<i>№ докум.</i>	<i>Подпись</i>	<i>Дата</i>		6

1 Мобильное приложение под ОС Андроид

1.1 Анализ предметной области

В современном мире создание видеоигр является одним из наиболее крупных сегментов индустрии развлечений. По степени влияния на потребителей и вовлеченности их в интерактивное окружение, предлагаемое видеоиграми, этот сегмент уже давно выделяется среди других видов развлечений.

С точки зрения используемой игровой платформы, самым крупным сегментом (51% рынка) стали мобильные игры – версии игр, адаптированных для смартфонов и планшетов. При этом на подсегмент игр для смартфонов приходится 80% всей выручки мобильного сегмента. Такой существенный вес мобильного направления легко объясняется доступностью самих мобильных устройств, стоимостью мобильного трафика, а также легкостью освоения мобильных игр. Согласно прогнозам аналитиков, в ближайшем будущем доля мобильного сегмента будет только увеличиваться [2].

На втором месте по весу находится сегмент игр для консолей – 26%. Игровые консоли *Odyssey, Atari, Fairchild Channel A, GameBoy, Nes(Dendy), Sega, PlayStation, Xbox, Nintendo*. Их основное назначение – запуск и воспроизведение видеоигр. Так как эти устройства не были предназначены для выполнения сложных компьютерных задач, они являются более доступны пользователю и по цене, и по простоте управления, нежели ЭВМ.

Десктопные решения суммарно занимают 23% рынка (34,9 млрд долл.), из которых 2% – это браузерные онлайн-игры, а остальные 21% – онлайн скачиваемые или коробочные версии компьютерных игр. Одной из причин роста выступают соревновательные игры, включая как коробочные, так и браузерные варианты.

Одна из тенденций современной жизни – популярность мобильных устройств. Среди причин популярности данных устройств на базе ОС

Android – доступность, мобильность, функциональные возможности. Магазины приложений «*Google play*» и «*Apple store*» предлагают пользователям разнообразные игры с яркой графикой и продуманным сюжетом. Немаловажен и тот факт, что многие являются бесплатными. Место, которое занимают игры в памяти мобильного устройства, невелико, процесс скачивания прост и доступен, так как производители оптимизируют характеристики устройств под минимальные требования для работы приложений. Все это смещает акцент игровой индустрии в пользу мобильных устройств.

1.1.1 Этапы разработки мобильного приложения

Разработка мобильных приложений — это создание программного обеспечения (ПО), которое будет работать на мобильном устройстве. Разработка мобильных игр предусматривает не только творческий процесс, но и предварительное планирование всех этапов с четко прописанным ТЗ [3].

Основные пункты разработки приложения представлены ниже.

Идея — ключевой момент, решающий, какая это будет игра, в каком направлении и жанре, для какой целевой аудитории создается. Необходимо продумать, что сделает игру уникальной и чем она будет отличаться от всех остальных, как и почему она станет популярной.

Выбор платформы — существует две основных: *Android* и *iOS*, оптимально - поддержка обоих ОС.

Создание концепта — это подготовительный этап, цель которого — отшлифовать идею будущей игры и собрать базовую геймдизайнерскую документацию. В них излагают краткое, но при этом емкое описание игры с ключевыми аспектами.

Прототипирование — на этом этапе команда разработчиков игры моделирует свой концепт: проверяет идеи и гипотезы, воплощая их в небольших простых прообразах. Например, создает 2-3 варианта визуального оформления уровней, чтобы в итоге выбрать самое удачное. То же касается механик и поведения персонажей. На прототип обычно тратят минимум

времени - тестируют только то, что нужно, и не обращают внимание на его «красоту».

Продакшн — самый ресурсозатратный этап создания мобильной игры. Главная задача - выпустить полноценную версию игры для внутреннего осмотра, закрытого бета-тестирования. На этом этапе создается игра. Разработчик готовит необходимый арт, анимирует, настраивает игровой баланс, программирует и много тестирует. Результат продакшна — первая версия игры.

Релиз — На этом этапе игру продвигают в *App Store* и *Google Play*, развивают комьюнити и, если необходимо, расширяют сервис техподдержки, запускают маркетинговые мероприятия, ведут работу над новым контентом.

Поддержка — после релиза начинается настоящая работа по интеграции обновлений, маркетингу и комьюнити [4].

1.1.2 Виды игровых приложений

Полной и завершённой классификации компьютерных игр на сегодняшний день не создано. Видеоигры очень разнообразны, поэтому в различных источниках можно встретить разные основания для классификации игр. Одна из таких классификаций представлена на рисунке 1, она включает в себя: количество игроков, визуальное представление, жанр, платформа, способ взаимодействия [5].



Рисунок 1 – Классификация игр

По количеству игроков игры делятся на одиночные и многопользовательские. По способу визуального представления игра может быть графической или текстовой. По используемой платформе игры могут быть кроссплатформенными или нативными, т.е. приложение создавалось для конкретной платформы. Самым крупным делением является классификация по жанру, которая позволяет ориентироваться в многообразии компьютерных игр. Выделяются следующие игровые жанры:

- аркадные игры;
- приключенческие игры;
- спортивные игры;
- симуляторы;
- ролевые игры;
- стратегические игры;
- обучающие игры.

Аркадные игры. Термин «аркада» появился ещё во времена существования игровых автоматов. Данный жанр еще называют динамическими играми. Игры достаточно просты в освоении, не содержат сложных изменений в игровом процессе. Однако выиграть сразу практически невозможно. Успех игрока во многом определяется его умениями и навыками, среди которых быстрота мышления, скорость реакции, свободное пользование клавиатурой. Игрок чаще всего управляет каким-то объектом и уничтожает противников на своём пути, что позволяет перейти с одного уровня на другой. При этом может быть использовано разнообразное оружие. В аркадной игре может быть разработана система различных бонусов: начисление очков, постепенно открываемые элементы игры, временное улучшение характеристик персонажа. Аркады популярны ещё и потому, что нет необходимости играть постоянно, не нужно сохраняться или следить за выполнением заданий. «*Super Meat Boy*», «*Subway Surfers*», «*Angry Birds*», «*Among Us*» – наиболее популярные аркадные игры.

Приключенческие игры или квест. Один из основных жанров компьютерных игр, представляющий собой интерактивную историю с главным героем, управляемым игроком. Важнейшими элементами игры в жанре квеста являются собственно повествование и исследование мира, а ключевую роль в игровом процессе играет решение головоломок и задач, требующих от игрока умственных усилий. Такие характерные для других жанров компьютерных игр элементы, как бои, экономическое планирование и задачи, требующие от игрока скорости реакции и быстрых ответных действий, в квестах сведены к минимуму или вовсе отсутствуют. «Portal», «Tomb Raider», «Machinarium», «L.A. Noire: The Complete», «Little Nightmares» – наиболее известные примеры приключенческих игр.

Спортивные игры. Основными пользователями данного жанра становятся любители спорта и фанаты. Все компьютерные игры – это аналоги настоящих видов спорта: авто- и мотогонки, единоборства, игры с мячом. Чаще всего в спортивных играх соперником пользователя становится компьютер, либо другой человек. Спортивные симуляторы, например, точно передают события, происходящие во время реального матча. Разработчики, привлекая потенциального пользователя, максимально приближают игру к реальности не только по части процесса. Они получают от спортивных клубов право на использование клубной и спортивной символики, реальной внешности игроков для создания персонажей. Среди спортивных игр встречается даже рыбалка – не столь популярная, но увлекательная игра, имитирующая ловлю рыбы. «NBA 2K», «FIFA», «PES», «Football Manager», «Rocket League» – наиболее популярные игры спортивного жанра.

Симуляторы. Игры-симуляторы занимают особое место в игровом разнообразии. Они имитируют различные действия, которые может совершать человек в реальной жизни. Игрок может управлять каким-либо техническим устройством: подводной лодкой, самолётом, автомобилем космическим кораблём или другим человеком. Однако симуляторы – это не только способ развлечения. Они широко используются в профессиональном

обучении. Симуляторы для обучения – это программное приложение, которое позволяет начинающему специалисту в виртуальном пространстве управлять процессом, оборудованием, механизмом, принимать решения в стандартных и нестандартных ситуациях. Наибольшее распространение получили технические, медицинские и бизнес-симуляторы, которые позволяют учиться эффективнее, совершенствовать навыки, минимизировать риски в профессиональной деятельности. «*Minecraft*», «*Sims*», «*Microsoft Flight Simulator*», «*Farming Simulator*», «Симулятор вождения ПДД: Учебное пособие для автошкол» – наиболее известные приложения данного жанра.

Ролевые (RPG) игры. По сюжету пользователи управляет вымышленными персонажами, которые осуществляют какие-либо действия в виртуальном мире. Персонажи игры имеют свои списки способностей, среди которых могут быть показатели силы, ловкости, интеллекта. В процессе игры персонаж может изменять свои базовые умения. Игры имеют захватывающий сюжет, пользователь, сжившись с персонажем, должен преодолеть замысловатые повороты сюжетных событий. «*Path of Exile*», «*Diablo*», «*Final Fantasy*», «*Mass Effect*» – примеры ролевых игр.

Стратегические игры. Стратегическая игра — жанр компьютерных игр, в которых игроку для победы необходимо применять стратегическое мышление. В популярных играх такого жанра игроку предлагается играть не за конкретного персонажа, а за их условные массы, к примеру, руководить строительством города или командовать целыми армиями в военных кампаниях. Некоторые разработчики выпускают исключительно экономические стратегии, где необходимо управлять сложной экономической системой, рассуждать логически, разрабатывать тактику и принимать верные решения, которые приведут к победе. «*Terraria*», «*Warcraft III*», «*Clash of Clans*» –наиболее известные игры-стратегии.

Обучающие игры. Многие специалисты сейчас говорят о колоссальном потенциале видеоигр в процессе обучения. Игрой нельзя заменить занятия, на которых происходит усвоение знаний, но она может стать отличным

дополнением, которое поможет по-новому взглянуть на изучаемый предмет. Игры позволяют совершенствовать умственные навыки: внимательность, логическое мышление, память. Без них не справиться с игровыми заданиями. Также многие обучающие игры способствуют развитию фантазии и творческого мышления, развивают навыки общения. К разработчикам обучающих игр предъявляются особые требования: игры должны быть эмоционально привлекательными, содержательными, чтобы сделать процесс усвоения новых знаний более успешным. «*Museum Madness*», «*Hello Kitty Painting*», «*IC: Познавательная коллекция. Играем и учимся.*» – примеры обучающих игр.

1.1.3 Анализ аналогов

Tower Defense (Башенная защита) – это жанр компьютерных стратегических игр. Задача игрока в играх подобного жанра - расправиться с наступающими врагами, до того, как они пересекут карту, с помощью строительства оборонительных сооружений (башен), атакующих, когда враги проходят вблизи. Противники и башни обычно различаются по характеристикам и цене. Когда враги побеждены, игрок зарабатывает деньги или очки, которые используются для покупки новых башен или модернизации уже имеющихся башен. Подбор вида башен и их расположение — неотъемлемая стратегия игры. Обычно враги пробегают через подобие лабиринта, что даёт игроку возможность стратегического размещения башен.

На сегодняшний момент вышло очень много игр данного жанра, и почти каждая старается внедрить что-то новое.

Серия *Bloons TD* - одна из самых популярных и успешных версий *Tower Defense* на мобильных устройствах. Последняя из них, *Bloons TD 6*, был запущен в 2018 году. В нем есть 20 карт, тонны улучшений, 19 башен, три пути обновления, автономная поддержка, новые игровые режимы и многое другое. Конечно, у предыдущих итераций игры тоже есть свои

прелести, и некоторые из них все еще получают обновления. На рисунке 2 представлен игровой интерфейс *Bloons TD 6*.

Количество скачиваний приложения: серия включает в себя 8 игр, скачивание которых составляет более 1.5 миллиона.

Средняя оценка игры: 4.7 балла.



Рисунок 2 – Интерфейс *Bloons TD 6*

Thing TD — это скорее гибрид, состоящий из элементов *Tower Defense* и *RPG*. Если вы чувствуете себя как дома в обоих жанрах, то здесь вы точно не ошибетесь. Игроку придется защищаться от демонов и нежити, сокрушая в процессе более 50 различных типов врагов. На протяжении 25 уровней вы можете доказать свои навыки и прибегнуть к помощи восьми разных героев, которые помогут вам. На рисунке 3 представлен игровой интерфейс *Thing TD*.

Количество скачиваний приложения: 100 000 скачиваний.

Средняя оценка игры: 4.0 балла.



Рисунок 3 – Интерфейс *Thing TD*

Defenders 2 - это то, что происходит, когда вы смешиваете игру в защиту башни и карточную игру вместе. Вы открываете башни, собирая карты. Тем не менее, есть 40 башен на выбор, все из которых должны быть разблокированы с помощью карт. Кроме того, в вашем распоряжении до 20 заклинаний и 29 боссов, от которых вам придется избавиться. Также есть режим *PvP*, если вы хотите проверить свои навыки против реальных игроков. На рисунке 4 представлен игровой интерфейс *Defenders 2*.

Количество скачиваний приложения: 80 000 скачиваний.

Средняя оценка игры: 4.2 балла.

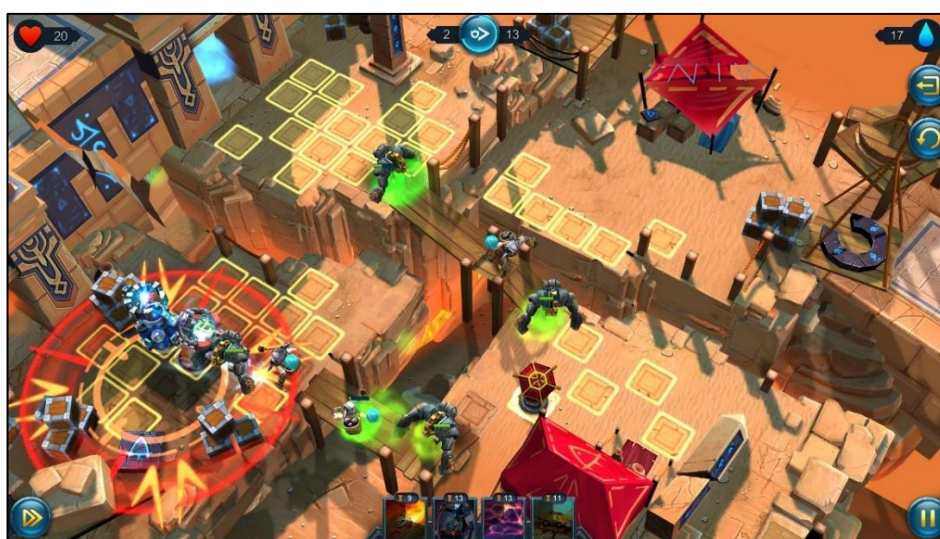


Рисунок 4 – Интерфейс *Defenders 2*

Изм.	Лист	№ докум.	Подпись	Дата

Element TD – одна из самых известных игр в жанре *Tower Defense*, основанной на комбинациях элементов. Элементы, которые вы выбираете, определяют башни, которые вы можете построить. Игра включает в себя: глобальную таблицу лидеров, чтобы соревноваться с друзьями и миром, 44 различных башни с уникальными способностями и несколькими путями улучшения, 56 различных типов крипов, которые бросят вызов вашей обороне, 5 разных карт, чтобы разнообразить свою тактику, 6 различных элементов (Свет, Тьма, Вода, Огонь, Природа, Земля). На рисунке 5 представлен игровой интерфейс *Element TD*.

Количество скачиваний приложения: 500 000 скачиваний.

Средняя оценка игры: 4.8 балла



Рисунок 5 – Интерфейс *Element TD*

1.1.4 Анализ инструментов разработки для реализации проекта

В настоящее время существует огромное количество как платных, так и бесплатных инструментов по разработке игр, начиная от простых библиотек для известных языков программирования, заканчивая крупными редакторами с обширным функционалом. Для создания качественной игры необходимо использовать качественные инструменты.

Движок — это программное обеспечение для разработки игр. Движки содержат множество разнообразных инструментов, которые сильно облегчают задачу: шаблоны игровой логики и поведения объектов, инструменты для дизайна пейзажей и ландшафта, работы с графикой и физикой объектов, визуальный редактор сцен, утилиту для анимаций и др. Ещё движки поддерживают скрипты и *API*. Скрипты помогают писать команды, которые будут выполняться игрой всё время или после каких-то действий игрока. *API* же помогает упростить написание скриптов. Движки «оживляют» игру. Поэтому игры уже давно не создают с нуля. Даже код не всегда надо писать — всё уже есть. Но чтобы процесс шёл быстрее, важно тщательно изучить один из движков [6].

Unity – это межплатформенная среда разработки приложений и игр, разработанная американской компанией *Unity Technologies*, работающая с операционными системами *Windows* и *OS X*. *Unity*, позволяет разрабатывать как двумерные, так и трехмерные приложения, и игры. Среда разрешает компилировать проект под множество устройств. В этот перечень попадают устройства с ОС для ПК и *MAC*: *Windows*, *OS X*, *Linux*, с ОС для смартфонов: *Android*, *iOS*, *Windows Phone*, и даже под игровые консоли: *PlayStation*, *Wii*, *Xbox*. Помимо всего прочего, проект может быть запущен и в браузере, но только с помощью специального плагина от компании разработчиков *Unity* – *UWP (unity web player)* [7].

Среда имеет распространенный в среде редакторов *Drag&Drop*-интерфейс, который состоит из различных окон, которые, в свою очередь, очень просто настраиваются по высоте, ширине и размерам. Имеются также инструменты графики и дизайна: инструмент временной шкалы для роликов, управление камерой, эффекты пост-обработки (цветокоррекция), инструменты анимации, инструменты для проектирования уровней и освещения. Игровой движок *Unity3D* полностью интегрирован в среду разработки *Unity*. Тесная интеграция позволяет прямо в редакторе получить

все то, на что будет способна выпущенная игра. Также *Unity* предоставляет *API* для доступа к средствам ввода и настройкам *Android* и *iOS*.

Среди явных достоинств среды разработки *Unity* можно выделить: мультиплатформенность, высокие возможности функционала, удобная визуальная и графическая составляющие, большая библиотека *Assets* и плагинов, позволяющие расширить функционал, возможность интегрировать игры со многими графическими или анимационными приложениями.

Unity поддерживает два сценарных языка, это *C#* (модифицированный) и *JavaScript*, что позволяет широкому кругу программистов пользоваться платформой. Отлично подойдет для разработки мобильных *2D* и *3D*-игр, одиночных и сетевых игр для десктопа, *VR*-игры [8]. На рисунке 6 представлен интерфейс *Unity* и окно пользовательского интерфейса.

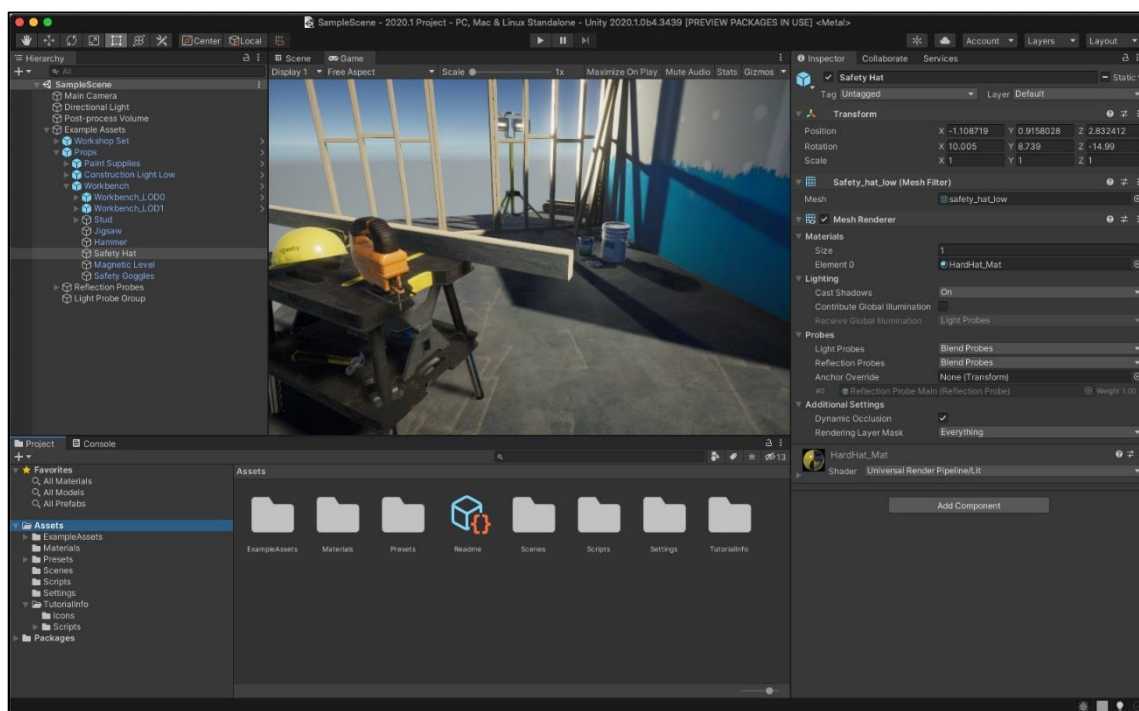


Рисунок 6 – Интерфейс *Unity IDE*

Unreal Engine – это среда разработки игр от американской компании *Epic Games*, написанный на языке *C++*, и запускаемый на всех наиболее популярных ОС для ПК и *Mac*: *Windows*, *Linux*, *OS X*.

Кроссплатформенность среды позволяет создавать игры для большинства операционных систем и платформ, в этот список входят: ПК, Mac, Xbox, GameCube, Wii, PlayStation, iOS, Android и Nintendo Switch. Unreal Engine имеет целый ряд упрощенных методов портирования, используя модульную систему зависимых компонентов. Всё необходимое можно сделать прямо в программе: здесь есть целых два редактора для работы с эффектами и материалами и Blueprints — инструмент, который позволяет строить логику игры через блок-схемы. Поддерживает язык программирования, C++, и имеет ряд достоинств и недостатков.

К достоинствам среды разработки Unreal Engine можно отнести: высокие возможности функционала, мультиплатформенность, гибкость и простота интерфейса, поддержка графического реактора для создания алгоритмов, возможность создавать игры для VR устройств. Отлично подойдет для разработки AAA-игр с большим бюджетом, шутеры, приключенческие экшены, VR-игры. На рисунке 7 представлен интерфейс Unreal Engine и окно пользовательского интерфейса.



Рисунок 7 – Интерфейс Unreal Engine IDE

Изм.	Лист	№ докум.	Подпись	Дата

CryEngine — мощный движок от немецкой компании *Crytek*. На нём можно делать масштабные игры для любых платформ — десктопа, веба и мобильных устройств. Очевидные достоинства данного движка являются: мультиплатформенность, широкие графические возможности значительно превышающие, *Unity* и *Unreal Engine*, глубокий и реалистичный артхаусный свет, реалистичная физика объектов, возможность выстраивать сложную многоуровневую систему, продвинутая анимация. *CryEngine* подойдёт опытным командам, которые выпускают дорогие и ресурсоёмкие проекты. Но и изучить *CryEngine* не так легко, по сложности изучения он будет на первом месте из трёх движков. Отлично подойдет для разработки крупномасштабных ресурсоёмких игр и игр под ПК и консоли, включая *PS5* и *Xbox One*. На рисунке 8 представлен интерфейс *CryEngine* и окно пользовательского интерфейса.

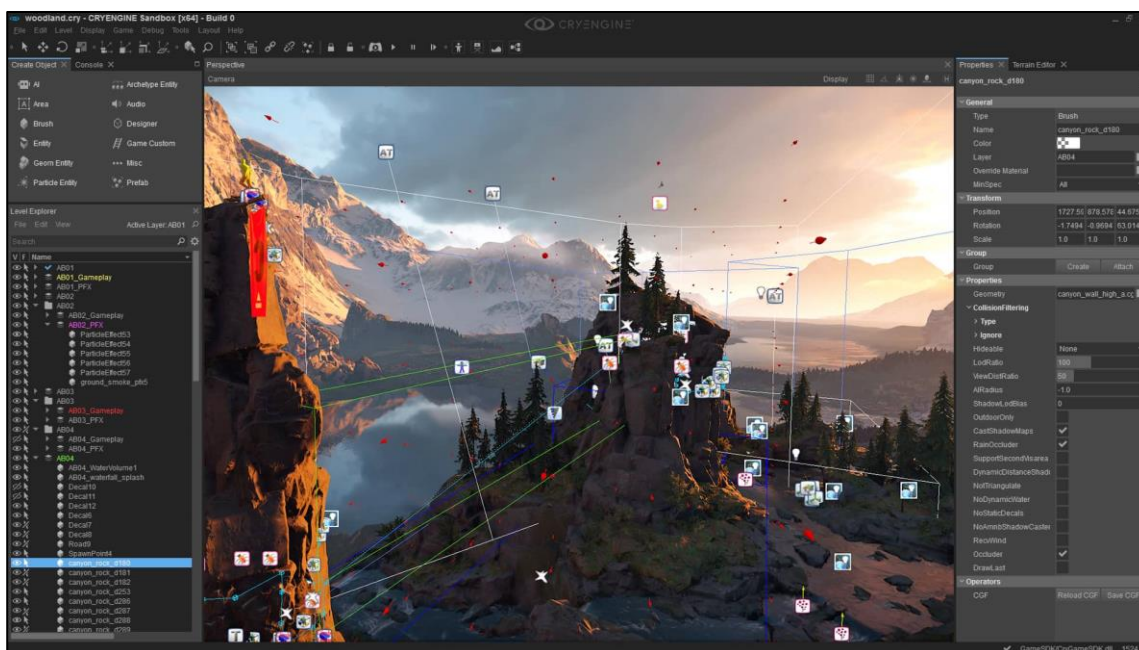


Рисунок 8 – Интерфейс *CryEngine IDE*

Учитывая все преимущества и недостатки представленных средств разработки игровых приложений – был выбран движок *Unity IDE*.

1.1.5 Анализ языков программирования

На текущий момент есть широкий выбор языков программирования, позволяющих задавать условия и команды в игре. Движки много берут на себя, но все равно программировать нужно много. Перемещение по меню, переход между локациями, управление персонажем, движение камеры, смена музыки, диалоги, система квестов — для всего этого нужно писать код. Выбор языка зависит от специфики проекта, поставленных целей и задач, а также необходимости создания кода с нуля или адаптации его под тот или иной игровой движок.

Сегодня для создания игр для ПК, консолей и мобильных устройств могут использоваться несколько языков одновременно. Самыми популярными среди разработчиков игр на сегодняшний день считаются следующие.

C++ — компилируемый и статически типизированный язык программирования. Поддерживает ООП и процедурное программирование. Также является кроссплатформенным языком, который позволяет эффективно управлять памятью и при этом не загружать её. Без него не обходится ни одна разработка серьезного игрового продукта. Кроссплатформенность позволяет создавать игры для разных платформ: веб, онлайн или мобильные устройства. Этот язык помогает создавать механику игры — взаимодействие игрока с миром и мира с игроком. На нём пишут оптимизированные крупномасштабные игры уровня AAA. C++ также используют в мобильных играх, где нужно бережно обходиться с памятью устройства и поддерживать высокую производительность игры. Отлично подойдет для ресурсоёмких десктопных игр, игр на *Unreal Engine*, мобильные игры.

C# — компилируемый и статически типизированный язык программирования. Объектно-ориентированный язык программирования. Уступает по скорости C++, но обладает более ёмким, в плане возможностей, синтаксисом. Не настолько производителен как C++ и, соответственно, не

подойдёт для больших требовательных проектов. Но *C#* отличается кроссплатформенностью и универсальностью, поэтому на нём можно создавать 2D- и 3D-игры для *Android* и *iOS*. Знание *C#* позволяет разрабатывать мобильные приложения, которые успешно запускаются на всех моделях современных консолей, смартфонов и планшетов. Этот язык применялся при разработке "движков" популярных приложений: *Subway Serfers*, *Веселая Ферма*, *Angry birds* и других. Ключевую роль язык высокого уровня *C#* играет при разработке крупных проектов, связанных с созданием сайтов и интернет-порталов. Применение *C#* позволяет реализовать сложные интернет-магазины, форумы, доски объявлений, интернет-банкинг. Отлично подойдет для игр на разные платформы, игры на *Unity*, мобильные игры.

Java — является как компилируемым, так и интерпретируемым языком программирования. Популярность этого языка обусловлена его многопоточностью и возможностью беспрепятственного взаимодействия с памятью мобильных устройств. Надёжный кроссплатформенный язык, который предназначен для создания мобильных игр для *Android*. Всё потому, что эта операционная система сама по себе отчасти написана на *Java*. А *IDE Android Studio* — популярная программа для разработки мобильного софта на *Android*, которая этот язык «понимает» по умолчанию. Отличается хорошим взаимодействием с движками и программами графического и звукового оформления. На *Java* часто реализуют серверную структуру для многопользовательских игр.

Если необходимо разработать приложения для *IOS*, используются языки *Swift* и *Objective-C*. *Objective-C* был создан как объектно-ориентированное расширение языка *C*. Начиная с 90-х он используется для написания программного обеспечения для устройств *Apple*. Разработчики приложений на *Objective-C* для мобильных устройств *Apple* имеют доступ к многочисленным фреймворкам для создания игр. *Swift* используется для разработки приложений под *IOS 7* и выше. Несмотря на короткое время существования на рынке, *Swift* обогнал *Objective-C* по популярности. Он

отличается от предшественника безопасностью типов, автоматическим управлением памятью и безопасностью при использовании указателей.

JavaScript — лучше всего подходит для интерактивных онлайн-игр, поскольку дает возможность интегрировать код в других языках программирования. Такая межплатформенность приводит к созданию все большего числа мобильных игр. Язык поддерживает большое количество библиотек и фреймворков. Также преимуществом языка является автоматическое управление памятью. Для каких игр подойдет: браузерные игры, мобильные игры.

Учитывая все преимущества и недостатки представленных языков программирования – был выбран язык *C#*.

1.2 Проектирование и разработка

1.2.1 Структура приложения

Разработанный проект содержит несколько каталогов, в которых хранятся: шаблоны объектов, игровые сцены, скрипты, изображения, шрифты, вражеские объекты, иконки приложения, материалы, спрайты, объекты башен. Файловая структура представлена на рисунке 9.

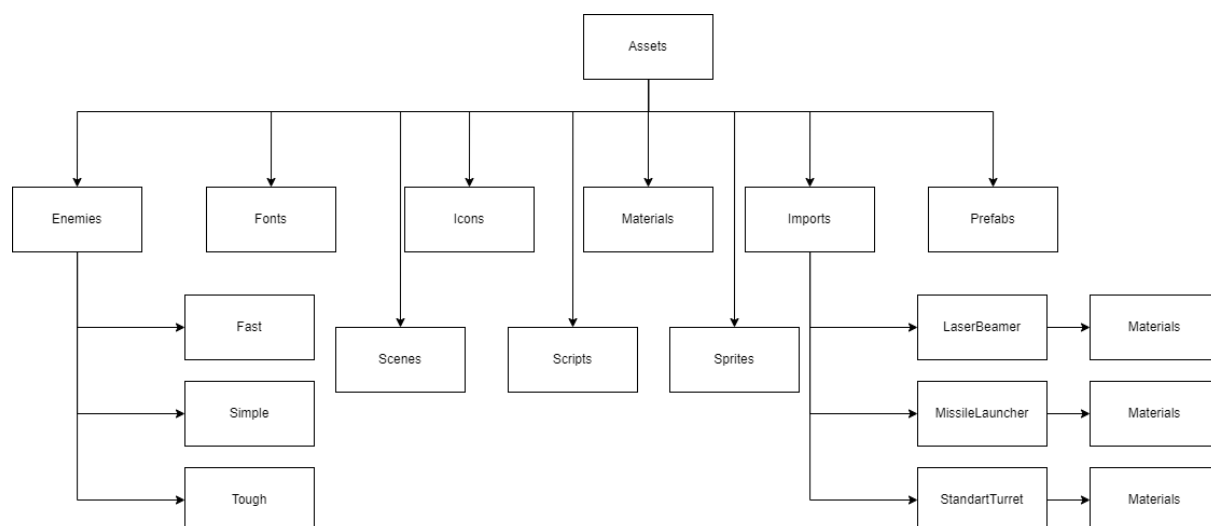


Рисунок 9 – Файловая структура приложения

Каталог *Enemies* включает в себя три подкаталога: *Fast, Simple, Tough*. Каждый из них включает в себя свойства определенного вражеского объекта. Каждый их подкаталогов содержит *prefab* (экземпляр) вражеского объекта, его материал и эффект при уничтожении.

В папке *Fonts* находятся шрифты, используемые для текста в игре.

Директория *Scenes* содержит все сцены игры, в которые может перейти пользователь.

В папке *Icons* загружены 2D-текстуры башен.

В папке *Scripts* находятся скрипты с описанием всех классов и взаимодействий.

В директории *Materials* содержатся все материалы, используемые при игровом процессе.

Каталог *Sprites* содержит 2D-текстуры спрайтов.

Каталог *Imports* включает в себя три подкаталога: *Laser Beamer, Missile Launcher, Standard Turret*. Каждый из них включает в себя свойства определенного объекта типа башни. Каждый их подкаталогов содержит 3D-объект башни, его материал и объект *Bullet*, который является снарядом.

В директории *Prefabs* находятся готовые шаблоны игровых объектов, с помощью которых можно составить игровой уровень после его проектирования.

1.2.2 Реализация игрового поля

Первое, что необходимо для игры типа «*Tower Defense*» – это игровое поле. Игровое поле было решено сделать из кубических блоков. Начальный вариант поля имеет размер 16x16. Были удалены некоторые блоки и заменены на блоки другого типа, которые будут представлять из себя путь, который должен пройти вражеский объект. Общий вид игрового поля представлен на рисунке 10.

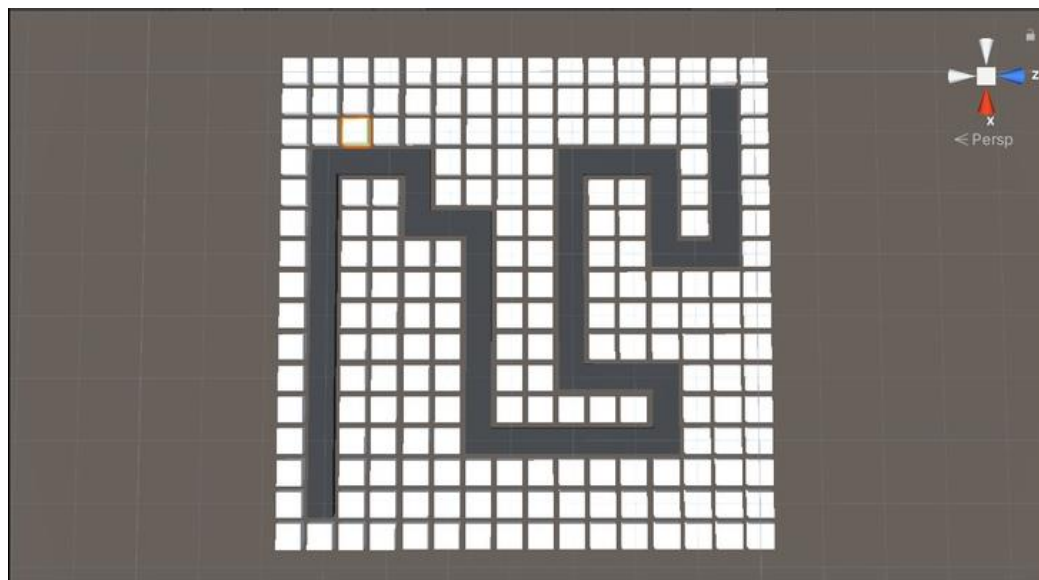


Рисунок 10 – Игровое поле

Следующей целью было создать точки старта, которая представляет из себя место появления вражеских объектов, и точку финиша, к которой движется объект. Точки старта и финиша представлены на рисунке 11. Действие вражеских мобов – путь из точки *A* (*START*) в точку *B* (*END*). В точке *A* – объект создается и начинает свой путь. Точка *B* – объект стремиться к ней, при её достижении объект уничтожается. На рисунке 12 представлен вид на игровое поле в режиме Game.

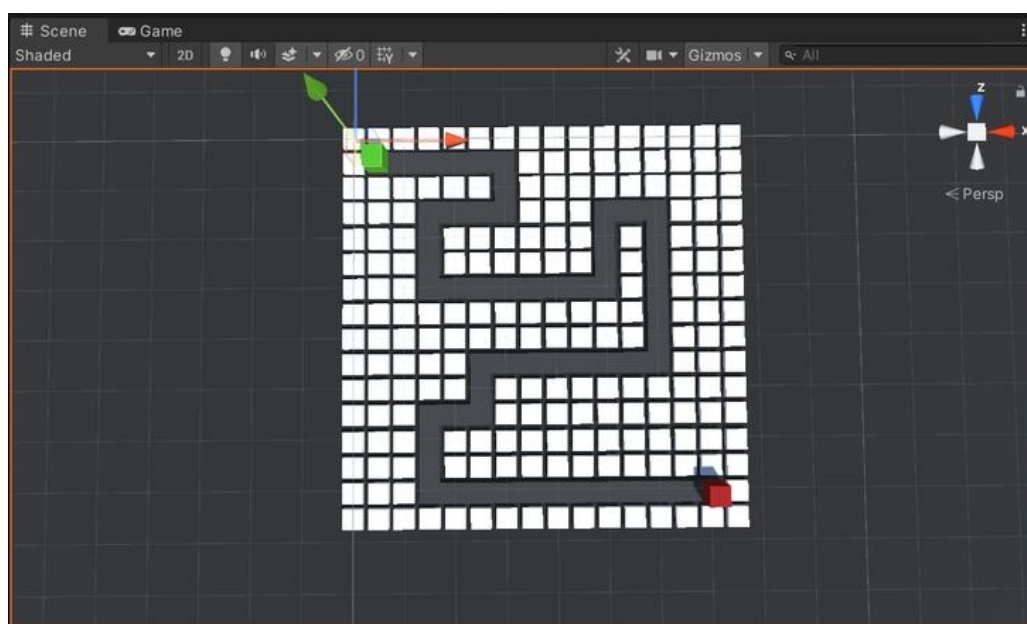


Рисунок 11 – Вид сверху игрового поля

Изм.	Лист	№ докум.	Подпись	Дата

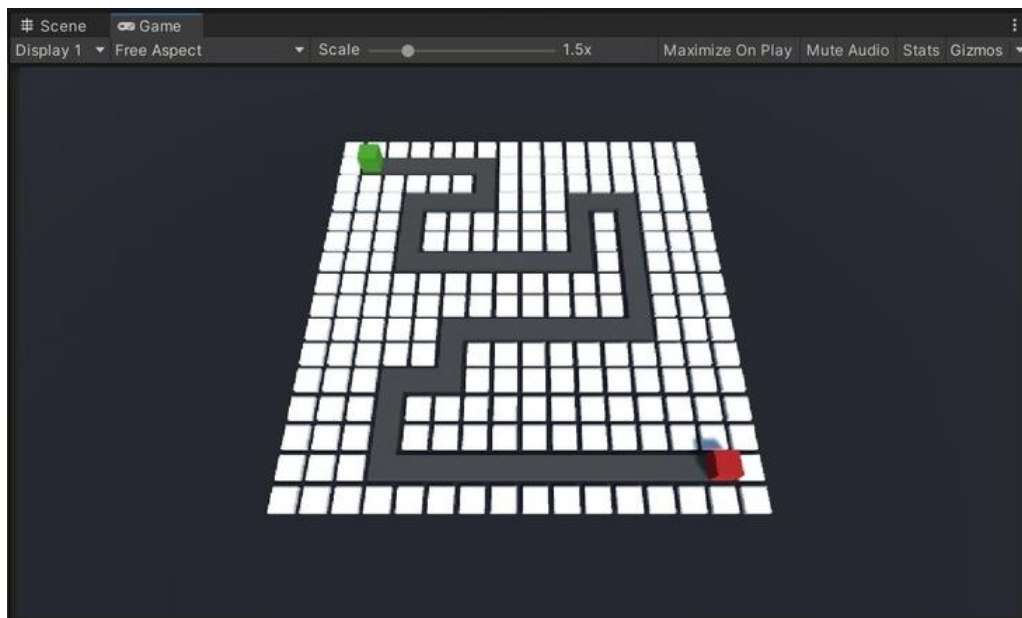


Рисунок 12 – Игровое поле в режиме *Game*

1.2.3 Реализация противников

Противник представляет из себя объект сферического типа (рисунок 13). Над объектом расположена полоса, представляющая здоровье противника. При нанесении атаки от башен, здоровье врага будет уменьшаться, в зависимости от нанесенного урона (рисунок 14).

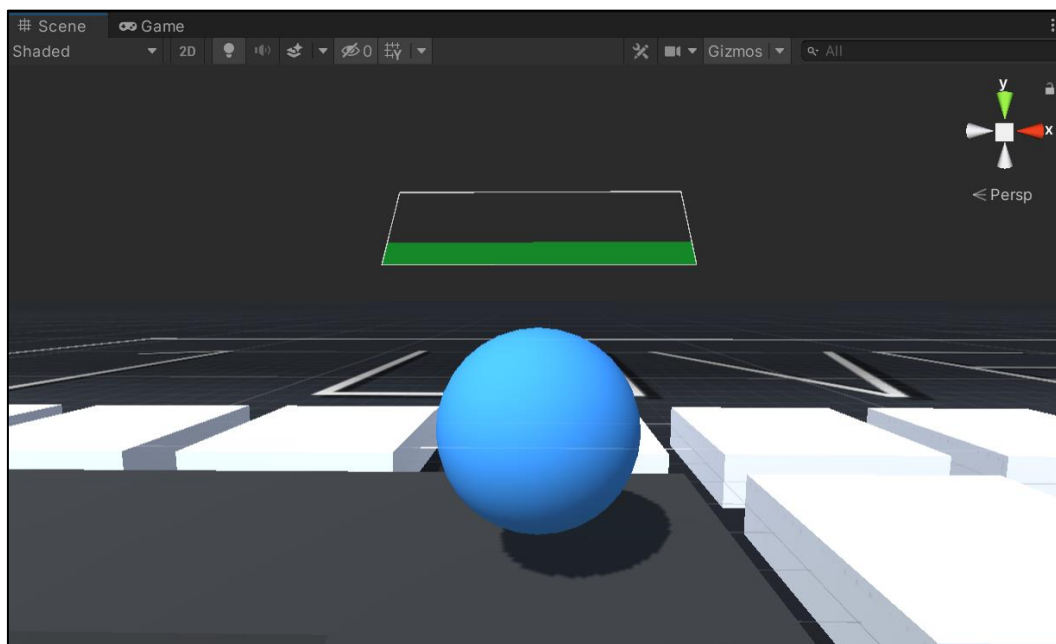


Рисунок 13 – Вражеский объект

Изм.	Лист	№ докум.	Подпись	Дата

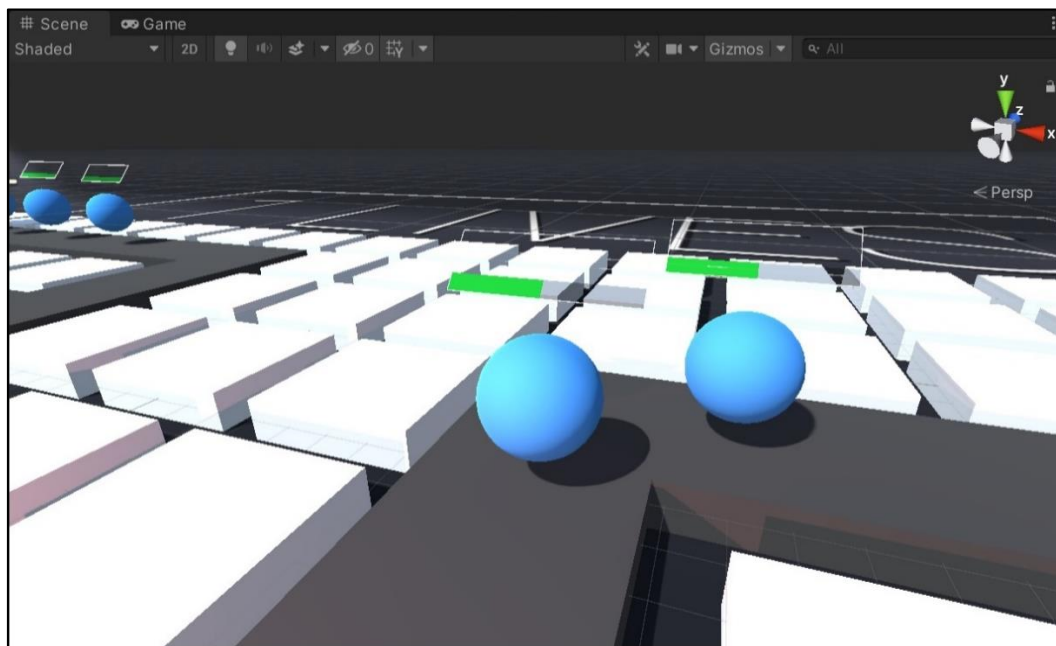


Рисунок 14 – Изменение здоровья при попадании

В классе *Enemy.cs* содержатся основные характеристики вражеского объекта. Поля класса *Enemy.cs* представлены в таблице 1.

Таблица 1 – Описание полей класса *Enemy.cs*

Тип данных	Название	Описание
<i>float</i>	<i>startSpeed</i>	Стартовая скорость объекта
<i>float</i>	<i>speed</i>	Скорость объекта
<i>float</i>	<i>startHealth</i>	Стартовое здоровье объекта
<i>float</i>	<i>health</i>	Здоровье объекта
<i>int</i>	<i>worth</i>	Количество игровой валюты, которое будет получено игроком при уничтожении объекта
<i>Image</i>	<i>healthBar</i>	Объект типа <i>Image</i> , который отображает полосу здоровья
<i>bool</i>	<i>isDead</i>	Проверка уничтожен ли объект

Класс *Enemy.cs* включает в себя 4 метода. Далее рассмотрим описание методов данного класса.

Private void Start() – метод, который устанавливается начальное здоровье объекта, его скорость передвижения.

Public void TakeDamage() – метод, предназначенный для отображения здоровья вражеского объекта. Также отображает изменение здоровья при нанесении урона.

Public void Slow() – метод, уменьшающий показатели скорости, при нанесении урона от башни типа *LaserBeamer*.

Private void Die() – метод, проверяющий, что объект должен быть уничтожен, после чего прибавляет значение к игровой валюте, отображает эффект при уничтожении объекта и уничтожает объект с игрового поля.

Класс *Wave.cs* содержит поля, отвечающие противников и *GameObject* ему – создающий объект враги через готовый экземпляр объекта. Поля класса *Wave.cs* представлены в таблице 2.

Таблица 2 – Описание полей класса *Wave.cs*

Тип данных	Название	Описание
<i>GameObject</i>	<i>enemy</i>	<i>Prefab</i> вражеского объекта
<i>int</i>	<i>count</i>	Количество волн
<i>float</i>	<i>rate</i>	Количество врагов

Основная цель вражеских мобов – пересечь карту. Для того, чтобы объект двигался были добавлены объекты *Waypoints*, точки к которым будет стремиться вражеский объект. Именно они определяют путь мобов (рисунок 15, 16). Поля класса *Waypoints.cs* представлены в таблице 3.

Таблица 3 – Описание полей класса *Waypoints.cs*

Тип данных	Название	Описание
<i>Transform[]</i>	<i>points</i>	Массив объектов <i>waypoints</i>

Класс *Waypoints.cs* содержит один метод *public void Awake()*, который состоит из цикла, где возвращает каждый элемент массива *points* и преобразует его в объект, который будет отображен на карте, к которому будет двигаться противник.

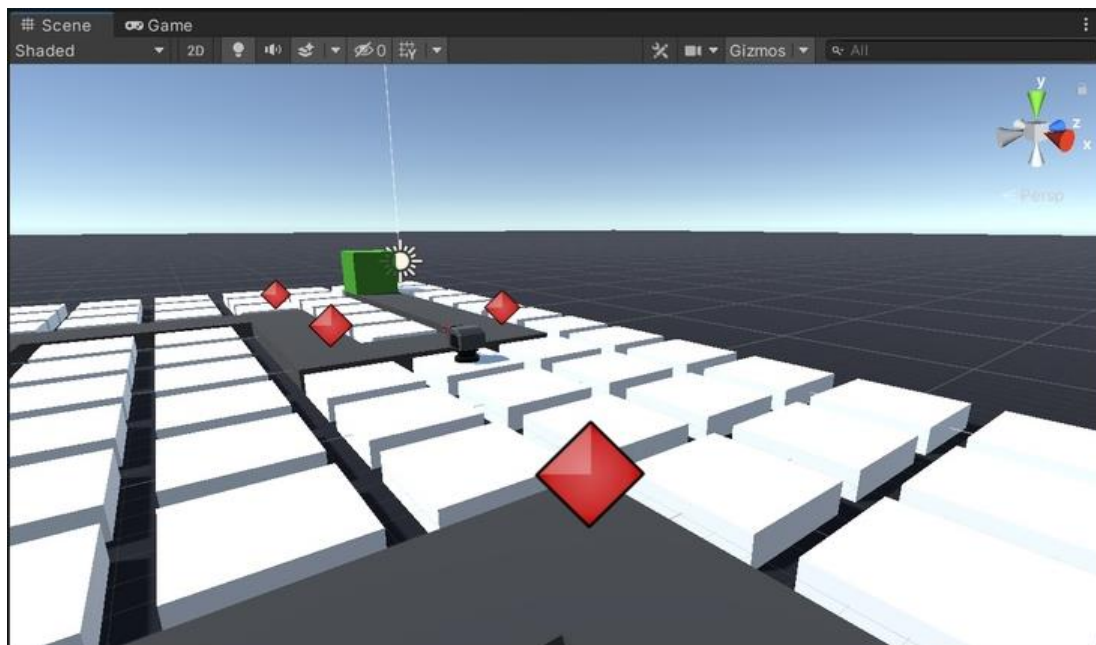


Рисунок 15 – Игровое поле с *Waypoints*

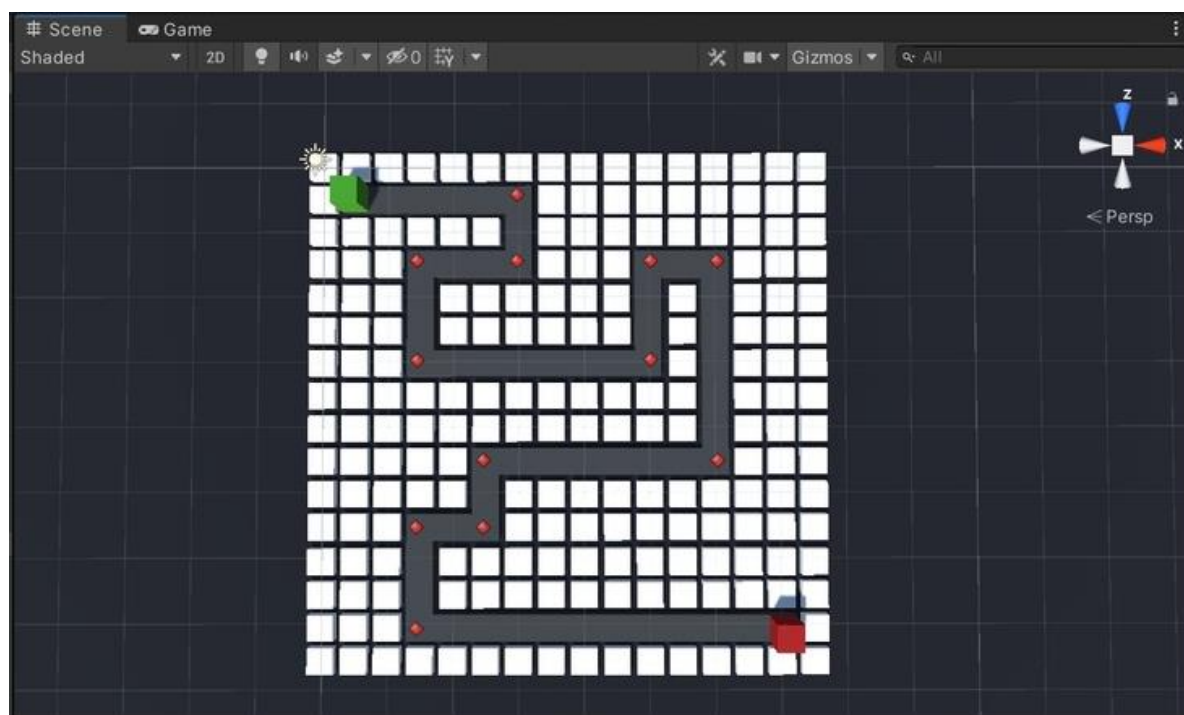


Рисунок 16 – *Waypoints*

В классе *EnemyMovement.cs* определены методы перемещения объекта. Поля класса *EnemyMovement.cs* представлены в таблице 4.

Таблица 4 – Описание полей класса *EnemyMovement.cs*

Тип данных	Название	Описание
<i>Transform</i>	<i>target</i>	Объект класса <i>Transform</i> , который присваивается объекту <i>Waypoints</i>
<i>int</i>	<i>wavePointIndex</i>	Индекс текущей волны
<i>Enemy</i>	<i>enemy</i>	Объект класса <i>Enemy</i>

Класс *EnemyMovement.cs* включает в себя 4 метода. Описание методов данного класса представлено ниже.

Private void Start() – метод, который присваивает объекты *enemy* и *target*.

Private void Update() – метод, отвечающий за передвижение объекта, устанавливает скорости для перемещения для противника.

Private void GetNextWaypoint() – метод, представляющий собой путь объекта, движение от одного *Waypoints* к другому.

Private void EndPath() – метод, предназначенный для уменьшения жизней игрока, остановки порождения новой волны при достижении врага конца поля.

После добавления возможности появления вражеского объекта, был создан генератор волн противников. С точки старта появляется объект и начинает двигаться по своему пути, спустя некоторое время появляется следующая волна и так далее, пока количество волн текущего уровня не закончится (рисунок 17). За генерацию волн объектов отвечает класс *WaveSpawner.cs*. Поля класса *WaveSpawner.cs* представлены в таблице 5.

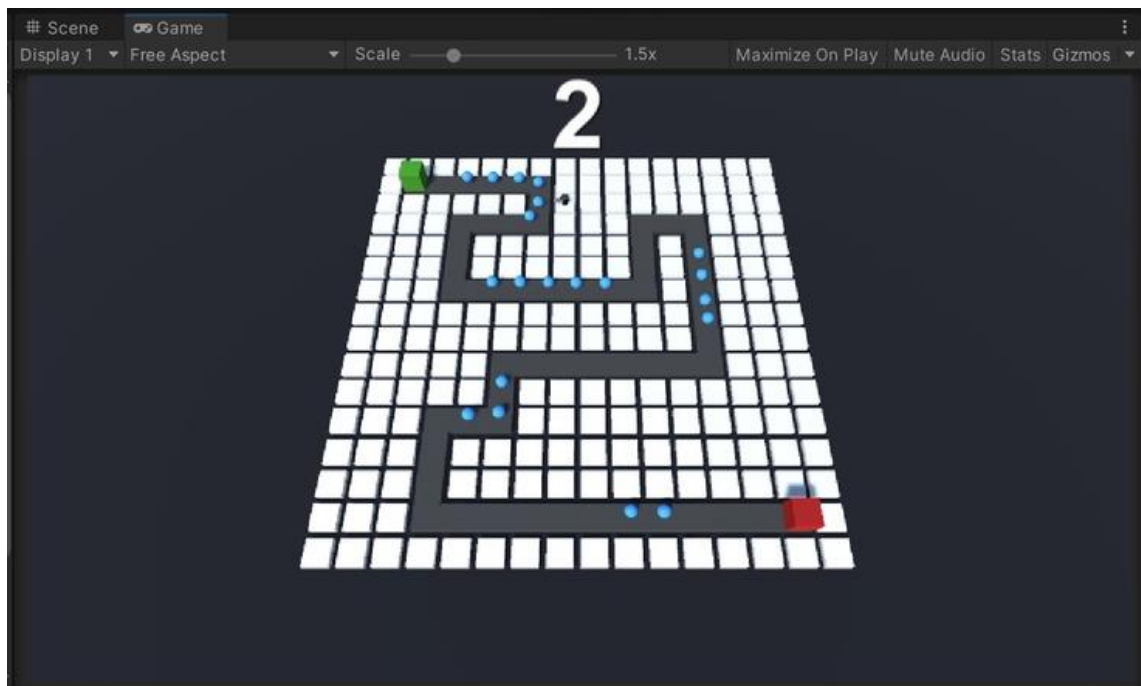


Рисунок 17 – Движение волн мобов

Таблица 5 – Описание полей класса *WaveSpawner.cs*

Тип данных	Название	Описание
<i>int</i>	<i>EnemyiesAlive</i>	Количество не уничтоженных объектов
<i>Wave[]</i>	<i>waves</i>	Массив количества волн
<i>Transform</i>	<i>spawnPoint</i>	Объект класса <i>Enemy</i>
<i>float</i>	<i>timeBetweenWaves</i>	Время до появления новой волны
<i>Text</i>	<i>waveCountdownText</i>	Объект типа <i>Text</i> , отображает надпись в текстовом поле
<i>float</i>	<i>countDown</i>	Время до появления новой волны
<i>int</i>	<i>waveIndex</i>	Номер волны
<i>GameManager</i>	<i>gameManager</i>	Объект класса <i>GameManager</i>

Класс *WaveSpawner.cs* включает в себя 4 метода.

Private void Start() – начальный метод, который устанавливает значение текущей волны.

Private void Update() – метод, предназначенный для проверки закончилась ли волна, также отображает время до новой волны.

Private void SpawnWave() – метод определяет номер волны которая должна быть создана, количество необходимых врагов.

IEnumerator SpawnEnemy() – метод, отвечающий за создание нового объекта в точке старта.

1.2.4 Реализация башен

Следующий шаг – добавление в игру башен, которые будут уничтожать противников. В игре представлен выбор из трех видов башен: *Standard Turret* (рисунок 18), *Missile Launcher* (рисунок 19), *Laser Beamer* (рисунок 20). 3D модели башен были созданы в *Blender* и добавлены в *Unity* проект. Каждая из турелей состоит из двух частей – основание и голова башни, подвижная часть турели (рисунок 21). Подвижная часть башни реагирует на врагов и следует по направлению за ними.

Каждая из башен имеет свой собственный и уникальный тип атаки. Для *Standard Turret* – это обычный снаряд, *Missile Launcher* – разрывной снаряд, наносящий урон врагам поблизости, *Laser Beamer* – лазер, наносящий периодический урон.

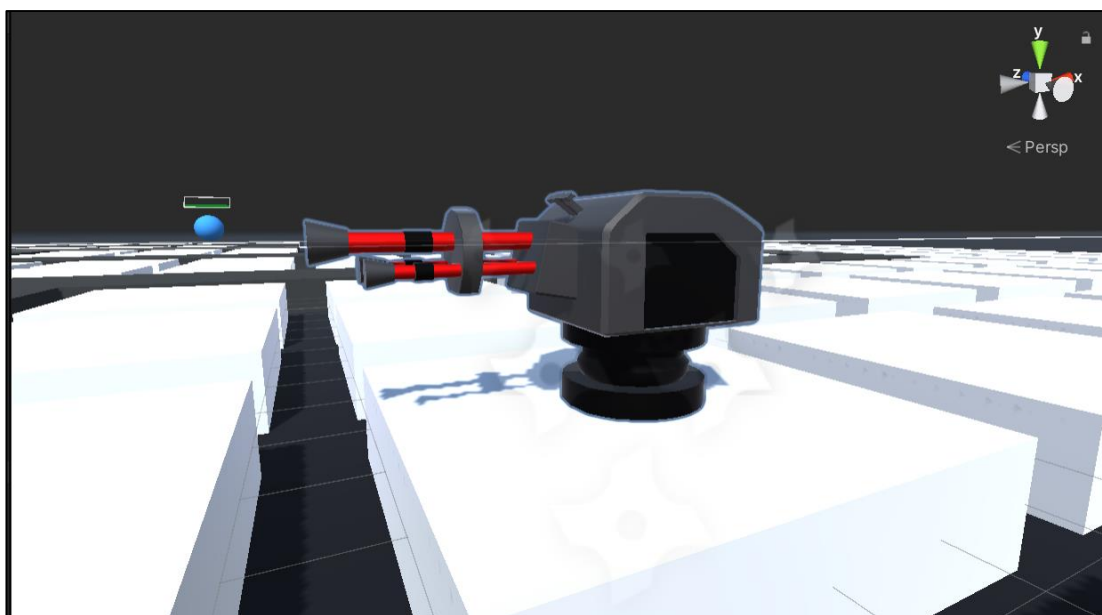


Рисунок 18 – *Standard Turret*

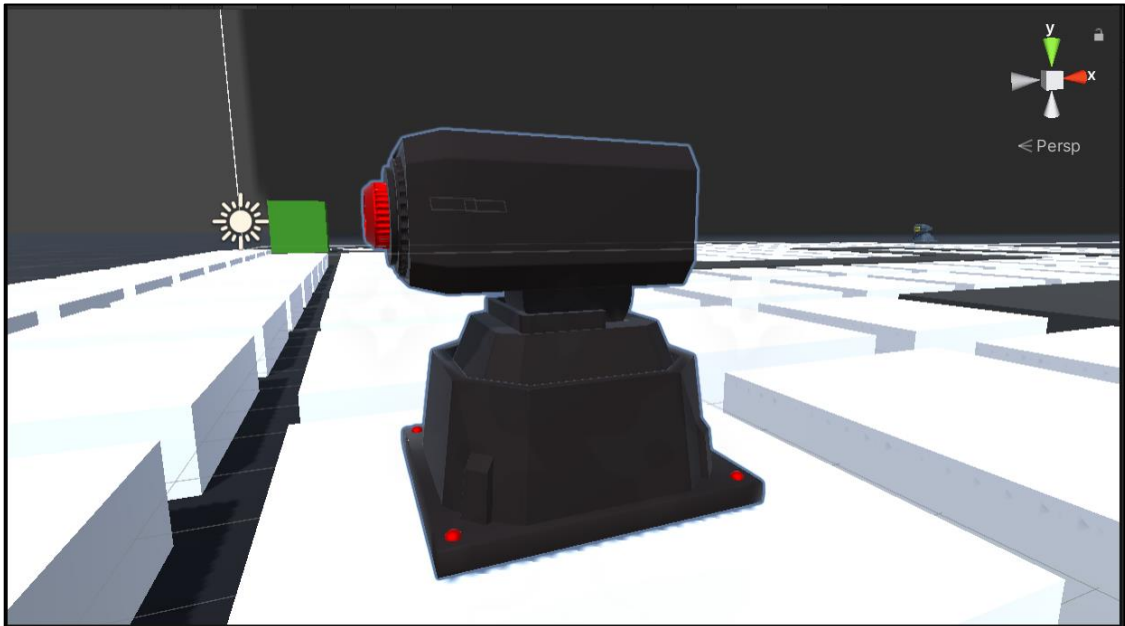


Рисунок 19 – *Missile Launcher*

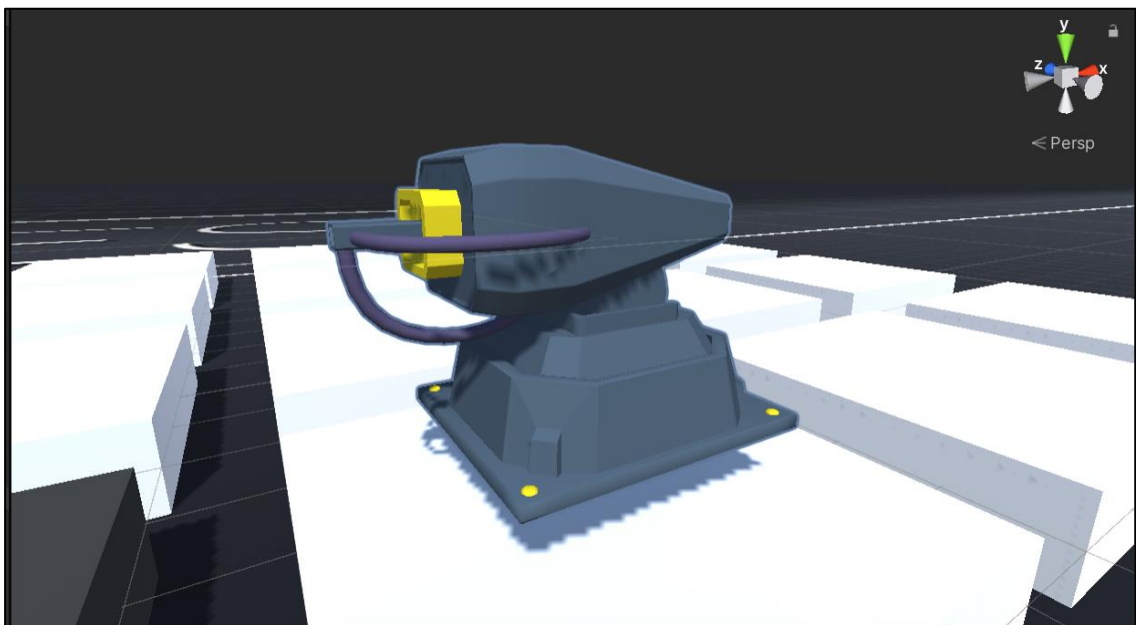


Рисунок 20 – *Laser Beamer*

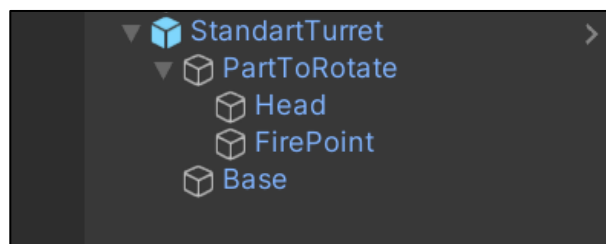


Рисунок 21 – Составные части башни

Класс *Turret.cs* отвечает за основные действия башен, а именно выстрелы, реакция на врагов, движение головы башен, переключение на другой объект, отдельные методы анимации для каждой из башен. Поля класса *Turret.cs* представлены в таблице 6.

Таблица 6 – Описание полей класса *Turret.cs*

Тип данных	Название	Описание
<i>Transform</i>	<i>target</i>	Объект типа <i>Transform</i> , являющийся башней
<i>Enemy</i>	<i>targetEnemy</i>	Объект класса <i>Enemy</i>
<i>float</i>	<i>range</i>	Расстояние реагирования башни
<i>GameObject</i>	<i>bulletPrefab</i>	Экземпляр снаряда башни
<i>float</i>	<i>fireRate</i>	Количество снарядов
<i>float</i>	<i>fireCountdown</i>	Значение, являющееся временем до появления следующего снаряда
<i>bool</i>	<i>useLaser</i>	Значение использования лазера
<i>LineRenderer</i>	<i>lineRenderer</i>	Объект типа <i>LineRenderer</i> , создающий луч
<i>int</i>	<i>damageOverTime</i>	Значение периодического урона башни
<i>float</i>	<i>slowAmount</i>	Значение замедления
<i>string</i>	<i>enemyTag</i>	<i>Tag</i> противника
<i>Transform</i>	<i>partToRotate</i>	Объект <i>Transform</i> , являющийся головой башни
<i>float</i>	<i>turnSpeed</i>	Скорость поворота головы башни
<i>Transform</i>	<i>firePoint</i>	Объект <i>Transform</i> , являющийся снарядом башни

Класс *Turret.cs* включает в себя 7 методов. Их описание приведено ниже.

Private void Start() – начальный метод, который устанавливает стартовое положение башни.

Private void Update() – метод, который проверяет установлена ли башня. Если да, то проверяет какой у неё тип.

Private void LockOnTarget() – метод отвечает за движение головы башни.

Private void Laser() – метод отвечает за выстрел из *LaserBeamer* башни.

Private void UpdateTarget() – метод отвечает за реакцию башни на врага. Он следит за перемещением *enemies* (вражеский объект). В случае, когда *enemies* появляются в поле *Gizmos*, башня реагирует на него. В момент, когда *enemies* покидают поле, башня переключает свое внимание на другой ближайший объект, если он имеется.

Private void OnDrawGizmosSelected() – метод, в котором определяется объект *Gizmos*, который устанавливает значение расстояния, в котором башня будет реагировать на движение врагов.

Private void Shoot() – метод, отвечающий за выстрел башни. Создает объект типа *Bullet*, который является экземпляром снаряда и направляет его во врага.

Bullet.cs – класс, отвечающий за снаряды башен, определяет урон, радиус взрыва, скорость снаряда, эффекты при попадании. Поля класса *Bullet.cs* представлены в таблице 7.

Таблица 7 – Описание полей класса *Bullet.cs*

Тип данных	Название	Описание
<i>Transform</i>	<i>target</i>	Объект типа <i>Transform</i> , являющийся вражеским объектом
<i>float</i>	<i>speed</i>	Скорость снаряда
<i>float</i>	<i>explosionRadius</i>	Радиус в котором будет получен урон от разрывного снаряда

Тип данных	Название	Описание
<i>int</i>	<i>damage</i>	Урон снаряда
<i>GameObject</i>	<i>impactEffect</i>	Объект эффекта при попадании по вражескому объекту

Класс *Bullet.cs* включает в себя 6 методов, описанных ниже.

Private void Update() – метод, который при выполненных условиях производит выстрел по врагу, со скоростью и уроном установленным для каждой из башни.

Public void Seek() – конструктор класса, принимающий объект *target*.

Private void HitTarget() – метод, предназначенный для добавления эффекта при попадании по врагу.

Private void Explode() – метод необходимый для *MissileLauncher*, наносит урон в области разрывного снаряда.

Private void Damage() – метод, проверяющий попал ли снаряд по объекту. При попадании переход в метод получения урона класса *Enemy*.

Private void OnDrawGizmosSelected() – метод, который определяет объект *Gizmos*, устанавливающий значение области действия для разрывного снаряда.

В игре типа «*Tower Defense*» имеется возможность улучшения и продажи башен (рисунок 22). Улучшенные версии башен имеют увеличенные характеристики, в зависимости от типа башни это может быть урон от снаряда, его скорость или более сильное замедление, а также измененный дизайн и размеры.



Рисунок 22 – Панель улучшения и продажи башен в игре

Класс *TurretBlueprint.cs* содержит *prefabs* для простых и улучшенных версий башен (рисунок 23, 24, 25), их стоимость, а также метод *Private int GetSellAmount()*, определяющий какой должна быть цена, которую получит игрок за продажу. Поля класса *TurretBlueprint.cs* представлены в таблице 8.

Таблица 8 – Описание полей класса *TurretBlueprint.cs*

Тип данных	Название	Описание
<i>GameObject</i>	<i>prefab</i>	Экземпляр стандартной башни
<i>int</i>	<i>cost</i>	Значение стоимости башни
<i>GameObject</i>	<i>upgradePrefab</i>	Экземпляр улучшенной башни
<i>int</i>	<i>upgradeCost</i>	Значение улучшения башни

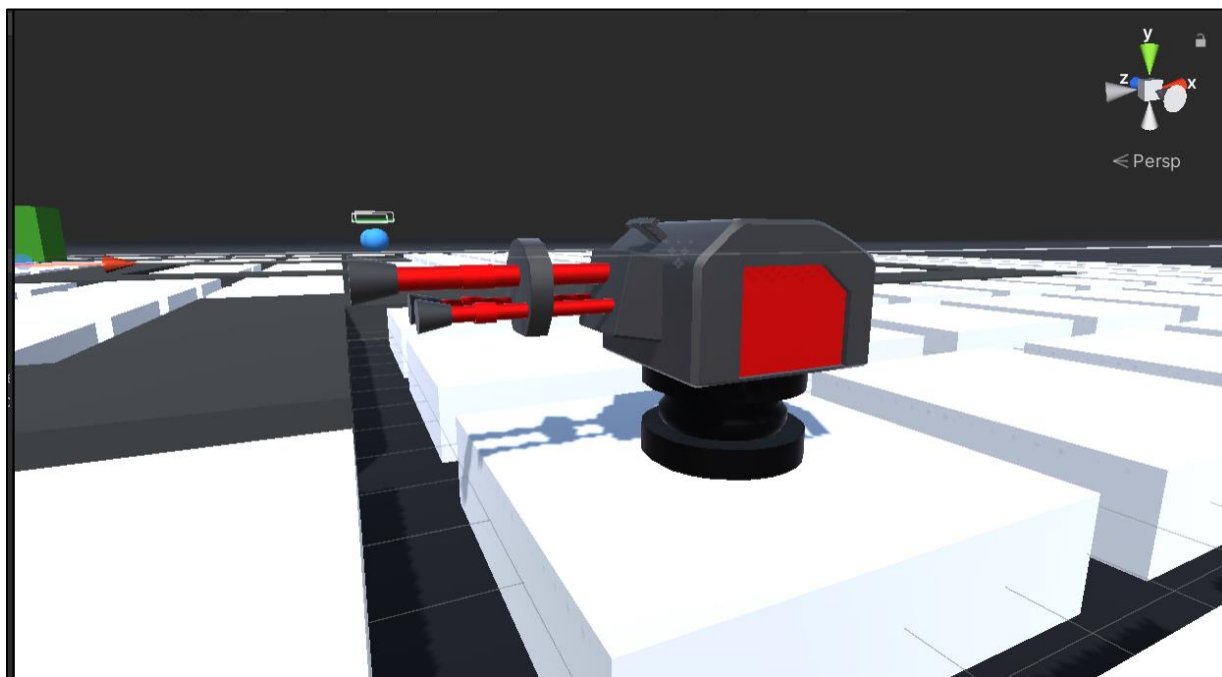


Рисунок 23 – *StandardTurret_Upgraded*

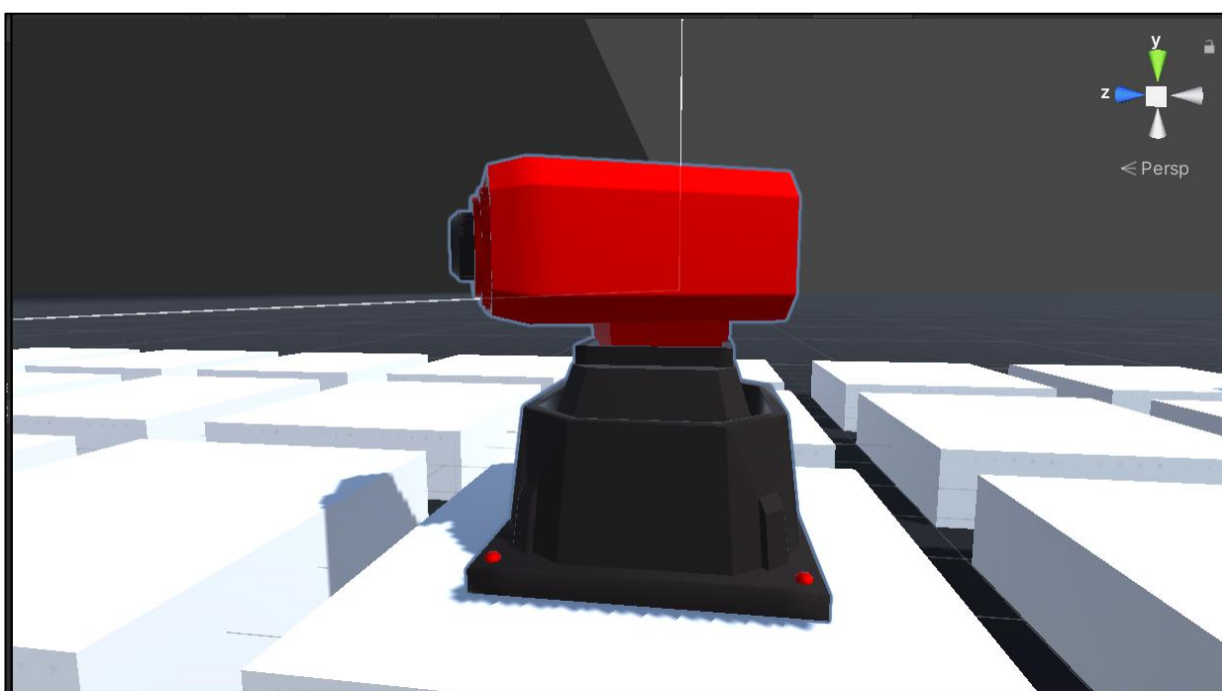


Рисунок 24 – *MissileLauncher_Upgraded*

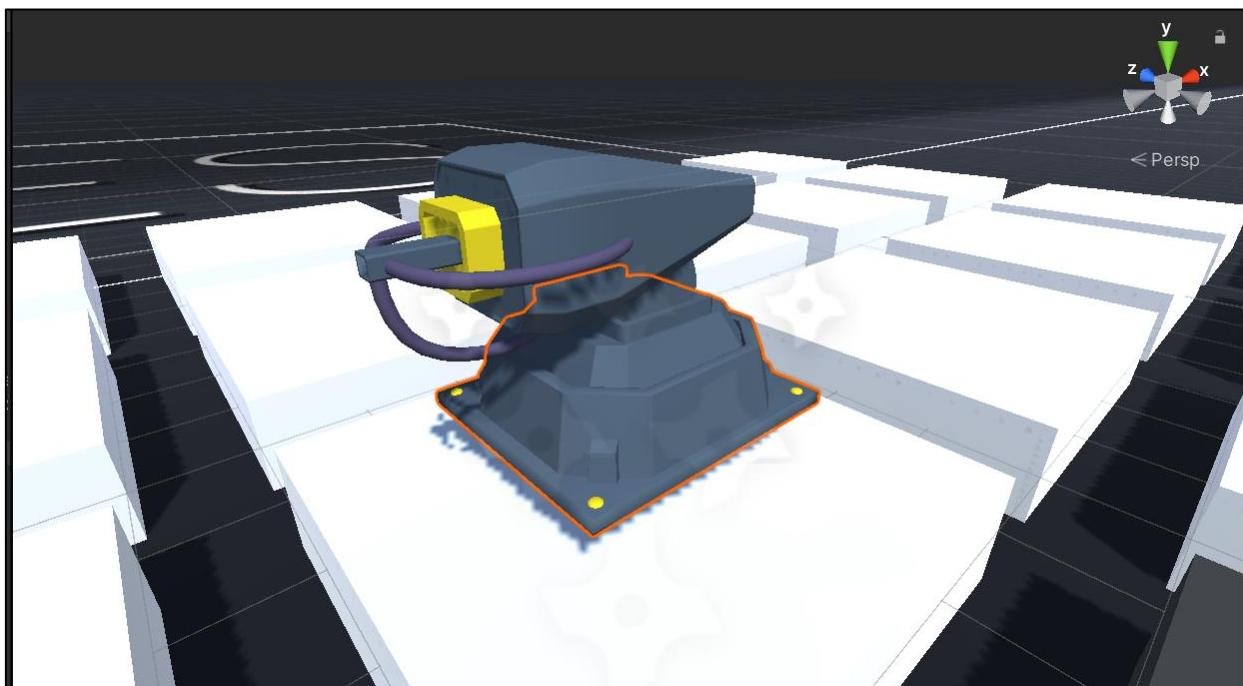


Рисунок 25 – *LaserBeamer_Upgraded*

Чтобы установить объект на игровое поле необходимо выбрать иконку башни в меню покупки и навестись на одну из доступных ячеек игрового поля. В случае если ячейка занята или недоступна она подсвечивается красным цветом. Следующие классы отвечают за выбор и установку башен:

Класс *Node.cs* – отвечает за игровое поле и игровые возможности взаимодействия с ним. Поля класса *Node.cs* представлены в таблице 9.

Таблица 9 – Описание полей класса *Node.cs*

Тип данных	Название	Описание
<i>Color</i>	<i>hoverColor</i>	Объект типа <i>Color</i> , который устанавливает цвет при наведении на ячейку
<i>Color</i>	<i>notEnoughMoneyColor</i>	Объект типа <i>Color</i> , который устанавливает цвет при отсутствии валюты и наведении на ячейку
<i>Color</i>	<i>startColor</i>	Объект типа <i>Color</i> , который устанавливает начальный цвет

Тип данных	Название	Описание
<i>Renderer</i>	<i>rend</i>	Объект типа <i>Renderer</i>
<i>GameObject</i>	<i>turret</i>	Экземпляр объекта башни
<i>TurretBlueprint</i>	<i>turretBlueprint</i>	Объект класса <i>TurretBlueprint</i>
<i>bool</i>	<i>isUpgraded</i>	Значение, которое проверяет можно ли улучшить башню
<i>Vector3</i>	<i>positionOffset</i>	Объект типа <i>Vector3</i> , который возвращает смещение
<i>BuildManager</i>	<i>buildManager</i>	Объект класса <i>GameManager</i>

Класс *Node.cs* включает в себя 8 методов. Рассмотрим основные методы данного класса.

Private void Start() – начальный метод, устанавливающий стартовые значения цвета поля, создает объект для ячейки.

Public Vector3 GetBuildPosition() – метод, возвращающий значение позиции и смещения ячейки.

Private void OnMouseEnter() – метод выбора ячейки. Проверяет можно ли установить башню на выбранную ячейки, устанавливает цвета при наведении на ячейку.

Private void OnMouseExit() – метод, отвечающий за изменение цвета при отведении курсора с ячейки.

Private void OnMouseDown() – метод, принимающий нажатие на ячейку для установки башни.

Private void BuildTurret() – метод, создающий объект башни на ячейке, добавление эффекта при строительстве, изменение валюты игрока.

Public void UpgradeTurret() – метод, предназначенный для улучшения башни, создания объекта улучшенной башни, удаления старого объекта.

Public void SellTurret() – метод, который применяет эффект уничтожения башни, уничтожает объект, добавляет значение к валюте после продажи.

Класс *NodeUI.cs* – отвечает за отображение вспомогательного графического интерфейса. Поля для класса *NodeUI.cs* представлены в таблице 10.

Таблица 10 – Описание полей класса *NodeUI.cs*

Тип данных	Название	Описание
<i>GameObject</i>	<i>ui</i>	Объект графического интерфейса
<i>Node</i>	<i>target</i>	Объект башни, который будет выделен для взаимодействия
<i>Text</i>	<i>upgradeCost</i>	Объект типа <i>Text</i> , содержащий значение улучшения
<i>Button</i>	<i>upgradeButton</i>	Объект типа <i>Button</i> - кнопка, состоящая из панелей для текста
<i>Text</i>	<i>sellAmount</i>	Объект типа <i>Text</i> , содержащий значение продажи

Класс *NodeUI.cs* включает в себя 4 метода, описанных ниже.

Public void SetTarget() – метод выбора установленной башни, устанавливает необходимые значения цены для улучшения и продажи. Содержит эффект появления графического интерфейса.

Public void Hide() – метод, который убирает графический интерфейс при выборе другого действия.

Public void Upgrade() – метод, содержащий графический интерфейс, для улучшения башни.

Public void Sell() – метод, содержащий графический интерфейс, для продажи башни.

Класс *BuildManager.cs* – отвечает за экземпляры стандартных и улучшенных башен. Проверяет можно ли установить башни в указанные поля и имеется ли такая возможность у игрока. Поля класса *BuildManager.cs* представлены в таблице 11.

Таблица 11 – Описание полей класса *BuildManager.cs*

Тип данных	Название	Описание
<i>TurretBlueprint</i>	<i>turretBlueprint</i>	Объект класса <i>TurretBlueprint</i>
<i>Node</i>	<i>selectedNode</i>	Объект класса <i>Node</i>
<i>NodeUI</i>	<i>nodeUI</i>	Объект класса <i>NodeUI</i>
<i>GameObject</i>	<i>buildEffect</i>	Графический эффект при строительстве башни
<i>GameObject</i>	<i>sellEffect</i>	Графический эффект при продаже башни
<i>BuildManager</i>	<i>instance</i>	Объект класса <i>BuildManager</i>

Класс *BuildManager.cs* включает в себя 7 методов.

Private void Awake() – метод проверки, определяет существует ли объект башни.

Public bool CanBuild() – метод, проверяющий можно ли установить башню.

Public bool HasMoney() – метод, проверяющий имеет ли игрок деньги для покупки башни.

Public void SelectTurretToBuild() – метод, который выделяет выбранный объект башни.

Public void SelectedNode() – метод, который выделяет выбранную ячейку.

Public void DeselectNode() – метод, который убирает выделение с выбранного объекта.

Public TurretBlueprint GetTurretToBuild() – возвращает объект башни.

1.2.5 Реализация интерфейса пользователя

Интерфейс пользователя состоит из нескольких панелей, каждая из которых отвечает за свой функционал и реагирует на события, происходящие в игровом процессе.

Для управления магазином описан класс *Shop.cs*. Данный класс содержит поля каждого из типа башен (рисунок 26). В окне *Inspector* (рисунок 27) устанавливаются экземпляры башен, их цена, цена для улучшенной версии и экземпляр улучшенной башни. Поля класса *Shop.cs* представлены в таблице 12.

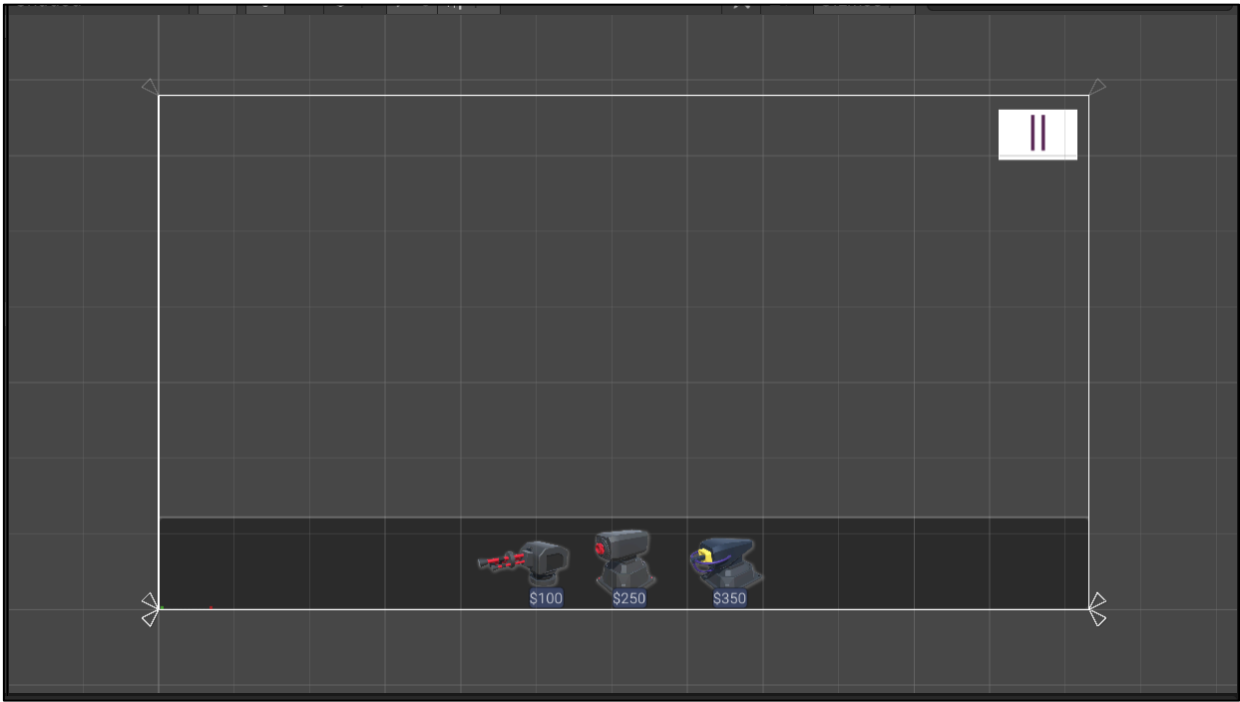


Рисунок 26 – Панель магазина

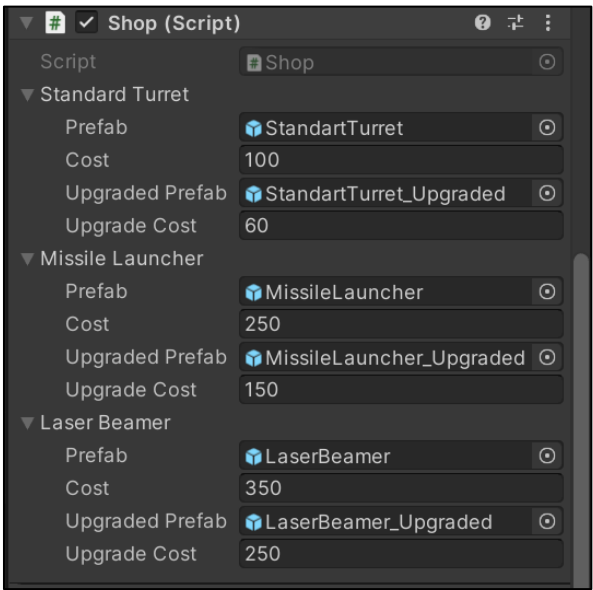


Рисунок 27 – Значения магазина

Таблица 12 – Описание полей класса *Shop.cs*

Тип данных	Название	Описание
<i>BuildManager</i>	<i>buildManager</i>	Объект класса <i>buildManager</i>
<i>TurretBlueprint</i>	<i>standartTurret</i>	Объект класса <i>TurretBlueprint</i>
<i>TurretBlueprint</i>	<i>missileLauncher</i>	Объект класса <i>TurretBlueprint</i>
<i>TurretBlueprint</i>	<i>laserBeamer</i>	Объект класса <i>TurretBlueprint</i>

Класс *Shop.cs* включает в себя 3 метода, представленные ниже.

Public void SelectStandartTurret() – выбирает и создает экземпляр данной башни;

Public void SelectMissileLauncher() – выбирает и создает экземпляр данной башни;

Public void SelectLaserBeamer() – выбирает и создает экземпляр данной башни.

Класс *GameManager.cs*. Поля класса *GameManager.cs* представлены в таблице 13.

Таблица 13 – Описание полей класса *GameManager.cs*

Тип данных	Название	Описание
<i>bool</i>	<i>GameIsOver</i>	Проверка на конец игры
<i>GameObject</i>	<i>gameOverUI</i>	Графический интерфейс конца игры
<i>GameObject</i>	<i>completeLevelUI</i>	Графический интерфейс победного уровня

Класс *GameManager.cs* включает в себя 4 метода. Их описание приведено ниже.

Private void Start() – устанавливает значение *GameisOver* = *false*.

Private void Update() – проверяет закончилась ли игра, имеются ли жизни у игрока.

Private void EndGame() – отображает графический интерфейс конца игры. Устанавливает значение *GameIsOver = true*.

Public void WinLevel() – отображает графический интерфейс победы. Устанавливает значение *GameIsOver = true*.

Меню паузы имеет 3 активные кнопки: *Continue* – выходит из меню паузы и возобновляет игру, *Retry* – начинает новую игру с текущим уровнем, *Menu* – запускает новую сцену, переходит в основное меню. Меню паузы представлено на рисунке 28.

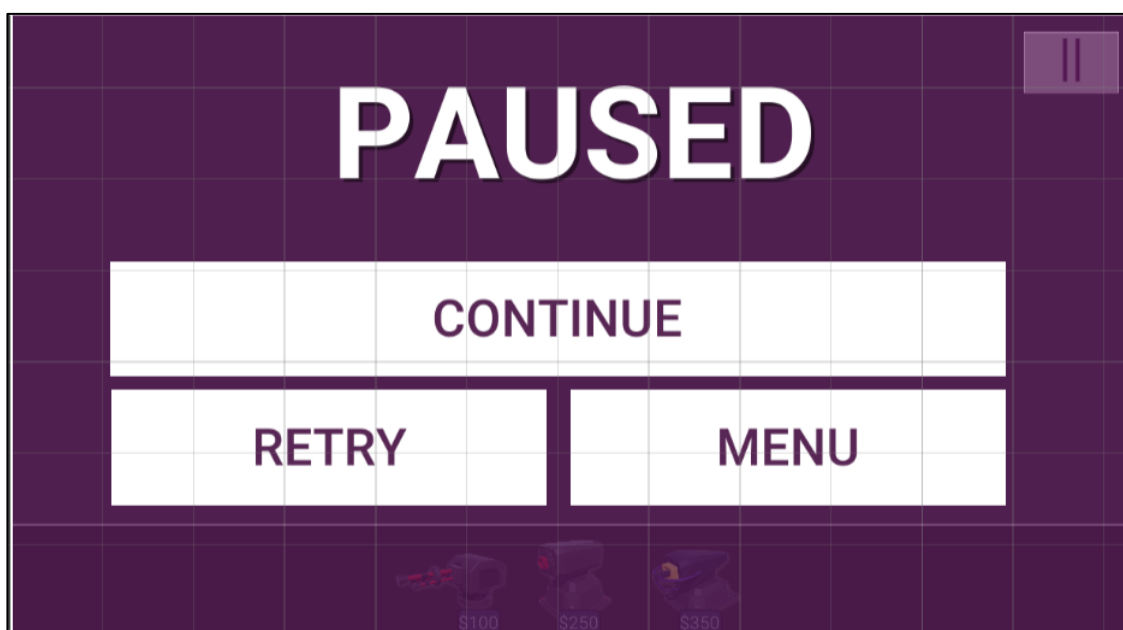


Рисунок 28 – Меню паузы

За отображение меню паузы и взаимодействия с ним отвечает класс *PauseMenu.cs*. Поля класса *PauseMenu.cs* представлены в таблице 13.

Таблица 13 – Описание полей класса *PauseMenu.cs*

Тип данных	Название	Описание
<i>GameObject</i>	<i>ui</i>	Объект графического интерфейса
<i>SceneFader</i>	<i>sceneFader</i>	Объект затемнения экрана
<i>string</i>	<i>menuSceneName</i>	Строка с именем главного меню

Класс *PauseMenu.cs* включает в себя 3 основных метода.

Private void Toggle() – активирует графический интерфейс паузы.

Public void Retry() – активирует новую сцену, новый уровень.

Public void Menu() – активирует новую сцену, основное меню.

Меню выигрышного уровня содержит запись количества пройденных волн и имеет 2 кнопки: *Continue* – выходит из меню паузы, возобновляет игру, *Menu* – запускает новую сцену, переходит в основное меню. Меню конца уровня представлено на рисунке 29.

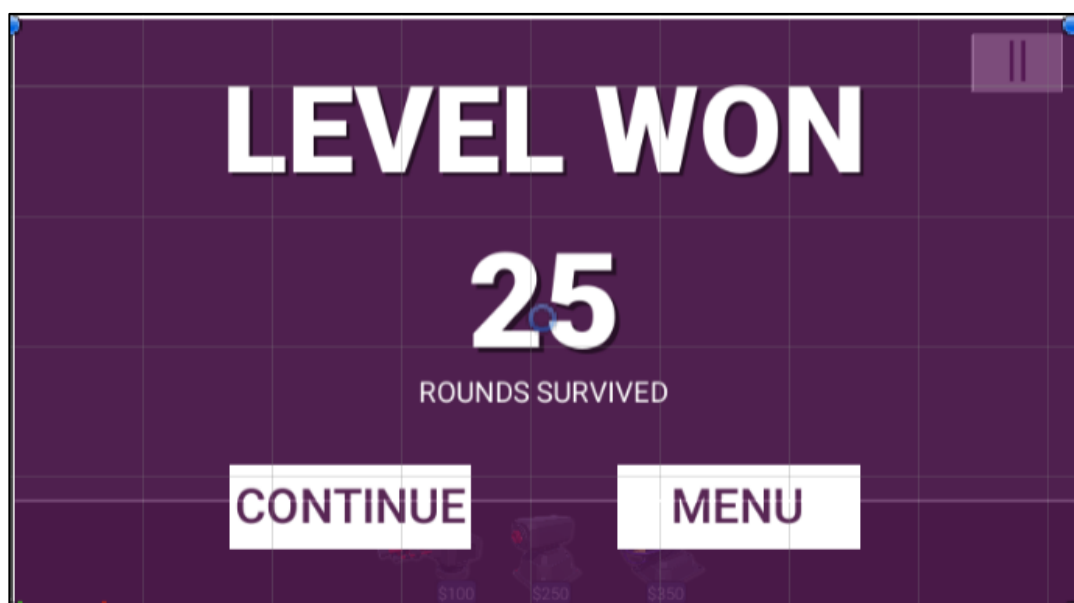


Рисунок 29 – Меню конца уровня

За отображение меню победы и взаимодействия с ним отвечает класс *CompleteLevel.cs*. Поля класса *CompleteLevel.cs* представлены в таблице 14.

Таблица 14 – Описание полей класса *CompleteLevel.cs*

Тип данных	Название	Описание
<i>SceneFader</i>	<i>SceneFader</i>	Объект затемнения экрана
<i>string</i>	<i>menuSceneName</i>	Строка с именем главного меню
<i>string</i>	<i>nextLevel</i>	Строка с именем следующего уровня

Тип данных	Название	Описание
<i>int</i>	<i>levelToUnlock</i>	Номер следующего уровня

Класс *CompleteLevel.cs* включает в себя 2 метода.

Public void Continue() – активирует новую сцену, следующий уровень;

Public void Menu() – активирует новую сцену, основное меню.

Меню конца игры содержит запись количества пройденных волн и имеет 2 кнопки: *Retry* – выходит из меню паузы, начинает новую игру, *Menu* – запускает новую сцену, переходит в основное меню. Меню поражения представлено на рисунке 30.

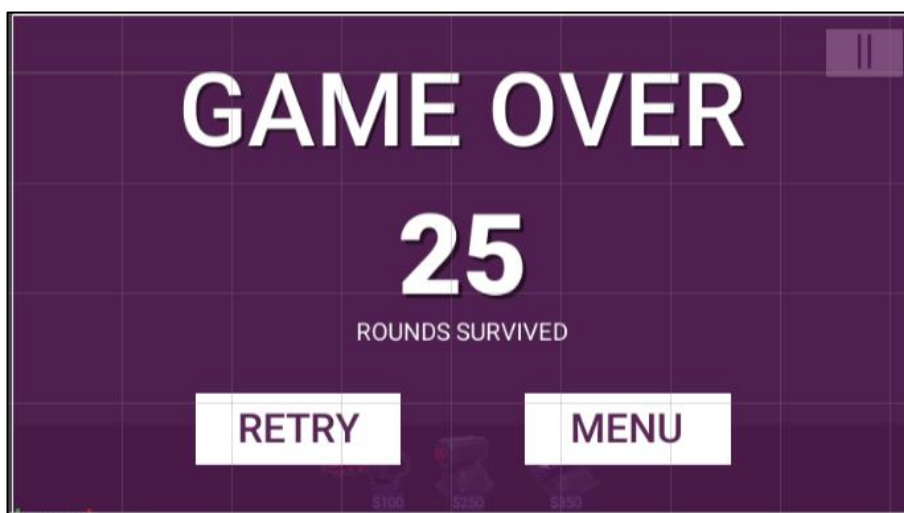


Рисунок 30 – Меню поражения

За отображение меню конца игры и взаимодействия с ним отвечает класс *GameOver.cs*. Поля класса *GameOver.cs* представлены в таблице 15.

Таблица 15 – Описание полей класса *GameOver.cs*

Тип данных	Название	Описание
<i>SceneFader</i>	<i>sceneFader</i>	Объект затемнения экрана
<i>string</i>	<i>menuSceneName</i>	Строка с именем главного меню

Класс *GameOver.cs* включает в себя 2 метода.

Public void Retry() – активирует новую сцену, новый уровень;

Public void Menu() – активирует новую сцену, основное меню.

Количество пройденных волн записывается в отдельном классе и отображается на панелях при победе и поражении игрока – *RoundsSurvived.cs*. Данный класс содержит метод *IEnumerator AnimateText()*, записывающий и отображающий текст с использованием анимации.

Главное меню игры является первой сценой и начальным экраном при запуске игры. Главное меню представлено на рисунке 31. Имеет 2 кнопки: *Play* – запускает новую сцену, *Quit* – выход из игры.



Рисунок 31 – Главное меню игры

Класс *MainMenu.cs*. Поля класса *MainMenu.cs* представлены в таблице 16.

Таблица 16 – Описание полей класса *MainMenu.cs*

Тип данных	Название	Описание
<i>SceneFader</i>	<i>sceneFader</i>	Объект затемнения экрана
<i>string</i>	<i>levelToLoad</i>	Строка с именем загрузки сцены

Класс *MainMenu.cs* включает в себя 2 метода.

Public void Play() – активирует новую сцену, выбор уровня;

Public void Quit() – метод выхода из игры.

Меню выбора уровня игры содержит кнопки для выбора уровня прохождения. Главное меню представлено на рисунке 32.

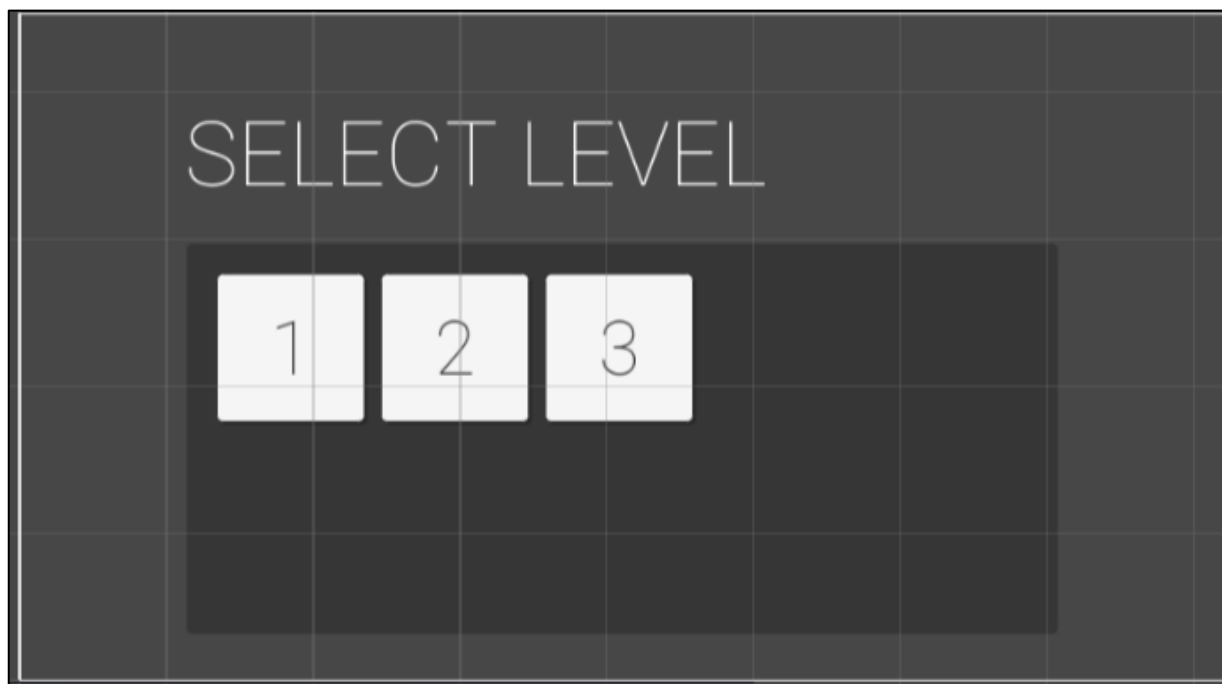


Рисунок 32 – Меню выбора уровня

Класс *LevelSelector.cs*. Поля класса *LevelSelector.cs* представлены в таблице 17.

Таблица 17 – Описание полей класса *LevelSelector.cs*

Тип данных	Название	Описание
<i>SceneFader</i>	<i>sceneFader</i>	Объект затемнения экрана
<i>Button[]</i>	<i>levelButton</i>	Массив объектов кнопки

Класс *LevelSelector.cs* включает в себя 2 метода.

Private void Start() – определяет нумерацию уровней;

Public void Select() – выбор уровня, переход к сцене уровня.

1.3 Анализ результатов

Было реализовано мобильное игровое приложение. Представлено описание игрового приложения, которое должно стать конечным продуктом данной работы. Проработана компьютерная графика и интерфейс пользователя приложения. Основным поддерживающим языком является язык *C#*, реализовано в среде разработки *Unity*. Перед разработкой программного кода была продумана логика игры, для этой цели использовалось схемное и блочное описание используемых классов и процедур. Был написан код, который использовался при проведении тестирования игрового приложения. На рисунке 34 представлена *UML*-диаграмма вариантов использования итогового приложения.

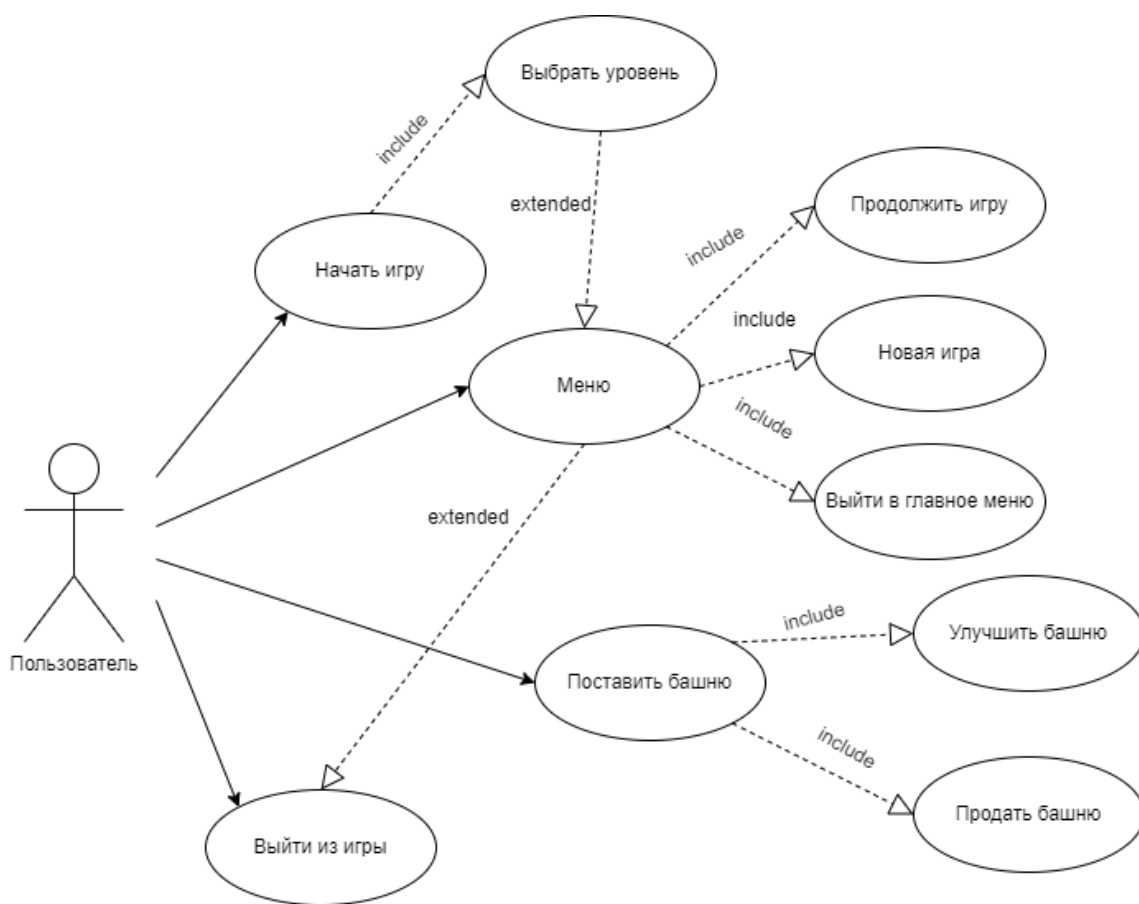


Рисунок 34 – *UML*-диаграмма вариантов использования

1.3.1 Тестирование приложения

Для проверки корректности работы игрового приложения оно было протестировано напрямую в системе *Unity*, через специальные asset модули. После этого был выполнен *build* проекта для платформы *Android*, сформировали *apk* файл, который можно загружать на мобильное устройство. На рисунке 35 представлена *UML*-диаграмма развертывания компонентов. В ходе проверки ошибок и недочетов выявлено не было.

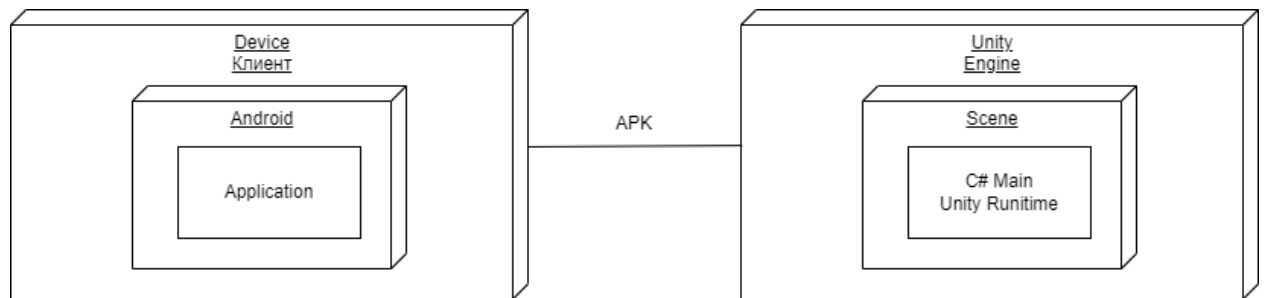


Рисунок 35 - *UML*-диаграмма развертывания компонентов

На рисунке 36 продемонстрировано главное меню игры. Оно состоит из двух кнопок: *Play* – начать игру, *Quit* – выйти из игры, закрыть приложение.



Рисунок 36 – Главное меню приложения

При нажатии *Play* происходит переход к сцене выбора уровня, который представлен на рисунке 37.

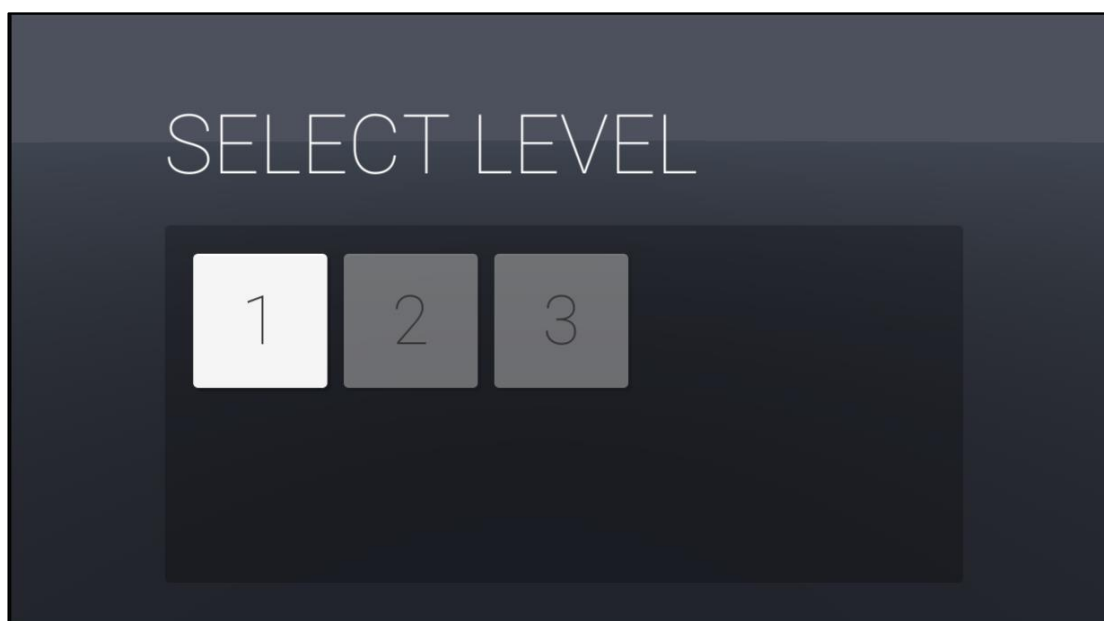


Рисунок 37 – Интерфейс меню выбора игрового уровня

Вид игрового интерфейса на устройстве представлен на рисунке 38. Он включает в себя игровое поле, снизу представлено меню выбора башни, количество игровой валюты, время до появления противников и количество жизней. Игровой процесс представлен на рисунке 39.

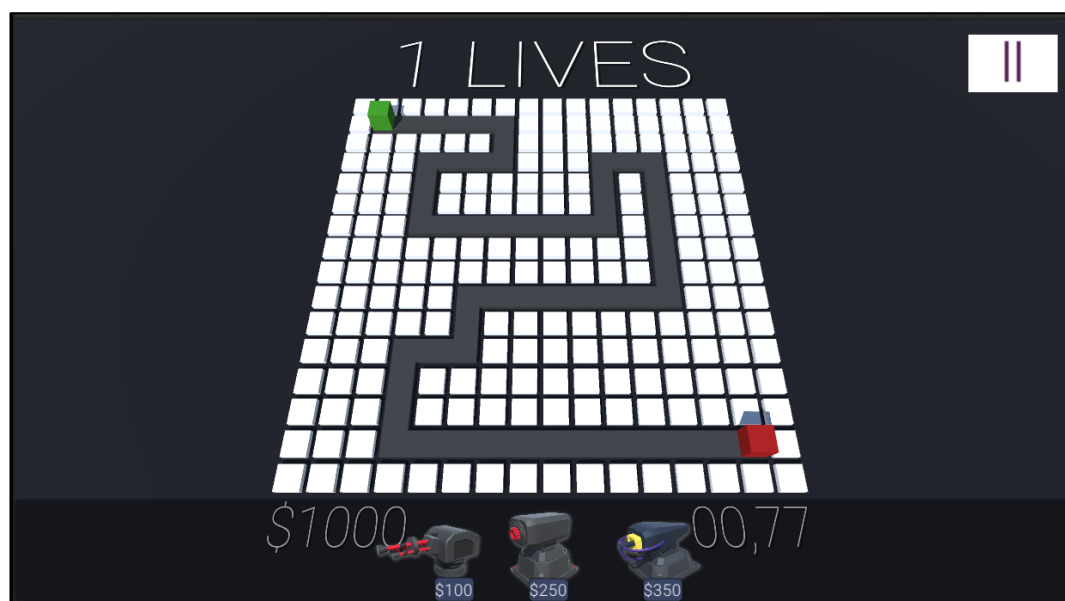


Рисунок 38 – Игровой интерфейс приложения

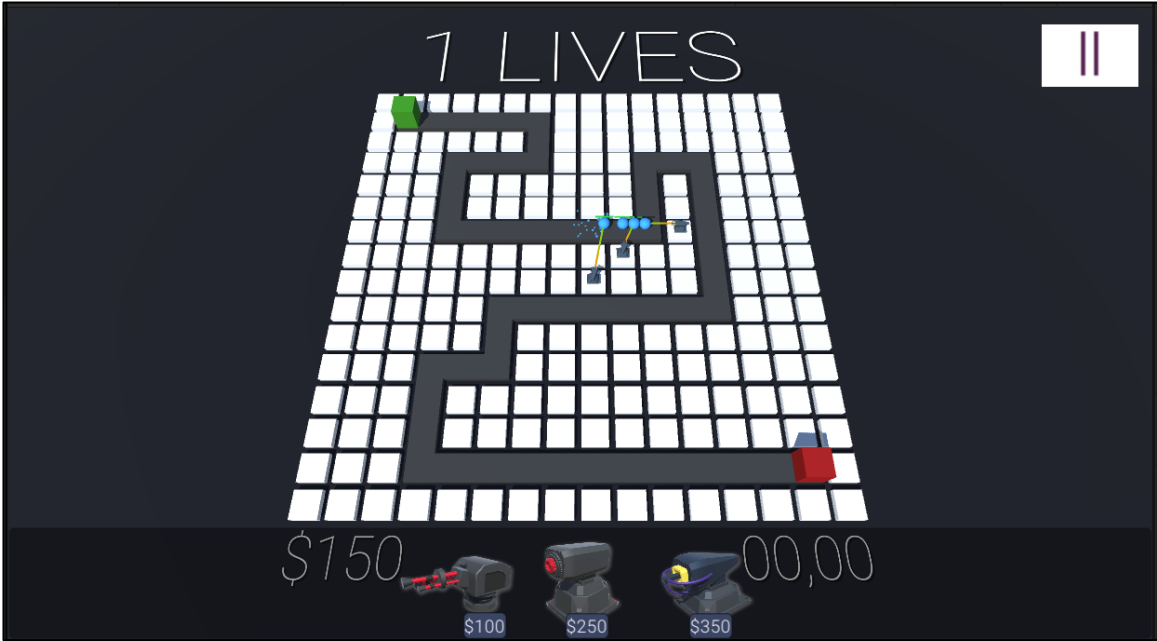


Рисунок 39 – Игровой процесс

На рисунке 40 представлено меню паузы, которое можно вызвать в любой момент времени. На нем отображено количество пройденных волн и три кнопки для взаимодействия: *Continue* – переход к следующему уровню, *Retry* – новая попытка для прохождения уровня, *Menu* – выход в главное меню.

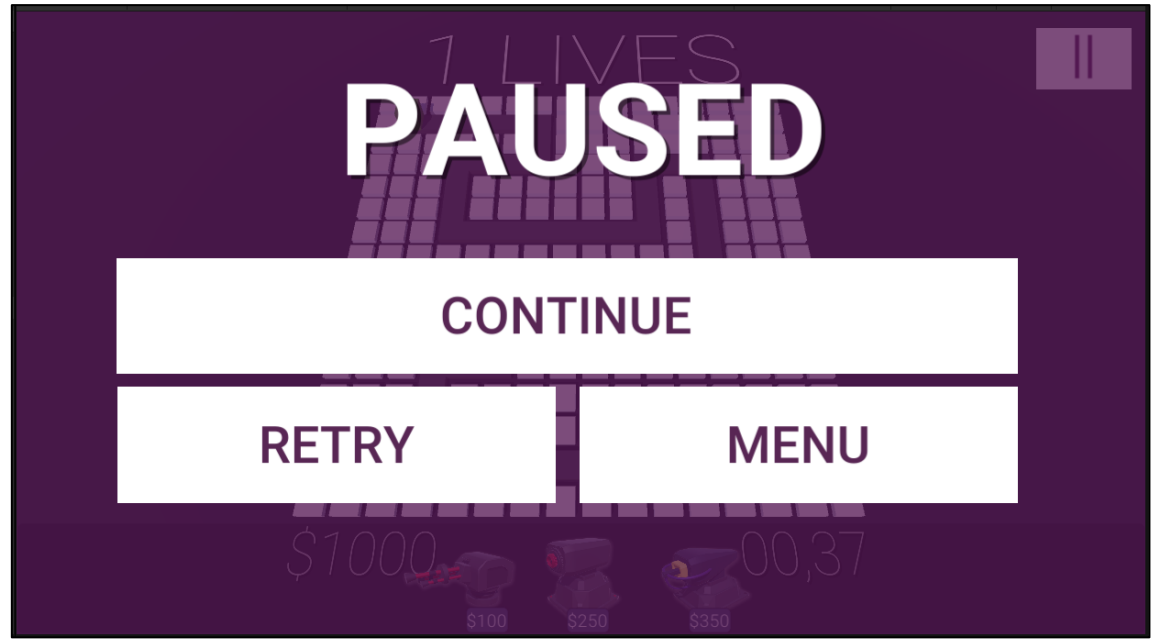


Рисунок 40 – Пауза

На рисунке 41 представлено меню, которое появляется при успешном прохождении уровня. На нем отображено количество пройденных волн и две кнопки для взаимодействия: *Continue* – переход к следующему уровню, *Menu* – выход в главное меню.

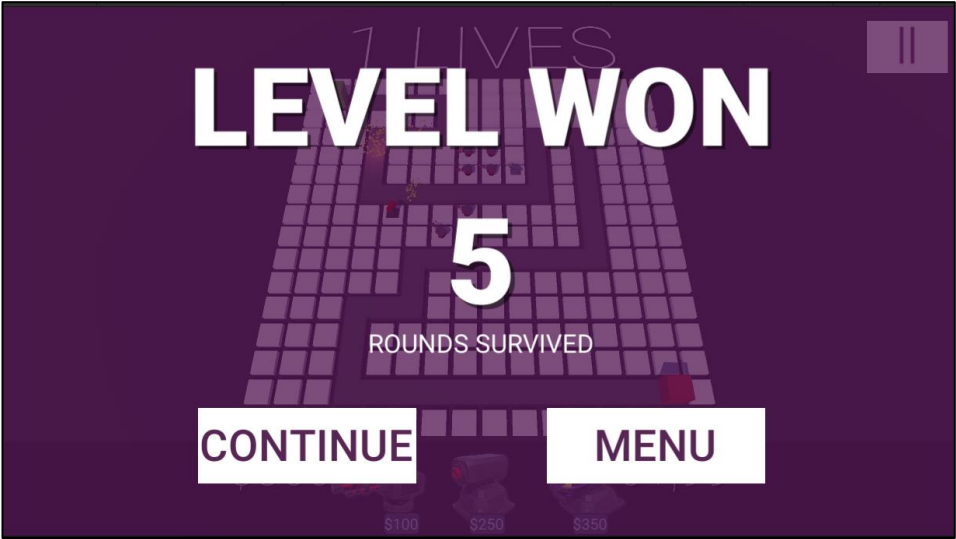


Рисунок 41 – Меню при прохождении уровня

На рисунке 42 представлено меню, которое появляется при успешном прохождении уровня. На нем отображено количество пройденных волн и две кнопки для взаимодействия: *Retry* – новая попытка для прохождения уровня, *Menu* – выход в главное меню.

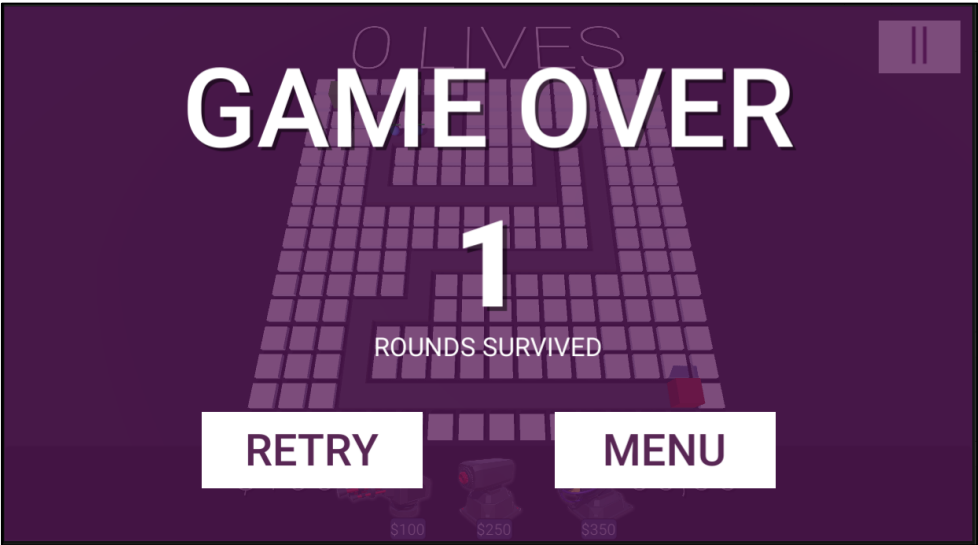


Рисунок 42 – Конец игры

Использование игрового приложения «*Tower Defense*» на виртуальных и физических мобильных устройствах не выявило ошибок и проблем во время работы приложения. Тесты показали, что приложение работает корректно и может использоваться игроком.

1.3.2 Преимущества и недостатки

Приложение разрабатывалось в межплатформенной среде *Unity*, таким образом данный проект можно адаптировать для любой другой доступной платформы.

К недостаткам игрового приложения следует отнести простоту, недостаточные функциональные возможности приложения, отсутствие широкого контента.

Современные средства разработки и концепции программирования позволяют быстро адаптировать и расширить исходный код для новой платформы. Однако данная реализация требует серьезного подхода и высококвалифицированной команды разработчиков. Разработка проработка и оптимизация объектов выходит за рамки одной работы.

2 Безопасность жизнедеятельности

В настоящий момент во всем мире широкое применение во многих сферах деятельности получили персональные электронно-вычислительные машины (ПЭВМ) и видеодисплейные терминалы (ВДТ). С ними, так или иначе, сталкивается все большее количество людей, которые решают при помощи ПЭВМ самые разнообразные задачи - от вычислительных до чисто коммуникационных.

Современная ПЭВМ интегрирует в себе функции мощного программного калькулятора, «интеллектуальной» пишущей машинки, узла связи, увлекательной игрушки, аудио-видео центра.

Между тем, как показывают исследования гигиенистов, длительная работа на ПЭВМ и ВДТ может привести к серьезным проблемам со здоровьем и профессиональным заболеваниям.

2.1 Вредоносные факторы

Обеспечение безопасной жизнедеятельности программиста в значительной степени зависит от правильной оценки опасных, вредных производственных факторов. В перечень основных вредных и опасных производственных факторов во время трудового процесса входят:

- высокая температура, характерная для отдельных элементов компьютерной техники, влияющая на общий температурный фон в рабочем помещении;
- повышенная или пониженная влажность воздуха;
- значительный уровень зрительной нагрузки, испытываемых работником;
- превышающий уровень шума;
- риск поражения электрическим током при случайном контакте с отдельными элементами техники;
- повышенный уровень ионизирующего излучения;

- повышенный уровень электромагнитных полей;
- повышенный уровень статического электричества;
- недостаточная освещенность рабочей зоны у компьютера;
- прямая и отраженная блеклость экрана дисплея, негативно влияющий на зрение работника в длительной перспективе.

В результате многочисленных исследований особенностей и условий труда пользователей ПЭВМ и ВДТ специалисты по гигиене труда выявили следующие сопутствующие работе вредные психофизиологические факторы: монотонность труда, умственное перенапряжение, перенапряжение зрительного и слухового анализаторов, повышенные нервно-эмоциональные перегрузки.

Действия указанных вредоносных факторов приводит к снижению работоспособности программиста, вызванное развивающимся утомлением.

2.2 Классификация и категорирование помещения

По взрывопожарной и пожарной опасности помещения подразделяются на категории А, Б, В1-В4, Г и Д, а здания - на категории А, Б, В, Г и Д. Категории и классы выбираются для каждого конкретного случая, в зависимости от характеристик веществ и материалов, находящихся в помещении; от значения удельной пожарной нагрузки; от наличия или отсутствия автоматизированной системы пожаротушения и т. д. [9].

Ниже приведены категории помещения и характеристик веществ и материалов, находящихся в помещении, по взрывопожарной и пожарной опасности:

А — повышенная взрывопожароопасность. Горючие газы, легковоспламеняющиеся жидкости с температурой вспышки не более 28 °С в таком количестве, что могут образовывать взрывоопасные парогазовоздушные смеси, при воспламенении которых развивается расчетное избыточное давление взрыва в помещении, превышающее 5 килопаскалей, и (или) вещества и материалы, способные взрываться и гореть

при взаимодействии с водой, кислородом воздуха или друг с другом, в таком количестве, что расчетное избыточное давление взрыва в помещении превышает 5 кПа.

Б — взрывопожароопасность. Горючие пыли или волокна, легковоспламеняющиеся жидкости с температурой вспышки более 28 °С, горючие жидкости в таком количестве, что могут образовывать взрывоопасные пылевоздушные или паровоздушные смеси, при воспламенении которых развивается расчетное избыточное давление взрыва в помещении, превышающее 5 кПа.

В1 — В4 — пожароопасность. Горючие и трудногорючие жидкости, твердые горючие и трудногорючие вещества и материалы (в том числе пыли и волокна), вещества и материалы, способные при взаимодействии с водой, кислородом воздуха или друг с другом только гореть, при условии, что помещения, в которых они находятся (обращаются), не относятся к категории А или Б.

Г — умеренная пожароопасность. Негорючие вещества и материалы в горячем, раскаленном или расплавленном состоянии, процесс обработки которых сопровождается выделением лучистого тепла, искр и пламени, и (или) горючие газы, жидкости и твердые вещества, которые сжигаются или утилизируются в качестве топлива.

Д — пониженная пожароопасность. Негорючие вещества и материалы в холодном состоянии.

Согласно Нормам пожарной безопасности НПБ 105-03, помещения с ЭВМ и ПЭВМ относятся к категории В (пожароопасные).

2.3 Требования к помещению

В производственных помещениях, в которых работа с использованием компьютеров является основной, должны обеспечиваться оптимальные параметры микроклимата. Конкретные показатели микроклимата на рабочем месте приведены в таблице 18:

					ПГУ 1.090301.21.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		59

Таблица 18 - Оптимальные нормы показателей микроклимата в рабочей зоне помещений с ПЭВМ и ВДТ

Период года	Категория работ	Температура воздуха, °С	Относительная влажность воздуха, %	Скорость движения воздуха, м/с
Холодный	Легкая – 1а	22 – 24	40 – 60	Не более 0,1
	Легкая – 1б	21 – 23	40 – 60	Не более 0,1
Теплый	Легкая – 1а	23 – 25	40 – 60	Не более 0,1
	Легкая – 1б	22 – 24	40 – 60	Не более 0,1

Офис является помещением 1а категории, где выполняются легкие физические работы, поэтому соблюдаются следующие требования: средняя температура воздуха в помещении офиса составляет +22°С, относительная влажность – 47%, атмосферное давление – 750 мм.рт.ст., содержание пыли – не более 10 мг/м воздуха рабочего места, максимальные размеры частиц – 2 мкм, скорость движения воздуха до 0,1м/с.

Для поддержания оптимального микроклимата, независимо от наружных условий: в холодное время года используется водяное отопление, в теплое время года применяется кондиционирование воздуха. Температура воздуха в помещении офиса не должна опускаться ниже +19°С, и не должна превышать +25°С.

Помещения должны находиться вдали от силовых кабелей, высоковольтных трансформаторов и другого оборудования, способного создавать помехи для офисной техники. В комнатах необходимо организовать защитное заземление, обеспечивающее безопасную работу компьютеров. Звукоизоляция помещений должна отвечать гигиеническим требованиям и обеспечивать нормируемые параметры шума. Поверхность пола должна быть ровной, удобной для очистки и влажной уборки, обладать антистатическими свойствами.

В помещениях с ПЭВМ необходимо проводить ежедневную влажную уборку, а также систематическое проветривание после каждого часа работы.

2.4 Требования к освещению

Недостаточность освещения приводит к напряжению зрения, ослабляет внимание, приводит к наступлению преждевременной утомленности. Чрезмерно яркое освещение вызывает ослепление, раздражение и резь в глазах. Неправильное направление света на рабочем месте может создавать резкие тени, блики, дезориентировать работающего. Все эти причины могут привести к несчастному случаю или профзаболеваниям.

Для освещения помещений и рабочих мест с ПЭВМ и ВДТ должно применяться естественное, искусственное и совмещенное освещение.

Помещения, предназначенные для размещения рабочих мест пользователей персональных компьютеров, должны иметь естественное и искусственное освещение. Естественное освещение помещений, предназначенных для размещения рабочих мест, должно осуществляться через световые проемы, ориентированные преимущественно на север и северо-восток. По отношению к световым проемам должны располагаться так, чтобы естественный свет падал сбоку, преимущественно слева. В случае иной ориентации световых проемов, необходимо предусматривать эффективные средства регулирования интенсивности естественного освещения. Оконные проемы в помещениях, предназначенных для использования компьютеров, необходимо оборудовать устройствами регулирования интенсивности естественного освещения типа жалюзи, занавесей, внешних козырьков и др.

Поскольку естественного света недостаточно, практически всегда устанавливают искусственное освещение. Искусственное освещение в помещениях для эксплуатации ПЭВМ должно осуществляться системой общего равномерного освещения. Допускается применять светильники прямого свет с люминисцентными лампами. Допускается применение

светильников местного освещения с лампами накаливания. Светильники должны располагаться в виде сплошных или прерывистых линий сбоку от рабочих мест параллельно линии зрения пользователя при разном расположении компьютеров.

В производственных и административно-общественных помещениях в случаях преимущественной работы с документами следует применять системы комбинированного освещения, то есть к общему освещению дополнительно устанавливаются светильники местного освещения, предназначенные для освещения зоны расположения документов.

Освещенность на поверхности рабочего стола должна быть в пределах от 300 до 500 лк. Светильники местного освещения не должны быть слишком яркими и не должны создавать бликов на поверхности экрана, так как ее освещенность не должна превышать 300 лк. Кроме этого, настольные лампы должны иметь непросвечивающий отражатель с защитным углом не менее 40 градусов. Для обеспечения нормативных значений освещенности в помещениях следует проводить чистку стекол оконных проемов и светильников не реже двух раз в год и проводить своевременную замену перегоревших ламп.

Так как при работе на компьютере основная нагрузка ложится на глаза, поэтому большие требования предъявляются к видеотерминальным устройствам (экранам). Предпочтительным является плоский экран, позволяющий избежать наличие на нем ярких пятен за счет отражения световых потоков. Особенно важен цвет экрана. Он должен быть нейтральным. Допустимы ненасыщенные светло-зеленые, желто-зеленые, желто-оранжевые, желто-коричневые тона. О качестве экранов судят по отсутствию мерцания и постоянству яркости. При прямом контрасте (темные символы на светлом фоне) частота мельканий должна быть не менее 80Гц. Оптимальная высота расположения экрана должна соответствовать направлению взгляда оператора в секторе 5-35° по отношению к горизонтали. Большой наклон экрана может привести к появлению бликов от

светильников. При работе с ЭВМ взгляд должен падать на экран под прямым углом и отклоняться от горизонтали на 20°. Для устранения бликов и снижения влияния электромагнитного излучения экраны ВДТ должны быть снабжены защитными фильтрами.

2.5 Требования по организации рабочего места

Санитарными правилами регламентирована площадь на одно рабочее место:

- при использовании компьютеров с мониторами на базе электронно-лучевой трубки площадь должна быть не менее 6 кв. м;
- при использовании компьютеров с жидкокристаллическими или плазменными мониторами - не менее 4,5 кв. м.

Рабочие столы желательно размещать таким образом, чтобы видеодисплейные терминалы были ориентированы боковой стороной к световым проемам. Расставлять столы желательно так, чтобы естественный свет падал преимущественно слева.

При размещении рабочих мест с компьютерами расстояние между рабочими столами должно быть не менее 2 метров, а расстояние между боковыми поверхностями видеомониторов – не менее 1,2 метра. Если с помощью компьютеров выполняется творческая работа, требующая значительного умственного напряжения или высокой концентрации внимания, рабочие места рекомендуется изолировать друг от друга перегородками высотой 1,5 - 2 метра.

Конструкция рабочего стола должна обеспечивать оптимальное размещение на рабочей поверхности используемого оборудования с учетом его количества и конструктивных особенностей, характера выполняемой работы. При этом допускается использование рабочих столов различных конструкций, отвечающих современным требованиям эргономики. Оптимальными размерами поверхности рабочего стола для ПЭВМ являются: ширина – от 800 до 1400 мм, глубина – 800 и 1000 мм при нерегулируемой его высоте, равной 725 мм.

Монитор на столе желательно располагать на расстоянии 60-70 см от глаз пользователя, но не ближе 50 см с учетом размеров алфавитно-цифровых знаков и символов, клавиатуру – на расстоянии 100-300 мм от края стола, обращенного к пользователю, или на специальной регулируемой по высоте рабочей поверхности, отделенной от основной столешницы.

Стул должен обеспечивать поддержание рациональной рабочей позы, позволять изменять ее с целью снижения напряжения мышц спины и шейно-плечевой области. Более приемлемым является подъемно-поворотное кресло, которое регулируется по высоте и углам наклона спинки. Регулировка каждого параметра при этом должна быть независимой, легко осуществимой и иметь надежную фиксацию.

Рабочее место пользователя ПЭВМ предлагается оборудовать подставкой для ног, имеющей ширину не менее 300 мм, глубину не менее 400 мм, регулировку по высоте в пределах до 150 мм и по углу наклона опорной поверхности подставки до 20 градусов. Желательно, чтобы при этом поверхность подставки была рифленой и имела по переднему краю бортик высотой 10 мм.

2.6 Требования к организации работы

Организация работы с ПЭВМ осуществляется в зависимости от вида и категории трудовой деятельности [10]. Виды трудовой деятельности подразделяются на три группы:

- работа по считыванию информации с экрана монитора с предварительным запросом отнесена к группе А;
- работа по вводу информации – к группе Б;
- творческая работа в режиме диалога с ПЭВМ – к группе В.

При выполнении в течение рабочей смены работ, относящихся к разным видам трудовой деятельности, за основную работу с ПЭВМ следует принимать такую, которая занимает не менее 50% времени в течение рабочей смены или рабочего дня.

Для видов трудовой деятельности устанавливается три категории тяжести и напряженности работы с ПЭВМ. Максимальные значения соответствуют III категории работы с ПЭВМ, их значения определяются:

- для группы А – по суммарному числу считываемых знаков за рабочую смену, но не более 60 000 знаков за смену;
- для группы Б – по суммарному числу считываемых или вводимых знаков за рабочую смену, но не более 40 000 знаков за смену;
- для группы В – по суммарному времени непосредственной работы с ПЭВМ за рабочую смену, но не более 6 ч за смену.

Для предупреждения преждевременной утомляемости пользователей ПЭВМ рекомендуется организовывать рабочую смену путем чередования работ с использованием ПЭВМ и без него.

В случаях, когда характер работы требует постоянного взаимодействия с компьютером (набор текстов или ввод данных и т.п.) с напряжением внимания и сосредоточенности, при исключении возможности периодического переключения на другие виды трудовой деятельности, не связанные с ПЭВМ, предлагается осуществлять перерывы на 10-15 мин. через каждые 45-60 мин. работы. Продолжительность непрерывной работы без регламентированного перерыва не должна превышать 1 часа.

При работе с ПЭВМ в ночную смену (с 22:00 до 6:00) независимо от категории и вида трудовой деятельности продолжительность регламентированных перерывов желательно увеличивать на 30%.

2.7 Расчет освещения рабочего места оператора ПЭВМ

Целью расчета является определение числа и мощности светильников, необходимых для обеспечения освещенности, достаточной для работы персонала вычислительного центра (ВЦ). Тип источников света - газоразрядные (люминесцентные лампы низкого давления, имеющие форму цилиндрической трубки), светильники - прямого света. Система

освещения общая, так как она создает равномерное освещение по всему объему ВЦ.

Яркость светильников общего освещения в зоне углов излучения от 50 до 90 градусов с вертикалью в продольной и поперечной плоскостях должна составлять не более 200 кд/м², защитный угол светильников должен быть не менее 40 градусов.

Общее освещение следует выполнять в виде сплошных или прерывистых линий светильников, расположенных сбоку от рабочих мест, параллельно линии зрения пользователя при рядом расположении ПК и ВДТ.

Расчет системы освещения осуществляется методом коэффициента использования светового потока, который выражается отношением светового потока, падающего на расчетную поверхность, к суммарному потоку всех ламп. В помещении имеется два окна. Расположим светильники в два ряда параллельно длинной стороне помещения, имеющего размеры 8 х 4 м и высотой 3 м. Светильники в рядах расположены с зазором в 1,5 м, расстояние между рядами 1,5 м, установлены на потолке. Высота рабочих мест составляет 0,75 м, поэтому расчетная высота h (высота подвеса светильников над рабочей поверхностью) будет равна 2,25 м.

Искусственное освещение в помещениях с ПК следует осуществлять системой общего равномерного освещения. В соответствии со СНиП 23-05-93 освещенность на поверхности стола в зоне размещения рабочего документа от системы общего освещения должна быть 300-500 лк. В качестве источников света общего освещения следует применять преимущественно люминесцентные лампы мощностью 35-65 Вт типа ЛБ.

Световой поток группы ламп светильника найдем по следующей формуле (2.1):

$$\Phi_{\text{св}} = (E_{\text{н}} * S * k_{\text{з}} * Z) / (N * n), \quad (2.1)$$

где E_n - требуемый нормативный уровень освещенности рабочей поверхности. Возьмем $E_{норм} = 300$ лк – это наиболее оптимальное значение для данного помещения;

$k_3 = 1,5$ - коэффициент запаса, учитывающий запыленность светильников и износ люминесцентных ламп в процессе эксплуатации, при условии чистки светильников не реже 4-х раз в год;

S – площадь помещения;

$Z = 1,1$ - коэффициент неравномерности освещения;

N — количество светильников;

η - коэффициент использования светового потока, выбирается из таблиц в зависимости от типа светильника, размеров помещения, коэффициентов отражения стен ρ_c и потолка ρ_n помещения, показателя помещения i ;

$\rho_n = 0.7$ (цвет поверхности - белый);

$\rho_c = 0.5$ (цвет поверхности - светлый);

Расчёт площади помещения находится по формуле (2.2):

$$S = A * B = 8 * 4 = 32 \text{ м}^2. \quad (2.2)$$

где a – длина;

b – ширина.

Количество светильников в помещении можно определить по следующей формуле (2.3):

$$N = S / h^2 = 32 * (2,25^2) = 6,3 \text{ шт.} \quad (2.3)$$

Поскольку светильники расположены в два ряда, то их число выбираем четным.

Показатель помещения можно определить по формуле (2.4):

$$i = (A * B) / ((A + B) * h) = (8 * 4) / ((8 + 4) * 2,25) = 1,18. \quad (2.4)$$

Тогда, на основании значений ρ_n , ρ_c и i по таблице выбираем $\eta = 0,42$.

Таким образом, подставляя полученные значения в формулу (2.1) можно рассчитать световой поток группы ламп светильника:

$$\Phi_{\text{св}} = \frac{300 * 32 * 1,5 * 1,1}{6 * 0,42} = 6743 \text{ лм.}$$

Учитывая, что светильник рассчитан на 4 лампы, получим:

$$\Phi_{\text{д}} = \frac{\Phi_{\text{св}}}{4} = 1686 \text{ лм.}$$

$\Phi_{\text{д}} = 1686$ лм - световой поток одной лампы.

По найденному значению светового потока можно определить тип и мощность лампы. Этому значению соответствует лампа ЛД40 мощностью 40 Вт со световым потоком 2100 лм. На практике допускается отклонение светового потока выбранной лампы от рассчитанного до $\pm 20\%$, т.е. лампа выбрана верно.

В системе освещения используется 24 лампы по 40 Вт каждая. Таким образом, общая потребляемая мощность:

$$P_0 = 24 * 40 = 960 \text{ Вт.}$$

Учитывая, что в таких лампах потери мощности могут составлять до 25 %, рассчитаем запас мощности:

$$P_p = 960 * 0,25 = 240 \text{ Вт.}$$

Тогда общая мощность сети должна быть:

$$P = P_0 + P_p = 960 + 240 = 1200 \text{ Вт.}$$

Таким образом, рассчитанная система общего освещения позволяет:

– обеспечить возможность нормальной деятельности людей в условиях отсутствия или недостаточности естественного освещения;

- обеспечить сохранность зрения;
- повысить производительность труда, безопасность работы;
- сохранить архитектурный облик помещения ВЦ.

					ПГУ 1.090301.21.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		69

Заключение

В ходе выполнения данной работы было создано стратегически-развивающее мобильное игровое приложение «*Tower Defense*».

В ходе выполнения проекта решался ряд задач, среди которых необходимо выделить следующие:

- анализ игровой индустрии;
- этапы разработки мобильного приложения;
- анализ существующих игровых классификаций;
- определение набора задач для разрабатываемого приложения;
- выбор средств разработки приложения;
- написание программного кода;
- создание графической части приложения;
- проектирование структурных схем функционирования приложения.

Были определены основные функции, которые должны быть реализованы в современном игровом приложении, что позволило перейти к реализации проектных решений. Приложение полностью соответствует всем заявленным требованиям.

В работе содержится программа, устанавливаемая на *Android*-устройство. В разработанном приложении реализованы актуальные технические решения. За время проектирования и реализации данного проекта была освоена межплатформенная среда разработки компьютерных игр *Unity*, улучшены навыки разработки приложений на языке программирования *C#*.

Игровое приложение было протестировано на мобильных устройствах, приложение работает корректно и может использоваться игроком.

Список использованных источников

1. Статья об истории развитии компьютерных игр. [Электронный ресурс] / Режим доступа: <http://cpu3d.com/histgame/statya-ob-istorii-razvitiyakompyuternyh>.
2. Глобальный обзор игровой индустрии: тренды, инсайды и прогнозы. [Электронный ресурс] / Режим доступа: <https://adindex.ru/news/researches/2019/01/25/230750.phtml>, свободный.
3. Рыто Майер *Android*. Программирование приложений для планшетных компьютеров и смартфонов — М.: Эксмо, 2011— 222 с
4. Мандел Тео. Разработка пользовательского интерфейса: Пер. с англ. — М: ДМК Пресс, 2004 — 416 с.
5. *Android для программистов: создаём приложения* [Текст] / П. Дейтел [и др.]. — СПб.: Питер, 2013. — 560 с.: ил. — ISBN 978-5-459-01646-8.
6. Джереми Гибсон Бонд. *Unity и C#*. Геймдев от идеи до реализации. Второе издание. СПб.: Питер, 2019. — 930 с.
7. Джозеф Хокинг. *Unity и C#*. *Unity* в действии. Мультиплатформенная разработка на *C#*. Второе международное издание. СПб.: Питер, 2019. — 352 с.
8. Джон Мэннинг, Пэрис Батфилд-Эддисон. *Unity* для разработчика. Мобильные мультиплатформенные игры. Второе международное издание. СПб.: Питер, 2019. — 352 с.
9. Нормы пожарной безопасности НПБ 105-03 "Определение категорий помещений, зданий и наружных установок по взрывопожарной и пожарной опасности" / утв. Приказом №314 МЧС РФ от 18 июня 2003 г.
10. СанПиН 2.2.2_2.4.1340-03 гигиенические требования к ПЭВМ и организации работы / Министерство здравоохранения РФ, 2003.

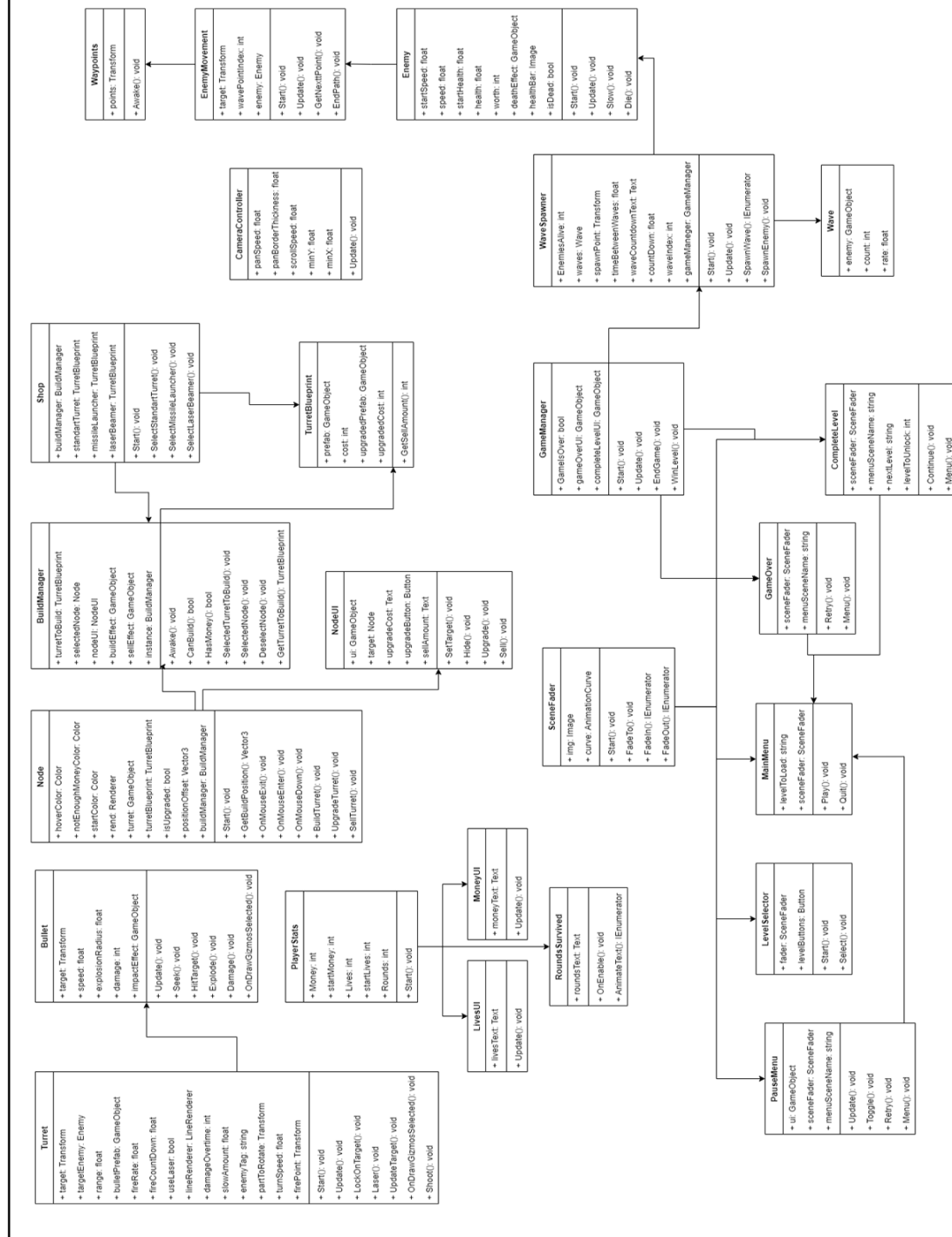
Приложение А

(Обязательное)

Мобильное приложение под ОС Андроид

UML-диаграмма классов

					ПГУ 1.090301.21.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		72

[illegible]

Изм.	Лист	№ докум.	Подпись	Дата

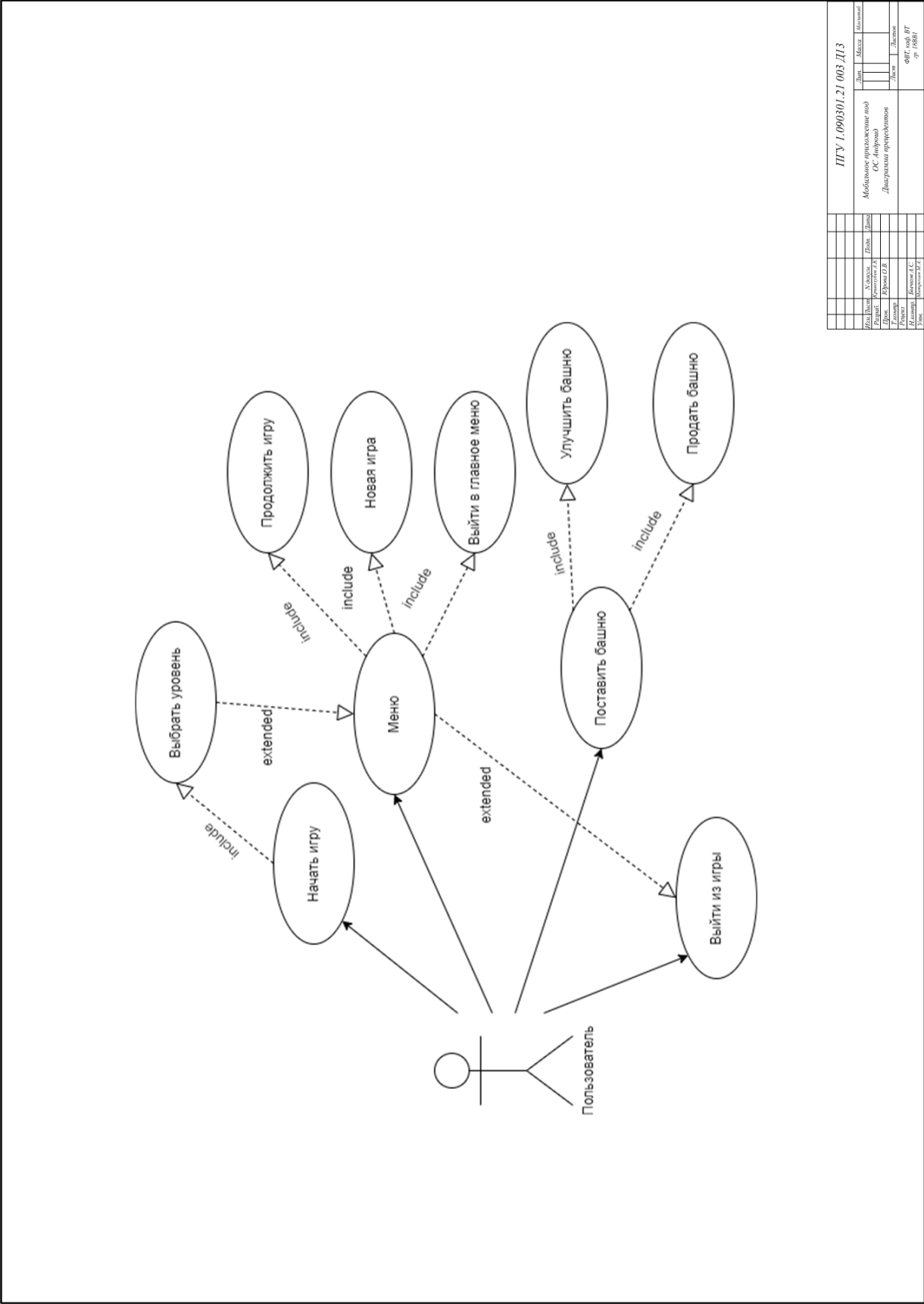
Приложение Б

(Обязательное)

Мобильное приложение под ОС Андроид

UML-диаграмма прецедентов

					ПГУ 1.090301.21.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		74



ПГУ 1.090301.21.003 Л13									
Мобильное приложение под ОС Android									
Диаграммы прецедентов									
Имя	Фамилия	Дата	Лист	Страница	Имя	Фамилия	Дата	Лист	Страница
Имя	Фамилия	Дата	Лист	Страница	Имя	Фамилия	Дата	Лист	Страница
Имя	Фамилия	Дата	Лист	Страница	Имя	Фамилия	Дата	Лист	Страница
Имя	Фамилия	Дата	Лист	Страница	Имя	Фамилия	Дата	Лист	Страница
Имя	Фамилия	Дата	Лист	Страница	Имя	Фамилия	Дата	Лист	Страница
Имя	Фамилия	Дата	Лист	Страница	Имя	Фамилия	Дата	Лист	Страница
Имя	Фамилия	Дата	Лист	Страница	Имя	Фамилия	Дата	Лист	Страница

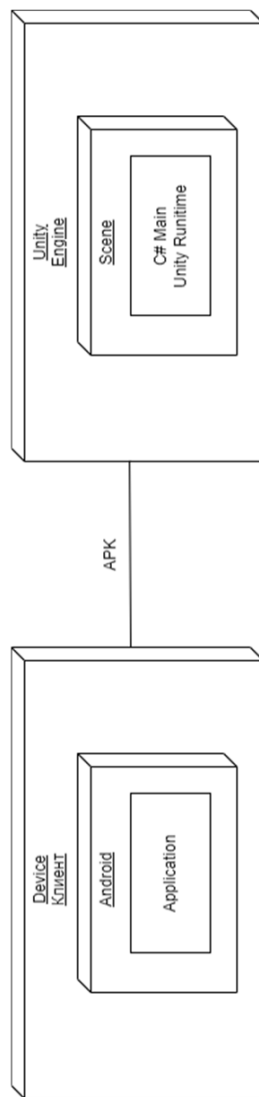
Приложение В

(Обязательное)

Мобильное приложение под ОС Андроид

UML-диаграмма развертывания

					ПГУ 1.090301.21.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		76



ПГУ 1.090301.21 004 П19											
Мобильное приложение под ОС Android						Дисциплина разработки					
Дан	Дан	Характер	Дата	Дата	Дата	Дан	Дан	Дата	Дата	Дата	Дата
Дан	Дан	Характер	Дата	Дата	Дата	Дан	Дан	Дата	Дата	Дата	Дата
Дан	Дан	Характер	Дата	Дата	Дата	Дан	Дан	Дата	Дата	Дата	Дата
Дан	Дан	Характер	Дата	Дата	Дата	Дан	Дан	Дата	Дата	Дата	Дата
Дан	Дан	Характер	Дата	Дата	Дата	Дан	Дан	Дата	Дата	Дата	Дата
Дан	Дан	Характер	Дата	Дата	Дата	Дан	Дан	Дата	Дата	Дата	Дата
Дан	Дан	Характер	Дата	Дата	Дата	Дан	Дан	Дата	Дата	Дата	Дата
Дан	Дан	Характер	Дата	Дата	Дата	Дан	Дан	Дата	Дата	Дата	Дата
Дан	Дан	Характер	Дата	Дата	Дата	Дан	Дан	Дата	Дата	Дата	Дата
Дан	Дан	Характер	Дата	Дата	Дата	Дан	Дан	Дата	Дата	Дата	Дата

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.090301.21.001 ПЗ

Приложение Г

(Обязательное)

Мобильное приложение под ОС Андроид

UML-диаграмма последовательностей

					ПГУ 1.090301.21.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		78

Приложение Д

(Обязательное)

Мобильное приложение под ОС Андроид

UML-диаграмма деятельности

					ПГУ 1.090301.21.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		80

Приложение Е

(Обязательное)

Мобильное приложение под ОС Андроид

Листинг программы

					ПГУ 1.090301.21.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		82

Файл *BuildManager.cs*

```
using UnityEngine;

public class BuildManager : MonoBehaviour
{
    private TurretBlueprint turretToBuild;
    private Node selectedNode;
    public NodeUI nodeUI;

    public GameObject standartTurretPrefab;
    public GameObject missileLauncherPrefab;
    public GameObject buildEffect;
    public GameObject sellEffect;
    public static BuildManager instance;

    void Awake()
    {
        if (instance)
        {
            return;
        }
        instance = this;
    }

    public bool CanBuild { get { return turretToBuild != null; } }
    public bool HasMoney { get { return PlayerStats.Money >= turretToBuild.cost; } }

    public void SelectTurretToBuild(TurretBlueprint turret)
    {
        turretToBuild = turret;
        DeselectNode();
    }

    public void SelectedNode(Node node)
    {
        if (selectedNode == node)
        {
            DeselectNode();
            return;
        }

        selectedNode = node;
        turretToBuild = null;
        nodeUI.SetTarget(node);
    }

    public void DeselectNode()
    {
        selectedNode = null;
    }
}
```

					ИГУ 1.090301.21.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		83

```

        nodeUI.Hide();
    }

    public TurretBlueprint GetTurretToBuild()
    {
        return turretToBuild;
    }
}

```

Файл *Bullet.cs*

```

using UnityEngine;

public class Bullet : MonoBehaviour
{
    private Transform target;
    public float speed = 70f;
    public float explosionRadius = 0f;
    public int damage = 50;
    public GameObject impactEffect;

    void Update()
    {
        if (target == null)
        {
            Destroy(gameObject);
            return;
        }

        Vector3 dir = target.position - transform.position;
        float distanceThisFrame = speed * Time.deltaTime;

        if (dir.magnitude <= distanceThisFrame)
        {
            HitTarget();
            return;
        }

        transform.Translate(dir.normalized * distanceThisFrame,
Space.World);
        transform.LookAt(target);
    }

    public void Seek(Transform _target)
    {
        target = _target;
    }

    void HitTarget()
    {
        GameObject effectIns =
        (GameObject)Instantiate(impactEffect, transform.position,
transform.rotation);
    }
}

```

```

        Destroy(effectIns, 5f);

        if (explosionRadius > 0)
        {
            Explode();
        }
        else
        {
            Damage(target);
        }

        Destroy(gameObject);
    }

    void Explode()
    {
        Collider[] colliders =
Physics.OverlapSphere(transform.position, explosionRadius);
        foreach (Collider collider in colliders)
        {
            if (collider.tag == "Enemy")
            {
                Damage(collider.transform);
            }
        }
    }

    void Damage(Transform enemy)
    {
        Enemy e = enemy.GetComponent<Enemy>();
        if (e != null)
        {
            e.TakeDamage(damage);
        }
    }

    void OnDrawGizmosSelected()
    {
        Gizmos.color = Color.red;
        Gizmos.DrawWireSphere(transform.position,
explosionRadius);
    }
}

```

Файл *CameraController.cs*

```

using System.Collections;
using System.Collections.Generic;
using UnityEngine;
public class CameraController : MonoBehaviour
{
    public float panSpeed = 30f;
    public float panBorderThickness = 10f;

```

```

public float scrollSpeed = 5f;
public float minY = 10f;
public float maxY = 80f;

void Update()
{
    if (GameManager.GameIsOver)
    {
        enabled = false;
        return;
    }

    if (Input.GetKey("w") || Input.mousePosition.y >=
Screen.height - panBorderThickness)
    {
        transform.Translate(Vector3.forward * panSpeed *
Time.deltaTime, Space.World);
    }

    if (Input.GetKey("s") || Input.mousePosition.y <=
panBorderThickness)
    {
        transform.Translate(Vector3.back * panSpeed *
Time.deltaTime, Space.World);
    }

    if (Input.GetKey("d") || Input.mousePosition.x >=
Screen.width - panBorderThickness)
    {
        transform.Translate(Vector3.right * panSpeed *
Time.deltaTime, Space.World);
    }

    if (Input.GetKey("a") || Input.mousePosition.x <=
Screen.height - panBorderThickness)
    {
        transform.Translate(Vector3.left * panSpeed *
Time.deltaTime, Space.World);
    }

    float scroll = Input.GetAxis("Mouse ScrollWheel");
    Vector3 pos = transform.position;
    pos.y -= scroll * 1000 * scrollSpeed * Time.deltaTime;
    pos.y = Mathf.Clamp(pos.y, minY, maxY);
    transform.position = pos;
}
}

```

Файл *CompleteLevel.cs*

using System.Collections;

					ИПУ 1.090301.21.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		86

```

using System.Collections.Generic;
using UnityEngine;

public class CompleteLevel : MonoBehaviour
{
    public SceneFader sceneFader;
    public string menuSceneName = "MainMenu";

    public string nextLevel = "Level02";
    public int levelToUnlock = 2;

    public void Continue()
    {
        PlayerPrefs.SetInt("levelReached", levelToUnlock);
        sceneFader.FadeTo(nextLevel);
    }

    public void Menu()
    {
        sceneFader.FadeTo(menuSceneName);
    }
}

```

Файл *Enemy.cs*

```

using UnityEngine;
using UnityEngine.UI;

public class Enemy : MonoBehaviour
{
    public float startSpeed = 10f;
    [HideInInspector]
    public float speed;
    private float startHealth = 100;
    public float health;
    public int worth = 50;

    public GameObject deathEffect;

    [Header("Unity Stuff")]
    public Image healthBar;

    private bool isDead = false;
    void Start()
    {
        speed = startSpeed;
        health = startHealth;
    }

    public void TakeDamage(float amount)
    {
        health -= amount;
        healthBar.fillAmount = health / startHealth;
    }
}

```

```

        if (health <= 0 && !isDead)
        {
            Die();
        }
    }

    public void Slow(float pct)
    {
        speed = startSpeed * (1f - pct);
    }

    void Die()
    {
        isDead = true;
        PlayerStats.Money += worth;

        GameObject effect = Instantiate(deathEffect,
transform.position, Quaternion.identity);
        Destroy(effect, 5f);

        WaveSpawner.EnemyiesAlive--;

        Destroy(gameObject);
    }
}

```

Файл *EnemyMovement.cs*

```

using UnityEngine;

[RequireComponent(typeof(Enemy))]
public class EnemyMovement : MonoBehaviour
{
    private Transform target;
    private int wavePointIndex = 0;

    private Enemy enemy;
    void Start()
    {
        enemy = GetComponent<Enemy>();
        target = Waypoints.points[0];
    }

    void Update()
    {
        Vector3 dir = target.position - transform.position;
        transform.Translate(dir.normalized * enemy.speed *
Time.deltaTime, Space.World);
        if (Vector3.Distance(transform.position,
target.position) <= 0.4f)
        {
            GetNextWaypoint();
        }
    }
}

```

```

        enemy.speed = enemy.startSpeed;
    }

    void GetNextWaypoint()
    {
        if (wavePointIndex >= Waypoints.points.Length - 1)
        {
            EndPath();
            return;
        }

        wavePointIndex++;
        target = Waypoints.points[wavePointIndex];
    }

    void EndPath()
    {
        PlayerStats.Lives--;
        WaveSpawner.EnemyiesAlive--;
        Destroy(gameObject);
    }
}

```

Файл *GameManager.cs*

```

using UnityEngine;

public class GameManager : MonoBehaviour
{
    public static bool GameIsOver;
    public GameObject gameOverUI;
    public GameObject completeLevelUI;

    void Start()
    {
        GameIsOver = false;
    }

    void Update()
    {
        if (GameIsOver)
        {
            return;
        }

        if (PlayerStats.Lives <= 0)
        {
            EndGame();
        }
    }

    void EndGame()

```

```

    {
        GameIsOver = true;
        gameOverUI.SetActive(true);
    }

    public void WinLevel()
    {
        GameIsOver = true;
        completeLevelUI.SetActive(true);
    }
}

```

Файл *GameOver.cs*

```

using UnityEngine;
using UnityEngine.SceneManagement;

public class GameOver : MonoBehaviour
{
    public SceneFader sceneFader;
    public string menuSceneName = "MainMenu";

    public void Retry()
    {
        sceneFader.FadeTo(SceneManager.GetActiveScene().name);
    }

    public void Menu()
    {
        sceneFader.FadeTo(menuSceneName);
    }
}

```

Файл *LevelSelector.cs*

```

using UnityEngine;
using UnityEngine.UI;

public class LevelSelector : MonoBehaviour
{
    public SceneFader fader;
    public Button[] levelButtons;

    void Start()
    {
        int levelReached = PlayerPrefs.GetInt("levelReached",
1);

        for (int i = 0; i < levelButtons.Length; i++)
        {
            if (i + 1 > levelReached)
            {
                levelButtons[i].interactable = false;

```

```

    }
}

public void Select(string levelName)
{
    fader.FadeTo(levelName);
}
}

```

Файл *LivesUI.cs*

```

using UnityEngine;
using UnityEngine.UI;

public class LivesUI : MonoBehaviour
{
    public Text livesText;

    void Update()
    {
        livesText.text = PlayerStats.Lives.ToString() + "
LIVES";
    }
}

```

Файл *MainMenu.cs*

```

using UnityEngine;
using UnityEngine.SceneManagement;

public class MainMenu : MonoBehaviour
{
    public string levelToLoad = "MainLevel";
    public SceneFader sceneFader;
    public void Play()
    {
        sceneFader.FadeTo(levelToLoad);
    }

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

```

Файл *MoneyUI.cs*

```

using UnityEngine;
using UnityEngine.UI;

public class MoneyUI : MonoBehaviour
{
    public Text moneyText;
}

```

```

void Update()
{
    moneyText.text = "$" + PlayerStats.Money.ToString();
}
}

```

Файл *Node.cs*

```

using UnityEngine;
using UnityEngine.EventSystems;

public class Node : MonoBehaviour
{
    public Color hoverColor;
    public Color notEnoughMoneyColor;
    private Color startColor;
    private Renderer rend;

    [HideInInspector]
    public GameObject turret;
    [HideInInspector]
    public TurretBlueprint turretBlueprint;
    [HideInInspector]
    public bool isUpgraded = false;

    public Vector3 positionOffset;

    BuildManager buildManager;

    void Start()
    {
        rend = GetComponent<Renderer>();
        startColor = rend.material.color;
        buildManager = BuildManager.instance;
    }

    public Vector3 GetBuildPosition()
    {
        return transform.position + positionOffset;
    }

    void OnMouseEnter()
    {
        if (EventSystem.current.IsPointerOverGameObject())
        {
            return;
        }

        if (!buildManager.CanBuild)
        {
            return;
        }
    }
}

```

```

    }

    if (buildManager.HasMoney)
    {
        rend.material.color = hoverColor;
    }
    else
    {
        rend.material.color = notEnoughMoneyColor;
    }
}

void OnMouseExit()
{
    rend.material.color = startColor;
}

void OnMouseDown()
{
    if (EventSystem.current.IsPointerOverGameObject())
    {
        return;
    }

    if (turret != null)
    {
        buildManager.SelectedNode(this);
        return;
    }

    if (!buildManager.CanBuild)
    {
        return;
    }

    BuildTurret(buildManager.GetTurretToBuild());
}

void BuildTurret(TurretBlueprint blueprint)
{
    if (PlayerStats.Money < blueprint.cost)
    {
        return;
    }
    PlayerStats.Money -= blueprint.cost;

    GameObject _turret = Instantiate(blueprint.prefab,
    GetBuildPosition(), Quaternion.identity);
    turret = _turret;

    turretBlueprint = blueprint;
}

```

```

        GameObject effect =
Instantiate(buildManager.buildEffect, GetBuildPosition(),
Quaternion.identity);
        Destroy(effect, 5f);
    }

    public void UpgradeTurret()
    {
        if (PlayerStats.Money < turretBlueprint.upgradeCost)
        {
            return;
        }

        PlayerStats.Money -= turretBlueprint.upgradeCost;
        Destroy(turret);

        GameObject _turret =
Instantiate(turretBlueprint.upgradedPrefab, GetBuildPosition(),
Quaternion.identity);
        turret = _turret;

        GameObject effect =
Instantiate(buildManager.buildEffect, GetBuildPosition(),
Quaternion.identity);
        Destroy(effect, 5f);

        isUpgraded = true;
    }

    public void SellTurret()
    {
        PlayerStats.Money += turretBlueprint.GetSellAmount();

        GameObject effect = Instantiate(buildManager.sellEffect,
GetBuildPosition(), Quaternion.identity);
        Destroy(effect, 5f);

        Destroy(turret);
        turretBlueprint = null;
    }
}

```

Файл *NodeUI.cs*

```

using UnityEngine;
using UnityEngine.UI;

public class NodeUI : MonoBehaviour
{
    public GameObject ui;
    private Node target;
    public Text upgradeCost;
    public Button upgradeButton;
}

```

```

public Text sellAmount;
public void SetTarget(Node _target)
{
    target = _target;
    transform.position = target.GetBuildPosition();

    if (!target.isUpgraded)
    {
        upgradeCost.text = "$" +
target.turretBlueprint.upgradeCost;
        upgradeButton.interactable = true;
    }
    else
    {
        upgradeCost.text = "DONE";
        upgradeButton.interactable = false;
    }

    sellAmount.text = "$" +
target.turretBlueprint.GetSellAmount();

    ui.SetActive(true);
}

public void Hide()
{
    ui.SetActive(false);
}

public void Upgrade()
{
    target.UpgradeTurret();
    BuildManager.instance.DeselectNode();
}

public void Sell()
{
    target.SellTurret();
    BuildManager.instance.DeselectNode();
}
}

```

Файл *PauseMenu.cs*

```

using UnityEngine;
using UnityEngine.SceneManagement;

public class PauseMenu : MonoBehaviour
{
    public GameObject ui;
    public SceneFader sceneFader;
    public string menuSceneName = "MainMenu";
    void Update()

```

```

    {
        if (Input.GetKeyDown(KeyCode.Escape))
        {
            Toggle();
        }
    }

    public void Toggle()
    {
        ui.SetActive(!ui.activeSelf);

        if (ui.activeSelf)
        {
            Time.timeScale = 0f;
        }
        else
        {
            Time.timeScale = 1f;
        }
    }

    public void Retry()
    {
        Toggle();
        sceneFader.FadeTo(SceneManager.GetActiveScene().name);
    }

    public void Menu()
    {
        Toggle();
        sceneFader.FadeTo(menuSceneName);
    }
}

```

Файл *PlayerStats.cs*

```

using UnityEngine;

public class PlayerStats : MonoBehaviour
{
    public static int Money;
    public int startMoney = 400;

    public static int Lives;
    public int startLives = 20;

    public static int Rounds;
    void Start()
    {
        Money = startMoney;
        Lives = startLives;
        Rounds = 0;
    }
}

```

```
}
```

Файл *RoundsSurvived.cs*

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

public class RoundsSurvived : MonoBehaviour
{
    public Text roundsText;

    void OnEnable()
    {
        StartCoroutine(AnimateText());
    }

    IEnumerator AnimateText()
    {
        roundsText.text = "0";
        int round = 0;

        yield return new WaitForSeconds(.7f);

        while (round < PlayerStats.Rounds)
        {
            round++;
            roundsText.text = round.ToString();

            yield return new WaitForSeconds(.05f);
        }
    }
}
```

Файл *SceneFader.cs*

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

public class SceneFader : MonoBehaviour
{
    public Image img;
    public AnimationCurve curve;

    void Start()
    {
        StartCoroutine(FadeIn());
    }

    public void FadeTo(string scene)
    {
        StartCoroutine(FadeOut(scene));
    }
}
```

```

    }

IEnumerator FadeIn()
{
    float t = 1f;

    while (t > 0f)
    {
        t -= Time.deltaTime;
        float a = curve.Evaluate(t);
        img.color = new Color(0f, 0f, 0f, a);
        yield return 0;
    }
}

IEnumerator FadeOut(string scene)
{
    float t = 0f;

    while (t < 1f)
    {
        t += Time.deltaTime;
        float a = curve.Evaluate(t);
        img.color = new Color(0f, 0f, 0f, a);
        yield return 0;
    }
    SceneManager.LoadScene(scene);
}
}

```

Файл *Shop.cs*

```

using UnityEngine;

public class Shop : MonoBehaviour
{
    BuildManager buildManager;
    public TurretBlueprint standardTurret;
    public TurretBlueprint missileLauncher;
    public TurretBlueprint laserBeamer;
    public void SelectStandardTurret()
    {
        buildManager.SelectTurretToBuild(standardTurret);
    }

    public void SelectMissileLauncher()
    {
        buildManager.SelectTurretToBuild(missileLauncher);
    }

    public void SelectLaserBeamer()
    {
        buildManager.SelectTurretToBuild(laserBeamer);
    }
}

```

```
}
}
```

Файл *Turret.cs*

```
using UnityEngine;

public class Turret : MonoBehaviour
{
    private Transform target;
    private Enemy targetEnemy;

    [Header("General")]
    public float range = 15f;

    [Header("Use Bullets (default)")]
    public GameObject bulletPrefab;
    public float fireRate = 1f;
    private float fireCountdown = 0f;

    [Header("Use Laser")]
    public bool useLaser = false;
    public LineRenderer lineRenderer;
    public int damageOverTime = 30;
    public float slowAmount = .5f;

    [Header("Unity Setup Fields")]
    public string enemyTag = "Enemy";
    public Transform partToRotate;
    public float turnSpeed = 10f;

    public Transform firePoint;

    void Start()
    {
        InvokeRepeating("UpdateTarget", 0f, 0.5f);
    }

    void Update()
    {
        if (target == null)
        {
            if (useLaser)
            {
                if (lineRenderer.enabled)
                {
                    lineRenderer.enabled = false;
                }
            }
            return;
        }
        LockOnTarget();
    }
}
```

```

        if (useLaser)
        {
            Laser();
        }
    else
    {
        if (fireCountdown <= 0f)
        {
            Shoot();
            fireCountdown = 1f / fireRate;
        }
        fireCountdown -= Time.deltaTime;
    }
}

void LockOnTarget()
{
    Vector3 dir = target.position - transform.position;
    Quaternion lookRotation = Quaternion.LookRotation(dir);
    Vector3 rotation =
Quaternion.Lerp(partToRotate.rotation, lookRotation,
Time.deltaTime * turnSpeed).eulerAngles;
    partToRotate.rotation = Quaternion.Euler(0f, rotation.y,
0f);
}

void Laser()
{
    targetEnemy.TakeDamage(damageOverTime * Time.deltaTime);
    targetEnemy.Slow(slowAmount);

    if (!lineRenderer.enabled)
    {
        lineRenderer.enabled = true;
    }

    lineRenderer.SetPosition(0, firePoint.position);
    lineRenderer.SetPosition(1, target.position);
}

void UpdateTarget()
{
    GameObject[] enemies =
GameObject.FindGameObjectsWithTag(enemyTag);
    float shortestDistance = Mathf.Infinity;
    GameObject nearestEnemy = null;

    foreach (GameObject enemy in enemies)
    {
        float distanceToEnemy =
Vector3.Distance(transform.position, enemy.transform.position);
        if (distanceToEnemy < shortestDistance)
        {

```

```

        shortestDistance = distanceToEnemy;
        nearestEnemy = enemy;
    }
}

if (nearestEnemy != null && shortestDistance <= range)
{
    target = nearestEnemy.transform;
    targetEnemy = nearestEnemy.GetComponent<Enemy>();
}
else
{
    target = null;
}
}

private void OnDrawGizmosSelected()
{
    Gizmos.color = Color.red;
    Gizmos.DrawSphere(transform.position, range);
}

void Shoot()
{
    GameObject bulletGo =
    (GameObject)Instantiate(bulletPrefab, firePoint.position,
    firePoint.rotation);
    Bullet bullet = bulletGo.GetComponent<Bullet>();

    if (bullet != null)
    {
        bullet.Seek(target);
    }
}
}

```

Файл *TurretBlueprint.cs*

```

using UnityEngine;
[System.Serializable]
public class TurretBlueprint
{
    public GameObject prefab;
    public int cost;

    public GameObject upgradedPrefab;
    public int upgradeCost;

    public int GetSellAmount()
    {
        return cost / 2;
    }
}

```

Файл Waves.cs

```
using UnityEngine;

[System.Serializable]
public class Wave
{
    public GameObject enemy;
    public int count;
    public float rate;
}
```

Файл WaveSpawner.cs

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

public class WaveSpawner : MonoBehaviour
{
    public static int EnemyiesAlive = 0;
    public Wave[] waves;
    public Transform spawnPoint;

    public float timeBetweenWaves = 5f;
    public Text waveCountdownText;

    private float countDown = 2f;
    private int waveIndex = 0;

    public GameManager gameManager;

    void Start()
    {
        EnemyiesAlive = 0;
    }
    void Update()
    {
        if (EnemyiesAlive > 0)
        {
            return;
        }

        if (waveIndex == waves.Length)
        {
            gameManager.WinLevel();
            enabled = false;
        }

        if (countDown <= 0f)
        {
            StartCoroutine(SpawnWave());
        }
    }
}
```

```

        countdown = timeBetweenWaves;
        return;
    }
    countdown -= Time.deltaTime;
    countdown = Mathf.Clamp(countdown, 0f, Mathf.Infinity);
    waveCountdownText.text = string.Format("{0:00.00}",
countdown);
}

IEnumerator SpawnWave()
{
    PlayerStats.Rounds++;
    Wave wave = waves[waveIndex];
    EnemyiesAlive = wave.count;

    for (int i = 0; i < wave.count; i++)
    {
        SpawnEnemy(wave.enemy);
        yield return new WaitForSeconds(1f / wave.rate);
    }
    waveIndex++;
}

void SpawnEnemy(GameObject enemy)
{
    Instantiate(enemy, spawnPoint.position,
spawnPoint.rotation);
}
}

```

Файл *Waypoints.cs*

```

using UnityEngine;

public class Waypoints : MonoBehaviour
{
    public static Transform[] points;
    void Awake()
    {
        points = new Transform[transform.childCount];

        for (int i = 0; i < points.Length; i++)
        {
            points[i] = transform.GetChild(i);
        }
    }
}

```

Приложение Ж

(Обязательное)

Мобильное приложение под ОС Андроид

Презентация

					ПГУ 1.090301.21.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		104

Мобильное приложение под ОС Android

Автор работы:
студент группы 18ВВ1 Кривоzubов А. К.

Научный руководитель:
к.т.н., доцент каф. ВТ Юрова О. В.

Активация Windows
Чтобы активировать Windows, перейдите в раздел
"Параметры".



Цель работы

Цель работы – разработка мобильного приложения под ОС Андроид.

Объект работы – мобильное игровое приложение, выполняющее следующие функции:

- Совместимость с актуальными версиями *Android*;
- Поддержка любых разрешений экрана;
- Соответствие классическим правилам игры;
- Поддержка добавления и удаления новых объектов;
- Реализация строительства оборонительных сооружений (башен);
- Создание и генерация противников;
- Содержать интуитивно понятный графический интерфейс.

Активация Windows
Чтобы активировать Windows, перейдите в раздел
"Параметры".

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.090301.21.001 ПЗ

Лист

105

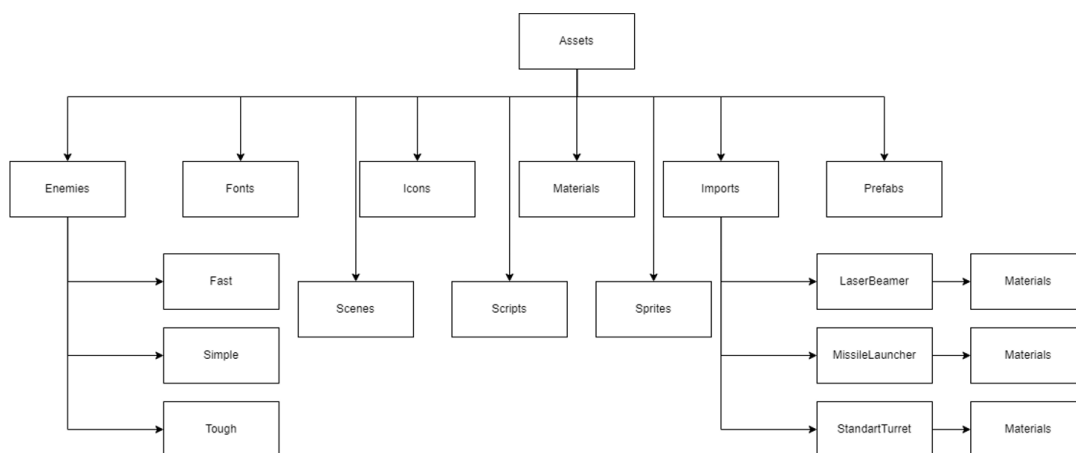
Технологии, применяемые при проектировании и разработке



Активация Windows
Чтобы активировать Windows, перейдите в раздел "Параметры".



Файловая структура приложения



Активация Windows
Чтобы активировать Windows, перейдите в раздел "Параметры".

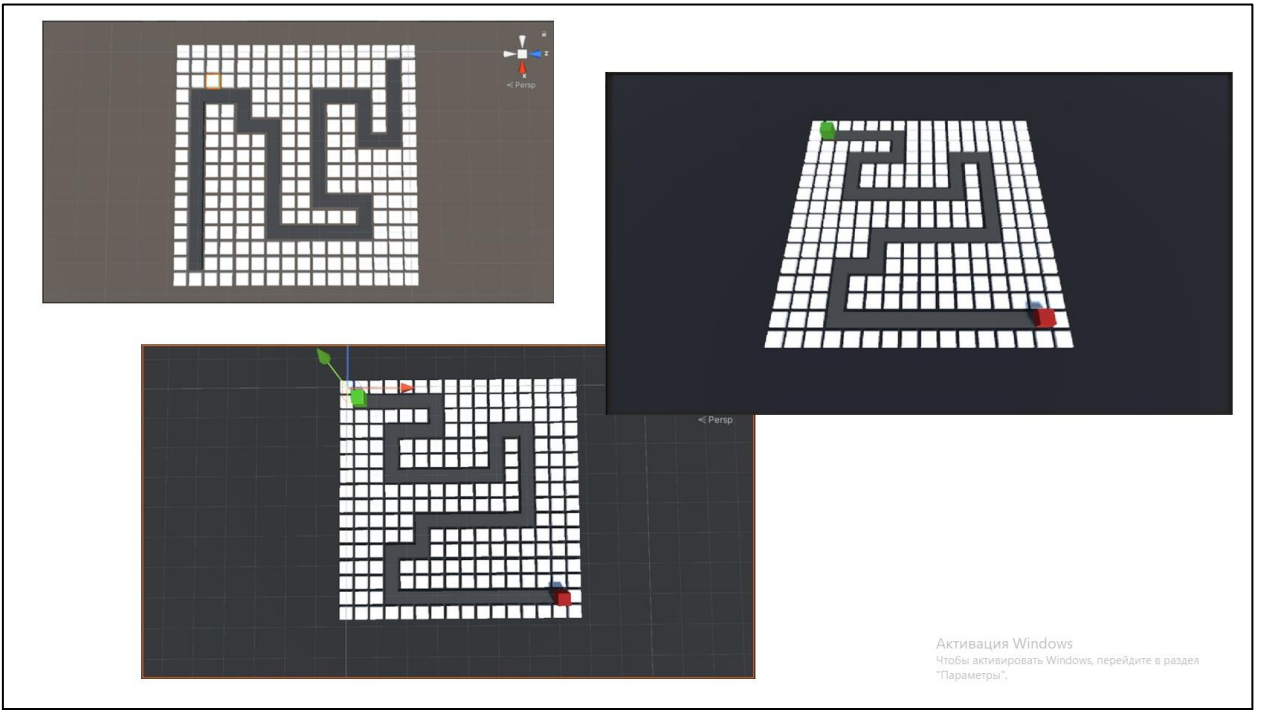
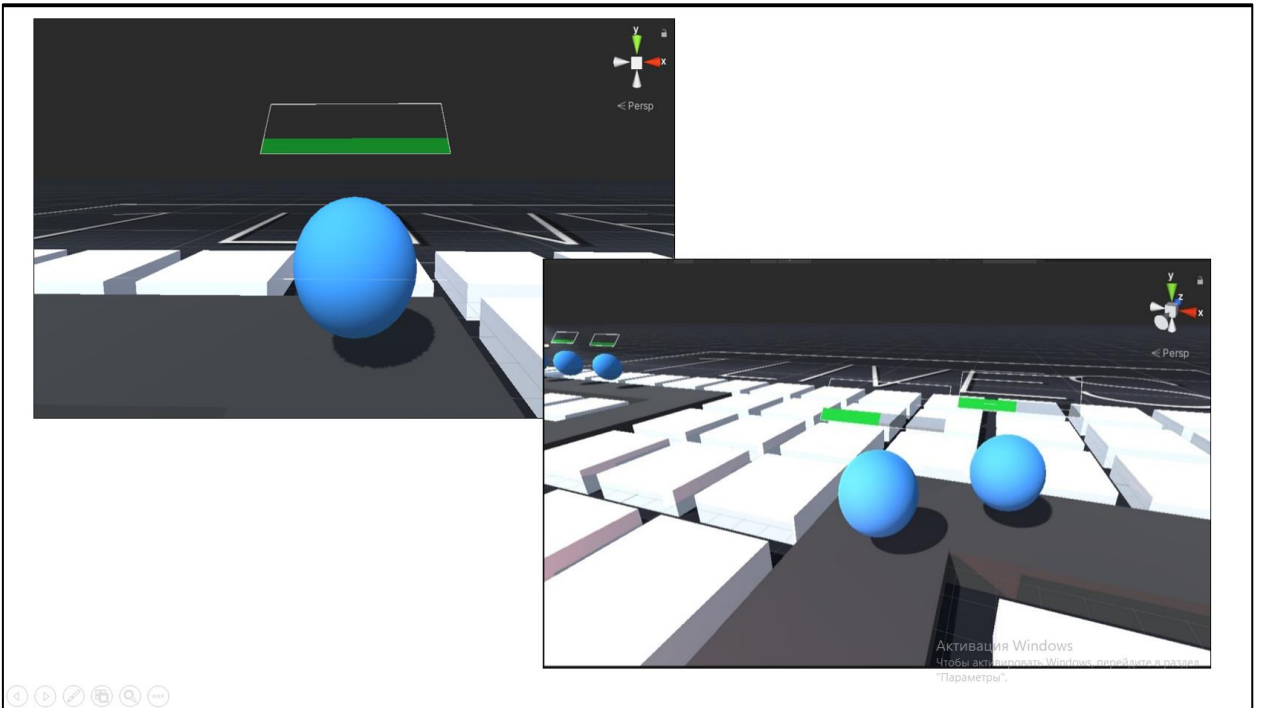


Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.090301.21.001 ПЗ

Лист

106

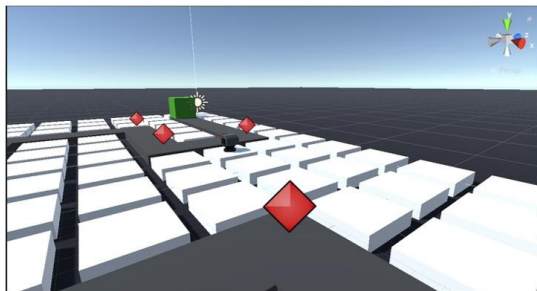
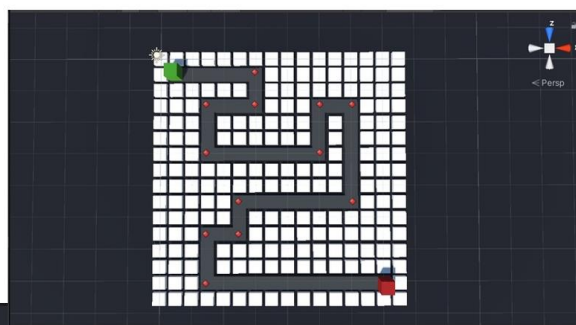
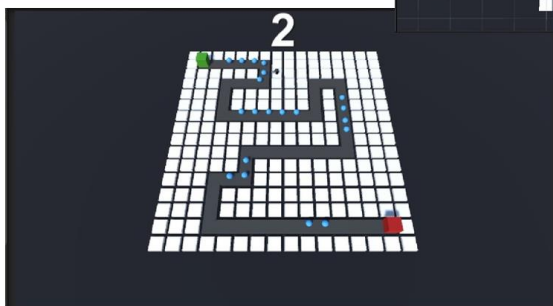


Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.090301.21.001 ПЗ

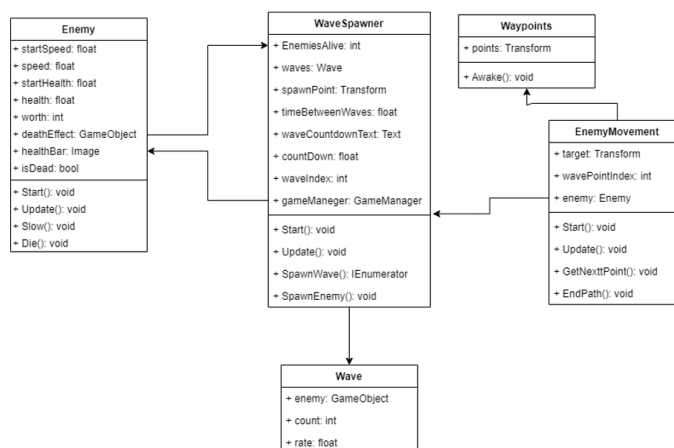
Лист

107

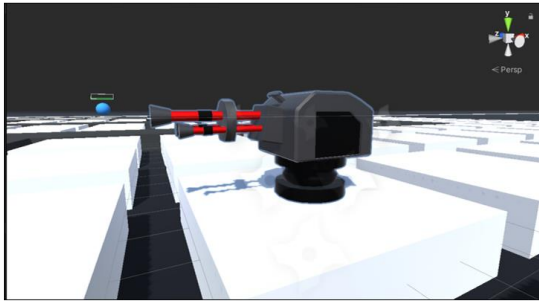
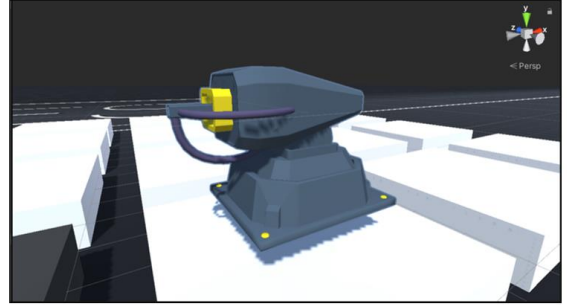
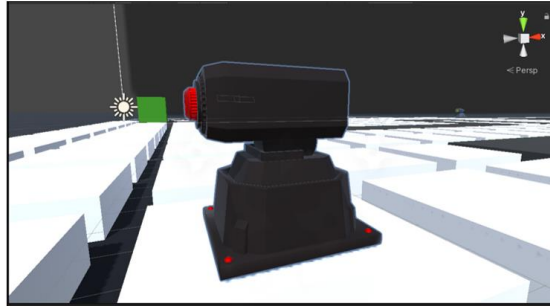


Активация Windows
Чтобы активировать Windows, перейдите в раздел
"Параметры".

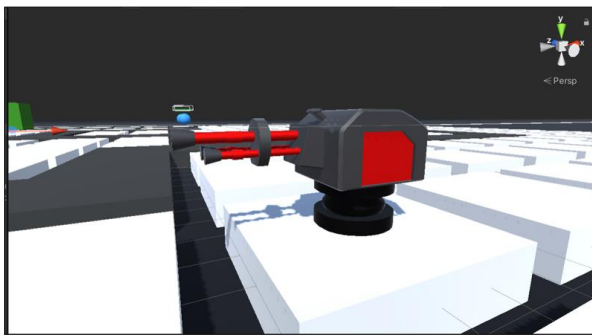
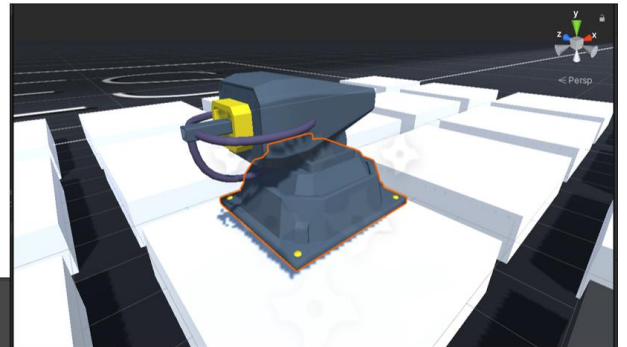
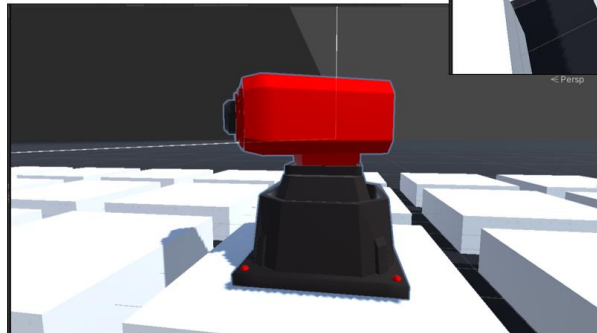
UML-диаграмма классов противников



Активация Windows
Чтобы активировать Windows, перейдите в раздел
"Параметры".



Активация Windows
Чтобы активировать Windows, перейдите в раздел
"Параметры".



Активация Windows
Чтобы активировать Windows, перейдите в раздел
"Параметры".

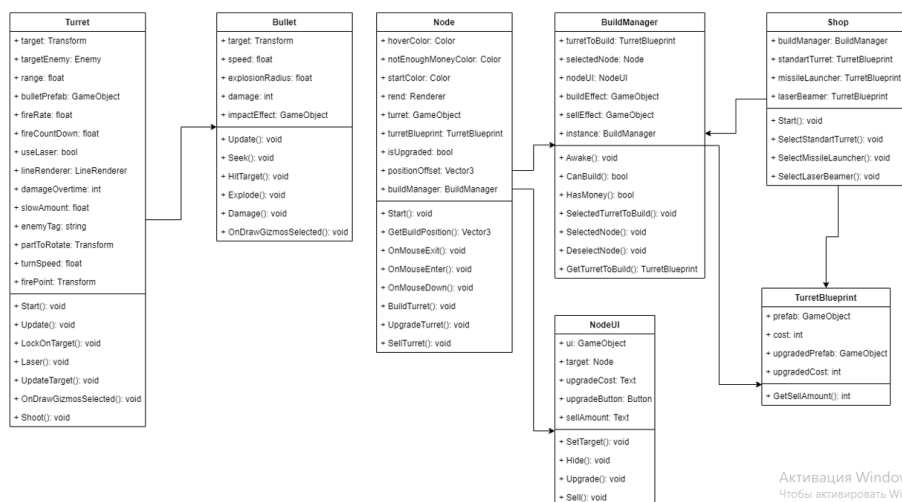
Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.090301.21.001 ПЗ

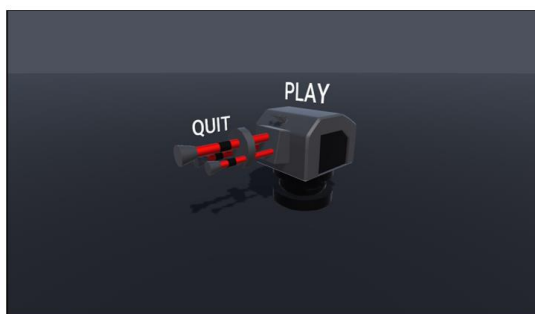
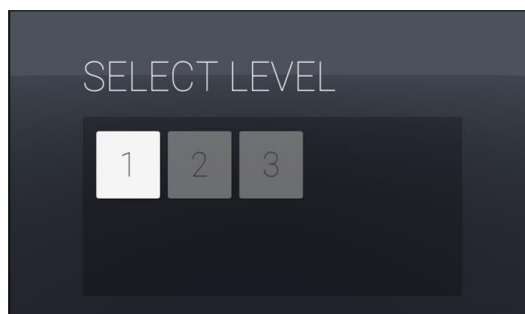
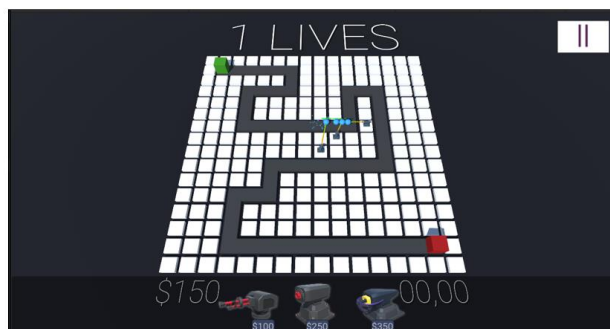
Лист

109

UML-диаграмма классов башен

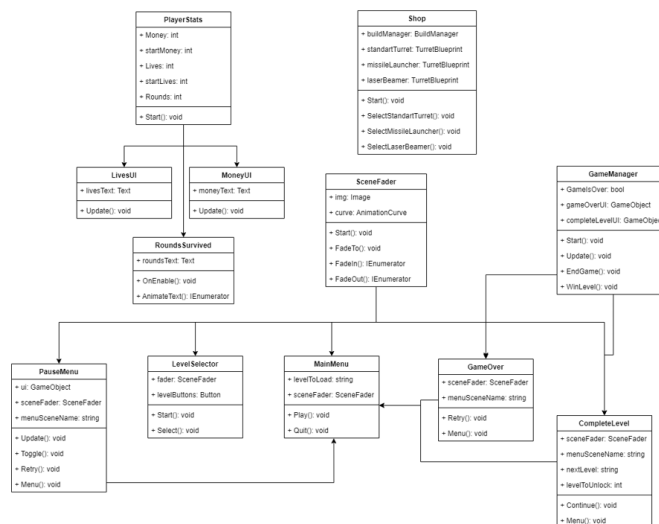


Активация Windows
Чтобы активировать Windows, перейдите в раздел "Параметры".

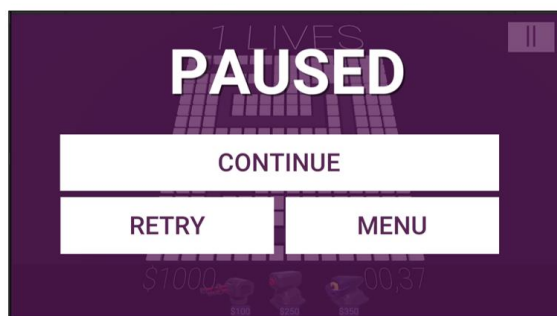
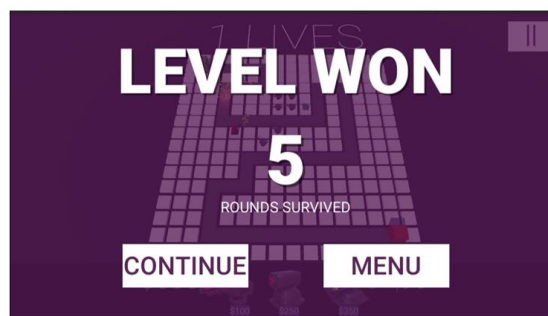
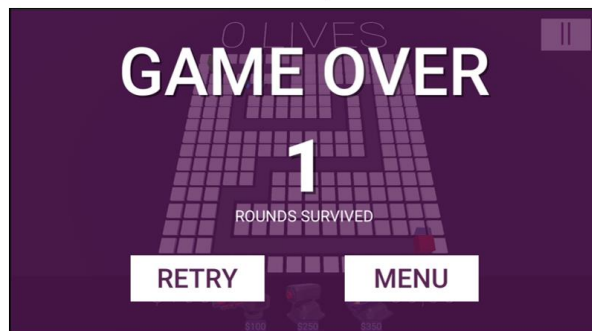


Активация Windows
Чтобы активировать Windows, перейдите в раздел "Параметры".

UML-диаграмма классов интерфейса пользователя

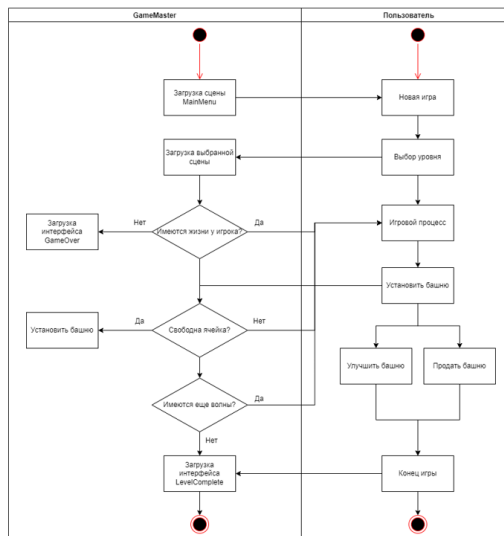


Активация Windows
Чтобы активировать Windows, перейдите в раздел
"Параметры".



Активация Windows
Чтобы активировать Windows, перейдите в раздел
"Параметры".

UML-диаграмма деятельности



Активация Windows
Чтобы активировать Windows, перейдите в раздел
"Параметры".

UML-диаграмма прецедентов



Активация Windows
Чтобы активировать Windows, перейдите в раздел
"Параметры".

Результат работы

В ходе выполнения данной работы было спроектировано, разработано и протестировано мобильное игровое приложение «*Tower Defense*».

В ходе выполнения проекта решался ряд задач, среди которых необходимо выделить следующие:

- анализ игровой индустрии;
- этапы разработки мобильного приложения;
- анализ существующих игровых классификаций;
- определение набора задач для разрабатываемого приложения;
- выбор средств разработки приложения;
- написание программного кода;
- создание графической части приложения и интерфейс пользователя;
- проектирование структурных схем функционирования приложения.

Активация Windows
Чтобы активировать Windows, перейдите в раздел
"Параметры".

Спасибо за внимание!

Активация Windows
Чтобы активировать Windows, перейдите в раздел
"Параметры".

					ПГУ 1.090301.21.001 ПЗ	Лист
						113
Изм.	Лист	№ докум.	Подпись	Дата		