Министерство науки и высшего образования Российской Федерации
ФГБОУ ВО «Алтайский государственный университет»
Школа развития цифровых компетенций «Digital Up» (цифровая кафедра)

ИНДИВИДУАЛЬНОЕ ЗАДАНИЕ

на итоговый проект «Разработка двумерного платформера “Mooneater”»
по ДПП ПП «Основы Gamedev и VR-разработки»

п/п №	Задание	Исполнитель	Рабочий график (план) выполнения
1	Изучить основы разработки 2D-платформеров и существующие аналоги в жанре.	Манакова Я. Н. Пяткова А. Е.	01.03.2025 – 15.03.2025
2	Разработать концепцию игры, включая геймплей, уровни, персонажа и игровые механики.		16.03.2025 – 31.03.2025
3	Реализовать прототип игры на движке Unity. Протестировать игру, доработать найденные недочёты геймплея и игрового баланса.		01.04.2025 – 25.05.2025
4	Подготовить финальную версию, отчет и презентацию проекта.		25.03.2025 – 31.05.2025

Руководитель проекта
старший преподаватель кафедры культурологии и дизайна, Каратаев А. А.

_____ /_____ / «___ » _____ 2025 г.
(подпись)

СОГЛАСОВАНО:

Руководитель ДПП ПП
канд. физ.-мат. наук, доцент Козлов Д.Ю.

_____ /_____/ «____»_____2025 г.
(подпись)

Министерство науки и высшего образования Российской Федерации
ФГБОУ ВО «Алтайский государственный университет»
Школа развития цифровых компетенций «Digital Up» (цифровая кафедра)

Отчет о выполнении группового итогового проекта по ДПП ПП

«Основы Gamedev и VR-разработки»

«Разработка двумерного платформера “Mooneater”»

Исполнители:
Манакова Я. Н.
Пяткова А. Е.

«____» _____ 2025 г.

Руководитель проекта
ст. пр. Каратаев А. А.
«____» _____ 2025 г.

г. Барнаул, 2025

1. Цель проекта

Целью проекта является разработка игры для мобильных android-устройств в жанре 2D-платформер с игровым процессом, включающим в себя механики взаимодействия игрока с миром, искусственного интеллекта противников и системы развития характеристик персонажа игрока.

2. Задачи проекта и исполнители

Для достижения поставленной цели были сформированы и выполнены следующие задачи:

1. Изучить основы разработки 2D-платформеров и существующие аналоги в жанре.
2. Разработать концепцию игры, включая геймплей, уровни, персонажа и игровые механики.
3. Реализовать прототип игры на движке Unity. Протестировать игру, доработать найденные недочёты геймплея и игрового баланса.
4. Подготовить финальную версию, отчет и презентацию проекта.

3. Актуальность и востребованность проекта

Актуальность настоящего проекта обусловлена востребованностью технологий геймификации в современном мире. Платформеры остаются одним из самых востребованных жанров среди геймеров. По данным Steam (2025), максимальный суммарный онлайн 2D-платформеров из топ-100 самых популярных превышает 1 млн. игроков.

В данном проекте разрабатываемая игра несёт образовательную функцию и является инструментом для изучения основ геймдизайна, базовых навыков работы с игровым движком Unity и программирования в рамках курса цифровой кафедры «Основы Gamedev и VR-разработки». Разработка платформера является полноценным проектом в сфере геймдизайна и позволит пройти через все этапы создания игры самостоятельно, существенно улучшить понимание этого процесса, а решение реальных задач внутри проекта позволит развиться разработчикам как специалистам.

Полученный опыт разработки платформера в дальнейшем можно применить в сферах образования и бизнеса в формате геймификации задач данных сфер или в сфере гейминга для создания самостоятельного игрового продукта.

4. Общие сведения о проделанной работе

Этапы работы над проектом были следующими:

1. Изучить основы разработки 2D-платформеров и существующие аналоги в жанре.
2. Разработать концепцию игры, включая геймплей, уровни, персонажа и игровые механики.
3. Реализовать graybox прототип игры на движке Unity. Провести промежуточное тестирование.
4. Создать визуальное оформление игры и внедрить его в проект.
5. Протестировать игру, доработать найденные недочёты геймплея и игрового баланса.

В начале разработки были изучены популярные 2D-платформеры, такие как Hollow Knight, Celeste, Night in the Woods и другие из игрового опыта исполнителей данного проекта. Выбрана упрощенная концепция прохождения уровней, близкая к Hollow Knight, так как эта игра является самой популярной в своём жанре по версии Steam.

В ходе курса «Основы Gamedev и VR-разработки» были изучены базовые инструменты движка Unity, такие как работа с объектами, сценами, звуками и освещением. Изучена структура языка программирования C# и работа со скриптами в Unity. Движок отлично подходил для выполнения задач по разработке, так как имеет гибкие возможности настройки проекта, является бесплатным для некоммерческих продуктов и позволяет тонко настраивать поведение каждого игрового объекта благодаря наличию в C# библиотеки UnityEngine, специально созданной для разработки игр. Помимо данной библиотеки использовались также библиотеки System для настройки некоторых асинхронных функций и TMPro для работы с текстовыми UI-элементами.

Объектный язык программирования C# имеет гибкие возможности, создание классов для каждой задачи позволило объединить скрипты игры в налаженную систему и повысило эффективность разработки. Для оптимизации рутинных процедур в написании кода использовалась нейросеть Deepseek. Результаты её работы корректировались под проект. Использование нейронной сети позволило структурировать код игры и подбирать наиболее подходящие для доработки в проекте функции.

Для генерации фоновой музыки использовалась нейронная сеть Riffusion. Фоновые звуки были взяты в бесплатной онлайн-фонотеке Freesound.

Разработка игры началась с создания концепции. С помощью облачного хранилища была реализована база со всей документацией и расчётами для игры. Разработан нарратив игры, продуманы механики игрока, врагов, интерактивных и бонусных предметов, создан экономический элемент (игровая валюта) и система бонусов для развития характеристик игрока. Разработаны концепции способностей игрока (передвижение с помощью

джойстика, прыжок и удар с помощью кнопок; здоровье и урон) и врагов (четыре типа врагов; передвижение, режим патрулирования и преследования, атака, здоровье и урон), интерактивных и декоративных предметов на локациях, меню магазина для покупки бонусов и главное меню, а также общий стиль игры (визуальное решение и саунд-дизайн). Созданы схематичные концепции трёх уровней игры, на которых отображено примерное расположение игрока, врагов, интерактивных и декоративных предметов, а также проложен путь игрока от начала уровня до его конца.

Геймплей игры заключается в прохождении трёх уровней и прохождении финального босса игры. По сюжету главный герой должен вернуть луну в мир духов, от света которой зависит их существование, и найти свою жену, которая попала в мир духов до событий игры. Луну захватил озлобленный дух, Пожиратель Луны, в честь которого и названа игра. Каждый последующий уровень сложнее предыдущего из-за повышения количества сильных врагов. На каждом уровне игрок может собирать игровую валюту с поверженных NPC, чтобы повысить свои параметры, купив в магазине один из трёх бонусов: здоровья, урона или дропа. Каждый уровень происходит «инфляция» денег игрока и цены в магазине растут, что даёт игроку пространство для принятия решений и выбора тактики прохождения игры. Также на каждом уровне есть интерактивные элементы: статуя-чекпоинт, лунный свет и колючки. Чекпоинт позволяет сохранять прогресс на уровне, лунный свет и колючки при столкновении с игроком повышают или понижают его текущее здоровье соответственно.

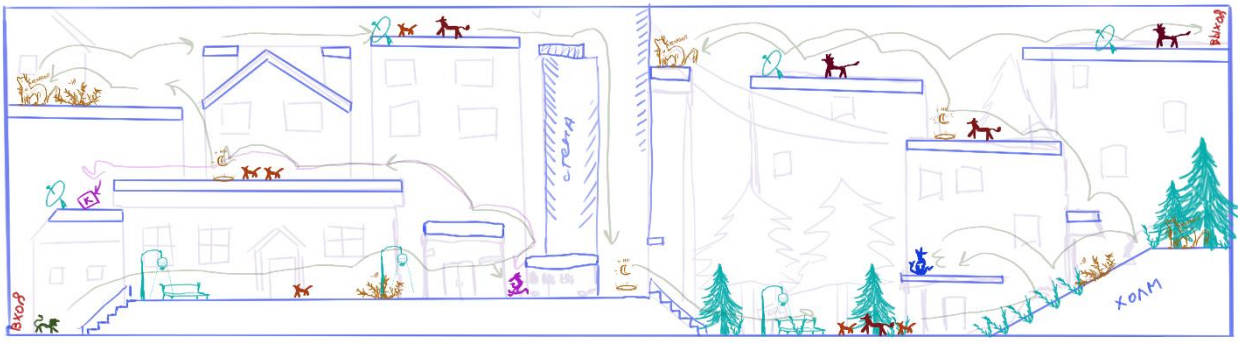


Рис. 1. Схема 1 уровня игры.

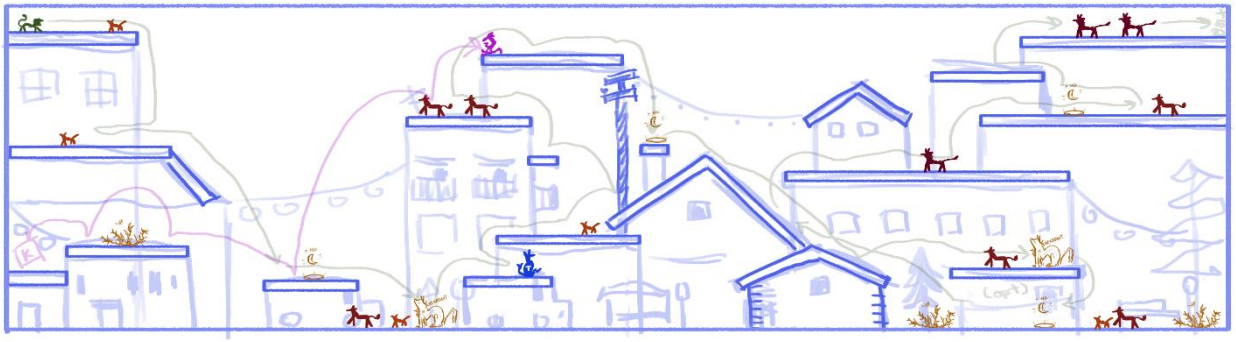


Рис. 2. Схема 2 уровня игры.

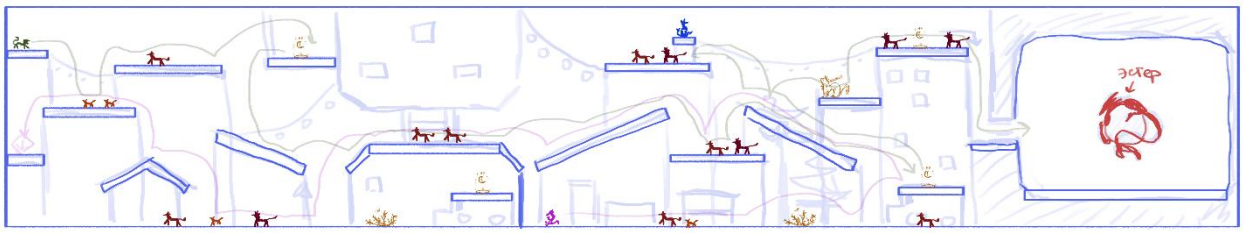


Рис. 3. Схема 3 уровня игры.

Работе непосредственно в Unity предшествовал этап создания игрового баланса. С помощью электронной таблицы были рассчитаны показатели боевой системы игры: максимальные и минимальные показатели здоровья и урона игрока, зависящие от них показатели здоровья и урона врагов. Показатели врагов рассчитывались следующим образом:

maxHP * Показатель типа = HP врага

maxDMG * Показатель типа = DMG врага

где maxHP — максимальное здоровье персонажа игрока (со всеми бонусами),

maxDMG — максимальный урон персонажа игрока (со всеми бонусами),

Показатель типа — множитель от 0,2 до 2, соответствующий каждому из 4 типов врагов.

В дальнейшем после тестирования урон сильного врага и босса был проставлен вручную без учёта формулы для балансировки сложности игры.

В другой электронной таблице был рассчитан экономический баланс игры: количество дропа монеток с каждого типа врага, количество врагов каждого типа на каждом уровне и стоимости бонусов на каждом уровне. В игре создана «инфляция» — каждый следующий уровень стоимость бонусов повышается. Стоимость бонусов рассчитана относительно дропа с разных типов врагов с применением дополнительных множителей от 2 до 3. У бонусов урона и здоровья стоимость всегда одинаковая, у магнитика (бонус дропа) — выше, так как он позволяет собирать больше монет.

Выкупить всё, кроме самого магнитика, возможно и без него, но это будет медленнее. В случае покупки магнитика на 1-2 уровне игры возможно выкупить все бонусы. В случае, если купить магнитик на 1 уровне и все остальные бонусы на последнем по максимальной стоимости — у игрока останется 25 монет. Но это маловероятный сценарий, так как сложность прохождения повышается с каждым уровнем.

Разработка проекта выполнялась итеративно. Каждое нововведение тестировалось на отдельной тестовой сцене перед добавлением на основные уровни. После создания

основных механик игрока (классы PlayerController, PlayerHealth и PlayerAttack), врагов (классы NewEnemyAI, EnemyHealth и EnemyAttack) и интерактивных элементов (MoonlightLogic, SpikesDamage, CheckpointLogic) для каждого уровня был создан graybox-прототип, заполнен врагами и интерактивными элементами. Проведено первичное тестирование и выявлены недочеты в расположении платформ, сбалансирована скорость врагов и игрока, настроены маршруты патрулирования врагов.

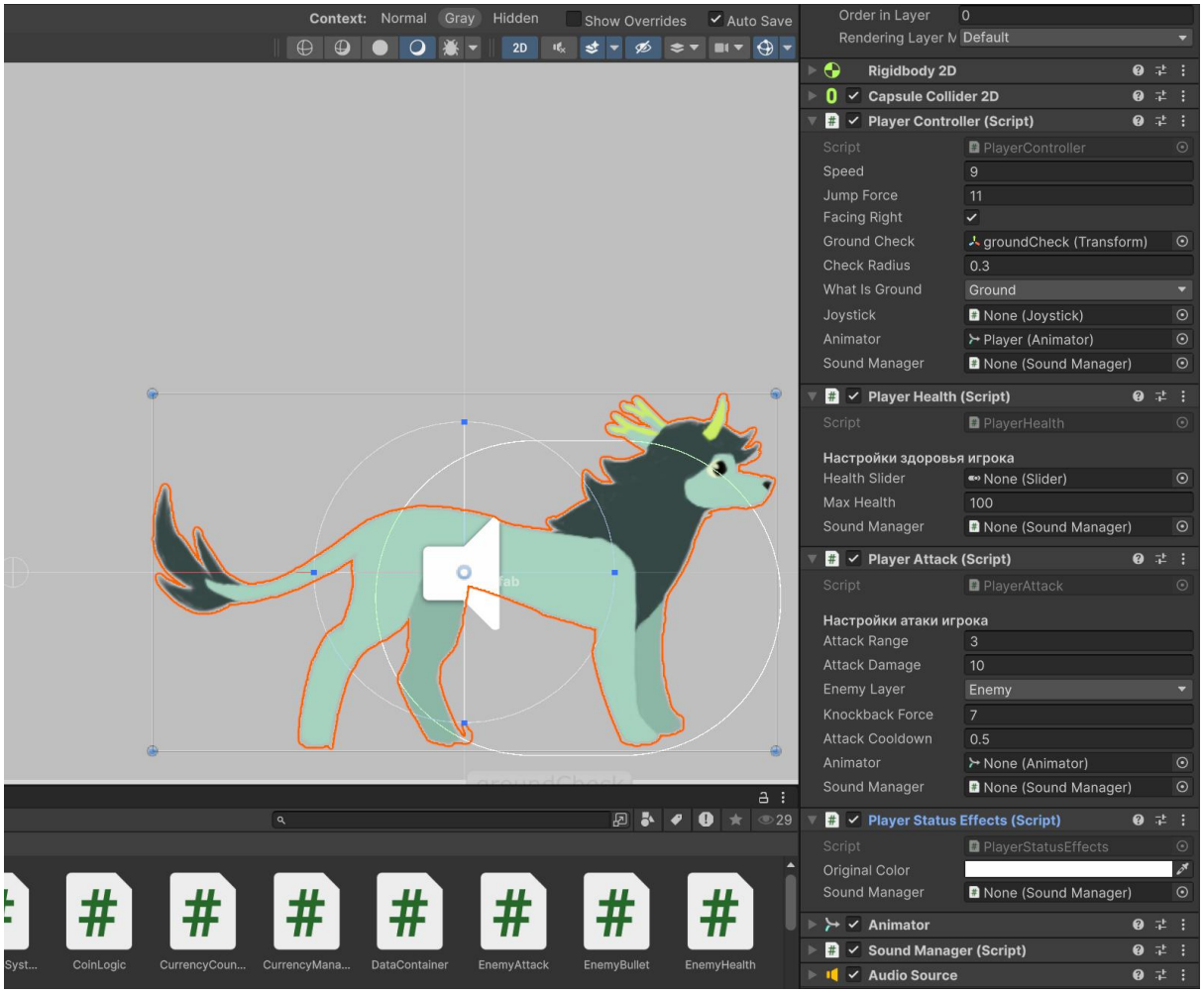


Рис. 4. Структура объекта игрока в редакторе Unity.

Передвижение игрока выполнено с помощью джойстика FixedJoystick из Unity Asset Store и кнопок (прыжок и атака). Джойстик, кнопки управления, а также кнопки паузы, магазина, счётчик монет и слайдер здоровья игрока находятся на UI элементе Canvas, привязанном к камере игрока. Камера игрока имеет один скрипт CamMove, позволяющем ей следовать за игроком.

Отдельное внимание стоит уделить скрипту интеллекта врагов, NewEnemyAI. Он разделяет поведение врага на три типа поведения, между которыми враг переключается в зависимости от ситуации: патрулирование, преследование и поиск пути.

Изначально врагу задаются две точки, начальная и конечная, в рамках которых он перемещается от одной точки к другой, патрулируя свой участок платформы. Если враг в определённом радиусе от себя обнаруживает игрока, то он переключается в режим преследования — перемещается (в том числе и прыгает по платформам) за игроком и пытается его атаковать. Скорость варга в режиме преследования выше, чем скорость при патрулировании.

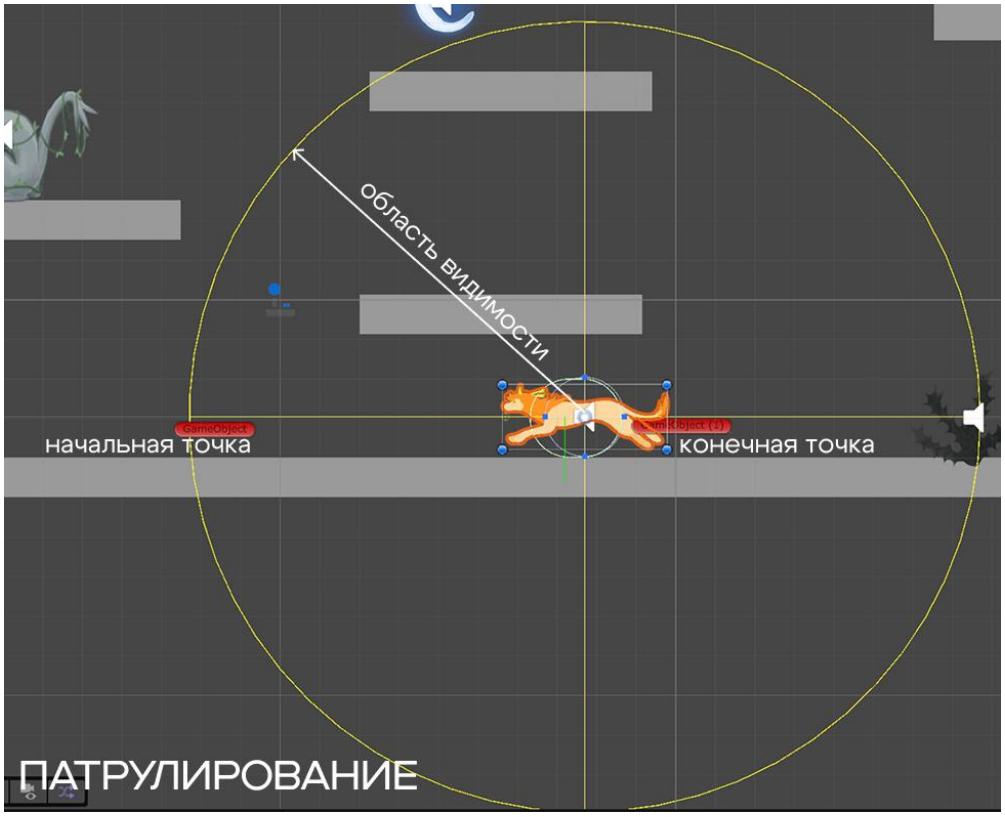


Рис. 4. Режим патрулирования врага в редакторе Unity.

Поиск пути — режим, при котором враг, обнаружив игрока под своей платформой, ищет ближайший верхний вертекс коллайдера своей платформы, не закрытый стеной, и движется к нему, чтобы спрыгнуть вниз, к игроку. Если свободного пути нет — враг снова переключается в режим патрулирования через несколько секунд.

Рис. 5. Режим преследования врага в редакторе Unity.

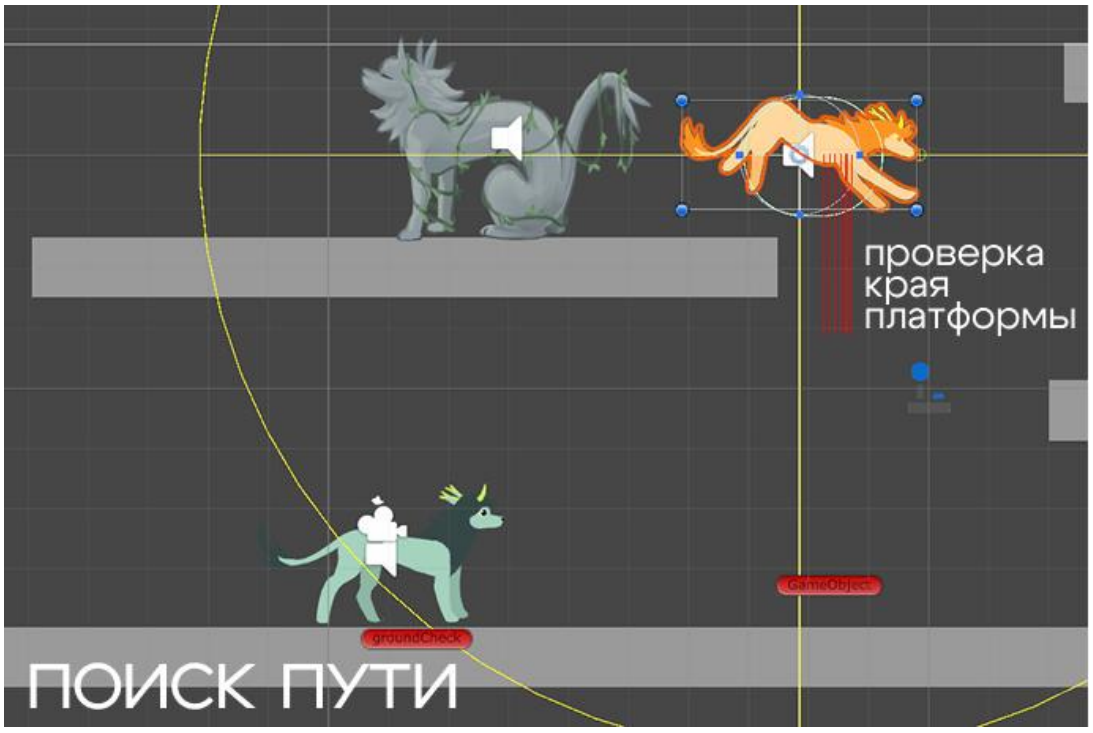


Рис. 5. Режим поиска пути врага в редакторе Unity.

Данная логика позволила сделать врагов динамичными: они умеют прыгать на платформы и с них, искать кратчайший путь вниз и преследовать игрока в поле своего зрения. Помимо описанных задач, скрипт позволяет выбрать тип врага из четырёх: слабый, нормальный, сильный и босс (и один дополнительный тип Iyu для NPC жены главного героя, появляющейся в конце игры). Другие классы, такие как EnemyHealth или EnemyAttack, берут информацию о типе врага из этого скрипта управления и присваивают для каждого типа врага свои заранее указанные показатели урона и здоровья, определяют возможность особенной атаки или супер-атаки для босса.

Далее был разработан и внедрен магазин (классы ShopLogic, ShopItem) и система дропа монет с врагов (классы CoinLogic, CoinDropSystem, CurrencyManager), добавлена система бонусов игрока (класс PlayerBonusSystem).

Для дропа монет был создан префаб монеты Coin, к которому был прикреплен класс CoinLogic, отвечающий за возможность подбора монеты игроком. К префабу каждого типа врага кроме босса был добавлен класс CoinDropSystem. Данный скрипт отвечает за количество дропа для каждого типа врага, учитывая при этом наличие или отсутствие бонуса дропа у игрока. Информация о сумме монет игрока и функции операций с ними содержатся в статическом классе CurrencyManager.

Логика самого магазина реализована с помощью UI элемента Canvas, за возможность открыть и закрыть который отвечает скрипт ShopLogic. При открытии магазина время в игре останавливается до тех пор, пока магазин не будет закрыт. Каждый

бонус в магазине имеет своё название, стоимость, количество и описание. Эти параметры задаёт класс ShopItem, привязанный к каждому из трёх объектов бонусов на Canvas магазина.

Купленные бонусы учитываются в статическом классе PlayerBonusSystem. Он представляет из себя простой счётчик для каждого типа бонуса. Информация из этого скрипта поступает в скрипты PlayerHealth, PlayerAttack и CoinDropSystem для добавления бонусов к соответствующим параметрам. В свою очередь, скрипт ShopItem передаёт информацию о купленных бонусах в PlayerBonusSystem.

Далее была создана система сохранения информации об убитых врагах на уровне и полученных монетах и бонусах, реализована система чекпоинтов (статический класс DataContainer для обработки данных чекпоинтов и сохранений).

Каждый объект чекпоинта имеет класс CheckpoinLogic, отвечающий за срабатывание логики чекпоинта при столкновении коллайдеров игрока и чекпоинта. У каждого чекпоинта на каждом уровне имеется свой уникальный индекс от 1 до 7, позволяющий возрождать игрока при смерти на конкретной позиции на уровне. Чекпоинт срабатывает только один раз, повторно сохранится на одном и том же чекпоинте нельзя.

Статический класс DataContainer служит памятью игры. Он содержит в себе временные и действительные списки убитых врагов, собранных частиц луны, количество денег игрока и его купленные бонусы. Во временные списки и переменные записываются данные до достижения чекпоинта. Если чекпоинт не будет взят, то временные данные просто очистятся. При достижении чекпоинта временные данные присваиваются действивтельным. Действительные данные используются при возрождении игрока: они позволяют не активировать врагов и частицы луны, убитых и собранных до чекпоинта, и запоминать, сколько у игрока было денег и бонусов на момент взятия чекпоинта.

Также DataContainer содержит информацию о индексе проходимого уровня и имеет функции сохранения, загрузки и очистки данных игрока в PlayerPrefs. Первые две функции вызываются в меню игры при запуске или по окончании уровня. Функция очистки доступна к вызову при нажатии кнопки в настройках и позволяет начать игру полностью заново после перезагрузки игры.

При первом столкновении CheckpoinLogic активирует функцию UpdateChekpoint, обновляя текущий индекс чекпоинта и действительные списки и переменные в DataContainer (именно по текущему индексу определяется место возрождения).

Затем было создано главное меню игры и выбор уровней (класс MainMenu). Меню представляет из себя сцену с двумя Canvas: один для главного экрана и второй для меню выбора уровня. Доступные уровни определяются по индексу уровня из DataContainer.

Игра не позволяет перепроходить уровень повторно после полного его прохождения в связи с линейной связанностью уровней. Все уровни открываются только после полного прохождения игры и доступны для посещения игроком.

Далее было проведено повторное тестирование, направленное уже на балансировку боевой системы и бонусов, цен в магазине и расположения и работы чекпоинтов.

Затем для игры были отрисованы спрайты всех персонажей и предметов в программе Krita, созданы анимации игрока и врагов. За основу всех персонажей игры был взят спрайт-лист главного персонажа игры. Для каждого типа врагов он был перекрашен в свой цвет.

Анимирование персонажей выполнялось с помощью компонента Animator в Unity. Из спрайт-листа было создано 5 анимаций: бег, прыжок вверх, падение вниз, атака, спокойное состояние. Переключение между анимациями осуществляется через переменные типа bool (jumping, isAttacking) и float (horizontalMove, verticalMove), отслеживающих состояние персонажа. Для некоторых анимаций были созданы события: событие воспроизведения звука передвижения для анимации бега, а для атаки — прекращение атаки после 1 проигрывания анимации.

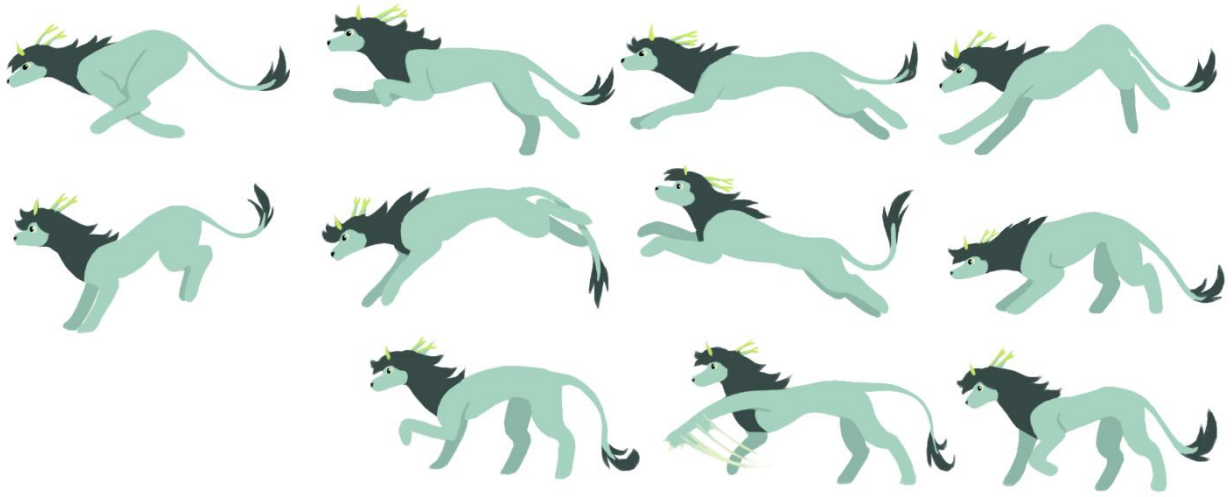


Рис. 4. Спрайт-лист главного персонажа игры.

В программе Figma были созданы UI элементы для кнопок меню игры. Добавлены звуки действий персонажей, интерактивных предметов, фоновая музыка уровней и меню. Звуки для действий персонажей помещались в специальный класс SoundManager, который прикреплялся к префабам врагов и игрока. В классе содержится список звуков и метод проигрывания звука из списка по индексу. Данный метод вызывался в скриптах передвижения и атаки персонажей для воспроизведения необходимых звуков в момент совершения действий.

После этого игра была протестирована полностью на компьютере внутри редактора Unity, затем создан первый билд и отправлен 4 пользователям андроид-смартфонов для тестирования.

По результатам тестирования игроками были найдены важные недочеты в системе сохранений и работе магазина. Они были успешно исправлены, новый билд снова прошёл тестирование и по его результатам больше не было найдено ошибок. Разработка была окончена.

5. Результаты проекта

В результате работы над проектом была получена готовая игра для андроид-смартфонов в жанре 2D-платформер. Игроки, проходившие игру при тестировании, отметили интересное визуальное решение и игровой процесс, который заставлял их подбирать тактику для прохождения игры и пробовать проходить её несколько раз с разными вариантами покупки бонусов.

Участниками тестирования отмечалось также и визуальное решение игры: оно создавало приятные эмоции во время игрового процесса, положительно влияя на общее впечатление от игры.

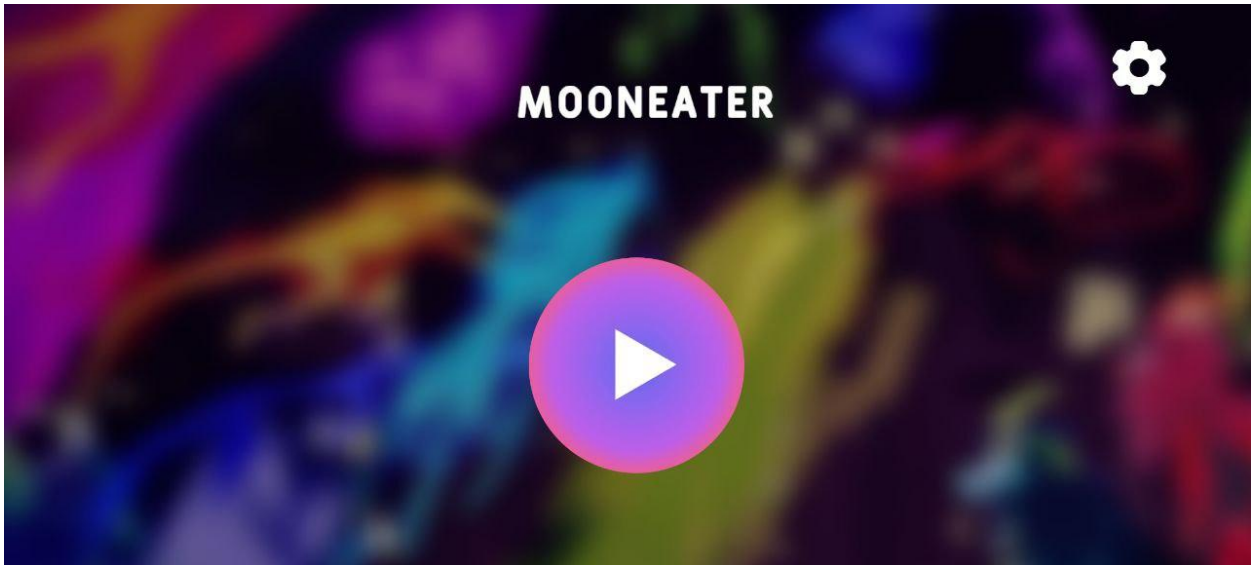
Готовая игра стала полноценным продуктом, хорошо показала себя на Android-устройствах разных производителей. Проблем с совместимостью и производительностью обнаружено не было.

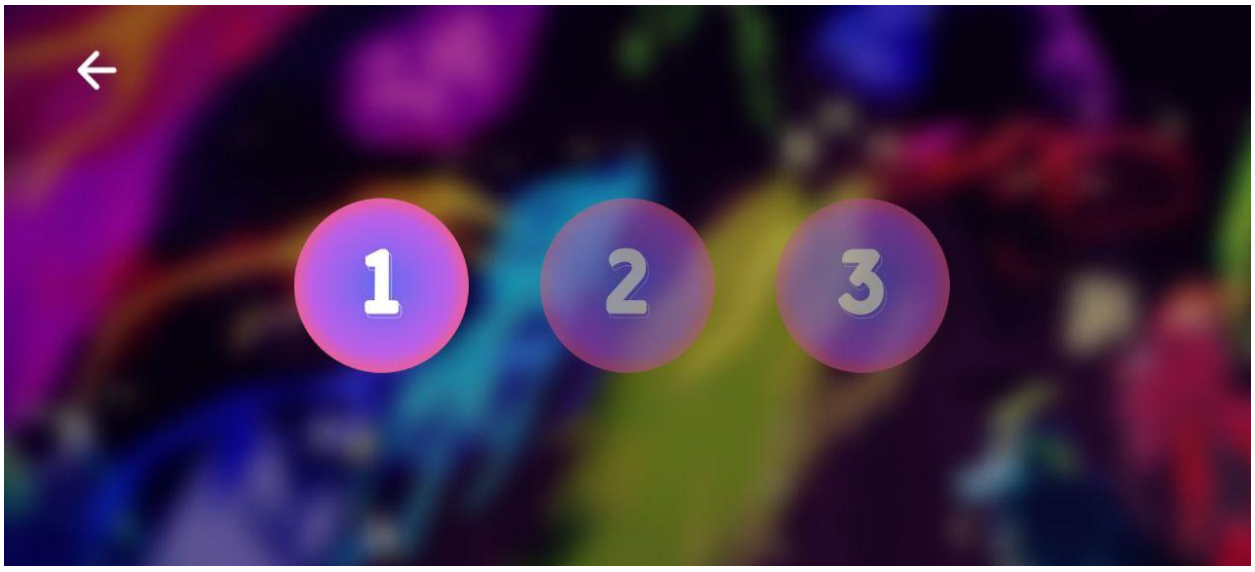
Цель проекта была достигнута — платформер Mooneater был реализован в полной мере: созданы механики игрока, окружения и интеллект врагов, есть действующая система развития навыков персонажа по ходу игры.

У проекта есть точки роста, работа в направлении которых позволит развить небольшую игру в полномасштабный проект. Основные механики уже созданы, но участники тестирования отметили нехватку сюжетной составляющей внутри самой игры. В будущем в игру необходимо будет внедрить механику внутриигровых диалогов и/или квестов, которые бы разнообразили геймплей и дали игрокам почву для более глубоких размышлений о нарративе игры.

Приложение 1

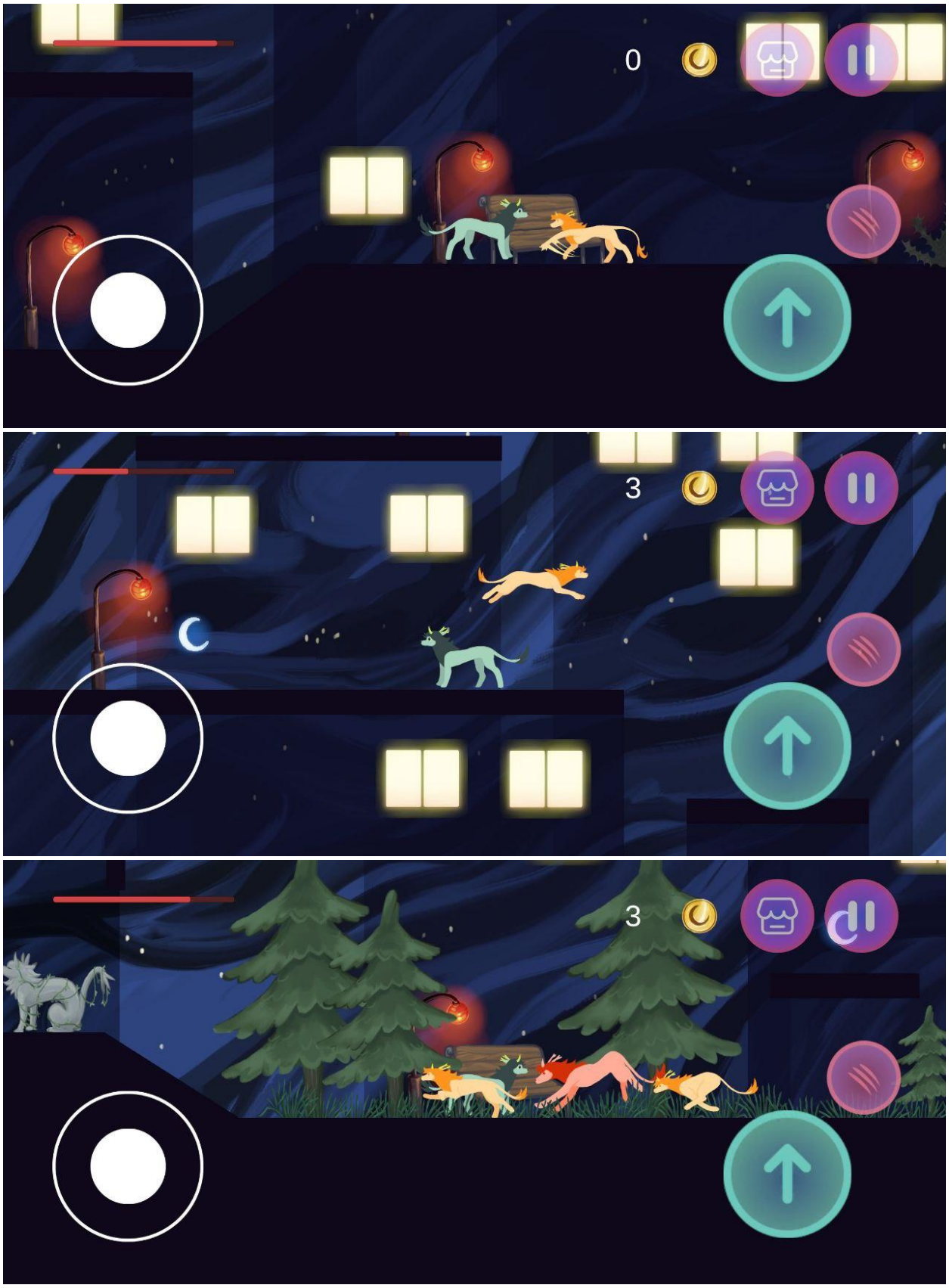
Скриншоты, демонстрирующие работу созданного программного продукта.



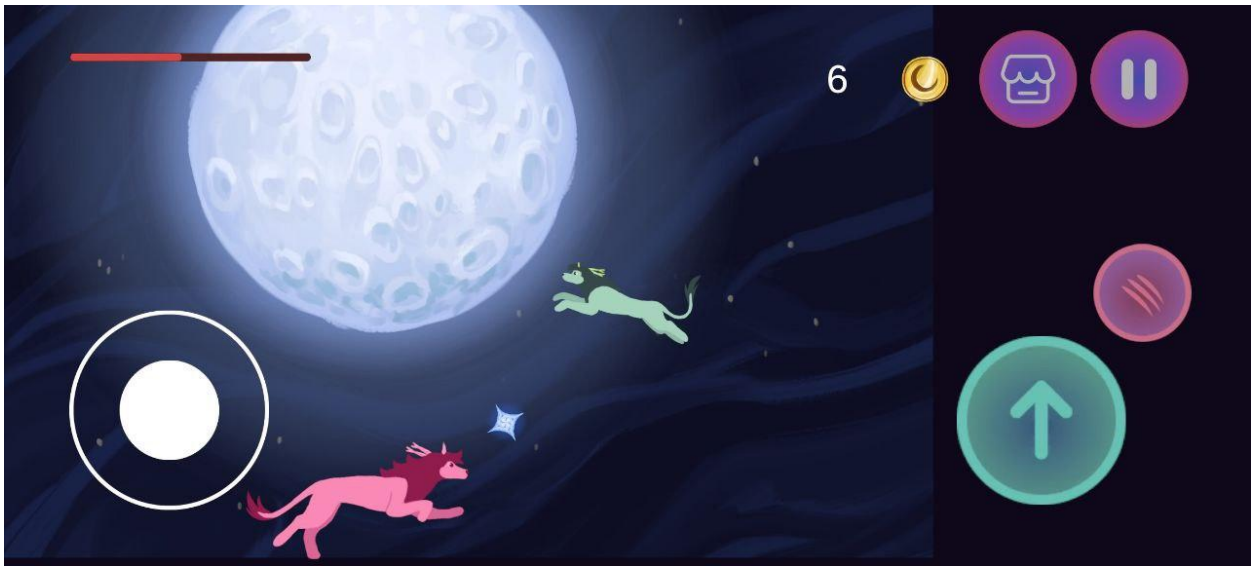


Главное меню игры.

Внутриигровой магазин бонусов.



Игровой процесс и UI-интерфейс игрока.



Финальное сражение с боссом игры.

Приложение 2

Программный код.

Все программные файлы игры находятся в открытом репозитории в папке Scripts.

Ссылка на репозиторий: https://github.com/yawninkitty/Mooneater

Приложение 3

Презентация проекта находится в облачном хранилище по ссылке:
https://drive.google.com/file/d/1fnUjAkfb2sBpJ4IKHfbxyhKM1XEqqMMQ/view?usp=sharing