Министерство образования и науки Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего образования
«Российский государственный профессионально-педагогический университет»

2D-ПЛАТФОРМЕР С ПРОЦЕДУРНОЙ ГЕНЕРАЦИЕЙ УРОВНЕЙ

Выпускная квалификационная работа

по направлению подготовки 09.03.02 Информационные системы и технологии

профилю подготовки «Информационные технологии в медиаиндустрии»

Индификационный номер ВКР: 120

Екатеринбург 2018

Министерство образования и науки Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего образования
«Российский государственный профессионально-педагогический университет»
Институт инженерно-педагогического образования
Кафедра информационных систем и технологий

К ЗАЩИТЕ ДОПУСКАЮ
Заведующая кафедрой ИС
_____Н. С. Толстова
«__»_____2018г.

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

2D-ПЛАТФОРМЕР С ПРОЦЕДУРНОЙ ГЕНЕРАЦИЕЙ УРОВНЕЙ

Исполнитель:
обучающаяся группы ИТм-402 Д. Р. Ганиева
Руководитель:
ст. преподаватель И. А. Садчиков
Нормоконтролер: Н. В. Хохлова

Екатеринбург 2018

АННОТАЦИЯ

Выпускная квалификационная работа состоит из игры «Be Light» и пояснительной записки на 67 страницах, содержащей 34 рисунков, 2 таблиц, 30 источников литературы, а также 5 приложений на 15 страницах.

Ключевые слова: ИГРА, КОМПЬЮТЕРНАЯ ИГРА, ВИДЕОИГРЫ, РАЗРАБОТКА, КОНЦЕПЦИЯ, UNITY, DRAGON BONES.

Ганиева Д. Р. 2D-платформер с процедурной генерацией уровней: выпускная квалификационная работа / Д. Р. Ганиева; Рос. гос. проф.-пед. ун-т, Ин-т инж.-пед. образования, Каф. информ. систем и технологий. — Екатеринбург, 2018. — 67 с.

В работе рассмотрена разработка компьютерный игры.

Целью данной работы является проектирование концепта игры и ее разработка.

Для достижения цели были проанализированы специализированные литературные и интернет-источники по вопросам разработки компьютерных игр, выбран жанр для разрабатываемого проекта, изучены похожие, уже существующие разработки, создана концепция игры, изучены методы работы с соответствующим программным обеспечением.

Основываясь на данных материалах, была разработана компьютерная игра в жанре платформер с процедурной генерацией уровней.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Научно-технический прогресс, набравший к концу XX века головокружительную скорость, послужил причиной появления такого чуда современности как компьютер и компьютерные технологии. Изобретение компьютеров послужило переломным моментом в развитии многих отраслей промышленности. Вместе с появлением компьютеров появились компьютерные игры, которые сразу же нашли массу поклонников.

С совершенствованием компьютеров совершенствовались и игры, привлекая все больше и больше людей. Особенным успехом компьютерные игры пользуются у школьников младших и средних классов. На сегодняшний день компьютерная техника достигла такого уровня развития, что позволяет программистам разрабатывать очень реалистические игры с хорошим графическим и звуковым оформлением.

В рамках дипломной работы разрабатывалась игра в жанре платформер с процедурной генерацией уровней. Для этого изучено специализированное программное обеспечение (ПО) и разработан собственный продукт с нуля.

Объект исследования — компьютерная игра.

Предмет исследования — методы проектирования и разработки компьютерной игры.

Цель — художественно-технически спроектировать концепт игры и реализовать его с помощью специализированного программного обеспечения.

Задачи:

- проанализировать литературу и интернет-источники по вопросам разработки компьютерных игр;
- выбрать жанр для разрабатываемого проекта;
- изучить похожие существующие разработки;
- создать концепт игры;
- изучить методы работы с соответствующим ПО;

- разработать продукт.

Компьютерная игра — программа, написанная на одном из языков программирования, с использованием движка, созданная для развлечения.

Компьютерные игры можно классифицировать по разным аспектам, рассмотрим самый важный из них: жанр (таблица 1).

Таблица 1 — Классификация компьютерных игр по жанрам

Название	**Описание**	**Примеры игр**
Шутеры	Игры, главной механикой которых является стрельба.	Call of Duty, Battlefield, Counter-Strike 1.6
Аркада	Игры, в которых игроку необходмио быстро выполнять какие-либо действия(стрелять, бегать, собирать очки и др.), как правило, не имеют сюжета, механика игры проста и интуитивно понятна.	Plant vs Zombies, Pacman, Bomberman
RPG	Основная цель игрового процесса это отыгрывание определенной роли. Имеют проработанный сюжет.	Fallout, Dark Souls, Assasin’s creed
Гонки	Соревнование на определенном виде техники, обычно гоночные автомобили.	Colin McRae Dirt 2, Need for Speed, Test Drive Unlimited
Файтинг	Игры имитирующие драки, рукопашные бои.	Mortal Kombat, Street Fighter IV, Tekken
Платформер	Игроку нужно передвигаться по миру (обычно происходит в 2D пространстве) прыгать, уклоняться от ловушек.	Super Meat Boy, Super Mario, Sonic the Hedgehog
Симулятор	Максимально приближенное к реальности управление какой-либо техникой.	Ил-2 Штурмовик, Silent Hunter 3, Microsoft Flight Simulator
Стратегии	Жанр компьютерных игр, в котором от игрока требуется планировать, управлять ресурсами, разрабатывать стратегии в соответствии с меняющейся обстановкой.	Warcraft III, Heroes of Might and Magic III, Civilization IV
Текстовые	Решение загадок с помощью текстовых команд.	Adamant MUD, Arctic MUD, RMUD
Квест	Игроку предлагают головоломку, загадку, которую игрок должен решить, как правило, игры такого жанра имеют линейный сюжет.	World of Goo, Braid, Crazy Machines
Спортивная	Главный аспект этого жанра — соревнование. Обычно представляют собой спортивные игры и соревнования, перенесенные в виртуальную реальность.	Beidjing 2008, NBA’09, FIFA’09

1 АНАЛИТИЧЕСКАЯ ЧАСТЬ

1.1 Выбор жанра игры с примерами существующих разработок

Прежде чем начать разработку нужно выбрать жанр для будущего проекта, было решено делать игру-платформер с процедурной генерацией уровней. Для того, чтобы сделать игру интересной, нужно ознакомится с существующими разработками похожих игр: выделить интересные механики, плюсы и минусы, узнать мнение людей о них и многое другое.

1.2 Игры в жанре платформер

Платформер (англ. platformer) — жанр компьютерных игр, в которых основной чертой игрового процесса является прыгание по платформам, лазанье по лестницам, собирание предметов, обычно необходимых для завершения уровня. Далее приведены примеры игр данного жанра.

1.2.1 Ori in the blind forest

Очень атмосферная игра с красивой графикой (рисунок 1), прекрасным звуковым сопровождением и интересными персонажами, которым хочется сопереживать. Она сделана в жанре метроидвания, что подразумевает немного расширенную версию классического платформера: игрок «развивается» получая новые способности, которые позволяют добраться до ранее недоступных мест на карте.

Игра имеет интересную систему боя: около главного персонажа летает маленький огонек, который выпускает шарики энергии в ближайшего монстра на расстоянии. Такое «самонаведение» действует в некотором радиусе от игрока.

Рисунок 1 — Заставка игры Ori in The Blind forest

Персонажи созданы и анимированы с помощью 3D технологий, однако весь мир нарисован в 2D и анимирован с помощью системы «костей», а не классической покадровой анимации.

Интерфейс во время самого игрового процесса представляет собой индикатор жизней (зеленые кружочки справа) и индикатор маны (синие кружочки слева), большой круг по середине является индикатором опыта, который пополняется от собранных в игре бонусов и предоставляет игроку выбирать новые умения (рисунок 2). Цифра внутри круга показывает количество имеющихся на данный момент полных кругов опыта.

Сначала игрок может только бегать, прыгать и стрелять, однако потом у него появляются и другие умения, например: карабканье по стенам, планирование в воздухе, сильный «топот», мощный удар энергией и др.

Очень отзывчивое управление, оно интуитивно понятно. Используется компьютерная мышь (обе клавиши) и кнопки клавиатуры wasd.

Монстры так же разнообразны, почти все реагируют на присутствие игрока поблизости: пауки начинают стрелять шарикам, лягушки прыгают и пытаются раздавить, однако ядовитые колючки неподвижны и наносят урон

только при соприкосновении. Игра постоянно преподносит новых врагов с уникальными способностями.

Рисунок 2 — Скриншот из игры Ori in The Blind forest

После смерти игрок появляется в месте последнего сохранения, которые он может создавать сам, используя ману.

Очень насыщенная на разнообразные события игра.

1.2.2 Hollow knight

Мрачноватая игра в жанре метроидвания, где игроку предлагается исследовать заброшенный город жуков, помогать встреченным на пути персонажам (рисунок 3).

Красивая 2D графика в мультяшном стиле. Выдержана в черном, белом и серых цветах для придания атмосферы. Интересный сюжет, который раскрывается с помощью разговоров с персонажами в игре.

В этой игре предлагается только система ближнего боя, главный герой атакует своеобразным мечем, в качестве которого выступает гвоздь. Интересно то, что герой с помощью атак бить не только врагов и специальные

«двери», а также взаимодействовать с окружающей средой, ломая камни, статуи, срезая высокую траву, что не дает преимуществ, но скрашивает довольно пустые локации игры.

Рисунок 3 — Заставка игры Hollow Knight.

Управление не особо удобное для тех, кто привык к мыши и клавиатуре, осуществляется оно с помощью стрелок и кнопок «z», «x», «a» на английской раскладке клавиатуры.

Враги наносят урон только при соприкосновении с игроком, однако почти все реагируют на его присутствие: начинают двигаться в сторону игрока или даже ускоряться. Когда наносится удар время для игрока замирает, и он отпрыгивает назад, во время этих ударов нельзя управлять игроком, что порой приводит к плохим последствиям. После смерти игрок появляется в последней зоне отдыха, а на месте гибели остается злой призрак.

Враги не очень разнообразны, однако в игре есть много уникальных и очень сложных боссов.

Интересно то, что по карте разбросаны разные персонажи, с которыми можно взаимодействовать: купить карту и другие предметы, поговорить, а некоторым нужно помочь найти выход из лабиринта подземных ходов.

По звукам в игре можно определять близко ли находятся уникальные персонажи.

Интерфейс представляет собой большую колбу, в которую попадают частицы «души», которые можно потратить на восстановление жизней (рисунок 4).

Рисунок 4 — Скриншот из игры Hollow Knight

Маленькие белые значки — это жизни, и чуть ниже индикатор монет, которые вы можете обменять на полезные вещи или улучшение предметов.

1.2.3 Never alone

Never Alone это игра — платформер с элементами головоломок (рисунок 5). Создана на основе мифологии коренных народов Аляски. В игре так же можно ознакомиться с некоторыми из историй, которые будут открываться во время прохождения игры.

Несмотря на то, что эта игра является платформером и перемещение в ней возможно лишь в 2D плоскости, выполнена она в 3D графике (рисунок 6).

Рисунок 5 — Заставка игры Never Alone.

Главных персонажей двое: девочка и песец, между ними можно легко переключаться, и пока игрок управляет одним персонажем, второй следует за ним. Герои имеют разные способности: песец может вызывать духов-помощников, а девочка может толкать ящики. Однако искусственный интеллект персонажа, неуправляемого игроком в данный момент, часто приводит к его гибели, что является ошибкой разработчиков.

Рисунок 6 — Скриншот из игры Never Alone

Существуют довольно пустые зоны, без каких-либо головоломок или других активностей.

Никаких индикаторов во время игры нет.

1.3 Игры с процедурной генерацией

Процедурная генерация контента является одним из наиболее актуальных и активно развивающихся направлений исследований в сфере мультимедиа, в частности в индустрии видеоигр. Под процедурной генерацией контента понимают автоматическое и полуавтоматическое создание и динамическое изменение различных составляющих частей игр, в том числе игровых объектов и уровней, двумерной и трехмерной графики, эффектов, звуков, музыки, персонажей, сюжетов и др.

Процедурная генерация предоставляет ряд преимуществ по сравнению с «ручной» настройкой уровней:

- разнообразие: такой мир не будет повторяться, каждый раз перед игроками появляются все новые виды проходов и переходов;
- скорость разработки: разработчику не нужно расставлять все блоки уровней и придумывать креативные извилистые ходы, достаточно описать алгоритм для создания объектов, и игра сама будет расставлять все по местам.

Рассмотрим некоторые игры, в которых присутствует процедурная генерация.

1.3.1 Shadow of mordor

Процедурная генерация в этой игре касается внешности и взаимодействий орков. Каждый орк уникален, так как все характеристики генерируются игрой самостоятельно (рисунок 7). Внешность, манера речи, характер и даже имя — все создается по алгоритму.

Рисунок 7 — Скриншот игры Shadow of Mordor

Так же орки могут получать звания за убийства игроков, становиться сильнее, обучаясь новым навыкам. После сражений они получают уникальные шрамы. Взаимодействия орков так же сильно влияют на сюжет. Игроки могут манипулировать взаимоотношениями орков, чтобы повернуть ситуацию в выгодное положение. Процедурная генерация врагов в этой игре позволяет каждому юниту быть уникальным. Это придает реалистичности игре, ведь ни в одном подобном войске не должно быть армии одинаковых клонов с однотипным поведением.

1.3.2 Spelunky

Эта игра (рисунок 8) генерирует каждый уровень самостоятельно за пару секунд до начала игровой сессии. Помимо мира и расстановки врагов рандомизации подвержены и механики игры: правила могут изменяться по ходу прохождения: можно получить новые способности, которые выбираются случайным образом, выкуп для перехода на следующий уровень каждый

раз меняется. Это делает игру непредсказуемой для игрока: никогда не знаешь, что ждет тебя дальше.

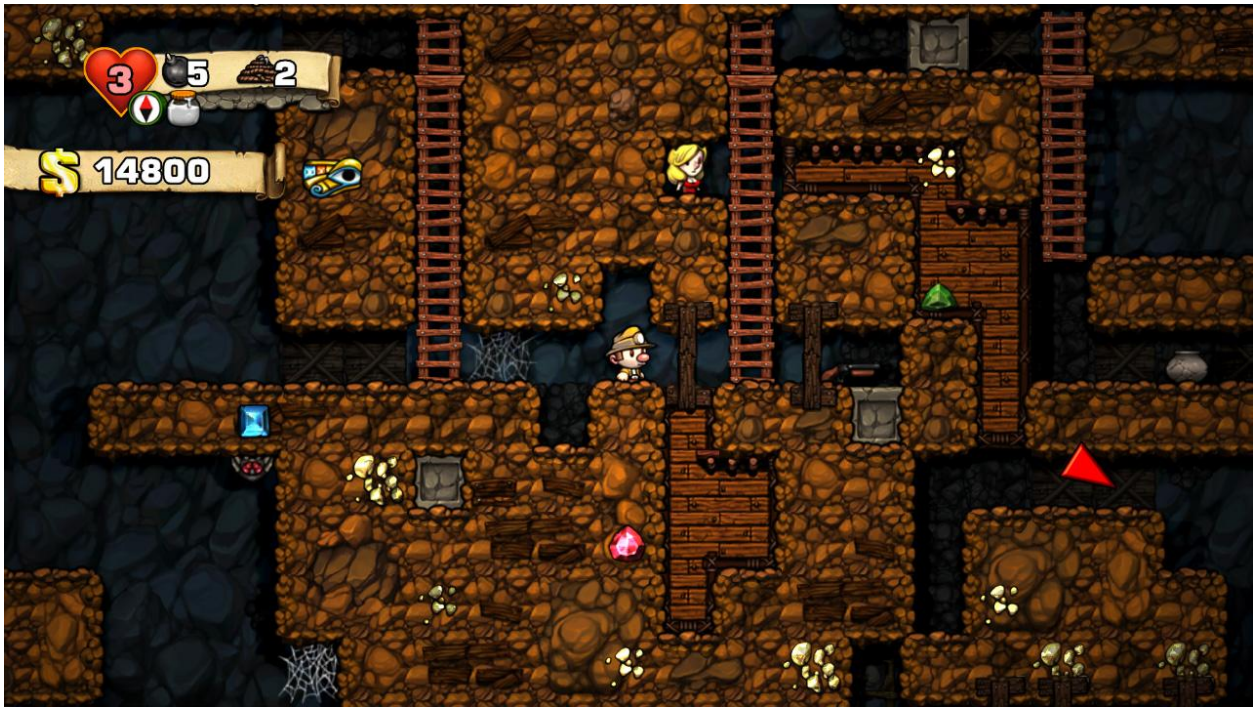

Рисунок 8 — Скриншот из игры spelunky

Когда игрок открывает новые локации, ловушки, врагов и другие предметы, они записываются в особый дневник, который является своеобразной «напоминалкой» о характеристиках и поведении тех или иных вещей и монстров.

Процедурная генерация в spelunky располагает игрока к изучению всех препятствий и возможностей, встречающихся на пути.

На игровом интерфейсе можно увидеть количество жизней, имеющихся предметов и золота, которое можно обменять на предметы в магазинах на уровне.

1.3.3 Rogue legacy

Rogue Legacy это платформер с процедурной генерацией мира (рисунок 9). Играем мы за воина, который пытается спасти своего короля из замка, однако если персонаж умирает, мы идем заново в этот же замок, но только с

одним из его потомков. Получается, что этот замок вместе с вами штурмует не первое поколение рыцарей династии.

Рисунок 9 — Скриншот из игры Rogue Legacy

Однако каждому из последующих героев достается все имущество предыдущего, что позволяет улучшать вещи, докупать предметы, необходимые в бою.

Бой преимущественно ближний, однако есть особые возможности у некоторых классов персонажа, позволяющие метать клинки на дальние расстояния, пользоваться магией и др.

Процедурная генерация в этой игре делает ее непредсказуемой и сложной для первых «поколений» героев.

1.4 Другие рассмотренные игры

Помимо игр в жанре платформер и игр с процедурной генерацией, следует отметить и другие игры с похожими механиками, например, игры для мобильных устройств в жанре «раннер». Раннер — это жанр, являющийся разновидностью платформера, особенностью которого является постоянное

движение по горизонтали, до того момента, пока игрок не погибает, теоретически уровень можно назвать бесконечным.

Dark lands

Dark lands мобильная игра в жанре раннер (рисунок 10), в которой игроку нужно бесконечно преодолевать препятствия, убивать монстров и различных боссов, собирая по пути разны бонусы.

Рисунок 10 — Заставка игры Dark lands

В игре использована векторная графика, все кроме фона выдержано в черно-белых тонах (рисунок 11).

Игра сама генерирует мир по мере прохождения игры.

Главный герой сам бежит вправо, его можно заставить прыгнуть, пригнуться, остановиться, атаковать и прикрыться щитом.

Мобы разнообразны, имеют разные способности, как и боссы.

Существует внутриигровой магазин, в котором можно улучшить предметы за найденные в игре бонусы или купить их за реальные деньги.

Рисунок 11 — Скриншот из игры Dark lands

Интерфейс прост: справа наверху указана полоска жизней, рядом с ней пройденное расстояние с начала игры, справа кнопка меню паузы.

1.5 Выбор программного обеспечения для реализации проекта

Среди многочисленного ПО, нужно выбрать наиболее подходящие для разработки проекта. В приоритете программы со свободной лицензией, широким функционалом, удобством интерфейса и возможностью экспортировать файлы в нужное расширение.

1.5.1 Игровой движок

Игровой движок — это центральный программный компонент компьютерных и видеоигр или других интерактивных приложений с графикой, обрабатываемой в реальном времени. Он обеспечивает основные технологии, упрощает разработку.

Для сравнения были выбраны два популярных движка для профессиональной разработки игр (таблица 2).

Таблица 2 — Сравнительная характеристика некоторых игровых движков

Критерий	**Unity**	**Unreal Engine**
1	2	3
Краткое описание	Межплатформенная среда разработки 2D и 3D компьютерных игр, которая позволяет создавать приложения, работающие под более чем 20 различными операционными системами.	Набор инструментов для разработки игр, имеющий широкие возможности: от создания двухмерных игр на мобильные до AAA-проектов для консолей
Стоимость	Бесплатна для персонального пользования (при выручке за игру до 100 000$ в год). Для коммерческого использования при прибыли от игры до 200 000$ в год оплачивается подписка 35$ в месяц, свыше этого подписка составляет 125$, повышение стоимости подписки открывает дополнительные возможности движка.	Unreal Engine 4 распространяется бесплатно. Однако до тех пор, пока вы не выпустите свой первый коммерческий продукт на основе UE4, далее следует платить процент от продаж Вашей игры
На какие платформы разрабатывается	Ios, android, windows, Windows phone, Mac, linux, webgl, ps4, psvite, xbox one, Wii U, Nintendo 3Ds, Oculus Rift, Google Cardboard Android & ios, steam, Playstation VR, Gear VR, Windows moxed reality, Daydream, Android TV, Samsung smart TV, tvos, Nintendo Switch, fireos, Facebook Gameroom, Apple arkit, Google arcore, Vuforia	Windows PC, Mac, Linux, ios и Android, HTML5. Также есть встроенная поддержка Виртуальной реальности для Oculus Rift. Помимо этого UE4 поддерживает Xbox One и playstation 4 (включая Project Morpheus)
Какого вида игры можно создавать	Игры любого жанра, симуляторы и др.	Игры (2D-3D; RTS, Action-RPG, Shooter, Racing, MMO-игры и любой другой жанр и направление), симуляторы и даже программное обеспечение. Вы можете использовать UE4 для архитектурной визуализации и многое другого

Окончание таблицы 2

1	2	3
Языки программирования	C#, javascript, Boo (убрали из новой версии движка)	C++ и Blueprint (собственный визуальный язык программирования)
Интерфейс	Имеет простой Drag&Drop интерфейс	Нельзя перетаскивать объекты прямо в окно редактора.
Открытость кода	Закрыт	Благодаря открытому исходному коду, вы можете самостоятельно добавить поддержку дополнительных устройств, или же оптимизировать/дополнить уже существующие.
Обучающие ресурсы	Присутствуют на официальном сайте	Присутствуют на официальном сайте
Документация	Документация понятна, но переведена на русский язык только частично	Исчерпывающая, но только на английском языке
Плагины	Можно писать в самом движке, так же можно выбрать из множества в магазине	Пишутся только на c++, маленькое количество доступных плагинов
Графика	Реалистичная хорошая графика	В новых версиях движка графика тоже на высоком уровне, однако еще не достает до UE
Общее впечатление	Профессиональная среда разработки игр, с удобным приятным интерфейсом	Профессиональная среда разработки игр, имеющая хорошую графику, но сложный и запутанный интерфейс

После проведенного исследования, мною был выбран игровой движок Unity 2018.

1.5.2 Графические редакторы

В данном проекте использовались следующие графические редакторы:

1. Paint Tool SAI — растровый графический редактор, с удобным функционалом для цифрового рисования. Использовался для рисования скетчей, фонов, платформ и других элементов окружения.

2. Corel Draw — векторный графический редактор. Использовался для отрисовки элементов интерфейса, персонажей и их деталей и др.

3. Adobe Photoshop — растровый графический редактор с мощным инструментарием для постобработки изображений. Использовался для создания бесшовных текстур, постобработки элементов окружения, автоматизированного сохранения отдельных частей изображений.

1.5.3 Программное обеспечение для анимации

Для создания анимации движений персонажа и монстров в игре использовалась программа Dragon Bones. Dragon Bones — это бесплатная программа для создания скелетной анимации.

1.5.4 Другие программы и плагины

Во время проекта использовался плагин Sprite Packer для Unity. Это бесплатный плагин с официального магазина Unity, который позволяет сохранить отдельные картинки с покадровой анимацией в единый лист спрайтов, что облегчает работу с покадровой анимацией в движке и увеличивает производительность игры.

1.5.5 Общий алгоритм реализации проекта

Алгоритм реализации проекта включает в себя следующие шаги:

1. Создание проекта, настройка системы контроля версий (GitHub).
2. Написание скриптов поведения персонажа, камеры.
3. Разработка и написание алгоритма процедурной генерации платформ.
4. Написание скриптов поведения врагов первого уровня.

5. Отрисовка платформ, фонов, бонусов первого уровня, создание их префабов.

6. Написание скриптов движения фонов, поведения бонусов.

7. Отрисовка персонажей первого уровня, их анимирование и настройка в игровом движке.

8. Настройка начала локации первого уровня и обучения.

9. Написание скриптов генерации монстров и бонусов во время игры.

10. Отрисовка, написание кода и настройка интерфейса.

11. Настройка игрового менеджера и перехода на следующий уровень.

12. Оптимизация проекта: создание и настройка пула объектов.

13. Создание игровых меню.

14. Отрисовка платформ второго уровня, создание их префабов.

15. Написание алгоритма генерации платформ второго уровня.

16. Написание скриптов поведения персонажа и врагов второго уровня.

17. Отрисовка персонажей второго уровня, их анимирование и настройка в игровом движке.

18. Настройка начала локации второго уровня и обучения.

19. Корректировка скриптов генерации монстров и бонусов, для большей универсальности.

20. Создание визуальных эффектов интерфейса.

21. Отрисовка объектов третьего уровня, создание их префабов.

22. Настройка начала локации третьего уровня и обучения.

23. Написание алгоритма генерации объектов третьего уровня.

24. Корректировка скриптов поведения персонажа на 3ем уровне и настройка его префаба.

25. Настройка начального и конечного экрана игры.

26. Добавление звуков и музыки.

27. Настройка баланса игры.

28. Уменьшение размера проекта и его компиляция.

2 ПРОЕКТНАЯ ЧАСТЬ

2.1 Актуальность проекта

Компьютерные игры стали неотъемлемой частью современного мира. Множество детей и подростков проводят за ними свое свободное время. Спрос на такой вид развлечения огромен, следовательно, рынок нуждается в интересных разработках в данном направлении.

Различие механик на разных уровнях, «взросление» персонажа и автоматическая генерация уровней являются главными особенностями данной разработки.

2.2 Характеристика потенциальной аудитории проекта

Целевой аудиторией проекта являются люди, потенциально имеющие следующие характеристики:

- возраст: дети, 8–13 лет;
- пол: в основном девочки, так как им, в основном, нравятся более сказочные и красочные вещи, нежели мальчикам, которых в данном возрасте больше интересует техника;
- интересы: казуальные компьютерные игры, сказки, мультики и др. в жанре фентази;
- психологический тип: исследователи (этот класс ориентирован на знание, на поглощение игрового контента, на изучение игровой механики и возможностей игры), накопители (люди, которых стимулирует внутриигровой рост, прогресс);
- что хотят от проекта: занять свободное время, попасть в интересный, сказочный и красочный мир, знать, как играть в игру.

2.2 Концепция

Далее речь пойдет о главных аспектах разработки игры.

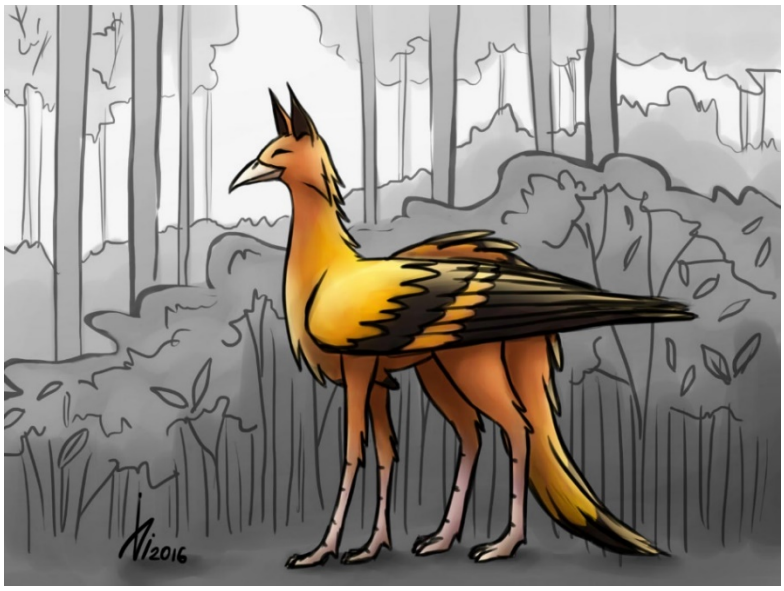

Рисунок 12 — Концепт-арт главного персонажа игры

2.2.1 Название и сюжет

Очень важно создать правильное название, ведь именно от него будет зависеть дальнейшее восприятие игры. Было просмотрено множество материалов по этой теме, и составлен список возможных вариантов, как на основе сюжета:

1. Страж Солнца.
2. Свет жизни.
3. Черное солнце.
4. Вечное затмение.
5. Дитя Солнца.
6. Дух Солнца.
7. Ничто не вечно (nothing is forever).
8. Без света Солнца (without Sunlight).
9. Там, где погас огонь.
10. Когда пропадут звезды.

11. Потерянный свет.

12. Будь светом (Be Light).

13. Теперь здесь темно.

Так и по возможному имени главного героя:

1. Огнелис.
2. Ину.
3. Куми.
4. Юна.
5. Соин.
6. Огни.
7. Солфи.
8. Аши.
9. Аер.

После проведения опроса среди тех, кто знал о разработке, я решила назвать игру «Be Light», что переводится как «Будь Светом». Этот вариант хорошо подходит к сюжету игры, краток, хорошо читается и легко запоминается.

Сюжет преподносится игроку в самом начале игры, в следующем виде: «Когда-то Солнце не давало Злу поглотить нас. Однако Темная сторона души всегда пряталась за каждым из живых существ. Тьма ждала своего часа — часа, когда наше Солнце ослабнет и перестанет испускать свет.

И вот он настал. Тени вырвались на свободу и поглотили светлые души своих хозяев. На мир надвигается Тьма, и только одно существо сможет спасти Солнце и всех существ в нашем мире. Огненный дух, который может собрать частички оставшегося света и принести их к Солнцу, чтобы помочь ему загореться вновь».

Далее управляя главным героем, игрок должен собрать достаточное количество огнесвета и добраться до Солнца, чтобы зажечь его вновь.

2.2.2 Игровая логика

За каждые 500 частиц огнесвета игрок поднимается на следующий уровень и персонаж модифицируется. Огнесвет — это игровые бонусы, которые появляются от смерти монстров или в случайном месте игровых уровней. Их можно потратить на восстановление жизней или на некоторые сильные способности, так же их количество влияет на уровень игрока и локации в которой он находится. Индикатор количества огнесвета — большая круглая колба, всегда находящаяся на интерфейсе. Количество жизней представлено в виде лисьего хвоста. Чтобы восстановить жизни нужно нажать клавишу «E» или «C» английской раскладки, при этом добавится до 20 единиц жизней и отнимется такое же количество огнесвета. «Восстановление жизни» имеет время перезарядки.

Уровни генерируются по мере прохождения, пока не наберется колво огнесвета для перехода. После этого главный герой превратится в более совершенную версию себя и получит новые возможности. При этом появляется переход на следующий уровень и генерация объектов (монстров, окружения и др.) продолжается уже на следующем уровне.

За игроком, на определенном расстоянии двигается черный туман, закрывающий путь назад, и наносящий урон игроку при соприкосновении. Все попавшие в туман объекты будут возвращены в пул объектов. После смерти (когда все жизни будут исчерпаны) игрок появляется у чекпоинта, который зависит от текущего положения черного тумана с количеством жизней, равным количеству огнесвета до смерти (не более 100), однако если их было 0, то это становится концом игры.

Управление осуществляется клавишами WASD или стрелками на клавиатуре. Левая кнопка мыши (ЛКМ) или «Z» — удар в ближнем бою и наносит 15 единиц урона. Правая кнопка мыши (ПКМ) или «X» — стрельба наносит 20 единиц урона и тратит 1 единицу огнесвета (на 2 и 3 уровне). Пробел — это клавиша для взрыва (способность открывается

только на 3 уровне) который отодвигает все объекты рядом, так же эта способность имеет время перезарядки и тратит 2 единицы огнесвета. Все действия имеют альтернативные клавиши для удобства игроков.

На 2-ом уровне при удержании кнопки «стрелка вверх» или «W» можно летать, однако эта способность доступна, когда у игрока достаточно ресурсов полета, который копится стоя на земле и тратится от полетов. По истечении этого времени способность отключается, и игрок падает вниз.

Каждый вид монстра имеет свои характеристики: уровень жизней, количество наносимого им урона, скорость передвижения, количество падающего с них огнесвета и дополнительных особенностей.

Сохранений в игре не предполагается.

В начале каждого уровня имеются подсказки по управлению персонажем и других механик игры.

Средняя игровая сессия предполагает 30 минут.

2.2.3 Локации

Все локации в игре должны быть темными по сюжету (Солнце не светит). Предполагается, что игрок преодолевает путь от поверхности земли, до Солнца (рисунок 13).

Рисунок 13 — Схема уровней

Поэтому мною были выбраны три локации с разным окружением и разными механиками:

1. Лес. Ночной таинственный лес, наполненный деревьями и причудливо сложенными камнями, которые покрыты травой и мхом. Он выполнен в теплых тонах. Снизу располагается постоянная земля, камни выступают в роли платформ. Гравитация нормальная.

2. Небо. Ночное синее небо с облаками, которое освещают звезды. Существуют верхняя и нижняя граница, не являющиеся платформами, наносят периодический урон. Облака играют роль платформ. Игрок имеет возможность полетов. Гравитация уменьшена.

3. Космос. Космическое пространство с туманностями, астероидами разных форм кометами и др. объектами. Существуют верхняя и нижняя граница, не являющиеся платформами, наносят периодический урон. Объекты уровня при столкновении наносят урон. Гравитация отсутствует.

Для локаций были подобраны референсы и нарисован их примерный вид (рисунок 14).

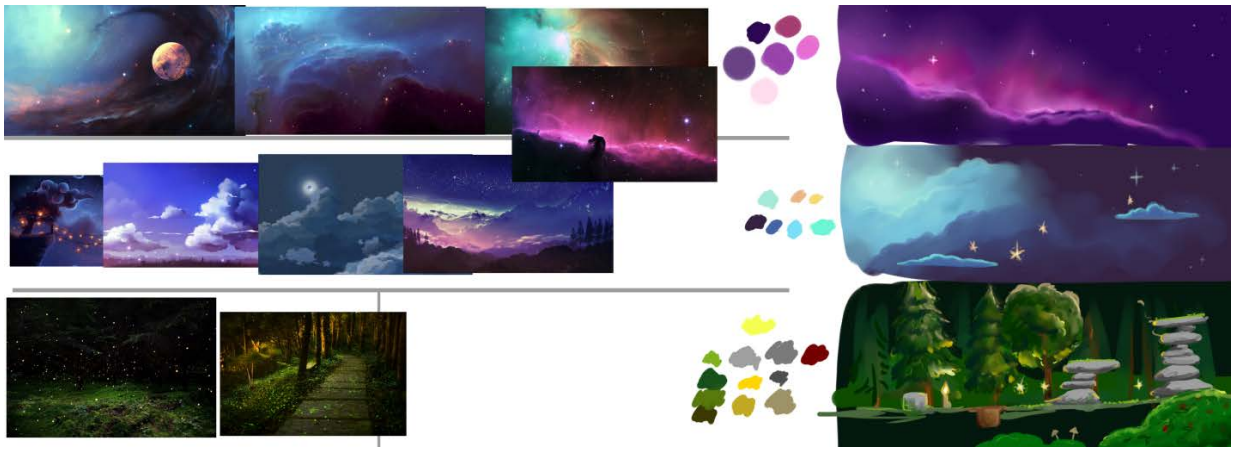

Рисунок 14 — Референсы, наброски и палитры локаций

Следуя представленным эскизам, создавалась конечная графика для уровней.

2.2.4 Главный герой

Дух леса, внешне похож на что-то среднее между лисицей и иволгой (рисунок 15).

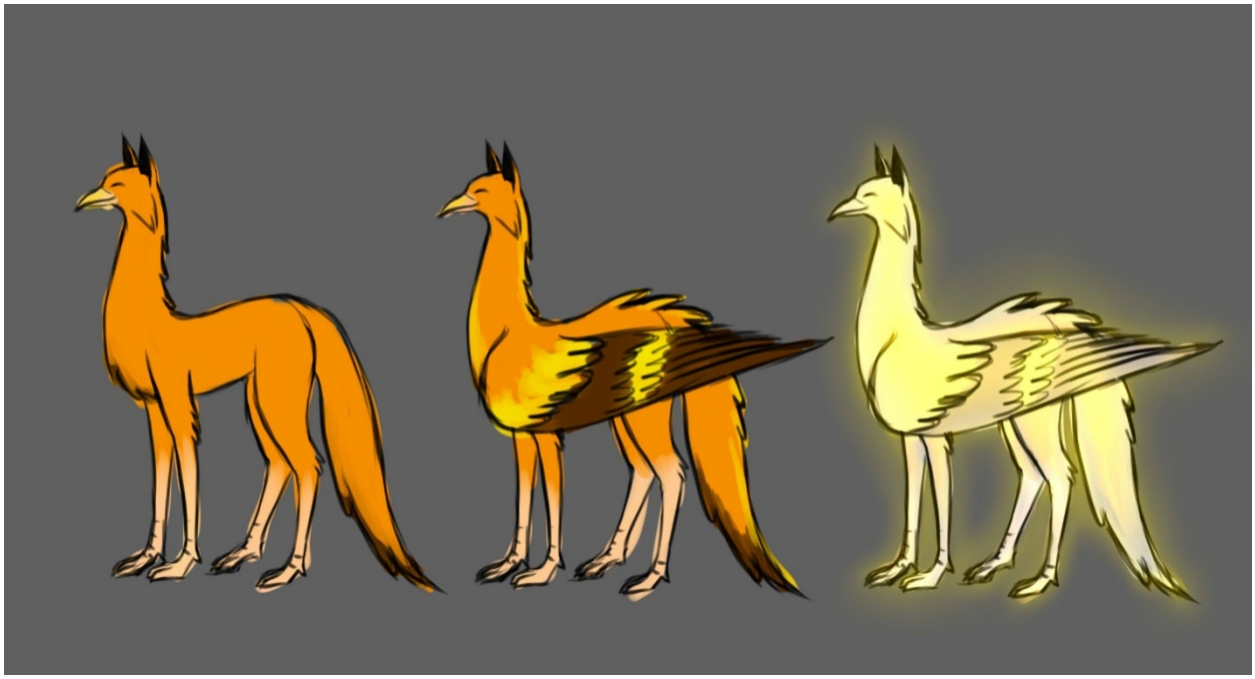

Рисунок 15 — Первичный набросок внешнего виды главного героя на разных уровнях

Имеет три уровня «взросления»:

1. Имеет лисье тело, но клюв и лапы птицы. Может атаковать врагов в ближнем бою. Имеет способность «стрелять» собранным огнем. Может бегать влево и вправо, совершать прыжок.

2. Лисица обретает крылья и немного видоизменяется, может летать, но только когда имеет ресурс полета. Он тратится при полете и восстанавливается, когда герой передвигается «по земле».

3. Лисица становится высшим духом, при этом меняется ее облик. В космосе гравитации нет, управление будет по типу «плавающего». Она приобретает новую возможность: создавать взрыв вокруг себя, который отодвигает объекты в определенном радиусе, при этом тратится некоторое количество огнесвета.

2.2.5 Враги и объекты, наносящие урон

Все монстры в игре это тени, черный дым, поэтому они выполнены в черном цвете. При потере жизней они становятся прозрачнее, пока свет огненного духа не заставит исчезнуть их вовсе.

На первом уровне монстры это тени лесных жителей, они обладают следующими характеристиками:

- волк — ходит свободно, однако если встретит препятствие или пропасть, развернется, бьет в ближнем бою;
- медведь — передвигается как волк, только медленнее. Бьет сильнее;
- рысь — передвигается как волк, пока рядом не появится игрок, в этом случае следует за ним, может прыгать;
- еж — быстро передвигается по горизонтали, если сталкивается с препятствием, разворачивается, умирает при столкновении с игроком.

На втором уровне монстры это тени птиц:

- сокол — летает по горизонтали, если видит игрока, следует за ним;
- ворон — передвигается как сокол, только медленнее;
- стриж — передвигается по диагонали, если встречает препятствие, разворачивается. При столкновении с персонажем погибает.

Так же имеются другие объекты, наносящие урон:

- черная тучка — является платформой, наносит периодический урон при соприкосновении, но восстанавливает ресурс полета;
- границы уровня — располагаются снизу и сверху на границах экрана. Наносят периодический урон при столкновении, не является платформой.

На третьем уровне нет объектов, наносящих активный урон, только пассивный урон при столкновении:

- несколько видов астероидов — свободно и крайне медленно передвигающиеся объекты, наносящие урон при столкновении;
- черная дыра — имеет собственную гравитацию, притягивает объекты к себе, убивает при соприкосновении;

- границы уровня — располагаются снизу и сверху на границах экрана. Наносят периодический урон при столкновении, не является платформой.

Кроме этого на всех уровнях есть черный туман, который следует за игроком на расстоянии, он убивает игрока при соприкосновении.

2.3 Разработка

После создания концепции игры, следует этап самой разработки. Далее последует описание различных аспектов разработки с подробными описаниями выполненных действий.

2.3.1 Написание кода

Поведение игровых объектов контролируется с помощью компонентов (Components), которые присоединяются к ним. Несмотря на то, что встроенные компоненты Unity могут быть очень разносторонними, вскоре обнаруживается, что вам их возможностей недостаточно, чтобы реализовать ваши собственные особенности геймплея. Чтобы исправить это, Unity позволяет создавать свои компоненты, используя скрипты. Они позволяют активировать игровые события, изменять параметры компонентов, и отвечать на ввод пользователя каким угодно способом. Unity изначально поддерживает два языка программирования: C# и UnityScript. По программе обучения, мною был изучен язык программирования c#, именно поэтому был выбран именно этот язык программирования для написания кода.

Игровая логика

Скрипт CameraController отвечает за переходы на следующие уровни и передвижение камеры.

Весь алгоритм расположен в методе Update. Он отвечает за воспроизведение меню паузы, меню проигрыша, передвижение камеры вслед за игро-

ком и перехода игрока на следующий уровень. О последнем действии поговорим подробней.

Скрипт имеет ссылку на игрока и постоянно проверяет числовое значение количества огнесвета в колбе, если оно превысило порог для прохождения текущего уровня, происходит следующее:

1. Выключается менеджер текущего уровня, который содержит в себе алгоритмы генерации платформ, монстров и бонусов на текущем уровне.
2. Изменяется глобальная переменная уровня.
3. На текущей позиции игрока активируется заготовка начала следующего уровня с соответствующими платформами и экранами обучения.
4. Запускаются визуальные эффекты превращения.
5. Когда эффект достигнет игрока, его префаб заменяется на префаб следующего уровня, в это же время смену внешнего вида игрока закрывают визуальные эффекты.
6. Камера включает вертикальный режим следования за игроком, пока игрок не окажется на высоте следующего уровня.
7. После этого удаляется пул объектов предыдущего уровня.
8. Затем включается менеджер генераций следующего уровня, который создает пул соответствующих объектов.
9. Камера включает горизонтальный режим следования за игроком, и игрок продолжает следование по горизонтали далее, однако если это был конечный уровень, запускаются объекты, вызывающиеся при победе.

Персонаж

Скрипт персонажа называется Character. Он отражает в себе механику главного героя. В нем содержаться характеристики игрока, такие как жизни, количество урона от атак, скорость передвижения, количество собранного огнесвета и др.

Другие переменные отражают: направление игрока, находится ли он на земле, находимся ли мы в состоянии прыжка и др.

Имеются ссылки на компоненты объекта: аниматор, физическое тело.

Так же скрипт хранит в себе ссылки на другие объекты уровня: префабы для разных видов атаки, существующий на сцене экземпляр черного тумана.

Разберем методы скрипта отдельно.

Start — для задания начальных параметров при инициализации объекта, характеристик, получения ссылок на другие объекты и компоненты.

FixedUpdate — метод, который вызывается через фиксированые промежутки времени, в данном скрипте используется для расчета важных параметров: проверка стоит ли персонаж на земле (вызывается метод CheckGround()), проверка жизней игрока (при 0 запасе жизней вызывается метод Die()), задается ускорение персонажа, которое зависит от направления и скорости.

Update — метод, вызываемый каждый кадр, напрямую зависит от производительности оборудования. Он отвечает за регистрирование нажатий клавиш игроком и запуск соответствующих методов. Так же он меняет цвет игрока, в зависимости от количества жизней.

Метод CheckGround проверяет, находится ли игрок на земле.

Метод Attack отвечает за вызов нужного префаба атаки, вычитание из общего количество огнесвета стоимости вызванного вида атаки, вызывается с помощью событий аниматора.

Метод OnTriggerEnter вызывается от столкновений текущего объекта с триггерами на сцене. В данном случае он считает количество собранного игроком огнесвета.

Метод ConvertLives отвечает за способность восстановления жизней за счет собранного огнесвета.

Метод Die отвечает за перерождение игрока, если у него имеется огнесвет в колбе или провоцирует конец игры, если огнесвета нет.

Метод PlayRunSound отвечает за воспроизведение звуков бега, вызывается через события аниматора объекта.

Так же в скрипте могут присутствовать другие методы.

Враги

Для проекта созданы скрипты для монстров, связанные следующей иерархией (рисунок 16).

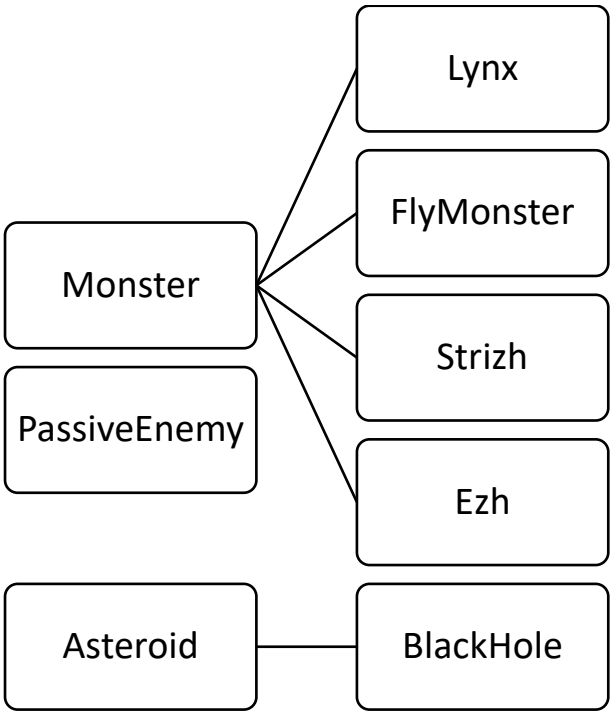

Рисунок 16 — Иерархия скриптов монстров

Многие из этих скриптов универсальны и подходят множеству врагов, различны в них лишь начальные переменные, которые являются характеристиками монстров.

Скрипт Monster содержит в себе основные характеристики и простой метод передвижения монстров, зависящий лишь от наличия препятствий. Им пользуются следующие объекты: wolf (волк), bear (медведь).

Скрипт Lynx наследует характеристики скрипта Monster, однако изменяет метод передвижения одноименного монстра (рыси), позволяет следовать за игроком и прыгать.

Скрипт FlyMonster наследует характеристики скрипта Monster, однако изменяет метод передвижения монстров, по типу птиц, позволяет следовать за игроком. Скрипт принадлежит следующим объектам: voron (ворон), sokol (сокол).

Скрипты Strizh, Ezh, Comet предназначены для одноименных объектов (стриж, еж, комета), они наследуют характеристики крипта Monster, изменяют метод передвижения и метод «умирания».

Скрипт PassiveEnemy предназначен для нанесения периодического урона при соприкосновении с игроком. Он используется во многих объектах, например, mushki (мошкара), black (черная тучка), DieArea (зоны смерти, границы уровней).

Скрипты Asteroid (астеорид) и BlackHole (черная дыра) используются для одноименных объектов.

Генерация платформ и других объектов

Генерация платформ первого уровня происходит по алгоритму скрипта ManagerPlatform (Приложение Б).

Изначально получается ссылка на объект Generator, прикрепленный к игроку, вычисляется его положение по оси x.

Дальше каждый кадр генерируется массив объектов, находящихся в определенном (маленьком) радиусе в позиции генератора по x и на высоте земли по оси y. Если в массиве присутствуют объекты — значит начальный, настроенный набор платформ еще не завершился. Однако если он пуст, то с этой позиции начнется генерация. Она уже будет зависеть только от положения генератора по оси x. Если текущая позиция генератора больше чем последняя позиция сохраненная позиция генерации объекта земля + длинна земли, то создается новая земля, в этот же момент проверяется прошло ли расстояние, выбранное для следующей генерации платформы. Если условие верно, то в данной позиции генерируется объект из заранее выбранного набора платформ (отличающихся лишь графически) и вызывается метод этого набора (первый раз выбирается 1 платформа из первого набора на расстоянии 0). В этом методе прописаны возможные варианты следующего набора платформ на возможных, заранее определенных, расстояниях от текущей. Все варианты просчитывались таким образом, чтобы не возникало тупиковых вариантов, таких как пропасти, из которых не выбраться, отдельно стоя-

щие высокие платформы, на которые не забраться и др. После этого алгоритм снова возвращается на проверку расположения генератора и сравнивает его положение с последним положением генерации и новым случайным шагом для следующей.

Для второго уровня используется упрощенный вариант алгоритма генерации платформ (Приложение В). При инициализации получается ссылка на генератор, затем точно так же, как в предыдущем алгоритме, вычисляется начало генерации. После его определения таким же методом генерируются границы уровня (сверху и снизу) и два раза вызывается метод генерации платформ, который отличается лишь текущей высотой генерации, они являются константами для уровня. В самом методе проверяется, находится ли генератор на расстоянии, равном предыдущему сохраненному положению генерации и случайному шагу, выбранному там же. Если условие верно генерируется случайная платформа из списка на случайной высоте, равной ±2 от константной высоты. Так же выбирается следующая позиция генерации с учетом длины текущей платформы, после этого программа начинает алгоритм сначала. Скрипт генерации платформ третьего уровня почти полностью идентичен скрипту для второго уровня.

Так же в проекте присутствует универсальный алгоритм генерации монстров (Приложение Д) и огнесвета (Приложение Г).

Другое

Так же в проекте присутствуют следующие скрипты:

- FireSphere (поведение частиц огнесвета, бонусов);
- Deleter (Поведение черноto тумана, младующего за игроком);
- Fire (скрипт для префабов атаки персонажа);
- Fon (движение статичного фона со звездами);
- GameLoading (экран загрузки игры);
- Learning (для отображения экранов обучения);
- ParalaxMenu (для моздания эффекта глубины в главном меню);
- Paralax (для создания эффекта паралакса для игровых фонов);

- Smoke (для прекращения жизненного цикла системы частиц, которая появляется после смерти монстров);
- Static (содержит некоторые константы для генераторов);
- UI (переносит показатели игрока на игровой интерфейс);
- Набор из четырех скриптов для создания пула объектов.

Так же в проекте встречаются и другие скрипты.

2.3.2 Визуальная часть

Визуальная составляющая игры очень важна, ведь именно от нее во многом зависит восприятие игры игроком.

Монстры

Сначала были сделаны эскизы персонажей, затем полная отрисовка существ и разбиение их на составные части, для дальнейшей анимации.

Рассмотрим создание монстра на примере моба первого уровня — медведя (рисунок 17).

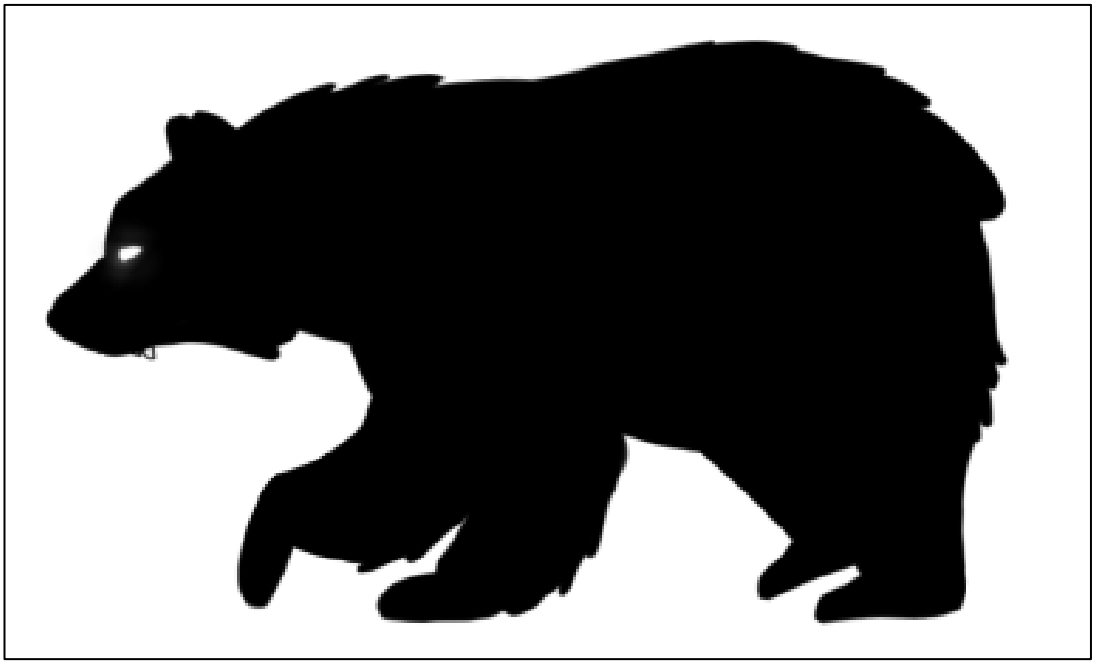

Рисунок 17 — Медведь

Далее он разбивался на составные части (рисунок 18).

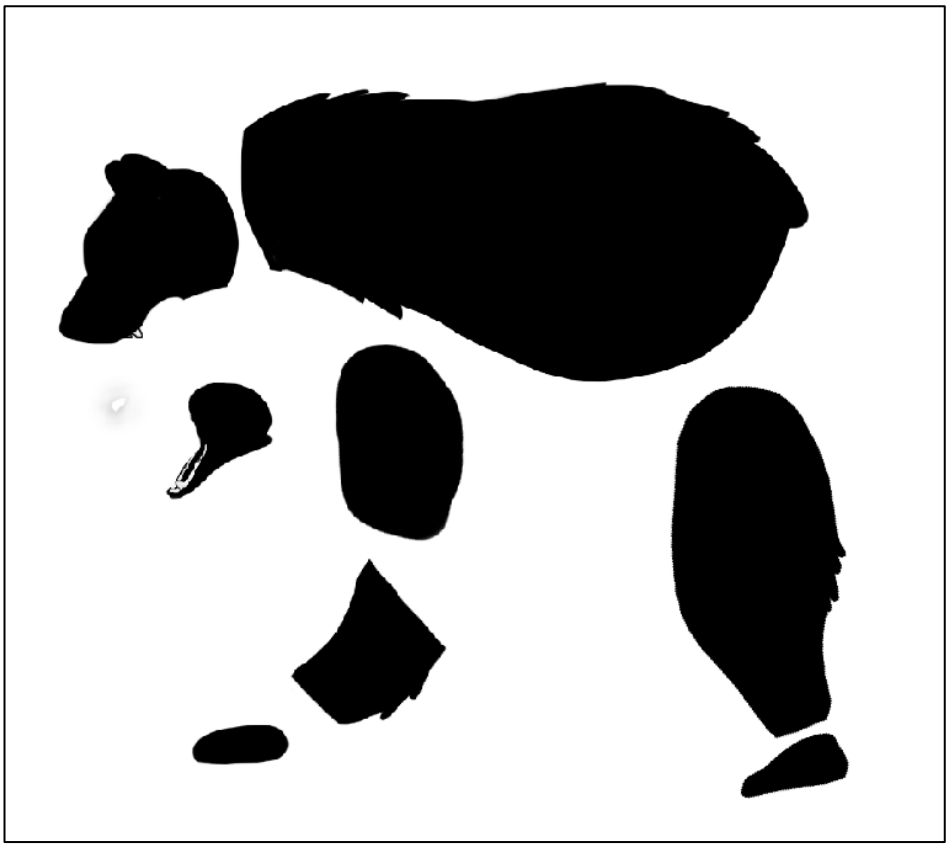

Рисунок 18 — Составные части монстра «Медведь»

Создавался скелет в программе для анимации Dragon Bones (рисунок 19).

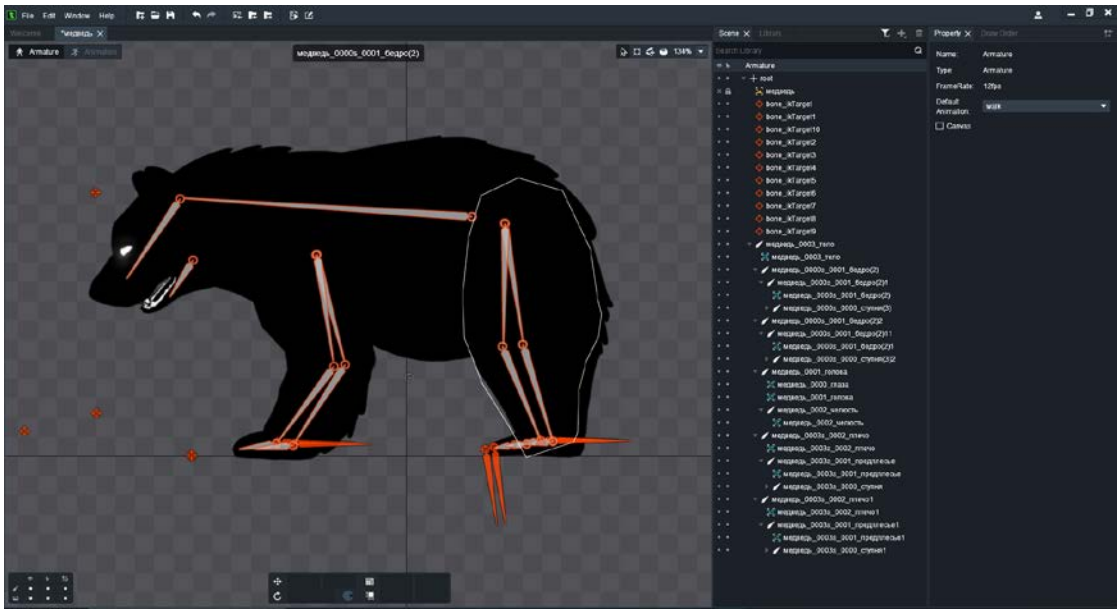

Рисунок 19 — Скелет для медведя

Далее создавались нужные виды анимаций для каждого из монстров (рисунок 20).

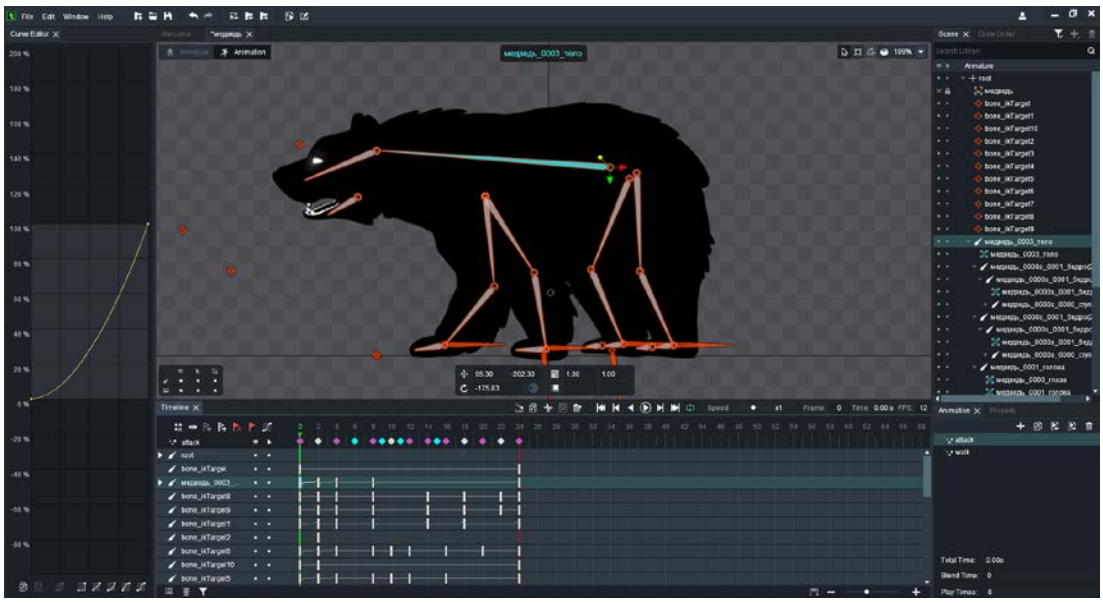

Рисунок 20 — Анимирование монстра

После анимация сохранялась в отдельные изображения, как покадровая анимация и с помощью плагина Sprite Packer преобразовывалась в лист спрайтов (рисунок 21).

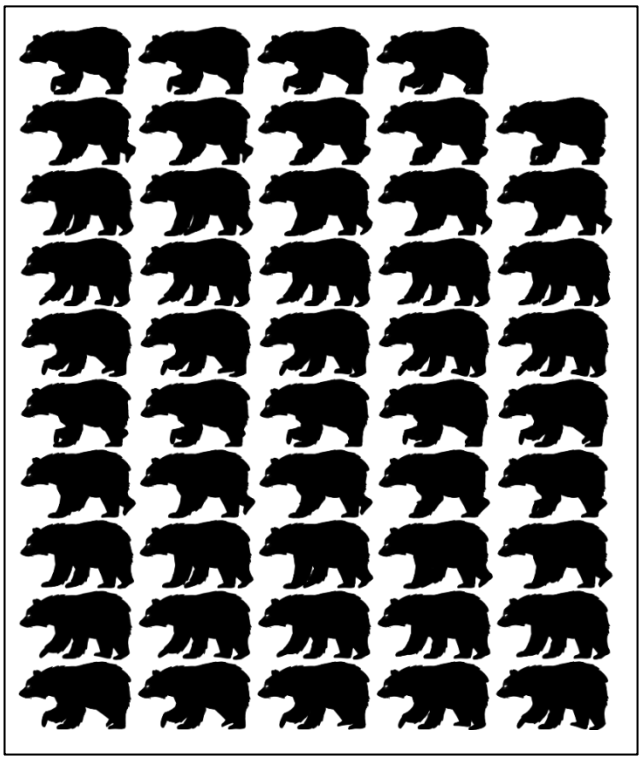

Рисунок 21 — Лист спрайтов с анимацией ходьба монстра медведь

Далее создавалась анимация в самом движке Unity и настраивались события, например, момент вызова метода удара (рисунок 22).

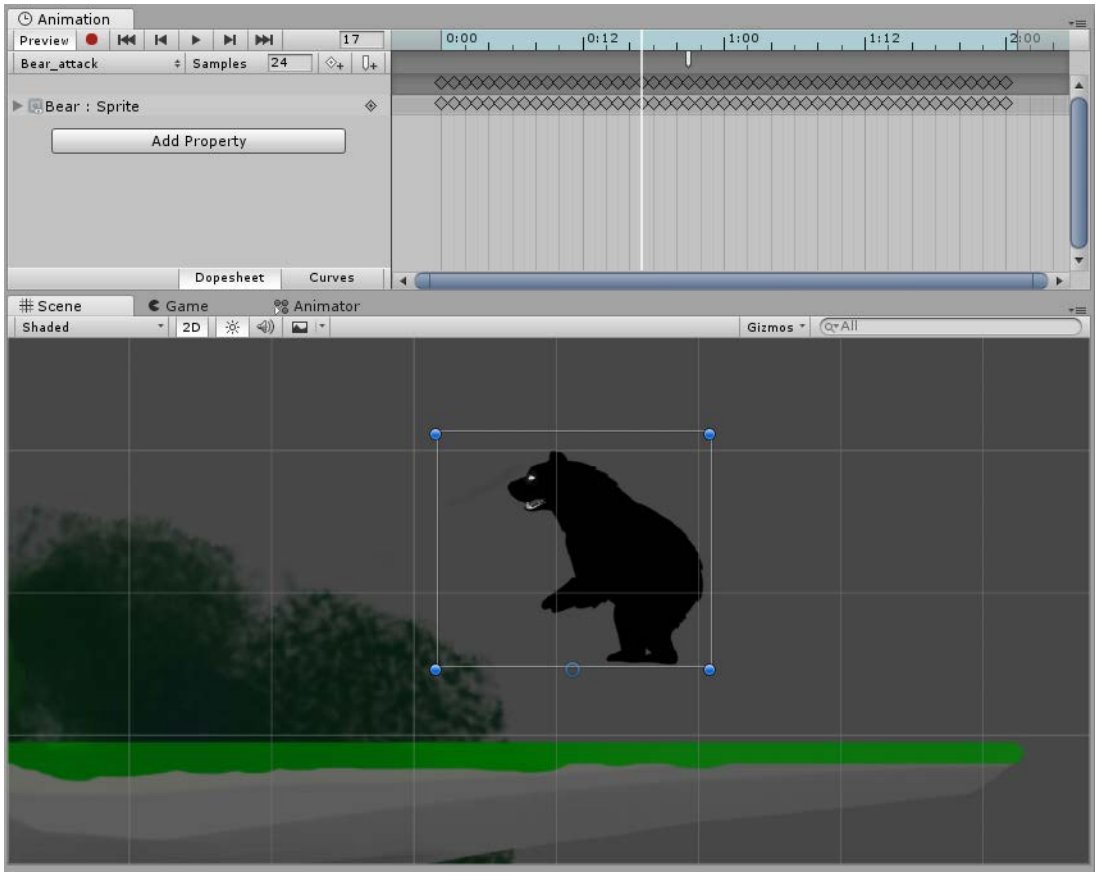

Рисунок 22 — Анимирование монстра в движке

После настройки всех анимаций монстра необходимо настроить машину состояний анимации (рисунок 23).

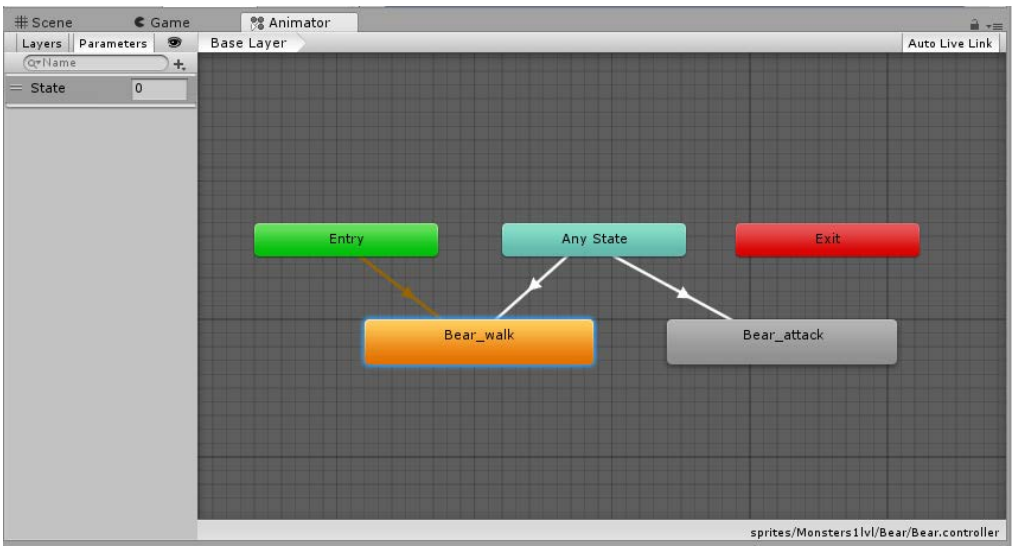

Рисунок 23 — Настройка переходов анимации

На этом визуальная часть работы с медведем закончена.

Главный персонаж

Главный герой — неотъемлемая часть игры, и она должна быть хорошо визуализирована, чтобы понравится игрокам.

Создание персонажа начиналось с эскиза. Затем в программе Corel Draw создавались детали персонажа. Они подготавливались таким образом, чтобы при настроенном скелете в программе для анимации, кости и другие части не выходили за допустимые пределы (рисунок 24).

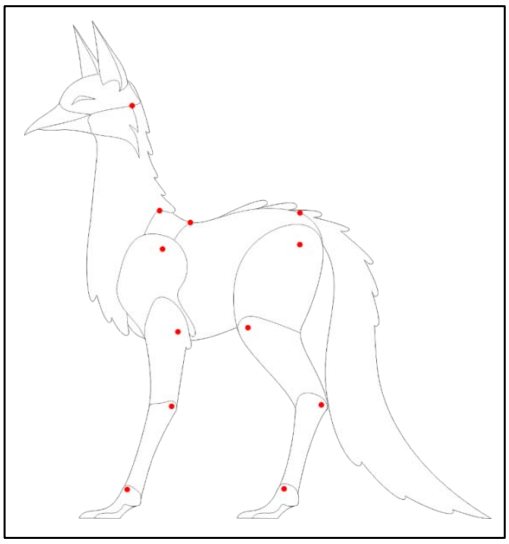

Рисунок 24 — Составные части главного персонажа

Далее подбирались цвета для каждого конкретного уровня лисы (рисунок 25).

Рисунок 25 — Внешний вид героя на втором уровне

Для второго уровня рисовалась покадровая анимация крыльев (рисунок 26).

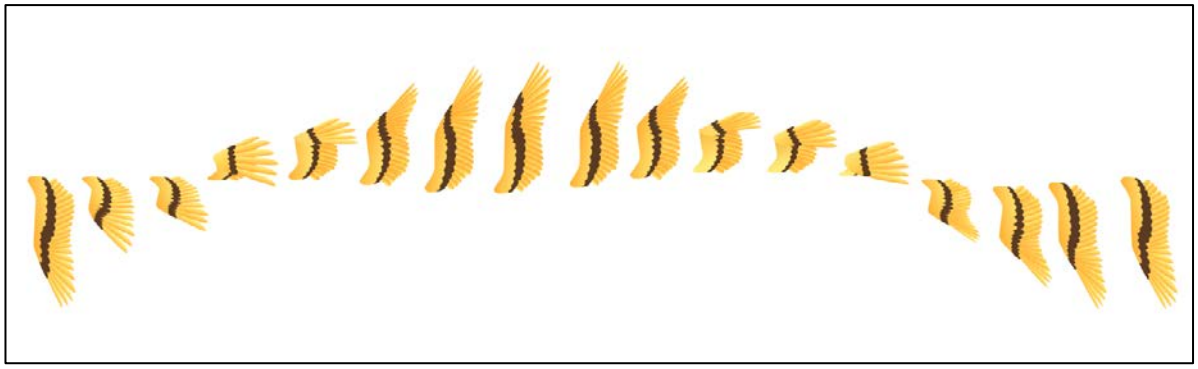

Рисунок 26 — Покадровая анимация движения крыльев

После этого детали существа (такие как голова, верхняя и нижняя части клюва, каждая из частей конечностей и др.) разделялись и сохранялись в отдельных файлах. Дальше все части изображения переносились в программу Dragon Bones, расставлялись по местам, затем создавался скелет персонажа (рисунок 27).

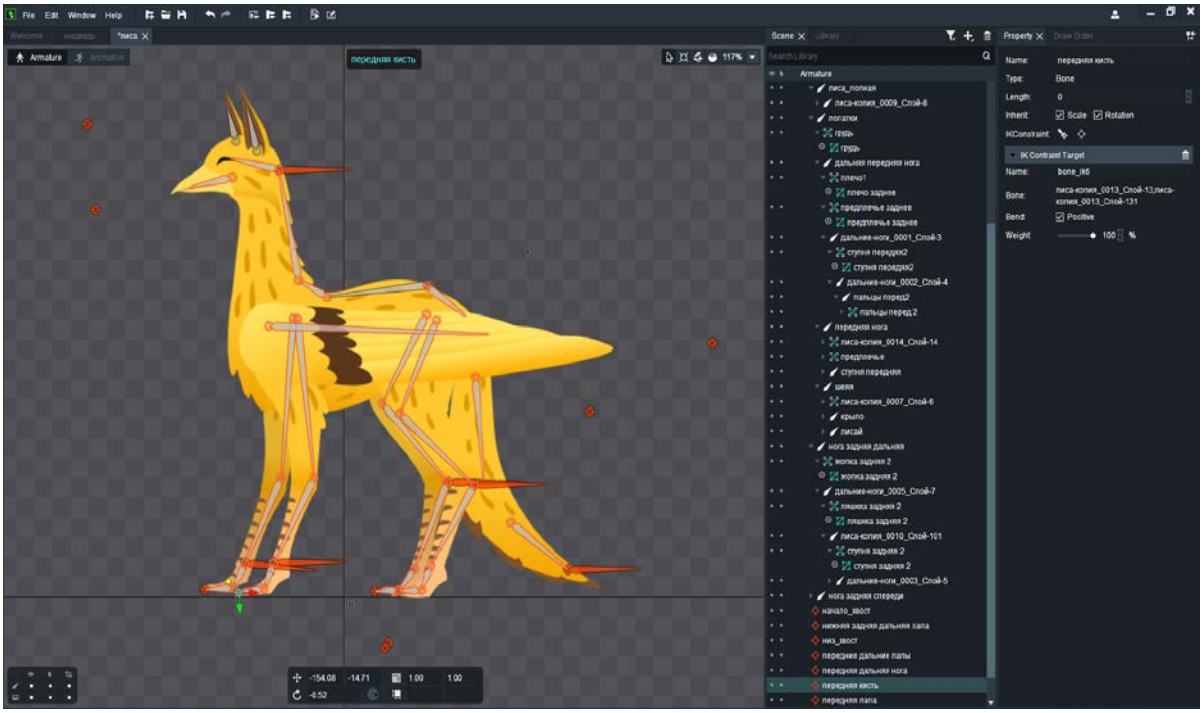

Рисунок 27 — Настройка скелета главного персонажа

Дальше создается требуемая анимация (рисунок 28).

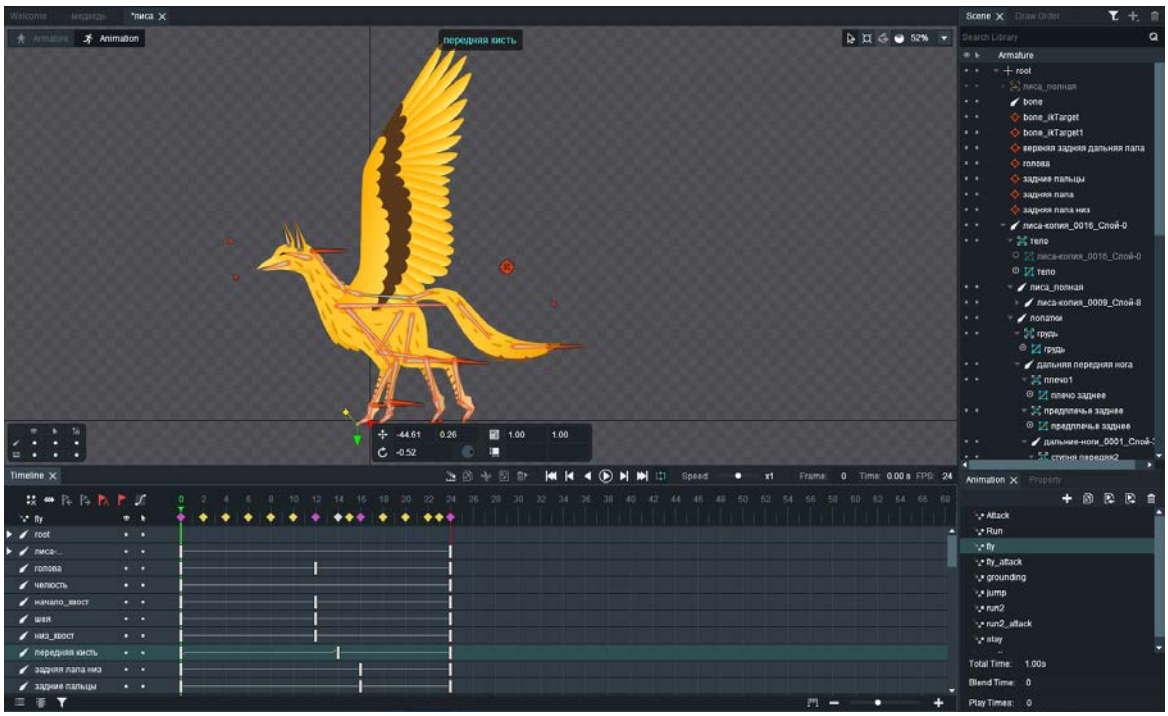

Рисунок 28 — Анимирование персонажа

Далее все анимации сохраняются в отдельные изображения, как покадровую анимацию и с помощью плагина Sprite Packer преобразовываются в листы спрайтов. Потом настраиваются анимации и машина состояний анимаций в движке Unity (рисунок 29).

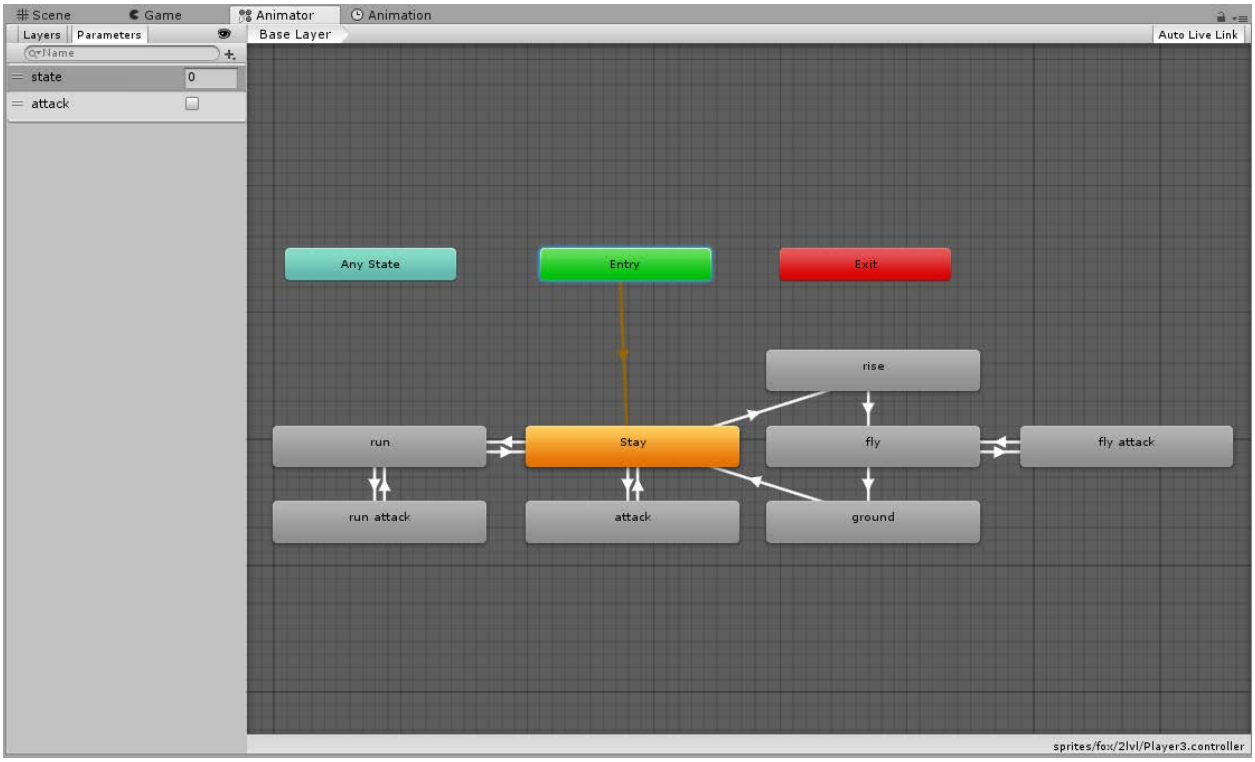

Рисунок 29 — Машина состояний анимации главного персонажа второго уровня

На этом визуальная часть работы с объектом персонажа второго уровня закончена.

Окружение

Окружение уровней состоит из различных фонов, двигающихся с разными скоростями, для создания эффекта глубины. Некоторые элементы фона уровня «небо» представлены на рисунке 30.

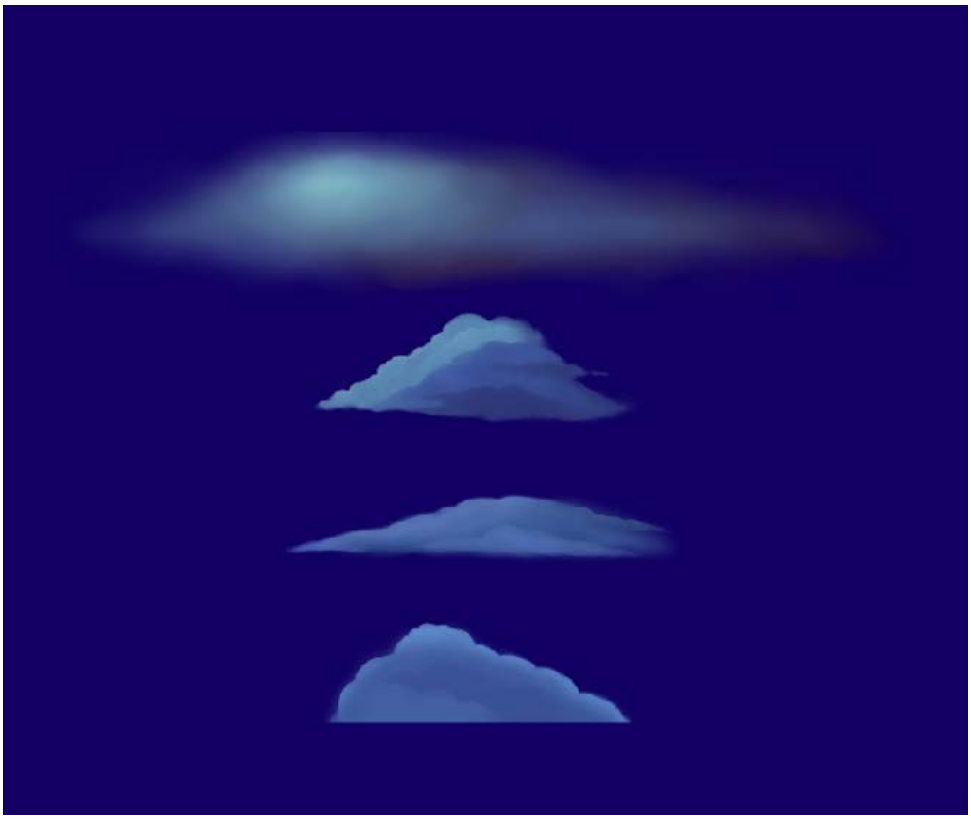

Рисунок 30 — Фоны уровня «небо»

Так же к объектам окружения относятся платформы. Для каждого вида платформы, которые отличаются между собой длинной, высотой, формой и другими характеристиками, было создано несколько спрайтов. Пример листа спрайтов с платформами первого уровня: «Лес», представлен на рисунке 31.

Интерфейс

Интерфейс — это индикаторы, всегда располагающиеся на экране во время игры (рисунок 32).

- цифры на большой колбе указывают на количество собранного огнесвета, а уровень ее заполненности влияет на переход на следующий уровень (когда она заполнится полностью, лиса превратится);

- горизонтальный оранжевый индикатор указывает на количество здоровья;
- голубое перо на оставшийся ресурс полета (меняет прозрачность);
- желтый огонек на возможность использования «взрыва»;
- свечение вокруг колбы показывает, что можно снова менять огне-свет на жизнь.

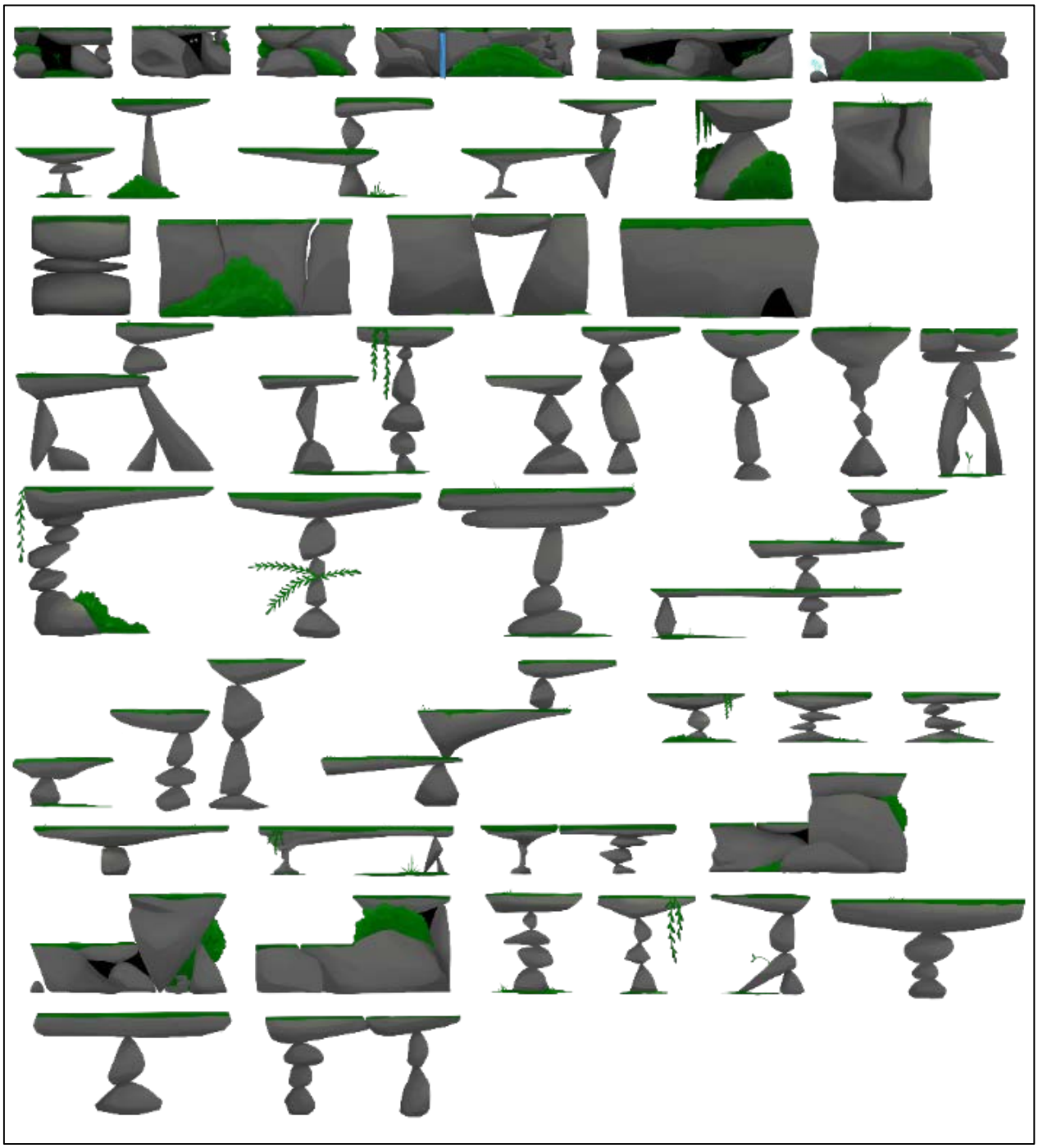

Рисунок 31 — Лист спрайтов платформ первого уровня

Рисунок 32 — Интерфейс

Меню паузы

Меню паузы вызывается по нажатии клавиши «esc» во время игрового процесса (рисунок 33).

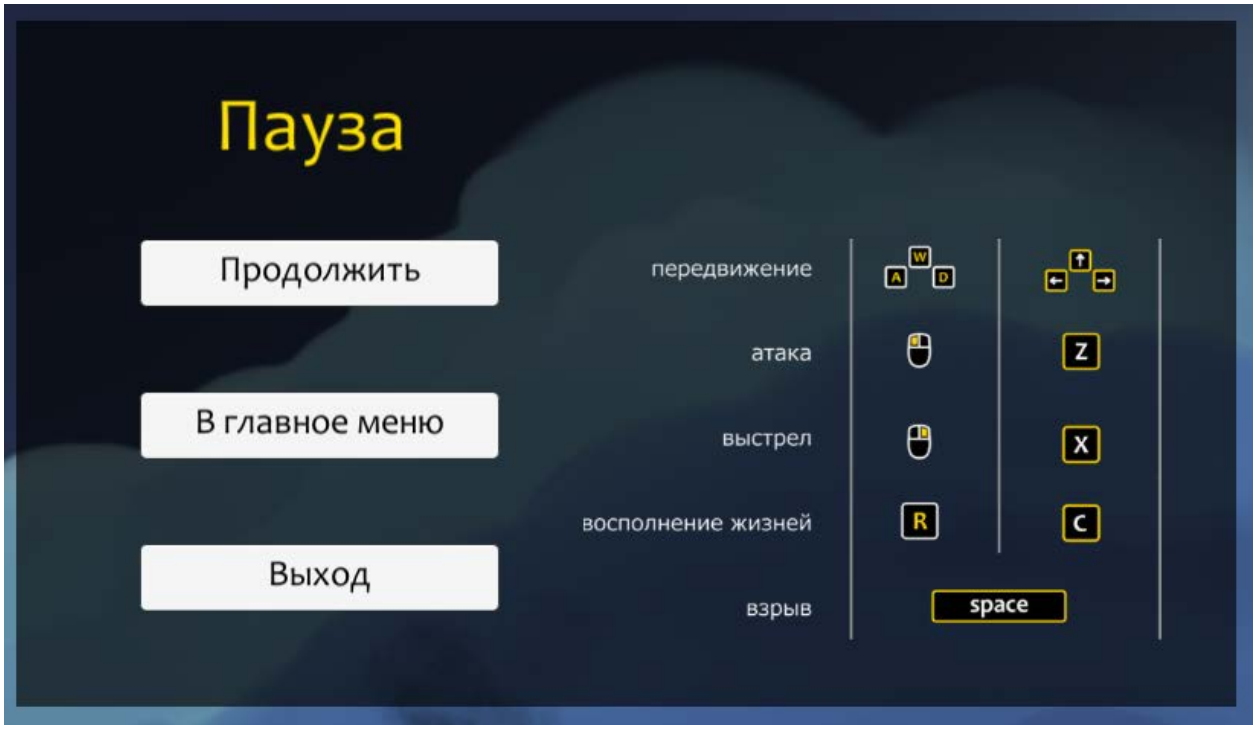

Рисунок 33 — Меню паузы (внутриигровое меню)

Полупрозрачный черный прямоугольник, на котором указано состояние игры в текущий момент: Пауза, краткая таблица с кнопками управления, кнопки: «Продолжить» (для продолжения игры), «В главное меню» (прину-

дительный конец игры и переход в главное меню), «Выход» (выход из игры полностью).

Главное меню

Главное меню вызывается при запуске игры, имеет всего две активные кнопки: «Играть» (для начала игры), «Выход» (для выхода из игры).

Внешне представляет собой сплошную темно-синюю область, заполненную системой частиц, имитирующей звезды и черный туман по периметру экрана (рисунок 34). Все системы частиц имеют анимацию. Звезды разделены на несколько «планов» и двигаются, в зависимости от положения курсора на экране. Так же, иногда по небу пролетает огненный дух, который является главным персонажем игры

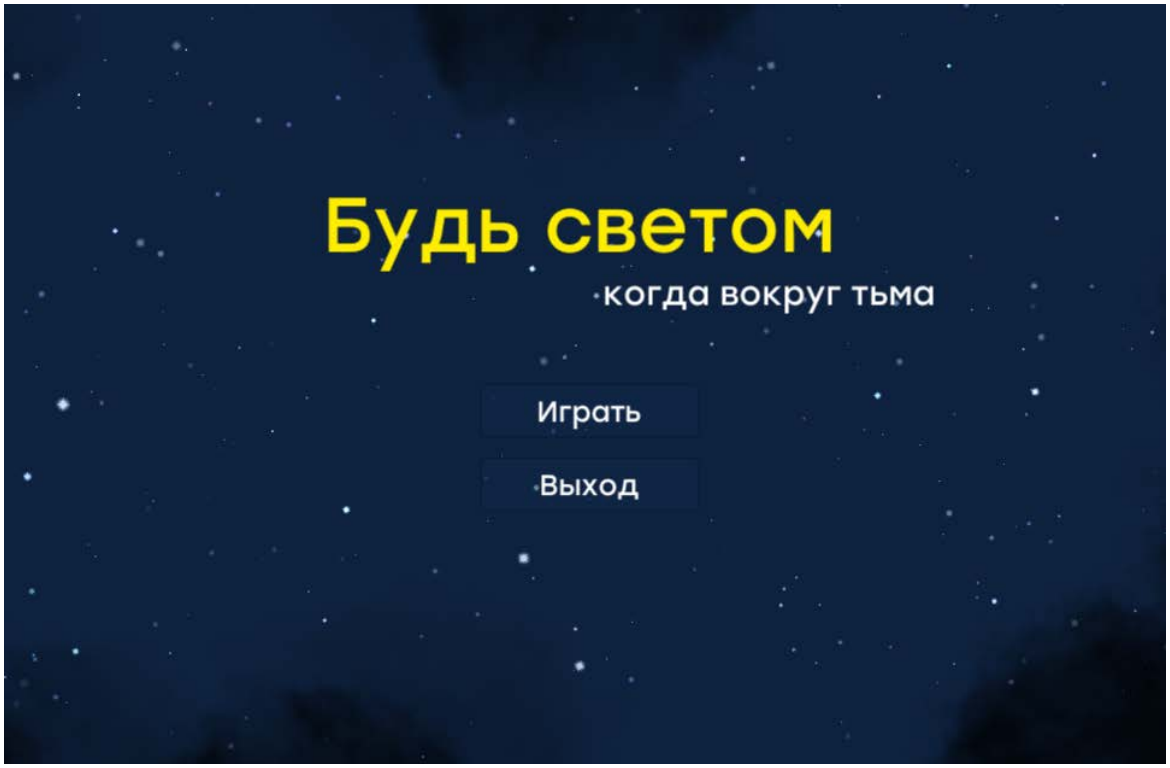

Рисунок 34 — Главное меню

2.3.3 Музыкальное сопровождение

Для звукового сопровождения, были специально созданы звуки и композиции, вот некоторые из них:

- главная тема (фоновая мелодия);

- хождение лисы по траве;
- приземление на траву;
- полет (хлопанье крыльев);

Так же в проекте присутствуют другие аудиофайлы.

Главная музыкальная тема для игры была написана и создана композитором и звукорежиссером данного проекта Авдеевым Владиславом Михайловичем, известным под псевдонимом Dale Maxflyer.

Некоторые звуки были созданы путем добработки исходных файлов, взятых с сайтов, распространяющих аудиофайлы для озвучки бесплатно, например: wav-library.net и freesound.

2.4 Калькуляция проекта

В ходе разработки проекта было создано:

- 5 уникальных алгоритмов генерации объектов;
- 36 скриптов с общим количеством строк равному 2849;
- 10 спрайтов с фонами;
- 10 уникальных существ для игры, для которых создано 27 скелетных анимаций, которые содержат 966 кадров;
- 8 машин состояний анимации;
- 276 кости для скелетной анимации;
- 78 спрайтов платформ;
- 3 игровых меню;
- 115 префабов объектов;
- 4 сцены;
- 126 коммитов на GitHub.

ЗАКЛЮЧЕНИЕ

На сегодняшний день компьютерные игры являются неотъемлемой частью индустрии развлечений. К сожалению, в России данная отрасль развита не столь сильно.

В работе рассмотрена разработка компьютерный игры.

Целью данной работы является проектирование концепта игры и ее разработка.

Для достижения цели были проанализированы специализированные литературные и интернет-источники по вопросам разработки компьютерных игр, выбран жанр для разрабатываемого проекта, изучены похожие, уже существующие разработки, создана концепция игры, изучены методы работы с соответствующим ПО.

Разработка собственного проекта с нуля позволила получить огромный опыт во многих областях, связанных с проектированием и реализацией игр, таких как: создание концепции игры, проработка игровых механик, создание концепт артов для персонажей и локаций игры, создание анимаций, написание алгоритмов и кодинг, сборка и настройка проекта в игровом движке.

Используя навыки программирования на C#, разработки интерфейсов, цифрового рисования, теоретических знаний по проектированию концепции игры, приобретенные во время обучения в институте, была создана компьютерная игра в жанре платформер с процедурной генерацией уровней. Для разработки были использованы игровой движок Unity, среда программирования Visual Studio, графические редакторов Adobe Photoshop, Corel Draw, Paint Tool SAI, программа для анимации Dragon Bones.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. 7 примеров использования процедурной генерации в играх, о которых полезно знать всем разработчикам [Электронный ресурс]. — Режим доступа: https://habrahabr.ru/company/ua-hosting/blog/275195/ (дата обращения: 29.04.2018).

2. Жанры компьютерных игр [Электронный ресурс]. — Режим доступа: http://gamesisart.ru/TableJanr.html#Horror/ (дата обращения: 15.05.2018).

3. Что такое компьютерная игра [Электронный ресурс]. — Режим доступа: Https://lektsia.com/4xa07b.html (дата обращения: 29.04.2018).

4. Меженин М. Г. Обзор систем процедурной генерации игр [Текст] / М. Г. Меженин // Вестник ЮУрГУ. Серия «Вычислительная математика и информатика». — 2015. — №1. — С. 5–20.

5. Платформер [Электронный ресурс]. — Режим доступа: https://ru.wikipedia.org/wiki/Платформер/ (дата обращения: 01.06.2018).

6. Технологии компьютерных игр [Электронный ресурс]. — Режим доступа: http://mirznanii.com/a/115426/tekhnologii-kompyuternykh-igr/ (дата обращения: 01.06.2018).

7. Игровой движок [Электронный ресурс]. — Режим доступа: https://dikc.academic.ru/dic.nsf/ruwiki/209444/ (дата обращения: 01.06.2018).

8. Руководство Unity: порядок выполнения функций событий [Электронный ресурс]. — Режим доступа: https://docs.unity3d.com/ru/530/Manual/ExecutionOrder.html (дата обращения: 01.06.2018).

9. Создание и использование скриптов [Электронный ресурс]. — Режим доступа: https://docs.unity3d.com/ru/530/Manual/CreatingAndUsingScripts.html (дата обращения: 01.06.2018).

10. Для кого эта игрушка или как определить целевую аудиторию [Электронный ресурс]. — Режим доступа: https://habr.com/post/253895/ (дата обращения: 03.06.2018).

11. Психотипы Бартла и балансировка аудитории [Электронный ресурс]. — Режим доступа: https://habr.com/company/mailru/blog/263839/ (дата обращения: 03.06.2018).

12. Подвижный бэкграунд в 2D платформере [Электронный ресурс]. — Режим доступа: https://null-code.ru/solution/73-podvizhnyy-bekgraund-v-2d-platformere.html (дата обращения: 16.05.2018).

13. 2d игра на unity. Подробное руководство [Электронный ресурс]. — Режим доступа: http://websketches.ru/blog/2d-igra-na-unity-podrobnoye-rukovodstvo-p1/ (дата обращения: 09.01.2018).

14. Руководство Unity [Электронный ресурс]. — Режим доступа: https://docs.unity3d.com/ru/530/Manual/ (дата обращения: 08.08.2017).

15. Unity3D.ru [Электронный ресурс]. — Режим доступа: http://unity3d.ru/distribution/viewforum.php?f=11/ (дата обращения: 26.02.2018).

16. MSDN [Электронный ресурс]. — Режим доступа: https://msdn.microsoft.com/library/ (дата обращения: 19.03.2018).

17. Ролик на YouTube: Создание платформера Unity [Электронный ресурс]. — Режим доступа: https://youtu.be/oYaGPeZW8JM (дата обращения: 28.08.2018).

18. Ролик на YouTube: Unity. Урок 2: Анимация [Электронный ресурс]. — Режим доступа: https://youtu.be/XutAcL3T96I/ (дата обращения: 11.03.2018).

19. Ролик на YouTube: unity 5 для начинающих, caroutines. [Электронный ресурс]. — Режим доступа: https://youtu.be/YsOgVU5S6dI/ (дата обращения: 20.05.2018).

20. Ролик на YouTube: Создание простого 2D платформера в Unity3D. [Электронный ресурс]. — Режим доступа: https://youtu.be/CAPVBTKk3Ww/ (дата обращения: 23.08.2017).

21. Ролик на YouTube: Particle System Trails, Unity Particle Effects [Электронный ресурс]. — Режим доступа: https://youtu.be/agr-QEsYwD0/ (дата обращения: 16.05.2018).

22. Плейлист на YouTube: Unity 5 C# уроки для начинающих [Электронный ресурс]. — Режим доступа: https://www.youtube.com/playlist?list =PL0lO_mIqDDFVNOKquWCHh4n-Ird5HRB_1/ (дата обращения: 02.05.2017).

23. Видеокурс unity user interface [Электронный ресурс]. — Режим доступа: https://itvdn.com/ru/video/unity-user-interface?utm_source=yb_promo_ unusin/ (дата обращения: 25.05.2018).

24. Unity Звук Звуки Музыка [Электронный ресурс]. — Режим доступа: https://youtu.be/99VW4mglwJA (дата обращения: 19.05.2018).

25. Канал на YouTube: уроки по Dragon bones [Электронный ресурс]. — Режим доступа: https://www.youtube.com/channel/UCEAN0pwhdEKewMk 7kTyVJGQ/featured/ (дата обращения: 04.02.2018).

26. Язык C# и .NET Framework [Электронный ресурс]. — Режим доступа: https://professorweb.ru/my/csharp/charp_theory/level1/infonet.php (дата обращения: 04.08.2017).

27. Учебник по новому gui в unity [Электронный ресурс]. — Режим доступа: http://websketches.ru/blog/uhebnik-po-novomu-gui-v-unity-1/ (дата обращения: 04.05.2018).

28. Кватернион [Электронный ресурс]. — Режим доступа: http://unitycode.blogspot.com/2012/04/blog-post_3413.html (дата обращения: 02.06.2018).

29. Unity: оптимизация производительности графики [Электронный ресурс]. — Режим доступа: https://docs.unity3d.com/ru/530/Manual/Optimizing GraphicsPerformance.html (дата обращения: 04.02.2018).

30. Про создание платформера на Unity [Электронный ресурс]. — Режим доступа: https://habr.com/company/microsoft/blog/236125/ (дата обращения: 14.08.2017).

ПРИЛОЖЕНИЕ А

Министерство образования и науки Российской Федерации

Федеральное государственное автономное образовательное учреждение

высшего образования

«Российский государственный профессионально-педагогический университет»

Институт инженерно-педагогического образования

Кафедра информационных систем и технологий

Направление подготовки 09.03.02 Информационные системы и технологии

Профиль подготовки «Информационные технологии в медиаиндустрии»

УТВЕРЖДАЮ
Заведующий кафедрой

_____ Н.С. Толстова
подпись и.о. фамилия

« ___ » _____ 2018 г.

ЗАДАНИЕ
на выполнение выпускной квалификационной работы бакалавра

студента (ки) *4* курса группы *ИТм-402*

Ганиевой Дианы Ринатовны
фамилия, имя, отчество полностью

1. Тема ***2d-платформер с процедурной генерацией уровней***

утверждена распоряжением по институту от «_____» _____ 20 г. № ___

2. Руководитель ***Садчиков Илья Александрович***
фамилия, имя, отчество полностью

_____ _____ *старший преподаватель* *РГППУ*
ученая степень ученое звание должность место работы

3. Место преддипломной практики *РГППУ, ИПО, кафедра информационных систем и технологий*

4. Исходные данные к ВКР *Книги по теории создания игр, книги по разработке игрового ПО, индустриальные интернет-сайты, youtube каналы про разработку игр*

5. Содержание текстовой части ВКР (перечень подлежащих разработке вопросов)

выбор жанра для разрабатываемого проекта, анализ похожих существующих разработок, выбор программного обеспечения, создание концепта игры, описание разработки продукта.

6. Перечень демонстрационных материалов *презентация выполненная в MS Power Point, обзорный видеоролик, игра «Be Light»*

7. Календарный план выполнения выпускной квалификационной работы

№ п/п	Наименование этапа дипломной работы	Срок выполнения этапа	Процент выполнения ВКР	Отметка руководителя о выполнении
1	Сбор информации по выпускной квалификационной работе	23.04.2018	5%	подпись
2	Выполнение работ по разрабатываемым вопросам и их изложение в пояснительной записке:	03.05.2018	65%	подпись
2.1	Анализ существующих разработок	03.05.2018	5%	подпись
2.2	Изучение теории и методов работы со необходимым программным обеспечением	04.05.2018	10%	подпись
2.3	Разработка игры	05.05.2018	25%	подпись
2.4	Тестирование и отладка	18.05.2018	20%	подпись
2.5	Компиляция игры	28.05.2018	5%	подпись
3	Оформление текстовой части ВКР	29.05.2018	10%	подпись
4	Выполнение демонстрационных материалов к ВКР	01.06.2018	10%	подпись
5	Нормоконтроль	06.06.2018	5%	подпись
6	Подготовка доклада к защите в ГЭК	13.06.2018	5%	подпись

8. Консультанты по разделам выпускной квалификационной работы

Наименование раздела	Консультант	Задание выдал		Задание принял	
		подпись	дата	подпись	дата

Руководитель _____ _____ *Задание получил* _____ _____
подпись дата подпись студента дата

9. Дипломная работа и все материалы проанализированы.
Считаю возможным допустить ***Ганиеву Д.Р.*** к защите выпускной квалификационной работы в государственной экзаменационной комиссии.

Руководитель _____ _____
подпись дата

10. Допустить ***Ганиеву Д.Р.*** к защите выпускной квалификационной работы
фамилия и. о. студента

в государственной экзаменационной комиссии (протокол заседания кафедры от «___» _____ 20___г., № _____)

Заведующий кафедрой _____ _____
подпись дата

ПРИЛОЖЕНИЕ Б

Скрипт ManagerPlatform

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

public class ManagerPlatform : MonoBehaviour
{
    [SerializeField]
    ForGen forgen;
    public Vector3 GenPos;//текущая позиция генерации
    float LastGenPos;//последняя позиция генератора

    public GameObject[] f = new GameObject[41]; //все префабы платформ в лесу
    public GameObject gr;//земля
    Vector3 GenPosGr;//текущая позиция генерации земли
    float lastGroundPos;//последняя позиция генерации земли
    public float GroundLenght = 4F;//длинна объекта земли

    float LastPos;//Координата последней сгенерированной платформы
    Vector3 PlatGenPos;//следующая выбранная позиция генерации

    Quaternion GenQ = new Quaternion(0, 0, 0, 0);//стандартный поворот объектов

    byte method;//номер метода платформы, вызывает соответствующий метод генерации плат-
формы
    bool EndStartPack;//флаг, показывающий закончились ли настроеные платформы

    System.Random rnd = new System.Random();
    int RndStep;//рандомный шаг для создания следущей платформы, ОБЯЗАТЕЛЬНО КРАТНО 2
    int RndPak;//из какого набора платформ выбирать
    int RndVid;//какую конкретно платформу из набора выбрать

    void Start()
    {
        EndStartPack = false;
        GenPosGr = Vector3.zero;
        GenPos = Vector3.zero;
        LastPos = 0;
        RndStep = 1;
    }

    void Update()
    {
        //начинаем ли генерацию
        if (EndStartPack == false)
        {//устанановка начального положения генерации
            Collider2D[] colliders = Physics2D.OverlapCircleAll(new Vec-
tor3(forgen.transform.position.x,-1F), 0.1F, 1 << 13);
            if ((colliders.Length < 1)&&(GenPos.x%2==0))
            {
                EndStartPack = true;
                GenPosGr = new Vector3((forgen.transform.position.x + Stat-
ic.StepGenGround), 0);
                GenPos = new Vector3((forgen.transform.position.x + Static.StepPlatf),
0);
                lastGroundPos = GenPosGr.x-4F;
                LastPos = forgen.transform.position.x + Static.StepPlatf;
                LastGenPos = GenPos.x - 2F;
```

```
                RndStep = 0;
            }
        }
        else
        {
            GenPosGr.x = forgen.transform.position.x;
            GenPos.x = forgen.transform.position.x + Static.StepPlatf;

            #region ground
            if (lastGroundPos + GroundLenght < forgen.transform.position.x)
            {
                lastGroundPos += GroundLenght;
                GameObject ground = PoolManager.GetObject(gr.name, new Vec-
tor3(lastGroundPos, GenPosGr.y - 3.0F), GenQ);
            }
            #endregion

            #region Platforms
            if ((LastGenPos+2F<=GenPos.x)&&(LastPos + RndStep <= GenPos.x))//каждую чет-
ную позицию генерации вызывай это
            {
                LastGenPos += 2;
                switch (method)
                {
                    case 0: F0(); break;
                    case 3: F3(); break;
                    case 6: F6(); break;
                    case 9: F9(); break;
                    case 12: F12(); break;
                    case 15: F15(); break;
                    case 18: F18(); break;
                    case 21: F21(); break;
                    case 24: F24(); break;
                    case 27: F27(); break;
                    case 30: F30(); break;
                    case 33: F33(); break;
                    case 36: F36(); break;
                    case 39: F39(); break;
                }
            }
            #endregion
        }
    }

    public int GetRandom(params int[] values)//для выбора рандомного числа из предложеных
(используется в методах создания платформ)
    {
        if (values != null)
        { return values[rnd.Next(values.Length)]; }
        else { return 4; }
    }

    void GenPlat(GameObject a1, GameObject a2, GameObject a3)//универсальный метод ран-
домного создания платформ из набора подобных
    {
        RndVid = rnd.Next(2);
        switch (RndVid)//выбираем внешний вид текущей платформы платформы
        {
            case 0: GameObject ForestPlatform = PoolManager.GetObject(a1.name, new Vec-
tor3(GenPos.x, a1.transform.position.y), GenQ); break;
            case 1: GameObject ForestPlatform2 = PoolManager.GetObject(a2.name, new Vec-
tor3(GenPos.x, a2.transform.position.y), GenQ); break;
            case 2: GameObject ForestPlatform3 = PoolManager.GetObject(a3.name, new Vec-
tor3(GenPos.x, a3.transform.position.y), GenQ); break;
        }
```

```
            LastPos = GenPos.x;
        }


        #region platforms descriptions
        void F0()
        {
            GenPlat(f[0], f[1], f[2]);//Генерирование рандомной платформы из набора
            RndPak = rnd.Next(12);//число для выбора следующего набора платформ
            switch (RndPak)//какая платформу можно поставить вслед за только что сгенериро-
ванной?
            {//выбираем метод по номеру первой платформы в наборе
                case 0: RndStep = GetRandom(2, 4, 8); method = 0; break;
                case 1: RndStep = GetRandom(2, 4, 8); method = 3; break;
                case 2: RndStep = GetRandom(2, 4, 8); method = 6; break;
                case 3: RndStep = 2; method = 9; break;
                case 4: RndStep = 2; method = 12; break;
                case 5: RndStep = 2; method = 15; break;
                case 6: RndStep = GetRandom(2, 4, 8); method = 24; break;
                case 7: RndStep = GetRandom(4, 8); method = 27; break;
                case 8: RndStep = GetRandom(4, 8); method = 30; break;
                case 9: RndStep = GetRandom(4, 8); method = 33; break;
                case 10: RndStep = 2; method = 36; break;
                case 11: RndStep = 2; method = 39; break;
            }
            RndStep = RndStep + 4;//увеличиваем шаг на длинну созданной платформы
        }

        void F3()
        {
            GenPlat(f[3], f[4], f[5]);
            RndPak = rnd.Next(12);
            switch (RndPak)
            {
                case 0: RndStep = GetRandom(2, 4, 8); method = 0; break;
                case 1: RndStep = GetRandom(2, 4, 8); method = 3; break;
                case 2: RndStep = GetRandom(2, 4, 8); method = 6; break;
                case 3: RndStep = 2; method = 9; break;
                case 4: RndStep = 2; method = 12; break;
                case 5: RndStep = GetRandom( 2); method = 15; break;
                case 6: RndStep = GetRandom(2, 4, 8); method = 24; break;
                case 7: RndStep = GetRandom(4, 8); method = 27; break;
                case 8: RndStep = GetRandom(4, 8); method = 30; break;
                case 9: RndStep = GetRandom(4, 8); method = 33; break;
                case 10: RndStep = 2; method = 36; break;
                case 11: RndStep = 2; method = 39; break;
            }
            RndStep = RndStep + 8;
        }

        void F6()
        {
            GenPlat(f[6], f[7], f[8]);
            RndPak = rnd.Next(14);
            switch (RndPak)
            {
                case 0: RndStep = GetRandom( 2, 4, 8); method = 0; break;
                case 1: RndStep = GetRandom( 2, 4, 8); method = 3; break;
                case 2: RndStep = GetRandom( 2, 4, 8); method = 6; break;
                case 3: RndStep = 2; method = 9; break;
                case 4: RndStep = 2; method = 12; break;
                case 5: RndStep = 2; method = 15; break;
                case 6: RndStep = 2; method = 18; break;
                case 7: RndStep = 2; method = 21; break;
                case 8: RndStep = GetRandom( 2, 4, 8); method = 24; break;
```

```
        case 9: RndStep = GetRandom(2, 4, 8); method = 27; break;
        case 10: RndStep = GetRandom(2, 4, 8); method = 30; break;
        case 11: RndStep = GetRandom(2, 4, 8); method = 33; break;
        case 12: RndStep = 2; method = 36; break;
        case 13: RndStep = 2; method = 39; break;
    }
    RndStep = RndStep + 8;
}

void F9()
{
    GenPlat(f[9], f[10], f[11]);
    RndPak = rnd.Next(10);
    switch (RndPak)
    {
        case 0: RndStep = GetRandom( 2, 4, 8); method = 0; break;
        case 1: RndStep = GetRandom( 2, 4, 8); method = 3; break;
        case 2: RndStep = GetRandom( 2, 4, 8); method = 6; break;
        case 3: RndStep = 2; method = 15; break;
        case 4: RndStep = GetRandom(2, 4, 8); method = 24; break;
        case 5: RndStep = GetRandom(2, 4, 8); method = 27; break;
        case 6: RndStep = GetRandom(2, 4, 8); method = 30; break;
        case 7: RndStep = GetRandom(2, 4, 8); method = 33; break;
        case 8: RndStep = 2; method = 36; break;
        case 9: RndStep = 2; method = 39; break;
    }
    RndStep = RndStep + 4;
}

void F12()
{
    GenPlat(f[12], f[13], f[14]);
    RndPak = rnd.Next(10);
    switch (RndPak)
    {
        case 0: RndStep = GetRandom( 2, 4, 8); method = 0; break;
        case 1: RndStep = GetRandom( 2, 4, 8); method = 3; break;
        case 2: RndStep = GetRandom( 2, 4, 8); method = 6; break;
        case 3: RndStep = 2; method = 15; break;
        case 4: RndStep = GetRandom( 2, 4, 8); method = 24; break;
        case 5: RndStep = GetRandom( 2, 4, 8); method = 27; break;
        case 6: RndStep = GetRandom( 2, 4, 8); method = 30; break;
        case 7: RndStep = GetRandom( 2, 4, 8); method = 33; break;
        case 8: RndStep = 2; method = 36; break;
        case 9: RndStep = 2; method = 39; break;
    }
    RndStep = RndStep + 8;
}


void F15()
{
    GenPlat(f[15], f[16], f[17]);
    RndPak = rnd.Next(12);
    switch (RndPak)
    {
        case 0: RndStep = GetRandom( 2, 4, 8); method = 0; break;
        case 1: RndStep = GetRandom( 2, 4, 8); method = 3; break;
        case 2: RndStep = GetRandom( 2, 4, 8); method = 6; break;
        case 3: RndStep = 2; method = 15; break;
        case 4: RndStep = 2; method = 18; break;
        case 5: RndStep = 2; method = 21; break;
        case 6: RndStep = GetRandom( 2, 4, 8); method = 24; break;
        case 7: RndStep = GetRandom( 2, 4, 8); method = 27; break;
        case 8: RndStep = GetRandom( 2, 4, 8); method = 30; break;
```

```
        case 9: RndStep = GetRandom( 2, 4, 8); method = 33; break;
        case 10: RndStep = 2; method = 36; break;
        case 11: RndStep = 2; method = 39; break;
    }
    RndStep = RndStep + 8;
}

void F18()
{
    GenPlat(f[18], f[19], f[20]);
    RndPak = rnd.Next(12);
    switch (RndPak)
    {
        case 0: RndStep = GetRandom(2, 4, 8); method = 0; break;
        case 1: RndStep = GetRandom(2, 4, 8); method = 3; break;
        case 2: RndStep = GetRandom(2, 4, 8); method = 6; break;
        case 3: RndStep = 2; method = 15; break;
        case 4: RndStep = 2; method = 18; break;
        case 5: RndStep = 2; method = 21; break;
        case 6: RndStep = GetRandom(2, 4, 8); method = 24; break;
        case 7: RndStep = GetRandom(2, 4, 8); method = 27; break;
        case 8: RndStep = GetRandom(2, 4, 8); method = 30; break;
        case 9: RndStep = GetRandom(2, 4, 8); method = 33; break;
        case 10: RndStep = 2; method = 36; break;
        case 11: RndStep = 2; method = 39; break;
    }
    RndStep = RndStep + 4;
}

void F21()
{
    GenPlat(f[21], f[22], f[23]);
    RndPak = rnd.Next(12);
    switch (RndPak)
    {
        case 0: RndStep = GetRandom(2, 4, 8); method = 0; break;
        case 1: RndStep = GetRandom(2, 4, 8); method = 3; break;
        case 2: RndStep = GetRandom(2, 4, 8); method = 6; break;
        case 3: RndStep = 2; method = 15; break;
        case 4: RndStep = 2; method = 18; break;
        case 5: RndStep = 2; method = 21; break;
        case 6: RndStep = GetRandom(2, 4, 8); method = 24; break;
        case 7: RndStep = GetRandom(2, 4, 8); method = 27; break;
        case 8: RndStep = GetRandom(2, 4, 8); method = 30; break;
        case 9: RndStep = GetRandom(2, 4, 8); method = 33; break;
        case 10: RndStep = 2; method = 36; break;
        case 11: RndStep = 2; method = 39; break;
    }
    RndStep = RndStep + 8;
}

void F24()
{
    GenPlat(f[24], f[25], f[26]);
    RndPak = rnd.Next(14);
    switch (RndPak)
    {
        case 0: RndStep = GetRandom( 2, 4, 8); method = 0; break;
        case 1: RndStep = GetRandom( 2, 4, 8); method = 3; break;
        case 2: RndStep = GetRandom( 2, 4, 8); method = 6; break;
        case 3: RndStep = 2; method = 9; break;
        case 4: RndStep = 2; method = 12; break;
        case 5: RndStep = 2; method = 15; break;
        case 6: RndStep = 2; method = 18; break;
        case 7: RndStep = 2; method = 21; break;
```

```
        case 8: RndStep = GetRandom( 2, 4, 8); method = 24; break;
        case 9: RndStep = GetRandom( 2, 4, 8); method = 27; break;
        case 10: RndStep = GetRandom( 2, 4, 8); method = 30; break;
        case 11: RndStep = GetRandom( 2, 4, 8); method = 33; break;
        case 12: RndStep = 2; method = 36; break;
        case 13: RndStep = 2; method = 39; break;
    }
    RndStep = RndStep + 12;
}

void F27()
{
    GenPlat(f[27], f[28], f[29]);
    RndPak = rnd.Next(12);
    switch (RndPak)
    {
        case 0: RndStep = GetRandom(2, 4, 8); method = 0; break;
        case 1: RndStep = GetRandom(2, 4, 8); method = 3; break;
        case 2: RndStep = GetRandom(2, 4, 8); method = 6; break;
        case 3: RndStep = 2; method = 9; break;
        case 4: RndStep = 2; method = 12; break;
        case 5: RndStep = 2; method = 15; break;
        case 6: RndStep = GetRandom(4, 8); method = 24; break;
        case 7: RndStep = GetRandom(4, 8); method = 27; break;
        case 8: RndStep = GetRandom(4, 8); method = 30; break;
        case 9: RndStep = GetRandom(4, 8); method = 33; break;
        case 10: RndStep = 2; method = 36; break;
        case 11: RndStep = 2; method = 39; break;
    }
    RndStep = RndStep + 4;
}

void F30()
{
    GenPlat(f[30], f[31], f[32]);
    RndPak = rnd.Next(12);
    switch (RndPak)
    {
        case 0: RndStep = GetRandom(2, 4, 8); method = 0; break;
        case 1: RndStep = GetRandom(2, 4, 8); method = 3; break;
        case 2: RndStep = GetRandom(2, 4, 8); method = 6; break;
        case 3: RndStep = 2; method = 9; break;
        case 4: RndStep = 2; method = 12; break;
        case 5: RndStep = 2; method = 15; break;
        case 6: RndStep = GetRandom(4, 8); method = 24; break;
        case 7: RndStep = GetRandom(4, 8); method = 27; break;
        case 8: RndStep = GetRandom(4, 8); method = 30; break;
        case 9: RndStep = GetRandom(4, 8); method = 33; break;
        case 10: RndStep = 2; method = 36; break;
        case 11: RndStep = 2; method = 39; break;
    }
    RndStep = RndStep + 8;
}

void F33()
{
    GenPlat(f[33], f[34], f[35]);
    RndPak = rnd.Next(12);
    switch (RndPak)
    {
        case 0: RndStep = GetRandom(2, 4, 8); method = 0; break;
        case 1: RndStep = GetRandom(2, 4, 8); method = 3; break;
        case 2: RndStep = GetRandom(2, 4, 8); method = 6; break;
        case 3: RndStep =2; method = 15; break;
        case 4: RndStep = 2; method = 18; break;
```

```
            case 5: RndStep = 2; method = 21; break;
            case 6: RndStep = GetRandom(2, 4, 8); method = 24; break;
            case 7: RndStep = GetRandom(4, 8); method = 27; break;
            case 8: RndStep = GetRandom(4, 8); method = 30; break;
            case 9: RndStep = GetRandom(4, 8); method = 33; break;
            case 10: RndStep = 2; method = 36; break;
            case 11: RndStep = 2; method = 39; break;
        }
        RndStep = RndStep + 8;
    }

    void F36()
    {
        GenPlat(f[36], f[37], f[38]);
        RndPak = rnd.Next(12);
        switch (RndPak)
        {
            case 0: RndStep = 2; method = 0; break;
            case 1: RndStep = 2; method = 3; break;
            case 2: RndStep = 2; method = 6; break;
            case 3: RndStep = 2; method = 15; break;
            case 4: RndStep = 2; method = 18; break;
            case 5: RndStep = 2; method = 21; break;
            case 6: RndStep = GetRandom(2, 4, 8); method = 24; break;
            case 7: RndStep = GetRandom(4, 8); method = 27; break;
            case 8: RndStep = GetRandom(4, 8); method = 30; break;
            case 9: RndStep = GetRandom(4, 8); method = 33; break;
            case 10: RndStep = 2; method = 36; break;
            case 11: RndStep = 2; method = 39; break;
        }
        RndStep = RndStep + 4;
    }

    void F39()
    {
        GenPlat(f[39], f[40], f[41]);
        RndPak = rnd.Next(12);
        switch (RndPak)
        {
            case 0: RndStep = 2; method = 0; break;
            case 1: RndStep = 2; method = 3; break;
            case 2: RndStep = 2; method = 6; break;
            case 3: RndStep = 2; method = 15; break;
            case 4: RndStep = 2; method = 18; break;
            case 5: RndStep = 2; method = 21; break;
            case 6: RndStep = GetRandom(2, 4, 8); method = 24; break;
            case 7: RndStep = GetRandom(4, 8); method = 27; break;
            case 8: RndStep = GetRandom(4, 8); method = 30; break;
            case 9: RndStep = GetRandom(4, 8); method = 33; break;
            case 10: RndStep = 2; method = 36; break;
            case 11: RndStep = 2; method = 39; break;
        }
        RndStep = RndStep + 8;
    }
    #endregion
}
```

ПРИЛОЖЕНИЕ В

Скрипт ManagerPlatformType2

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

public class ManagerPlatformType2 : MonoBehaviour {

    public GameObject[] sky = new GameObject[13]; //все платформы
    public float[] lenghts = new float[13]; //длины платформ
    public GameObject gr;//границы уровня
    Vector3 GenPos;//текущая позиция генерации
    Quaternion GenQ = new Quaternion(0, 0, 0, 0);//текущий разворот
    [SerializeField]
    ForGen forgen;//ссылка на генератор
    bool EndStartPack=false; //проверка на то, нужно ли начинать генерацию

    float y1 = 13F;//высота платформ 1
    float y2 = 17F;//высота платформ 2
    float x1;//расстояние между платформами на высоте 1
    float x2;//расстояние между платформами на высоте 2
    float x1last;//расстояние между платформами предыдущее для высоты 1
    float x2last;//расстояние между платформами предыдущее для высоты 2

    public float maxstep=10;//максимальный шаг между платформами
    int i; //счетчик
    float lastGroundPos;//последняя позиция генерации земли
    public float GroundLenght = 4F; //длина объекта земля

    void Start()
    {
        GameObject player = GameObject.Find("Player2(Clone)");
        forgen = player.transform.Find("Generator").GetComponent<ForGen>();
    }

    void Update()
    {
        //начинать ли генерацию платформ?
        if (EndStartPack == false)
        {//код когда генерация еще не началась
            Collider2D[] colliders = Physics2D.OverlapCircleAll(new Vec-
tor3(forgen.transform.position.x-GroundLenght, 9.5F), 0.1F,1<<12);
            if (colliders.Length<1)
            {
                EndStartPack = true;
                GenPos = new Vector3((forgen.transform.position.x + Static.StepPlatf),
0);
                lastGroundPos = forgen.transform.position.x;
            }
        }
        else
        {//код генерации
            GenPos.x = forgen.transform.position.x + Static.StepPlatf;

            if (lastGroundPos + GroundLenght < GenPos.x)
            {
                //генерирование границ
                GameObject Gran = PoolManager.GetObject(gr.name, new Vec-
tor3(lastGroundPos, 9), GenQ);
```

```
                //генерирование платформ
                GenPlat(ref x1, ref y1, ref x1last);
                GenPlat(ref x2, ref y2, ref x2last);
                lastGroundPos += GroundLenght;
            }
        }
    }

    void GenPlat(ref float x,ref float y,ref float xlast)//метод, генерирующий платформу
    {
        if (GenPos.x >=xlast+x)
        {
            xlast = GenPos.x;
            i = Random.Range(0, sky.Length);//выбираем любую платформу
            GameObject ForestPlatform = PoolManager.GetObject(sky[i].name, new Vec-
tor3(GenPos.x, Random.Range(-1,2)+y), GenQ);//создаем ее
            x = Random.Range(2, maxstep-4) + lenghts[i];//задаем через сколько нужно бу-
дет сделать следующую платформу на этом уровне высоты
        }
    }
   }
```

ПРИЛОЖЕНИЕ Г

Скрипт ManagerFire

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

public class ManagerFire : MonoBehaviour {


    [SerializeField]
    FireSphere FireSpherePrefab;
    ForGen forgen;

    System.Random rnd = new System.Random();
    int RndCol;//количество огоньков на одной высоте
    int RndY;//случайная высота
    float LastPos;//последняя позиция генерации
    //текущая позиция генерации по оси x и по оси y
    float YPos;
    float XPos;
    //текущий уровень локации
    [SerializeField]
    int lvl;
    [SerializeField]
    float minY;//самая низкая допустимая высота для этой локации
    [SerializeField]
    float maxY;//самая высокая точка генерации для этой локации
    int zaderzka=6;//расстояние, на котором генерируются огоньки, относительно генератора

    void Start()
    {
        LastPos = 20;
        RndCol = 3;
        YPos = 0.5F;
    }

    void Update()
    {
        if (forgen == null)
        {
            if (GameObject.Find("Player"))
            {
                forgen = GameOb-
ject.Find("Player").GetComponent<Character>().transform.Find("Generator").GetComponent<Fo
rGen>();
            }
            else
            {
                forgen = GameOb-
ject.Find("Player2(Clone)").GetComponent<Character>().transform.Find("Generator").GetComp
onent<ForGen>();
            }
        }

        switch (lvl)
        {
            case 1:
                {
                    if (forgen.transform.position.x - zaderzka > LastPos + 0.5)
```

```
                    {
                        if (RndCol < 1)
                        {
                            RndY = rnd.Next(7);//7 вида высоты
                            switch (RndY)//выбор высоты генерации
                            {
                                case 0: YPos = 0F; break;
                                case 1: YPos = 2F; break;
                                case 2: YPos = 4F; break;
                                case 3: YPos = 6F; break;
                                case 4: YPos = -1F; break;//будет пусто
                                case 5: YPos = -1F; break;//будет пусто
                                case 6: YPos = -1F; break;//будет пусто
                            }
                            YPos = YPos + (float)(rnd.NextDouble()) / 2 + 0.25F;//от 0,25
до 0,75
                            RndCol = rnd.Next(4) + 2;//максимально на 1 высоте
                        }
                        RndCol--;

                        XPos = forgen.transform.position.x - zaderzka;
                        //круг вокруг нижней линии персонажа. если в него попадают колай-
деры то массив заполняется ими
                        Collider2D[] colliders = Physics2D.OverlapCircleAll(new Vec-
tor2(XPos, YPos - 0.5F), 0.2F);
                        Collider2D[] nocolliders = Physics2D.OverlapCircleAll(new Vec-
tor2(XPos, YPos), 0.2F);//чтобы на месте создания не было коллайдеров

                        if ((colliders.Length > 0) && (nocolliders.Length == 0) && (YPos
> minY) && (YPos < maxY))//проверка на близость других коллайдеров и существование высоты
                        {
                            FireSphere firesphere = PoolManag-
er.GetObject(FireSpherePrefab.name, new Vector2(XPos, YPos), FireSpherePre-
fab.transform.rotation).GetComponent<FireSphere>();
                        }
                        LastPos = forgen.transform.position.x - zaderzka;
                    }
                    break;
                }
            case 2:
                {
                    if (forgen.transform.position.x - zaderzka > LastPos + 0.5)
                    {
                        if (RndCol < 1)
                        {
                            RndY = rnd.Next(7);//7 вида высоты
                            switch (RndY)//выбор высоты генерации
                            {
                                case 0: YPos = 12F; break;
                                case 1: YPos = 14F; break;
                                case 2: YPos = 16F; break;
                                case 3: YPos = 18F; break;
                                case 4: YPos = -1F; break;//будет пусто
                                case 5: YPos = -1F; break;//будет пусто
                                case 6: YPos = -1F; break;//будет пусто
                            }
                            YPos = YPos + (float)(rnd.NextDouble()) / 2 + 0.25F;//от 0,25
до 0,75
                            RndCol = rnd.Next(4) + 2;//максимально на 1 высоте
                        }
                        RndCol--;
                        XPos = forgen.transform.position.x - zaderzka;

                        Collider2D[] nocolliders = Physics2D.OverlapCircleAll(new Vec-
tor2(XPos, YPos), 0.4F);
```

```
                        if ((nocolliders.Length == 0) && (YPos > minY) && (YPos <
maxY))//проверка на близость других коллайдеров и существование высоты
                        {
                            FireSphere firesphere = PoolManag-
er.GetObject(FireSpherePrefab.name, new Vector2(XPos, YPos), FireSpherePre-
fab.transform.rotation).GetComponent<FireSphere>();
                        }
                        LastPos = forgen.transform.position.x - zaderzka;
                    }
                    break;
                }
case 3:
                {
                    if (forgen.transform.position.x - zaderzka > LastPos + 0.5)
                    {
                        if (RndCol < 1)
                        {
                            RndY = rnd.Next(7);//7 вида высоты
                            switch (RndY)//выбор высоты генерации
                            {
                                case 0: YPos = 22F; break;
                                case 1: YPos = 24F; break;
                                case 2: YPos = 26F; break;
                                case 3: YPos = 28F; break;
                                case 4: YPos = -1F; break;//будет пусто
                                case 5: YPos = -1F; break;//будет пусто
                                case 6: YPos = -1F; break;//будет пусто
                            }
                            YPos = YPos + (float)(rnd.NextDouble()) / 2 + 0.25F;//от 0,25
до 0,75
                            RndCol = rnd.Next(4) + 2;//максимально на 1 высоте
                        }
                        RndCol--;
                        XPos = forgen.transform.position.x - zaderzka;

                        Collider2D[] nocolliders = Physics2D.OverlapCircleAll(new Vec-
                        tor2(XPos, YPos), 0.4F);

                        if ((nocolliders.Length == 0) && (YPos > minY) && (YPos <
maxY))//проверка на близость других коллайдеров и существование высоты
                        {
                            FireSphere firesphere = PoolManag-
er.GetObject(FireSpherePrefab.name, new Vector2(XPos, YPos), FireSpherePre-
fab.transform.rotation).GetComponent<FireSphere>();
                        }
                        LastPos = forgen.transform.position.x - zaderzka;
                    }
                    break;
                }
        }
    }
}
```

ПРИЛОЖЕНИЕ Д

Скрипт ManagerUnits

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

public class ManagerUnits : MonoBehaviour {


    [SerializeField]
    ForGen forgen; //ссылка на генератор
    float GenPos;//текущая позиция генерации

    [SerializeField]
    GameObject[] monsters = new GameObject[3];//префабы монстров
    [SerializeField]
    int[] chance = new int[3];//массив шансов выпадения определенного вида монстра
    [SerializeField]
    float[] Hights=new float[3];//позиции генерации по Y для конкретного уровня

    int RndStep;//случайный шаг для генерации следующего монстра
    int RndPack;//число для выбора конкретного монстра (взаимодействует с массивом шансов
их выпадения)
    int RndY;//текущая случайная высота генерации монстра
    float LastPos;//последняя позиция генерации монстра

    //счетчики
    int i;
    int j;
    int sum;

    Collider2D[] colliders;//для опредения объектов в месте генерации монстра

    void Start()
    {
        GameObject player = GameObject.FindWithTag("Player");
        forgen = player.transform.Find("Generator").GetComponent<ForGen>();
        RndStep = 0;
        GenPos = Static.ForgenPosition+ Static.StepGenMonster;
        LastPos = GenPos;
    }

    void Update()
    {
        GenPos = forgen.transform.position.x + Static.StepGenMonster;
        if (GenPos > RndStep + LastPos)
        {
            RndPack = Random.Range(0,100);
            RndY = Random.Range(0, Hights.Length);//выбор случайной высоты
            i = -1;
            sum=0;
            //ищем в диапазоне какокого элемента массива шансов выпало рандомное значение
            for (j=0;j< chance.Length;)
            {
                if (RndPack - chance[j] - sum < 0)
                {
                    i = j;
                    colliders = Physics2D.OverlapCircleAll(new Vector3((RndStep + Last-
Pos), Hights[RndY]), 0.3F, 1 << 13);
```

```
                    j = chance.Length;
                }
                else
                {
                    sum += chance[j];
                    j++;
                }
            }
            if ((i != -1)&&(colliders.Length==0))//если рандом попал в диапазон  и на ме-
сте генерации нет объектов
            {
                GameObject monster = PoolManager.GetObject(monsters[i].name, new Vec-
tor3((RndStep + LastPos), Hights[RndY]), monsters[i].transform.rotation);
            }
            RndStep = Random.Range(5,20);
            LastPos = GenPos;
        }
    }
    }
```