Министерство образования и науки Российской Федерации
Федеральное государственное автономное образовательное учреждение
высшего образования
«Российский государственный профессионально-педагогический университет»

КОМПЬЮТЕРНАЯ ИГРА В ЖАНРЕ СКРОЛЛЕР

Выпускная квалификационная работа
по направлению подготовки 09.03.02 Информационные системы
и технологии
профилю подготовки «Информационные технологии в медиаиндустрии»

Идентификационный номер ВКР: 301

Екатеринбург 2017

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
«____» _____ 2017 г.

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

КОМПЬЮТЕРНАЯ ИГРА В ЖАНРЕ СКРОЛЛЕР

Исполнитель:
обучающийся группы № ИТм-401 А. А. Борисов
Руководитель:
ст. преподаватель И. А. Садчиков
Нормоконтролер: Б. А. Редькина

Екатеринбург 2017

АННОТАЦИЯ

Выпускная квалификационная работа состоит из компьютерной игры и пояснительной записки на 60 страницах, содержащей 25 рисунков, 2 таблицы, 29 источников литературы, а также 1 приложений на 1 страницах.

Ключевые слова: ИГРА, ВИДЕОИГРЫ, КОМПЬЮТЕРНАЯ ИГРА, РАЗРАБОТКА, ПРОЕКТ

Борисов, А. А. Компьютерная игра в жанре скроллер: выпускная квалификационная работа / А. А. Борисов; Рос. гос. проф.-пед. ун-т, Ин-т инж.-пед. образования, Каф. информ. систем и технологий. — Екатеринбург, 2017. — 60 с.

В работе рассмотрены вопросы разработки компьютерных игр.

Целью работы является создание компьютерной игры в жанре скроллер. Для достижения цели были проанализированы современные тенденции и методы разработки компьютерных игр, проанализирована рыночная среда и компьютерные игры конкурентов. Основываясь на данных материалах, была разработана компьютерная игра в жанре скроллер.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В современном мире индустрия цифровых развлечений занимает не последнее место в жизни людей. Видеоигры уже стали одним из самых популярных видов развлечений, а также являются частью мировой культуры. Тем не менее они продолжают развиваться и набирать популярность. Доходы крупных компаний на рынке компьютерных игр достигают миллиардов долларов. Упрощаются средства цифровой дистрибуции, открываются новые возможности для развития новым разработчикам, сектор так называемого indie (independent) рынка растёт с каждым годом. И это касается лишь игр, как средства развлечения и времяпрепровождения. Видеоигры достаточно быстро развиваются как средства обучения, набирая популярность в абсолютно разных сферах. От обучающих игр для детей дошкольного и школьного возраста, до своеобразных симуляторов ЖД механика. Новые разработки в сфере видеоигр всё сильнее стирают грань между играми как развлечением и играми, как инструмент для развития.

Видеоигры уходят корнями далеко в 1950-е годы, когда ещё не существовало ни персональных компьютеров, ни термина «видеоигры». Игры разрабатывались в качестве эксперимента, научной работы, или просто так, потому что это было интересно.

В 1972 году выходит первая игровая консоль The Magnavox Odyssey. Коммерчески успешна, превысив 350000 продаж.

1978-1979 – выход двух легендарных игр Space Invaders и Pac-Man. Обе игры оказали колоссальное влияние на дальнейшие развитие индустрии. Space Invaders пробудил интерес людей к видеоиграм и программированию, доказав, что игры могут противостоять другим крупным индустриям, например, кино. Space Invaders обрела популярность, как у себя на родине в Японии, так и за рубежом, принеся Atari за 30 лет 500000000 долларов. Pac-Man же стал одним из самых узнаваемых персонажей видео игр. Игра обрела по-

пулярность по всему миру, и была портрирована почти на все платформы, использовав при этом отличный от вездесущего на тот момент космического стиля. Игра не была ни шутером, ни pong-системой, и показала абсолютно новый на тот момент жанр, без насилия, и покрывала гораздо большую аудиторию. В дальнейшем это подтолкнуло разработчиков видеоигр на создание нестандартных аркад.

С 1980-х начинается эра персональных компьютеров, и переход игр с аркадных автоматов на ПК. В 80-х выходят игры, так же формирующие индустрию, и более того, некоторые серии существуют и разрабатываются до сих пор. Из игр тех лет можно выделить: The Legend of Zelda, вышедшею впервые в 1986 году, а последняя часть вышла в 2017, и игра не теряет популярности, Final Fantasy, в жанре JRPG, впервые выпущенная в 1987, и имеющая на сегодняшний день целую серию игр в разных жанрах и на разных платформах. Donkey Kong же не продолжился как серия, однако, положил начало для не менее легендарной серии игр про Mario, насчитывающую на сегодняшний день больше 20 игр, на множестве платформ, в самых разных жанрах.

С 1990-х по сегодняшний день сформировалось множество жанров в игровой индустрии, и каждый находит множество последователей. Игровая индустрия не имеет долгой истории, однако даже за столь короткий промежуток, потерпела сильные изменения, за почти 70 лет, игры сменили множество платформ, перейдя с университетских экспериментальных ЭВМ на персональные компьютеры, консоли и даже смартфоны. Развиваются площадки, для поддержки независимых разработчиков и цифровой дистрибуции игр. На сегодняшний день видеоигры уже имеют огромную популярность, и продолжают завоёвывать всё новых фанатов. И чем выше количество играющих людей, тем больше спрос на новые игры.

Объект исследования: видеоигры.

Предмет исследования: разработка видеоигры.

Цель выпускной квалификационной работы: разработка игры с применением тригонометрических функций.

Задачи:

- проанализировать предметную область и разработки внутри данной области;
- выбрать программные средства разработки на основе анализа представленного программного обеспечения;
- проанализировать тонкости работы с выбранным программным обеспечением;
- на основе выбранного жанра, реализовать программный продукт с использованием выбранных программных средств.

1 АНАЛИТИЧЕСКАЯ ЧАСТЬ

1.1 Анализ и общая характеристика предметной области

1.1.1 Терминология, типы и жанры игр для разработки

Так как видеоигры являются отдельной, независимой индустрией, со своими особенностями, существует множество терминов, которые относятся исключительно к видео играм. Более того, за счёт засилья множества жанров, сообществом и самими разработчиками было введено ещё больше понятий, для более узких моментов.

Видеоигра — электронная игра, генерирующая отклик, на основе взаимодействия с пользователем, в виде изображения на видеоустройство, как телевизор или монитор.

Разработка видеоигр — процесс создания видео игры, один человеком, или группой лиц, для одной или нескольких платформ. Для компаний характерно разбиение деятельности между сотрудниками, для реализации разных аспектов. Отдельные отделы или люди могу заниматься реализацией дизайна, графической составляющей, программированием, тестированием и прочим.

Разработчик видеоигр — разработчик, специализирующийся на разработке видеоигр и связанных с этим компонентов. Разработчиком может являться как один человек, так и компания.

Издатель – компания, финансирующая или заказывающая создание видео игры, для последующего распространения. Разработчики могут самостоятельно издавать свои игры, используя средства цифровой дистрибуции, минуя издателя.

Платформа — устройство, используемое для игры, например, персональный компьютер, или консоль.

Игровой контроллер — устройство ввода, используемое в играх или игровых системах, для обеспечения управления объектом или персонажем игры.

Игровой движок — набор систем, упрощающий часто используемые функции игры. Системы контролируют отдельные аспекты игры, например: графическая система, аудио система, система ввода. Так же имеет своё системное ядро. При необходимости реализации в игре, может существовать большее количество систем.

Физический движок — система, моделирующая физические законы внутри игры. Физические законы могут быть как приближены к реальным, там и изменены в случаи необходимости. Может быть частью игрового движка, или самостоятельной программой. Работает в режиме реального времени, дабы обеспечивать своевременный отклик.

IDE (интегрированная среда разработки) — программное приложение обеспечивающее комплексные возможности для программиста для разработки программ. Обычно состоит из текстового редактора кода, средств автоматизации сборки и дебаггера. Большинство современных IDE имеют интеллектуальные системы автодополнения кода.

Спрайт — растровое изображение, отображаемое на экране в контексте игры.

Анимация — последовательное отображение графический объектов.

Сеттинг — среда, окружение в котором происходит действие произведения. Включает в себя место, время, условия и антураж.

Ассет — игровой ресурс, используемый для создания игры.

Параллакс — изменение видимого положение объекта по отношению к удаленному фону в зависимости от нахождения наблюдателя.

Scroller — тип видеоигры, в которой задний фон прокручивается, чаще всего, сверху-вниз, либо слева-напрво, для создания иллюзии передвижения персонажа по игровому миру.

Top-down — тип видеоигры, в которой камера располагается строго над игровым полем.

Shoot ‘em up — жанр видеоигр, в которых игрок, управляющий каким-либо персонажем или техническим средством, сражается с большим количеством врагов при помощи стрельбы. Процесс игры часто изображён в очень стилизованной манере. Жанр зародился на аркадных игровых автоматах с игры Space Invaders. Пик популярности пришёлся на конец 80-х – начало 90-х, в основном в виде аркадных и консольных проектов.

Bullet hell — жанр для которого характерно большое количество вражеских снарядов на экране, направленных на уничтожение игрока. Чаще всего игры этого жанра являются сайд-скроллерами. Может существовать без основной механики Shoot ‘em up’a – уничтожения большого количества противников, и вообще не являться top-down shooter’ом.

Danmaku — жанр, взявший для своего название японский термин для заградительного огня. Danmaku существует исключительно в концепции vertical scroller'a, с расположением персонажа на нижней части экрана, и наличием многочисленных боссов по ходу геймплея. Другой отличительной чертой является огромное количество вражеских снарядов, покрывающих почти всё игровое поле, и чаще представляя из себя паттерны. Так же для этого жанра характерен маленький хитбокс персонажа и некоторых снарядов, обусловленный необходимостью дать игроку возможность дать игроку маневрировать в плотном облаке вражеских проджектайлов.

1.1.2 Термины наиболее характерные для проекта

Проджектайл (Projectiele) — часто применяемый термин в видеоиграх, обозначающий какой-либо снаряд, выпущенный игроком, или в игрока, с целью нанесения урона.

Паттерн (Pattern) — шаблон или узор атаки, применяемой противником, или игроком.

Бомба (Bomb) — ограниченная количеством использований возможность игрока расчистить игровое поле от завесы проджектайлов.

Death Bombing — термин применяемый в случаи использования игроком бомбы во время попадания вражеского снаряда, или в определённое время после этого, после которого у игрока не отнимается жизнь, а используется бомба. Требует от игрока очень хорошего тайминга.

Спелл — атака босса, определяющая паттерн, поведение и здоровья босса в момент использования.

HP (хп, hit points, health points) — количество очков жизни, определяющие как много урона может получить объект, прежде чем будет уничтожен. Может быть характерен как для игрока или противников, так и для отдельных объектов.

Скорость перемещения (movement speed, ms) — скорость перемещения игрока, противника или объекта по игровому полю.

Тайминг (timing) — термин, использующийся во многих видеоиграх, однако несколько различающийся от игры к игре. Основное понятие заключается в строгом соблюдении действий в зависимости от времени, с точностью до десятых долей секунды.

Грейз (graze) — понятие, обозначающее максимально близкий проход игрока к проджектайлом, оставаясь при этом не задетым. Существует для мотивации более агрессивной игры. Награждается увеличенным количеством очков.

Хитбокс (hitbox) — зона, являющаяся участком спрайта, в которую при попадании проджектайла игра засчитывает столкновение.

Фокус (Focus) — режим управления персонажа, в котором персонаж имеет сниженную скорость передвижения и подсветку хитбокса для возможности более точного управления, при маневрировании между проджектайлами, а также, опционально, другой тип атаки, для фокусировки огня на определённом объекте.

1.2 Анализ существующих разработок

Самым ярким и долгоживущим представителем Danmaku является серия игр Touhou Project, насчитывающая на сегодняшний день свыше 30 игр жанра (рисунок 1). Каждую игру можно считать каноничным представителем жанра, следующую всем его принципам. Игра состоит из нескольких стейджей(уровней), на протяжении которых, игрок уничтожает многочисленного противника и вынужден уклонятся от огромного количества снарядов, разбросанных по экрану. В середине и в конце уровня игрок сражается с мидбоссом и боссом, соответственно. Отличительной особенностью игры являются стилистика, характерная для аниме, огромное количество разнообразных персонажей, и разнообразными паттернами атак. Для игры, как и для жанра в целом, характерна необычайная сложность и геймплей, рассчитанный на множество перепрохождений.

Рисунок 1 — Скриншот игры «Touhou Project 7 Perfect Cherry Blossom»

Игра обрела необычайную популярность в Японии, и имеет некоторую известность за её пределами. Причиной популярности является огромное количество персонажей, незамысловатый сюжет, сеттинг построенный на японской мифологии и очень своеобразный геймплей, характерный для жанра.

Имеется некоторое количество новых инди-проектов в данном жанре (рисунок 2). Основное отличие большинства от Touhou Project, является некоторым отступлением от жанра. Часть проектов использует принцип left-right скроллеров, когда задний фон перемещается не сверху вниз, а слева направо. Таким образом игра имеет вид не сверху, а сбоку, что несколько видеизменяет паттерны, т.к. ориентироваться в покрытом проджектайлами экране, при горизонтальном положении игровых объектов, несколько сложнее. Другая часть игр оперирует не «пулевой завесой», а меньшим количеством проджектайлов, или других объектов, представляющих опасность игроку, но с более замысловатой механикой. Увеличение размеров проджектайлов, ограничение игрового пространства, или другие способы заставить игрока уклонятся не от большого количества опасных объектов, а маневрировать в усложнённых условиях геймплея.

Рисунок 2 — Скриншот игры «Steel Rain»

Однако, существуют и успешные представители жанра, среди инди-проектов (рисунок 3). Такие проекты заслуживают высокие оценки своей целевой аудитории, и имеют хорошие продажи в сервисах цифровой дистрибуции.

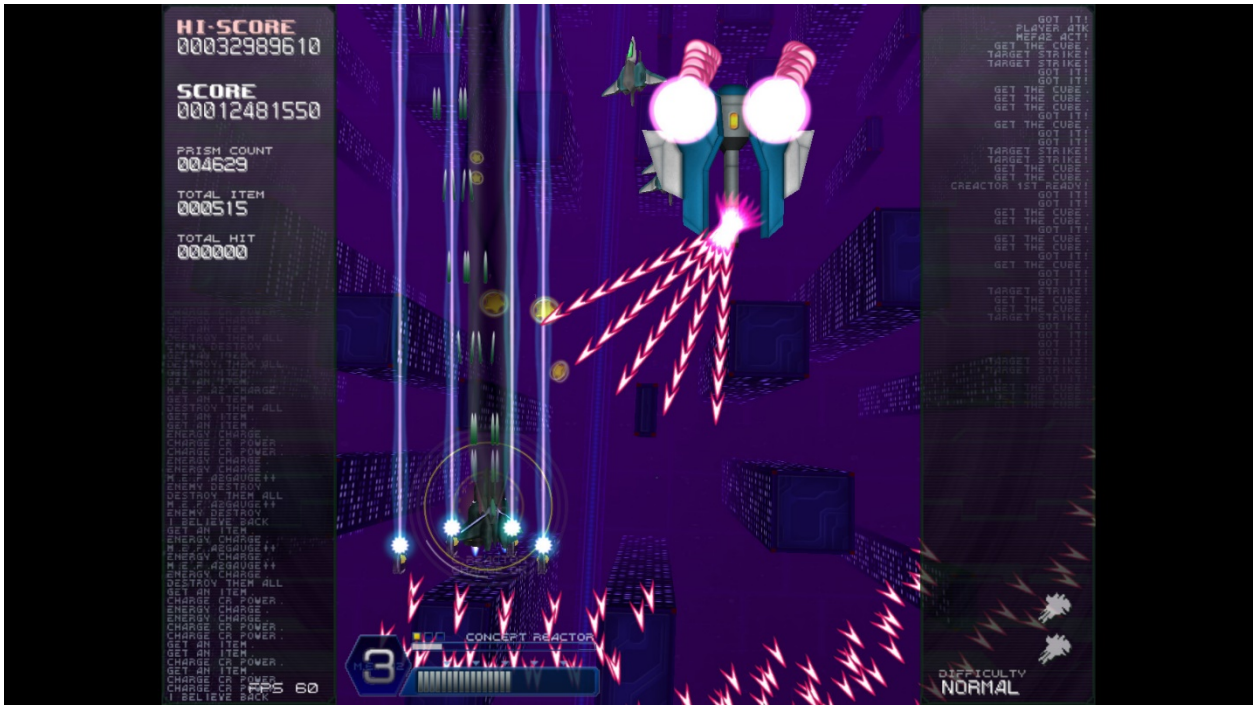


Рисунок 3 — Игра «RefRain - prism memories -»

1.3 Анализ средств разработки и обоснование выбора технологий

На сегодняшний день существуют множество игровых движков для разработки собственных проектов. Некоторые из них предоставляются бесплатно, при определённых условиях, другие же требуют покупки или оплаты подписки. Выбор движка должен зависеть от реализуемого проекта. Для сравнения было выбрано три игровых движка.

Game Maker — простой игровой движок, позволяющий создать игры для большого числа платформ (рисунок 4). Движок рассчитан на разработку 2D игр, однако имеет некоторые ограниченные возможности работы и с 3D. изначально разрабатывался голландским ученым Марком Овермассом для обучения детей начальной школы навыкам программирования, позже был продан компании YoYo Games. Движок в последствии был переписан, одна-

ко, были сохранены концепция и интерфейс, что позволило оставить низкий порог вхождения. Так же обладает большим и живым сообществом, собранным за долгие годы существования движка. Поддерживает скриптинг на C подобном языке GML – GameMaker Language. На основе GameMaker'a было создано множество успешных проектов, таких как: «Hotline Miami», «Undertale», «Gods Will Be Watching», «Fran Bow», «Hyper Light Drifter», «VA-11 HALL-A» и другие.

Движок предоставляется на платной основе, различаясь ценой, в зависимости от выбранной платформы. Есть возможность использовать триальную версию для ознакомления.

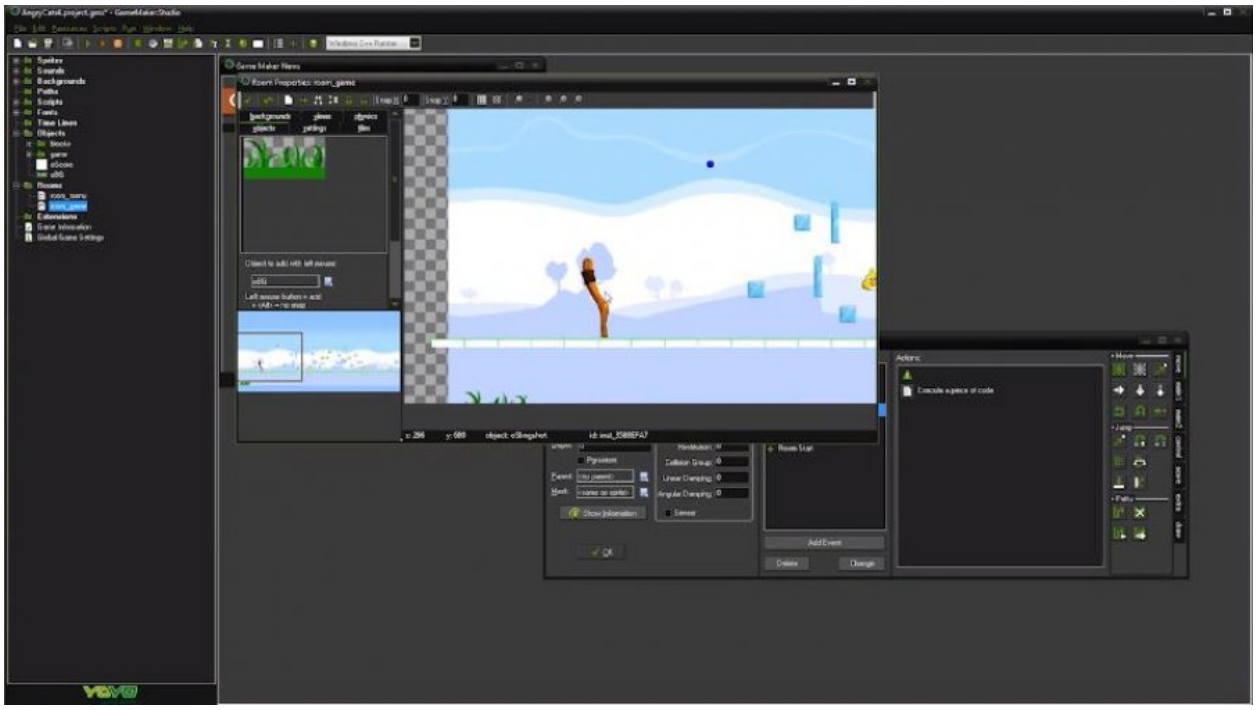


Рисунок 4 — Интерфейс Game Maker Studio 2

Unity — полноценный игровой движок для кроссплатформенной разработки 2D и 3D приложений и игр (рисунок 5). С 2015 года весь функционал стал абсолютно бесплатным, при условии дохода с игры менее 100000$ в год. Имеет собственную IDE, и поддерживает 2 языка программирования для создания скриптов – C# и JavaScript. Поддерживает импорт большого числа форматов, касательно как звуковых файлов, так и 3D-моделей. Имеет обширное сообщество, и некоторое количество литературы, посвящённой разработ-

ке игр на данном движке. Так же, имеет собственный магазин с ассетами. Относительная простота разработки, вкупе с достаточно обширными возможностями по работе с визуальной составляющей. В качестве физического движка используется nVidia PhysX. Движок широко популярен у инди разработчиков, однако, находит популярность и у именитых компаний. Имеет большой список популярных проектов, реализованных на данном движке с абсолютно разными жанрами, от 2D shoot'em up'а «Enter The Gungeon», до градостроительного симулятора «Cities Skylines» и cRPG «Pillars of Eternity».

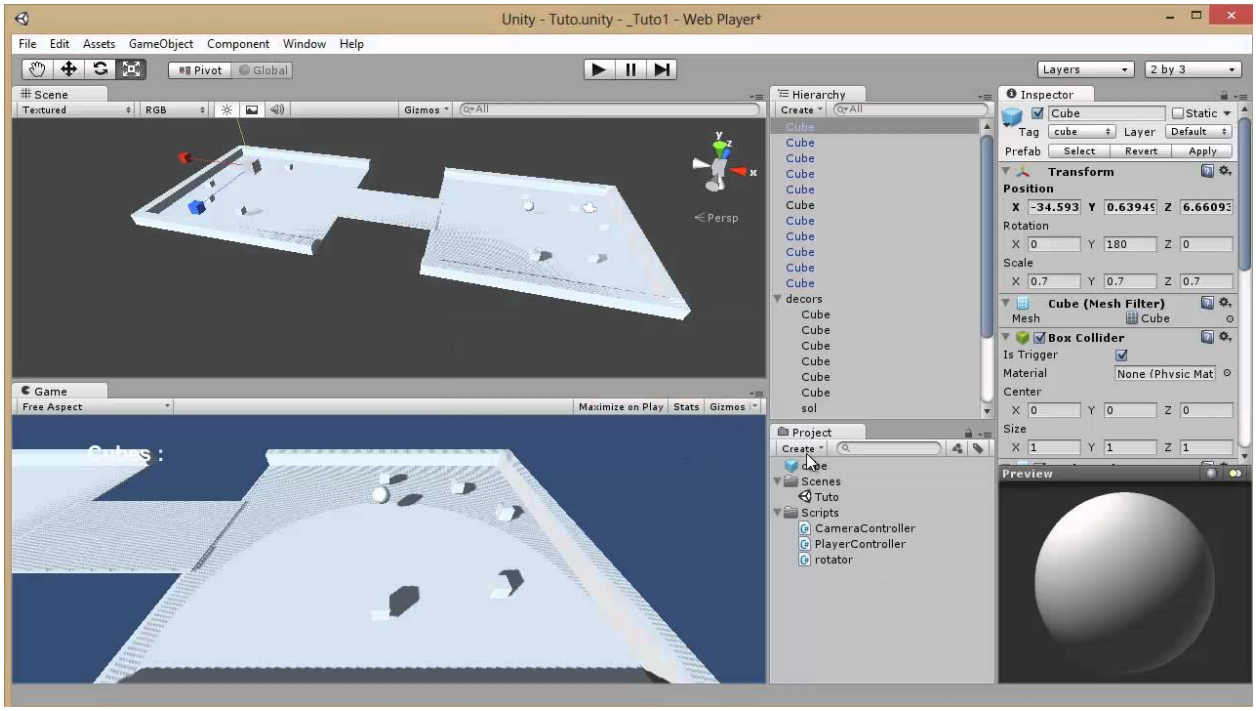


Рисунок 5 — Интерфейс Unity

Unreal Engine — игровой движок, созданный компанией Epic Games (рисунок 6). Написан на языке C++, поддерживает большинство платформ. С 2015 года является бесплатным, при условии, что прибыль от реализации приложений на движке не превышает 3000$ за квартал, в ином случаи, разработчикам движка отчисляется 5% от прибыли. Имеет систему Blueprints Visual Scripting, которая позволяет «собирать» игру из готовых частей, написанных самими разработчиками или сообществом, что сильно упрощает разработку и экономит время, вкупе с имеющимся магазином ассетов. При необходимости поддерживает написание собственного кода на C++, так же

имеет модуль, для работы с C#. Движок очень ресурсоёмкий, но и очень выдающийся в плане функционала. Часто используется для демонстрации новых разработок сторонними компаниями, например, nVidia.

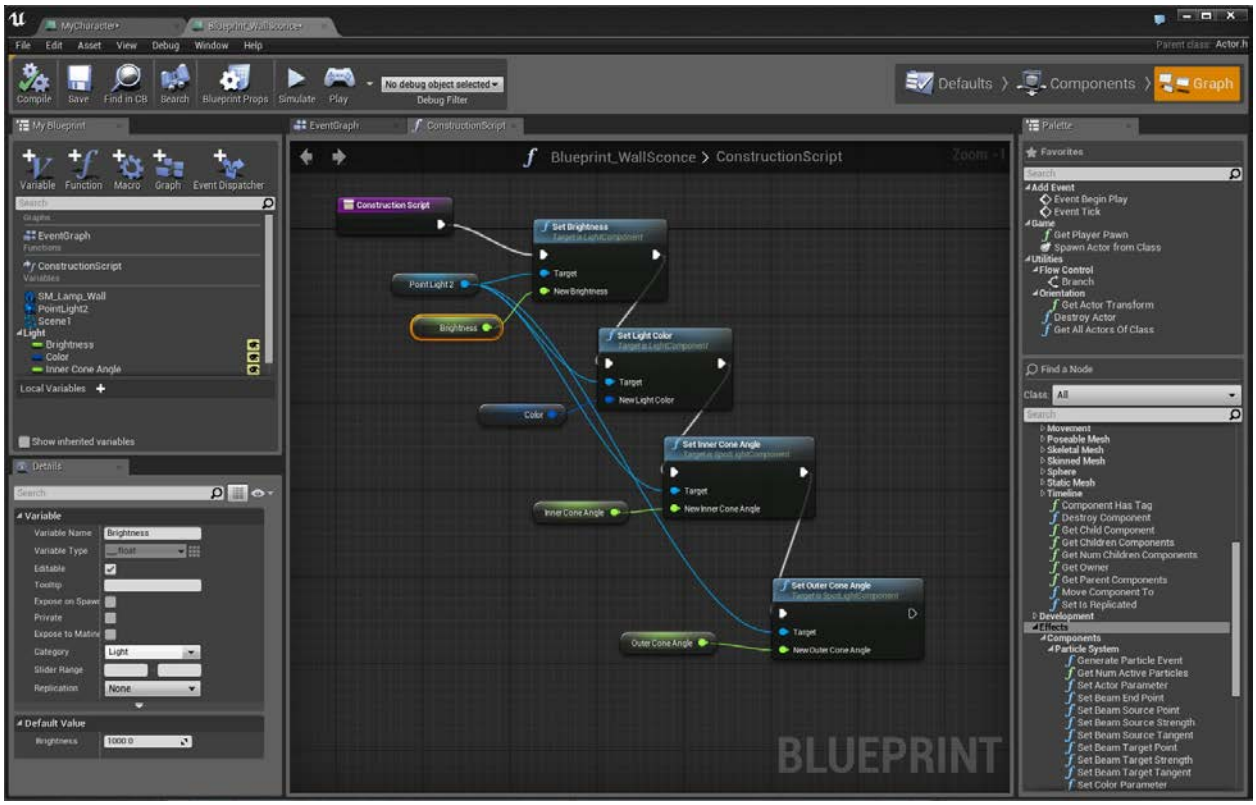


Рисунок 6 — Интерфейс Unreal Engine 4

На основе возможностей и доступности движков, выбор пал в сторону Unity, по причине бесплатности, в сравнении с Game Maker'ом. При сравнении с Unreal Engine 4, последний обеспечивает избыточный функционал, и требует на порядок больше ресурсов, что в контексте создания игры в жанре «Danmaku» будет большим минусом, т.к. большинство функций движка будет не задействовано, а значит, нет необходимости в его использовании.

Краткое сравнение игровых движков представлено в таблице 1.

Таблица 1 — Сравнение игровых движков

	GameMaker Studio 2	Unity	Unreal Engine 4
Поддержка 3D проектов	Есть.	Есть.	Есть.

Окончание таблицы 1

Поддержка VR	Нет.	Есть.	Есть.
Цена	Разовая покупка, от $99 до $399, в зависимости от платформы.	Имеется бесплатная версия, при условии дохода от игры не выше $100000 в месяц. Подписка от $35, в зависимости от размера команды и необходимых инструментов.	Бесплатен. При доходах выше $3000 за квартал 5% от прибыли перечисляется Разработчикам движка
Поддерживаемые платформы	Android, iOS, Windows Phone, HTML 5, Amazon, Tizen, XBox 360/XBox One, PS3/PS4/PSVita, SteamOS, Linux, Mac OS X, PC.	iOS, Android, Windows Phone, Tizen, Windows, Mac OS X, Linus, SteamOS, WebGL, PS3,PS4/PSVita, Xbox 360/Xbox One, Wii U, Nintendo 3DS, Android TV, Samsung Smart TV, tvOS, Nintendo Switch, Fire OS, Facebook Gameroom.	PS4, XBox One, Nintendo Switch, iOS, Android, PC, Mac OS X, Linux, SteamOS, HTML5.
Языки программирования	GML (GameMaker Studio 2 Language).	C#, JavaScript.	C++, система Blueprints.
Магазин	Есть. GameMaker: Marketplace.	Есть. Unity Asset Store.	Есть. UE4 Marketplace.

1.4 Общий алгоритм реализации проекта

1.4.1 Особенности физического движка Unity

В процессе разработки следует учитывать тонкости игрового движка. Выбранный жанр предполагает наличие на сцене большого количество проджектайлов, каждый из которых должен иметь возможность сталкиваться с игроком, а значит быть физическим объектом. Необходимо исследовать особенности расчёта физики в Unity, чтобы избежать возможных проблем в разработке.

В Unity можно выделить три «закона» для двухмерной физики. Нарушение этих «законов» приводит к падению производительности игры, так как физический движок будет перерасчитывать физику рядом с каждым объектом, нарушившим правила. Эти правила характерны для любого физического объекта, использующегося в игре, а это значит, что в условиях создания игры в вышеперечисленных жанрах, где характерно одновременное нахождение множества разнообразных физических объектов, необходимо строго соблюдать каждое правило. Нарушение не является ошибкой, а скорее особым случаем для физического движка, поэтому данные правила достаточно условны и являются особенностями работы физики в Unity [2].

1. Коллайдеры не должны двигаться, вращаться, включаться\выключаться и менять размер. Как только коллайдер добавлен на объект – нельзя оказывать какое-либо воздействие на него или объекты, в которых он содержится. Обычный коллайдер — исключительно статический объект. Дерево, например, может быть с одним коллайдером. Если дерево может упасть на игрока — дерево будет падать вместе с производительностью. Если это дерево растет из волшебного питательного облака, которое коллайдера не имеет, но может перемещаться — это будет сопровождаться падением производительности [2].

2. Если объект движется или вращается — он должен быть твердым телом т.е. иметь компонент Rigidbody. Rigidbody меняют отношение физического движка к объекту. На него начинают воздействовать внешние силы, он может иметь линейную и угловую скорости, а самое главное — твердое тело может двигаться и вращаться средствами физического движка, не вызывая полный пересчёт физики. Существует два типа твердых тел — обычные и кинематические. Обычные тела взаимодействуют друг с другом и обычными коллайдерами — одно тело не может пройти сквозь другое. Кинематические тела следуют упрощенным правилам симуляции — на них не воздействуют никакие внешние силы, гравитация — в том числе. Они свободно могут проходить друг через друга и коллайдеры, а вот обычные твердые тела они отталкивают, как будто имея бесконечную массу [2].

3. Если объект является твердым телом — двигаться и вращаться он должен через методы твердого тела. Нельзя использовать прямое обращение через transform к объекту, как только в него добавлен коллайдер. Это так же касается и иерархических связей. Если сдвинуть объект, содержащий дочерние твёрдые тела – произойдёт пересчёт физики [2].

Существует три уровня управления Rigidbody:

1. Самый высокий и, следовательно, естественный, уровень — через силы. Это методы AddForce и AddTorque. Физический движок учтет массу тела и правильно посчитает результирующую скорость. Все взаимодействия тел происходят на этом уровне [9].

2. Средний уровень — изменение скоростей. Это свойства velocity и angularVelocity. На их основе вычисляются силы, влияющие на тела при их взаимодействии, а также, очевидно, их положения в следующий момент времени. Если у твердого тела очень маленькая скорость — оно «засыпает», для экономии ресурсов [9].

3. Самый низкий уровень — непосредственно координаты объекта и его ориентация в пространстве. Это методы MovePosition и MoveRotation. На следующей итерации вычисления физики (это важно, поскольку каждый по-

следующий вызов метода в рамках одного кадра заменяет вызов предыдущего) они выполняют телепортацию объекта в новое положение, после которой он живет как раньше [9].

На включение, выключение и изменение масштабов объектов нельзя взаимодействовать через Rigidbody, а значит их изменение будет вызывать пересчёт физики и падение производительности. Исключением является выключение объекта, которое проходит абсолютно безболезненно.

1.4.2 Написание логики перемещения объектов в пространстве

Для упрощения дальнейшей разработки, игрок и все противники должны передвигаться по одним и тем же принципам, с использованием похожих скриптов. Игрок и противники должны быть физическими объектами, чтобы обрабатывать столкновения с проджектайлами и друг другом. Исходя из пункта 1.4.1, эти объекты должны иметь физическое тело, и передвигаться с помощью него. Для простоты взаимодействия и обращения к объектам, создаётся скрипт абстрактного класса, для взаимодействия с физическим телом. Игрок и противники, получают дочерние классы, учитывающие тонкости управления для игрока, и особенности поведения у противников.

Для большинства проджектайлов требуются отдельные скрипты, описывающие их передвижение, в зависимости от выбранного поведения, для каждого конкретного типа проджектайла. Это не позволит использовать абстрактный класс, как в случаи с игроком и противниками, однако, всё также требуется унификация взаимодействия и обращения. Для обеспечения простоты обращения к проджектайлам создаётся интерфейс, обеспечивающий возможность получать и изменять необходимые параметры, присутствующие у каждого проджектайла, и при этом сохранять уникальность их поведения, за счёт скриптов, наследующих этот интерфейс.

1.4.3 Написание генераторов паттернов

Одна из основных отличительных особенностей жанра «Danmaku» – паттерны атак, пускаемые противниками и боссами. Множество разнообразных атак можно создать одним хорошо написанным скриптом, но одним скриптом не всегда можно покрыть все желаемые варианты. Для игры потребуется два генератора, с большим количеством переменных (рисунок 7). В скриптах так же необходимо использовать тригонометрическую составляющую, для усложнения паттерна атаки. В качестве тригонометрической функции будет использоваться функция тангенса от времени, в зависимости от угла изменения вращения.

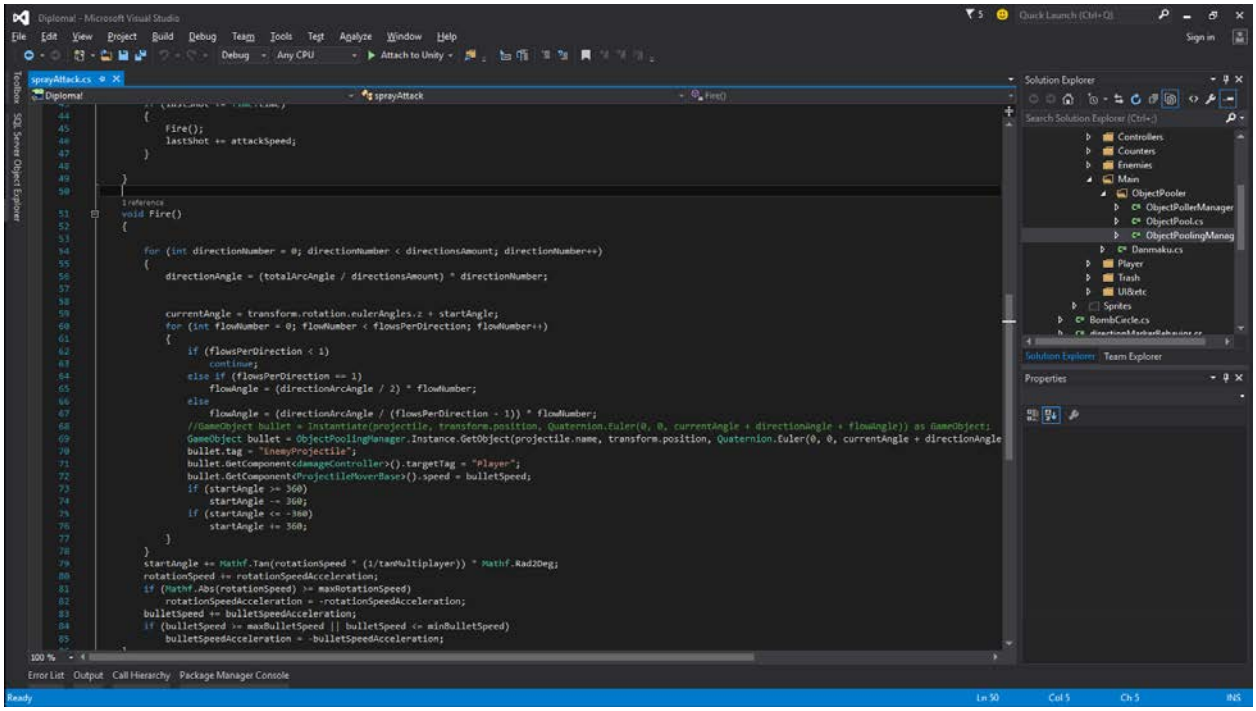


Рисунок 7 — Скрипт генератора паттернов

Первый генератор может изменять количество потоков снарядов, изменять угол между ними, вращать эти потоки, изменять скорость потока со временем и пускать их в цикл. Менять количество выпускаемых снарядов в секунду, их скорость, а также скорость последующих выпускаемых снарядов. Чем выше количество входных данных, используемых как переменные в

формуле генерации паттерна, тем выше вариативность возможных их вариаций. Результат работы первого генератора представлен на рисунках 8-10.

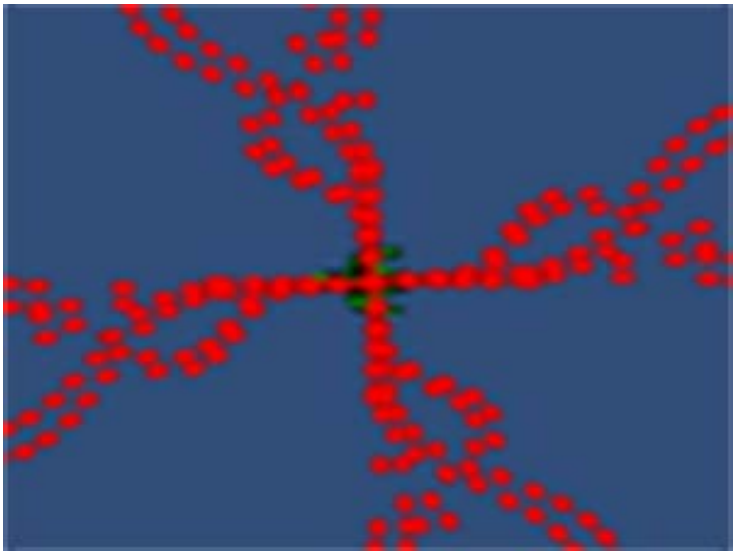


Рисунок 8 — Пример работы генератора паттернов без вращения

Генератор способен создавать как относительно статичные паттерны занимающие лишь определённый процент экрана, так и вращающиеся узоры, покрывающие непостоянную площадь. Для создания вращения используется тригонометрическая формула тангенса, с переменной-модификатором, позволяющей охватывать все возможные вариации применения данной функции.

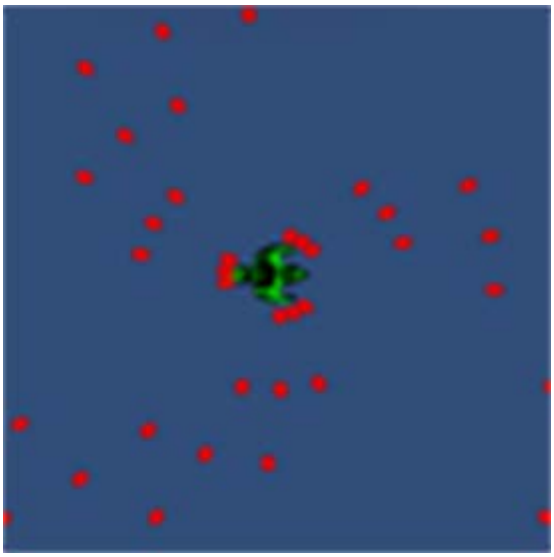


Рисунок 9 — Пример работы генератора паттернов с вращением

При создании паттерна важно учитывать его сложность и саму возможность игрока выжить в данном количестве проджектайлов. Основными при этом параметрами будут являться количество выпускаемых проджектайлов и их скорость.

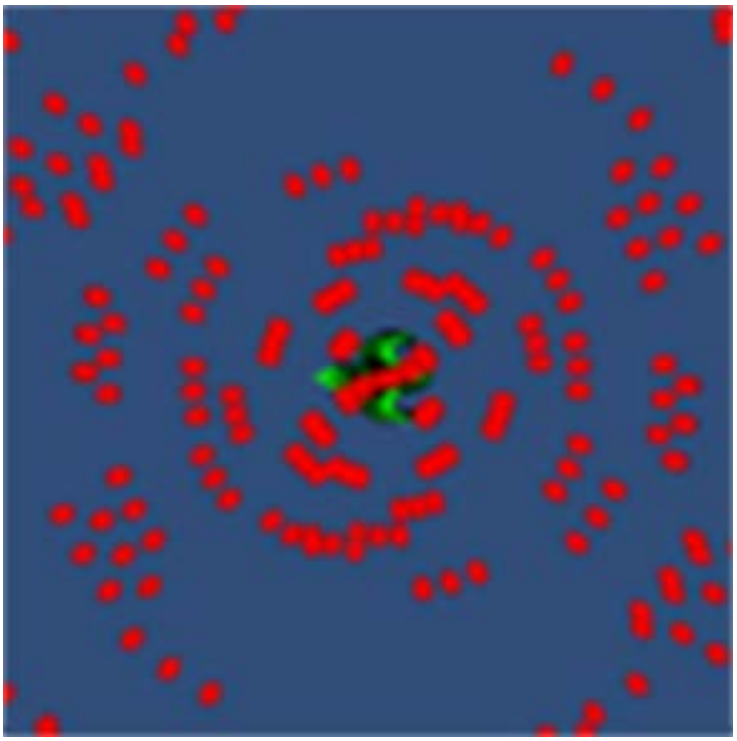


Рисунок 10 — Пример работы генератора паттернов с динамическими переменными

Второй генератор имеет более простое назначение, отчасти реализуемое первым. Отличие второго генератора, возможность проставлять задержку между волнами выпускаемых проджектайлов, большая вариативность со сторонами атаки и возможность в нужный момент уничтожить объект родитель.

1.4.4 Реализация визуальной части игры

После создания всех аспектов игры, касающихся геймплея и механик, можно легко подставить все имеющиеся графические материалы, на уже созданные объекты внутри игры. Это позволяет структурировать подход к созданию программного продукта и упростить разграничение этапов реализации проекта. После того как все необходимые элементы определены, и со-

зданы все используемые объекты, достаточно просто определиться, какие спрайты необходимы для дальнейшей реализации. Также создание интерфейса упрощается, при условии, что известны все элементы, входящие в его состав, а у всех счётчиков прописана логика. Основной проблемой визуальной составляющей являются проджектайлы, т.к. изначально неизвестно количество их возможных вариаций. Большинство проджектайлов должно иметь уникальный вид, различаясь формой, размерами или цветами, чтобы игроку было проще ориентироваться в ситуациях, когда на игровом поле присутствует огромное количество проджекталов. При известном количестве вариаций проджектайлов, легче реализовать все необходимые спрайты.

2 ПРОЕКТНАЯ ЧАСТЬ

2.1 Характеристика потенциальной аудитории проекта. Ориентированность работы

Целевой аудиторией проекта являются люди, играющие в компьютерные игры, но не желающие уделять им слишком много времени. К данной категории относятся студенты и работники в возрасте от 18 до 35 лет. За счёт отсутствия мультиплеера и чёткой соревновательной составляющей, лица младше 18 лет вряд ли найдут для себя ценность данного проекта. На это так же повлияет и особенность жанра, генерирующий сложность в нестандартных аспектах игры. Это же послужит причиной для сужения верхней планки возраста целевой аудитории до возраста 35 лет. Люди более старшего возраста, чаще всего, не заинтересованны в преодолении такого рода задач, представленных проектом. Для людей, подходящих под предполагаемую целевую аудиторию, проект может занять нишу игры-таймкиллера – игр, представляющих возможность убить время для игрока, в перерыве между делами. Так же, в соответствии с жанром, игра будет сложно проходима, особенно с первого раза, однако, не будет вызывать проблем с восприятием игрового процесса, за счёт простых к пониманию механик. Таким образом, человеку будет легко понимать ошибки, допущенные в прохождении, и это не оттолкнёт его от дальнейших попыток, а каждый пройденный момент будет восприниматься как небольшая победа над игрой, что должно сохранить интерес в дальнейшем прохождении. Игра требует концентрации на игровом процессе и собственных действиях, а также требует принимать решения, способные упростить, либо усложнить геймплей. Такая необходимость сможет отвлечь игрока, и, в некотором роде, расслабить, после долгих и монотонных дел. Таким образом, игра охватывает аудиторию людей, которым игра важна как

процесс, необременяющий элементами вне геймплея, что хорошо сочетается с концепцией таймкиллера.

2.2 Постановка задачи проекта

2.2.1 Актуальность проекта

Игр жанров «shoot em’ up» и «danmaku» не так много, однако, они имеют своих последователей, что гарантирует спрос выше предложения. Всё чаще игры, с незамысловатой механикой переходят в мобильный сегмент, где находят популярность, за счёт быстрой доступности и простоты управления. Даже «danmaku» игры, требующие чёткости движений смогли перейти на мобильный рынок и обрести там некоторую популярность. За счёт этого количество выходящих игр на ПК в определённых жанрах, включая «shoot em’ up» и «danmaku» не столь велико, а на мобильных устройствах не всегда возможно реализовать все задумки, касательно разрабатываемого проекта. С другой стороны, проекты, разрабатываемые на Unity, легко поддаются переносу на другие платформы, позволяя не переписывать логику приложения, а лишь изменив схему управления, если это позволяет созданная игровая механика. Так с ПК на мобильные платформы была перенесена популярная карточная игра Hearthstone, компании Blizzard, и наоборот, с мобильных платформ, на ПК была перенесена игра Fallout Shelter от Bethesda. Обе игры не поменялись геймплейно, изменилась лишь схема управления, для более удобного взаимодействия на новой платформе. Игры в целом на сегодняшний день имеют огромную популярность, которая продолжает расти, следовательно, расширяется и аудитория. Закрепившись в индустрии развлечений, наравне с кино, игры продолжат развитие, и будут оставаться актуальными ещё очень долгое время. Игры в жанре «Shooter» в любом их исполнении, всегда были отдельной, популярнейшей частью всей игровой индустрии.

Этот жанр зарождал индустрию, и задавал ей вектор эволюции, поэтому «стрелялки» актуальны всегда.

2.2.2 Цель и назначение проекта

Цель проекта – Анализ и изучение средств разработки компьютерных игры, для последующего создания компьютерной инди-игры на стыке двух жанров. Разработка уникального программного продукта, на основе слияния геймплея разных жанров, и использовании характерных особенностей от каждого. Разработка индивидуальной графической составляющей. Развитие у игроков концентрации, скорости реакции и умение быстро принимать решения. Получение опыта разработки полноценного, рабочего проекта – компьютерной игры с нуля, оптимизации кода компьютерной игры, при определённых задачах. Изучение паттернов реализации механики компьютерных игр.

Разработка компьютерных игр – долгий и трудоёмкий процесс, требующий знаний во многих направлениях компьютерных технологий. При разработке игры нужно учитывать несколько составляющих.

Графическую – разработать уникальный стиль игры, в случаи 2D игры, нарисовать спрайты, в случаи 3D, создать 3D-модели, для каждого случая, необходимо создать эффекты, соответствующие стилю игры. Разработать эргономичный, не перегруженный интерфейс, отображающий всю необходимую информацию, и не мешающий игровому процессу [4].

Аудио составляющая игры, может быть не столь ярко выраженной, однако, хорошее звуковое и музыкальное сопровождение многократно усиливают эффект, производимый игрой. Часто это способствует увеличению популярности игры

Некоторые игровые движки, например, Unreal Engine 4, позволяют создавать игры не программируя. Этот подход позволяет экономить время, на создании базовых механик, однако, для создания чего-то уникального, при-

дётся научиться программировать. Поэтому для создания уникальной игры, нужно хорошо знать язык, поддерживаемый движком.

Часто, игры обладают либо мультиплеером, либо кооперативным прохождением, а значит при их разработке нужно работать с сетевыми технологиями, что также повышает сложность кода и разработки игры.

Игры являются технологичным продуктом, требующим большого количества знаний, времени и ресурсов. Разработка инди игры позволяет получить ценный опыт, во многих направлениях, который поможет в дальнейшей деятельности, не обязательно связанной с компьютерными играми. Опыт полученный при работе с игровым движком, позволит ускорить процесс дальнейших разработок на этом движке.

2.2.3 Функционал и интерфейс проекта

Функционал проекта должен отвечать выбранным жанрам, что несколько облегчает его разработку. «Top-down» предполагает положение камеры исключительно над игроком и игровым полем. Для такого подхода существует несколько устоявшихся схем управления, отвечающие удобству управления, при решении определённой поставленной задачи. Т.к. игроку чаще всего необходимо стрелять и двигаться в разные стороны, было принято решение назначить для каждого действия свой контроллер. Перемещение осуществляется независимо от направления взгляда персонажа, и привязано к распространённой комбинации клавиатурных клавиш для управления – «WASD». Данная схема используется в подавляющем большинстве игр, позволяющих напрямую контролировать игрового персонажа, за счёт чего является наиболее известной и понятной. Направление взгляда игрока привязано к мыши, что позволяет игроку точнее позиционировать свои атаки, что очень важно в жанре «danmaku», в контексте которого противники, так же, как и герой, имеют хитбокс по площади меньший, чем непосредственно спрайт, к

которому он привязан. Эта особенность заставляет игрока проявлять более высокую точность, которой нельзя достичь использованием клавиатуры.

«Shoot em’ up» подразумевает уничтожение большого количества противников, а, следовательно, у игрока должна быть возможность их уничтожать. Для этого в игру должно быть добавлено какое-либо вооружение. В этом аспекте игры так же следует считаться и с жанром «danmaku», в котором для уклонения от огромного количества проджектайлов игроку нужна возможность более точной корректировки движения и поражения врагов с таким же маленьким хитбоксом. Для этого была создана возможность переходить в фокус мод, позволяющая переключаться между атаками и скоростью движения. Основной, не фокусный, тип атаки, должен покрывать максимальную площадь игрового поля, для зачистки слабых противников и быстрого перемещения по полю. Фокус мод, снижает скорость перемещения персонажа и подсвечивает его хитбокс, это позволяет использовать его в случаи нахождения в скоплении проджектайлов, для более точного контроля за персонажем и выводе его из опасных зон. Также меняется тип атаки, на строго прямую атаку, имеющую минимальную площадь, но максимальный урон, что позволяет игроку фокусироваться на более опасных противниках, вкупе позволяя ему при этом оставаться в живых, при нахождении в паттерне атаки этого противника.

«Danmaku» также предполагает наличие у главного героя бомб, для использования в случаи критической ситуации, когда игрок не может выйти самостоятельно из завесы проджектайлов. Бомба должна расчищать всё игровое поле от вражеских проджектайлов и давать игроку время для перемещения по полю в более безопасные и приемлемые для него условия. Существование бомбы обосновано тем, что иногда паттерны могут складываться в безвыходную ситуацию, в которой максимальное расстояние между двумя ближайшими проджектайлами будет меньше, чем размер хитбокса героя, что гарантирует потерю жизни. Такой исход будет нечестным по отношению к игроку, ведь потеря жизни должна быть следствием действий игрока, а не

случайного события. Игрок может использовать бомбы в любой момент игры, если посчитает, что не справится в данной ситуации.

По ходу геймплея игрок должен получать какие-либо бонусы, облегчающие его игру. В «danmaku» играх, как и во многих скроллерах, персонаж получает усиления атаки и дополнительные жизни по мере прохождения игры. Такой подход не требует от игрока принятия решений, а лишь увеличивает его возможности, равно пропорционально усложнения геймплея. Для большего разнообразия, в игру была введена система опыта. За уничтожение противника, игрок получает некоторое количество опыта, который, при достижении некоторого количества, давал игроку валюту, за которую игрок может улучшить свои характеристики, атаку или приобрести дополнительные жизни или бомбы. Таким образом, игроку приходится решать, что ему важнее в данный момент игры, ведь купив бомбу, в как ему кажется, безвыходной ситуации, он лишает себя возможности увеличить силу атаки в ближайшем будущем. Такая система будет вынуждать игрока принимать решения, на его взгляд, наиболее оптимальные в данный момент игры. Количество опыта, необходимого для получения валюты растёт каждый раз, при достижении данной отметки, что усложняет получение валюты со временем, и делает каждую последующую покупку всё более дорогой.

Интерфейс игры должен занимать минимальное положение на экране, и не занимать какое-либо пространство рядом с игроком. Первое требование обусловлено необходимостью игрока следить за положением противника на поле, для корректировки собственной атаки. В данной ситуации, интерфейс не должен мешать игроку. Второе требование должно соблюдаться, чтобы не мешать игроку уклоняться от вражеских атак. В играх с большим количеством проджектайлов, игроку приходится фокусироваться не на всём игровом поле, наблюдая направления вражеских атак, а на собственном хитбоксе, сопоставляя его размеры и своё движение, с движением близлежащих проджектайлов. Самая опасная и требующая внимания зона чаще всего имеет площадь немного большую, чем непосредственно спрайт игрока. Эта зона

должна быть чиста от любых индикаторов, чаще всего, затемняется все объекты, включая спрайт самого игрока. Подсвеченными остаются лишь хитбокс и проджектайлы, для обеспечения игроку максимальной визуальной точности происходящего и возможности корректировки своих действий.

Для интерфейса были выбраны «полые» иконки обозначения количество жизней и бомб, а также тонкий, вытянутый шрифт. Совокупность выбранных параметров, позволит выводить на экран максимум визуальной информации, занимая при этом минимальную площадь на экране.

Основную сложность в расчёте системных требований составляет наличие на игровой сцене тысяч физических объектов, и особенности работы движка с ними. Необходимо провести оптимизацию кода игры, для максимального снижения требований к процессору. Замена «дорогостоящих» по производительности операции, на более быстрые и простые, а также использование паттернов программирования, позволяющих экономить ресурсы вычислительной платформы. Данные решения особенно важны, при реализации проекта на мобильные платформы, т.к. они очень жёстко ограничены в вычислительных мощностях.

2.2.4 Входные данные к проекту

Идея реализовать проект зародилась на фоне малого количества игр, в жанре «danmaku», несмотря на схожесть с относительно популярным «shoot ‘em up». Для игры был придуман сеттинг и сюжет. Действие игры должно разворачиваться, как бы в организме человека, где главный герой будет неким лекарством, а его противники – разного рода болезни. Внешне игра не должна быть мрачной и отторгающей, а должна создавать некую комичность и простоту на уровне мультфильмов. Название игры должно отражать суть происходящего, поэтому лучшим вариантом являлось слово «panacea», с английского – панацея.

Панацея – мифологическое универсальное средство от всех болезней.

Определение слово, характеризует главного героя игры, как некоторого персонажа, стремящегося и уничтожающего болезни, которые встречаются ему на пути.

Существует сложность с визуализацией столь неоднозначного персонажа. Для упрощения создания спрайта, было принято решение максимально абстрагироваться от какого-либо конкретного образа. Персонаж не должен быть одушевлён и лишён какой-либо эмоциональной окраски. Таким образом был создан спрайт, схожий с космическим кораблём, являющийся оптимальным вариантом для персонажа, способного уничтожать противников (рисунок 11).

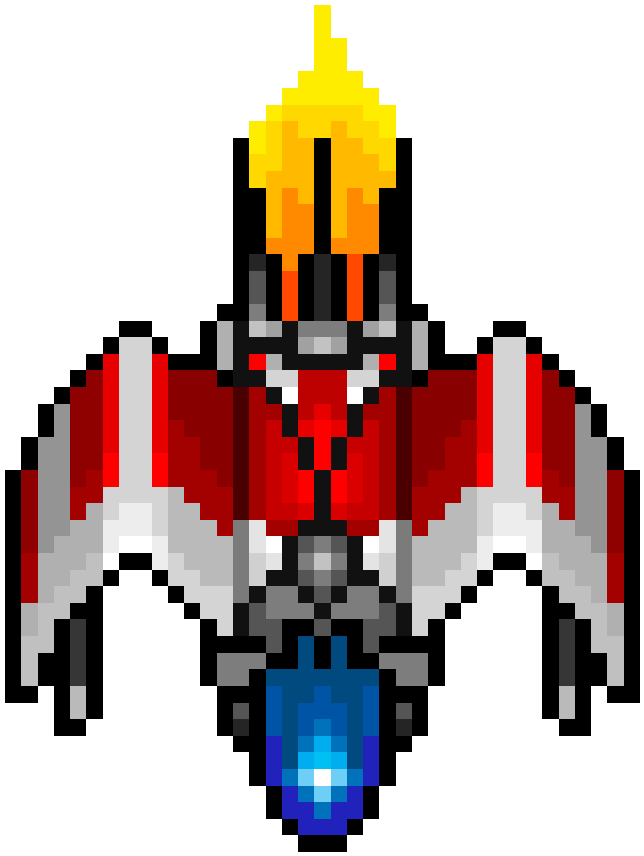


Рисунок 11 — Первый вариант спрайта главного героя

Первоначально планировалась графика в стиле «pixel art», поэтому первый спрайт имел размеры 64x64 пикселя, и не имел сглаживания и элементов с прозрачностью. Позже было принято решение отойти от данного стиля, и использовать обычные спрайты. Для этого были созданы эскизы

возможных вариантов. При создании эскизов выбор цветовой гаммы основывался на цветах красного креста — красного и белого (рисунок 12).

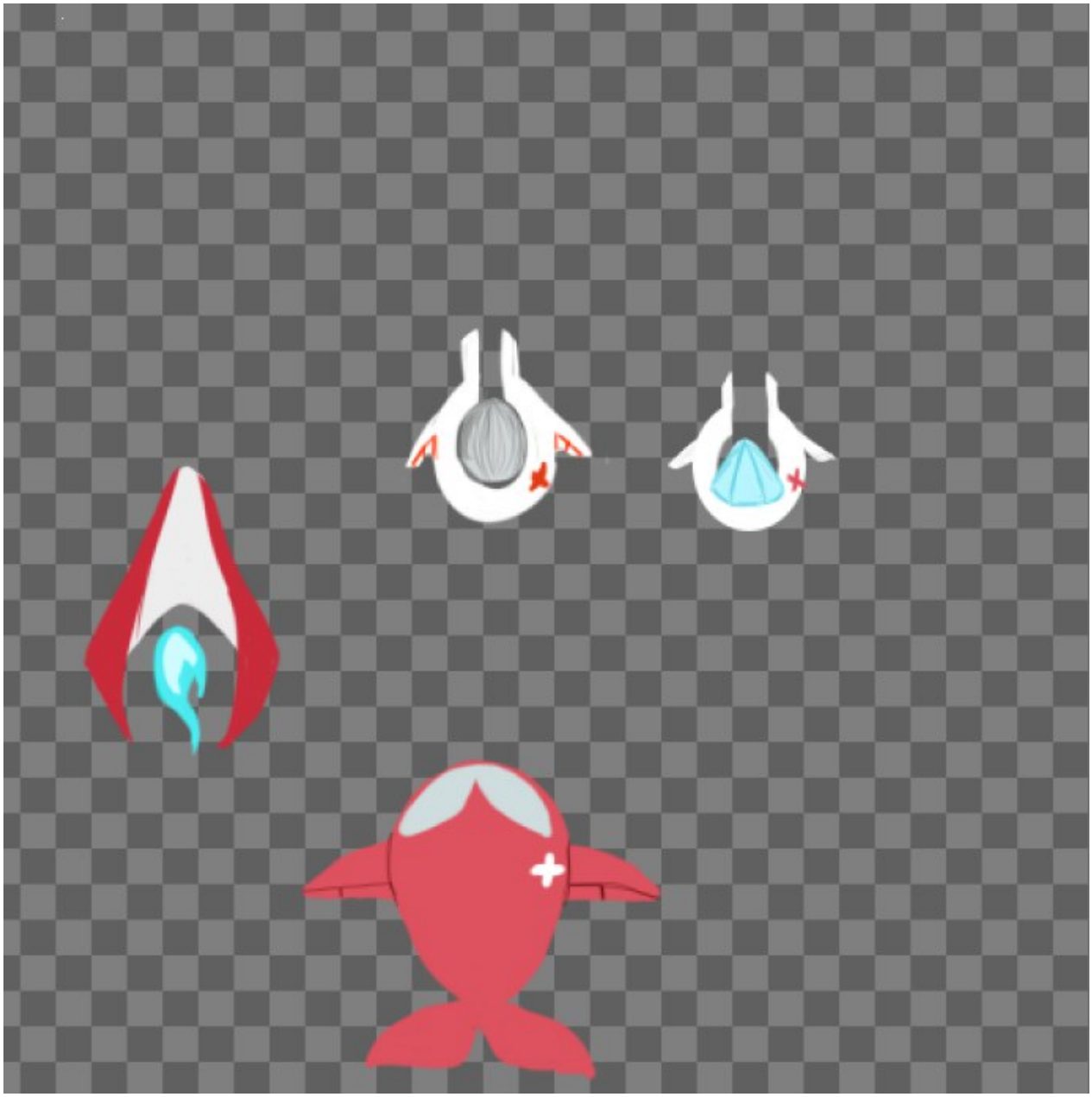


Рисунок 12 — Эскизы главного героя

После выбора концепции, эскиз был закончен. Выбор пал на корабль самой обтекаемой формы, так как персонаж в игре должен быть быстрым и мобильным. Также выбранный корабль имеет очень футуристический внешний вид, соответствующий популярному на сегодняшний день «sci-fi» сеттингу. Стиль сбыл выбран максимально простой, без излишней детализации,

которая может отвлечь игрока. Финальный вариант спрайта представлен на рисунке 13.

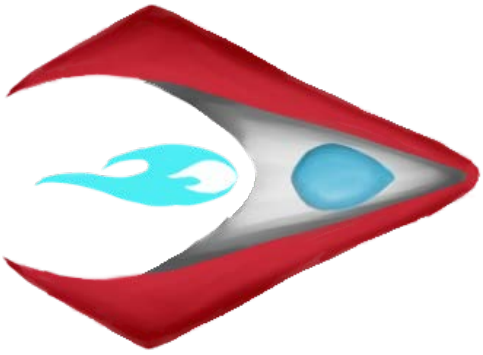


Рисунок 13 — Финальная версия спрайта главного героя

По аналогии были созданы спрайты противников. Под противниками в контексте игры понимаются болезни, в частности, вирусы и бактерии. Однако сам по себе образ бактерии не является чем-то интересным, поэтому было принято решение, в некотором роде одушевить эти образы, и на их основе создать спрайты (рисунок 14).

Рисунок 14 — Спрайт босса уровня (Грипп)

При разработки графической составляющей интерфейса изначально были известны необходимые компоненты, нуждающиеся в прорисовке. В соответствии с жанром, необходимо было визуализировать два значения: количество жизней и количество бомб игрока. Для визуализации было принято решение использовать стандартные фигуры, используя лишь контур и не закрашивая их. Финальные варианты иконок представлены на рисунках 15, 16.

Рисунок 15 — Иконка покупки жизни

Рисунок 16 — Иконка покупки бомбы

Для интерфейса было выбрано два цвета, фиолетовый один в качестве основного и светло-синий для надписей и активных элементов. Для минимизации объёма интерфейса, кроме основных индикаторов было решено создать всего одну панель, участвующую в игровом процессе (рисунок 17).

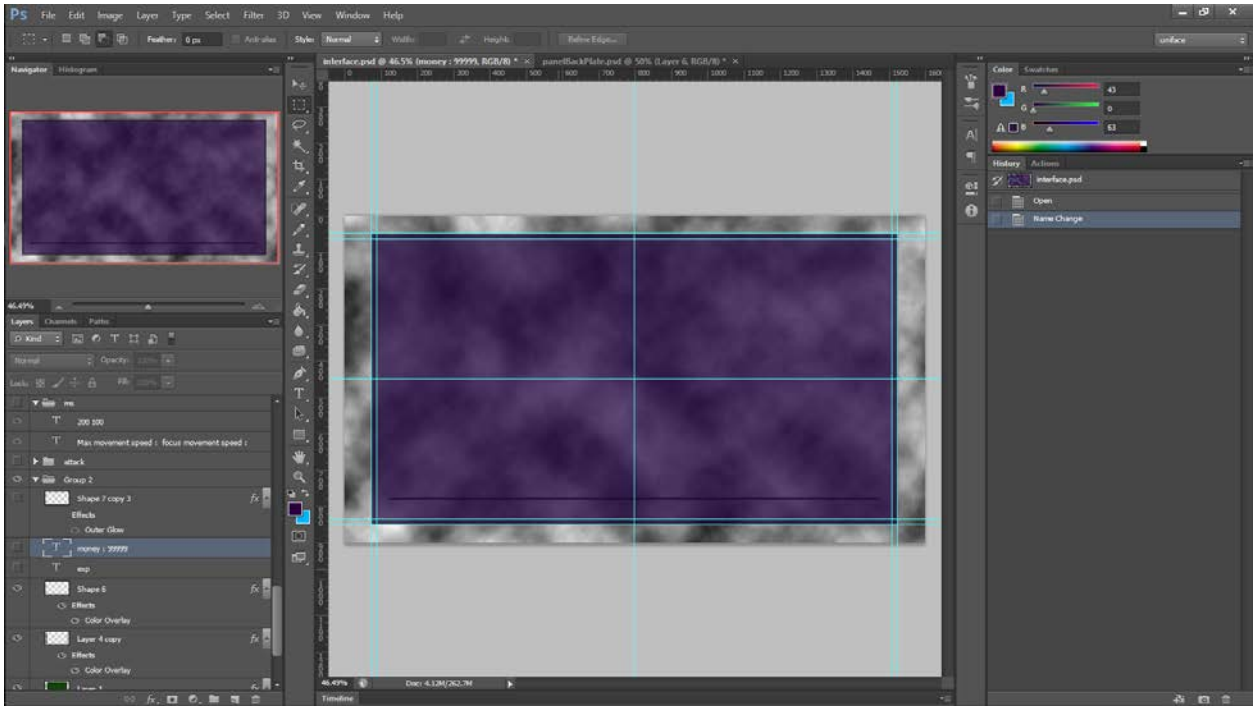


Рисунок 17 — Процесс создания панели

В итоге была создана панель, содержащая все возможные улучшения, что может получить игрок в процессе прохождения игры. Элементы на панели были сгруппированы по своему назначению, а счётчики важных для игрока значений вынесены в отдельную часть панели, что облегчает восприятие. Также были созданы спрайты кнопок улучшений, принимающий активную и неактивную форму в зависимости от обстоятельств и возможности улучшить

тот или иной параметр. Панель, как и весь интерфейс имеет единый шрифт. Финальная версия панели представлена на рисунке 18.

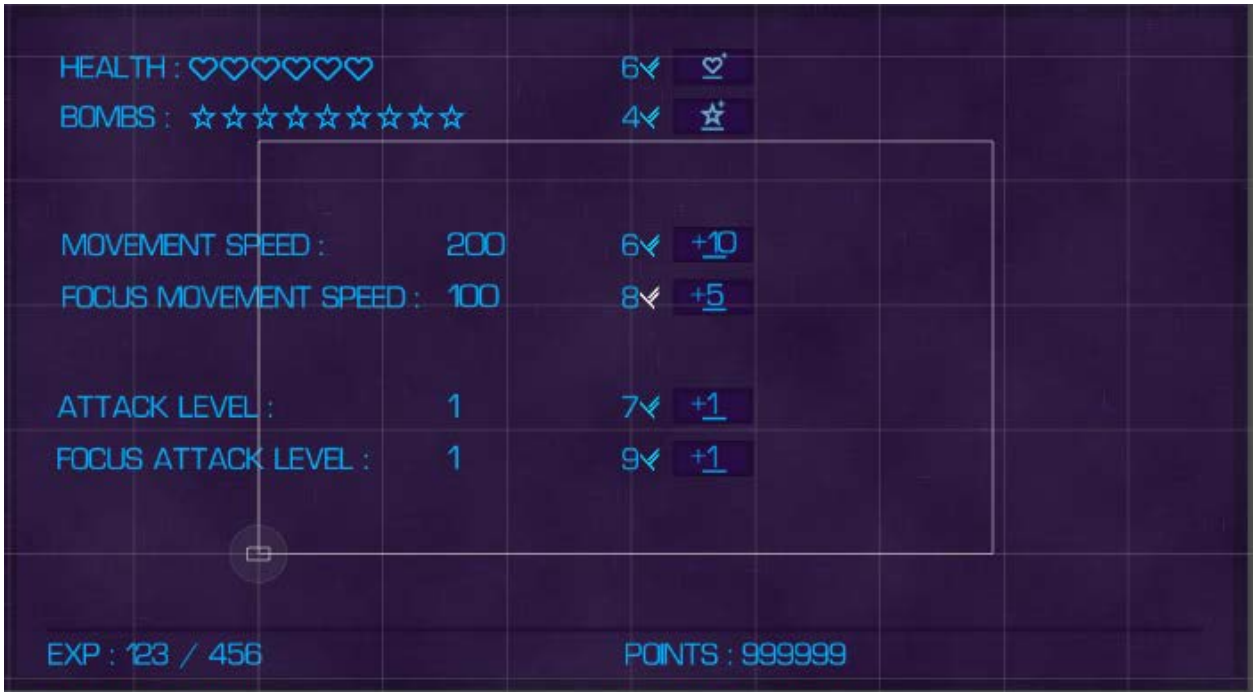


Рисунок 18 — Финальный вариант панели

Противники и игрок используют разные типы атак, и разные проджектайлы. Для лучшего контроля за игровым полем, игрок должен различать свои и вражеские проджектайлы, а также тип проджектайлов, несущие для него опасность. Однако, не стоит слишком сильно варьировать проджектайлы по форме, ибо они должны быть легко запоминающимися, и группироваться по форме своим поведением. Таки образом близкие по поведению проджектайлы должны быть одной формы, но разного, например, цвета. Разные же по форме проджектайлы должны разительно отличаться формой спрайта, и цвет в данном случаи несёт лишь второстепенное значение. Также форма и цвета на проджектайле должны подсказывать игроку, какая область проджектайла может нести для него угрозу, так как проджектайлы, как и игрок, имеют хитбокс не в величину всего спрайта, а лишь определённую часть, для упрощения возможности покинуть опасную зону в скоплении опасных объектов, включая самих противников, так как они тоже имеют хитбокс. Для этого было прорисовано несколько общих спрайтов, визуальное

отображение которых, в основном размер, могут быть изменены средствами движка, в случаи необходимости. Примеры спрайтов проджектайлов представлены на рисунке 19.

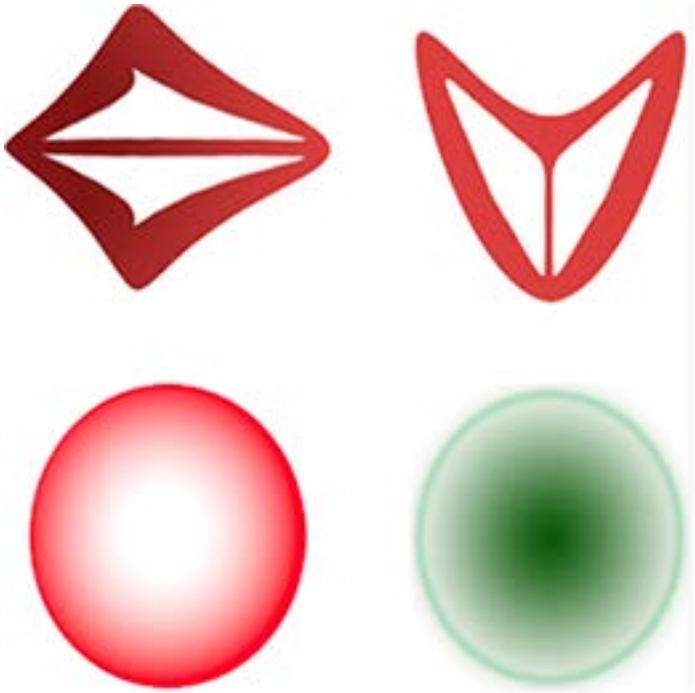


Рисунок 19 — Примеры проджектайлов

Каждый спрайт привязан к определённому типу проджектайла. Для увеличения вариативности спрайтов, использовалось изменение их цвета (рисунок 20).

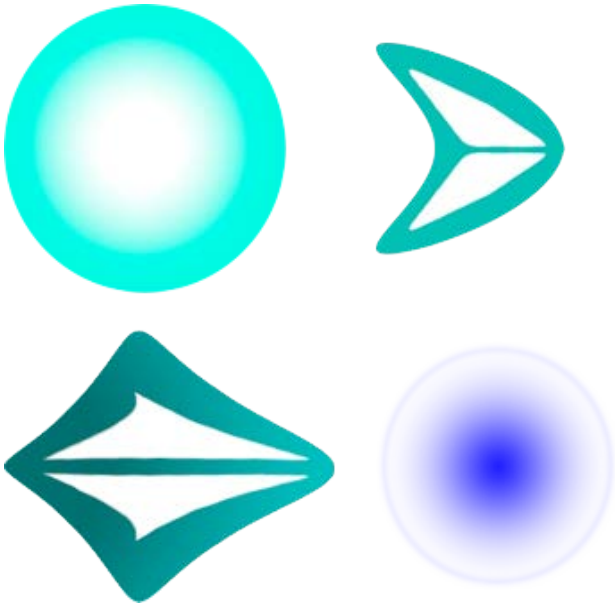
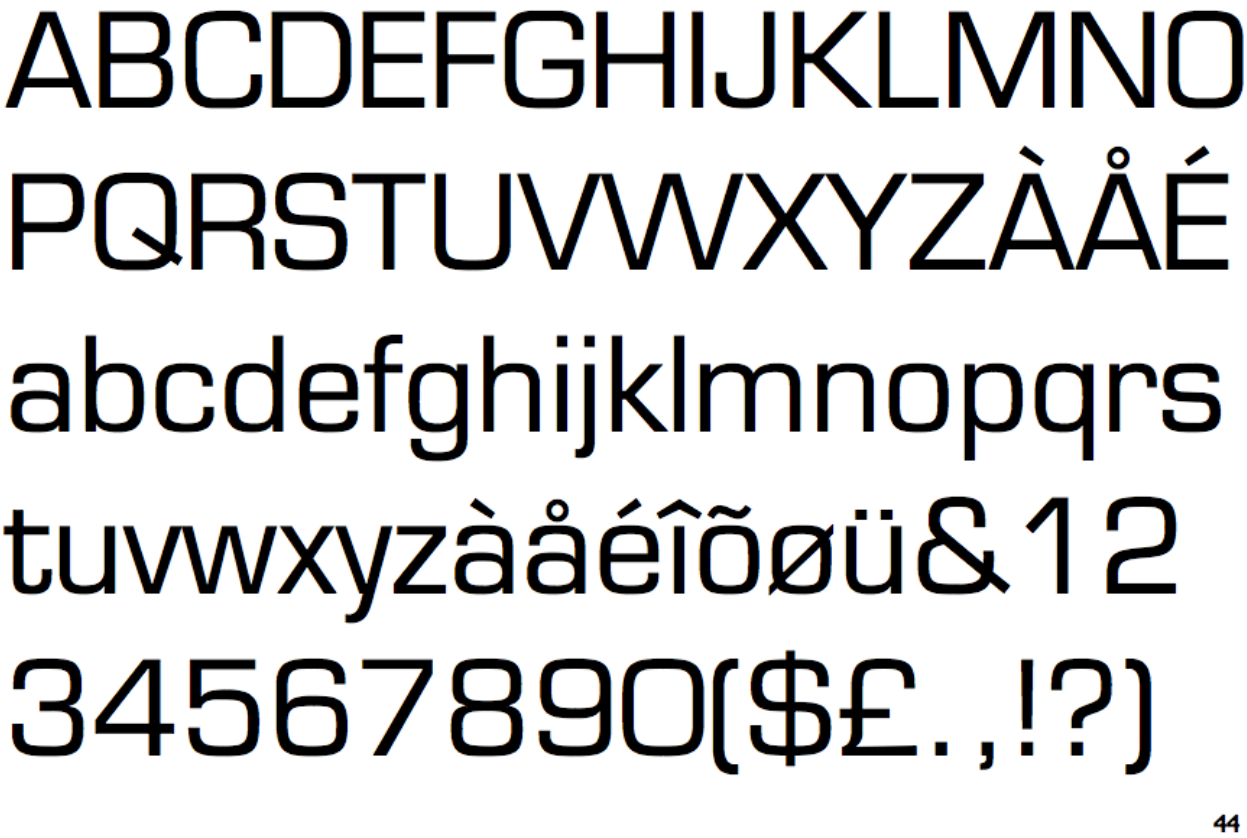


Рисунок 20 — Примеры проджектайлов

В качестве шрифта к проекту, был выбран шрифт «EuroStile» (рисунок 21). Его лицензия позволяет использовать его во многих ситуациях, в том числе при разработке программных продуктов, включая компьютерные игры.



44

Рисунок 21 — Пример шрифта «EuroStile»

2.2.5 Характеристика оборудования для реализации проекта

Для реализации проекты был использован персональный компьютер с приведёнными ниже характеристиками:

- Операционная система: Windows 10, разрядность 64 бит;
- процессор: Intel Core i7-5820K, частота процессора 3300МГц, 6 ядер;
- объём оперативной памяти: 16Гб;
- тип оперативной памяти: DDR4, частота 2400МГц;
- дискретная видеокарта: MSI GTX970 Gaming 4G, объём видеопамяти 3584Мб.

В качество дополнительного устройства ввода испльзовался графический планшет Wacom Intuos Pen S.

2.3 Жизненный цикл проекта. Описание поэтапной реализации проекта с указанием средств реализации

2.3.1 Этап написания базовой механики

Первым этапом в реализации игры являлось формулирование и написание базовой механики, включавшей в себя контроль персонажа, поведение камеры и тестовых противников. Для игрока были написаны скрипты передвижения, и поворота за мышью, также был добавлен скрипт обрабатывающий нажатие клавиш, и взаимодействующий с первыми двумя, для реализации движения персонажа. Для камеры был создан скрипт, позволяющий ей перемещаться вслед за игроком. Такую возможность можно получить, используя иерархию объектов, и переместив камеру в дочерние элементы игрока, что закрепит камеру за позицией игрока. Однако, при таком подходе теряется полный контроль за камерой, а любое воздействие на игрока также сказывается на камере. В некоторых проектах такой подход может иметь смысл, но в данном случаи от него было решено отказаться, и написать для камеры собственный контроллер. Кроме этого, был написан скрипт, позволяющий изменять соотношение сторон камеры, масштабируя его в условиях любого разрешения. Для противников были написаны свои скрипты, для изменения положения на игровом поле, и самый простой контроллер, позволяющий находить персонажа и следовать за ним.

Необходимо обеспечить взаимодействие между противниками и игроками, обеим сторонам нужно было оружие, счётчики урона и здоровья. Для этого был создан объект-проджектайл, который использовался для атаки обеими сторонами. Созданы контроллер здоровья и контроллер урона. И персонаж, и враги имеют в своём составе одни и тот же скрипт здоровья, но с

разными установленными значениями. На проджектайл установлен скрипт контроллера урона, который, при попадании в объект, распознаёт его тэг, сопоставляет его с установленным в нём параметром тэга противника, и на основе этого сравнения, либо взаимодействует с котроллером здоровья объекта, либо нанося урон, либо игнорируя его. Для возможности взаимодействия между проджектайлом и объектами, при столкновении, на них были установлены коллайдеры, для работы которых необходимо присутствия физического тела. В Unity физическое тело «правильно» двигать его же методами, поэтому скрипты перемещения как врагов, так и персонажа были переписаны. В конце данного этапа была получена минимальная механика, позволяющая управлять персонажем и уничтожать врагов прямо летящими проджектайлами, и возможность быть уничтоженным самому.

2.3.2 Создание абстрактных классов и упрощение взаимодействия

Последующее создание объектов вызывало необходимость создания уникального скрипта для каждого объекта. Это касалось как проджектайлов, так и врагов. Для упрощения этой задачи, уже написанные скрипты были упрощены и переписаны в абстрактные классы. На их основе, если возможно, были написаны новые скрипты для контроля за объектами, что позволяло сэкономить время и значительно упростить взаимодействие между объектами, вызывая не конкретные скрипты, а их родительские абстрактные классы. Также это позволило объединить некоторые классы и упростить контроллеры врагов и проджектайлов.

2.3.3 Создание генератора паттернов и пула объектов

Дальнейшая реализация предполагает возможность создавать и управлять большим количеством проджектайлов. Такую возможность позволяют реализовать два скрипта. Первый скрипт – пул объектов, позволяет много-

кратно использовать заранее созданные объект, находящиеся в пуле. Это распространённый паттерн проектирования, используемый в случаи, когда есть возможность повторно использовать уже созданные объекты, и, если это менее ресурсозатрато, чем уничтожение и создание новых объектов. В данном случаи, создание нового проджектайла занимает у движка во много раз больше времени, чем активация выключенного, заранее созданного противника, не смотря на тот факт, что включение объекта, вызывает перерасчёт физики рядом с ним. Скрипт пула объектов создаёт до начала игры несколько пулов определённых объектов, заданного размера и участвует во всех операциях, в которых предполагается уничтожение, или создание объекта, например, если у врага заканчивается здоровье, или персонаж сделал выстрел.

Сам скрипт не должен содержать излишнюю сложность, и должен быть максимально оптимизирован, так как его работа сопровождается многочисленными ресурсоёмкими процессами, которые будут нагружать игру. Исправление или рефакторинг кода, в случаи необходимости, не должны быть осложнены лишними переменными или методами, основная, действующая часть скрипта, отвечающая непосредственно за создание объектов, задавание их свойств и собственным изменением должна быть предельно простой, короткой и ёмкой. Ниже представлен фрагмент скрипта генератора паттернов, отвечающий за всю его работу, где количество констант снижено к необходимому для работы минимуму. Для увеличения количества возможных вариаций используются несколько циклов и условных операторов, позволяющие более тонко настраивать конечный результат. Также позволяется редактировать свойства не только генератора, но и свойства созданных им проджектайлов, независимо от его типа.

```
void Fire()
    {

        for (int directionNumber = 0; directionNumber < direc-
tionsAmount; directionNumber++)
        {
            directionAngle = (totalArcAngle / directionsAmount) * di-
rectionNumber;
```

```
            currentAngle = transform.rotation.eulerAngles.z + startAn-
gle;
            for (int flowNumber = 0; flowNumber < flowsPerDirection;
flowNumber++)
            {
                if (flowsPerDirection < 1)
                    continue;
                else if (flowsPerDirection == 1)
                    flowAngle = (directionArcAngle / 2) * flowNumber;
                else
                    flowAngle = (directionArcAngle / (flowsPerDirec-
tion - 1)) * flowNumber;
                GameObject bullet = ObjectPoolingManag-
er.Instance.GetObject(projectile.name, transform.position, Quaterni-
on.Euler(0, 0, currentAngle + directionAngle + flowAngle)) as GameOb-
ject;
                bullet.tag = "EnemyProjectile";
                bullet.GetComponent<damageController>().targetTag =
"Player";
                bullet.GetComponent<ProjectileMoverBase>().speed =
bulletSpeed;
                if (startAngle >= 360)
                    startAngle -= 360;
                if (startAngle <= -360)
                    startAngle += 360;
            }
        }
        startAngle += Mathf.Tan(rotationSpeed * (1/tanMultiplaier)) *
Mathf.Rad2Deg;
        rotationSpeed += rotationSpeedAcceleration;
        if (Mathf.Abs(rotationSpeed) >= maxRotationSpeed)
            rotationSpeedAcceleration = -rotationSpeedAcceleration;
        bulletSpeed += bulletSpeedAcceleration;
        if (bulletSpeed >= maxBulletSpeed || bulletSpeed <= minBul-
letSpeed)
            bulletSpeedAcceleration = -bulletSpeedAcceleration;
     }
```

Созданный генератор паттернов позволяет на основе разных входных параметров создавать множество различных паттернов (рисунок 22).

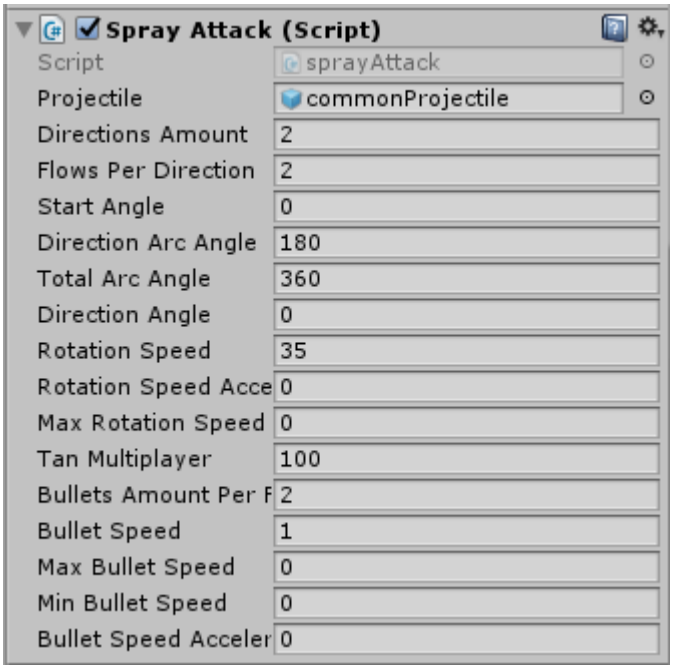


Рисунок 22 — Входные параметры генератора паттернов

Список параметров скрипта:

1. Projectile — тип снаряда, используемого в скрипте.
2. Directions Amount — количество направлений атаки скрипта.
3. Flows Per Direction — количество потоков снарядов на каждое направление.
4. Start Angle — начальный угол отсчёта вращения направлений атаки.
5. Direction Arc Angle — угол, занимаемый направлением, в который будут помещены все входящие в него потоки.
6. Total Arc Angle — полный угол всей атака, в который будут помещены все направления.
7. Direction Angle — высчитываемый скриптом угол между направлениями.
8. Rotation Speed — скорость изменения угла отсчёта вращения скрипта.
9. Rotation Speed Acceleration — ускорение скорости изменения угла.
10. Max Rotation Speed — максимальная скорость изменения угла отсчёта вращения. При достижении это величины, ускорение принимает обратную своей величину.

11. Tan Multiplier — дробная часть функции тангенса для вращения угла отсчёта.

12. Bullets Amount Per Flow — количество снарядов, генерируемых каждым потоком в секунду.

13. Bullet Speed — задаваемая скорость полёта снарядов.

14. Max Bullet Speed — Максимальная задаваемая скорость полёта снарядов.

15. Min Bullet Speed — Минимальная задаваемая скорость полёта снарядов.

16. Bullet Speed Acceleration — ускорение задаваемой скорости полёта снарядов. При достижении скорости заданной максимальной, или минимальной, ускорение становится равно своему обратному значению.

Для большего покрытия возможностей были написаны другие генераторы, позволяющие другие сценарии реализации паттернов. Комбинации паттернов и проджектайлов позволяют получить колоссальное множество различных вариаций атак. Внешний вид паттерна во многом зависит не только от генератора, но и от типа проджектайла, используемого в этом паттерне. Для максимальной вариативности было создано 13 типов проджектайлов, имеющих своё поведение на игровом поле.

2.3.4 Создание котроллера игры

После реализации всех необходимых комбинаций для создания противников и атак, необходимо структурировать геймплей, который будет вести игрока. Чтобы решить эту задачу, создан отдельный объект – контроллер игры, хранящий в себе множество скриптов, отвечающих за состояние игры и генерацию её частей. Контроллер хранит в себе скрипт пула объектов, скрипты, отвечающие за состояние игры, генерирующие комнаты по ходу прохождения, генерирующие врагов и считающие их, структурируя геймплей

и ведя игрока от комнаты к комнате до босса, контролирует состояние игрока.

По своей сути контроллер игры представляет из себя пустой объект, содержащий все скрипты, которые должны исполняться по ходу игры, но не имеют физического представления в ней. Для исполнения скриптов, наследующих класс MonoBehaviour, они обязаны быть помещены в иерархию сцены, так как не могут исполняться без созданного своего экземпляра внутри сцены. Для этого в Unity обычно создаются объекты, которые называют логически близкими именами к поведению этого скрипта. Таким образом в сцене присутствуют скрипты-контроллеры интерфейса, находящиеся в родительских объектах самого интерфейса. В данном же случаи контроллер игры – это набор жизненно важных для игры и геймплея скриптов, находящихся в дочерних или непосредственно самом объекте контроллера игры.

2.3.5 Создание интерфейса

Когда вся механика готова и находится в рабочем состоянии, в игру добавляется интерфейс, разработка которого представлена в пункте 2.2.4. Чтобы обеспечить взаимодействие всех элементов с интерфейсом, некоторые скрипты необходимо изменить, создав или переписать в них методы, отвечающие за доступность и расчёт данных. Интерфейс также нуждается в собственных скриптах-контроллерах, для перехвата нажатия клавиш, и перехвата информации из остальных скриптов. Для удобства, во время вызова панели интерфейса, изменяется скорость времени внутри игры, останавливая его. Это замораживает любые объекты внутри игры, так как расчёт их передвижения идёт через физическое тело, напрямую связанное с изменением времени внутри игры.

Спрятать все элементы интерфейса не сложно, если не задумываться о количестве действий нужно сделать игроку, чтобы выполнить какое-либо действие. Оставить все элементы на экране тоже не очень проблематично,

если не обращать внимания на процент занятого на экране пространства. При создании интерфейса, нужно соблюдать баланс между этими двумя крайностями, и баланс этот может смещаться в зависимости от конечной цели. В рамках проекта, существует необходимость максимально снизить площадь, занимаемую интерфейсом в процессе игры, и лишь в моменты паузы, например, при вызове панели улучшений, когда игра встаёт на условную паузу, можно отдать под интерфейс до 100% площади игрового экрана, так как это ничем не грозит игроку. Интерфейс должен помогать игроку в игре, а не являться очередным её противником [6].

2.3.6 Привязка спрайтов к объектам, создание заднего фона

Каждый объект в игре нуждается в визуализации, для этого были использованы спрайты, разработка которых представлена в пункте 2.2.4. Спрайты масштабируются на объекте, так как имеют разные размеры. Также, некоторые спрайты необходимо сместить, относительно центра объекта, для верного размещения хитбокса. Несколько спрайтов можно использовать на разных объектах, изменив их форму и цвет инструментами Unity. Движок позволяет складывать выбранный цвет с цветом спрайта, и изменять размер по осям X, Y и Z, независимо друг от друга.

Игра не предполагает наличие на заднем фоне каких-либо конкретных объектов, поэтому, при создании, спрайтов было принято решение максимально абстрагировать конечное изображение от каких-либо образов. Задачу также усложняет и выбранный сеттинг, где невозможно конкретно сказать, что должно быть на заднем плане, в качестве визуализации организма. Для решения этой проблемы задний фон должен состоять из трёх слоёв и иметь эффект «Паралла́кс». Для этого было создано три бесшовных текстуры, представленные на рисунках 23-25.

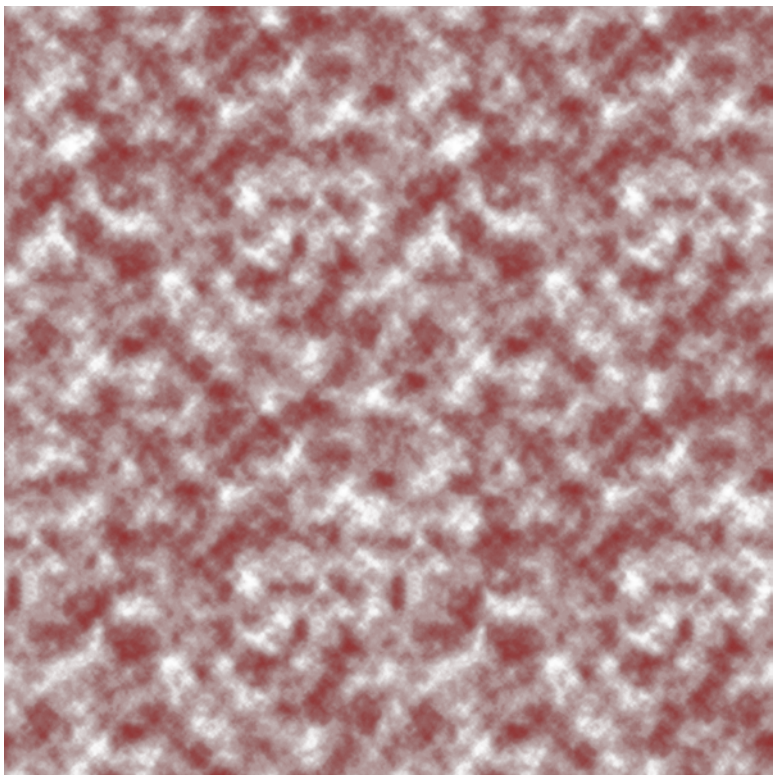


Рисунок 23 — Первый (верхний) слой заднего фона

Верхний слой имеет наибольшую прозрачность, и наименьший размер текстуры. Для него создан пустой файл с фильтром «Облака», убраны швы на краях картинки и создана прозрачность по всему белому цвету.

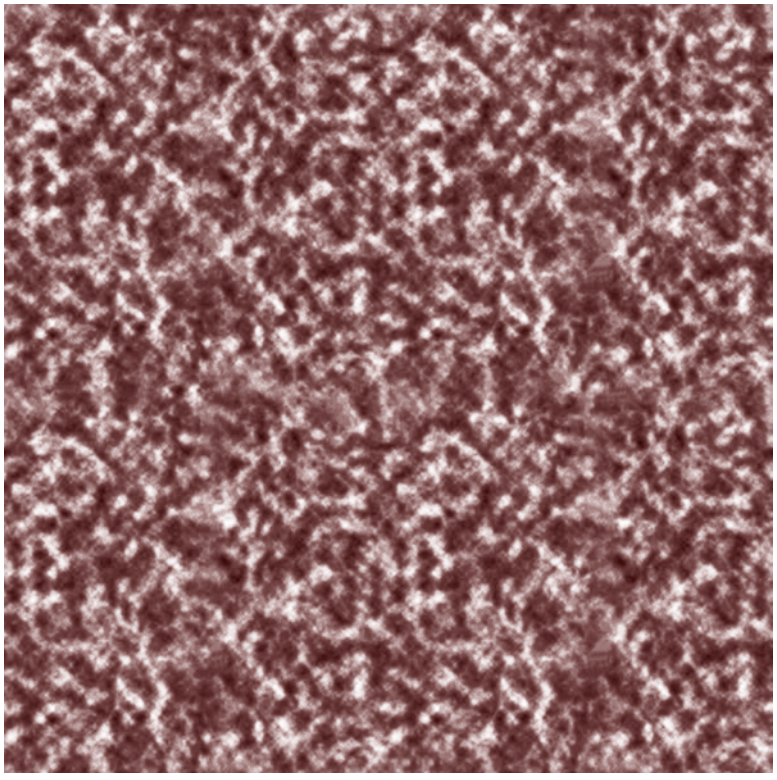


Рисунок 24 — Второй (средний) слой заднего фона

Средний слой также был создан с помощью фильтра «Облака», однако, текстура имеет больший размер, за счёт чего белого фона на ней меньше, и текстура выглядит плотнее. В процессе создания убраны швы и белый цвет заменён прозрачностью.

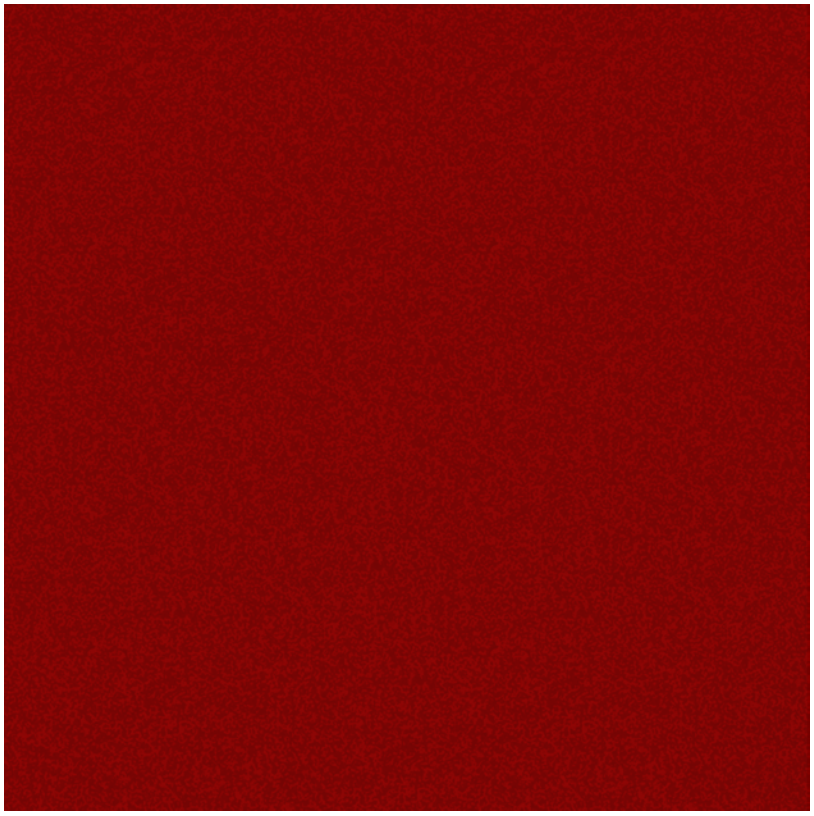


Рисунок 25 — Третий (задний) слой заднего фона

Задний слой не имеет прозрачности и является своего рода подложкой для первых двух, заканчивая создание эффекта «Параллакс». Сам слой при этом тоже находится в движении, однако, с гораздо меньшей скоростью.

Для создания эффекта, первые два слоя имеют прозрачность, чтобы с любого положения камеры на игровом поле, можно было увидеть все три слоя. Для создания эффекта в Unity, существует несколько подходов, однако, условие, что изначально неизвестны размеры мира, и где будут генерироваться комнаты, усложняет задачу. Нельзя разметить спрайты ровно на пути игрока изначально, но можно загружать задний фон по ходу движения игрока, либо замостить спрайтами поле, максимально возможное при генерации комнат. Оба варианта будут очень требовательны к вычислительной мощности, т.к. текстуры имеют большое разрешение, а для создания эффекта, их

необходимо перемещать с разной скоростью. В качестве альтернативы, был выбран метод, характерный для 3D проектов Unity. В качестве заднего фона, будет использоваться не 2D спрайт, а трёхмерный объект Plane (Плоскость), с материалом, в котором использована текстура спрайта. Для трёхмерных объектов в Unity необходимо освещение, поэтому, был также создан направленный источник света. Таким образом задний фон состоял из трёх плоскостей и источника света. Для создания эффекта «Параллакс» необходимо смещать текстуру на каждой панели с разной скоростью, используя параметр «offset». Именно необходимость использования смещения текстуры не позволяла использовать стандартный 2D спрайт, т.к. Unity не позволяет использовать смещение текстуры на спрайте. Величина смещения высчитывается от передвижения игрока, смещая первый уровень быстрее остальных. Самым медленным является третий уровень. Плоскости имеют ограниченную площадь, поэтому положение плоскостей, а также источника света, закреплялись за положением игрока. При перемещении по игровому полю, за игроком следовали источник света и панели, а на каждой панели, в зависимости от множителя скорости, смешались текстуру, образуя необходимый эффект. Unity позволяет использовать 3D объекты в 2D проектах и наоборот, так как создание любого типа проекта происходит в одной и той же среде. Изменения касаются лишь параметров редактора, и начальных задаваемых параметров каждого объекта. Например, при добавлении на объект изображения в 2D проекте, изображению присваивается шейдер спрайта, тогда как в 3D проекте – шейдер текстуры. Любой параметр можно изменить в процессе создания, поэтому особой разницы между выбранными вариантами проекта нет.

2.3.7 Этап тестирования

После всех этапов разработки и сборки проекта, необходимо протестировать готовый уровень игры, на наличие ошибок, несоответствий и уровень производительности. Визуальная часть игры работает исправно, все спрайты

имеют правильный размер, хитбоксы соответствуют элементам спрайтов. Интерфейс работает исправно, счётчики соответствуют реальным значениям, кнопки работают без ошибок.

Однако, возникает проблема с производительностью, когда на экране появляется некоторое количество определённого типа врагов. Пул объектов, отвечающий за проджектайлы, которыми пользуется данный тип противника, достигает верхней границы, и не способен больше увеличиваться. Таким образом, когда противник пытается выстрелить, он пытается вызвать из пула свободный объект, но так как пул уже использует все объекты, доступные для вызова, это приводит к ошибкам внутри игры и падению значения кадров в секунду. В нормальном режиме, при тестировании, значение кадров в секунду держится на отметке 75-80 кадров. Во время вызова с ошибками, значение кадров в секунду опускается до 13-15 кадров. При таком значении играть становится невозможно, т.к. при такой частоте обновления кадров, нельзя отследить траекторию полёта проджектайлов, более того, начинается рассинхронизация игрового процесса и расчётов движка, так как всё движение внутри игрового поля рассчитывается физическим движком, независимо от показателя кадров в секунду, а считывание нажатия клавиш игроком, напрямую зависит от частоты обновления кадров. Расширение пула нивелирует проблему с недостатком объектов для вызова, однако, при таком количестве проджектайлов, при вызове каждого нового объекта возникает ситуация, описанная в пункте 1.4.1, когда при активации физического объекта, с имеющимся в нём коллайдере, физический движок пересчитывает физику всех находящихся рядом объектов. При дальнейших исследованиях выяснилось, что такая ситуация возникает, при нахождении на экране от трёх с половиной тысяч объектов, что не обосновано геймплейно, так как большинство этих объектов не несёт ценности в игровом процессе. Паттерн атаки, используемый противниками, и приводящий к этой ситуации, несёт для игрока не на столько большую опасность, чтобы использовать так много объектов. На этих данных было принято решение изменить, как атаку противника, так и

модель его поведения, чтобы избежать ситуации падения производительности до критических отметок. Изменение типа противника и его атаки, приводит к невозможности использовать часть уровня, связанную с данным типом противника. Необходимо исправить этот фрагмент уровня, изменив скрипт генерации данной комнаты.

После исправлений найденных проблем, падения производительности на всех этапах игры не обнаружено. Количество одновременно находящихся объектов на экране не превышает трёх тысяч и не ведёт к критическим последствиям.

2.4 Технические требования к проекту

Технические требования к проекту рассчитаны на основе тестовых запусках на машинах, с разными характеристиками. Минимальным приемлемым показателем кадров в секунду была принята величина в 40 кадров в секунду. Также использовалась требования, опубликованные разработчиками движка, представленные на сайте Unity. На основе сведений, полученных от тестов, и изученных на сайте, составлены следующие минимальные технические требования:

- ОС: Windows XP SP2;
- процессор: Intel Core i5-4300u;
- видеокарта: Дискретная NVidia GeForce GT 420, либо интегрированная Intel HD Graphics 530;
- объём ОЗУ: 4Гб.
- тип ОЗУ: DDR3, частота 1333Мгц.

2.5 Калькуляция проекта

В ходе выполнения проекта было создано множество разных элементов, используемых в процессе реализации игры. Все созданные компоненты представлены в таблице 2.

Таблица 2 – Калькуляция проекта

Объект	Количество	Комментарий
Скрипты	83	Скрипты написаны на языке программирования C#, с использованием библиотек Unity, предоставляемых движком.
Спрайты	44	Нарисованы в программе Adobe Photoshop CC 2017, с использованием графического планшета. Под спрайтом также понимается спрайт, использованный в качестве текстуры для материала
Сцены	2	Сцена – контейнер для объектов игры. Каждая отдельная сцена может представлять, как, например, отдельный уровень, так и меню.
Проджектайлы	17	Каждый проджектайлый отличается внешним видом и поведением внутри игры.

Окончание таблицы 2

Префабы	27	Шаблон игрового объекта, хранящий все его компоненты и параметры. Существует для создания точных копий в процессе игры.
Материалы	5	Материалы используются в качестве текстур для разных компонентов движка.

ЗАКЛЮЧЕНИЕ

На сегодняшний день компьютерные игры являются неотъемлемой частью индустрии развлечений. К сожалению, в России данная отрасль развита не столь сильно, а в обществе до сих пор существует скептицизм по отношению к компьютерным играм, как к явлению. Не смотря на всё это, сектор инди игр всё сильнее развивается и набирает популярность, а средства разработки становятся всё более доступными для всех желающих. Тем не менее разработка компьютерных игр – долгий и трудоёмкий процесс, требующий много усилий.

В процессе выполнения проекта, были проанализированы предметная область, и имеющиеся внутри неё разработки. Исследованы программные средства раработки, проанализированы сильные и слабые стороны различного программного обеспечения для реализации и, на основе полученных данных, выбраны средства реализации проекта. В конечном итоге, была реализована полноценная компьютерная игра, с соответствующими выбранному направлению механиками и геймплеем. Для реализации было создано множество строк кода, позволяющие собрать цельную игровую механику, и формирующие полноценный геймплей, вкупе с созданными спрайтами, формирующими уникальную игровую стилистику. В процессе создания учитывались особенности работы игрового движка, что позволило реализовать основной аспект «danmaku» игр – нахождение огромного числа объектов на игровом поле.

Выполнение проекта в рамках выпускной квалификационной работы, посвящённой реализации компьютерной игры с нуля, позволило получить колоссальный опыт, полезный не только в рамках разработки компьютерных игр. На основе полученного опыта, можно продолжать совершенствование навыков в разработке, и непосредственно сами разработки игр. Игровой движок Unity является очень востребованным на рынке. Многие корпорации та-

кие как Google, Intel, Apple и другие, являются оффициальными партнёрами Unity, что обеспечивает развитие движка не только в направлении компьютерных игр. За счёт этого, среда использования движка имеет области, не связанные с разработкой игр. Движок часто используется для создания «try before you buy» проектов, позволяющие продемонстрировать товар покупателю, до его непосредственной покупки, что актуально в сфере строительства. Unity так же используется в разработке обучающих приложений, чья возрастная категория не ограничивается детьми. Всё это повышает ценность умений работы в выбранном средстве разработки.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Гуриков С.Р. Введение в программирование на языке Visual C#: учебное пособие [Текст] / С.Р. Гуриков — Москва: ФОРУМ : ИНФРА-М, 2013 — 448 с.

2. Дикинсон К. Оптимизация игр в Unity 5 [Текст] / К. Дикинсон. — Москва: ДМК, 2017. — 306 с.

3. История видеоигр [Электронный ресурс]. — Режим доступа: http://history-of-video-games.webflow.io/istoriya-videoigr (дата обращения: 23.04.2017).

4. Купер А. Интерфейс. Основы проектирования взаимодействия [Текст]/ А. Купер, Р. М. Рейманн, Д. Кронин. — Под общ. ред. А. Купера. – Санкт-Петербург: Питер, 2017. — 720 с.

5. Павловская Т.А. C# Программирование на языке высокого уровня [Текст] / Т.А. Павловская — Санкт-Петербург: Питер, 2013 — 432 с.

6. Уэйншенк С. 100 новых главных принципов дизайна. [Текст]/ С. Уэйншенк. — Санкт-Петербург: Питер, 2016. — 288 с.

7. Хабрахабр [Электронный ресурс]. — Режим доступа: https://habrahabr.ru/post/248391/ (дата обращения: 07.02.2017).

8. Хакер [Электронный ресурс]. — Режим доступа: https://xakep.ru/2014/09/05/game-development-engines-review/ (дата обращения: 10.12.2016).

9. Хокинг Дж. Unity в действии. Мультиплатформенная разработка на C# [Текст] / Дж. Хокинг — Пер. с англ. И. Рузмайкиной. — Санкт-Петербург: Питер, 2016. — 336 с.

10. app-s.ru [Электронный ресурс]. — Режим доступа: http://app-s.ru/index/genres_of_games/0-52 (дата обращения: 24.04.2017).

11. Blogger [Электронный ресурс]. — Режим доступа: http://gamedesigntheory.blogspot.ru/2010/09/controlling-aspect-ratio-in-unity.html (дата обращения: 10.03.2017).

12. Catlike Coding [Электронный ресурс]. — Режим доступа: http://catlikecoding.com/unity/tutorials/object-pools/ (дата обращения: 24.03.2017).

13. DevGam [Электронный ресурс]. — Режим доступа: http://devgam.com/sravnenie-igrovyx-dvizhkov-kakoj-vybrat (дата обращения: 10.12.2016).

14. evatotuts+ [Электронный ресурс]. — Режим доступа: https://gamedevelopment.tutsplus.com/tutorials/positioning-on-screen-indicators-to-point-to-off-screen-targets--gamedev-6644 (дата обращения: 14.03.2017).

15. GameMaker Studio 2 [Электронный ресурс]. — Режим доступа: http://docs2.yoyogames.com/ (дата обращения: 07.12.2016).

16. Gamer [Электронный ресурс]. — Режим доступа: http://www.gamer.ru/everything/kultovye-videoigry-vypusk-pervyy-1950-1980g (дата обращения: 19.12.2016).

17. Metanit [Электронный ресурс]. — Режим доступа: https://metanit.com/sharp/ (дата обращения: 25.02.2017).

18. Microsoft [Электронный ресурс]. — Режим доступа: https://docs.microsoft.com/ru-ru/dotnet/csharp/csharp (дата обращения: 23.03.2017).

19. MSDN [Электронный ресурс]. — Режим доступа: https://msdn.microsoft.com/library (дата обращения: 19.03.2017).

20. ProfessorWeb [Электронный ресурс]. — Режим доступа: https://professorweb.ru/my/csharp/charp_theory/level1/infocsharp.php (дата обращения: 30.03.2017).

21. Stack Overflow C# [Электронный ресурс]. — Режим доступа: https://stackoverflow.com/questions/tagged/c%23 (дата обращения: 13.01.2017).

22. Stack Overflow Unity [Электронный ресурс]. — Режим доступа: https://stackoverflow.com/questions/tagged/unity3d (дата обращения: 20.03.2017).

23. Steam [Электронный ресурс]. — Режим доступа: http://store.steampowered.com/ (дата обращения: 24.04.2017).

24. Touhou Wiki [Электронный ресурс]. — Режим доступа: https://ru.touhouwiki.net/wiki/%D0%97%D0%B0%D0%B3%D0%BB%D0%B0%D0%B2%D0%BD%D0%B0%D1%8F_%D1%81%D1%82%D1%80%D0%B0%D0%BD%D0%B8%D1%86%D0%B0 (дата обращения: 23.04.2017).

25. Unity Manual [Электронный ресурс]. — Режим доступа: https://docs.unity3d.com/Manual/ (дата обращения: 12.01.2017).

26. Unity [Электронный ресурс]. — Режим доступа: https://unity3d.com/ru/learn/tutorials/s/scripting (дата обращения: 25.01.2017).

27. Unity3D.ru [Электронный ресурс]. — Режим доступа: http://unity3d.ru/distribution/viewforum.php?f=11 (дата обращения: 01.02.2017).

28. Unreal Engine 4 Documentation [Электронный ресурс]. — Режим доступа: https://docs.unrealengine.com/latest/INT/ (дата обращения: 08.12.2016).

29. Unreal Engine [Электронный ресурс]. — Режим доступа: https://www.unrealengine.com/what-is-unreal-engine-4 (дата обращения: 08.12.2016).

ПРИЛОЖЕНИЕ

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
профилю подготовки «Информационные технологии в медиаиндустрии»

УТВЕРЖДАЮ
Заведующий кафедрой
_____Н. С. Толстова
«_____»_____2017 г.

ЗАДАНИЕ
на выполнение выпускной квалификационной работы бакалавра

студента 4 курса, группы ИТм-401 Борисова Алексей Алексеевича

1. Тема Разработка компьютерной игры в жанре скроллер
утверждена распоряжением по институту от 07.02.2017 г. № 73.

2. Руководитель старший преподаватель, Садчиков Илья Александрович

3. Место преддипломной практики РГППУ

4. Исходные данные к ВКР Дизайн документ. Анализ разработок-аналогов

5. Содержание текстовой части ВКР (перечень подлежащих разработке вопросов)
Разработка основных математических алгоритмов
Разработка и балансировка основных игровых механик
Изучение Unity 5 Engine
Создание спрайтов и графических элементов игры
Программирование игры и сборка демонстрационной версии
6. Перечень демонстрационных материалов
Презентация
Демонстрационная версия игры

7. Календарный план выполнения выпускной квалификационной работы

№ п/п	Наименование этапа ВКР	Срок выполнения этапа	Процент выполнения ВКР	Отметка руководителя о выполнении
1	Сбор информации по выпускной работе и сдача зачета по преддипломной практике	17.04.2017	15	
2	Выполнение работ по разрабатываемым вопросам их изложение в выпускной работе:			
	Разработка математических алгоритмов игры	01.10.2016	100%	
	Создание спрайтов и элементов UI	01.12.2016	100%	
	Внедрение алгоритмов в код программы	01.02.2017	100%	
	Создание процедурного генератора уровней	01.03.2017	100%	
	Разработка демонстрационной версии	01.05.2017	100%	
3	Оформление текстовой части ВКР	30.05.2017	5	
4	Выполнение демонстрационных материалов к ВКР	01.06.2017	5	
5	Нормоконтроль	03.06.2017	5	
6	Подготовка доклада к защите в ГЭК	09.06.2017	5	

8. Консультанты по разделам выпускной квалификационной работы

Наименование раздела	Консультант	Задание выдал		Задание принял	
		подпись	дата	подпись	дата

Руководитель _____ *Задание получил* _____ 21.06.2017
подпись дата подпись студента дата

9. Выпускная квалификационная работа и все материалы проанализированы.
Считаю возможным допустить Борисова Алексея Алексеевича к защите выпускной квалификационной работы в государственной экзаменационной комиссии.

Руководитель _____ _____
подпись дата

10. Допустить Борисова Алексея Алексеевича к защите выпускной квалификационной работы в государственной экзаменационной комиссии (протокол заседания кафедры от 14.06.2017 №12)

Заведующий кафедрой _____ _____
подпись дата