Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение высшего образования
«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

ЛЕСОСИБИРСКИЙ ПЕДАГОГИЧЕСКИЙ ИНСТИТУТ –
филиал Сибирского федерального университета

Высшей математики, информатики, экономики и естествознания
кафедра

УТВЕРЖДАЮ
Заведующий кафедрой
Храмова Л.Н. Храмова
подпись инициалы, фамилия
«13» 06 2023 г.

БАКАЛАВРСКАЯ РАБОТА

09.03.02 Информационные системы и технологии
код-наименование направления

РАЗРАБОТКА КОМПЬЮТЕРНОЙ ИГРЫ В ЖАНРЕ «АРКАДА» НА ПЛАТФОРМЕ UNITY

Руководитель 13.06.23 доцент, канд. пед. наук С. С. Ахтамова
подпись, дата должность, ученая степень инициалы, фамилия

Выпускник 13.06.2023 Н.К. Шинтяпин
подпись, дата инициалы, фамилия

Нормоконтролер 13.06.2023 Е.В. Киргизова
подпись, дата инициалы, фамилия

Лесосибирск 2023

РЕФЕРАТ

Выпускная квалификационная работа по теме «Разработка компьютерной игры в жанре «аркада» на платформе Unity» содержит 72 страниц текстового документа, 40 использованных источников, 2 таблицы, 20 рисунков, 13 приложений.

КОМПЬЮТЕРНЫЕ ИГРЫ, ЖАНР АРКАДА, ПЛАТФОРМА «UNITY», ГЕЙМДИЗАЙН, РАЗРАБОТКА ИГРЫ.

Повсеместное распространение и доступность игр и потоковых сервисов делают видеоигры мощным инструментом влияния на аудиторию. Продвигая политические лозунги и идеи через игроков и игровой контент, меняется мировоззрение подрастающего поколения, наиболее активно вовлеченного в игры.

Цель исследования – состоит в разработке компьютерной игры в жанре «аркада» на платформе Unity.

Объект исследования – компьютерные игры.

Предмет исследования – компьютерная игра в жанре «аркада».

В работе использовались следующие методы: анализ литературных источников, наблюдение, эксперимент, сравнительный анализ.

Основные задачи исследования:

– проанализировать продвижение игровой компьютерной индустрии в России;

– изучить и выбрать программные средства для разработки компьютерной игры на языке программирования C# в Unity;

– разработать сценарий, элементы компьютерной игры;

– спланировать оригинал игры.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Компьютерные игры появились относительно недавно, примерно 30 лет тому назад. И несмотря на это, они колоссально развиваются и приносят ежегодно миллиардные доходы. Не сложно понять, почему так быстро развилась компьютерная индустрия: всему поспособствовал большой выбор компьютерных технологий и появление сети Интернет. В результате – обычные развлечения стали не так популярны, как компьютерные. Чтобы поиграть, достаточно иметь компьютер, ноутбук или игровую приставку, а также лицензионную игру или ее копию, скачанную с интернета. Для пользователя не обязательно иметь знания, чтобы выбрать подходящею игру, нужно всего лишь понимать, какой жанр игры более интересен.

Примем тот факт, что в последнее время игры уже перестали быть как просто средство развлечения для свободного время провождения и отвлечения. Так, например есть игры, которые используют в разных учреждениях для обучения, такие игры называются развивающими, для практики симуляторами. Развивающие игры помогают обучать ребенка чему либо, а симуляторы обучают специалистов в разных областях: от пилотов самолета и до самой простой специальности.

Анализ состояния научной разработанности проблемы методического аспекта создания игры, свидетельствует о том, что ещё выдающиеся педагоги позитивно относились умениям проектирования игр как методу обучения и самосовершенствования. Учёными были созданы концепции, акцентировавшие внимание на дидактических возможностях игр для решения задач обучения и познания.

Цель исследования – разработать компьютерную игру в жанре «аркада» с помощью игрового движка Unity.

Объект исследования – компьютерные игры.

Предмет исследования – компьютерная игра в жанре «аркада».

Основные задачи исследования:

– проанализировать продвижение игровой компьютерной индустрии в России;

– изучить и выбрать программные средства для разработки компьютерной игры на языке программирования C# в Unity;

– разработать сценарий, элементы компьютерной игры;

– спланировать оригинал игры.

Методы исследования:

1. Обзор литературы и анализ существующих игр в жанре аркада.

2. Проектирование игровых механик и игрового процесса.

3. Разработка архитектуры игры.

4. Реализация игровой механики и геймплея.

5. Тестирование и оптимизация игры.

Этапы исследования:

1 этап (сентябрь 2022 г.) – изучение научных статей и книг, посвященные разработке игр и жанру аркада, а также произведен обзор существующих игр жанра аркада на особенности механики.

2 этап (ноябрь 2022 г.) – изучение процесса разработки игровой механики, системы и геймплея.

3 этап (декабрь 2022 г.) – определение основных компонентов игры и их взаимосвязей.

4 этап (февраль 2023 г.) – использование среды разработки Unity для создания игровых сцен, уровней и объектов, а также написание скриптов на C# для управления поведением игровых объектов.

5 этап (май 2023 г.) – выявление и исправление ошибок в игровой механике и геймплея.

1 Анализ предметной области

1.1 Исследование понятия и история появления компьютерных игр

Компьютерная игра – компьютерная программа, предназначенная для организации геймплея.

Видеоигры появились в середине XX века, но в те времена были никому не нужные и не стали популярны. А причиной всему этому стало дороговизна производства, для того периода времени.

Самой первой игрой можно назвать SpaceWar. Была создана Стивом Расселом и его помощником Мартином Греца в феврале 1962 года. На её разработку ушло 200 часов. Суть игры заключалась в том, что было два космических корабля которыми нужно управлять в невесомости рядом с гравитационным колодцем звезды и собирать ресурсы. Если ничего не предпринимать, то корабли сталкивались со звездой или друг с другом. Если игрок разбивался на космическом корабле игра считалась проигранной и законченной [1].

В следом наступал «Золотой век» видеоигровой индустрии, который длился с 1978 года по 1983 год. В эти года вышли игры, которые служили дальнейшим формирование современных игр. Именно этот век стал началом индустрии с многомиллиардной прибылью.

В наше время видеоигровая индустрия перестала быть только развлечением. Видеоигры стали использовать как обучение в некоторых странах, таких как Швеция, где игра Minecraft является предметом по графическому дизайну, а в Южной Корее Starcraft предметом по обучению будущих менеджеров. Так же не забудем упомянуть киберспорт. Этот спорт в России появился не так давно, но уже официально признан спортом. Награды за победу в киберспорте доходят до миллионов долларов, а также собирает до миллиона зрителей на разных платформах. Этот вид спорта появился в 1994 году вместе с вышедшей игрой Doom II, в которую можно было играть по локальной сети. Первые соревнования прошли в 1996 году, а в 2000 году

прошел первый чемпионат мира. Есть игры, которые не относятся к киберспорту, такие как на проверку умения. Так же соревнования проводятся не только на персональных компьютерах, а еще и на консолях особенно это распространено в Мексике [4].

1.2 Анализ жанров компьютерных игр

В наше время существует много разных игровых жанров для компьютера. Но одним из самых востребованных жанров по статистике считается классический шутер – это такой жанр игры, где игрок играет от первого лица или же от третьего лица, за главного персонажа с оружием в руках и ликвидирует всех подряд, выполняя тем самым сюжетные задания [28].

Сами игры квалифицируются по нескольким основным признакам:

1) жанр;

2) количество игроков;

3) платформа;

4) представление об игре.

Поговорим так же какие жанры игр существуют и что в их примечательно.

Файтинг – в этом жанре нам предстоит сражение на маленькой игровой арене, где происходит рукопашная схватка между двумя игровыми персонажами. Можно играть как за одного персонажа против искусственного интеллекта, так и играть против другого реального игрока по сети [31].

Аркады – один из самых первых жанров игры который появился, представляет собой мини-игру с небольшим сюжетом и с легким игровым процессом, тем самым способствует к интересу геймплея.

Приключения – чаще всего это большие открытые миры, которые надо исследовать, собирать ресурсы, проходить квестовые задания общаясь с другими персонажами, а также решать головоломки [37].

Хоррор – в этом жанре разработчики постарались сделать чтобы игрок испугаться почти в каждом моменте, начиная от устрашающей музыки и атмосферы и до выскакивания разных скримеров из не ожиданных мест.

Как мы уже увидели игры бывают разные по жанрам, но все же в эти игры объединяет одно, то что можно играть как в однопользовательском режиме, так и многопользовательском [8].

Так же разделим по визуальному предоставлению игры:

1. Тестовые – малое количество графических представлений, рассказывание сюжета и диалогов происходит с помощью текста.
2. 2D – элементы отрисованы в двухмерном виде.
3. 3D – элементы отрисованы в трехмерном виде.
4. По типу платформы:
5. Персональные компьютеры.
6. Игровые приставки.
7. Мобильные телефоны.

1.3 Алгоритм реализации проекта

Алгоритм реализации проекта – это последовательность шагов и действий, которые необходимо выполнить для создания и разработки игрового проекта в среде Unity с использованием двумерной графики. Этот алгоритм включает в себя настройку проекта, создание сцены, управление персонажем, логику игры, а также другие необходимые действия, связанные с разработкой проекта в Unity 2D [3].

Шаги для реализации проекта:

а) Определить жанр игры и ее основные механики:

1) Выбрать жанр игры

2) Определить основные механики игры (движение, сбор предметов, битвы с боссами, мультиплеер и т. д.).

б) Создать концепт игры:

1) Разработать идеи для игры и визуализировать ее в виде концепта (например, описание игровой механики и визуальный стиль).

2) Определить целевую аудиторию и рынок игр.

в) Определить необходимые ресурсы:

1) Определить необходимые ресурсы для создания игры (например, дизайнеров, программистов, художников и т. д.).

2) Рассмотреть возможность использования готовых ассетов для ускорения разработки игры.

г) Создать дизайн и прототип игры:

1) Создать дизайн игры, включая интерфейс, меню, уровни и т. д.

2) Создать прототип игры, чтобы протестировать механику игры.

д) Создать игровые объекты и анимации:

1) Создать игровые объекты, такие как персонажи, враги, оружие и т. д.

2) Создать анимации для игровых объектов.

е) Написать код игры:

1) Написать код игры на языке программирования C# в Unity.

2) Реализовать основные механики игры, такие как управление персонажем, взаимодействие с объектами и т. д.

ж) Тестирование игры:

1) Протестировать игру на наличие ошибок и недочетов.

2) Провести тестирование игры на различных устройствах и операционных системах.

и) Оптимизация игры:

1) Оптимизировать игру для улучшения производительности и устранения задержек.

2) Уменьшить размер игры для быстрой загрузки и экономии места на диске.

к) Развертывание игры:

1) Развернуть игру на платформах, которые вы выбрали для ее публикации (например, Steam, iOS, Android и т. д.).

2) Создать страницу игры в магазинах приложений и на социальных платформах для продвижения игры.

л) Поддержка и обновление игры:

1) Обеспечить поддержку игры после ее выпуска, реагируя на обратную связь от пользователей и исправляя ошибки.

2) Проводить регулярные обновления игры с добавлением новых уровней, механик и функций, чтобы сохранять интерес игроков к игре.

3) Организовать мониторинг работы игры и быстро реагировать на возникающие проблемы, чтобы минимизировать отрицательные последствия для пользователей.

4) Разработать систему обратной связи с пользователями игры и учесть их предложения и замечания в процессе ее доработки.

5) Обновлять игру в соответствии с требованиями платформы Unity и актуальными стандартами безопасности, чтобы гарантировать ее стабильность и безопасность для пользователей.

6) Обеспечивать совместимость игры с новыми устройствами и платформами, а также с новыми версиями операционных систем, чтобы расширять ее аудиторию и поддерживать конкурентоспособность на рынке.

м) Монетизация игры:

1) Разработать стратегию монетизации игры, которая может включать в себя различные варианты, такие как покупки внутри игры, рекламу, подписки и т.д.

2) Изучить рынок и конкурентов, чтобы определить оптимальную цену для покупок в игре и установить привлекательные условия для пользователей.

3) Интегрировать монетизационные механизмы в игру и убедиться, что они не нарушают игровой процесс и не ущемляют интерес пользователей.

4) Проводить анализ эффективности монетизации и оптимизировать ее, основываясь на полученных результатах и обратной связи от пользователей

5) Предоставить пользователю возможность выбора между покупками в игре и бесплатной игрой, чтобы увеличить число установок и сохранить интерес пользователей.

6) Следить за изменениями в законодательстве и стандартах монетизации игр, чтобы соответствовать им и избежать возможных проблем в будущем.

н) Поддержка и обновление игры:

1) Обеспечить поддержку игры после ее выпуска.

2) Проводить регулярные обновления игры с добавлением новых уровней, механик и функций, чтобы сохранять интерес игроков к игре.

п) Анализ и улучшение проекта:

1) Провести анализ продаж игры, ее рейтинга, обратной связи от пользователей и других метрик, чтобы понять, как улучшить игру и увеличить ее доходность.

2) Реализовать улучшения в игре на основе анализа метрик и обратной связи от пользователей.

р) Продвижение игры:

1) Разработать стратегию продвижения игры, включая использование социальных сетей, блогов, YouTube-каналов и других каналов, чтобы привлечь внимание к игре.

2) Рассмотреть возможность участия в игровых конференциях и выставках для продвижения игры.

с) Сопровождение игры:

1) Оказывать техническую поддержку и консультации пользователям игры, решать возникающие проблемы и отвечать на вопросы.

2) Поддерживать активность и участие в сообществах пользователей игры, чтобы увеличить ее популярность и поддержку.

1.4 Инструменты и приложения для создания игр

Unity – один из самых популярных инструментов для разработки игр, который используется многими игровыми студиями и независимыми разработчиками. Он обладает широкими возможностями, которые позволяют создавать игры для различных платформ, включая ПК, мобильные устройства, консоли и виртуальную реальность. Но какие инструменты и приложения могут помочь в разработке игр в Unity? Рассмотрим несколько из них [12].

1. Visual Studio

Visual Studio – это интегрированная среда разработки (IDE), которая может использоваться для разработки игр в Unity. Она предоставляет разработчикам мощные инструменты для написания кода, включая подсветку синтаксиса, автозаполнение и отладку. Кроме того, Visual Studio может быть интегрирован с Unity, что позволяет разработчикам легко переключаться между двумя средами и облегчает работу с проектом [33].

2. Unity Hub

Unity Hub – это приложение, которое позволяет управлять установленными версиями Unity и проектами. Оно также предоставляет доступ к Unity Learn, магазину активов Unity Asset Store и сообществу Unity. С помощью Unity Hub можно быстро создавать новые проекты и легко переключаться между разными версиями Unity.

3. Blender

Blender – это бесплатный инструмент для 3D-моделирования и анимации, который может использоваться для создания ассетов для игр в Unity. С помощью Blender можно создавать персонажей, объекты, анимации и многое другое. Кроме того, Blender поддерживает импорт и экспорт файлов в форматах, которые могут быть использованы в Unity.

4. Unity Recorder

Unity Recorder – это плагин для Unity, который позволяет записывать видео, звук и анимацию в процессе разработки игры. Это может быть полезно для создания презентаций и трейлеров игры, а также для отладки и тестирования.

5. ProBuilder

ProBuilder – это инструмент для создания геометрии прямо внутри Unity. С его помощью можно быстро создавать объекты, уровни и декорации, используя интуитивный интерфейс и инструменты. Кроме того, ProBuilder поддерживает импорт и экспорт файлов в форматах, которые могут быть использованы в других приложениях для 3D-моделирования [40].

6. Audacity

Audacity – это бесплатный аудио редактор, который может использоваться для создания и редактирования звуковых эффектов и музыки для игр в Unity. С помощью Audacity можно записывать звуки, обрезать, смешивать и добавлять эффекты к аудио файлам.

7. Git

Git – это система контроля версий, которая может использоваться для управления изменениями в проекте Unity. Она позволяет разработчикам сохранять разные версии проекта, работать с другими разработчиками и возвращаться к предыдущим версиям проекта в случае необходимости.

8. Unity Test Runner

Unity Test Runner – это инструмент для автоматического тестирования игровых сцен в Unity. Он может использоваться для проверки функциональности игры, а также для выявления ошибок и проблем в коде. С помощью Unity Test Runner можно создавать и запускать тесты автоматически или вручную [35].

9. Unity Analytics

Unity Analytics – это инструмент для сбора и анализа данных об использовании игр в Unity. Он может использоваться для мониторинга поведения игроков, выявления проблем и оптимизации игрового процесса.

Unity Analytics может также помочь разработчикам принимать решения на основе данных, связанных с игрой.

10. Unity Profiler

Unity Profiler – это инструмент для анализа производительности игры в Unity. Он может использоваться для выявления узких мест в игровом процессе, оптимизации производительности и устранения проблем. Unity Profiler позволяет отслеживать использование памяти, CPU и GPU, а также другие параметры производительности [18].

В заключение, Unity предоставляет широкие возможности для создания игр, и использование инструментов и приложений, которые помогают упростить и ускорить процесс разработки, и может быть очень полезным. Однако, это лишь некоторые из возможных инструментов и разработчики могут использовать другие инструменты в зависимости от своих потребностей.

1.5 Анализ существующих разработок

Сегодня разработка компьютерных игр на платформе Unity является одним из наиболее популярных направлений в индустрии игр. В частности, жанр аркады в Unity является одним из наиболее распространенных, поскольку он позволяет создавать простые и захватывающие игры, которые могут привлечь широкую аудиторию. В этом пункте мы рассмотрим существующие разработки в жанре аркада на платформе Unity и их особенности [22].

Первой игрой, которую стоит упомянуть, является «Geometry Dash», созданная разработчиком Robert Topala. Эта игра была выпущена в 2013 году и быстро стала очень популярной благодаря своей простой, но захватывающей игровой механике и музыкальному сопровождению. «Geometry Dash» позволяет игрокам управлять главным героем, преодолевая препятствия на пути к финишу, используя прыжки и другие движения. Особенностью этой игры является возможность создания собственных уровней, что делает ее еще более интересной и разнообразной для игроков [27].

Еще одной популярной игрой в жанре аркада на платформе Unity является «Super Hexagon» от разработчика Terry Cavanagh. Эта игра была выпущена в 2012 году и тоже получила множество положительных отзывов благодаря своей простой, но захватывающей игровой механике и уникальной визуальной стилистике. «Super Hexagon» предлагает игрокам управлять маленьким треугольником, избегая препятствий, которые появляются на пути [25].

Другой известной игрой в жанре аркада на платформе Unity является «Fruit Ninja» от Halfbrick Studios. Эта игра была выпущена в 2010 году и стала очень популярной благодаря своей простой, но увлекательной игровой механике и красочной визуальной стилистике. «Fruit Ninja» предлагает игрокам использовать свой палец, чтобы разрезать фрукты, избегая бомб и других препятствий [6].

Еще одной популярной игрой в жанре аркада на платформе Unity является «Angry Birds» от студии Rovio Entertainment. Эта игра была выпущена в 2009 году и стала одной из самых продаваемых мобильных игр всех времен благодаря своей забавной игровой механике и зрелищной визуальной стилистике. «Angry Birds» позволяет игрокам использовать птиц, чтобы разрушить строения, захваченные злыми свиньями [7].

Существуют также игры в жанре аркада на платформе Unity, которые являются продолжениями известных франшиз. Например, «Sonic the Hedgehog» и «Pac-Man» были перенесены на платформу Unity и стали доступны для игры на различных устройствах [10]. Эти игры сохраняют оригинальную игровую механику и визуальную стилистику, но с добавлением новых функций и возможностей, доступных благодаря использованию платформы Unity.

Жанр аркада на платформе Unity продолжает развиваться, и с каждым годом появляются новые увлекательные игры, которые привлекают все больше игроков. Разработчики постоянно ищут новые способы создания уникальных и захватывающих игр в этом жанре, используя возможности, предоставляемые платформой Unity [13].

2 Разработка проекта

2.1 Отбор целевой аудитории

Отбор целевой аудитории является важным этапом разработки компьютерной игры в жанре аркада на платформе Unity. Целевая аудитория – это группа людей, которые будут наиболее заинтересованы в игре, ее геймплее и функциональности. Определение целевой аудитории поможет разработчикам игры более точно сфокусироваться на нужных характеристиках и возможностях игры, что повысит ее популярность среди целевой аудитории и общей аудитории игроков.

Для определения целевой аудитории необходимо провести исследование рынка и анализ конкурентов в жанре аркада на платформе Unity. Нужно изучить интересы, предпочтения и поведение потенциальных пользователей, а также узнать о их устройствах и операционных системах. Например, целевая аудитория может быть молодежью в возрасте от 16 до 25 лет, любящих играть в аркады на смартфонах и планшетах с операционной системой iOS и Android [9].

Также важно учитывать особенности жанра аркада и платформы Unity при выборе целевой аудитории. Например, жанр аркада часто связан с быстрым и динамичным геймплеем, а платформа Unity позволяет создавать игры с различными уровнями сложности и графическими эффектами. Поэтому, целевая аудитория должна быть заинтересована в таких играх и иметь устройства с достаточной производительностью для запуска игры.

Определение целевой аудитории также поможет разработчикам игры создать наиболее эффективную маркетинговую стратегию, которая привлечет целевую аудиторию и повысит продажи игры. Например, можно использовать социальные сети и мобильную рекламу, чтобы достигнуть целевой аудитории [16].

В целом, определение целевой аудитории является важным этапом разработки компьютерной игры в жанре аркада на платформе Unity, который

поможет создать игру, которая будет наиболее интересной для целевой аудитории.

2.2 Задачи проекта

В данном параграфе рассмотрим задачи проекта.

1. Актуальность, цель и назначение

Актуальность проекта по разработке компьютерной игры в жанре аркада на платформе Unity заключается в том, что игры данного жанра остаются популярными среди широкой аудитории игроков, а платформа Unity является одной из наиболее популярных и эффективных платформ для создания игр. Разработка новой игры в жанре аркада на платформе Unity имеет потенциал привлечь большое количество игроков и стать успешным коммерческим продуктом [15].

Цель проекта – создание качественной и увлекательной компьютерной игры в жанре аркада на платформе Unity, которая будет успешно продаваться и популярна среди широкой аудитории игроков.

Назначение проекта – разработка и выпуск игры в жанре аркада на платформе Unity, которая предложит игрокам уникальный игровой опыт и будет конкурировать с другими играми на рынке.

2. Функционал проекта

Игровой мир – разработка уникального игрового мира с собственной историей, атмосферой и геймплеем. Включает в себя разработку игровых локаций, уровней, заданий и интерактивных объектов.

Игровые персонажи – создание уникальных персонажей, каждый из которых имеет свои особенности, навыки и способности. Включает в себя разработку моделей персонажей, анимаций, искусственного интеллекта и системы управления персонажами.

Игровая механика – разработка уникальной игровой механики, которая обеспечивает баланс между сложностью и увлекательностью игры. Включает в себя разработку системы управления, системы взаимодействия персонажей с окружающей средой, системы боя и других элементов геймплея.

Звуковое сопровождение – создание уникальной звуковой атмосферы, которая поможет погрузить игроков в игровой мир. Включает в себя разработку музыкального сопровождения, звуковых эффектов и диалоговых систем [21].

3. Сценарий

Игрок управляет персонажем, который должен победить различных противников и преодолеть препятствия. Игра состоит из одного мира, который имеет свою тематику и особенность.

Игрок начинает игру, где его цель – уничтожить всех врагов и победить босса. Враги разбросаны по всей карте, и игрок должен собирать аптечки для пополнения своих жизней. Аптечки представлены в виде сердец, которые находятся на разных местах карты. Если персонаж сталкивается с противником, он теряет определенное количество жизней. Если жизней не осталось, игрок проигрывает и начинает игру заново. В конце карты надо будет сразиться с одним боссом. Игра будет считаться законченной если игрок уничтожит врагов, по которым можно нанести урон, а также уничтожив босса.

4. Характеристика оборудования разработки

Для разработки игры на платформе Unity 2D и для написания кодов в Microsoft Visual Studio 2019, использовался персональный компьютер со следующими параметрами:

1. Процессор: Intel Core i5-9400 OEM, Базовая частота – 2900 МГц.
2. Видеокарта: MSI GeForce GTX 1660 SUPER VENTUS XS OC 6Гб.
3. ОЗУ: 16 ГБ.
4. ОС: Windows 10.

2.3 Реализация проекта

1. Этап разработки графического оформления

Проект разработки игры включал в себя два основных этапа: создание графического оформления и написание кода игры. Эти этапы были выполнены параллельно, поскольку написание кода не требовало наличия готовых анимаций и других элементов графического оформления. Кроме того, использование стандартных примитивов Unity для проверки работоспособности кода ускорило процесс разработки.

Важными понятиями в графическом оформлении были тайлы и спрайты. Тайлы представляют собой небольшие повторяющиеся элементы, которые применяются в создании крупных изображений, таких как уровней для 2D-игр. Во-вторых, они являются растровыми изображениями, которые служат для анимации объектов.

Все материалы, найденные в Интернете, используются для создания визуального дизайна. Всё это включает в себя анимацию главного героя и врага уровня, а также фоновые изображения, отображаемые за уровнем. Изображение фона и атласы спрайтов представлены на рисунках 1-5.

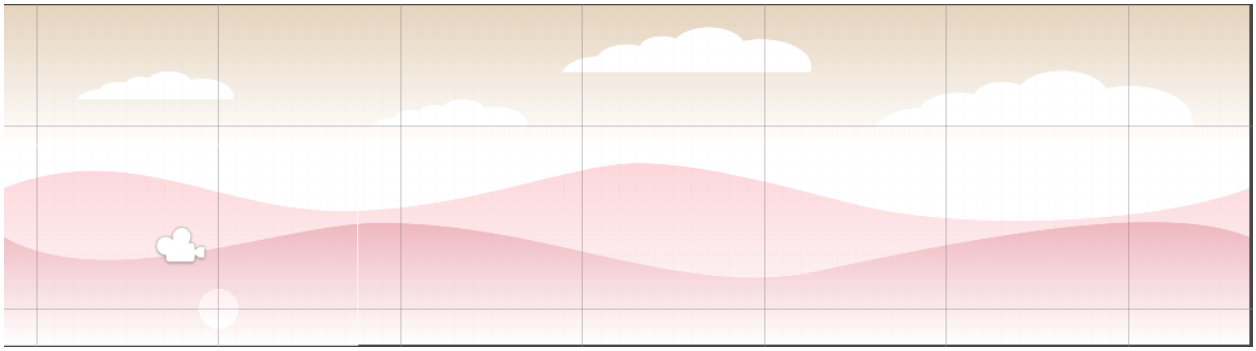

Рисунок 1 – Фоновое изображение

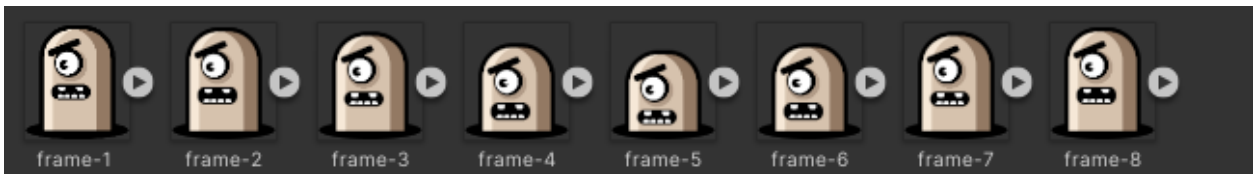

Рисунок 2 – Атлас спрайтов для врага

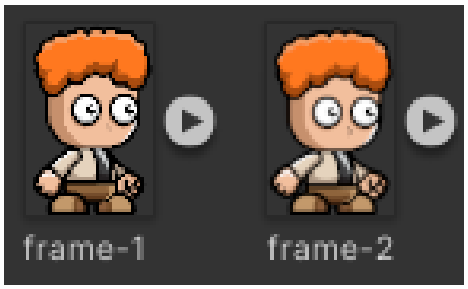

Рисунок 3 – Атлас спрайта главного героя в покое

Рисунок 4 – Атлас спрайта главного героя в прыжке

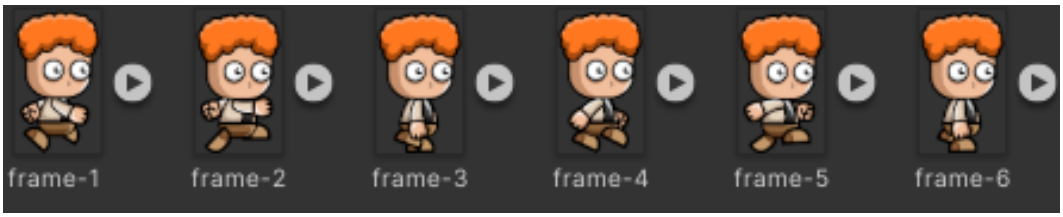

Рисунок 5 – Атлас спрайта главного героя при беге

Для создания тайловой графики были использованы тайлы из бесплатного набора, доступного в открытом доступе.

2. Этап разработки игры

а) Анализ и планирование:

1) Определение целей и концепции игры.

2) Исследование рынка и анализ конкурентов.

3) Определение требований к игре, включая геймплей, графику, звук и функциональность.

4) Создание документа со спецификацией требований.

б) Дизайн игры:

1) Создание игровых механик, правил и систем.

2) Разработка игрового мира, уровней и интерфейса пользователя.

3) Проектирование персонажей, объектов и арт-активов.

4) Создание концепт-артов и скетчей игровых элементов.

в) Разработка:

1) Выбор игрового движка и инструментов разработки.

2) Создание игровых объектов, персонажей и анимаций.

3) Написание игрового кода, включая управление персонажами, искусственный интеллект и игровую логику.

4) Интеграция графики, звука и спецэффектов.

5) Тестирование игры и исправление ошибок.

г) Оптимизация и доработка:

1) Улучшение производительности игры и оптимизация кода.

2) Доработка игровых механик и систем на основе обратной связи тестирования.

3) Балансировка уровней и сложности игры.

4) Добавление дополнительных функций и контента.

д) Тестирование и отладка:

1) Проведение систематического тестирования игры на различных платформах и устройствах.

2) Выявление и исправление ошибок, глюков и несоответствий требованиям.

3) Тестирование игры на проходимость и играбельность.

е) Релиз и монетизация:

1) Подготовка игры к релизу, включая создание маркетинговых материалов и промо-стратегию.

2) Размещение игры в цифровых магазинах и платформах.

3) Определение моделей монетизации, таких как платные загрузки, реклама или внутриигровые покупки.

4) Сопровождение игры после релиза, включая выпуск обновлений и поддержку сообщества.

ж) Оценка и анализ:

1) Сбор обратной связи от игроков и рецензентов.

2) Анализ успеха игры на ос.

Рассмотрим, как создавалась поэтапно игра. Для этого откроем Unity в котором увидим главное окно проекта представлено на рисунке 6, которое содержит различные окна для работы с сценами, анимациями и играми. Окно сцены предназначено для создания и редактирования игровых уровней или сцен. Здесь вы можете составлять композицию уровня, размещать и настраивать объекты, а также определять их расположение и взаимодействие.

Окно анимации предоставляет инструменты для создания, редактирования и управления анимациями объектов в двухмерных играх. В нем вы найдете удобную рабочую среду с функциями ключевых кадров, таймлайна, кривых анимации и другими инструментами, которые помогут создавать плавные и динамичные анимации. Окно игры предоставляет возможность предварительного просмотра игры в режиме реального времени с точки зрения игрока. Оно имитирует игровую среду, в которой вы можете проверить и оценить визуальные эффекты, геймплей, взаимодействия объектов и другие аспекты игрового процесса [39].

Так же есть окно Иерархии оно используется для отображения и управления иерархией объектов в текущей сцене. Оно представляет собой иерархическую структуру всех объектов, используемых в игре, и позволяет вам легко настраивать и взаимодействовать с каждым объектом.

Основные функции окна Иерархии:

1. Организация объектов: Окно Иерархии позволяет вам управлять порядком и организацией объектов в сцене. Вы можете создавать, удалять и переименовывать объекты, а также изменять их иерархическое положение внутри других объектов.

2. Взаимодействие с объектами: Окно Иерархии позволяет вам выбирать и выделять объекты для последующей работы с ними. Вы можете кликнуть на объект в окне Иерархии, чтобы выбрать его и изменять его свойства в окне Инспектора.

3. Работа с компонентами: Окно Иерархии позволяет вам добавлять, удалять и настраивать компоненты объектов. Компоненты представляют функциональные части объектов, такие как коллайдеры, скрипты, рендереры и другие, и окно Иерархии предоставляет доступ к этим компонентам для их настройки.

4. Работа с родительскими и дочерними объектами: Окно Иерархии позволяет вам создавать и управлять иерархическими связями между объектами. Вы можете делать объекты родительскими или дочерними по отношению к другим объектам, что позволяет организовывать объекты в группы и управлять их поведением и взаимодействием.

5. Активация и деактивация объектов: Окно Иерархии позволяет вам активировать и деактивировать объекты в сцене. Вы можете временно скрывать или отключать объекты, чтобы контролировать их видимость или поведение в игре.

6. Навигация по сцене: Окно Иерархии также служит для удобной навигации по сцене. Вы можете быстро находить нужные объекты, перемещаться по иерархии и настраивать их свойства без необходимости искать их визуально на сцене.

Внизу интерфейса вы найдете три панели: «Проект», «Аниматор» и «Консоль». В панели «Проект» отображаются все ресурсы, которые вы добавили в игру. Здесь вы найдете объекты, текстуры, аудиофайлы, скрипты,

анимации и другие элементы, которые могут быть использованы в вашем проекте.

Панель «Аниматор» предоставляет вам возможность создавать и настраивать анимации объектов. Вы можете устанавливать связи между различными анимациями и определять правила их перехода. Это поможет вам создать плавные и качественные анимации для вашей игры.

Панель «Консоль» предоставляет важную информацию о возникающих ошибках, предупреждениях и исключениях в процессе работы вашей игры. Она поможет вам быстро обнаруживать и исправлять проблемы, которые могут возникнуть в процессе разработки. Благодаря этой панели вы сможете эффективно отслеживать и решать проблемы, чтобы ваш проект был безупречным.

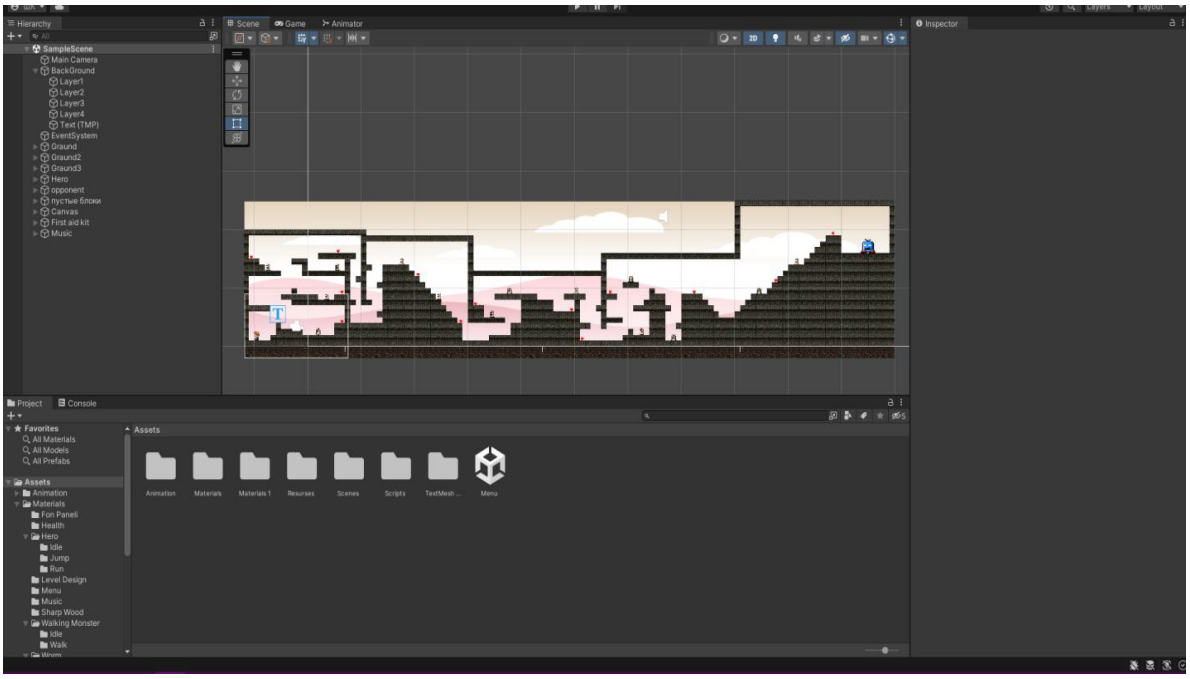

Рисунок 6 – Окно проекта Unity

Для начала важно добавить платформу в сцену игры, чтобы она служила поверхностью для передвижения героя и врагов. Без платформы игрок будет неконтролируемо падать в пространстве.

Для этого выполним следующие шаги:

а) В окне проекта найдем спрайт платформы, который мы создали ранее, и просто перетащим его в окно проекта. Теперь спрайт платформы будет отображаться среди всех объектов, добавленных в игру.

б) Далее, добавим платформу в текущую сцену. У нас есть два способа сделать это:

1) Просто перетащим спрайт платформы непосредственно из окна проекта в окно сцены, разместив его там, где хотим видеть платформу.

2) Создадим пустой объект в окне иерархии, а затем добавим спрайт платформы к этому объекту. Для этого создадим пустой объект в окне иерархии и перетащим спрайт платформы на него. Таким образом, спрайт платформы станет дочерним объектом пустого объекта.

После выполнения этих шагов платформа будет добавлена в сцену и готова к использованию в качестве поверхности для передвижения героя и врагов изображены на рисунке 7.

Далее, необходимо добавить коллайдер к платформе. Коллайдер необходим для определения границ объекта и управления их взаимодействием с другими объектами в игре. В Unity есть два типа коллайдеров: для трехмерных и для двумерных объектов. Так как наша игра двумерная, нам понадобится добавить двумерный коллайдер.

Чтобы добавить коллайдер к объекту, следуйте этим шагам:

1. Выделите объект в окне Иерархии, к которому вы хотите добавить коллайдер.

2. В окне Инспектора найдите раздел «Добавить компонент» или «Add Component» и раскройте его.

3. В поисковой строке введите «Collider2D» и выберите соответствующий компонент двумерного коллайдера. Нажмите на него, чтобы добавить его к объекту.

4. После добавления коллайдера в окне Инспектора вы сможете настроить его параметры, такие как форма, размеры и другие свойства коллайдера.

5. Если вам необходимо точно настроить размеры коллайдера, вы можете воспользоваться кнопкой «Edit Collider» (или «Редактировать коллайдер»), чтобы открыть редактор коллайдера и вручную изменить его форму и размеры.

После выполнения этих шагов коллайдер будет успешно добавлен к объекту, и движок Unity будет учитывать его границы и взаимодействие с другими объектами в игре.

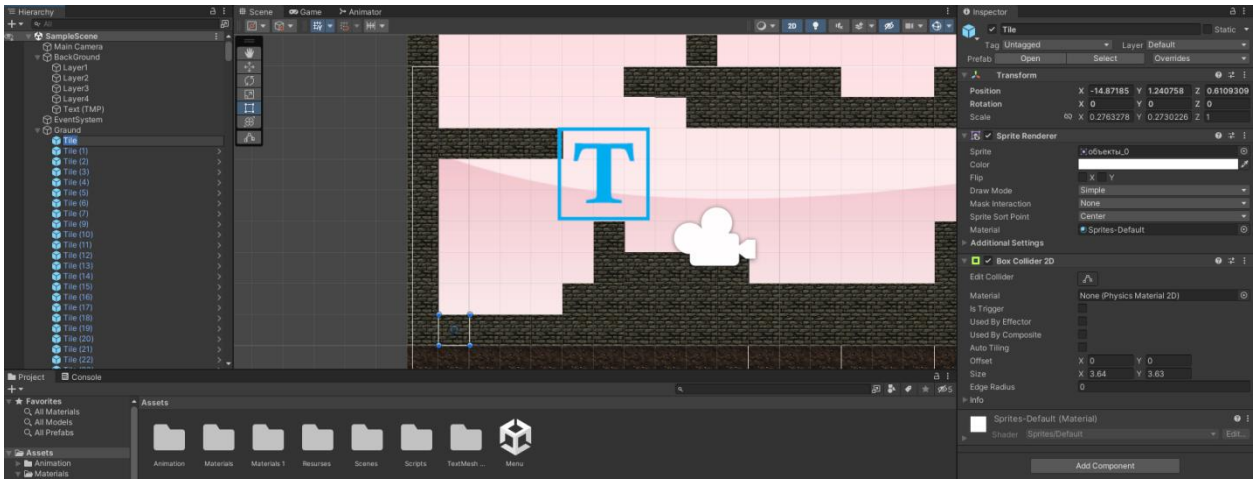

Рисунок 7 – Платформа в окне сцены и её свойства

В Unity есть возможность сохранять объекты со всеми их свойствами и скриптами для повторного использования на сцене. Этот функционал называется «префабами». Чтобы создать префаб, следуйте этим шагам:

1. Выделите объект, который вы хотите сделать префабом, в окне Иерархии.

2. Перетащите выделенный объект из окна Иерархии в окно Проекта. Можно создать отдельную папку в окне Проекта для хранения префабов, чтобы легче было их находить и управлять ими.

Теперь объект станет префабом, который можно повторно использовать на сцене без необходимости создавать новый объект. При изменении префаба

все его экземпляры на сцене также будут обновлены с учетом внесенных изменений. Это удобно, когда нужно использовать одинаковые объекты с одинаковыми свойствами и поведением.

Рекомендуется создавать отдельную папку в окне Проекта для хранения ваших префабов. Таким образом, они будут хорошо организованы и не потеряются среди остальных элементов игры, что сделает управление префабами более удобным и эффективным.

Затем мы создаем главного героя игры, добавляя двумерный объект и применяя к нему спрайт. Для реализации героя нам необходимо использовать коллайдер и компонент Rigidbody 2D, чтобы настроить физическую модель объекта.

Для того чтобы главный герой мог двигаться и выполнять различные действия, мы создадим скрипт на языке C#. В Unity есть несколько способов создания скрипта, включая создание его в окне Проекта или на объекте в окне Инспектора. После создания скрипта, Microsoft Visual Studio автоматически откроется, и в коде уже будут подключены основные библиотеки, а также будет создан стандартный метод Update. Мы напишем код для реализации движения персонажа и применим его к нашему герою.

Вот как мы можем выполнить этот процесс:

1. Создайте новый скрипт в окне Проекта, щелкнув правой кнопкой мыши на нужной папке, выбрав «Create» (Создать), а затем «C# Script» (Скрипт C#). Дайте скрипту соответствующее имя, которое отражает его функциональность, например «PlayerMovement» (Движение Персонажа).

2. После создания скрипта, щелкните по нему дважды, и Microsoft Visual Studio откроется. В окне редактора вы увидите стандартный код, который уже содержит подключение основных библиотек и метод Update.

3. В методе Update вы можете написать код для движения персонажа. Например, вы можете использовать функции Input.GetAxis для получения ввода от игрока и изменять позицию персонажа в соответствии с этим вводом.

4. Когда вы закончите написание кода для движения персонажа, сохраните файл в Microsoft Visual Studio и закройте редактор.

5. Теперь вернитесь в Unity. Присоедините скрипт к главному герою, выбрав его в окне Иерархии или окне Инспектора, а затем перетащите скрипт на него в окне Проекта.

После применения скрипта к главному герою, он будет иметь функциональность, описанную в коде, и сможет двигаться в соответствии с вашими указаниями. Вы можете настраивать параметры и логику скрипта в окне Инспектора для достижения желаемого поведения героя в игре изображено на рисунке 8.

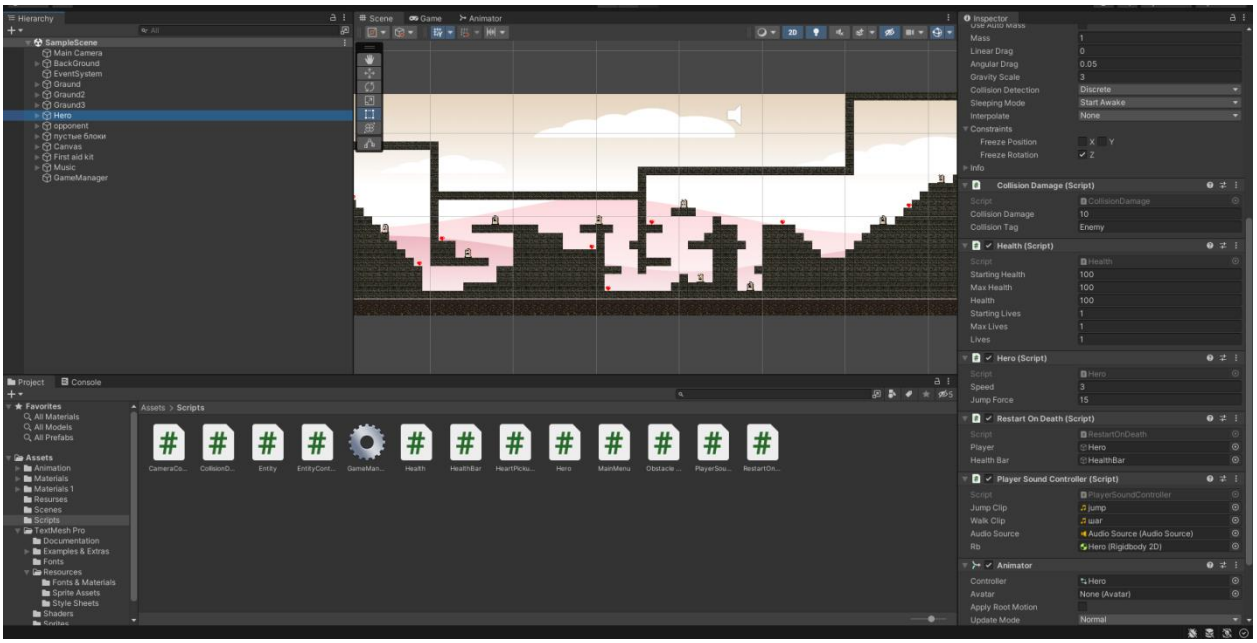

Рисунок 8 – Добавленный скрипт «Hero» передвижение для персонажа

Чтобы реализовать возможность прыжков героя, необходимо создать пустой дочерний объект у героя и назвать его «ground Check». Этот объект будет использоваться для определения того, находится ли персонаж на земле. Это необходимо, чтобы игрок не мог бесконечно прыгать вверх, удерживая клавишу прыжка. После этого мы сможем бегать по платформе из стороны в сторону, если запустим проект.

Далее мы создаем второй скрипт, который добавляем к камере и определяем условия, при которых камера будет следовать за героем по уровню. Если мы этого не сделаем, то не сможем следить за действиями персонажа.

Теперь мы можем создать врагов, которые создаются аналогичным образом, за исключением того, что игрок не управляет ими представлено на рисунке 9.

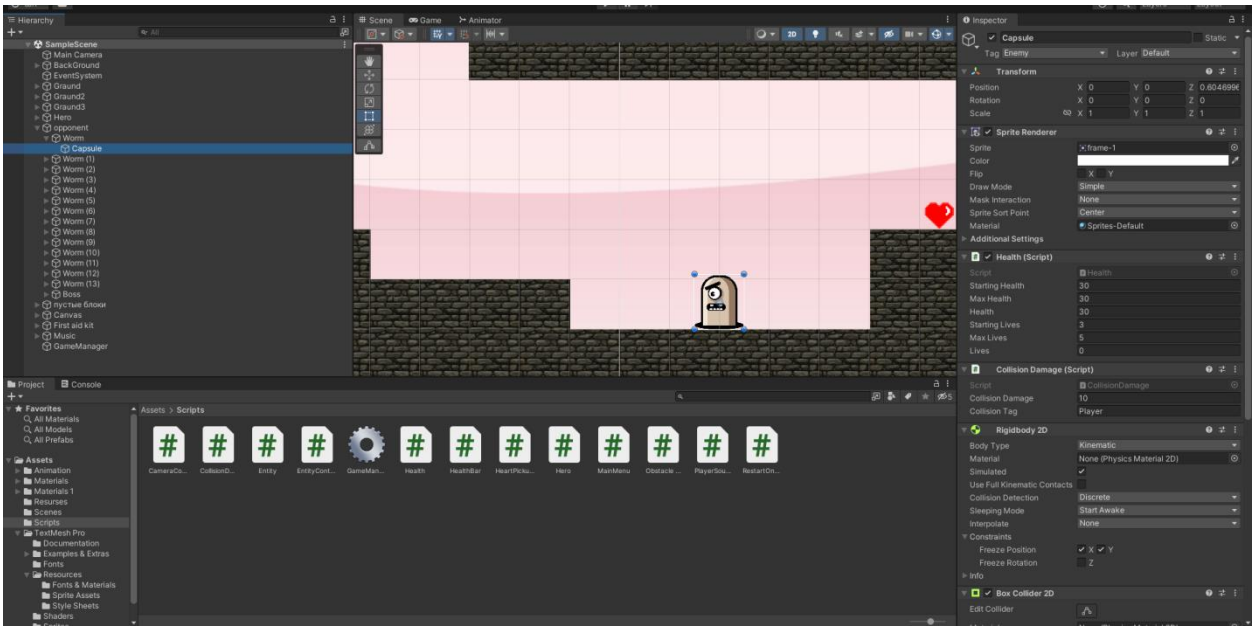

Рисунок 9 – Созданный враг на платформе

Затем создаём Animator Controller и добавляем его к персонажу, устанавливая правила и связи для анимаций объекта.

Для создания анимаций необходимо открыть окно Animation, выбрать нужный объект и создать новый клип анимации, затем перенести все кадры в окно в правильном порядке и настроить скорость анимации.

После создания всех клипов, переходим в окно Аниматора, где создаем связи и настраиваем условия переходов между анимациями, которые мы создали все это представлено на рисунке 10.

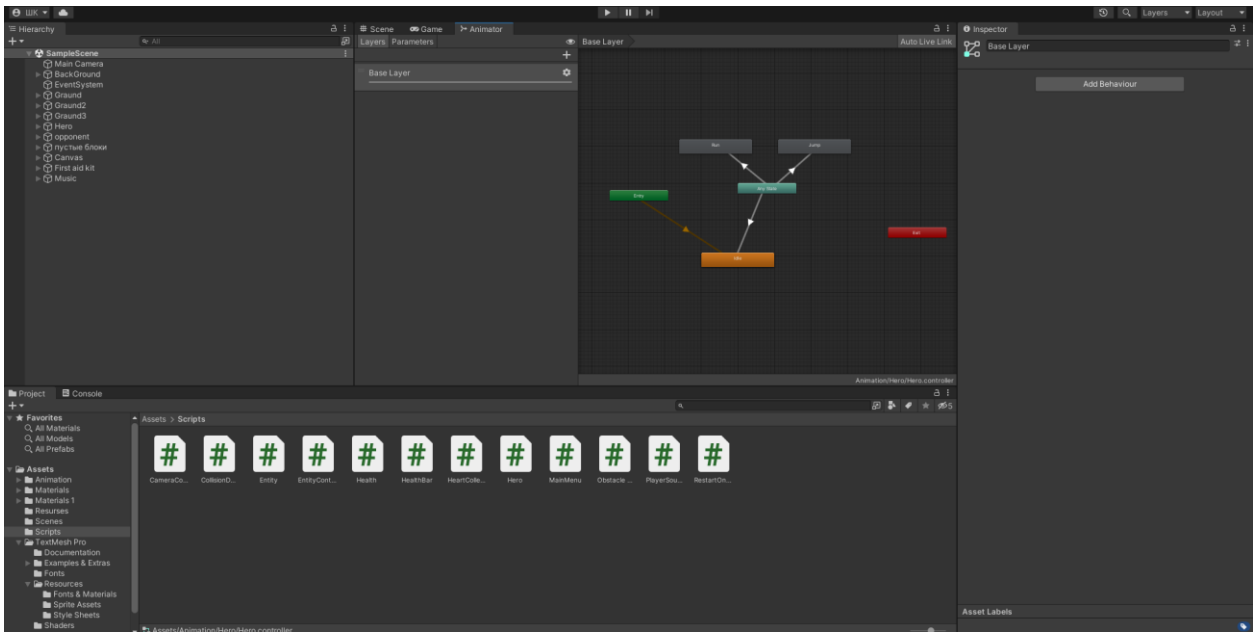

Рисунок 10 – Связи в Аниматоре

При запуске уровня игроку доступны различные действия: бегать в разные стороны, прыгать и т.д.

Далее необходимо создать уровень и разместить на нем врагов и объекты. Сначала нужно разработать примерный план уровня и определить расположение врагов и объектов [24].

Добавим в игру для главного персонажа и врагов количество жизней и количество того сколько они наносят урон друг другу. Для этого мы напишем два скрипта. Один будет отвечать за количество здоровья персонажа и врага, а другой за количество нанесенного урона. Так же укажем в коды, чтобы можно было указывать определенное количество урона и здоровья в самом Unity. Затем эти два кода привяжем к герою и врагу которые представлены на рисунках 11 – 12.

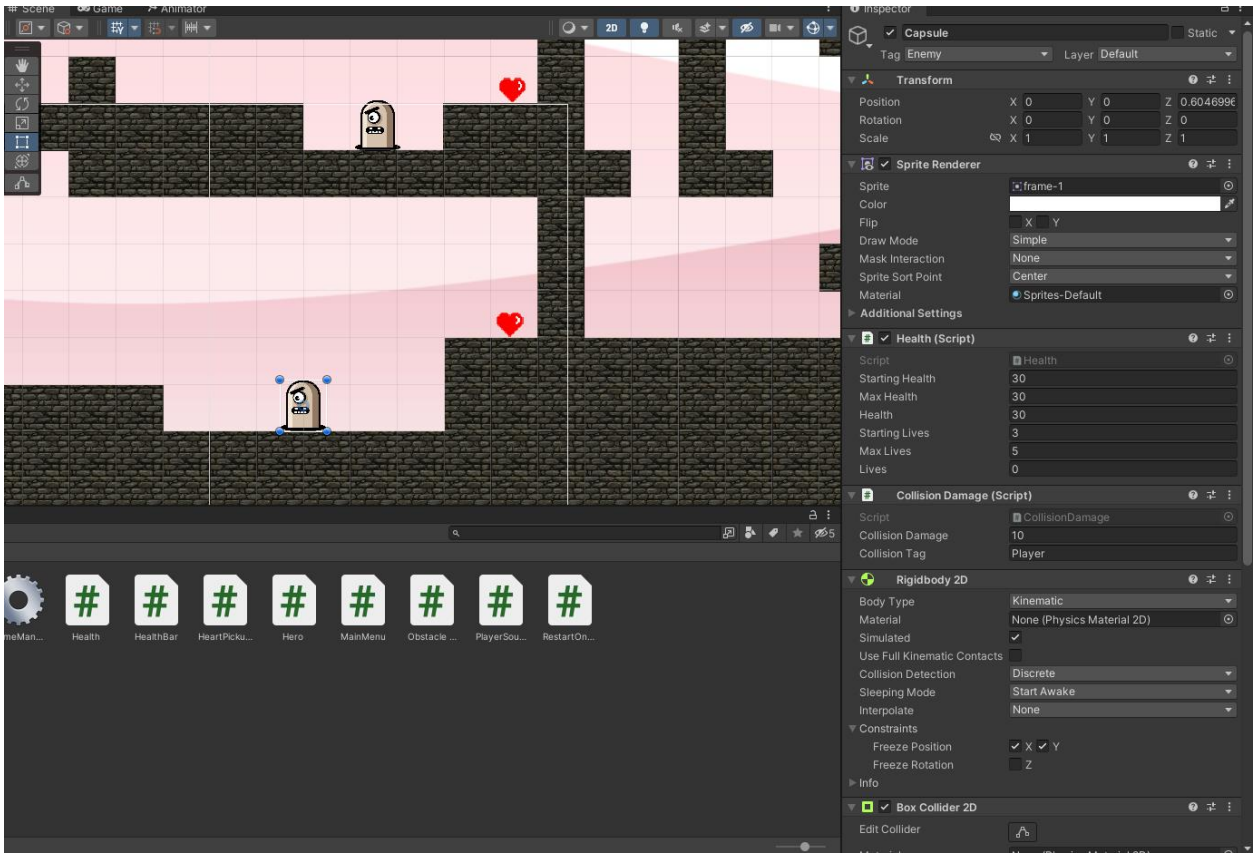

Рисунок 11 – Скрипт «Health» и «Collision Damage» для врага

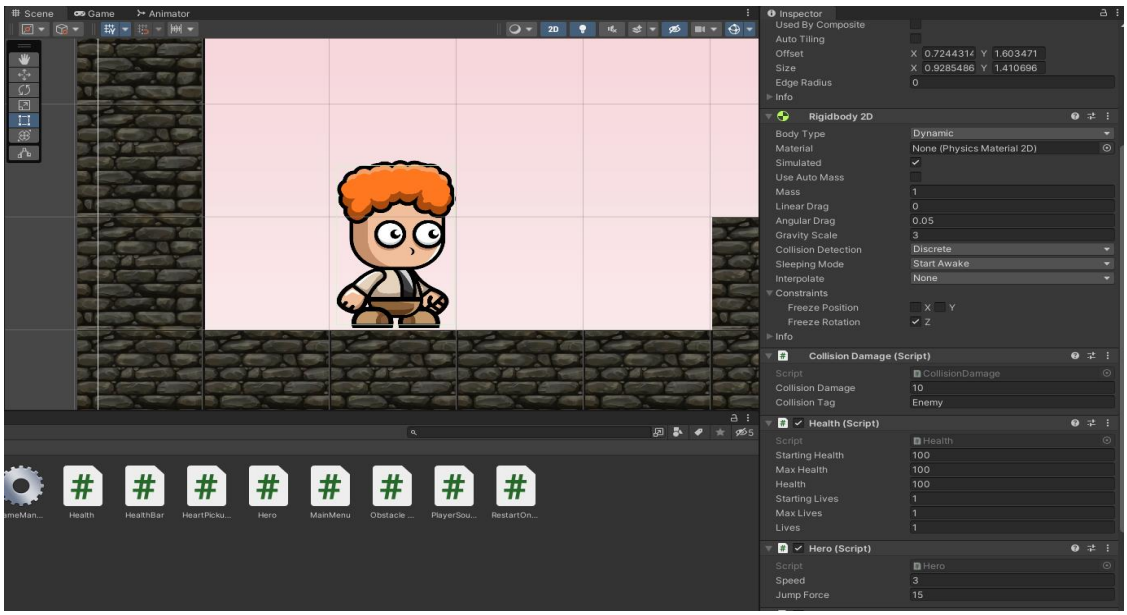

Рисунок 12 – Скрипт «Health» и «Collision Damage» для героя

Так же добавим для главного героя звуки прыжка и ходьбы, а также музыку для фонового прохождения игры. Для этого все так же создадим код в

папке Scripts и назовем в его Player Sound Controller [7]. В самом коде не забудем указать проверку для прыжка, чтобы звук воспроизводился, когда нажата клавиша пробела, так же само сделаем указание для ходьбы, чтобы звук воспроизводился, когда есть нажатие клавиш влево и право, а также на проверку того, что под персонажем есть блок Graund представлено на рисунке 13.

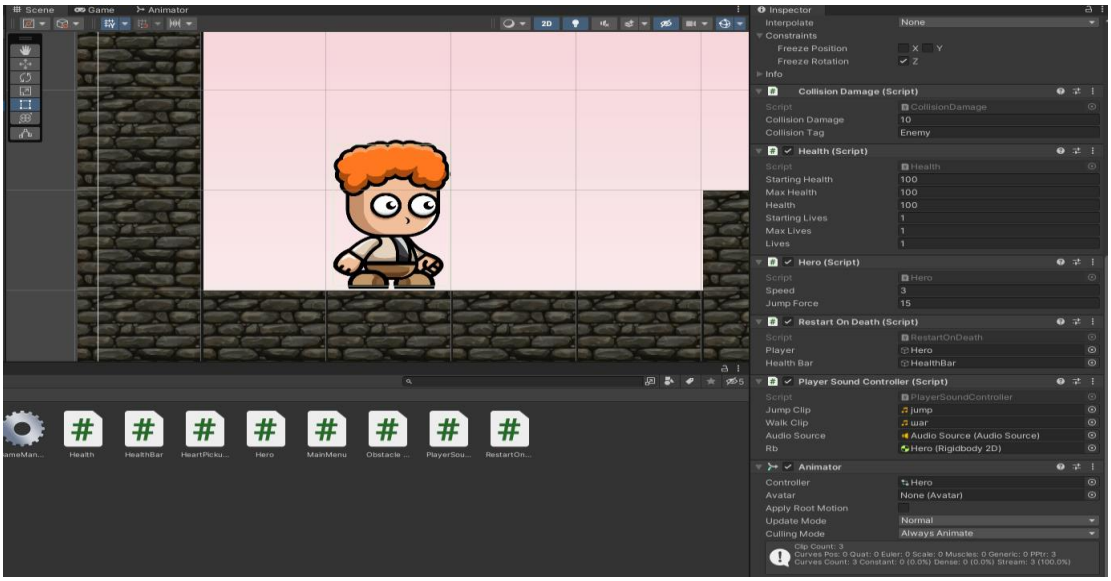

Рисунок 13 – Скрипт «Player Sound Controller» для звука прыжка и ходьбы героя

Был также реализован автоматический рестарт главного персонажа после смерти. Для этого было сделано следующее:

1. Создаем пустой объект на сцене.

2. Размещаем этот объект на том месте, где персонаж должен появиться после смерти.

3. Присваиваем объекту тег, «Restart On Death».

4. Создаём скрипт, который будет отслеживать смерть главного персонажа. В скрипте используем коллайдер и метод OnCollisionEnter2D, чтобы определить столкновение или вход персонажа в зону, сигнализирующую о его смерти [22].

5. В методе OnCollisionEnter2D проверяем, соприкасается ли персонаж с объектом, имеющим тег «Restart On Death».

6. Если условие выполнено (т.е., персонаж умер и соприкоснулся с объектом «Restart On Death»), выполните следующие действия:

1) Остановите все движения персонажа.

2) Установите его позицию в начальную точку или в позицию «Restart On Death».

3) Возможно, потребуется выполнить другие действия, например, сбросить его здоровье.

4) Сохранить и присоединить скрипт к персонажу представлено на рисунке 14.

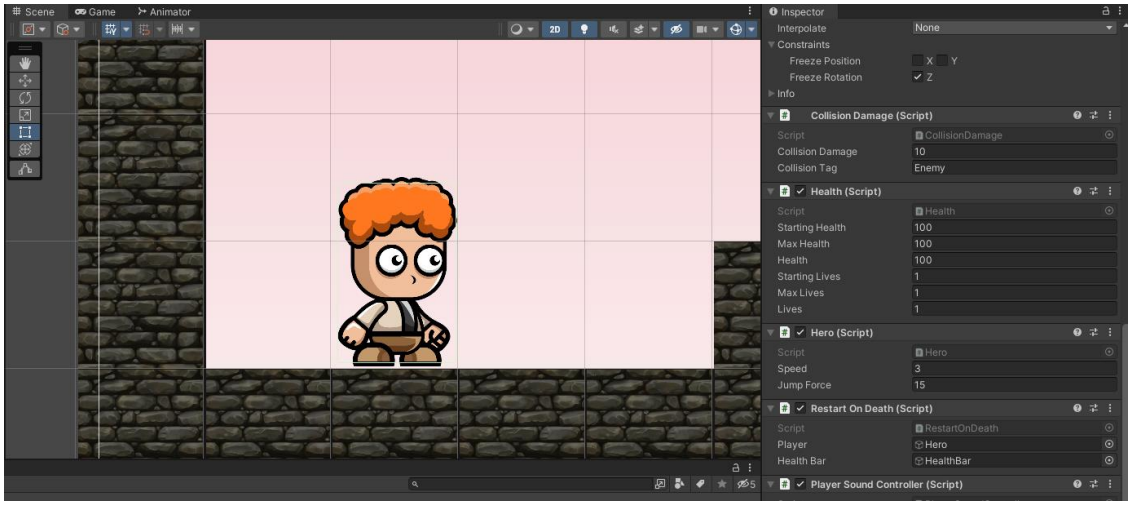

Рисунок 14 – Скрипт «Restart On Death» присоединенный к персонажу

Так же было реализовано, чтобы главный персонаж мог пополнять свои жизни, подбирая сердца, которые разбросаны по всему игровому миру. Чтобы реализовать эту задумку было проделано:

1. Создан спрайт сердца для отображения на экране.

2. Создан prefab для сердца, чтобы можно было многократно использовать его на уровне. Prefab содержит в себе компонент Sprite Renderer для отображения спрайта сердца.

3. Написан скрипт для объекта сердца, который будет отслеживать столкновение с персонажем. Добавлен компонент BoxCollider2D к сердцу;

4. Размещены сердца на уровне, которые изображены на рисунке 15.

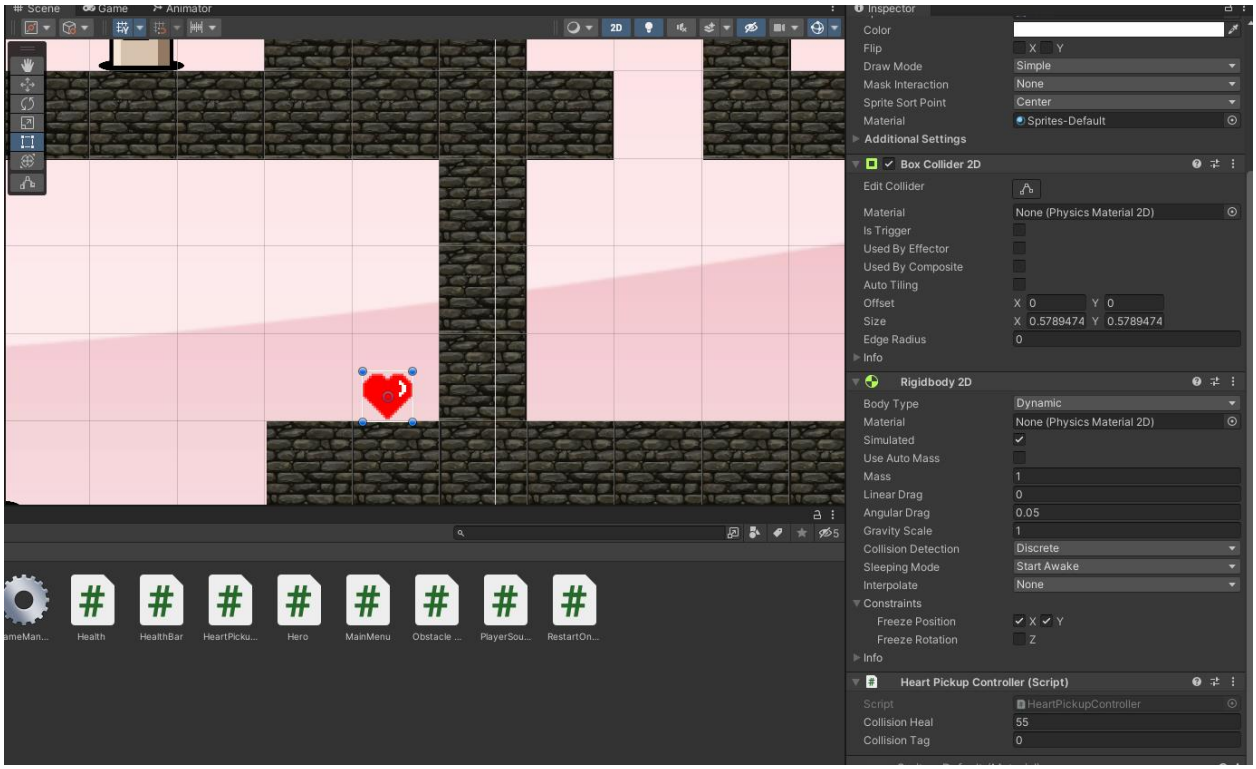

Рисунок 15 – Объект «Сердце» со скриптом

Следом создадим для главного персонажа полосу здоровья, а также добавим счёт врагов, которых осталось уничтожить. Для создания полосы здоровья, было сделано:

1. Создадим новый пустой объект в сцене Unity и назовём его «HealthBar».

2. Добавим на объект «HealthBar» компонент Sprite Renderer (компонент для отображения спрайтов) и присвоим ему спрайт, который будет использоваться в качестве фона полосы здоровья все это представлено на рисунке 16.

3. Создаём ещё один пустой объект и называем его «HealthFill» (или другое удобное имя). Расположите его внутри объекта «HealthBar» так, чтобы он находился внутри фона полосы здоровья.

4. Добавляем на объект «HealthFill» компонент Sprite Renderer и присваиваем ему спрайт, который будет использоваться для заполнения полосы здоровья.

5. Создаём новый скрипт и присваиваем его объекту «HealthBar». Открываем скрипт в редакторе кода.

6. В скрипте объявляем переменную, которая будет хранить текущее значение здоровья персонажа.

7. Прикрепляем скрипт к «HealthBar» и указываем в Unity на главного персонажа представлено на рисунке 17.

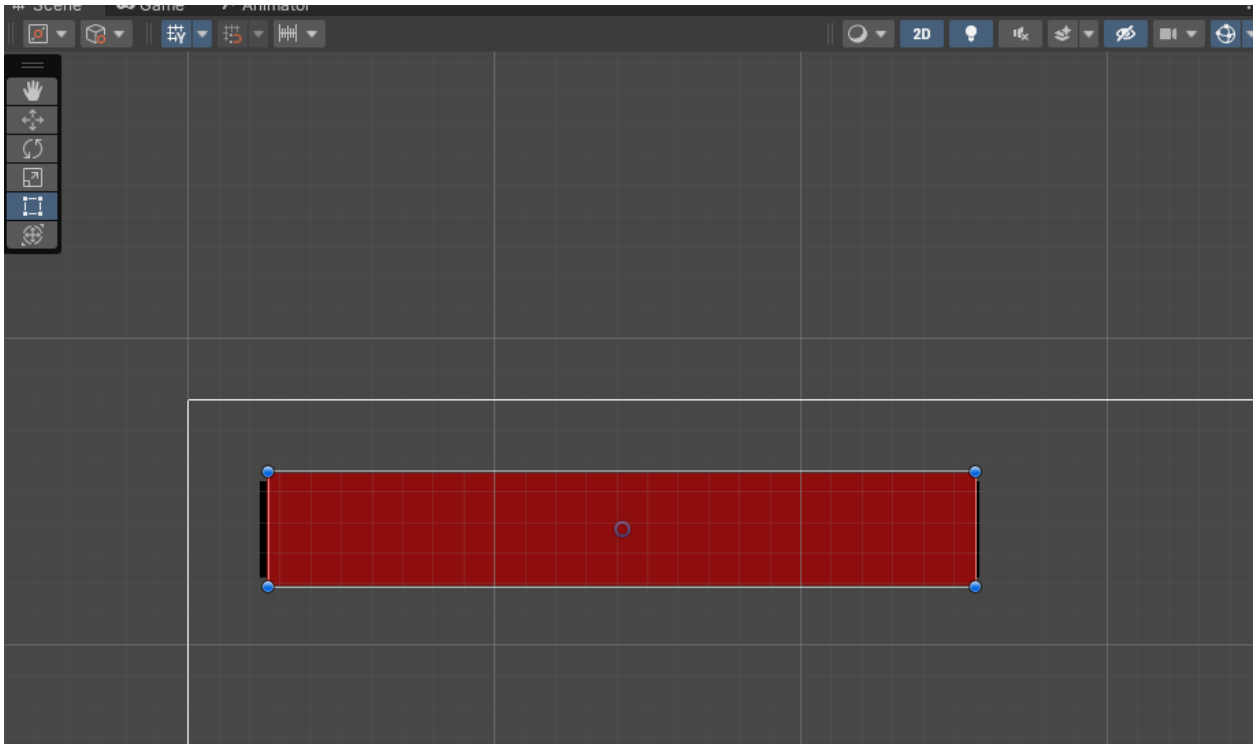

Рисунок 16 – Полоса здоровья для главного персонажа

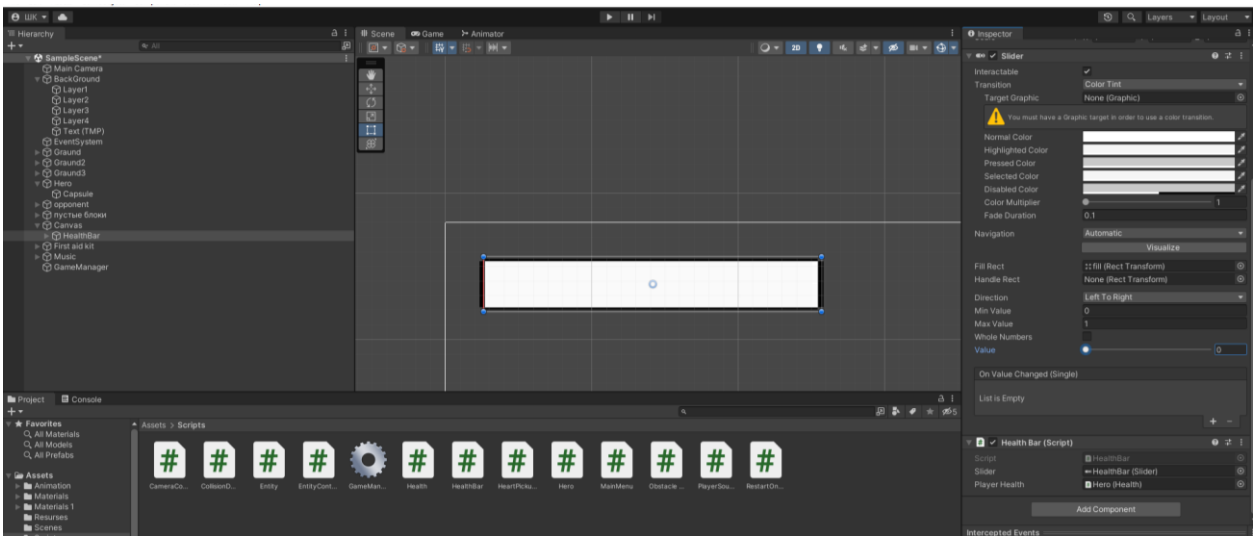

Рисунок 17 – Прикреплённый скрипт «Health Bar» и указание на объект «Hero»

Теперь создадим счёт для врагов, которых осталось уничтожить в игровом мире. Для этого была проделана следующая работа:

1. Создаём новый объект UI в сцене Unity. Например, выбираем GameObject → UI → Text(TMP).

2. Перемещаем текстовый объект так, чтобы он располагался в удобном месте на экране.

3. Настраиваем внешний вид текста. Изменяем размер, цвет и стиль текста, чтобы он соответствовал дизайну. Для этого использовались компоненты RectTransform и Text представленные на рисунке 18.

4. Создаём скрипт для управления счетчиком врагов.

5. Привязываем скрипт к текстовому объекту, перетащив его на компонент Text в инспекторе объекта.

6. В скрипте «EnemyCounter.cs» создаём публичное поле для текстового объекта, чтобы можно было получить к нему доступ из других скриптов.

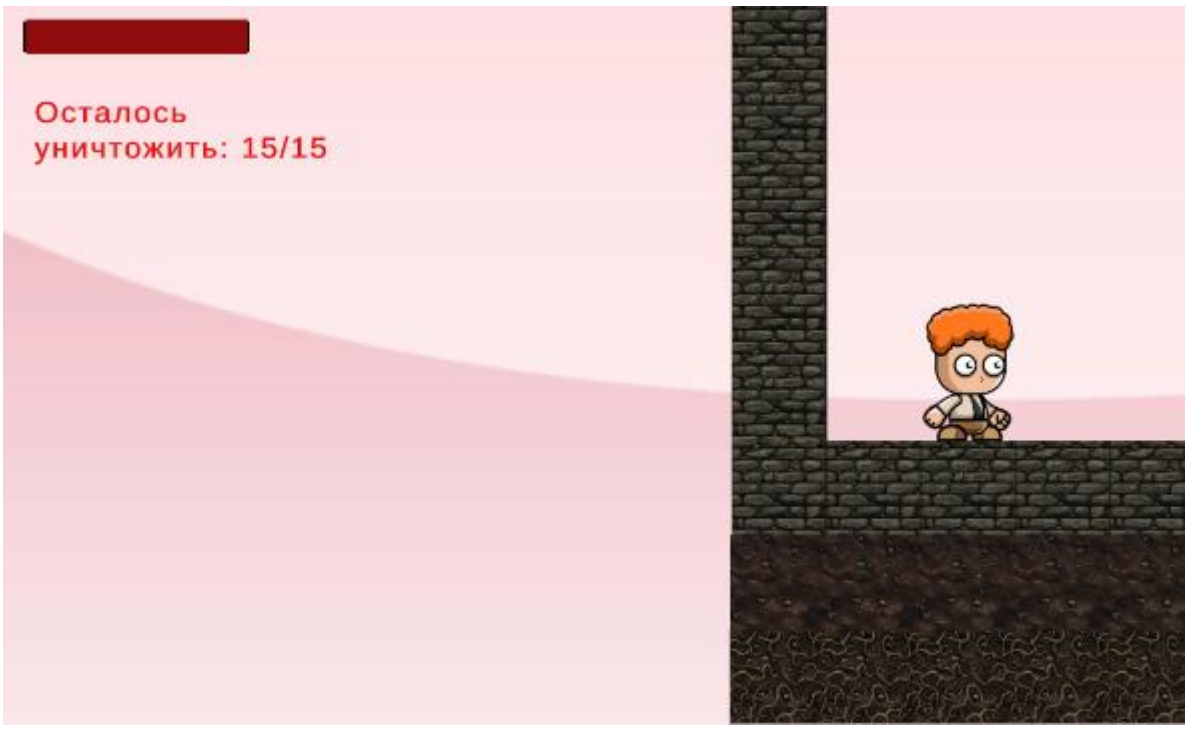

Рисунок 18 – Счётчик врагов которых осталось уничтожить

Так же создадим автоматический рестарт игры, после того как все враги будут уничтожены, и игра будет закончена. Для этого выполним не сложные действия.

1. Создаём новый пустой игровой объект. Нажав правой клавишей мышки по пустому месту в «Hierarchy» и выбираем «Create Empty».

2. Выбираем только что созданный пустой игровой объект в иерархии и переименуем его в «GameManager» [26].

3. Добавляем новый компонент скрипта к объекту «GameManager». Щелкнув правой кнопкой мыши на объекте «GameManager» в иерархии, выбираем «Create Empty Child» и назовите его «GameManage» представлено на рисунке 19.

4. Откроем созданный компонент скрипта «GameManage» в среде разработки, щелкнув по нему дважды в окне проекта.

5. После того как мы создали скрипт «GameManage», перетаскиваем его на игровой объект «GameManager» в иерархии или нажимаем кнопку «Add

Component» в окне Inspector и нахлдим скрипт «GameManager», чтобы добавить его к объекту.

6. Теперь мы можем использовать наш «GameManager» для управления игрой.

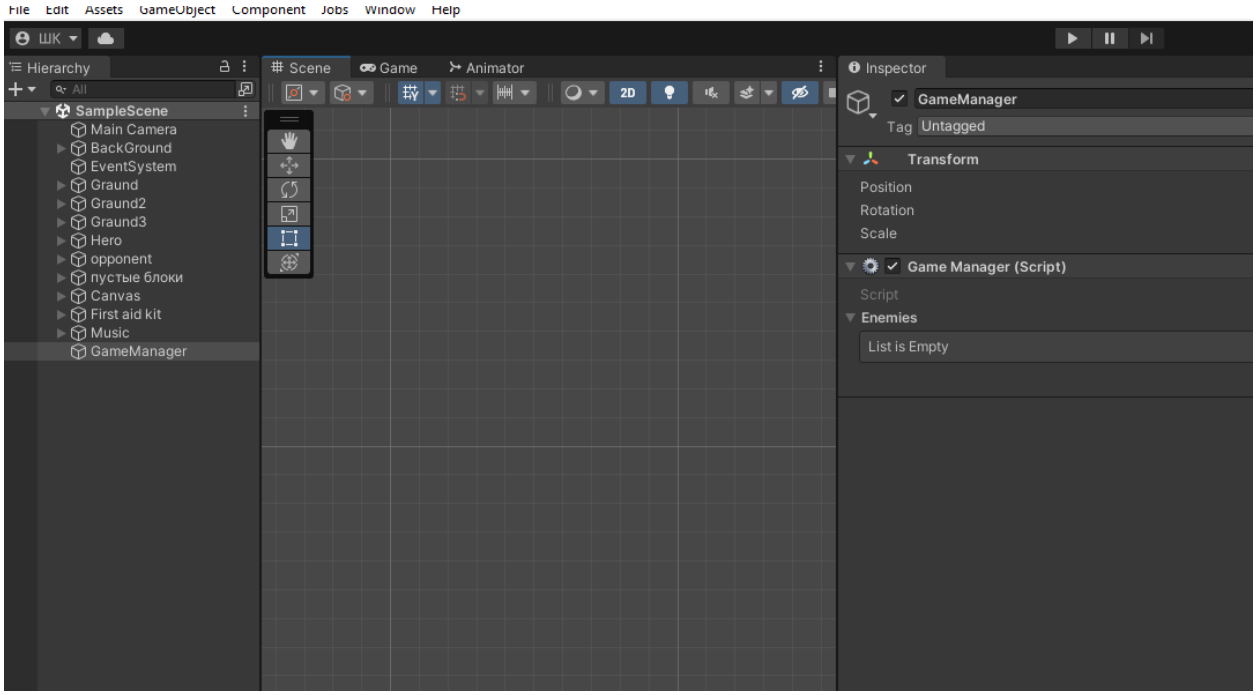

Рисунок 19 – Объект «Game Manager» для управления перезагрузки игры

И наконец создадим главное меню, где будут расположены кнопки «Играть» и «Выход» представлено на рисунке 20.

1. Создаём новую сцену в Unity для главного меню. Выбираем File → New Scene.

2. Создаём объекты кнопок в сцене Unity. Например, выберите GameObject → UI → Button.

3. Перемещаем кнопки на удобные места на экране.

4. Настраиваем внешний вид кнопок. Изменяем размер, цвет и текст кнопок, чтобы они соответствовали вашему дизайну. Для этого используем компоненты RectTransform и Text.

5. Создаём скрипт для обработки действий кнопок.

6. Привязываем скрипт к кнопкам, перетащив его на компонент Button в инспекторе объекта для каждой кнопки.

7. В скрипте MainMenu.cs добавляем функции-обработчика для кнопок «Играть» и «Выход».

8. Переключаемся на сцену главного меню и выбераем кнопку «Играть». В инспекторе объекта найдите компонент Button и укажите функцию-обработчик «PlayGame» из скрипта MainMenu.cs.

9. Для кнопки «Выход» также укажите функцию-обработчик «QuitGame» из скрипта MainMenu.cs.

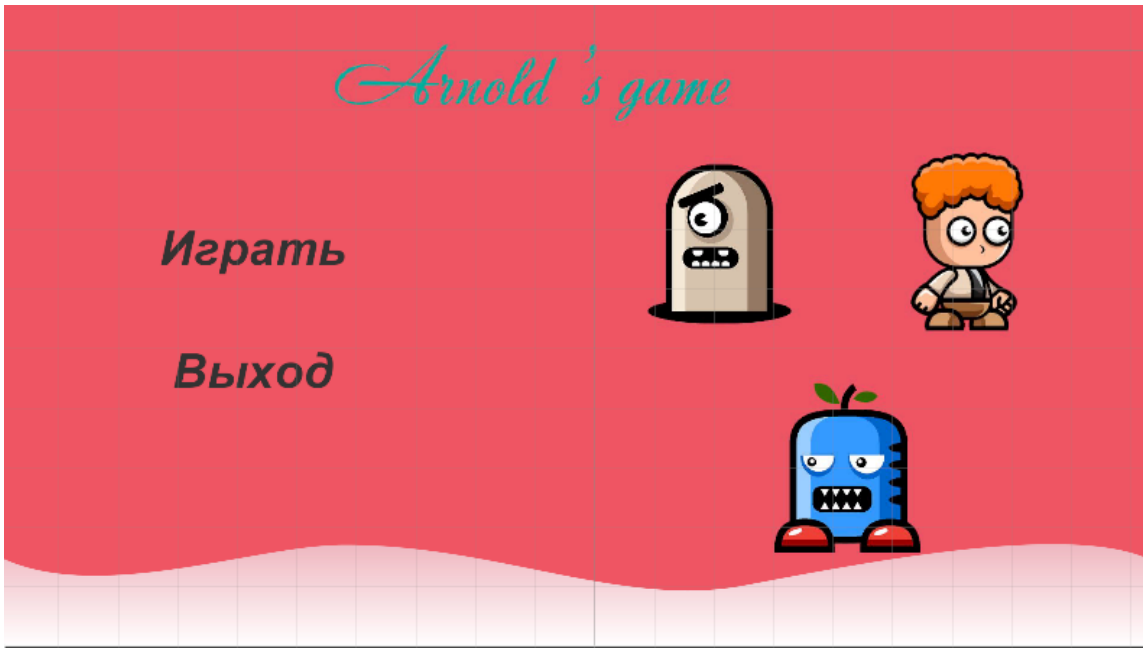

Рисунок 20 – Главное меню игры

3 Тестирование

На данном этапе разработки проекта не требуется проводить обширное тестирование, так как реализована лишь небольшая часть запланированных функций игры. Вместо этого, проводится небольшая проверка представленная в таблице 1 для удостоверения соответствия основным функциональным требованиям [36].

Таблица 1 – Функциональное тестирование

№	Назначение	Действия	Ожидаемый результат	Полученный результат	Итог
1	Проверка на корректное реагирование персонажа на поступающие команды о передвижении	Используя клавиши движения, пробежать часть уровня	Персонаж будет двигаться в положенном направлении, корректное проигрывание анимации	Персонаж передвигался в нужном направлении, корректное отображение анимации	Пройден
2	Проверка на корректный прыжок	Используя клавишу прыжка, подпрыгнуть несколько раз подряд	Игрок может сделать прыжок только с земли, корректное проигрывание анимации	Персонаж способен совершить прыжок только с земли, верное исполнение анимации	Пройден
3	Проверка поведения врагов	Подойти к врагу на достаточно близкое расстояние.	Враг должен наносить урон персонажу, при столкновении с ним	Враг наносит урон, как и должен это делать	Пройден
4	Проверка на корректную работу главного меню	1)Запустить уровень 2)Выйти из игры	Если нажать на кнопку «Играть», то игра начинается. Если нажать на кнопку «Выход», то игра закрывается	Все кнопки нажимаются и работают, как и задумывалось	Пройден
5	Проверка на корректный сбор объектов	Собирание объектов	Все объекты «сердец» можно собрать, и они восполняют жизни главного персонажа	Объект «сердце» собирается и здоровье персонажа пополняется	Пройден
6	Проверка на корректную реализацию события проигрыша игрока	Позволить персонажу умереть	При потери персонажем всего своего здоровья, игра должна начаться с начала	Главный персонаж умирает, и игра перезапускается автоматически	Пройден

Данное тестирование подтверждает, что на текущем этапе реализации проекта игра соответствует большинству функциональных требований, определенных в начальной задаче.

4. Технические характеристики

Важно понимать, что данный проект находится в стадии прототипирования и не является полноценной игрой, готовой для релиза. Разработка высококачественного продукта требует значительного времени и усилий. Например, создание даже небольшой мобильной игры с процедурно генерируемыми уровнями может занять более года при работе над проектом не более чем трех человек [13]. Увеличение числа участников команды может незначительно ускорить разработку, но в целом это несущественное изменение.

На данный момент сложно определить точные системные требования для будущего проекта, так как множество элементов еще не было реализовано. В настоящее время у нас есть только основные игровые механики, и технические характеристики для разрабатываемой компьютерной игры могут значительно измениться в будущем [14].

На текущем этапе мы можем сказать, что для запуска данного прототипа потребуется компьютер, соответствующий минимальным системным требованиям для корректной работы игрового движка Unity. Наиболее важной характеристикой является видеокарта, которая должна поддерживать DirectX 9 с шейдерами не ниже версии 3.0. Это означает, что данный прототип, вероятно, будет работать на большинстве современных компьютеров. Однако, при дальнейшей разработке и добавлении новых функций, возможно, потребуются более высокие системные требования.

5. Калькуляция проекта

Для реализации проекта были использованы множество различных элементов, начиная от статичных картинок и заканчивая несколькими скриптами для реализации механики игры.

Во время разработки игры было создано 2 сцены: Главное меню и Игровой уровень. В главном меню присутствуют лишь фоновое изображение и 2 кнопки, отвечающие за запуск Игрового уровня и выход из игры.

Общее количество объектов, которые были использованы или созданы во время реализации проекта, представлены в таблице 2.

Таблица 2 – Калькуляция элементов игры

Элемент	Комментарий
Фоновые изображения	4 растровых изображения
Тайлы	1 набор из 1024 тайла 2 набор из 121 тайла 3 набор из 121 тайла
Главный герой	3 анимации, 10 спрайтов
Враг	3 анимации, 16 спрайтов
Скрипты	13 скриптов с общим количеством строчек кода 630
Прочие элементы	3 спрайта, 3 аудиозаписи

ЗАКЛЮЧЕНИЕ

Овладение средой разработки Unity имеет значительный вес в современном мире, где игровая индустрия все более востребована. Игры перестали быть просто развлечением и нашли применение в других сферах, таких как наука и образование. Поэтому развитие этой области является одним из ключевых аспектов в современном обществе.

В процессе анализа доступных источников было проведено исследование компьютерных игр, в результате которого представлена классификация игр по четырем критериям. Однако, из-за относительной молодости игровой индустрии и отсутствия систематизации, подробная классификация не была составлена.

В работе представлен алгоритм разработки видеоигр. Проведен анализ популярных средств (платформ) разработки, их сравнительный анализ и выбор наиболее актуальных для начинающих разработчиков. При выборе приоритетных средств разработки учитывалась их доступность и функциональность.

В ходе анализа существующих разработок выявлены их преимущества и недостатки. При разработке игры с простой игровой механикой было рекомендовано обратить внимание на дополнительные элементы, такие как сюжет и графическое оформление, чтобы привлечь и удержать игрока, продлить жизненный цикл проекта [37].

Исходя из проведенного исследования, было принято решение разработать прототип двумерного платформера для одного игрока на игровом движке Unity. Это решение было принято по нескольким причинам:

1. Создание двумерной графики проще по сравнению с трехмерной.

2. Платформер является более простым жанром для реализации игровой механики.

3. Unity – бесплатный игровой движок, позволяющий разрабатывать приложения на языке программирования C#.

После принятия решения о выборе Unity в качестве среды разработки, мы приступили к изучению и освоению ее инструментов, разработке самого проекта. В процессе разработки были усвоены необходимые знания и навыки, связанные с игровым движком Unity и языком программирования C#: создание сцен; создание анимаций; создание и написание скриптов; настройка объектов; создание UI; компиляция проекта [1].

В ходе реализации проекта были решены и выполнены все поставленные следующие задачи.

Работа полезна желающим научиться созданию игр на Unity.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Алексеев, И. Ю. Разработка развивающих игр для детей на платформе Unity / Ю. И. Алексеев // Молодежная наука как фактор и ресурс инновационного развития : сборник статей II Международной научно-практической конференции. – 2020. – С. 94–97. – URL: https://www.elibrary.ru/item.asp?id=44537273 (дата обращения: 29.10.2022).

2. Арстанова, Л. Г. Занятия и развлечения со старшими дошкольниками. Разработки занятий, бесед, игр и развлечений на нравственные темы / Л. Г. Арстанова. – Москва : Учитель, 2017. – 324 с.

3. Архангельская, М. Д. Бизнес этикет, или игра по правилам / М. Д. Архангельская. – Москва : Эксмо, 2015. – 160 с.

4. Бартон, Д. Р. Биржевые стратегии. Игры без риска / Д. Р. Бартон, Сьюггеруд. – Москва : Санкт-Петербург: 2010. – 400 с.

5. Вакуленко, Ю. А. Веселая грамматика. Разработки занятий, задания, игры / Ю. А. Вакуленко. – Москва : Учитель, 2017. – 780 с.

6. Введение в программирование игр на Unity: [Электронный ресурс]. URL: https://mva.microsoft.com/ru/training-courses/-unity-8635 (дата обращения 01.04.23).

7. Винокуров Д.А. Разработка развивающей мобильной игры на платформе Unity / Д.А. Винокуров // наука молодых - наука будущего. – 2023. – URL: https://www.elibrary.ru/item.asp?id=53950874 (дата обращения: 03.03.2023).

8. Гейг, М. Разработка игр на Unity 2018 за 24 часа: пер. с англ. Райтмана М. А. – Москва : Эксмо, 2020. – 464 с.:ил.

9. Делаем игру за 6 уровков: [Электронный ресурс]. URL:http://tceh.com/e/unity/ (дата обращения 25.01.23).

10. Джейсон, Финкэнон Flash-реклама. Разработка микросайтов, рекламных игр и фирменных приложений с помощью Adobe Flash / Финкэнон Джейсон. – Москва : Рид Групп, 2012. – 945 с.

11. Джейсон, Финкэнон Flash-реклама. Разработка микросайтов,рекламных игр и фирменных приложений с пом / Финкэнон Джейсон. – Москва : РИД ГРУПП ООО Москва, 2012. – 288 с.

12. Дикинсон, К. Оптимизация игр в Unity 5: пер. с англ. Рагимова Р. Н. – Москва : ДМК Пресс, 2017. – 306 с.:ил.

13. Игровой дизайн, гейм дизайн (game design) / GameDev.ru – Разработка игр: [Электронный ресурс]. URL: http://www.gamedev.ru/gamedesign/terms/game play (дата обращения 10.01.23).

14. Издание о разработке и обо всем: [Электронный ресурс]. URL: https://tproger.ru/tag/unity/ (дата обращения 18.10.22).

15. Кокче, С. М. Передовой опыт работы: «Интеллектуальное развитие детей посредством развивающих игр» / С. М. Кокче // Наука молодых – наука будущего. – 15.08.2014 / URL: https://nsportal.ru/detskiy-sad/raznoe/2014/08/15/peredovoy-opyt-raboty-intellektualnoe-razvitie-detey-posredstvom (дата обращения: 13.04.2023).

16. Любанова, Т. П. Бизнес-план: опыт, проблемы. Содержание бизнес-плана, пример разработки / Т. П. Любанова, Л. В. Мясоедова Т. А. Грамотенко, и др.. – Москва : Приор, 2012. – 204 с.

17. Основы анимации в Unity: пер. с англ. Р.Рагимова. – М.: ДМК Пресс, 2016.– 176 с.: ил / А.Торн ISBN 978-5-97060-377-2

18. Паласиос, Х. Unity 5.x. Программирование искусственного интеллекта в играх: пер. с англ. Р. Н. Рагимова. – Москва : ДМК Пресс, 2017. – 272 с.: ил.

19. Паласиос, Хорхе Unity 5.x. Программирование искусственного интеллекта в играх / Хорхе Паласиос. – Москва : ДМК Пресс, 2016. – 849 с.

20. Петелин, Р. Ю. Fruity Loops Studio / Р. Ю. Петелин, Ю. В. Петелин – Санкт-Петербург : БХВ-Петербург, 2009. – 480 с.

21. Платов, В. Я. Деловые игры. Разработка, организация, проведение. Учебник / В. Я. Платов. – Москва : Профиздат, 2010. – 192 с.

22. Попов, А. А. Воспитание доброжелательности у детей 6–7 лет в процессе социо-игры / А. А. Попов, Е.В. Горшков, А.З Асроров // Тенденции развития науки и образования : сборник статей II Международной научно-практической конференции. – 2019. – С. 8-13. – URL: https://www.elibrary.ru/item.asp?id=50106522 (дата обращения: 09.11.2022).

23. Прахов, А. А. Самоучитель Blender 2.6. / А. А. Прахов – Санкт-Петербург : БХВ-Петербург, 2013. – 384 с.: ил.

24. Разработка 2D игры: [Электронный ресурс]. URL: https://null-code.ru/ (дата обращения 08.03.23).

25. Рябова, Т. В. Воспитание доброжелательности у детей 6–7 лет в процессе социо-игры / Т. В. Рябова // Молодежная наука как фактор и ресурс инновационного развития : сборник статей II Международной научно-практической конференции. – 2023. – С. 99–110. – URL: https://www.elibrary.ru/item.asp?id=50106522%20 (дата обращения: 28.10.2022).

26. Сергеев, Е. С. Разработка профориентационной vr-игры на платформе unity / Е. С. Сергеев, А. Е. Сухова, И. С Максимов, Н. А. Сенаторов // Научное обозрение. Технические Науки: сборник статей II Международной научно-практической конференции. – 2021. – С. 38-42. – URL: https://www.elibrary.ru/item.asp?id=45691799 (дата обращения: 09.11.2022).

27. Создание первой игры на Unity, от идеи до релиза: [Электронный ресурс]. URL: https://habr.com/post/321038/ (дата обращения 14.12.22)

28. СТО 4.2-07-2014 Общие требования к построению, изложению и оформлению документов учебной деятельности: [Электронный ресурс]. URL: https://about.sfu-kras.ru/docs/8127/pdf/799090 (дата обращения: 05.02.2023).

29. Таран, В. А. Играть на бирже просто?! / В. А. Таран. – Москва : Санкт-Петербург: 2016. – 272 с.

30. Торн, А. Искусство создания сценариев в Unity / пер. с англ. Р. Н. Рагимова. – Москва : ДМК Пресс, 2016. – 360 с.: ил.

31. Торн, А. Основы анимации Unity / пер. с англ. Р. Н. Рагимова. – Москва : ДМК Пресс, 2016. – 176 с.: ил.

32. Финни, К. 3D-игры. Все о разработке (+ CD-ROM) / К. Финни. – Москва : Бином. Лаборатория знаний, 2015. – 133 с.

33. Финни, К. 3D-игры. Все о разработке (+ CD-ROM) / К. Финни. – Москва : Бином. Лаборатория знаний, 2011. – 976 с.

34. Хахаев, И. А. Графический редактор GIMP: первые шаги / И. А. Хахаев – Москва : ALT Linux; Издательский дом ДМК-пресс, 2009. – 232 с.: ил.

35. Шрейер, Д. Кровь, пот и пиксели. Обратная сторона индустрии видеоигр / пер. с англ. Л. И. Степанова. – Москва : Издательство «Эксмо», 2018. – 368 с.: ил.

36. Unity – Руководство: Руководство Unity: [Электронный ресурс]. URL: https://docs.unity3d.com/ru/530/Manual/UnityManual.html (дата обращения: 12.11.2022).

37. Unity : официальный сайт / Unity Technologies, 2023 – . – URL: https://unity.com/download (дата обращения: 15.09.2022).

38. Unity в действии. Мультиплатформенная разработка на C#. – Москва.: Питер, 2018. – 608 с.

39. Unity, платформа разработки в реальном времени: [Электронный ресурс]. URL: unity.com/ru

40. Unity: Game Engin: [Электронный ресурс]. URL: http://unity3d.com (дата обращения: 11.10.2022).

ПРИЛОЖЕНИЕ А

Скрипт для перезагрузки уровня если персонаж умирает

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;
public class RestartOnDeath : MonoBehaviour
{
    public GameObject player;
    public GameObject healthBar;
    private Vector3 startingPosition;
    private Health playerHealth;
    private GameManager gameManager;
    private bool isRestarting = false; // Переменная для отслеживания состояния перезагрузки
    void Start()
    {
        if (player != null && player.transform != null)
        {
            startingPosition = player.transform.position;
            playerHealth = player.GetComponent<Health>();
        }
        gameManager = FindObjectOfType<GameManager>();
    }
    void Update()
    {
        if (playerHealth != null && playerHealth.lives <= 0)
        {
            RestartLevel();
```

```
        }
        if (gameManager != null && gameManager.EnemyCount <= 0)
        {
            RestartLevel();
        }
    }
    void RestartLevel()
    {
        if (!isRestarting) // Проверка, не выполняется ли уже перезагрузка
        {
            isRestarting = true; // Установка флага перезагрузки
            if (player != null && player.transform != null)
            {
                player.transform.position = startingPosition;
            }
            ResetGame();
            SceneManager.LoadScene(SceneManager.GetActiveScene().name);
        }
    }
    void ResetGame()
    {
        if (playerHealth != null)
       {
            playerHealth.lives = 1;
        }
    }
}
```

ПРИЛОЖЕНИЕ Б

Скрипт персонажа для звуков прыжка и ходьбы

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
public class PlayerSoundController : MonoBehaviour
{
    public AudioClip jumpClip;
    public AudioClip walkClip;
    public AudioSource audioSource;
    public Rigidbody2D rb;
    private bool isJumping = false;
    private bool isWalking = false;
    void Update()
    {
        if (Input.GetKeyDown(KeyCode.Space) && !isJumping)
        {
            audioSource.PlayOneShot(jumpClip);
            isJumping = true;
        }
        else if (rb.velocity.y == 0)
        {
            isJumping = false;
        }
        if ((Input.GetKey(KeyCode.A) || Input.GetKey(KeyCode.LeftArrow) ||
Input.GetKey(KeyCode.D) || Input.GetKey(KeyCode.RightArrow)) &&
rb.IsTouchingLayers(LayerMask.GetMask("Graund")) && Mathf.Abs(rb.velocity.x)
> 0.01f && !isWalking)
        {
```

```
            audioSource.PlayOneShot(walkClip);
            isWalking = true;
        }
        else if (Mathf.Abs(rb.velocity.x) < 0.01f)
        {
            isWalking = false;
        }
    }
}
```

ПРИЛОЖЕНИЕ В

Скрипт персонажа для нанесения урона по врагам

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
public class Obstacle : MonoBehaviour
{
   private void OnCollisionEnter2D(Collision2D collision)
   {//проверяем объект с которым мы столкнулись и если это игрок то наносим урон
      if (collision.gameObject == Hero.Instance.gameObject)
      {
         Hero.Instance.GetDamage();
      }
   }
}
```

ПРИЛОЖЕНИЕ Г

Скрипт главного меню

```
using UnityEngine;
using UnityEngine.SceneManagement;
public class MainMenu: MonoBehaviour
{
    public void PlayGame()
    {
        SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex + 1);
    }
    public void QuitGame()
    {
        Application.Quit();
    }
}
```

ПРИЛОЖЕНИЕ Д

Скрипт для передвижения и других функций персонажа

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
public class Hero: Entity
{
    [SerializeField] private float speed = 3f;
    [SerializeField] private float jumpForce = 15f;
    private bool isGrounded = false;
    private Rigidbody2D rb;
    private SpriteRenderer sprite;
    private Animator anim;
    public static Hero Instance { get; set; }
    private States State
    {
        get { return (States)anim.GetInteger("state"); }
        set { anim.SetInteger("state", (int)value); }
    }
    private void Awake()
    {
        rb = GetComponent<Rigidbody2D>();
        anim = GetComponent<Animator>();
        sprite = GetComponentInChildren<SpriteRenderer>();
        Instance = this;
    }
    private void FixedUpdate()
    {
        CheckGround();
```

```
    }
    private void Update()
    {
        if (isGrounded)
            State = States.Idle;
        if (Input.GetButton("Horizontal"))
            Run();
        if (isGrounded && Input.GetButtonDown("Jump"))
            Jump();
    }
    private void Run()
    {
        if (isGrounded)
            State = States.Run;
        Vector3 dir = transform.right * Input.GetAxis("Horizontal");
        transform.position = Vector3.MoveTowards(transform.position,
transform.position + dir, speed * Time.deltaTime);
        sprite.flipX = dir.x < 0.0f;
    }
    private void Jump()
    {
        rb.AddForce(transform.up * jumpForce, ForceMode2D.Impulse);
    }
    private void CheckGround()
    {
        Collider2D[] collider = Physics2D.OverlapCircleAll(transform.position, 0.5f);
        isGrounded = collider.Length > 1;
        if (!isGrounded)
            State = States.Jump;
    }
```

```
    public override void GetDamage()
    {
    }
}
public enum States
{
    Idle,
    Run,
    Jump
}
```

ПРИЛОЖЕНИЕ Е

Скрипт объекта «Сердце» для пополнения жизней персонажу

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
public class HeartPickupController: MonoBehaviour
{
    public int collisionHeal = 20;
    public int collisionTag;
    private void OnCollisionEnter2D(Collision2D collision)
    {
        // Проверяем столкновение с игровым объектом персонажа
        if (collision.gameObject.CompareTag("Player"))
        {
            // Получаем объект персонажа
            GameObject player = collision.gameObject;
            // Получаем или добавляем компонент Health к объекту персонажа
            Health health = player.GetComponent<Health>();
            if (health == null)
            {
                health = player.AddComponent<Health>();
            }
            // Увеличиваем количество жизней персонажа
            health.SetHealth(collisionHeal);
            // Удаляем объект "Heart" из сцены
            Destroy(gameObject);
        }
    }
}
```

ПРИЛОЖЕНИЕ Ж

Скрипт для отображения количества жизней в виде полосы

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
public class HealthBar: MonoBehaviour
{
    public Slider slider;
    public Health playerHealth;
    private void Start()
    {
        AddMaxHealth(playerHealth.maxHealth);
    }
    private void Update()
    {
        AddHealth(playerHealth.health);
    }
    public void AddMaxHealth(int health)
    {
        slider.maxValue = health;
        slider.value = health;
    }
    public void AddHealth(int health)
    {
        slider.value = health;
    }
}
```

ПРИЛОЖЕНИЕ И

Скрипт для отображения количества жизней

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
public class Health: MonoBehaviour
{
    public int startingHealth = 3; // Начальное количество здоровья.
    public int maxHealth = 5; // Максимальное количество здоровья.
    public int health; // Текущее количество здоровья.
    public int startingLives = 1; // Начальное количество жизней.
    public int maxLives = 1; // Максимальное количество жизней.
    public int lives; // Текущее количество жизней.
    private void Start()
    {
        health = startingHealth;
        lives = startingLives;
    }
    public void TakeDamage(int damageAmount)
    {
        health -= damageAmount;
        if (health <= 0)
        {
            lives--;
            if (lives <= 0)
            {
                Die();
            }
            else
```

```
            {
                Respawn();
            }
        }
    }
    private void Die()
    {
        if (gameObject.CompareTag("Player"))
        {
            RestartLevel();
        }
        else
        {
            Destroy(gameObject);
        }
    }
    private void Respawn()
    {
        health = startingHealth;
    }
    private void RestartLevel()
    {
        string currentSceneName =
UnityEngine.SceneManagement.SceneManager.GetActiveScene().name;
        UnityEngine.SceneManagement.SceneManager.LoadScene(currentSceneName);
    }
    public void AddHealth(int healthToAdd)
    {
        if (healthToAdd < 0)
        {
```

```
        Debug.LogWarning("Attempted to add negative health.");
        return;
    }
    health += healthToAdd;

    if (health > maxHealth)
    {
        health = maxHealth;
    }
}
public void AddLife(int lifeToAdd)
{
    if (lifeToAdd < 0)
    {
        Debug.LogWarning("Attempted to add negative life.");
        return;
    }
    lives += lifeToAdd
    if (lives > maxLives)
    {
        lives = maxLives;
    }
}
public void TakeHit(int damageAmount)
{
    if (damageAmount < 0)
    {
        Debug.LogWarning("Attempted to add negative damage.");
        return;
    }
```

```
        health - = damageAmount;
        if (health <= 0)
        {
            Die();
        }
    }
    public void SetHealth(int bonusHealth)
    {
        health += bonusHealth;
        if (health > maxHealth)
        {
            health = maxHealth;
        }
    }
}
```

ПРИЛОЖЕНИЕ К

Скрипт для перезапуска игры после выигрыша

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;
public class GameManager: MonoBehaviour
{
    [SerializeField]
    private List<GameObject> enemies = new List<GameObject>();
    public List<GameObject> EnemiesList { get { return enemies; } }
    public int EnemyCount { get { return enemies.Count; } }
    private static GameManager instance;
    public static GameManager Instance
    {
        get { return instance; }
    }
    private void Awake()
    {
        if (instance != null && instance != this)
        {
            Destroy(gameObject);
            return;
        }

        instance = this;
        DontDestroyOnLoad(gameObject);
    }
    private void Start()
```

```
    {
        FindEnemies();
    }
    private void FindEnemies()
    {
        GameObject[] enemyObjects =
GameObject.FindGameObjectsWithTag("Enemy");
        foreach (GameObject enemyObject in enemyObjects)
        {
            enemies.Add(enemyObject);
        }
    }
    public void EnemyDestroyed(GameObject enemy)
    {
        StartCoroutine(RemoveEnemyWithDelay(enemy, 5f));
        EntityController entityController = FindObjectOfType<EntityController>();
        entityController.OnEntityDestroyed(enemy); // Вызов метода
OnEntityDestroyed из EntityController
    }
    private IEnumerator RemoveEnemyWithDelay(GameObject enemy, float delay)
    {
        yield return new WaitForSeconds(delay);
        EnemiesList.Remove(enemy);
        CheckWinCondition();
    }
    private void CheckWinCondition()
    {
        if (EnemyCount == 0 && SceneManager.GetActiveScene().name ==
"GameScene")
        {
```

```
            RestartGame();
        }
    }
    private IEnumerator RestartGameCoroutine()
{
        yield return new WaitForSeconds(0.5f);
SceneManager.LoadScene(SceneManager.GetActiveScene().name);
        FindEnemies();
    }
    private void RestartGame()
    {
        StartCoroutine(RestartGameCoroutine());
    }
}
```

ПРИЛОЖЕНИЕ Л

Скрипт для отображения текстом сколько врагов осталось на уровне

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using TMPro;
public class EntityController : MonoBehaviour
{
    public TMP_Text entityCountText; // Текстовый объект для отображения счетчика врагов
    public string enemyTag = "Enemy"; // Тег врага
    private int totalEntityCount; // Общее количество врагов
    private int remainingEntityCount; // Количество оставшихся для уничтожения врагов
    private int collisionCount; // Количество столкновений с врагом
    void Start()
    {
        // Инициализация начальных значений счетчиков
        GameObject[] enemies = GameObject.FindGameObjectsWithTag(enemyTag);
        totalEntityCount = enemies.Length;
        remainingEntityCount = totalEntityCount;
        UpdateEntityCountText();
        collisionCount = 0;
    }
    // Метод вызывается при уничтожении врага
    public void OnEntityDestroyed(GameObject enemy)
    {
        if (enemy.CompareTag(enemyTag))
        {
```

```
            collisionCount++; // Увеличиваем счетчик столкновений
            if (collisionCount >= 3) // Проверяем, достигли ли трех столкновений
            {
                Debug.Log("Entity destroyed: " + enemy.name);
                // Уменьшаем текущее количество врагов
                remainingEntityCount--;
                // Уничтожаем объект врага
                Destroy(enemy);
                // Обновляем текстовый счетчик
                UpdateEntityCountText();
                collisionCount = 0; // Сбрасываем счетчик столкновений
            }
        }
    }
    // Метод для обновления текстового счетчика
    private void UpdateEntityCountText()
    {
        Debug.Log("Updating entity count text");
        // Обновляем текстовое значение счетчика врагов
        entityCountText.text = "Осталось уничтожить: " +
remainingEntityCount.ToString() + "/" + totalEntityCount.ToString();
    }
    // Обработка столкновений
    private void OnCollisionEnter(Collision collision)
    {
        if (collision.gameObject.CompareTag(enemyTag))
        {
            // Столкновение с врагом
            GameObject enemy = collision.gameObject;
```

```
            GameManager.Instance.EnemyDestroyed(enemy); // Вызов метода
EnemyDestroyed из GameManager
        }
    }
    private void Update()
    {
        int remainingEnemies = GameManager.Instance.EnemyCount;
        List<GameObject> enemyList = GameManager.Instance.EnemiesList;
        Debug.Log("Remaining Enemies: " + remainingEnemies);
        Debug.Log("Enemy List Count: " + enemyList.Count);
    }
}
```

ПРИЛОЖЕНИЕ М

Класс для всех врагов

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
public class Entity: MonoBehaviour
{//класс который будут наследовать все монстры и герой
    public virtual void GetDamage()
    {
    }
    public virtual void Die()
    {
        Destroy(this.gameObject);
    }
}
```

ПРИЛОЖЕНИЕ Н

Скрипт для нанесения урона

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
public class CollisionDamage : MonoBehaviour
{
    public int collisionDamage = 10;
    public string collisionTag;
    private void OnCollisionEnter2D(Collision2D coll)
    {
        if (coll.gameObject.tag == collisionTag)
        {
            Health health = coll.gameObject.GetComponent<Health>();
            if (health != null)
            {
                health.TakeDamage(collisionDamage);
                GameManager.Instance.EnemyDestroyed(coll.gameObject);
            }
        }
    }
}
```

ПРИЛОЖЕНИЕ П

Скрипт MainCamera для слежения за персонажем

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using UnityEngine;
namespace Assets.Scripts
{
   internal class CameraConroller : MonoBehaviour
   {
      [SerializeField] private Transform player;//приватное поле игрока
      private Vector3 pos;//сюда будем записывать координаты движения
      private void Awake()
      {
         if (!player)
            player = FindObjectOfType<Hero>().transform;//проверка на то что
найден игрок или нет
      }
      private void Update()
      {
         pos = player.position;//позиция игрока
         pos.z = -10f;//зафиксируем позицию камеры по оси z
         transform.position = Vector3.Lerp(transform.position, pos,
Time.deltaTime);//перемещаем камеру на координаты игрока
      }
   }
}
```