МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное
образовательное учреждение высшего образования
«Южно-Уральский государственный университет
(национальный исследовательский университет)»

Высшая школа электроники и компьютерных наук
Кафедра «Электронные вычислительные машины»

ДОПУСТИТЬ К ЗАЩИТЕ
Заведующий кафедрой ЭВМ
_____ Г.И. Радченко
«___»_____ 2021 г.

Разработка 2.5D игры с применением шейдерной графики на платформе Unity

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
К ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ

Руководитель работы,
к.т.н., доцент каф. ЭВМ
_____В. А. Парасич
«___»_____2021 г.

Автор работы,
студент группы КЭ-405
_____В. В. Кондратенко
«___»_____2021 г.

Нормоконтролёр,
ст. преп. каф. ЭВМ
_____С. В. Сяськов
«___»_____2021 г.

Челябинск-2021

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное
образовательное учреждение высшего образования
«Южно-Уральский государственный университет
(национальный исследовательский университет)»
Высшая школа электроники и компьютерных наук
Кафедра «Электронные вычислительные машины»

УТВЕРЖДАЮ

Заведующий кафедрой ЭВМ

_____ Г.И. Радченко

«___»_____2021 г.

ЗАДАНИЕ

на выпускную квалификационную работу бакалавра

студенту группы КЭ-405

Кондратенко Виктору Витальевичу

обучающемуся по направлению

09.03.01 «Информатика и вычислительная техника»

1. **Тема работы: «**Разработка 2.5D игры с применением шейдерной графики на платформе Unity» утверждена приказом по университету от 26 апреля 2021 г. №714-13/12
2. **Срок сдачи студентом законченной работы:** 1 июня 2021г.
3. **Исходные данные к работе:**

- описанное техническое задание;
- средства реализации – по выбору автора;
- обучающий материал, статьи по Unity;
- синтаксис C#;

- данные с игровых хакатонов;
- официальная документация Unity [1];
- советы с онлайн-форумов.

4. **Перечень подлежащих разработке вопросов:**

- анализ существующих игровых проектов, со схожей технологией разработки;
- описание сюжета, подходящего под техническое задание;
- выбор методов и средств реализации;
- рассмотрение средств реализации шейдерной составляющей;
- реализация спроектированных частей игры;
- тестирование и совершенствование программного продукта.

5. **Дата выдачи задания:** 1 февраля 2021 г.

Руководитель работы _____/*В. А. Парасич*/

Студент _____/*В. В. Кондратенко*/

КАЛЕНДАРНЫЙ ПЛАН

Этап	Срок сдачи	Подпись руководителя
Введение и обзор литературы	4.03.2021	
Разработка модели, проектирование	19.04.2021	
Реализация системы	11.04.2021	
Тестирование, отладка, эксперименты	15.05.2021	
Компоновка текста работы и сдача на нормоконтроль	26.05.2021	
Подготовка презентации и доклада	31.05.2021	

Руководитель работы _____/*В. А. Парасич*/

Студент _____/*В. В. Кондратенко*/

АННОТАЦИЯ

В. В. Кондратенко. Разработка 2.5D игры с применением шейдерной графики на платформе Unity. – Челябинск: ФГАОУ ВО «ЮУрГУ (НИУ)», ВШЭКН; 2021, 76 с., 20 ил., библиогр. список – 22 наим.

В рамках выпускной квалификационной работы производится анализ технологий по реализации шейдерной графики. Проводится ознакомление и реализация шейдеров для улучшения игрового продукта с использованием Shader Graph.

Проводится выбор перечня используемых инструментов/методов/языков, которые определят структуру разработки и благодаря этому, сформируется план по реализации. Рассматриваются преимущества и недостатки использования методов реализации игрового продукта.

Результатом работы являются навыки проектирования и разработки приложения, а также улучшение собственных профессиональных качеств автора.

Пояснительная записка к выпускной квалификационной работе оформлена в текстовом редакторе Microsoft Word 2016.

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ 7
1 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ 9
1.1 ОБЗОР АНАЛОГОВ 9
1.2 АНАЛИЗ ОСНОВНЫХ ТЕХНОЛОГИЧЕСКИХ РЕШЕНИЙ 10
1.3 ВЫВОД 16
2 ОПРЕДЕЛЕНИЕ ТРЕБОВАНИЙ 17
1.1 ФУНКЦИОНАЛЬНЫЕ ТРЕБОВАНИЯ 17
2.2 НЕФУНКЦИОНАЛЬНЫЕ ТРЕБОВАНИЯ 19
3 ПРОЕКТИРОВАНИЕ 21
3.1 АРХИТЕКТУРА ПРЕДЛАГАЕМОГО РЕШЕНИЯ 21
4 РЕАЛИЗАЦИЯ 26
4.1 ГЛАВНОЕ МЕНЮ 26
4.2 ИГРОВОЙ ПЕРСОНАЖ 28
4.3 ИНТЕРФЕЙС 29
4.4 ИСПОЛЬЗУЕМЫЕ ШЕЙДЕРЫ 32
4.5 НЕИГРОВЫЕ ПЕРСОНАЖИ 35
4.6 ИГРОВОЙ ПРОЦЕСС 35
4.7 ФАЙЛОВАЯ СТРУКТУРА ПРОЕКТА 36
5 ТЕСТИРОВАНИЕ 43
ЗАКЛЮЧЕНИЕ 46
БИБЛИОГРАФИЧЕСКОЙ СПИСОК 48
ПРИЛОЖЕНИЕ А 50

ВВЕДЕНИЕ

Темой является разработка 2.5D игры с использованием шейдерной графики на платформе Unity.

В настоящее время игровая индустрия активно развивается, поэтому имеются перспективы создания новых компьютерных игр. Игра способствует отвлечению человека от рутины, это способ показать что-то новое, также игры могут оказать поддержку или психологическую помощь игроку.

Для того чтобы визуально преобразовать игру используются шейдеры для определения окончательных параметров объекта или изображения, описание поглощения и рассеяния света, наложения текстуры, отражения и преломления, затенения, смещения поверхности и множество других параметров.

Существуют языки программирования шейдеров, инструменты для создания шейдеров без программирования. Современные платформы разработки видеоигр, такие как Unity и Unreal Engine, все чаще включают редакторы на основе узлов, которые могут создавать шейдеры без необходимости программирования; вместо этого, пользователю предоставляется ориентированный граф связанных узлов, который позволяет направлять различные текстуры, карты и математические функции в выходные значения. Затем автоматическая компиляция превращает график в настоящий скомпилированный шейдер. Всё это дает понимание принципов работы шейдеров и возможной последующей связи автора с графическими вычислениями.

Целью выпускной квалификационной работы является создание игры с 2.5D отображением графики, которая представлена в виде изометрической камеры на 2D спрайты, и работа с объектами игрового мира с помощью шейдерной обработки графики для получения навыков работы с новыми технологиями, а также внесение вклада в развитие инди-проектов.

Поэтому необходимо провести теоретический анализ возможностей инструментария по реализации шейдеров и места в проекте, где они могли бы быть

применены. Чтобы определиться, какие шейдеры можно использовать, необходимо сформировать вид проекта.

Так как весь проект можно реализовать разными путями, то результаты, удовлетворяющие как разработчика, так и независимых экспертов, будут достигаться постепенно, используя эмпирические методы.

Для достижения поставленной цели, необходимо решить следующие задачи.

1. Составление сюжета и персонажей, а также возможной цели для игры. Определение концепта игры.
2. Ознакомление и выбор версии платформы Unity.
3. Поиск и выбор дополнительного программного обеспечения для разработки игры.
4. Изучение работы и методов создания, и изменения шейдеров.
5. Поиск готовых ассетов для работы.
6. Создание скриптов работы персонажей и локации.
7. Определение минимальных системных требований.
8. Тестирование предварительного продукта с помощью различных людей (друзья/форумы).
9. Полное или частичное исправление багов.

Для реализации ВКР необходимо подобрать инструментарий, для этого нужно рассмотреть несколько вариантов и выбрать тот, который больше подойдёт по мнению автора.

1 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

1.1 ОБЗОР АНАЛОГОВ

Данный проект уже имеет определенные ограничения из темы, но не выявлен жанр для того, чтобы получить более понятные рамки для работы и просмотра аналогов [22]. Для данного проекта выбраны следующие жанры.

Платформер — жанр компьютерных игр, основной целью которых является перемещение персонажа игрока между точками в визуализированной среде. Часто характеризуются, но не определяются, тем, что они интенсивно используют прыжки и лазание, чтобы ориентироваться в среде игрока и достигать своей цели [2].

Приключение — один из основных жанров компьютерных игр, представляющий собой интерактивную историю с главным героем, управляемым игроком. Важнейшими элементами игры являются собственно повествование и исследование мира, а ключевую роль в игровом процессе играет решение головоломок и задач [4].

Перед началом определения требований просмотрим несколько проектов использующих 2.5D.

HADES

Hades - это ролевая игра в жанре рогалик [3], игра представлена в изометрической проекции. Игрок начинает прохождение игры, пытаясь пробиться через несколько комнат, при этом расположение комнат предопределено заранее, но их порядок и враги, которые появляются, определяются случайным образом [5].

Bastion

Bastion - это ролевая игра в жанре экшн с многоуровневой структурой. Персонаж игрока, «Малыш», перемещается по парящей в фэнтезийной тематике окружающей среде, которая образует дорожки, когда игрок приближается к краю. Уровни состоят из одной плоскости и просматриваются изометрически [6].

Don’t Starve

Don't Starve - это приключенческая игра со случайно генерируемым открытым миром и элементами выживания и игры в жанре рогалик. В бою нужно навести указатель мыши и щелкнуть мышью, в то время как другие действия управляются с клавиатуры, или с помощью встроенной поддержки. Цель состоит в том, чтобы выжить как можно дольше, и на экране отображается количество дней, в течение которых игрок выжил. Игра хранит несколько записей о прогрессе игрока, помимо общего количества очков опыта [7].

BattleToads

BattleToads - игра представляющая собой платформу с прокруткой beat 'em up (Возможностью перемещаться персонажем по глубине) с различными элементами гонок, скалолазанием и полетом, а также с препятствиями на транспортных средствах. [8]

1.2 АНАЛИЗ ОСНОВНЫХ ТЕХНОЛОГИЧЕСКИХ РЕШЕНИЙ

Разработка, редактирование, просмотр кода частей приложения будет происходить в Visual Studio Code и Visual Studio 2019 Enterprise с использованием дополнительных плагинов (расширений) для облегчения разработки.

Для создания (модификации) текстур (спрайтов) будут использоваться приложения Adobe.

Основную роль играет игровой движок, который будет использован для создания игры, поэтому выделим несколько из них.

Unity

Unity [9] является игровым движком, который позволяет создавать как 3D, так и 2D проекты. Язык программирования для написания скриптов используется C#. Данный игровой движок активно развивается и имеет поддержку шейдеров, а также инструмент для создания шейдеров без кода (Shader Graph).

Описание:

1. Основной узел совместной работы для художников, дизайнеров и программистов. Базовая платформа Unity ускоряет разработку, позволяя видеть результат изменений в реальном времени. 2D- и 3D-сцены, анимации и кат-сцены можно создавать непосредственно в Unity Editor.
2. Разработав продукт, вы сможете выпустить его более чем на 20 платформах, включая Windows, Mac, iOS, Android, PlayStation, Xbox, Nintendo Switch, а также на ведущих AR- и VR-платформах.
3. При разработке графики очень важен контроль над визуальными аспектами и наличие необходимого инструментария. Возможности Unity в области рендеринга в реальном времени позволяют вам создавать визуализации высочайшего качества благодаря Scriptable Render Pipeline.

Unreal Engine

Unreal Engine [10] открытый и продвинутый инструмент для создания 3D в реальном времени. Постоянно развиваясь, чтобы служить не только своей первоначальной цели в качестве современного игрового движка, сегодня он дает создателям из разных отраслей свободу и контроль для предоставления передового контента, интерактивного опыта и иммерсивных виртуальных миров. Язык программирования для скриптов — C++. Имеется поддержка шейдеров через систему узлов и кода.

Описание.

1. Движок и редактор реального времени, который предлагает фотореалистичный рендеринг, динамическую физику и эффекты, реалистичную анимацию, надежный перевод данных и многое другое.
2. Поскольку спрос на навыки работы с 3D в реальном времени находится на рекордно высоком уровне, изучение Unreal Engine — отличный способ раскрыть свой карьерный потенциал. От более 100 часов бесплатных онлайн-уроков для самостоятельного изучения до обучения под руководством

инструктора и возможностей для расширения ваших знаний на мероприятиях по всему миру —- существует целый ряд вариантов, соответствующих вашему стилю обучения.

GODOT

Godot [12] предоставляет огромный набор общих инструментов, поэтому можно сосредоточиться на создании своей игры, не изобретая колесо. Godot полностью бесплатен и имеет открытый исходный код под очень либеральной лицензией MIT. Никаких условий, никаких гонораров, ничего. Игра принадлежит разработчику до последней строчки кода движка. Godot использует язык шейдеров схожий с GLSL ES 3.0 [11].

Описание:

1. Большие или маленькие идеи легко адаптируются к архитектуре Godot на основе узлов, облегчая вашу жизнь.
2. Инновационный дизайн средства 3D-рендеринга, благодаря которому ваши работы будут выглядеть великолепно с минимальными усилиями;
3. Выделенный 2D-движок, работающий в пиксельных координатах, с множеством встроенных инструментов.
4. Объектно-ориентированный API с языковыми опциями, такими как GDScript, C #, C ++ и визуальные сценарии.

Wintermute

Wintermute Engine [14] инструмент для создания и запуска графических приключенческих игр «point and click», как традиционных 2D, так и современных игр 2.5D. В комплект входит интерпретатор времени выполнения и редакторы графического интерфейса пользователя для управления и создания игрового контента, а также документация, демонстрационные данные и готовые шаблоны. Рендер шейдеров благодаря OpenGL [13].

Описание:

1. Язык сценариев использует общий синтаксис типа C, аналогичный JavaScript, C ++, C #, Java или PHP.
2. Поскольку видеокарты с 3D-ускорением являются сегодня стандартом, WME может использовать 3D-ускорение для создания быстрой 2D-графики в высоком разрешении с расширенными графическими эффектами, такими как прозрачность, альфа-смешивание и сглаживание. На старых компьютерах WME может работать в так называемом «режиме совместимости», при котором не требуется 3D-ускоритель, но все расширенные графические эффекты отключены.
3. Чтобы ускорить разработку, WME предоставляет набор инструментов графического интерфейса для проектирования игровых сцен, анимации и управления содержимым проекта. Инструменты работают на самом движке.

Cocos2d

Cocos Creator [15] — это инструмент для разработки сценариев, компонентов сущностей и игр на основе данных, ориентированный на создание контента. Он поставляется с простым в использовании рабочим процессом создания контента и мощным набором инструментов разработчика для игровой логики и создания высокопроизводительных игр. Имеется поддержка шейдеров.

Описание:

1. Зрелая кроссплатформенная среда разработки игр с открытым исходным кодом, которая поддерживает создание 2D и 3D игр.
2. Движок предоставляет богатые функции, такие как рендеринг графики, графический интерфейс, аудио, сеть, физика, ввод данных пользователем и т.д.
3. Его ядро написано на C ++ и поддерживает разработку на C ++, Lua или JavaScript. Cocos2d-x развертывается в системах iOS, Android, HTML5,

Windows и Mac с функциями, ориентированными на собственные мобильные платформы.

Из рассмотренных платформ для разработки можно выделить Unity и Unreal Engine. Unity имеет более низкий порог вхождения, так как обладает большим количеством пользователей, а следовательно большим учебным материалом, в Unity имееются простые инструменты для начинающих одиночных разработчиков, когда Unreal Engine предусмотрен на то, что пользователь уже знаком с основами, также влияет и основной язык скриптов C++.

Посмотрим функционал, с которым будем работать в Unity.

Shader Graph

Shader Graph [16] позволяет с легкостью разрабатывать шейдеры в визуальном интерфейсе с отображением результатов в реальном времени.

Описание:

1. Предназначена для работы с функцией Scriptable Render Pipeline. Главные узлы, которые работают с Universal Render Pipeline и High Definition Render Pipeline, входят в стандартный комплект установки.
2. Поддерживает расширение для работы с любым процессом рендеринга.
3. Позволяет задать собственное поведение узла непосредственно в шейдерном графе или через файлы HLSL.
4. Создавать шейдеры в графическом интерфейсе Shader Graph и использовать их в Visual Effect Graph для изменения внешнего вида и методов отрисовки качественных визуальных эффектов.
5. Модуль Blackboard для оптимизации шейдеров с помощью Shader Level of Detail.

Так как данный визуальный интерфейс использует функцию Scriptable Render Pipeline, нужно определить, какой будет использоваться в данном проекте.

Universal Render Pipeline

Universal Render Pipeline [17] превратился в мощное графическое решение, которое сочетает в себе красоту, скорость и производительность и поддерживается на всех платформах, на которые ориентировано Unity.

Описание:

1. Открытый и гибкий рендеринг, настраиваемый с помощью сценария C#.
2. Масштабируемое качество графики для соответствия производительности устройства.
3. Функции постобработки включают сглаживание, глубину резкости, размытие в движении, проекцию Панини, цветение, искажение линз, хроматическую аберрацию, цветокоррекцию и отображение тонов, виньетирование, зернистость пленки и 8-битное дизеринг.

High Definition Render Pipeline

High Definition Render Pipeline [18] обеспечивает бескомпромиссную производительность графического процессора. Добивайтесь разнообразных образов, от фотореализма до стилизованных.

Описание:

1. HDRP создан специально для амбициозных проектов с высокой точностью воспроизведения.
2. HDRP нацелен на высокопроизводительное оборудование, такое как ПК, Xbox и PlayStation.
3. Используйте HDRP для создания высококачественных игр, автомобильных демонстраций, архитектурных приложений и всего, что отдает приоритет качеству графики. HDRP использует физическое освещение и материалы и поддерживает как прямой, так и отложенный пути рендеринга.

Основные настройки проекта были рассмотрены, также существует множество других настроек, которые будут уже установлены по мере создания проекта. Для реализации проекта желательно иметь систему контроля версии, чтобы не потерять

все файлы в случае неопределенных обстоятельств, то есть необходимо иметь Git и приложение для отображения интерфейса, чтобы работать с репозиторием. Так как разработка введется без команды и, следовательно, будет использоваться не полный функционал, то будет выбран удобный для автора инструмент.

1.3 ВЫВОД

Из рассмотренных технологических средств и опираясь на текущий опыт автора выбор платформы Unity оправдан, так как используется язык сценариев C#, а также имеется инструмент для создания шейдеров, который будет использоваться, с помощью графов Shader Graph, что облегчит знакомство с шейдерной составляющей. Для проекта будут настроен Universal Render Pipeline для рендеринга шейдеров, так как HDRP используется для более реалистичных сцен, а разработка будет ввестись с упором в 2D. Для контроля версий будет использоваться репозиторий GitHub по опыту работы автора, а также графический интерфейс для работы с репозиторием GitHub Desktop. Также будут использоваться другие вспомогательные инструменты для достижения цели.

2 ОПРЕДЕЛЕНИЕ ТРЕБОВАНИЙ

Требования к системе в целом

1. Наличие единого уровня сложности.
2. Наличие одного игрового персонажа под управлением пользователя.
3. Наличие не менее 3 персонажей.
4. Наличие звукового сопровождения: при действиях игровых персонажей, при запуске игровых скриптов из-за действий персонажей или изменения сцены, в процессе игры.
5. Наличие 3 этапов: пролог, завязка, развязка.
6. Наличие 3 различных локаций.
7. Камера должна следовать за персонажем, которым управляет игрок.
8. Взаимодействие с неигровыми персонажами может происходить только в виде диалога со звуковым сопровождением.
9. Возможность перезапуска уровня.
10. Наличие сохранений.

1.1 ФУНКЦИОНАЛЬНЫЕ ТРЕБОВАНИЯ

1. Взаимодействие с главным игровым меню включает в себя следующие возможности:
    1) переходить в меню настроек;
    2) запуск игры;
    3) выход из игры.
2. Возможность сохранение и загрузки игрового прогресса.
3. Интуитивно понятный игровой интерфейс.
4. Взаимодействие с игровыми предметами.
5. Реализовать несколько шейдеров с помощью Shader Graph.

Реализовать подсистемы

1. Запуск игры.
2. Меню настроек игры.
3. Выход из игры.
4. Срабатывание скриптов.
5. Шейдерная графика.

Функциональные требования к подсистеме “Запуск игры”

1. Должен загружаться дата-файл, хранящий информацию о пройденном игровом процессе.
2. Местоположение игрового персонажа устанавливается на точку возрождения в начальной локации.
3. В случае, если используется новый профиль, вначале загружается диалоговое окно с подсказками по управлению, позже начинается игра.

Функциональные требования к подсистеме “Меню настроек игры”

В данные настройки можно заходить как во время игры, так и находясь в главном меню игры, данные настройки позволяют регулировать следующее.

1. Изменение звука в игре.
2. Изменение разрешения экрана.
3. Просмотр клавиш управления.

Функциональные требования к подсистеме “Выход из игры”

При использовании из игрового меню происходит переход в главное меню игры.

При использовании из главного меню игры происходит завершение процесса игры.

Функциональные требования к подсистеме “Срабатывание скриптов”

По мере прохождения уровней и перехода в определенные точки, должны срабатывать события, которые так или иначе влияют на игру и открывают “двери” к дальнейшему прохождению.

Функциональные требования к подсистеме “Шейдерная графика”

1. Реализовать симуляцию движения объектов, таких как деревья и т.д.
2. Реализовать имитацию воды.
3. Реализовать освещение от некоторых объектов.
4. Реализовать имитацию тумана.

2.2 НЕФУНКЦИОНАЛЬНЫЕ ТРЕБОВАНИЯ

1. Работа игры на ОС Windows 8/8.1/10.
2. Разработка игры на платформе движка Unity версии 2020.3.2f1.
3. Скрипты/физика/поведение/шейдеры должны быть разработаны на языке программирования C#.
4. Поддержка контроллеров (геймпадов) совместимых с PC и технологией XInput.

Требования к пользователям

Для данного приложения может быть допущен любой пользователь, который умеет обращаться с компьютерной периферией.

Требования к объёму данных

Реализация хранилища с игровым процессом, а также должен быть реализован файл конфигурации настроек игры.

Требования к временным характеристикам

Передача информации между подсистемами не должна превышать более 5 секунд, а загрузка сцен и самого приложения не должна занимать более 15 секунд, при условии соответствия минимальных системных требований.

Требования к входным/выходным данным

Входными данными являются.

1. Локальные файлы сохранения (файлы из облака в случае подключения игры к службе цифровой дистрибуции).
2. Ввод данных для управления персонажем проходит с помощью периферии, такими как клавиатура, мышь, контроллер.

Выходными данными являются файлы сохранения, которые будут записываться как локально, так и на облако, в случае подключение к службе.

Требования к обучению

Пользователь при первом запуске игры должен получить памятку по управлению, которую также может просмотреть в настройках игры.

3 ПРОЕКТИРОВАНИЕ

Данная компьютерная игра предназначена для спокойного времяпрепровождения, следовательно, с не громким и нагруженным музыкальным сопровождением с возможным конечным моральным воодушевлением, прошедшего игру пользователя [21].

Игровой процесс “рассказывает” некоторую короткую историю, при этом процесс повествования сюжета украшается с помощью шейдерных возможностей, которые работают с отображением объектов и эффектов во время игрового процесса.

Основной упор разработки идёт на реализацию игровой составляющей, когда само меню представляет собой короткий список возможностей с базовыми функциями, при этом отображение меню связано с игровым процессом пользователя.

Разрабатываемый проект можно разделить на следующие части.

1. Работа с данными.
2. Работа с анимациями.
3. Дизайн уровней.
4. Реализация шейдеров.
5. Интерфейс.
6. Главное меню.
7. Звуковое сопровождение.
8. Реализация событий.

3.1 АРХИТЕКТУРА ПРЕДЛАГАЕМОГО РЕШЕНИЯ

Работа с данными

В данном модуле происходит чтение и запись информации в дата-контейнеры. Для проекта дата-контейнерами будут являться scriptable objects [19], которые позволят наделить проект модульностью, редактированием и отладкой. Это позволит обращаться различным объектам на сцене к этому файлу, который хранится как

обычный файл и доступен всем, тем самым убирая некоторые сложные связи между объектами на сцене и позволяя с лёгкостью редактировать связи.

Благодаря этому составлена структура работы с дата-контейнерами на рисунке 1, по которой будет реализована связь данных.

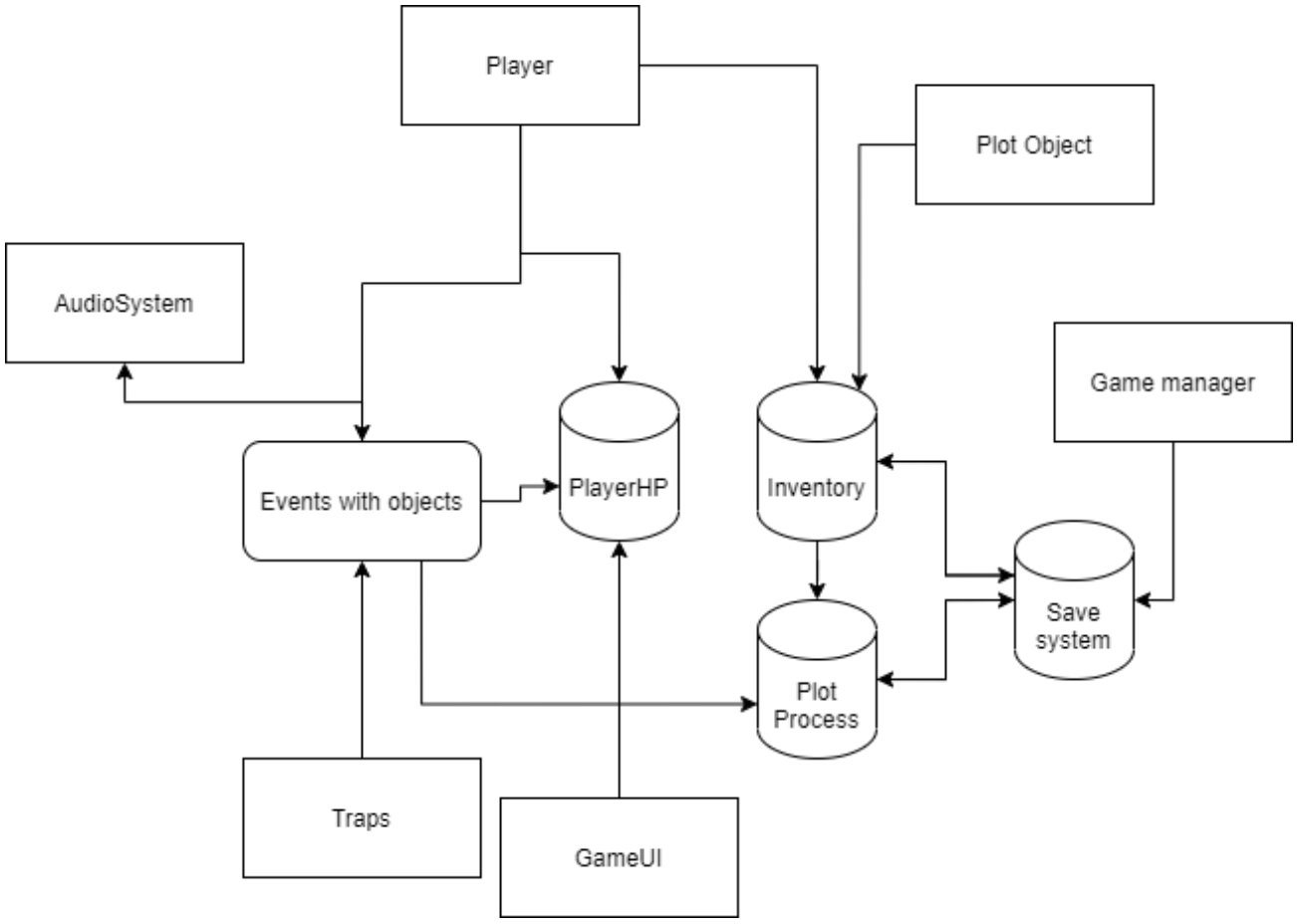

Рисунок 1 – Связь объектов и дата-контейнеров

Работа с анимациями

Данный модуль предполагает реализацию анимации для действий различных объектов. Основная необходимая анимация для главного игрового персонажа является передвижение по восьми направлениям.

Необходима анимация самих объектов, с которыми взаимодействует главный персонаж, анимация будет реализована спрайтами. Данные анимации будут

реализованы с помощью компонента аниматора, в котором находится контроллер, управляющий активацией анимации.

Используя функционал “Animation” можно изменять скорость воспроизведения анимации, что позволит подобрать оптимальный переход, также можно создавать анимацию, не имея дополнительных спрайтов, и вызов какой-либо функции при активации.

Дизайн уровней

В данном проекте находится 3 локации с “туннелями” между ними. Все локации находится на одной сцене, но доступ к ним открывается при соблюдении определенных правил. В начале каждой локации находятся точки возрождения, чтобы восстановить персонажа в случае выхода из игры и последующего возвращения. В каждой локации необходимо пройти головоломку, после прохождения которой можно получить “ключ”.

Локация “HomePlace” – начальная локация откуда начинается повествование игры, в данном месте игрок каждый раз появляется при запуске игры, также данная локация является конечной при сборе всех “ключей”.

Локация “ForgottenDungeon” – заключительная локация с заснеженным биомом, проход в данную локацию доступен только владея песочным ключом. Туннель в данную локацию является переходом из травяных спрайтов в заснеженные спрайты поверхности.

Локация “HiddenGarden” – локация с растительностью, которая является первой доступной локацией для прохождения. Туннель в данную локацию представляется подобием болота с мертвыми деревьями и туманом.

Локация “Desert” – вторая по прохождению локация, которая находится в песочном биоме, пройти в локацию можно только владея ключом из сада. Туннель в данную локацию является переход травы в пустынные спрайты.

Реализация шейдеров

В данном модуле происходит разработка шейдеров для игрового проекта с помощью инструмента Shader Graph, который использует параметры, установленного в данный проект, рендера графики. По реализованным шейдерам будут созданы материалы для привязки их к объектам на сцене в качестве компонентов рендеринга, которые будут влиять на объект во время игрового процесса.

Интерфейс

В игровом интерфейсе отображено количество здоровья. Также интерфейс захватывает элементы UI игрового процесса, то есть весь всплывающий текст или диалоги являются элементами интерфейса в том числе и игровые элементы головоломок.

Главное меню

Является отдельной сценой, которая загружает информацию о прогрессе из дата-файла. В данной сцене можно выполнить следующие действия.

1. Продолжить игровой процесс – запустить последнее место сохранением с полученными ключами.
2. Начать новую игру – запустить игровую сцену со сбросом достигнутых достижений игроком.
3. Параметры игры – меню с возможностью изменения звука, настройкой разрешения экрана и выбора средства ввода для управления.
4. Выход – закрытие приложения и окончание его процесса.

Звуковое сопровождение

Звуковое сопровождение будет цикличным для сцены главного меню и игровой сцены. Для передвижения по локациям будет свой набор звуков для шагов, выбирающийся случайно.

Реализация событий

Данный модуль реализуется с помощью системы Unity Events, которая будет использовать как scriptable objects, хранящие тот или иной функционал, так и существующие объекты на сцене, изменяя их параметры или вызывая методы.

Основные события будут происходить с помощью действий самого игрока, а именно перемещение персонажа в область срабатывания скрипта, взаимодействие с некоторыми игровыми объектами или вызов игрового меню.

4 РЕАЛИЗАЦИЯ

Во время реализации проекта использовалась платформа Unity [20], версии 2020.3.2f1, для разработки игры с использованием дополнительных ресурсов из Unity Asset Store, а также сторонних источников.

Для редактирования спрайтов использовался редактор Adobe Photoshop 2021.

Для сохранения файлов проекта использовался репозиторий на GitHub, взаимодействие с которым происходило с помощью распределённой системой управления версиями Git и графической оболочкой GitHub Desktop.

4.1 ГЛАВНОЕ МЕНЮ

При запуске игры перед игроком предстаёт сцена главного меню, отображенного на рисунке 2. На данной сцене игрок выполняет навигацию по меню с помощью компьютерной мыши.

Рисунок 2 – Главное меню

Используя опцию “Continue”, игрок продолжает игру с последнего места сохранения.

Используя опцию “New Game”, игрок начинает новую игру со сброшенным пройденным прогрессом.

Опция “Exit” Завершает процесс приложения, закрывая его.

Опция “Options” открывает перед игроком параметры настроек, изображенные на рисунке 3.

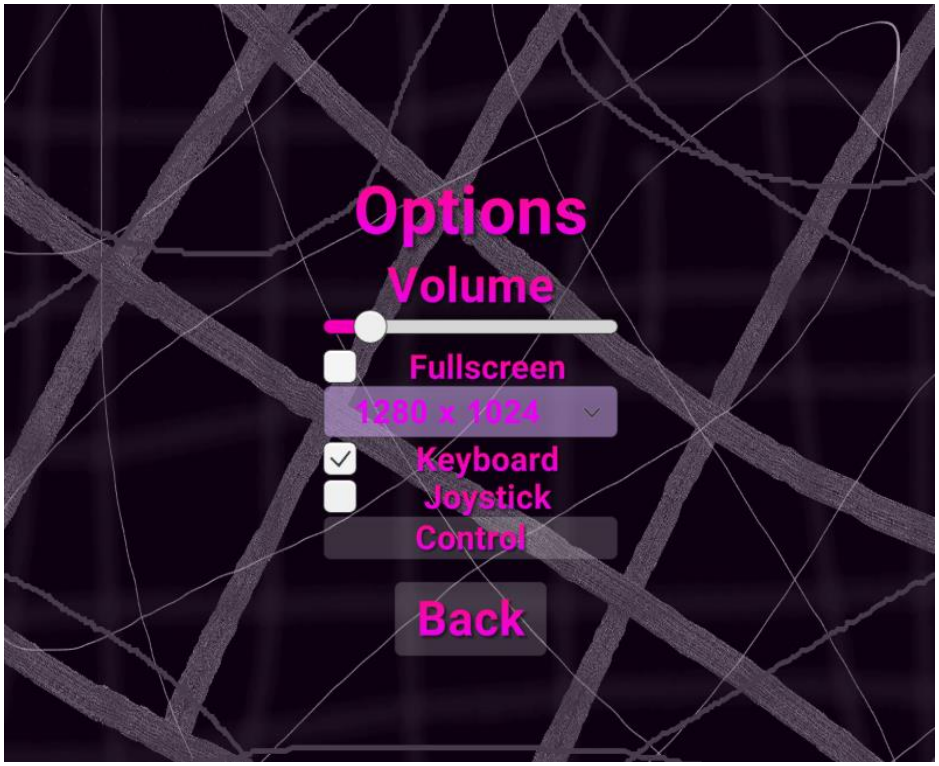

Рисунок 3 – Главное меню, меню настроек

С помощью данного меню можно изменить следующие параметры.

1. Изменять громкость звукового сопровождения как на сцене главного меню, так и на игровой сцене.
2. Сделать игру в полноэкранном или оконном режиме.
3. Выбрать разрешение экрана из списка поддерживаемых разрешений монитора.
4. Выбрать устройство ввода, которое будет использовать игрок.

5. Посмотреть управление для клавиатуры и геймпада.
6. Вернуться в главное меню.

4.2 ИГРОВОЙ ПЕРСОНАЖ

Игрок управляет ведьмой, которая была взята из доступных для пользования ресурсов и изображена на рисунке 4.

Рисунок 4 – Спрайт главного персонажа

Реализация анимации передвижения главного персонажа с помощью использования аниматора и blend tree для структурированности переходов состояний и изображена на рисунке 5.

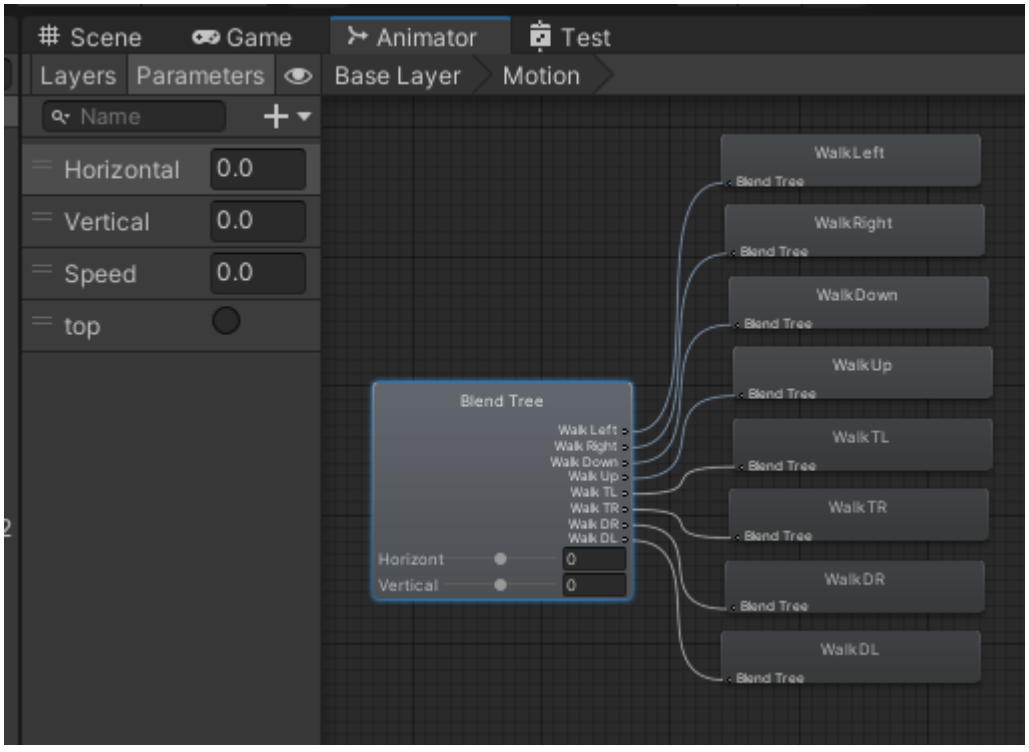

Рисунок 5 – Blend tree для анимации персонажа

Передача параметров для аниматора происходит с помощью получения данных с устройств ввода и передачи их через скрипт.

Персонаж под управлением игрока имеет следующие параметры.

1. Запас здоровья и его регенерация, возрождение.
2. Анимированное перемещение со звуковым сопровождением шагов.
3. Состояния свободного движения и взаимодействия с объектами.

4.3 ИНТЕРФЕЙС

Во время игрового процесса интерфейс игрока отображен на рисунке 6. На интерфейсе показана шкала здоровья персонажа и его иконка, весь получаемый урон отображается путём уменьшения красной зоны с помощью используемого скрипта SliderSet, взаимодействующего с UI.

Рисунок 6 – UI в игровом процессе

Во время игры игрок с помощью клавиатуры или геймпада может вызвать паузу, изображенную на рисунке 7.

Рисунок 7 – Пауза во время игрового процесса

Меню паузы имеет следующий функционал:

1. Продолжение игры с помощью опции “Resume”.
2. Переход в меню настроек, функционал аналогичен настройкам из главного меню.
3. Переход в главное меню с помощью опции “Exit”.

Диалоговое меню также отображается в UI и показано на рисунке 8. Окно диалога переводит состояние персонажа в “взаимодействие” запрещая ему перемещение, пока диалог не будет завершён. Выбор ответных сообщений можно производить как с помощью компьютерной мыши, так и с помощью выбранного устройства ввода.

Рисунок 8 – Диалоговое окно

При взаимодействии с объектами, которые вызывают игру в паззлы, появляется в UI и изображён на рисунке 9.

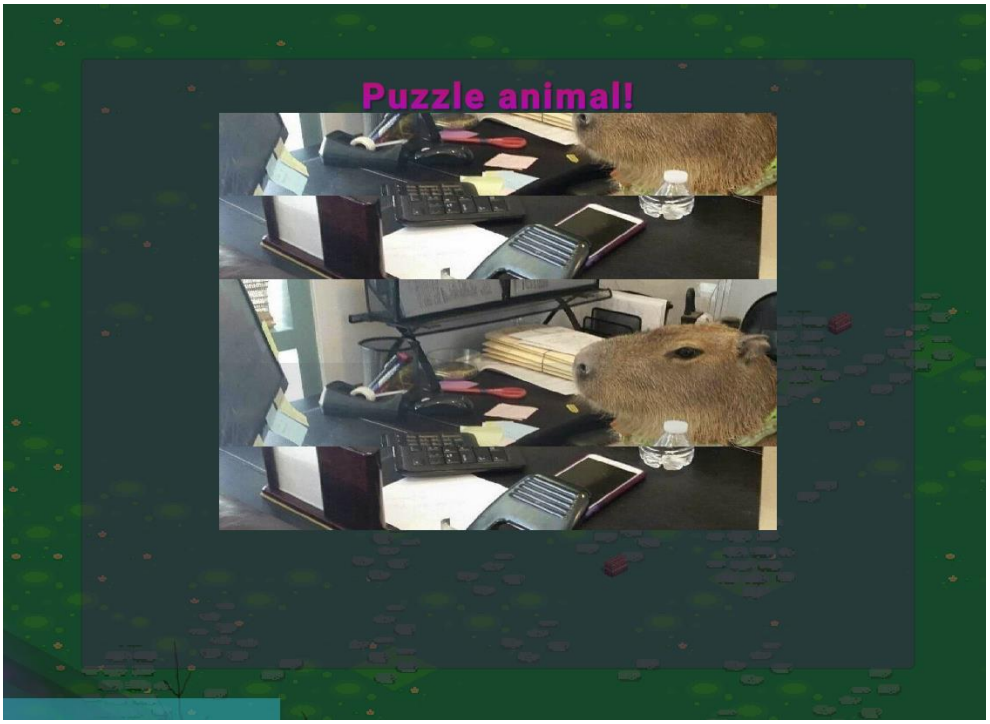

Рисунок 9 – Игра в паззлы

Паззл представляет из себя поле, в котором находится установленное количество частей паззла. При запуске паззла изображение на каждой части паззла выбирается случайно. Взаимодействие с паззлом осуществляется с помощью выбранного устройства ввода или компьютерной мыши.

4.4 ИСПОЛЬЗУЕМЫЕ ШЕЙДЕРЫ

С помощью используемого инструмента Shader Graph реализованы шейдеры «покачивания» для спрайтов растительности, «расщепление» игровых объектов, «туман» и «песчаная буря». Рассмотрим один из реализованных шейдеров, отображённый на рисунках 10.1 – 10.3.

Для управления над некоторыми узлами шейдера вне редактора созданы настроечные свойства.

1. MainSprite – управляет параметром _MainTex, который является спрайтом, над которым будут происходить преобразования.
2. WindMovement – управляет скоростью изменения преобразований над спрайтом.
3. WindDensity – управляет плотностью шумов, накладываемые на спрайт.
4. WindStrenght – управляет множителем усиления плотности шумов.
5. WindInfluenceMask – служит для изменения маски для воздействия шейдера над спрайтом.

На рисунке 10.1 подготовляются параметры направления карты шума, который реализован с помощью узла “Gradient Noise”. Направление получается за счёт использования глобальных координат и наложенного на него смещения. Смещением является вектор направления, который зависит от времени, зависимость получается умножением параметров. С помощью полученного направления и плотности генерируется маска накладываемого шума. Следующий узел корректирует плотность шума, уменьшая его слабые участки.

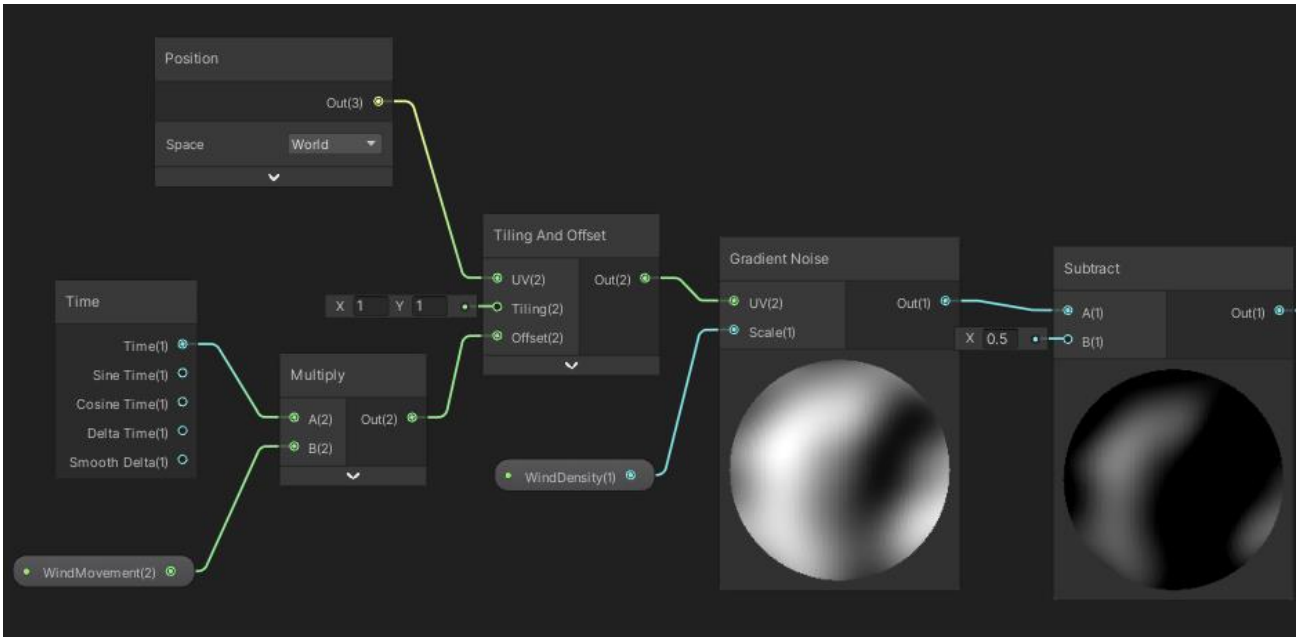

Рисунок 10.1 – Шейдер «покачивания» растительности

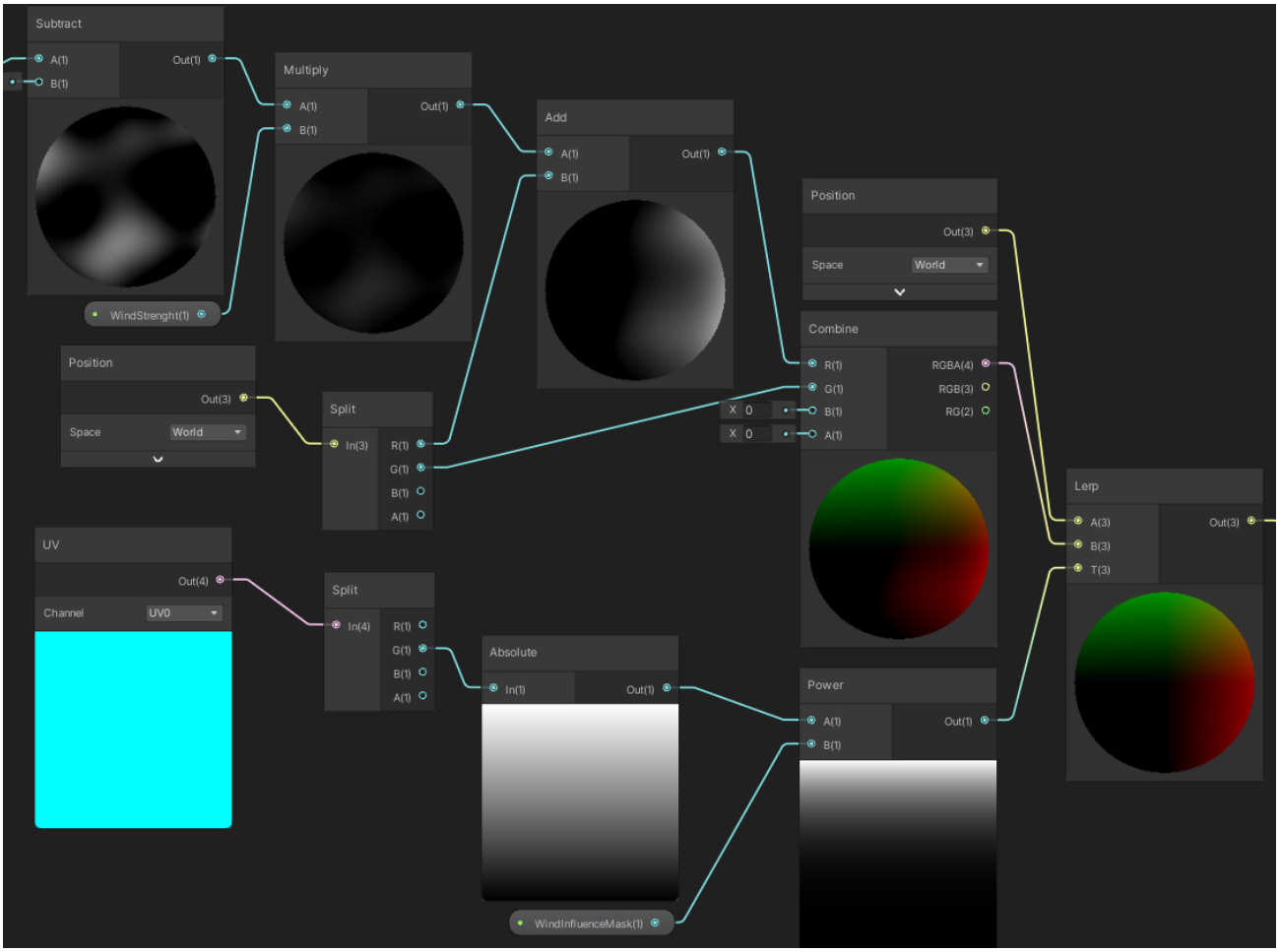

Рисунок 10.2 – Шейдер «покачивания» растительности

На рисунке 10.2 в верхней части узлов корректируется наличие шума преимущественно по x-координате спрайта. Нижняя сеть узлов образует маску уменьшая воздействие шейдера на спрайт по y-координате, далее можно скорректировать параметры усиления маски. В конце узел “Lerp” интерполирует параметры шума, позицию объекта и маску.

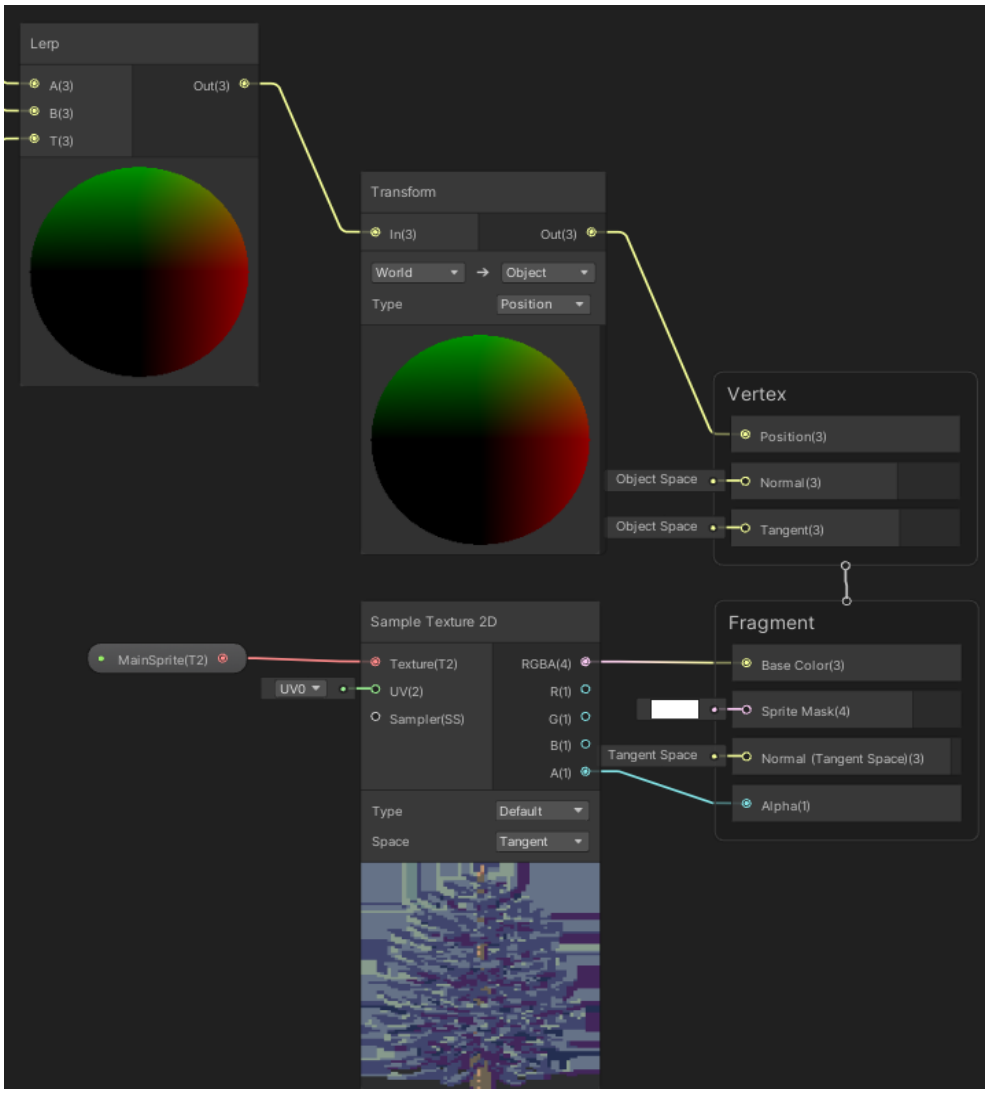

Рисунок 10.3 – Шейдер «покачивания» растительности

На рисунке 10.3 отображен конечные узлы графа, на котором в узел “Transform” поступает результат интерполяции графа и накладывается на позицию объекта. Нижняя часть графа служит для установки текстуры, цвета (RGB) и альфа канала.

4.5 НЕИГРОВЫЕ ПЕРСОНАЖИ

В игре присутствуют три неигровых персонажа изображенные на рисунке 11. Данные персонажи имеют аниматор для “Idle” состояния и не могут передвигаться, с ними может взаимодействовать игрок, чтобы начать диалог.

Рисунок 11 – Неигровые персонажи

4.6 ИГРОВОЙ ПРОЦЕСС

Игрок может свободно перемещаться по карте, ограничениями для перемещений являются коллайдеры игровых объектов и границы карты. Игрок также может взаимодействовать с некоторыми игровыми объектами, если находится в радиусе их взаимодействия. А также ставить игру на паузу, вызвав игровое меню.

Прохождение головоломок активирует событие, пропускающее персонажа вперёд, к ключу.

Сохранения происходят с помощью “порталов”, изображённых на рисунке 12, с которыми необходимо взаимодействовать, чтобы записать текущие координаты возрождения и прогресс персонажа.

Рисунок 12 – Сохранение позиции персонажа

Общение с неигровыми персонажами повествует об очередности локации с подсказками.

4.7 ФАЙЛОВАЯ СТРУКТУРА ПРОЕКТА

Для разработки приложения на платформе Unity была получена файловая структура, изображённая на рисунке 13, имеющая следующие основные каталоги файлов.

1. Animators – контроллеры и анимации для игровых объектов.
2. Audio – звуковое сопровождение для объектов.
3. Pallets – палитра спрайтов для создания окружения.
4. Prefabs – готовые объекты с настроенными параметрами для удобного клонирования и переопределения.
5. Scenes – сцены, используемые в игре.
6. Scriptable – дата контейнеры и описывающие их классы.
7. Scripts – классы, реализующие логику в игре.
8. Shaders – созданные с помощью Shader Graph шейдеры и их материалы.
9. SpritePackage – все используемые и найденные ассеты с спрайтами.

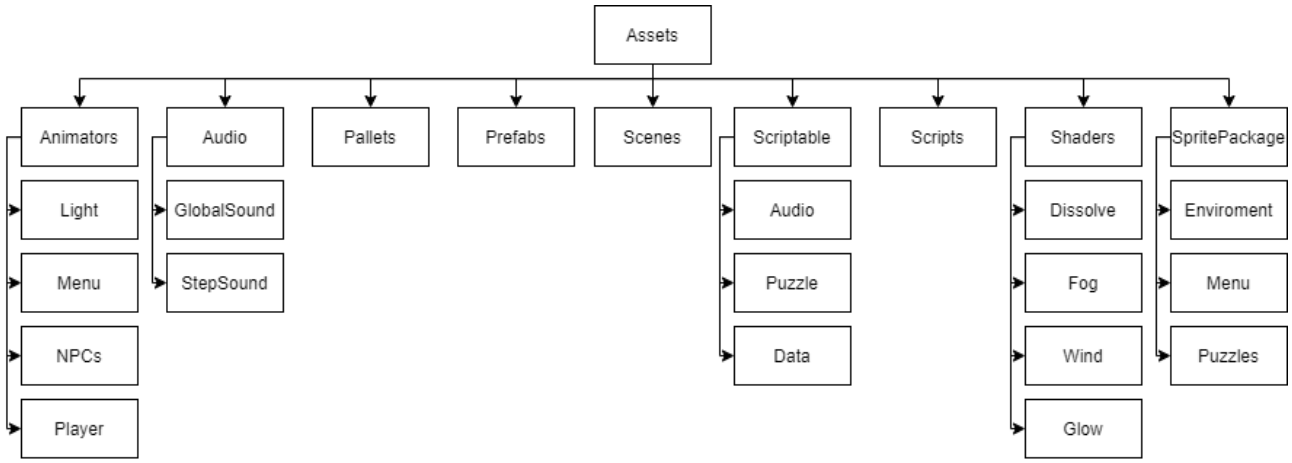

Рисунок 13 – Файловая структура проекта

В процессе реализации было использовано 28 скриптов на языке C#, перечислим их.

1. ChangeFootSound – отвечает за изменение звуков шагов, когда персонаж переходит на другую локацию, путём взаимодействия с “прозрачным” коллайдером, реализация в листинге А.10. приложения А.
2. LevelLoader – отвечает за логику работы загрузочного экрана, чтобы во время заполнения загрузочной шкалы все игровые объекты были готовы и не оказали влияния на производительность, реализация в листинге А.11. приложения А.
3. ModuleInstaller – отвечает за установку менеджера по вводу данных (Input Manager) на используемой сцене.
4. PauseMenu – отвечает за логику работы игровой паузы, во время которой игрок не может взаимодействовать с игровым миром, также устанавливается в “0” время для скорости сцены, реализация в листинге А.13. приложения А.
5. ActionBehavior – отвечает за логику взаимодействия со стороны игрока с игровыми объектами, например для того, чтобы подобрать “ключ”, реализация в листинге А.14. приложения А.

6. Death – отвечает за визуальное отображение окна смерти и возрождения персонажа на месте “портала”, в котором он был сохранён, реализация в листинге А.15. приложения А.
7. Stats – отвечает за логику работы характеристик персонажа, является абстрактным классом и несёт в себе логику максимального и текущего параметра характеристики, а также восстанавливающие параметры характеристики, реализация в листинге А.16. приложения А.
8. Health – дочерний класс Stats, определяющий здоровье персонажа.
9. InputManager – отвечает за проверку нажатия клавиши с клавиатуры и геймпада, путём переопределения реализации проверки в зависимости от настроек, реализация в листинге А.12. приложения А.
10. InteractionBehavior – отвечает за логику взаимодействия с игровыми объектами, хранит в себе общую логику и в случае взаимодействия вызывается игровое событие, выполняющее функцию у указанного объекта, реализация в листинге А.17. приложения А.
11. BaseItem – отвечает за логику существования предметов, хранящее в себе название и спрайт объекта, реализация в листинге А.18. приложения А.
12. Item – дочерний класс BaseItem.
13. MainMenu – отвечает за логику работы опций в главном меню, реализация в листинге А.19. приложения А.
14. MenuManager – отвечает за логику работы менеджера в главном меню для сохранения и загрузки данных с последующим заполнением дата-контейнеров, реализация в листинге А.20. приложения А.
15. OptionMenu – отвечает за логику работы настроек игры, реализация в листинге А.21. приложения А.

16. PlayerController – отвечает за передвижение персонажа и выставление параметров для проигрывания анимации, реализация в листинге А.22. приложения А.
17. Puzzle – отвечает за логику работы игры в паззл, который принимает в себя дата-контейнер со спрайтами паззла и далее заполняет игровое поле, реализация в листинге А.6. приложения А.
18. PuzzleBehavior – вспомогательный класс для работы игры в паззл, служит для навигации и взаимодействию по частям паззла, реализация в листинге А.7. приложения А.
19. SavePoint – отвечает за сохранение позиции игрока и вызов события у игрового менеджера для сохранения игры, реализация в листинге А.25. приложения А.
20. SliderSet – отвечает за отображение характеристик в UI, реализация в листинге А.9. приложения А.
21. Traps – отвечает за работу объектов, наносящих урон, реализация в листинге А.8. приложения А.
22. GameSettings – отвечает за описание хранимых данных для игровых настроек, таких уровень звука, разрешение, полноэкранный режим и устройство управления, реализация в листинге А.24. приложения А.
23. PlayerProgress – отвечает за описание хранимых данных для прогресса игрока, такие как собранные ключи и место возрождения, реализация в листинге А.23. приложения А.
24. GamePuzzle – отвечает за хранение паззлов, позволяя создавать несколько экземпляров с различными спрайтами, реализация в листинге А.5. приложения А.
25. AudioEvent – отвечает за проигрывание звука, несёт базовую логику, реализация в листинге А.4. приложения А.

26. SimpleAudioSteps – дочерний класс AudioEvent для звуков шагов, который хранит в себе звуки для ходьбы со случайными параметрами для проигрывания, реализация в листинге А.3. приложения А.
27. PlayerData – хранит в себе все параметры настроек и прогресса и конструктор заполнения данных, реализация в листинге А.1. приложения А.
28. SaveSystem – организовывает работу над сериализацией класса PlayerData для сохранения данных в виде файла на дисковом накопителе, а также десериализацию, для получения сохранённых данных, реализация в листинге А.2. приложения А.
29. GameManager - отвечает за логику работы менеджера в игровом мире для сохранения данных, реализация в листинге А.26. приложения А.

По реализованным скриптам составлена общая структура созданных классов. На рисунке 14.1 отображены классы, унаследованные от класса ScriptableObjects. На рисунке 14.2 отображены классы не имеющие наследования. На рисунках 14.3 - 14.4 отображены классы, унаследованные от класса MonoBehavior.

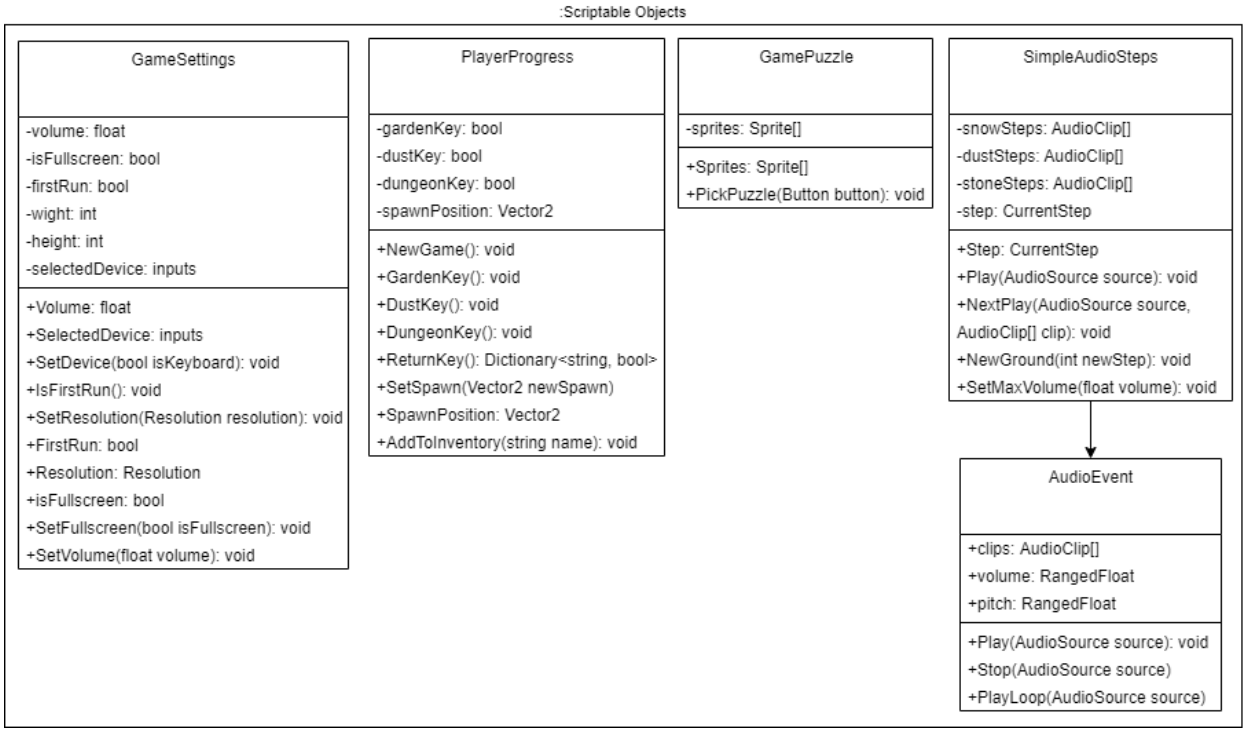

Рисунок 14.1 – Общая структура классов

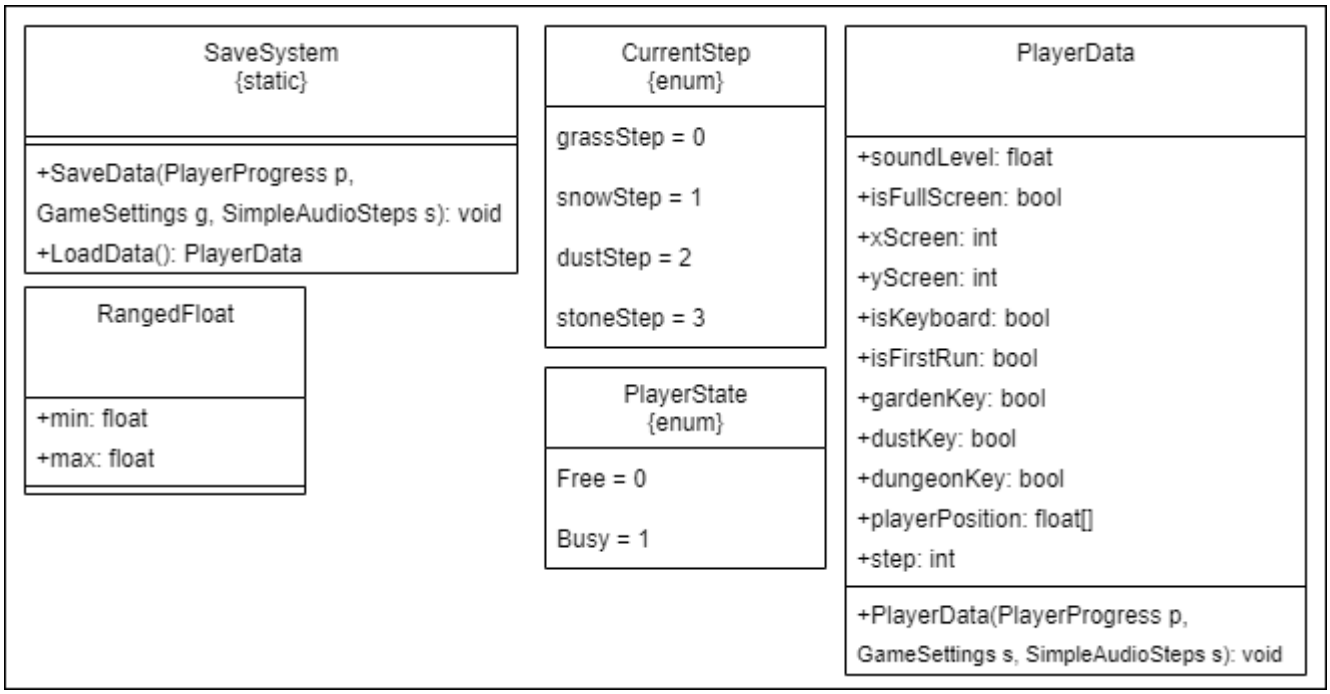

Рисунок 14.2 – Общая структура классов

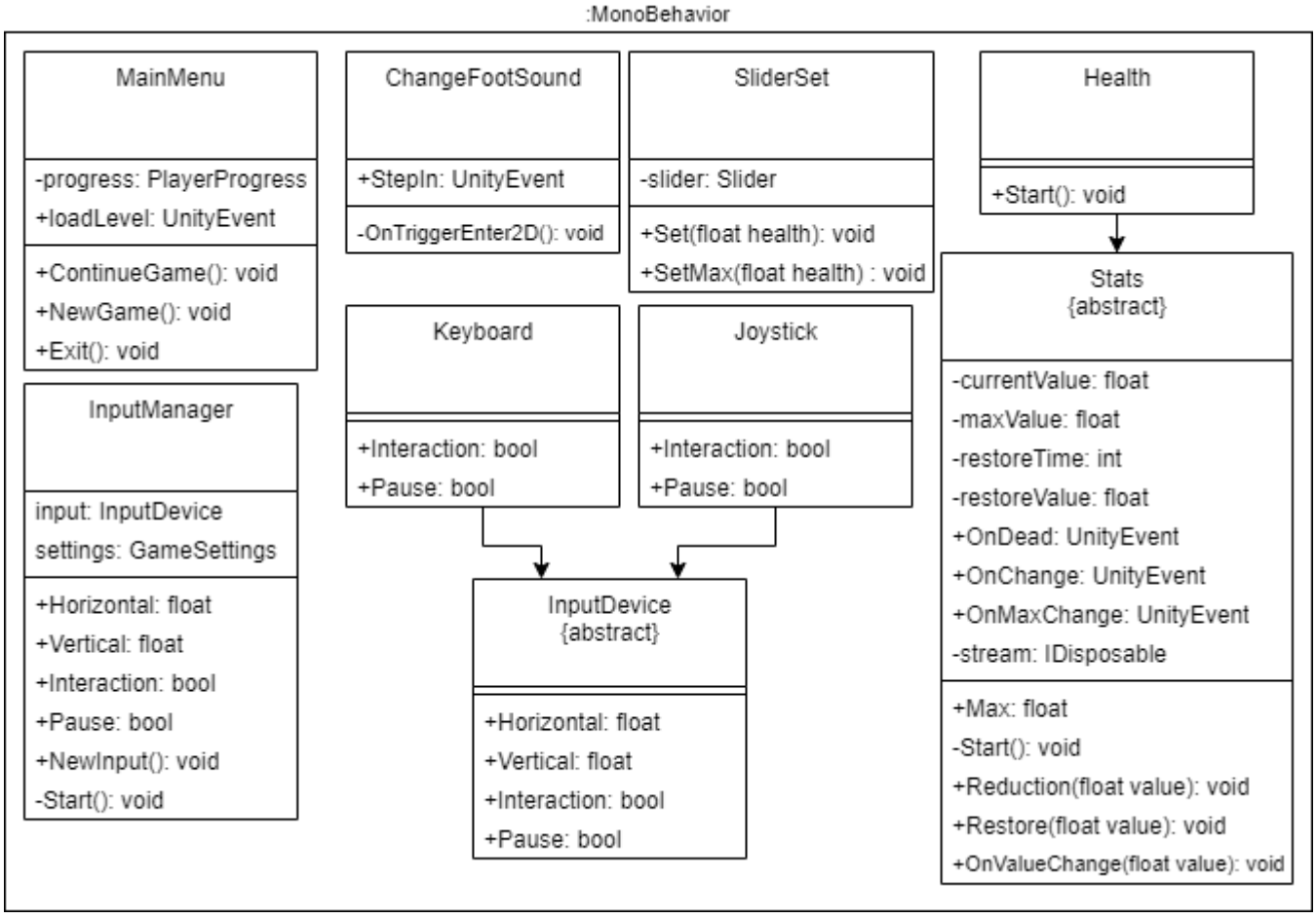

Рисунок 14.3 – Общая структура классов

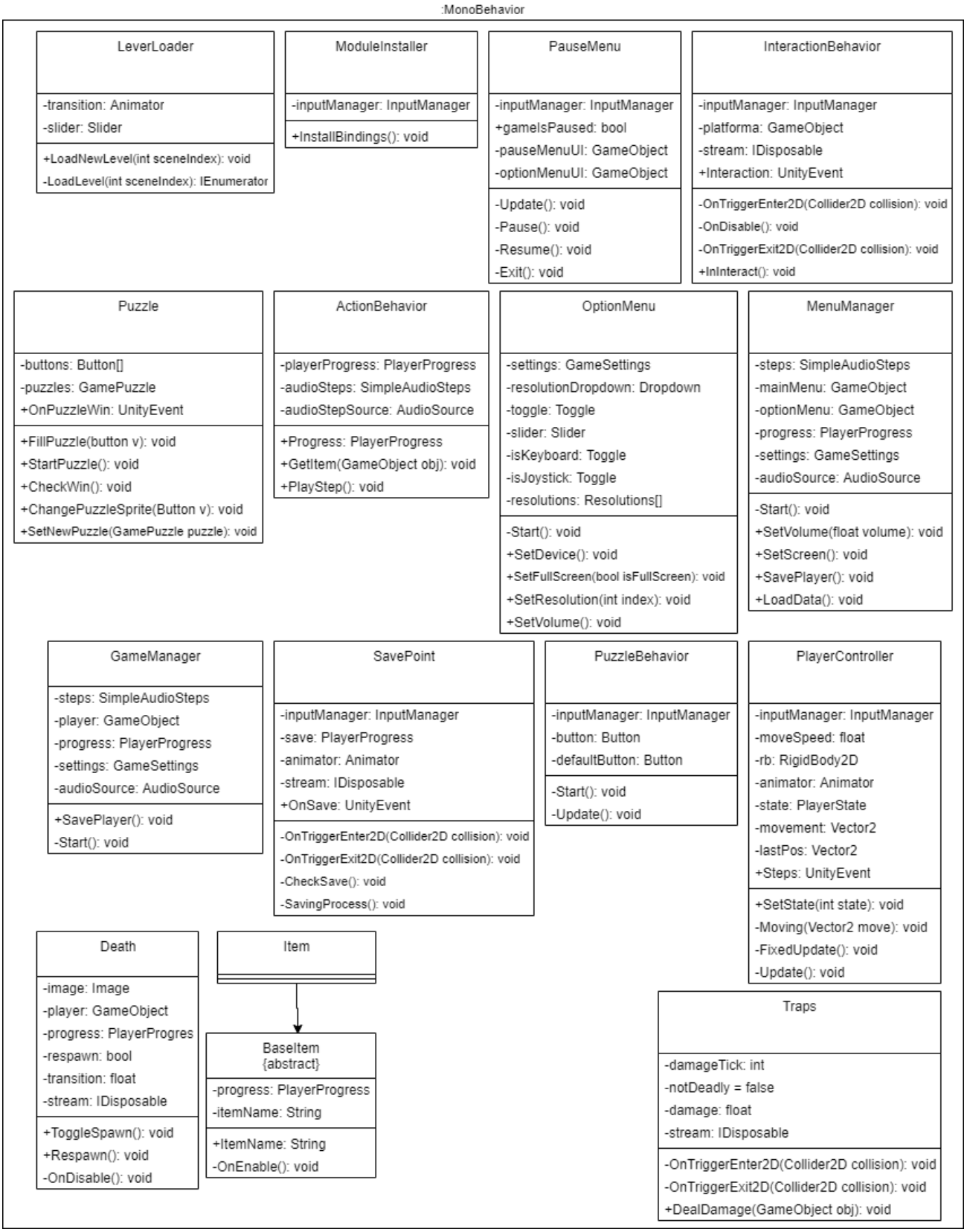

Рисунок 14.4 – Общая структура классов

5 ТЕСТИРОВАНИЕ

На этапе предварительного/окончательного выпуска продукта необходимо проводить тесты на поиск возможных багов игрового функционала и неправильной работы графики.

Для тестирования игра будет выдана группе лиц, которые согласятся испытать данную игру на различные игровые ошибки, а также дать отзыв о возможных исправлениях и нововведениях. После некоторого времени тестирования и получения отзывов, предпринять действия по исправлению ошибок. Контроль процесса тестирования и предоставления доступа к тестированию будет возлагаться на разработчика.

Тестирование проводилось в собранной версии игры для проверки всех функциональных требований, которые были описаны в техническом задании, и чтобы пользователю было комфортно играть. Тестирование проводится на персональном компьютере с помощью ввода с клавиатуры и геймпада.

Технические характеристики тестового стенда:

1. Процессор: Intel® Xeon® E5-2678 v3.
2. Оперативная память: 32 Гб.
3. Система: Windows 10 64-Bit.
4. Видеокарта: Nvidia GeForce GTS-450 1Гб.

Во время тестирования с помощью утилиты MSI Afterburner просматривался фреймрейт во время игры, изображённого на рисунке 15, в верхнем правом углу экрана. Количество фреймрейта ограничено сборкой и является значением 60, что мы и получаем по время прохождения, при этом ЦПУ и ГПУ не подвержены значительной нагрузке.

Рисунок 15 – Проверка потребляемых ресурсов

Во время тестирования также проверяется возможно смерти персонажа для просмотра правильности возрождения, для этого необходимо встать в зону, смертельную для персонажа. Результат данного теста успешен и персонаж появляется на последнем портале, где было произведено сохранение позиции.

Все диалоги с неигровыми персонажами корректно работают и позволяют разговаривать по несколько раз.

Если выйти в меню и зайти снова в игру, достигнутые ключи сохраняются у персонажа, а головоломки можно снова пройти, что означает успешное сохранение прогресса игрока.

При запуске новой игры прогресс успешно сбрасывается и ключи появляются на своих местах, для достижения которых необходимо пройти головоломки.

Во время игрового процесса можно поменять устройство ввода, и оно успешно примется, включая все используемые сцены, тоже самое касается и других настроек игры.

При выходе из игры в главном меню процесс игры успешно закрывается и не отслеживается в утилите “диспетчер задач”, что совпадает с логическим функционалом опции.

ЗАКЛЮЧЕНИЕ

В рамках представленной выпускной квалификационной работы была спроектирована и разработана компьютерная игра “Yoake” и собрана для персональных компьютеров на операционной системе Windows. Для реализации поставленной задачи были выполнены следующие шаги.

1. Определение концепта игры.
2. Ознакомление и выбор версии платформы Unity.
3. Поиск и выбор дополнительного программного обеспечения для разработки игры.
4. Изучение работы и методы создания/изменения шейдеров.
5. Поиск готовых ассетов для работы.
6. Создание скриптов, персонажей и локации.
7. Добавление звукового сопровождения.
8. Тестирование предварительного продукта.
9. Исправление багов.

Реализованный игровой проект использует шейдеры для получения лучшей картинки, преимуществом использования материала для объектов из шейдеров является то, что нет необходимости рисовать анимацию поведения для каждого спрайта, при этом можно создавать несколько материалов и управлять их параметрами, что увеличивает гибкость и удобство использования шейдеров в проекте.

Для передачи и использования данных служат дата-контейнеры, у которых нет строгой привязки к объектам на сцене и можно с лёгкостью передавать данные с одной сцены на другую, что упрощает процесс разработки.

В ходе реализации проекта были получены навыки работы с платформой Unity. Определение концепта игры и визуальной составляющей игр с используемыми

механиками повысило опыт работы в качестве гейм-дизайнера. Создание скриптов повысило навык программирования и взаимодействия с платформой Unity.

У реализованного игрового проекта имеются следующие варианты его улучшения.

1. Добавление новых головоломок и паззлов.
2. Добавление персонажей.
3. Добавление новых локаций и улучшение дизайна карты.
4. Добавление новых шейдеров.
5. Составление различных квестовых цепочек.
6. Выпуск игры на платформы цифровой дистрибуции.

БИБЛИОГРАФИЧЕСКОЙ СПИСОК

1. Unity User Manual / Страница документации // URL: https://docs.unity3d.com/Manual/index.html (Дата обращения: 28.2.2021)
2. Platform game / Страница описания // URL: https://en.wikipedia.org/wiki/Platform_game (Дата обращения: 28.2.2021)
3. Roguelike / Страница описания // URL: https://ru.wikipedia.org/wiki/Roguelike (Дата обращения: 28.2.2021)
4. Adventure game / Страница описания // URL: https://en.wikipedia.org/wiki/Adventure_game (Дата обращения: 28.2.2021)
5. Hades (video game) / Страница описания // URL: https://en.wikipedia.org/wiki/Hades_(video_game) (Дата обращения: 28.2.2021)
6. Bastion (video game) / Страница описания // URL: https://en.wikipedia.org/wiki/Bastion_(video_game) (Дата обращения: 28.2.2021)
7. Don't Starve / Страница описания // URL: https://en.wikipedia.org/wiki/Don%27t_Starve (Дата обращения: 28.2.2021)
8. Battletoads (video game) / Страница описания // URL: https://en.wikipedia.org/wiki/Battletoads_(video_game) (Дата обращения: 28.2.2021)
9. Платформа Unity / Страница описания // URL: https://unity.com/ru/products/unity-platform (Дата обращения: 28.2.2021)
10. Unreal Engine / Страница описания // URL: https://www.unrealengine.com/en-US/ (Дата обращения: 28.2.2021)
11. Язык шейдеров / Страница документации // URL: https://docs.godotengine.org/ru/stable/tutorials/shading/shading_reference/shading_language.html (Дата обращения: 28.2.2021)
12. GODOT / Страница описания // URL: https://godotengine.org/ (Дата обращения: 28.2.2021)
13. Wintermute Engine / Страница описания // URL: http://dead-code.org/home/ (Дата обращения: 28.2.2021)
14. About Wintermute Engine / Страница описания // URL: http://dead-code.org/home/index.php/features/ (Дата обращения: 28.2.2021)
15. Cocos Creator / Страница описания // URL: https://www.cocos.com/en/ (Дата обращения: 28.2.2021)
16. Shader Graph / Страница описания // URL: https://unity.com/ru/shader-graph (Дата обращения: 28.2.2021)
17. Universal Render Pipeline / Страница описания // URL: https://unity.com/srp/universal-render-pipeline (Дата обращения: 28.2.2021)

18. High Definition Render Pipeline / Страница описания // URL: https://unity.com/srp/High-Definition-Render-Pipeline (Дата обращения: 28.2.2021)
19. ScriptableObject / Страница описания // URL: https://docs.unity3d.com/Manual/class-ScriptableObject.html (Дата обращения: 28.2.2021)
20. Хокинг, Д. Unity in Action. Multiplatform game development in C# with Unity 5 / Джозеф Хокинг // Manning Publications. - https://www.manning.com/books/unity-in-action. – С. 352.
21. Шелл, Д . Геймдизайн: Как создать игру, в которую будут играть все / Джесси Шелл // Альпина Паблишер. - https://www.google.ru/books/edition/Геймдизайн_Как_создат/-jadDwAAQBAJ?hl=ru&gbpv=0. – С. 1-152.
22. Шрейер, Д. Кровь, пот и пиксели. Обратная сторона индустрии видеоигр / Джейсон Шрейер // Эксмо. - https://www.google.ru/books/edition/Кровь_пот_и_пиксели_Об/tVBjDwAAQBAJ?hl=ru&gbpv=0. – С. 1-36.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД СКРИПТОВ

Листинг А.1 – Исходный код скрипта PlayerData

```
[System.Serializable]
public class PlayerData {

    //Хранимые данные для игровых настроек
    public float soundLevel;
    public bool isFullscreen;
    public int xScreen;
    public int yScreen;
    public bool isKeyboard;
    public bool isFirstRun;

    //Хранимые данные для игрового прогресса
    public bool gardenKey;
    public bool dustKey;
    public bool dungeonKey;
    public float[] playerPosition;

    //Данные для звукового сопровождения шагов
    public int step;

    //Конструктор по сбору всех текущих параметров
    public PlayerData(PlayerProgress playerProgress, GameSettings gameSettings, SimpleAudioSteps
steps) {
        //PlayerProgress
        gardenKey = playerProgress.ReturnKey()["gardenKey"];
        dustKey = playerProgress.ReturnKey()["dustKey"];
        dungeonKey = playerProgress.ReturnKey()["dungeonKey"];
        playerPosition = new float[2];
        playerPosition[0] = playerProgress.SpawnPosition.x;
        playerPosition[1] = playerProgress.SpawnPosition.y;

        //GameSettings
        soundLevel = gameSettings.Volume;
        isFullscreen = gameSettings.IsFullscreen;
        xScreen = gameSettings.Resolution.width;
        yScreen = gameSettings.Resolution.height;
        if (gameSettings.SelectedDevice == inputs.keyboard) {
            isKeyboard = true;
        } else
            isKeyboard = false;
        isFirstRun = gameSettings.FirstRun;
```

```
        //StepSettings
        switch (steps.Step) {
            case CurrentStep.grassStep:
                step = 0;
                break;
            case CurrentStep.snowStep:
                step = 1;
                break;
            case CurrentStep.dustStep:
                step = 2;
                break;
            case CurrentStep.stoneStep:
                step = 3;
                break;
        }
    }
}
```

Листинг А.2 – Исходный код скрипта SaveSystem

```
using UnityEngine;
using System.IO;
using System.Runtime.Serialization.Formatters.Binary;

public static class SaveSystem {

    //Функция для сериализации файла сохранения для игры в бинарный вид
    public static void SaveData(PlayerProgress playerProgress, GameSettings gameSettings,
SimpleAudioSteps steps) {
        BinaryFormatter formatter = new BinaryFormatter();
        string path = Application.persistentDataPath + "/gameData.yoake";
        FileStream stream = new FileStream(path, FileMode.Create);

        PlayerData data = new PlayerData(playerProgress, gameSettings, steps);

        formatter.Serialize(stream, data);
        stream.Close();
    }

    //Функция для десериализации файла сохранения для игры из бинарного вида
    public static PlayerData LoadData() {
        string path = Application.persistentDataPath + "/gameData.yoake";
        if (File.Exists(path)) {
            BinaryFormatter formatter = new BinaryFormatter();
            FileStream stream = new FileStream(path, FileMode.Open);

            PlayerData data = formatter.Deserialize(stream) as PlayerData;
            stream.Close();
```

```
            return data;
        } else {
            return null;
        }
    }
}
```

Листинг А.3 – Исходный код скрипта SimpleAudioSteps

```
using UnityEngine;

//Данные для выбора текущего звука передвижения
public enum CurrentStep {
    grassStep = 0,
    snowStep = 1,
    dustStep = 2,
    stoneStep = 3
}

[CreateAssetMenu(menuName = "AudioEvent/Steps")]
public class SimpleAudioSteps : AudioEvent {

    [SerializeField] private AudioClip[] snowSteps;
    [SerializeField] private AudioClip[] dustSteps;
    [SerializeField] private AudioClip[] stoneSteps;
    [SerializeField] private CurrentStep step;

    public CurrentStep Step {
        get {
            return step;
        }
    }

   //Метод для выбора пула проигрываемого звука шагов
    public override void Play(AudioSource source) {
        switch(step) {
            case CurrentStep.grassStep:
                NextPlay(source, clips);
                break;
            case CurrentStep.snowStep:
                NextPlay(source, snowSteps);
                break;
            case CurrentStep.dustStep:
                NextPlay(source, dustSteps);
                break;
            case CurrentStep.stoneStep:
                NextPlay(source, stoneSteps);
                break;
```

```
        }
    }

    //Выбор случайного звука из полученного пула
    //и его проигрывание из компонента объекта
    public void NextPlay(AudioSource source, AudioClip[] clip) {
        if (clip.Length == 0)
            return;

        source.clip = clip[Random.Range(0, clip.Length)];

        source.volume = Random.Range(volume.min, volume.max);
        source.pitch = Random.Range(pitch.min, pitch.max);

        source.Play();
    }

    //Переопределение текущего звука для шагов
    public void NewGround(int newStep) {
        step = (CurrentStep)newStep;
    }

    //Установка проигрываемого звука в зависимости от настроек звука игры
    public void SetMaxVolume(float volume) {
        this.volume.max = volume;
        if (this.volume.max < 0.1f) {
            this.volume.min = 0f;
        } else {
            this.volume.min = 0.1f;
        }
    }
}
```

Листинг А.4 – Исходный код скрипта AudioEvent

```
using System;
using UnityEngine;

[Serializable]
public class RangedFloat {
    public float min;
    public float max;
}
//Родительский класс для создания конкретных классов, работающих со звуком
public abstract class AudioEvent : ScriptableObject {
    public AudioClip[] clips;

    public RangedFloat volume;
```

```
    public RangedFloat pitch;

    public abstract void Play(AudioSource source);

    public void Stop(AudioSource source) {
        source.Stop();
    }

    public void PlayLoop(AudioSource source) {
        Play(source);

        source.loop = true;
    }
}
```

Листинг А.5 – Исходный код скрипта GamePuzzle

```
using UnityEngine;
using UnityEngine.UI;
using Random = UnityEngine.Random;

//Класс по заполнению игрового поля паззла
[CreateAssetMenu(menuName = "Content/GamePuzzle")]
public class GamePuzzle : ScriptableObject {
    [SerializeField] private Sprite[] sprites;

    public Sprite[] Sprites {
        get {
            return sprites;
        }
    }

    //Выбор случайного спрайта из паззла для поля
    public void PickPuzzle(Button button) {
        if (sprites.Length == 0)
            return;
        button.image.sprite = sprites[Random.Range(0, sprites.Length)];
    }
}
```

Листинг А.6 – Исходный код скрипта Puzzle

```
using System;
using UnityEngine;
using UnityEngine.Events;
using UnityEngine.UI;

public class Puzzle : MonoBehaviour {
```

```
[SerializeField] private Button[] buttons;
[SerializeField] private GamePuzzle puzzles;

public UnityEvent OnPuzzleWin;

//Заполнения одного поля паззла
public void FillPuzzle(Button v) {
    puzzles.PickPuzzle(v);
}

//Прохождение по всем полям паззла и запуск заполнения
public void StartPuzzle() {
    if (buttons.Length == 0)
        return;
    foreach (var v in buttons) {
        FillPuzzle(v);
    }
    buttons[0].Select();
}

//Проверка правильно составленного паззла
public void CheckWin() {
    int i = 0;
    int counter = 0;
    foreach (var v in buttons) {
        if (v.image.sprite == puzzles.Sprites[i++]) {
            counter++;
        }
    }
    if (counter == buttons.Length) {
        OnPuzzleWin?.Invoke();
    }
}

//Изменение спрайта паззла при взаимодействии
public void ChangePuzzleSprite(Button button) {
    int i = Array.IndexOf(puzzles.Sprites, button.image.sprite);
    if (i == puzzles.Sprites.Length - 1)
        i = -1;
    button.image.sprite = puzzles.Sprites[++i];
}

//Переопределение картинки паззла для игры
public void SetNewPuzzle(GamePuzzle puzzle) {
    puzzles = puzzle;
    StartPuzzle();
}
```

```
}
```

Листинг А.7 – Исходный код скрипта PuzzleBehavior

```
using UnityEngine;
using Zenject;
using UnityEngine.UI;
using UnityEngine.EventSystems;

public class PuzzleBehavior : MonoBehaviour {

    [Inject]
    private InputManager inputManager;

    private Button button;
    [SerializeField] private Button defaulButton;

    // Start is called before the first frame update
    void Start() {
        button = gameObject.GetComponent<Button>();
    }

    // Update is called once per frame
    //Проверка нажатия на элемент паззла и сохранение навигации в поле паззла
    void Update() {
        try {
            if (EventSystem.current.currentSelectedGameObject.GetComponent<Button>() == button &&
inputManager.Interaction) {
                button.onClick?.Invoke();
            }
        } catch {
            defaulButton.Select();
        }
    }

}
```

Листинг А.8 – Исходный код скрипта Traps

```
using MyBox;
using System;
using UniRx;
using UnityEngine;

public class Traps : MonoBehaviour {
    [SerializeField] private int damageTick;
    [SerializeField] private bool notDeadly = false;

    [ConditionalField(nameof(notDeadly))]
```

```
    [SerializeField] protected float damage;

    [SerializeField]
    private IDisposable stream;

    //При входе в зону действия вызывается метод нанесения урона
    private void OnTriggerEnter2D(Collider2D collision) {
        var gameObj = collision.gameObject;

        if (stream == null && gameObj.CompareTag("Player")) {
            stream = Observable.Interval(TimeSpan.FromSeconds(damageTick)).Subscribe(_ => {
                DealDamage(gameObj);
            });
        }
    }

    //Нанесение количественного урона или смертельного объекту
    public void DealDamage(GameObject obj) {
        var health = obj.transform.root.GetComponent<Health>();
        if (!notDeadly) {
            health.OnDead?.Invoke();
            stream?.Dispose();
            stream = null;
        } else {
            health.Reduction(damage);
        }
    }

    //Прекращение вызова метода по нанесению урона при выходе из зоны действия
    private void OnTriggerExit2D(Collider2D collision) {
        stream?.Dispose();
        stream = null;
    }
}
```

Листинг А.9 – Исходный код скрипта SliderSet

```
using UnityEngine;
using UnityEngine.UI;

public class SliderSet : MonoBehaviour
{
    [SerializeField] private Slider slider;

   //Выставление текущего параметра для слайдера
    public void Set(float health) {
        slider.value = health;
    }
```

```
    //Выставление максимального параметра для слайдера
    public void SetMax(float health) {
        slider.maxValue = health;
    }
}
```

Листинг А.10 – Исходный код скрипта ChangeFootSound

```
using UnityEngine;
using UnityEngine.Events;

public class ChangeFootSound : MonoBehaviour {

    public UnityEvent StepIn;

    //По вхождению в зону действия происходит вызов события
    //Событие вызывает смену проигрываемого пула звуков при ходьбе
    private void OnTriggerEnter2D(Collider2D collision) {
        var gameObj = collision.gameObject;
        if (gameObj.CompareTag("Player")) {
            StepIn?.Invoke();
        }
    }
}
```

Листинг А.11 – Исходный код скрипта LevelLoader

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

public class LevelLoader : MonoBehaviour
{
    [SerializeField] private int sceneIndex;
    [SerializeField] private Animator transition;
    [SerializeField] private Slider slider;

    //Метод вызывающий коротину работающая на каждый кадр
    public void LoadNewLevel(int sceneIndex) {
        StartCoroutine(LoadLevel(sceneIndex));
    }

    //Коротина, во время которой подгружаются все объекты переходной сцены
    IEnumerator LoadLevel(int sceneIndex) {
        transition.SetTrigger("Start");
        AsyncOperation operation = SceneManager.LoadSceneAsync(sceneIndex);
        while (!operation.isDone) {
            float progress = Mathf.Clamp01(operation.progress / .9f);
```

```
            Debug.Log(progress);
            slider.value = progress;
            yield return null;
        }
    }
}
```

Листинг А.12 – Исходный код скрипта InputManager

```
using UnityEngine;

//Перечисление для определения устройства ввода
public enum inputs {
    keyboard = 0,
    joystick = 1
}

//Родительский класс с общими методами по обработке нажатия клавиш
public abstract class InputThings {
    //Получение значения о горизонтальном перемещении
    public float Horizontal {
        get {
            return Input.GetAxis("Horizontal");
        }
    }
    //Получение значения о вертикальном перемещении
    public float Vertical {
        get {
            return Input.GetAxis("Vertical");
        }
    }
    //Абстрактный метод по получению информации о нажатии клавиши для взаимодействия
    public abstract bool Interaction {
        get;
    }
    //Абстрактный метод по получению информации о нажатии клавиши для паузы
    public abstract bool Pause {
        get;
    }
}

//Дочерний класс с переопределением методов для работы с клавиатурой
public class Keyboard : InputThings {
    public override bool Interaction {
        get {
            return Input.GetKeyDown(KeyCode.E);
        }
```

```
    }

    public override bool Pause {
        get {
            return Input.GetKeyDown(KeyCode.Escape);
        }
    }
}
//Дочерний класс с переопределением методов для работы с геймпадом
public class Joystick : InputThings {
    public override bool Interaction {
        get {
            return Input.GetKey("joystick button 0");
        }
    }
    public override bool Pause {
        get {
            return Input.GetKey("joystick button 1");
        }
    }
}

//Класс по передачи данных о вводе с клавиатуры или геймпада
public class InputManager : MonoBehaviour {
    private InputThings input;
    [SerializeField] private GameSettings settings;

    public float Horizontal {
        get {
            return input.Horizontal;
        }
    }
    public float Vertical {
        get {
            return input.Vertical;
        }
    }
    public bool Interaction {
        get {
            return input.Interaction;
        }
    }
    public bool Pause {
        get {
            return input.Pause;
        }
    }
```

```
    //Переопределение устройства ввода
    public void NewInput() {
        switch (settings.SelectedDevice) {
            case inputs.keyboard:
                input = new Keyboard();
                break;
            case inputs.joystick:
                input = new Joystick();
                break;
        }
    }
    //Функция, срабатывающая перед всеми методами обновления
    private void Start() {
        NewInput();
    }
}
```

Листинг А.13 – Исходный код скрипта PauseMenu

```
using UnityEngine;
using UnityEngine.SceneManagement;
using Zenject;

public class PauseMenu : MonoBehaviour {
    [Inject]
    private InputManager inputManager;

    public static bool GameIsPaused = false;

    [SerializeField] private GameObject pauseMenuUI;
    [SerializeField] private GameObject optionMenuUI;

    // Update is called once per frame
    //Проверка снятия или установка игры на паузу
    void Update()
    {
        if (inputManager.Pause) {
            if (GameIsPaused) {
                Resume();
                optionMenuUI.SetActive(false);
            } else {
                Pause();
            }
        }
    }

   //Включение окна меню паузы и “заморозка” игрового процесса
    private void Pause() {
```

```
        pauseMenuUI.SetActive(true);
        Time.timeScale = 0f;
        GameIsPaused = true;

    }

    //Выключение окна меню паузы и “разморозка” игрового процесса
    public void Resume() {
        pauseMenuUI.SetActive(false);
        Time.timeScale = 1f;
        GameIsPaused = false;
    }

    //Переход на сцену “Главного меню”
    public void Exit() {
        Time.timeScale = 1f;
        SceneManager.LoadScene("Menu");
    }
}
```

Листинг А.14 – Исходный код скрипта ActionBehavior

```
using UnityEngine;

public class ActionsBehaviour : MonoBehaviour
{
    [SerializeField] private PlayerProgress playerProgress;
    [SerializeField] private SimpleAudioSteps audioSteps;
    [SerializeField] private AudioSource audioStepSource;

    public PlayerProgress Progress {
        get {
            return playerProgress;
        }
    }

    //Подбор игрового предмета и его удаление на сцене
    public void GetItem(GameObject obj) {
        var itemName = obj.GetComponent<Item>().ItemName;
        var keys = playerProgress.ReturnKey();
        if (keys.ContainsKey(itemName)) {
            playerProgress.AddToInventory(itemName);
            Destroy(obj);
        }
    }

    //Проигрывание звука ходьбы
    public void PlayStep() {
```

```
        if (audioStepSource.isPlaying == false) {
            audioSteps.Play(audioStepSource);
        }
    }
}
```

Листинг А.15 – Исходный код скрипта Death

```
using System;
using UniRx;
using UnityEngine;
using UnityEngine.UI;

public class Death : MonoBehaviour {

    [SerializeField] private Image image;
    [SerializeField] private GameObject player;
    [SerializeField] private PlayerProgress progress;

    private bool respawn = false;
    private float transition = 0f;

    private IDisposable stream;

    //Метод по запуску “экрана смерти” и “возрождения” персонажа
    public void ToggleRespawn() {
        gameObject.SetActive(true);
        respawn = false;
        stream = Observable.EveryUpdate().Subscribe(_ => {
            Respawn();
        });
    }

    //Выполняет роль анимации “экрана смерти” и “возрождения” персонажа
    public void Respawn() {
        if (!respawn) {
            transition += Time.fixedDeltaTime * 0.1f;
            image.color = Color.Lerp(new Color(0, 0, 0, 0), Color.black, transition);
            if (image.color == Color.black) {
                respawn = true;
                player.transform.position = progress.SpawnPosition;
                var restore = player.transform.root.GetComponent<Health>();
                restore.Restore(restore.Max);
                transition = 0;
            }
        } else {
```

```
            transition += Time.fixedDeltaTime * 0.1f;
            image.color = Color.Lerp(image.color, new Color(0, 0, 0, 0), transition);
        }
    }

    private void OnDisable() {
        stream?.Dispose();
        stream = null;
    }
}
```

Листинг А.16 – Исходный код скрипта Stats

```
using System;
using UniRx;
using UnityEngine;
using UnityEngine.Events;
using UnityEngine.UI;

public abstract class Stats : MonoBehaviour {

    [SerializeField] private float currentValue;
    [SerializeField] private float maxValue;
    [SerializeField] private int restoreTime;
    [SerializeField] private float restoreValue;

    public UnityEvent OnDead;
    public UnityEvent<float> OnChange;
    public UnityEvent<float> OnMaxChange;

    private IDisposable stream;

    public float Max {
        get {
            return maxValue;
        }
    }

    //Выставление начальных параметров для интерфейса
    // и подпись на изменение значений параметра
    public virtual void Start() {
        OnMaxChange.Invoke(maxValue);
        OnChange.Invoke(currentValue);
        OnChange.AddListener(OnValueChange);
        if (stream == null && currentValue < maxValue) {
```

```
            stream = Observable.Interval(TimeSpan.FromSeconds(restoreTime)).Subscribe(_ => {
                Restore(restoreValue);
            });
        }
    }

    //Уменьшение параметра на значение
    public void Reduction(float value) {
        currentValue -= value;
        if (currentValue < 0.1f) {
            OnDead?.Invoke();
        }
        stream?.Dispose();
        stream = null;
        OnChange?.Invoke(currentValue);
    }

    //Восстановление параметра на значение
    public void Restore(float value) {
        currentValue += value;
        if (currentValue >= maxValue) {
            currentValue = maxValue;
            stream?.Dispose();
            stream = null;
        }
        OnChange?.Invoke(currentValue);
    }

    //При изменении значения будет запуск функции через промежуток времени
    //по восстановлению параметра
    public void OnValueChange(float value) {
        if (stream == null && currentValue < maxValue) {
            stream = Observable.Interval(TimeSpan.FromSeconds(restoreTime)).Subscribe(_ => {
                Restore(restoreValue);
            });
        }
    }
}
```

Листинг А.17 – Исходный код скрипта InteractionBehavior

```
using System;
using UniRx;
using UnityEngine;
using UnityEngine.Events;
using Zenject;
```

```
public class InteractionBehavior : MonoBehaviour {
    [Inject]
    private InputManager inputManager;

    [SerializeField] private GameObject platforma;

    private IDisposable stream;

    public UnityEvent Interaction;

    //При вхождении игрока в зону действия запускается проверка нажатия клавиши
    //взаимодействия. При её нажатии вызывается событие
    private void OnTriggerEnter2D(Collider2D collision) {
        var gameObj = collision.gameObject;
        if (gameObj.CompareTag("Player")) {

            var clickStream = Observable.EveryUpdate()
            .Where(_ => inputManager.Interaction);

            stream = clickStream.Subscribe(xs => Interaction?.Invoke());
        }
    }
    private void OnDisable() {
        stream?.Dispose();
    }
    private void OnTriggerExit2D(Collider2D collision) {
        stream?.Dispose();
        stream = null;
    }

    public void InInteract() {
        gameObject.transform.SetParent(platforma.transform);
    }
}
```

Листинг А.18 – Исходный код скрипта BaseItem

```
using UnityEngine;

public abstract class BaseItem : MonoBehaviour {
    [SerializeField] private MapHolder mapHolder;
    [SerializeField] private PlayerProgress progress;
    [SerializeField] private string itemName;

    public string ItemName {
        get {
            return itemName;
        }
```

```
    }

    //Если предмет уже находится у персонажа, то он будет удалён со сцены
    private void OnEnable() {
        var checkItem = progress.ReturnKey();
        if (checkItem.ContainsKey(gameObject.tag) && checkItem[gameObject.tag] == true) {
            Destroy(gameObject);
        }
    }
```

Листинг А.19 – Исходный код скрипта MainMenu

```
using UnityEngine;
using UnityEngine.Events;

public class MainMenu : MonoBehaviour {
    [SerializeField] private PlayerProgress progress;

    public UnityEvent loadLevel;

    //Срабатывание события при нажатии опции продолжить игру
    public void ContinueGame() {
        loadLevel?.Invoke();
    }

    //Сброс игрового прогресса при начале новой игры
    public void NewGame() {
        progress.NewGame();
        loadLevel?.Invoke();
    }

    //Закрытие приложения
    public void Exit() {
        Debug.Log("Quited");
        Application.Quit();
    }
}
```

Листинг А.20 – Исходный код скрипта MenuManager

```
using UnityEngine;

public class MenuManager : MonoBehaviour {

    [SerializeField] private GameSettings settings;
    [SerializeField] private AudioSource audioSource;
    [SerializeField] private GameObject mainMenu;
    [SerializeField] private GameObject optionMenu;
```

```
[SerializeField] private PlayerProgress progress;
[SerializeField] private SimpleAudioSteps steps;

//При старте происходит загрузка данных с файла сохранения
//и начальная установка отображение меню
void Start() {
    LoadData();
    mainMenu.SetActive(true);
    optionMenu.SetActive(false);
    Setscreen();
}

//Установка значения звука
public void SetVolume(float volume) {
    audioSource.volume = volume;
}


//Установка параметров из файла сохранения
//При первом запуске устанавливаются стандартные значения
public void Setscreen() {
    Screen.fullScreen = settings.IsFullscreen;
    if (settings.FirstRun) {
        var resolution = settings.Resolution;
        Screen.SetResolution(resolution.width, resolution.height, settings.IsFullscreen);
        audioSource.volume = settings.Volume;
    } else {
        settings.IsFirstRun();
        audioSource.volume = 0.5f;
        settings.SetVolume(audioSource.volume);
        var resolution = Screen.currentResolution;
        settings.SetResolution(resolution);
        Screen.SetResolution(resolution.width, resolution.height, settings.IsFullscreen);
        settings.SetDevice(true);
    }
}

//Метод для вызова сохранения текущих параметров в файл
public void SavePlayer() {
    SaveSystem.SaveData(progress, settings, steps);
}

//Метод по установке полученных значений из файла сохранения
//в необходимые дата-контейнеры
public void LoadData() {
    //set playerProgress
    PlayerData data = SaveSystem.LoadData();
```

```
        Vector2 position = new Vector2(data.playerPosition[0], data.playerPosition[1]);
        progress.SetSpawn(position);
        if (data.gardenKey) {
            progress.GardenKey();
        }
        if (data.dustKey) {
            progress.DustKey();
        }
        if (data.dungeonKey) {
            progress.DungeonKey();
        }
        //set gameSettings
        settings.SetVolume(data.soundLevel);
        settings.SetFullscreen(data.isFullscreen);
        settings.SetResolution(new Resolution { width = data.xScreen, height = data.yScreen });
        settings.SetDevice(data.isKeyboard);
        if (data.isFirstRun) {
            settings.IsFirstRun();
        }

        //set Steps
        steps.NewGround(data.step);
    }
}
```

Листинг А.21 – Исходный код скрипта OptionMenu

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

public class OptionMenu : MonoBehaviour {
    [SerializeField] private GameSettings settings;
    [SerializeField] private Dropdown resolutionDropdown;
    [SerializeField] private Toggle toggle;
    [SerializeField] private Slider slider;
    [SerializeField] private Toggle isKeyboard;
    [SerializeField] private Toggle isJoystick;

    private Resolution[] resolutions;

    //Настраивается окно меню настроек, а также со всеми доступными разрешениями экрана
    private void Start() {
        SetDevice();
        resolutions = Screen.resolutions;
        resolutionDropdown.ClearOptions();
        int currentResolutionIndex = 0;
        List<string> options = new List<string>();
```

```
    for (int i = 0; i < resolutions.Length; i++) {
        string option = resolutions[i].width + " x " + resolutions[i].height;
        options.Add(option);
        if (settings.FirstRun) {
            if (resolutions[i].width == settings.Resolution.width &&
            resolutions[i].height == settings.Resolution.height) {
                currentResolutionIndex = i;
            }
        }
        else if(resolutions[i].width == Screen.currentResolution.width &&
          resolutions[i].height == Screen.currentResolution.height) {
            currentResolutionIndex = i;
            SetResolution(i);
        }
    }
    resolutionDropdown.AddOptions(options);
    resolutionDropdown.value = currentResolutionIndex;
    resolutionDropdown.RefreshShownValue();
    toggle.isOn = settings.IsFullscreen;
    SetVolume();
}

//Управление группой переключателей
public void SetDevice() {
    switch(settings.SelectedDevice) {
        case inputs.keyboard:
            isKeyboard.isOn = true;
            break;
        case inputs.joystick:
            isJoystick.isOn = true;
            break;
    }
}

//Выставление игры в полноэкранный режим
public void SetFullscreen(bool isFullscreen) {
    Screen.fullScreen = isFullscreen;
    settings.SetFullscreen(isFullscreen);
}

//Установка для игры разрешения экрана
public void SetResolution(int index) {
    settings.SetResolution(resolutions[index]);
    Screen.SetResolution(resolutions[index].width, resolutions[index].width, settings.IsFullscreen);
}

//Установка уровня громкости звука
```

```
    public void SetVolume() {
        slider.value = settings.Volume;
    }
}
```

Листинг А.22 – Исходный код скрипта PlayerController

```
using UnityEngine;
using UnityEngine.Events;
using Zenject;

//Перечисления для состояний
public enum PlayerState {
    Free = 0,
    busy = 1
}

public class PlayerController : MonoBehaviour
{
    [Inject]
    private InputManager inputManager;

    [SerializeField] private float moveSpeed = 5f;
    [SerializeField] private Rigidbody2D rb;
    [SerializeField] private Animator animator;
    [SerializeField] private PlayerState state;

    Vector2 movement;
    Vector2 lastPos;

    public UnityEvent Steps;

    //Установка нового состояния
    public void SetState(int state) {
        if (this.state != (PlayerState)state) {
            this.state = (PlayerState)state;
        }
    }

    //Управление аниматором
    private void Update() {
        animator.SetFloat("Speed", movement.magnitude);
        if (movement.magnitude < 0.1f) {
            animator.SetFloat("Vertical", lastPos.y);
            animator.SetFloat("Horizontal", lastPos.x);
        } else {
            animator.SetFloat("Vertical", movement.y);
            animator.SetFloat("Horizontal", movement.x);
```

```
            lastPos = movement;
        }
    }

    //Передвижение объекта
    private void Moving(Vector2 move) {
        move = Vector2.ClampMagnitude(move, 1);
        Vector2 newPos = rb.position + move * moveSpeed * Time.fixedDeltaTime;
        rb.MovePosition(newPos);
        if (move.magnitude > .5f) {
            Steps?.Invoke();
        }
    }

    //Получение значений для вектора движения
    private void FixedUpdate() {
        if (state == PlayerState.Free) {
            movement = new Vector2(inputManager.Horizontal, inputManager.Vertical);
            Moving(movement);
        }
    }
}
```

Листинг А.23 – Исходный код скрипта PlayerProgress

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

[CreateAssetMenu(menuName = "Content/PlayerProgress")]
public class PlayerProgress : ScriptableObject {
    //If player complete GameZones
    [SerializeField] private bool gardenKey;
    [SerializeField] private bool dustKey;
    [SerializeField] private bool dungeonKey;

    //For save points or die moments
    [SerializeField] private Vector2 spawnPosition;

    //Сброс прогресса
    public void NewGame() {
        gardenKey = false;
        dustKey = false;
        dungeonKey = false;
        spawnPosition = new Vector2(0, 0);
    }

    public void GardenKey() {
        gardenKey = true;
```

```
    }

    public void DustKey() {
        dustKey = true;
    }

    public void DungeonKey() {
        dungeonKey = true;
    }

    //Получение словаря текущего прогресса
    public Dictionary<string, bool> ReturnKey() {
        var list = new Dictionary<string, bool>();
        list.Add("gardenKey", gardenKey);
        list.Add("dustKey", dustKey);
        list.Add("dungeonKey", dungeonKey);
        return list;
    }

   //Установка новой позиции для возрождения
    public void SetSpawn(Vector2 newSpawn) {
        spawnPosition = newSpawn;
    }

    public Vector2 SpawnPosition {
        get {
            return spawnPosition;
        }
    }

    //Установка значения при получении предмета
    public void AddToInventory(string name) {
        if (name == "gardenKey") {
            gardenKey = true;
        }
        if (name == "dustKey") {
            dustKey = true;
        }
        if (name == "dungeonKey") {
            dungeonKey = true;
        }
    }
}
```

Листинг А.24 – Исходный код скрипта GameSettings

```
using MyBox;
using UnityEngine;
```

```
[CreateAssetMenu(menuName = "Content/GameSettings")]
public class GameSettings : ScriptableObject {
    [SerializeField] private float volume;
    [SerializeField] private bool isFullscreen;
    [SerializeField] private bool firstRun = false;
    [SerializeField] private inputs selectedDevice;

    [ConditionalField(nameof(firstRun))]
    [SerializeField] private int wight;
    [ConditionalField(nameof(firstRun))]
    [SerializeField] private int height;

    public float Volume {
        get {
            return volume;
        }
    }
    public inputs SelectedDevice {
        get {
            return selectedDevice;
        }
    }

    //Установка устройства ввода
    public void SetDevice(bool isKeyboard) {
        if (isKeyboard) {
            selectedDevice = inputs.keyboard;
        } else {
            selectedDevice = inputs.joystick;
        }
    }

    //Первый запуск
    public void IsFirstRun() {
        firstRun = true;
    }

    //Установка разрешения экрана
    public void SetResolution(Resolution resolution) {
        wight = resolution.width;
        height = resolution.height;
    }
    public bool FirstRun {
        get {
            return firstRun;
        }
```

```
    }
    public Resolution Resolution {
        get {
            Resolution resolution = new Resolution {
                width = wight,
                height = height
            };
            return resolution;
        }
    }
    public bool IsFullscreen {
        get {
            return isFullscreen;
        }
    }

    //Установка полноэкранного режима
    public void SetFullscreen(bool isFullscreen) {
        this.isFullscreen = isFullscreen;
    }

    //Установка уровня громкости звука
    public void SetVolume(float volume) {
        this.volume = volume;
    }
}
```

Листинг А.25 – Исходный код скрипта SavePoint

```
using UnityEngine;
using UniRx;
using System;
using Zenject;
using UnityEngine.Events;

public class SavePoint : MonoBehaviour {
    [Inject]
    private InputManager inputManager;

    [SerializeField]
    private PlayerProgress save;
    [SerializeField]
    private Animator animator;
    private IDisposable stream;
    public UnityEvent OnSave;

    //Обработка взаимодействия персонажа с объектом в зоне действия
    private void OnTriggerEnter2D(Collider2D collision) {
```

```
        var gameObj = collision.gameObject;
        if (gameObj.CompareTag("Player")) {
            var clickStream = Observable.EveryUpdate()
            .Where(_ => inputManager.Interaction);

            stream = clickStream.Subscribe(xs => CheckSave());
        }
    }
    private void OnTriggerExit2D(Collider2D collision) {
        stream.Dispose();
    }

    //Сохранение прогресса
    private void CheckSave() {
        if (inputManager.Interaction) {
            save.SetSpawn(gameObject.transform.position);
            SavingProcess();
            OnSave?.Invoke();
        }
    }
    //Вызов анимации сохранения
    private void SavingProcess() {
        animator.SetTrigger("Saving");
    }
}
```

Листинг А.26 – Исходный код скрипта GameManager

```
using UnityEngine;

public class GameManager : MonoBehaviour {
    [SerializeField] private GameObject player;
    [SerializeField] private PlayerProgress progress;
    [SerializeField] private GameSettings settings;
    [SerializeField] private AudioSource audioSource;
    [SerializeField] private SimpleAudioSteps steps;

    // Перемещение персонажа по точку сохранение и установка громкости звука
    void Start() {
        player.transform.position = progress.SpawnPosition;
        audioSource.volume = settings.Volume;
    }

    //Метод для вызова сохранения текущих параметров в файл
    public void SavePlayer() {
        SaveSystem.SaveData(progress, settings, steps);
    }
}
```