МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Минский КОЛЛЕДЖ предпринимательства

Отделение «Программное обеспечение информационных технологий»

**Игра «Королевство Вивальди»**

Техническое задание к курсовому проекту по учебной дисциплине

«Конструирование программ и языки программирования»

МКП КП 2-40 01 01 33 006 ПЗ

**Автор проекта**

Учащегося 3 курса группы П-31 /Р.Т.Винничук/

**Руководитель**

Преподаватель /А.В.Чеботарёв /

Минск, 2019

**МИНСКИЙ КОЛЛЕДЖ ПРЕДПРИНИМАТЕЛЬСТВА**

Отделение

«Программное обеспечение информационных технологий»

Специальность

«Программное обеспечение информационных технологий»

2-40 01 01

УТВЕРЖДАЮ

Председатель цикловой комиссии

\_\_\_\_\_\_\_\_\_\_/\_Ю.А.Доманова\_\_\_\_\_\_\_\_\_\_\_\_/

**ЗАДАНИЕ**

**на курсовой проект**

Учащемуся\_Винничуку Родиону Тарасовичу\_

курса\_\_\_3\_\_\_\_ группы\_\_\_\_\_П-31\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

по учебной дисциплине \_Конструирование программ и языки программирования\_\_\_\_\_\_

Тема курсового проекта \_Игра «Королевство Вивальди»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Исходные данные:\_действия пользователя в процессе игры\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Состав проекта**

**Пояснительная записка**

Содержание разделов Срок выполнения

Введение 20.03 \_\_\_\_\_\_\_\_\_\_\_\_

Постановка задачи\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 29.03\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Вычислительная система\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 03.04\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Проектирование \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 16.04\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Описание программа\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 23.04\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Отладка и испытание программы\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_30.04\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Описание применения\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 14.05\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Заключение\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 15.05\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Графическая часть проекта**

Диаграмма классов – плакат\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_28.05\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Диаграмма сущность–связь плакат \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_28.05\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Дата выдачи «20»\_\_\_\_03\_\_\_\_\_\_2019 г.

Срок завершения «28» \_\_05\_ 2019 г.

**Преподаватель-руководитель курсового проекта** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Подпись учащегося**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**СОДЕРЖАНИЕ**

*yу*

Изм.

Лист

№ докум.

Подпись

Дата

Лист

3

*МКП КП 2-40 01 01 33 006 ПЗ*

Разраб.

Винничук

Пров.

*Чеботарев*

Реценз.

Н. контр.

Утв.

*Игра «Королевство Вивальди»*

Лит.

Листов

МКП

Введение 5

1 Постановка задачи 8

1.1 Формулировка задания 8

1.2 Обоснование необходимости автоматизации 9

1.3 Определение данных и их представление 11

2 Вычислительная система 12

2.1 Обоснование выбора языка программирования 12

2.2 Обоснование выбора среды разработки 13

2.3 Требования к конфигурации программного и аппаратного обеспечения 16

3 Проектирование 18

3.1 Проектирование интерфейса 18

3.2 Описание алгоритма 19

4 Описание программы 20

4.2 Физическая структура 22

4.3 Особенности интерфейса 23

5 Отладка и испытание программы 27

5.1 Тестовые примеры 27

5.2 Анализ полученных результатов 32

6 Описание применения программы 38

6.1 Назначение программы 38

6.2 Условия применения 38

Заключение 39

Список использованых источников 40

Изм.

Лист

№ докум.

Подпись

Дата

*МКП КП 2-40 01 01 33 006 ПЗ*

Лист

4

Приложение А (обязательное) Листинг программы 41

ВВЕДЕНИЕ

Требуется спроектировать игру «Королевство Вивальди». В ходе этой игры игроку выпадает роль принцессы «Алендоры» которая пытается найти древний артефакт, обладающий несведущими возможностями. Жанром этой игры будет 2D-Platformer, так-же в игре будет присутствовать сюжетная линия.

Платформеры появились в начале 1980-х, когда игровые консоли не были достаточно мощными, чтобы отображать трёхмерную графику или видео. Они были ограничены статическими игровыми мирами, которые помещались на один экран, а игровой герой был виден в профиль. Персонаж лазал вверх и вниз по лестницам или прыгал с платформы на платформу, часто сражаясь с противниками и собирая предметы, улучшающие характеристики. Первыми играми этого типа были Space Panic и Apple Panic.

Платформер (англ. platformer, platform game) – жанр компьютерных игр, в которых основной чертой игрового процесса является прыгание по платформам, лазанье по лестницам, собирание предметов, обычно необходимых для завершения уровня. Некоторые предметы, называемые пауэр-апами (англ. power-up), наделяют управляемого игроком персонажа особой силой, которая обычно иссякает со временем к примеру: силовое поле, ускорение, увеличение высоты прыжков. Коллекционные предметы, оружие и «пауэер-ап» собираются обычно простым прикосновением персонажа и для применения не требуют специальных действий со стороны игрока. Реже предметы собираются в «инвентарь» героя и применяются специальной командой (такое поведение более характерно для аркадных головоломок).

Уровни, как правило, изобилуют секретами скрытые проходы в стенах, высокие или труднодоступные места, нахождение которых существенно облегчает прохождение и подогревает интерес игрока.

Игры подобного жанра характеризуются нереалистичностью, рисованной мультяшной графикой. Героями таких игр обычно бывают мифические существа (к примеру: драконы, гоблины) или антропоморфные животные.

Основной целью выполнения проекта является изучение Unity и его возможностей, а так-же систематизация знаний при разработке скриптов на языке C# в Visual Studio.

Задачами выполнения курсового проекта являются обобщение и углубление знаний при разработке скриптов, увеличение знаний о разработке игра на Unity, расширение знаний о составлении научно-исследовательской работы, расширение навыков поиска информации.

Пояснительная записка содержит описание созданного программного средства и включает следующие разделы:

1. «Введение», в котором находится краткое описание документа.
2. «Постановка задачи», которая содержит подробное описание всех процедур и действий, происходящих в программе.
3. «Вычислительная система», где идёт описание языка программирования для реализации проекта, его достоинств и недостатков, описание выбранной среды разработки для реализации проекта, ее достоинств и недостатков и описание операционной системы (ОС), в которой была реализована программа.
4. «Проектирование», в котором идёт описание структуры внешнего пользовательского интерфейса, компонентов для реализации взаимодействия приложения с пользователем и функционально-независимых подзадач проекта.
5. «Описание программы», где содержится описание всех модулей, форм, входящих в приложение, их назначение, описание всех подпрограмм, входящих в приложение, сгруппированных по формам или библиотечным модулям.
6. «Отладка и испытание программы», для краткого описания используемых методов тестирования, проверка обработки исключений, описания реакции программы на каждый тестовый пример с приведение скриншотов, анализ полученных результатов, выводы по результатам тестирования.
7. «Описание применения программы», для сведений о назначении программного средства, области применения, классов решаемых задач, ограничениях, накладываемых на область применения, а так же краткого руководства по работе с программным средством, описание интерфейса со скриншотами всех форм, описание установки и запуска программы.
8. «Заключение», для краткой формулировки проблемы и пути решения проблемы, полноты раскрытия проблемы, а также описание используемых методов и средств, найденных нетрадиционных способов разработки, возможностей дальнейшей модификации проекта, степени автоматизации работ на различных этапах разработки.
9. ПОСТАНОВКА ЗАДАЧИ

1.1 Формулировка задания

Необходимо разработать главное меню с кнопками «Новая игра», «Продолжить», «Настройки», «Об игре», «Обучение» и «Выход».

Требуется разработать селектор уровней, на котором пользователь сможет выбирать уровень для прохождения, сможет изменять личный арсенал за накопленную валюту.

Требуется разработать механику главного героя, схожую с играми в жанре 2D-Platformer, а именно:

1. Реализовать систему управления игроком.
2. Реализовать систему стрельбы
3. Реализовать систему бонусов для игрока (изменение личного арсенала, дополнительная жизнь)
4. Реализовать логику главного босса
5. Реализовать обучение для начинающих
6. Реализовать различные препятствия, которые игроку придётся преодолевать.
7. Реализовать различных противников для игрока с уникальными характеристиками.
8. Реализовать рассказчика, который будет рассказывать дальнейшие действия.
9. Реализовать конец каждого уровня, в котором игрок будет переходить на локации.

Главным героем «Королевство Вивальди» является принцесса «Алендора» которая пытается найти древний артефакт, обладающий несведущими возможностями. Проходя различные локации, принцесса сражается со злыми силами, которые не дают забрать древний артефакт. На её пути будет встречаться рассказчик, который сможет помочь рассказать куда идти дальше и ждёт впереди. В самом конце принцессу «Алендора» ждёт сражение с главным злодеем, победив которого она сможет забрать древний артефакт.

Действия персонажа будут происходить по нажатию на разные зоны экрана смартфона. Левая часть отвечает за движение игрока налево, правая часть за движение направо и центральная часть экрана за прыжок игрока.

1.2 Обоснование необходимости автоматизации

Платформеры появились в начале [1980-х](https://ru.wikipedia.org/wiki/1980-%D0%B5) и стали [трёхмерными](https://ru.wikipedia.org/wiki/%D0%A2%D1%80%D1%91%D1%85%D0%BC%D0%B5%D1%80%D0%BD%D0%B0%D1%8F_%D0%B3%D1%80%D0%B0%D1%84%D0%B8%D0%BA%D0%B0) ближе к концу [1990-х](https://ru.wikipedia.org/wiki/1990-%D0%B5). Через некоторое время после образования жанра у него появилось данное название, отражающее тот факт, что в платформерах [геймплей](https://ru.wikipedia.org/wiki/%D0%93%D0%B5%D0%B9%D0%BC%D0%BF%D0%BB%D0%B5%D0%B9) сфокусирован на прыжках по платформам в противовес стрельбе. Правда, во многих платформерах присутствует стрелковое оружие, в таких, например, как [Super](https://ru.wikipedia.org/wiki/Blackthorne) Mario или Terraria.

Super Mario – [видеоигра](https://ru.wikipedia.org/wiki/%D0%92%D0%B8%D0%B4%D0%B5%D0%BE%D0%B8%D0%B3%D1%80%D0%B0) в жанре [платформера](https://ru.wikipedia.org/wiki/%D0%9F%D0%BB%D0%B0%D1%82%D1%84%D0%BE%D1%80%D0%BC%D0%B5%D1%80), разработанная и выпущенная в 1985 году японской компанией [Nintendo](https://ru.wikipedia.org/wiki/Nintendo) для платформы [Famicom](https://ru.wikipedia.org/wiki/Famicom). Занесена в «[Книгу рекордов Гиннесса](https://ru.wikipedia.org/wiki/%D0%9A%D0%BD%D0%B8%D0%B3%D0%B0_%D1%80%D0%B5%D0%BA%D0%BE%D1%80%D0%B4%D0%BE%D0%B2_%D0%93%D0%B8%D0%BD%D0%BD%D0%B5%D1%81%D1%81%D0%B0)» как самая продаваемая игра в истории. После успеха игры её главный герой, [Марио](https://ru.wikipedia.org/wiki/%D0%9C%D0%B0%D1%80%D0%B8%D0%BE_(%D0%BF%D0%B5%D1%80%D1%81%D0%BE%D0%BD%D0%B0%D0%B6_%D0%B8%D0%B3%D1%80%D1%8B)), стал символом компании Nintendo и одним из самых известных игровых персонажей в мире. Главное окно программы изображено на рисунке 1.1.

****

Рисунок 1.1 – Главное окно программы Super Mario

По пути Марио собирает монеты и бонусы, ударяя по блокам со знаком вопроса, а также выискивая секретные хранилища монет в кирпичных стенах. При наборе ста монет Марио получает дополнительную [«жизнь»](https://ru.wikipedia.org/wiki/%D0%9E%D1%87%D0%BA%D0%B8_%D0%B6%D0%B8%D0%B7%D0%BD%D0%B8), изначально у Марио есть три «жизни». Главное окно программы изображено на рисунке 1.2.



Рисунок 1.2 – Главное окно программы Terraria

Перед началом игры игрок должен создать персонажа и задать его внешность с помощью ряда настроек, а после этого нужно выбирать параметры для процедурной генерации. Затем генерируется мир, размер которого варьируется от выбора игрока.

Вывод: на основании тех фактов о Super Mario и Terraria в игре «Королевство Вивальди» будет реализовано обучение для новых пользователей, которые впервые заходят в данную игру. В обучении игрок сможет ознакомится с физикой главного героя, за которого ему придётся играть, узнать о препятствиях, которые могут встретится на его пути. Но в моей игре не будет на столько большого количества функционала, связанного с прохождением уровней игры таких как разнообразные объекты на сцене, различные бонусы, и разнообразных функционал главного персонажа.

1.3 Определение данных и их представление

В разрабатываемом курсовом проекте можно определить два типа используемых данных (входные и выходные). Входные данные представляют из себя данные, входящие в игру из-за действий игрока, а выходные – те данные, что выводятся в игре в результате каких-либо действий.

Входные данные:

1. Управление персонажей (оси Ox и Oy, удар мечом и т.д.);
2. Количество жизней;
3. Количество пройденных уровней;
4. Обучение в игре.

Выходные данные:

1. Уменьшение (увеличение) количества здоровья героя и ботов;
2. Уменьшение (увеличение) количества очков;
3. Изменение сцен при переходе из одной в другую;
4. Изменение параметров игры в результате изменения настроек;
5. ВЫЧИСЛИТЕЛЬНАЯ СИСТЕМА

2.1 Обоснование выбора языка программирования

C# является объектно – ориентированным языком, но поддерживает также и **компонентно-ориентированное** программирование. Разработка современных приложений все больше тяготеет к созданию программных компонентов в форме автономных и само описательных пакетов, реализующих отдельные функциональные возможности. Важная особенность таких компонентов – это модель программирования на основе свойств, методов и событий. Каждый компонент имеет атрибуты, предоставляющие декларативные сведения о компоненте, а также встроенные элементы документации. C# предоставляет языковые конструкции, непосредственно поддерживающие такую концепцию работы. Благодаря этому C# отлично подходит для создания и применения программных компонентов.

Вот лишь несколько функций языка C#, обеспечивающих надежность и устойчивость приложений: **сборка мусора** автоматически освобождает память, занятую уничтоженными и неиспользуемыми объектами; **обработка исключений** предоставляет структурированный и расширяемый способ выявлять и обрабатывать ошибки; **строгая типизация** языка не позволяет обращаться к неинициализированным переменным, выходить за пределы индексируемых массивов или выполнять неконтролируемое приведение типов.

Так-же популярный бесплатный игровой движок Unity поддерживает написание скриптов на языке C#, что позволяет использовать все преимущества языка для разработки различных 2D и 3D игр.[5, с 11]

2.2 Обоснование выбора среды разработки

Unity – межплатформенная среда разработки компьютерных игр. Unity позволяет создавать приложения, работающие под более чем 20 различными операционными системами, включающими персональные компьютеры, игровые консоли, мобильные устройства, интернет-приложения и другие. Выпуск Unity состоялся в 2005 году и с того времени идёт постоянное развитие.

Основными преимуществами Unity являются наличие визуальной среды разработки, межплатформенной поддержки и модульной системы компонентов. К недостаткам относят появление сложностей при работе с многокомпонентными схемами и затруднения при подключении внешних библиотек.

На Unity написаны тысячи игр, приложений и симуляций, которые охватывают множество платформ и жанров. При этом Unity используется как крупными разработчиками, так и независимыми студиями.

Возможности движка Unity почти безграничны. Редактор Unity имеет простой Drag&Drop интерфейс, который легко настраивать, состоящий из различных окон, благодаря чему можно производить отладку игры прямо в редакторе. Движок поддерживает два скриптовых языка: C#, JavaScript (модификация). Ранее была поддержка Boo (диалект Python), но его убрали в 5-ой версии. Расчёты физики производит физический движок PhysX от NVIDIA.

Проект в Unity делится на сцены (уровни) – отдельные файлы, содержащие свои игровые миры со своим набором объектов, сценариев, и настроек. Сцены могут содержать в себе как, собственно, объекты (модели), так и пустые игровые объекты – объекты, которые не имеют модели («пустышки»). Объекты, в свою очередь содержат наборы компонентов, с которыми и взаимодействуют скрипты. Также у объектов есть название (в Unity допускается наличие двух и более объектов с одинаковыми названиями), может быть тег (метка) и слой, на котором он должен отображаться. Так, у любого объекта на сцене обязательно присутствует компонент Transform – он хранит в себе координаты местоположения, поворота и размеров объекта по всем трём осям. У объектов с видимой геометрией также по умолчанию присутствует компонент Mesh Renderer, делающий модель объекта видимой.

К объектам можно применять коллизии (в Unity т. н. коллайдеры – collider), которых существует несколько типов.

Также Unity поддерживает физику твёрдых тел и ткани, а также физику типа Ragdoll (тряпичная кукла). В редакторе имеется система наследования объектов; дочерние объекты будут повторять все изменения позиции, поворота и масштаба родительского объекта. Скрипты в редакторе прикрепляются к объектам в виде отдельных компонентов.

При импорте текстуры в Unity можно сгенерировать alpha-канал, mip-уровни, normal-map, light-map, карту отражений, однако непосредственно на модель текстуру прикрепить нельзя – будет создан материал, которому будет назначен шейдер, и затем материал прикрепится к модели. Редактор Unity поддерживает написание и редактирование шейдеров. Редактор Unity имеет компонент для создания анимации, но также анимацию можно создать предварительно в 3D-редакторе и импортировать вместе с моделью, а затем разбить на файлы.

Unity 3D поддерживает систему Level Of Detail (сокр. LOD), суть которой заключается в том, что на дальнем расстоянии от игрока высоко-детализированные модели заменяются на менее детализированные, и наоборот, а также систему Occlusion culling, суть которой в том, что у объектов, не попадающих в поле зрения камеры не визуализируется геометрия и коллизия, что снижает нагрузку на центральный процессор и позволяет оптимизировать проект. При компиляции проекта создается исполняемый (.exe) файл игры (для Windows), а в отдельной папке – данные игры (включая все игровые уровни и динамически подключаемые библиотеки).

Движок поддерживает множество популярных форматов. Модели, звуки, текстуры, материалы, скрипты можно запаковывать в формат .unityassets и передавать другим разработчикам, или выкладывать в свободный доступ. Этот же формат используется во внутреннем магазине Unity Asset Store, в котором разработчики могут бесплатно и за деньги выкладывать в общий доступ различные элементы, нужные при создании игр. Чтобы использовать Unity Asset Store, необходимо иметь аккаунт разработчика Unity. Unity имеет все нужные компоненты для создания мультиплеера. Также можно использовать подходящий пользователю способ контроля версий. К примеру, Tortoise SVN или Source Gear.

В Unity входит Unity Asset Server – инструментарий для совместной разработки на базе Unity, являющийся дополнением, добавляющим контроль версий и ряд других серверных решений.

Microsoft Visual Studio – линейка продуктов компании Microsoft, включающих интегрированную среду разработки программного обеспечения и ряд других инструментальных средств. Данные продукты позволяют разрабатывать как консольные приложения, так и приложения с графическим интерфейсом, в том числе с поддержкой технологии Windows Forms, а также веб-сайты, веб-приложения, веб-службы как в родном, так и в управляемом кодах для всех платформ.

Visual Studio включает в себя редактор исходного кода с поддержкой технологии IntelliSense и возможностью простейшего рефакторинга кода. Встроенный отладчик может работать как отладчик уровня исходного кода, так и отладчик машинного уровня. Остальные встраиваемые инструменты включают в себя редактор форм для упрощения создания графического интерфейса приложения, веб-редактор, дизайнер классов и дизайнер схемы базы данных. Visual Studio позволяет создавать и подключать сторонние дополнения (плагины) для расширения функциональности практически на каждом уровне, включая добавление поддержки систем контроля версий исходного кода (как, например, Subversion и Visual SourceSafe), добавление новых наборов инструментов (например, для редактирования и визуального проектирования кода на предметно-ориентированных языках программирования) или инструментов для прочих аспектов процесса разработки программного обеспечения.[3,с 12-14]

2.3 Требования к Конфигурации программного и аппаратного обеспечения

Требования к конфигурации программного и аппаратного обеспечения

В качестве ОС для разработки курсового проекта была выбрана платформа Android. Достоинствами данной платформы является интуитивно понятный пользователю интерфейс, надежность, производительность. Android – операционная система для смартфонов, планшетов Большое количество современных приложений оптимизированно именно под платформу Android.

Достоинства Android:

1. Простор в выборе персонализации. Каждый пользователь может скачать лаунчер на свой вкус и свободно установить его.
2. Возможность иметь более одного аккаунта.
3. Наличие функции расширения памяти путем добавления SD-карты.
4. Свободное скачивание файлов с любого ресурса. Но существует значительный риск получить с нужным вам приложением вредный вирус.
5. Защищённость (гаджет на платформе Android сложно заразить вирусом или вывести из строя по незнанию);
6. Облачное хранение данных, автоматическое сохранение резервных копий (первый вариант допускает совместное использование файлов на всех устройствах Apple, второй обезопасит от потери всех пользовательских данных в случае поломки или кражи гаджета);
7. Многозадачность (свёрнутые приложения не потребляют ресурсы).
8. Это же касается и свободы действий в использовании программ по умолчанию. Можно установить и более удобный проигрыватель видео и аудио, и приложение для просмотра изображений или даже камеру с расширенными установками.
9. Помимо этого, пользователи системы Android могут выбирать клавиатуру и способ набора текста среди десятков платных и бесплатных предложений в Google Play.
10. И, наконец – для многих пользователей это главное преимущество – пользователям смартфона или планшета на Android не требуется дополнительных программ для доступа к памяти гаджета. Устройства подключаются к компьютеру при помощи USB-провода, и все папки со всеми содержащимися в них файлами оказываются доступными. [4,c 14-15]

Минимальные системные требования для разработки:

* ОЗУ – 4096МБ;
* свободной памяти 1500МБ;
* разрешение экрана 800\*600;
* процессор Intel Core I3 или эквивалентный;
* устройства ввода: клавиатура, мышь;
* устройства вывода: монитор.

3 ПРОЕКТИРОВАНИЕ

3.1 Проектирование интерфейса

Интерфейс ­ совокупность средств, методов и правил взаимодействия между элементами системы.

Интерфейс пользователя, обеспечивает передачу информации между пользователем-человеком и программно-аппаратными компонентами компьютерной системы.

На данный момент в проекте существует десять сцен. Сцена Menu представляет собой главное меню игры. Сцена Training представляет собой обучение для игры. Сцена LevelSelect содержит селектор уровней для игры. Сцена Shop представляет собой магазин улучшений и покупок новых кораблей. Сцены Level1, Level2, Level3, Level4, Level5 представляют собой космические уровни игры.

На сцене меню располагаются 5 кнопок. Первая отвечает за продолжение игрового процесса, пройденного ранее при запуске игры. Вторая отвечает за игровой магазин. Третья за прохождение обучения. Четвертая за сброс всего игрового процесса. Пятая за выход из приложения.

На сцене выбора уровня находится 6 кнопок. Первые 5 отвечают за выбор уровня, а шестая отвечает за возврат в главное меню.

На сцене магазина находится изображение текущей с текущим количеством кредитов, кнопки для покупки улучшений и кнопка выхода из магазина обратно в главное меню.

На сцене освоения земель, где будет происходить процесс победы добра над злом от неприятелей располагается главный персонаж, основные кнопки управлением игроком удара и текст с количеством заработанных монет.

3.2 Описание алгоритма

В ходе работы были найдены сложные алгоритмы:

1. Алгоритм удара. При создании алгоритма удара, сложным оказалось то, как необходимо было создавать событие на нажатие и заставлять анимацию удара не зацикливаться.

Входные данные: нажатие кнопки удара, анимация удара

Выходные данные: скорость анимации, с которой она проигрывалась, объект меча и его взаимодействие с другими объектами.

1. Алгоритм передвижения персонажа. При создании алгоритма передвижения, главной проблемой было то, как задать плавный прыжок персонажу, и как сделать так, чтобы персонаж мог прыгать один раз.

Входные данные: скорость движения, точка проверки соприкосновения с землёй, какой тэг объектов считать землёй.

Выходные данные: Персонаж с плавным передвижением и одиночным прыжком.

1. Алгоритм смены сцен. При создании алгоритма смены сцен, существенная проблема была в том, чтобы определить какая сцена загружена на данный момент и какую нам надо запустить.

Входные данные: номер необходимой сцены.

Выходные данные: запуск или перезапуск сцены по её номеру.

1. Алгоритм покупки. При создании алгоритма покупки, существенная проблема была в том, чтобы определить какое улучшение уже куплено, а какое ещё нет.

Входные данные: номер предыдущего улучшения.

Выходные данные: покупка улучшения и появление его в типизированном файле.

4 ОПИСАНИЕ ПРОГРАММЫ

После разработки проекта итоговая логическая структура имеет следующий вид:

Методы, общие для всех классов:

1. Update() – Встроенный в библиотеку UnityEngine метод, который постоянно проверяется и выполняется;
2. Start() – Встроенный в библиотеку UnityEngine метод, который выполняется, во время старта скрипта.

Методы входящие в класс Hero:

1. ChechWhereToFace – Метод который проверяет в которую сторону смотрит игрок и в каком он сейчас направлении и в нужный момент меняет направление игрока.
2. Hurt – Метод который дает усилие отталкивания игроку когда он сталкивается с огнём.
3. SetAnimationState() – Метод который отвечает за проверку того какая анимация должна воспроизводиться в тот или иной момент.
4. OnTriggerEnter() – Встроенный в библиотеку UnityEngine метод, который проверяет на вхождение в триггер нашего объекта, других объектов.

Методы входящее в класс WinMenu:

1. MainMenu() – Метод который отвечает за сохранение пройденного уровня.
2. LoadMainMenu() – Метод который отвечает за открытие сцены с селектором уровней.
3. LoadNextLebel() – Метод который определяет какой уровень был сохранен последним и открывает следующий если пройден предыдущий.

Методы, входящие в класс Goblin:

1. OnTriggerEnter() – Встроенный в библиотеку UnityEngine метод, который проверяет на вхождение в триггер нашего объекта, других объектов;
2. ChechWhereToFace() – Метод который проверяет в которую сторону смотрит игрок и в каком он сейчас направлении и в нужный момент меняет направление игрока.
3. LateUpdate() – Встроенный в библиотеку UnityEngine метод, который отвечает за отрисовку последнего кадра объекта.

Методы, входящие в класс Pause:

PauseGame() – Метод который отвечает за остановку игрового процесса и сохранение результата игры.

Resume() – Метод который отвечает за снятие игры с паузы и загрузку всей информации игрока.

Quit() – Метод который отвечает за выход их приложения.

MainMenu() – Метод который отвечает за остановку игрового процесса и переход в селектор уровней.

Методы, входящие в класс ButtonHit:

OnPointerClick() – Метод который отлавливает событие на момент нажатия на кнопку и производит анимацию удара.

OnPointerUp() – Метод который отвечает за остановку анимации игрока в момент отжатой кнопки удара.

Методы, входящие в класс LevelControlScript:

YouWin() – Метод который отвечает за проверку вывода монет на игровых уровнях и переход на следующий уровень.

YouLose() – Метод который отвечает за не пройденный уровень и не сохраняет уровень как пройденный.

Методы, входящие в класс Reset:

Resets() – Метод который отвечает за сброс всех сохранений игрока которые были сохранены в процессе игры.

Метод, входящий в класс CupScript

OnTriggerEnter() – Встроенный в библиотеку UnityEngine метод, который проверяет на вхождение в триггер нашего объекта, других объектов;

4.2 Физическая структура

После разработки проекта, физическая структура стала иметь следующий вид:

Класс ButtonHit – класс, отвечающий за удар игрока по противникам.

Класс Cloud – класс, отвечающий за движение объектов на селекторе уровней.

Класс Goblin – класс, отвечающий за удар и основные характеристики противников.

Класс Hero – класс, отвечающий за характеристики главного героя такие как передвижение и его взаимодействие его с другими объектами.

Класс MenuControlScript – класс, отвечающий за переключение между сценами.

Класс CupScript – класс, отвечающий за пройденный уровень.

Класс HealthBarScript – класс, отвечающий за отображение жизней зимнего голема.

Класс LevelControlScript – класс, отвечающий за сохранение и переключение между различным переключением меню.

Класс LevelNumberScript – класс, отображающий уровень на котором находится игрок.

Класс Planet – класс, отображающий настройки экрана.

Класс HeroBoss – класс, отвечающий за главного босса героя и отображение его характеристик.

Класс LoseMenu – класс, отвечающий за не пройденный уровень.

Класс Pause – класс, отвечающий за меню паузы в процессе игры.

Класс Reset – класс, отвечающий за сброс всех сохранений игрока.

Класс WinMenu – класс, отвечающий за удачно пройденный уровень.

4.3 **Особенности интерфейса**

Видом основного игрового интерфейса для наземных сражений показан на рисунке 4.1.



Рисунок 4.1 – Изображение игрового интерфейса из игры Королевство Вивальди

На нём изображены: количество жизней персонажа в правом верхнем углу, количество монет в левом верхнем углу и кнопка паузы в правом верхнем углу.

Меню выглядит так, как показано на рисунке 4.2.



Рисунок 4.2 – Изображение главного меню из игры Королевство Вивальди

На нём изображены 5 кнопок в виде закругленных прямоугольников, название игры.

Меню тренировочного лагеря выглядит так, как показано на рисунке 4.3.



Рисунок 4.3 – Изображение меню тренировочного лагеря из игры Королевство Вивальди

Селектор уровней выглядит так, как показано на рисунке 4.4.



Рисунок 4.4 – Изображение селектора уровней из игры Королевство Вивальди

На нём изображены кнопки в виде цифр, которые загружают определенные уровни.

Магазин улучшений выглядит так, как показано на рисунке 4.5.



Рисунок 4.5 – Изображение сцены магазина улучшений из игры Королевство Вивальди

На нём изображены кнопка выбора/покупки улучшения, текст количества кредитов и кнопка возврата к главному меню.

Видом основного игрового интерфейса для сражений различных локациях показан на рисунке 4.6

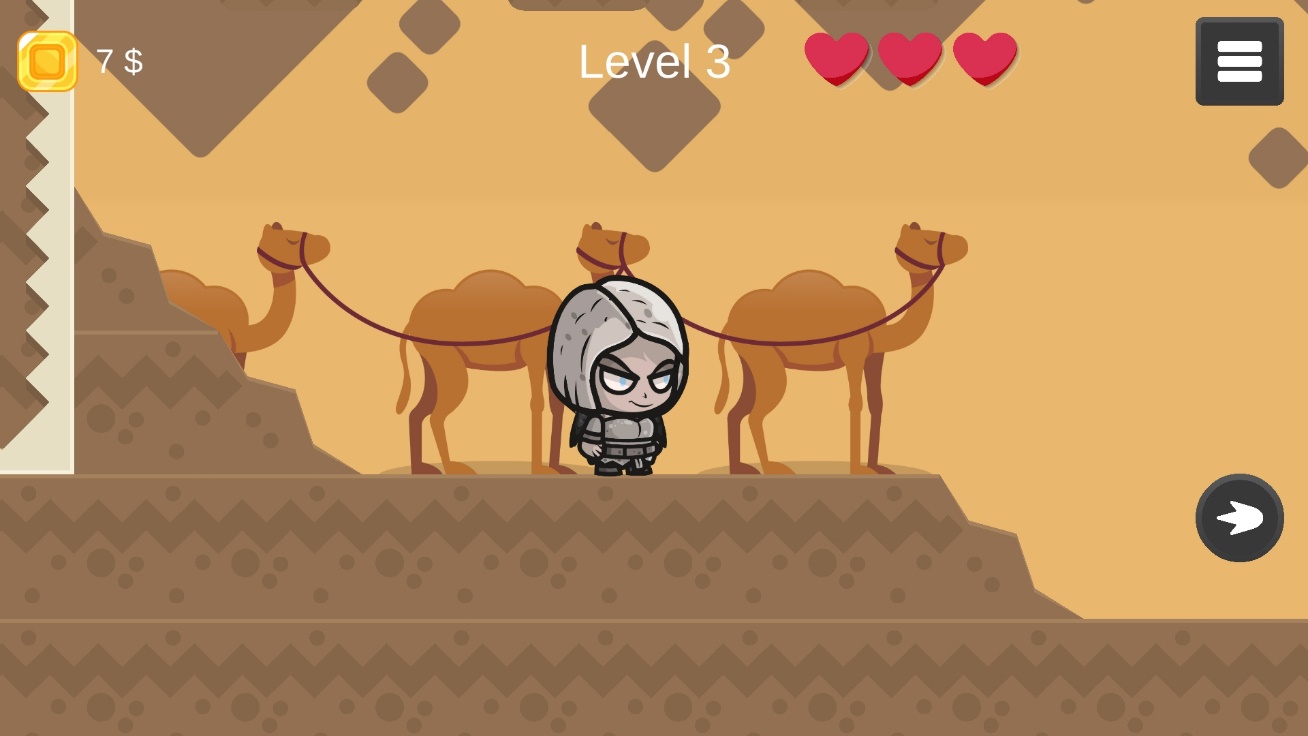


Рисунок 4.6 – Изображение основного интерфейса сражений на одной из локации

1. ОТЛАДКА И ИСПЫТАНИЕ ПРОГРАММЫ
   1. Тестовые примеры

Тестирование программного обеспечения – процесс исследования, испытания программного продукта, имеющий своей целью проверку соответствия между реальным поведением программы и её ожидаемым поведением на конечном наборе тестов, выбранных определенным образом.

Существует несколько признаков, по которым принято производить классификацию видов тестирования. Обычно выделяют следующие:

1. По объекту тестирования;
2. По знанию внутреннего строения системы;
3. По степени автоматизации;
4. По степени изолированности;
5. По времени проведения тестирования;
6. По признаку позитивности сценариев;
7. По степени подготовленности к тестированию.

После разработки проекта, были составлены такие тесты:

1. Тесты интерфейса:

* тест уменьшения количества здоровья, при столкновении с врагом, как показано на рисунке 5.1.

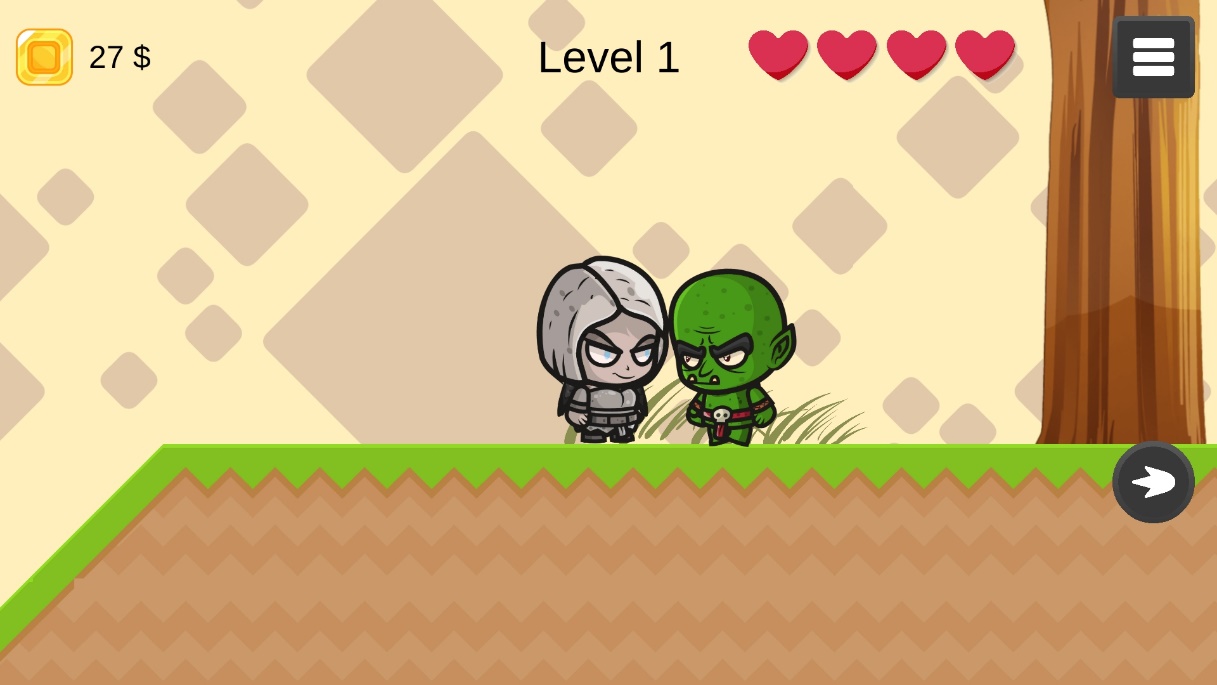


Рисунок 5.1 –­ Изображение приближения к врагу в игре Королевство Вивальди

* тест на переход на уровень после нажатия соответствующей кнопки, как показано на рисунке 5.2.



Рисунок 5.2 – Изображение перед выбором уровня в игре Королевство Вивальди

* тест на показ меню паузы при нажатии на соответствующую кнопку, как показано на рисунке 5.3:



Рисунок 5.3 – Изображение интерфейса до нажатия паузы в игре Королевство Вивальди

1. Тесты игрового персонажа:

* тест работы анимации передвижения, как показано на рисунке 5.4.

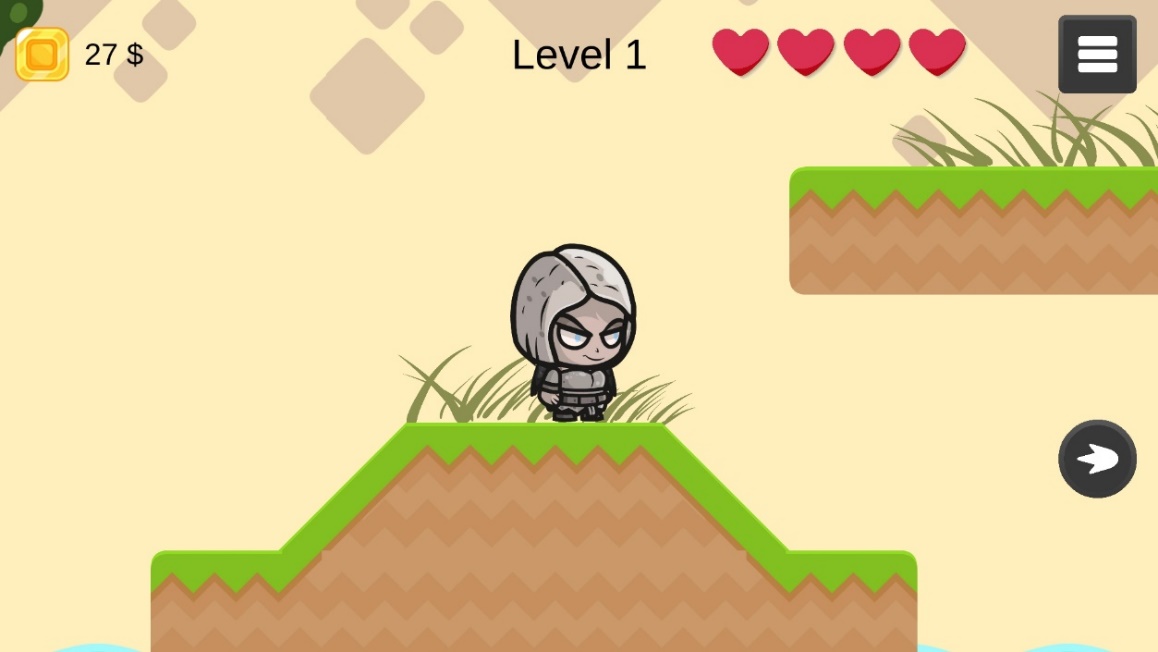


Рисунок 5.4 – Изображение покоя персонажа в игре Королевство Вивальди

* тест одиночного прыжка от земли, как показано на рисунке 5.5:



Рисунок 5.5 – Изображение прыжка персонажа, до второго нажатия кнопки прыжка в игре Королевство Вивальди

* тест на зависание персонажа с края платформы, как показано на рисунке 5.6.



Рисунок 5.6 – Изображение прыжка персонажа в платформу сбоку, до возможного застревания в игре Королевство Вивальди

1. Тесты игровой механики:

* тест битва за жизнь, как показано на рисунке 5.7.

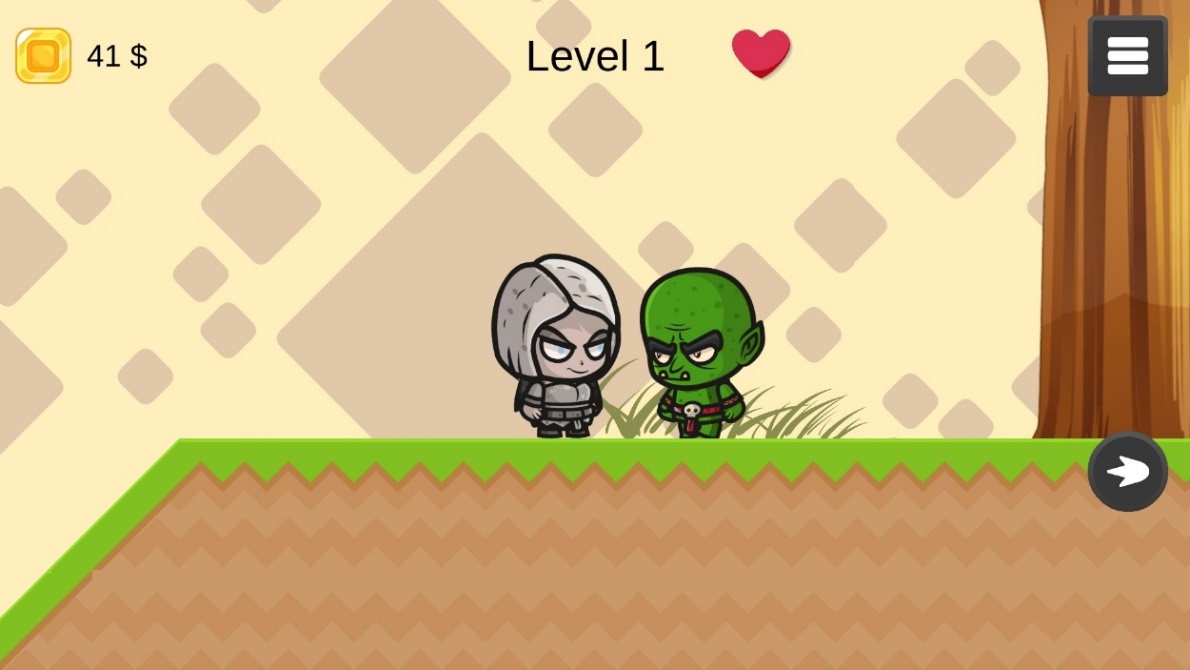


Рисунок 5.7 – Изображение перед смертью главного героя в игре Королевство Вивальди

* тест удара мечом главного героя, как показано на рисунке 5.8.

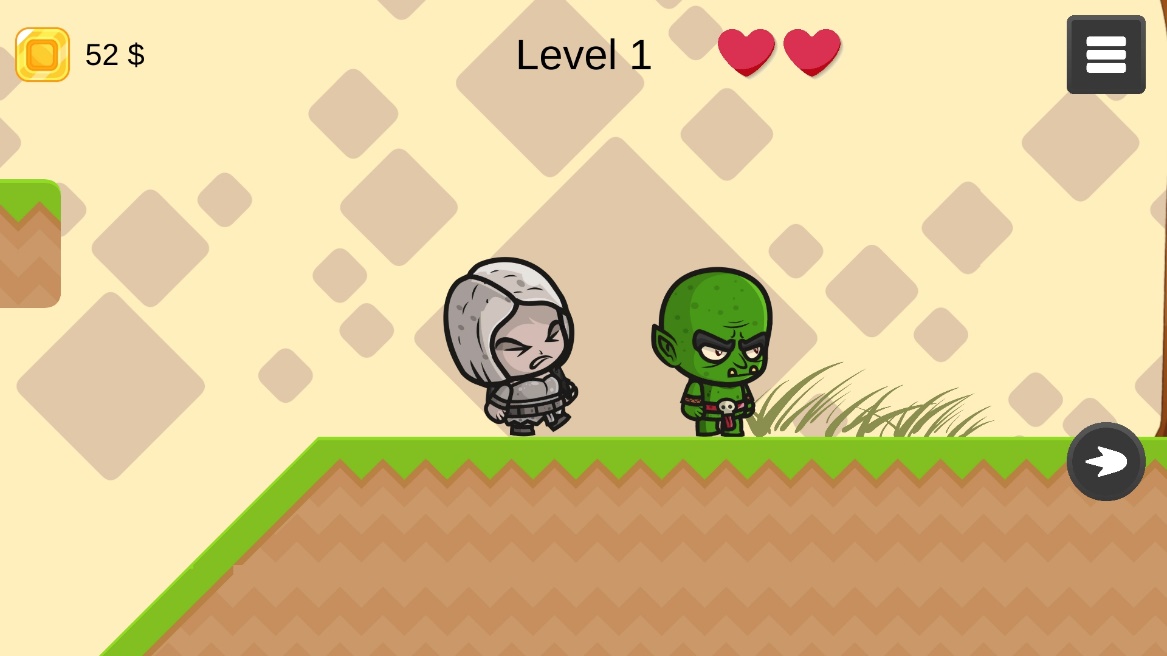


Рисунок 5.8 – Изображение перед ударом мечом в игре Королевство Вивальди

* 1. Анализ полученных результатов

После выполнения тестов интерфейса, были выявлены следующие результаты:

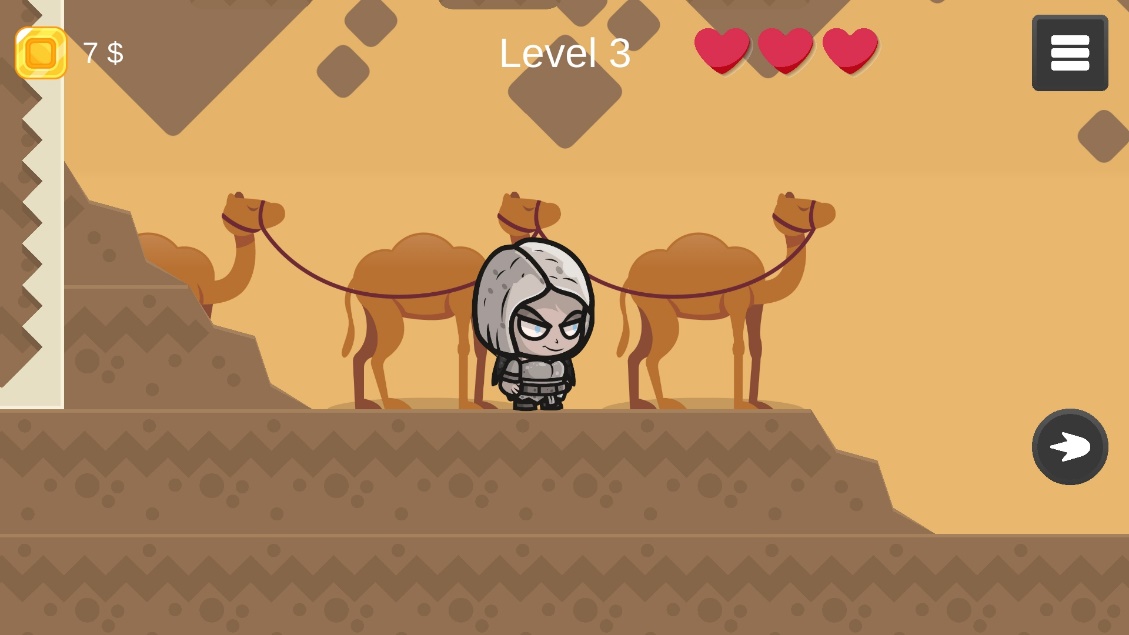
1. Результат теста на уменьшение количества отображаемого здоровья был удовлетворителен, как показано на рисунке 5.9.

  
Рисунок 5.9 – Изображение после столкновения с врагом в игре Королевство Вивальди

1. Результат теста на показ меню паузы был удовлетворителен, как показано на рисунке 5.10.

  
Рисунок 5.10 – Изображение меню паузы в игре Королевство Вивальди

1. Результат теста на выбор уровня из селектора уровней оказался удовлетворительным, как показано на рисунке 5.11.

  
Рисунок 5.11 – Изображение запущенного игрового уровня в игре Королевство Вивальди

После тестирования интерфейса, все проведённые тесты прошли проверку и было выявлено, что всё работает верно и без ошибок.

После проведения тестов игрового персонажа, были выявлены следующие результаты:

1. Результат теста изменения анимации персонажа во время движения был удовлетворителен и из анимации покоя, во время движения, персонаж перешёл в анимацию, как показано на рисунке 5.12.

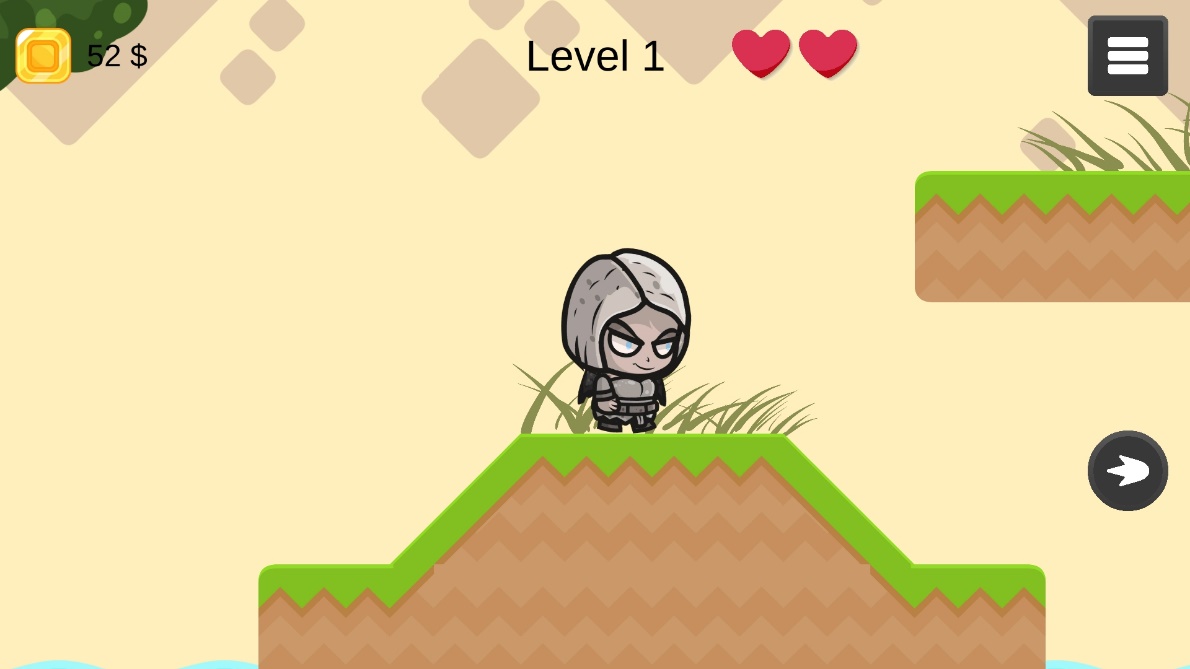


Рисунок 5.12 – Изображение после начала движения в игре Королевство Вивальди

1. Результат теста одиночного прыжка от земли оказался удовлетворителен и повторный прыжок, без прикосновения к земле совершить нельзя, как показано на рисунке 5.13.



Рисунок 5.13 – Изображение после повторного нажатия на кнопку прыжка в игре Королевство Вивальди

1. Результат теста на зависания персонажа у края платформы был удовлетворительным, и персонаж не завис, как показано на рисунке 5.14.



Рисунок 5.14 – Изображение после попытки зависания у края платформы в игре Космический защитник

После тестирования игрового персонажа, все проведённые тесты прошли проверку. Поэтому было выявлено, что персонаж не требует доработок.

После проведения тестов на игровой механике, были выявлены следующие результаты:

1. Результат теста перед смертью главного героя был удовлетворительным, игрок погибает, как показано на рисунке 5.15.



Рисунок 5.15 – Изображение погибшего игрока в игре Королевство Вивальди

1. Результат теста на появление появлением меча был удовлетворительным как показано на рисунке 5.16.



Рисунок 5.16 – Изображение появившегося меча в игре Королевство Вивальди

После тестирования игровой механики, все проведённые тесты прошли успешно. Поэтому было выявлено механика не требует доработок.

1. ОПИСАНИЕ ПРИМЕНЕНИЯ ПРОГРАММЫ
   1. Назначение программы

Программа предназначена для развлекательных целей. Любой, кто захочет может поиграть в эту игру, так как даже если дети захотят поиграть в эту игру, то родители могут не беспокоится из-за того, что в этой игре нет явного насилия или запрещённых элементов.

Областью применения является развлечение людей.

* 1. Условия применения

Минимальные системные требования:

* Операционная система Android;
* ОЗУ – 1024МБ;
* Свободное время в памяти 500МБ;
* разрешение экрана 1280\*720;
* процессор MediaTek MT6589W;
* устройства ввода: сенсорный экран;
* устройства вывода: сенсорный экран.

Для начала работы с приложением необходимо пройти обучение после будет возможность проходить игровые уровни без каких – либо затруднений. Справа на игровом уровне вы увидите кнопку для перехода в меню паузы. Нажав на кнопку «Resume» возобновится игровой процесс. Кнопка «Restart» переиграет уровень, на котором вы находитесь. Клик по кнопке «Exit» завершит работу приложения и вернет в главное меню смартфона.

ЗАКЛЮЧЕНИЕ

После создания данного курсового проекта, я изучил основы работы в Unity. Были изучены основы создания анимации, работы с двумерными объектами на сцене, придачи им различных свойств и т.д.

Так же я систематизировал свои знания в сфере разработки скриптов для Unity на языке C# в Visual Studio.

Были выполнены все задачи, поставленные при выполнении этого курсового проекта:

1. Обобщены и углублены знания при разработке скриптов;
2. Увеличены знания о разработке игр на Unity;
3. Расширены знания о составлении научно-исследовательской работы;
4. Расширены навыки поиска информации.

В добавок к этому, в ходе разработки курсового проекта, также были выучены основы работы в Illustrator для создания графических элементов, а также анимации;

Проблемы, возникающие во время создания курсового проекта, решались изменением способа работы с объектами или добавлением дополнительных скриптов, для меньшей нагрузки обработчика

Данная курсовая работа может модифицироваться в дальнейшем для улучшения графических, программных и сюжетных возможностей:

1. Графические изменения (улучшение качества анимации персонажей, улучшение графической составляющей в виде текстур, объектов и элементов интерфейса);
2. Увеличение количества уровней.
3. Оптимизация игры путем уменьшения размера спрайтов и исправлений в коде
4. Добавление босса на главную локацию.
5. Добавление бонусов в игровые локации.

СПИСОК ИСПОЛЬЗОВАНЫХ ИСТОЧНИКОВ

1. Албахари Б., Албахари Д., C# 7.0. Карманный справочник «ДИАЛЕКТИКА» Москва 2017. – 1040 с.
2. Грин Дж., Стиллмен Э., Изучаемый С#. 3-е издание «ПИТЕР» СанктПетербург 2014. – 320 с.
3. Культин. Н.Б C# в задачах и примерах «БХВ-Петербург» СанктПетербург 2013. – 496 с.
4. Мартин Р., Чистый код «ПИТЕР» Санкт-Петербург 2013. – 873 с.
5. Рихтер Д., CLR via C# «ПИТЕР» Санкт-Петербург 2008. – 345 с.
6. Шарп Дж., Microsoft Visual C# Полное руководство «ПИТЕР» Санкт-Петербург 2017. – 540 с.
7. Шилдт Г., C# 4.0 Полное руководство «Вильямс», 2011. – 834 с.
8. METANIT.COM – Сайт о программировании. Режим доступа: https://metanit.com -- Дата доступа 25.04.2019.
9. Microsoft. Режим доступа: https://www.microsoft.com/ru-by/ – Дата доступа 20.04.2019.

**ПРИЛОЖЕНИЕ А**

(Обязательное)

Листинг программы

public class Goblin : MonoBehaviour

{

float dirX;

float moveSpeed = 3f;

public float firstPosition, secondPosition;

public static bool isAttacking = false;

public static Animator anim;

public static bool facingRight { get; private set; }

private GameObject \_hero;

private GameObject \_goblin;

private bool \_localscaleHero;

private bool \_localscaleGoblin;

Rigidbody2D rb;

Vector3 localScale;

int OntriggerInt;

private void Awake()

{

\_goblin = gameObject;

\_localscaleHero = false;

\_localscaleGoblin = false;

}

void Start()

{

OntriggerInt = 0;

facingRight = false;

localScale = transform.localScale;

rb = GetComponent<Rigidbody2D>();

dirX = -1f;

anim = GetComponent<Animator>();

}

private void OnCollisionExit2D(Collision2D collision)

{

OntriggerInt = 0;

isAttacking = false;

}

private void OnCollisionEnter2D(Collision2D collision)

{

if (collision.gameObject.CompareTag("Hero"))

{

OntriggerInt = 1;

isAttacking = true;

}

}

private void OnTriggerEnter2D(Collider2D collision)

{

if (collision.gameObject.CompareTag("Hero") && isAttacking)

{

if (Hero.healthPoints == 0)

{

isAttacking = false;

}

Hero.healthPoints -= 1;

}

}

void LateUpdate()

{

CheckWhereToFace();

}

void CheckWhereToFace()

{

if (dirX > 0)

facingRight = true;

else if (dirX < 0)

facingRight = false;

if (((facingRight) && (localScale.x < 0)) || ((!facingRight) && (localScale.x > 0)))

localScale.x \*= -1;

transform.localScale = localScale;

}

void Update()

{

if ((transform.position.x < firstPosition) || ((!facingRight) && (OntriggerInt == 1) && (!Hero.facingRight)) )

{

dirX = 1f;

}

if ((transform.position.x > secondPosition) || ((Hero.facingRight) && (facingRight) && (OntriggerInt == 1)))

{

dirX = -1f;

}

if (isAttacking)

{

anim.SetBool("isAttacking", true);

}

else

if (!isAttacking)

{

anim.SetBool("isAttacking", false);

}

}

private void FixedUpdate()

{

if (!isAttacking)

{

rb.velocity = new Vector2(dirX \* moveSpeed, rb.velocity.y);

}

else

rb.velocity = Vector2.zero;

}

}

public class CupScript : MonoBehaviour {

void OnTriggerEnter2D(Collider2D col)

{

if (LevelControlScript.sceneIndex == 6 && Hero.daggerInt == 1 )

{

LevelControlScript.instance.youWin();

}

if (LevelControlScript.sceneIndex != 6)

{

LevelControlScript.instance.youWin();

}

}

}

public class HealthBarScript : MonoBehaviour

{

Image healthBar;

float maxHealth = 100f;

public static float health;

void Start()

{

healthBar = GetComponent<Image>();

health = maxHealth;

}

void Update()

{

healthBar.fillAmount = health / maxHealth;

}

}

public class Hero : MonoBehaviour

{

int IsButtonhit = 0;

Rigidbody2D rb;

public Animator anim;

public static float dirX, moveSpeed = 5f;

public static int healthPoints;

public static int scoreCoinsLevel;

bool isHurting, isDead;

public static bool facingRight = false;

Vector3 localScale;

Text coinText;

public GameObject dagger; // Меч выпадающий из босса

public GameObject daggerImg; // Меч на Canvas

public static int daggerInt;

int PlatformInt;

public GameObject heath1, heath2, heath3, health4, health5;

public GameObject[] CanvasNarrator;

public GameObject[] Narrator;

public AudioSource[] audioSource;

bool isMoving = false;

int isHealthSold, isHealthSold2;

void Start()

{

facingRight = true;

Time.timeScale = 1;

scoreCoinsLevel = 0;

healthPoints = 3;

heath1.gameObject.SetActive(true);

heath2.gameObject.SetActive(true);

heath3.gameObject.SetActive(true);

isHealthSold = PlayerPrefs.GetInt("isHealthSold");

isHealthSold2 = PlayerPrefs.GetInt("isHealthSold2");

rb = GetComponent<Rigidbody2D>();

anim = GetComponent<Animator>();

localScale = transform.localScale;

daggerInt = 0;

if (LevelControlScript.sceneIndex == 6)

{

Narrator[1].SetActive(false);

}

if (isHealthSold == 1)

{

health4.SetActive(true);

healthPoints = 4;

}

if (isHealthSold2 == 1)

{

health5.SetActive(true);

healthPoints = 5;

}

}

void Update()

{

if (healthPoints > 5)

{

healthPoints = 5;

}

switch (healthPoints)

{

case 5:

heath1.gameObject.SetActive(true);

heath2.gameObject.SetActive(true);

heath3.gameObject.SetActive(true);

health4.gameObject.SetActive(true);

health5.gameObject.SetActive(true);

break;

case 4:

heath1.gameObject.SetActive(true);

heath2.gameObject.SetActive(true);

heath3.gameObject.SetActive(true);

health4.gameObject.SetActive(true);

health5.gameObject.SetActive(false);

break;

case 3:

heath1.gameObject.SetActive(true);

heath2.gameObject.SetActive(true);

heath3.gameObject.SetActive(true);

health4.gameObject.SetActive(false);

health5.gameObject.SetActive(false);

break;

case 2:

heath1.gameObject.SetActive(true);

heath2.gameObject.SetActive(true);

heath3.gameObject.SetActive(false);

health4.gameObject.SetActive(false);

health5.gameObject.SetActive(false);

break;

case 1:

heath1.gameObject.SetActive(true);

heath2.gameObject.SetActive(false);

heath3.gameObject.SetActive(false);

health4.gameObject.SetActive(false);

health5.gameObject.SetActive(false);

break;

case 0:

heath1.gameObject.SetActive(false);

heath2.gameObject.SetActive(false);

heath3.gameObject.SetActive(false);

health4.gameObject.SetActive(false);

health5.gameObject.SetActive(false);

if (healthPoints == 0)

{

dirX = 0;

Goblin.isAttacking = false;

anim.SetTrigger("isDead");

LevelControlScript.instance.youLose();

}

break;

}

if (CrossPlatformInputManager.GetButtonDown("Jump") && !isDead && PlatformInt == 1 || ( rb.velocity.y == 0 && CrossPlatformInputManager.GetButtonDown("Jump")))

{

rb.AddForce(Vector2.up \* 700f);

audioSource[1].Play();

}

if (rb.velocity.x != 0 && dirX != 0 && rb.velocity.y == 0)

isMoving = true;

else

isMoving = false;

if (isMoving)

{

if (audioSource[5] != null)

{

if (!audioSource[5].isPlaying && LevelControlScript.sceneIndex == 6)

audioSource[5].Play();

}

if (audioSource[0] != null)

{

if (!audioSource[0].isPlaying && LevelControlScript.sceneIndex != 6)

audioSource[0].Play();

}

}

else

{

if (audioSource[5] != null)

{

audioSource[5].Stop();

}

if (audioSource[0] != null)

{

audioSource[0].Stop();

}

}

SetAnimationState();

if (!isDead)

{

dirX = CrossPlatformInputManager.GetAxisRaw("Horizontal") \* moveSpeed; //Для телефона

}

}

void FixedUpdate()

{

if (!isHurting)

rb.velocity = new Vector2(dirX, rb.velocity.y);

}

void LateUpdate()

{

CheckWhereToFace();

}

public void ReturnAnamation()

{

anim.SetBool("isSlashing", false);

}

void SetAnimationState()

{

if (dirX == 0)

{

anim.SetBool("isWalking", false);

}

if (rb.velocity.y == 0 || PlatformInt==1)

{

anim.SetBool("isJumping", false);

anim.SetBool("isFalling", false);

}

if (Mathf.Abs(dirX) == 5 && rb.velocity.y == 0)

{

anim.SetBool("isWalking", true);

}

if (rb.velocity.y > 0 && PlatformInt == 0)

{

anim.SetBool("isJumping", true);

}

if (rb.velocity.y < 0 && PlatformInt == 0)

{

anim.SetBool("isJumping", false);

anim.SetBool("isFalling", true);

}

}

void CheckWhereToFace()

{

if (dirX > 0)

facingRight = true;

else if (dirX < 0)

facingRight = false;

if (((facingRight) && (localScale.x < 0)) || ((!facingRight) && (localScale.x > 0)))

localScale.x \*= -1;

transform.localScale = localScale;

}

private void OnCollisionExit2D(Collision2D collision)

{

PlatformInt = 0;

}

private void OnCollisionEnter2D(Collision2D collision)

{

if (collision.gameObject.CompareTag("Platform"))

{

PlatformInt = 1;

}

if (collision.gameObject.CompareTag("Goblin") && ButtonHit.Kicking == true)

{

audioSource[4].Play();

Destroy(collision.gameObject);

}

if (collision.gameObject.CompareTag("Boss") && ButtonHit.Kicking == true)

{

HealthBarScript.health -= 10f;

if (HealthBarScript.health == 0)

{

Destroy(collision.gameObject);

dagger.SetActive(true);

}

audioSource[4].Play();

}

if (collision.gameObject.CompareTag("Dagger"))

{

Destroy(collision.gameObject);

daggerImg.SetActive(true);

audioSource[6].Play();

daggerInt = 1;

Narrator[0].SetActive(false);

Narrator[1].SetActive(true);

}

}

private void OnTriggerExit2D(Collider2D collision)

{

if (collision.CompareTag("Narrator"))

CanvasNarrator[0].SetActive(false);

if (collision.CompareTag("NarratorSecond"))

CanvasNarrator[1].SetActive(false);

if (collision.CompareTag("Narrator3"))

CanvasNarrator[2].SetActive(false);

if (collision.CompareTag("Narrator4"))

CanvasNarrator[3].SetActive(false);

if (collision.CompareTag("Narrator5"))

CanvasNarrator[4].SetActive(false);

}

void OnTriggerEnter2D(Collider2D col)

{

if (col.CompareTag("Coin"))

{

PlayerPrefs.SetInt("Coin", PlayerPrefs.GetInt("Coin") + 1);

PlayerPrefs.Save();

audioSource[2].Play();

Destroy(col.gameObject);

scoreCoinsLevel += 1;

}

if (col.CompareTag("BoxColiderLoseDownLevel"))

{

healthPoints -= healthPoints;

}

if (col.CompareTag("Narrator"))

CanvasNarrator[0].SetActive(true);

if (col.CompareTag("NarratorSecond"))

CanvasNarrator[1].SetActive(true);

if (col.CompareTag("Narrator3"))

CanvasNarrator[2].SetActive(true);

if (col.CompareTag("Narrator4"))

CanvasNarrator[3].SetActive(true);

if (col.CompareTag("Narrator5"))

CanvasNarrator[4].SetActive(true);

if (col.tag==("Hearth"))

{

healthPoints += 1;

Destroy(col.gameObject);

}

if (col.CompareTag("Goblin"))

{

if (healthPoints > 0)

{

anim.SetTrigger("isHurting");

audioSource[3].Play();

StartCoroutine("Hurt");

}

}

if (col.CompareTag("Fire"))

{

healthPoints -= 1;

if (healthPoints > 0)

{

anim.SetTrigger("isHurting");

audioSource[3].Play();

StartCoroutine("Hurt");

}

if (healthPoints == 0)

{

dirX = 0;

anim.SetTrigger("isDead");

LevelControlScript.instance.youLose();

}

}

}

IEnumerator Hurt()

{

isHurting = true;

rb.velocity = Vector2.zero;

if (facingRight)

rb.AddForce(new Vector2(-200f, 200f));

else

rb.AddForce(new Vector2(200f, 200f));

yield return new WaitForSeconds(0.5f);

isHurting = false;

}

}

public class LevelControlScript : MonoBehaviour {

Text textLoseScoreCoins;

Text textWiScoreCoins;

public static LevelControlScript instance = null;

public GameObject scoreCoinsLose;

public GameObject scoreCoinsWin;

public static GameObject levelSign;

public static int sceneIndex, levelPassed;

public GameObject LoseMenu;

public GameObject WinMenu;

void Start () {

Goblin.isAttacking = false;

Application.targetFrameRate = 61;

if (instance == null)

instance = this;

else if (instance != this)

Destroy (gameObject);

levelSign = GameObject.Find ("LevelNumber");

sceneIndex = SceneManager.GetActiveScene().buildIndex;

levelPassed = PlayerPrefs.GetInt ("LevelPassed");

PlayerPrefs.Save();

textLoseScoreCoins = scoreCoinsLose.gameObject.GetComponent<Text>();

textWiScoreCoins = scoreCoinsWin.gameObject.GetComponent<Text>();

}

public void youWin()

{

levelSign.gameObject.SetActive(false);

textWiScoreCoins.text = "Coins: " + Hero.scoreCoinsLevel.ToString();

if (LevelControlScript.levelPassed < LevelControlScript.sceneIndex)

PlayerPrefs.SetInt("LevelPassed", LevelControlScript.sceneIndex - 1);

Invoke("Win", 1f);

}

public void youLose()

{

levelSign.gameObject.SetActive (false);

textLoseScoreCoins.text = "Coins: " + Hero.scoreCoinsLevel.ToString();

Invoke("Lose", 1f);

}

void loadNextLevel()

{

SceneManager.LoadScene (sceneIndex+1);

}

void loadMainMenu()

{

SceneManager.LoadScene ("LevelSelect");

}

void Lose()

{

Time.timeScale = 0;

Goblin.isAttacking = false;

LoseMenu.SetActive(true);

}

void Win()

{

Time.timeScale = 0;

Goblin.isAttacking = false;

WinMenu.SetActive(true);

}

}

public class LevelNumberScript : MonoBehaviour {

Text text;

int sceneIndex;

void Start () {

text = GetComponent<Text> ();

sceneIndex = SceneManager.GetActiveScene ().buildIndex-1;

}

void Update () {

if (sceneIndex == 6)

{

text.text = "Training camp";

}

if (sceneIndex == 5)

{

text.text = "Ice Golem";

}

else

text.text = "Level " + sceneIndex;

}

}

public class LoseMenu : MonoBehaviour

{

public string nameScene;

public void MainMenu()

{

Time.timeScale = 1;

SceneManager.LoadScene("LevelSelect");

}

public void Quit()

{

Application.Quit();

}

public void ResetLevel()

{

Time.timeScale = 1; // Время не приостановлено

SceneManager.LoadScene(nameScene);

}

}

public class MainMenuControlScript : MonoBehaviour {

public Button Button2Level, Button3Level, Button4Level, Button5Level;

int levelPassed;

public AudioSource audioSource;

void Start () {

levelPassed = PlayerPrefs.GetInt ("LevelPassed");

Button2Level.interactable = false;

Button3Level.interactable = false;

Button4Level.interactable = false;

Button5Level.interactable = false;

if (levelPassed > 4)

{

levelPassed = 4;

}

switch (levelPassed) {

case 1:

Button2Level.interactable = true;

break;

case 2:

Button2Level.interactable = true;

Button3Level.interactable = true;

break;

case 3:

Button2Level.interactable = true;

Button3Level.interactable = true;

Button4Level.interactable = true;

break;

case 4:

Button2Level.interactable = true;

Button3Level.interactable = true;

Button4Level.interactable = true;

Button5Level.interactable = true;

break;

}

}

public void levelToLoad (int level) //Загрузка сцены указывается в Unity на OnClick

{

audioSource.Play();

SceneManager.LoadScene (level+1);

}

public void resetPlayerPrefs()

{

Button2Level.interactable = false;

Button3Level.interactable = false;

Button4Level.interactable = false;

Button5Level.interactable = false;

PlayerPrefs.DeleteAll ();

}

public void Menu()

{

audioSource.Play();

SceneManager.LoadScene("Menu");

}

}

public class MenuControls : MonoBehaviour

{

MainMenuControlScript main;

public AudioSource audioSource;

int level;

public MenuControls(MainMenuControlScript main)

{

this.main = main;

}

public void PlayPressed()

{

audioSource.Play();

SceneManager.LoadScene("LevelSelect");

}

public void TrainingScene()

{

audioSource.Play();

SceneManager.LoadScene("Training");

}

public void resetPlayerPrefs()

{

audioSource.Play();

PlayerPrefs.DeleteAll();

}

public void Shop()

{

audioSource.Play();

SceneManager.LoadScene("Shop");

}

public void ExitPressed()

{

audioSource.Play();

Application.Quit();

}

private void Start()

{

level = PlayerPrefs.GetInt("LevelPassed");

Debug.Log(level);

Application.targetFrameRate = 61;

}

}

public class Pause : MonoBehaviour

{

public AudioSource audioSource;

public GameObject PauseMenu;

public void pauseGame()

{

Time.timeScale = 0; //Время остонавливается

PauseMenu.SetActive(true);

}

public void Resume()

{

Time.timeScale = 1;

PauseMenu.SetActive(false);

}

public void MainMenu()

{

Time.timeScale = 1;

SceneManager.LoadScene("LevelSelect");

}

public void Quit()

{

Application.Quit();

}

}

public class Reset : MonoBehaviour

{

public void Resets()

{

PlayerPrefs.DeleteAll();

}

}

public class Score : MonoBehaviour

{

Text scoreText;

void Start()

{

scoreText = GetComponent<Text>();

}

void Update()

{

scoreText.text = Convert.ToString(PlayerPrefs.GetInt("Coin")) + " $";

}

}

public class ScrollScript : MonoBehaviour

{

float scroollSpeed = -5f;

Vector2 startPos;

void Start()

{

startPos = transform.position;

}

void Update()

{

float newPos = Mathf.Repeat(Time.time \* scroollSpeed, 20);

transform.position = startPos + Vector2.right \* newPos;

}

}

public class ShopControlScript : MonoBehaviour

{

int moneyAmount; //Сумма денег

public static int isHealthSold; // Бонус Здоровье продано

int isHealthSold2;

int isLevelUpSold;

int isLevelUpSold2;

public Text moneyAmountText;

public Text healthPrice;

public Text levelUpPrice;

public Button buyButton;

public Button buyButtonLevelUp;

private void Start()

{

buyButton.interactable = false;

buyButton.interactable = false;

moneyAmount = PlayerPrefs.GetInt("Coin");

}

private void Update()

{

moneyAmountText.text = "Money: " + moneyAmount.ToString() + "$";

isLevelUpSold = PlayerPrefs.GetInt("isLevelUpSold");

isLevelUpSold = PlayerPrefs.GetInt("isLevelUpSold2");

isHealthSold2 = PlayerPrefs.GetInt("isHealthSold2");

isHealthSold = PlayerPrefs.GetInt("isHealthSold");

if(moneyAmount >= 200 && isLevelUpSold == 0)

buyButtonLevelUp.interactable = true;

if (moneyAmount < 200 && isLevelUpSold == 0)

buyButtonLevelUp.interactable = false;

if (moneyAmount >= 250 && isLevelUpSold == 1)

buyButtonLevelUp.interactable = true;

if (moneyAmount < 250 && isLevelUpSold == 1)

buyButtonLevelUp.interactable = false;

if (isLevelUpSold == 1 && isLevelUpSold2 == 0)

levelUpPrice.text = "Price: 250 $";

if (isLevelUpSold == 1 && isLevelUpSold2 == 1)

{

levelUpPrice.text = "Max Level Up";

buyButtonLevelUp.interactable = false;

}

// Продажа другого бонуса

if (moneyAmount >= 75 && isHealthSold == 0)

buyButton.interactable = true;

if(moneyAmount < 75 && isHealthSold == 0)

buyButton.interactable = false;

if (moneyAmount >= 100 && isHealthSold == 1)

buyButton.interactable = true;

if (moneyAmount < 100 && isHealthSold == 1)

buyButton.interactable = false;

if(isHealthSold == 1 && isHealthSold2 == 0)

healthPrice.text = "Price: 100 $";

if(isHealthSold == 1 && isHealthSold2 == 1)

{

healthPrice.text = "Max Health";

buyButton.interactable = false;

}

}

public void BuyLevelUp()

{

if (isLevelUpSold == 0)

{

moneyAmount -= 200;

PlayerPrefs.SetInt("isLevelUpSold", 1);

PlayerPrefs.SetInt("LevelPassed", PlayerPrefs.GetInt("LevelPassed") + 1);

levelUpPrice.text = "Price: 250 $";

}

if (isLevelUpSold == 1 && isLevelUpSold2 == 0)

{

moneyAmount -= 250;

PlayerPrefs.SetInt("isLevelUpSold2", 1);

PlayerPrefs.SetInt("LevelPassed", PlayerPrefs.GetInt("LevelPassed") + 1);

healthPrice.text = "Max Level Up";

buyButtonLevelUp.gameObject.SetActive(false);

}

PlayerPrefs.SetInt("Coin", moneyAmount);

PlayerPrefs.Save();

}

public void BuyHealth()

{

if(isHealthSold == 0)

{

moneyAmount -= 75;

PlayerPrefs.SetInt("isHealthSold", 1);

healthPrice.text = "Price: 100 $";

}

if (isHealthSold == 1 && isHealthSold2 == 0)

{

moneyAmount -= 100;

PlayerPrefs.SetInt("isHealthSold2", 1);

healthPrice.text = "Max Health";

buyButton.gameObject.SetActive(false);

}

PlayerPrefs.SetInt("Coin",moneyAmount);

PlayerPrefs.Save();

}

public void Exit()

{

SceneManager.LoadScene("Menu");

}

}

public class Spider : MonoBehaviour

{

public float dirX;

public float moveSpeed = 0.1f;

public float first;

public float second;

Rigidbody2D rb;

void Start()

{

rb = GetComponent<Rigidbody2D>();

}

void Update()

{

if (transform.position.y < first)

dirX = 1f;

else if (transform.position.y > second)

dirX = -1f;

}

void FixedUpdate()

{

rb.velocity = new Vector2(0, dirX \* moveSpeed);

}

}

public class WinMenu : MonoBehaviour

{

public void MainMenu()

{

Debug.Log(LevelControlScript.sceneIndex == 6);

if (LevelControlScript.sceneIndex == 6)

{

SceneManager.LoadScene("LevelSelect");

Time.timeScale = 1;

if (LevelControlScript.levelPassed < LevelControlScript.sceneIndex)

PlayerPrefs.SetInt("LevelPassed", LevelControlScript.sceneIndex - 1);

}

else

{

if (LevelControlScript.levelPassed < LevelControlScript.sceneIndex)

PlayerPrefs.SetInt("LevelPassed", LevelControlScript.sceneIndex - 1);

LevelControlScript.levelSign.gameObject.SetActive(false);

SceneManager.LoadScene(LevelControlScript.sceneIndex + 1);

Invoke("loadNextLevel", 1f);

}

}

void loadMainMenu()

{

SceneManager.LoadScene("LevelSelect");

}

void loadNextLevel()

{

SceneManager.LoadScene(LevelControlScript.sceneIndex + 1);

}

}