СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Основой дипломного проекта является разработка компьютерной игры “Path in the forest” в жанре 2D-платформер на Unity.

В процессе разработки изучаются возможности программы Visual Studio 2019 и игрового движка Unity. Во время разработки должны быть учтены различные аспекты, которые помогут пользователю освоить приложение, для этого необходимо создать понятный и простой интерфейс, обучение, где пользователь сможет без проблем усвоить различные механики игры, что бы во время игрового процесса (gameplay) у него не возникало сложностей.

“Path in the forest” является игровым приложением в известном жанре двухмерного платформера. Основу игрового процесса этого вида игр составляют прыжки по платформам, сбор различных предметов и/или сражения с врагами с победой над ними для завершения уровня. При желании разработчик может добавить в свою игру дополнительные механики, которые помогут выделить игру на фоне остальных.

Данный проект посвящён разработке технической демонстрационной версии компьютерной игры «Path in the forestt». Проект относится к жанру двумерных платформеров.

1 Теоретические основы для создания компьютерной игры “Path in the forest”

1.1 Постановка задачи

Задачей данной работы является разработка технической демонстрационной версии компьютерной игры “Path in the forest” в жанре 2D-платформер на Unity.

Техническая демонстрация, техническая демонстрационная версия, техно-демоверсия (англ. tech demo) - прототип, приближенный пример или неполная версия продукта, которая создана с целью продемонстрировать идею, производительность, метод или особенности какого-либо программного продукта. Технические демонстрационные версии могут использоваться как демонстрации для инвесторов, партнёров, журналистов или даже потенциальных потребителей для того, чтобы убедить их в жизнеспособности данного продукта, материального товара или идеи.

1.2 Способы решения задачи. Преимущества и недостатки

Основным средством разработки данного проекта является игровой движок Unity и Microsoft Visual Studio 2019.

Unity — межплатформенная среда разработки компьютерных игр, разработанная американской компанией Unity Technologies.

Основными преимуществами Unity являются наличие визуальной среды разработки, межплатформенной поддержки и модульной системы компонентов.

В первую очередь движок Unity дает возможность разрабатывать игры, не требуя для этого каких-то особых знаний. Здесь используется компонентно-ориентированный подход, в рамках которого разработчик создает объекты (например, главного героя) и к ним добавляет различные компоненты (например, визуальное отображение персонажа и способы управления им).

Благодаря удобному Drag & Drop интерфейсу и функциональному графическому редактору движок позволяет рисовать карты и расставлять объекты в реальном времени и сразу же тестировать получившийся результат.

Второе преимущество движка – наличие огромной библиотеки ассетов и плагинов, с помощью которых можно значительно ускорить процесс разработки игры. Их можно импортировать и экспортировать, добавлять в игру целые заготовки – уровни, врагов, паттерны поведения ИИ и так далее.

Третья сильная сторона Unity 3D – поддержка огромного количества платформ, технологий, API. Созданные на движке игры можно легко портировать между ОС Windows, Linux, OS X, Android, iOS, на консоли семейств PlayStation, Xbox, Nintendo, на VR- и AR-устройства.

Unity поддерживает DirectX и OpenGL, работает со всеми современными эффектами рендеринга, включая новейшую технологию трассировки лучей в реальном времени.

К недостаткам относят появление сложностей при работе с многокомпонентными схемами и затруднения при подключении внешних библиотек.

Создание масштабных, сложных сцен с множеством компонентов может негативно повлиять на производительность игры, в результате чего разработчикам придется потратить дополнительное время и ресурсы на оптимизацию, а возможно – и удаление некоторых элементов из проекта.

Кроме того, приложения, созданные на Unity, довольно «тяжеловесны»: даже самая простая пиксельная игра может занимать несколько сотен мегабайт на ПК.

В качестве редактора кода был выбран Visual Studio.

Интегрированная среда разработки Visual Studio является творческой стартовой площадкой, которую можно использовать для редактирования, отладки и сборки кода, а также для публикации приложения. В дополнение к стандартному редактору и отладчику, предоставляемых большинством интегрированных сред разработки, Visual Studio включает компиляторы, средства завершения кода, графические конструкторы и многие другие функции для улучшения процесса разработки программного обеспечения.

Ниже перечислены основные преимущества IDE-среды Visual Studio.

Встроенный Web-сервер. Для обслуживания Web-приложения ASP.NET необходим Web-сервер, который будет ожидать Web-запросы и обрабатывать соответствующие страницы. Наличие в Visual Studio интегрированного Web-сервера позволяет запускать Web-сайт прямо из среды проектирования, а также повышает безопасность, исключая вероятность получения доступа к тестовомуWeb-сайту с какого-нибудь внешнего компьютера, поскольку тестовый сервер может принимать соединения только с локального компьютера.

Поддержка множества языков при разработке. Visual Studio позволяет писать код на своем языке или любых других предпочитаемых языках, используя все время один и тот же интерфейс (IDE). Более того, Visual Studio также еще позволяет создавать Web-страницы на разных языках, но помещать их все в одно и то же Web-приложение.

Меньше кода для написания. Для создания большинства приложений требуется приличное количество стандартного стереотипного кода, и Web-страницы ASP. NET тому не исключение.

Интуитивный стиль кодирования. По умолчанию Visual Studio форматирует код по мере его ввода, автоматически вставляя необходимые отступы и применяя цветовое кодирование для выделения элементов типа комментариев. Такие незначительные отличия делают код более удобным для чтения и менее подверженным ошибкам.

Более высокая скорость разработки. Многие из функциональных возможностей Visual Studio направлены на то, чтобы помогать разработчику делать свою работу как можно быстрее. Удобные функции, вроде функции IntelliSense (которая умеет перехватывать ошибки и предлагать правильные варианты), функции поиска и замены (которая позволяет отыскивать ключевые слова как в одном файле, так и во всем проекте) и функции автоматического добавления и удаления комментариев (которая может временно скрывать блоки кода), позволяют разработчику работать быстро и эффективно.

Возможности отладки. Предлагаемые в Visual Studio инструменты отладки являются наилучшим средством для отслеживания загадочных ошибок и диагностирования странного поведения. Разработчик может выполнять свой код по строке за раз, устанавливать интеллектуальные точки прерывания, при желании сохраняя их для использования в будущем, и в любое время просматривать текущую информацию из памяти.

В качестве недостатка можно отметить невозможность отладчика (Microsoft Visual Studio Debugger) отслеживать в коде режима ядра. Отладка в Windows в режиме ядра в общем случае выполняется при использовании WinDbg, KD или SoftICE.

Но помимо Visual Studio есть и другие редакторы, которые были рассмотрены для создания проекта. О них подойдёт речь ниже.

Sublime Text

Sublime Text — кроссплатформенный текстовый редактор для написания программного кода на различных языках программирования (Groovy, Erlang, C+, Java и т.д.), а также верстки веб-документов.

Его основными преимуществами являются в первую очередь широкая функциональность, за счет достаточно разнообразного набора базовых (встроенных) возможностей редактора, а также подключаемых расширений Sublime Text представляет собой, по сути, упрощенную среду разработки, а не просто инструмент для написания кода

Он также поддерживает множество языков программирования, и автоматически подстраивается под выбранный язык программирования — в частности, исправляет специфичные ошибки, подсвечивает синтаксис, предлагает несколько вариантов заполнения.

Не мало важно, что редактор обладает хорошим быстродействием, благодаря чему он может запускаться даже на ПК. Также он обладает простым и понятным интерфейсом, и еще широкой поддержкой.

К недостаткам можно отнести в первую очередь отнести платную лицензию, которая составляет 99 долларов США. Так же многие подключаемы

модули, разработанные Асторонними программистами, могут вызвать сбои в редакторе.

Atom

Atom — это текстовый редактор с открытым исходным кодом, разработанный в 2014 году GitHub. Созданный с использованием Node JS и HTML, он поддерживает ОС Windows, Mac и Linux.

Главными преимуществами редактора в первую очередь его визуальная составляющая – он богат такими функциями, как мини-карты отдельных папок, деревья папок для организации в стиле перетаскивания и более 2 900 тем. Которые вы можете загрузить прямо со страницы их тем. Помимо визуальной ориентации, Atom также полностью настраивается. В программе есть замечательные плагины и пакеты, которые позволяют пользователям создавать практически всё, что угодно. Поскольку они были разработаны GitHub, интеграция Atom с Git безупречна.

Основным недостатком является его скорость. При работе с длинными списками расширений, файлов и плагинов она начинает тормозить. Так же сам редактор не подходит слабым ПК, так как из-за хороший визуальной части, как и при работе с выше упомянутыми компонентами будет сильно тормозить.

1.3 Выбор алгоритма решения задачи и определение его сложности

В ходе разработки компьютерной игры было решено выбрать движок Unity по причинам, описанным выше. В первую очередь движок распространяется бесплатно, также у движка есть большое сообщество, которое делится своим опытом разработки и готово помочь в сложных вопросах.

Ещё одной причиной выбора Unity стало поддержка объектно-ориентированного языка программирования C#, на котором будет писаться код программы и интеграция программы Visual Studio.

Специальные библиотеки для Unity загружаются отдельно в Visual Studio, через интерфейс самой Visual Studio.

1.4 Определение модели проектирования и технологии программирования

За последние 15 – дет объектно-ориентированная технология стала основным методом промышленной разработки программного обеспечения.

Вместе с развитием объектно-ориентированного программирования развивались и объектно-ориентированные методы разработки программного обеспечения, охватывающие стадии анализа и проектирования.

Основные идеи объектно-ориентированного подхода опираются на следующие положения:

– Программа представляет собой модель некоторого реального процесса, части реального мира: модель содержит не все признаки и свойства представляемой ею части реального мира, а только те, которые нужны для разрабатываемой программной части

– Модель реального мира или его части может быть описана как совокупность взаимодействующих между собой объектов

– Объект описывается набором атрибутов (свойств), значения которых определяет состояние объекта, и набором операций (действий), которые может выполнить объект

– Взаимодействие объектов осуществляется посылкой специальных сообщений от одного объекта к другому

– Объекты, описанные одним и тем же набором атрибутов и способные выполнить один и тот же набор операция, представляют собой класс однотипных объектов

Объектно-ориентированный подход дает следующие преимущества:

– Уменьшает сложность программного обеспечения

– Повышает его надежность

– Обеспечивает возможность модификации отдельных компонентов программ без изменения остальных компонентов

– Обеспечивает возможность повторно использовать отдельные компоненты программного обеспечения;

Систематическое применение объектно-ориентированного подхода позволяет разрабатывать хорошо структурированные, надежные в эксплуатации, достаточно модифицируемые программные системы. Это объясняет интерес программистов к объектно-ориентированному подходу и объектно-ориентированным языкам программирования.

На данный момент все ныне существующие игровые движки используют параллельные вычисления, однако не во всех аспектах – так, например многопоточной обработке подвергаются рендеринг изображения, обработка физических вычислений, обработка звуковых дорожек. Несмотря на это, логика игры, ее правила и механики обрабатываются в одном потоке одного ядра ЦП, что, при большом масштабе проекта выливается в так называемые “фризы” – моменты, когда в очереди на выполнение оказывается множество команд и время обработки значительно вырастает.

Решить данную проблему призван архитектурный паттерн Entity – Component – System (ECS), который заменяет традиционную для разработки игр парадигму объектно-ориентированного программирования на параллельное программирование и работу с данными.

Проведем сравнительный анализ наиболее популярных игровых движков:

Unity

Unity является одним из популярных движков, использующий язык программирования C#. Данный продукт предоставляет множество платных и бесплатных ассетов, которые помогут разработчикам в создании игры. Также существует пробная версия дополнительных программных пакетов D.O.T.S., позволяющая применять параллельные вычисления ко всем игровым составляющим — логика, физика, видео- и звуковые потоки.

Unreal Engine

Unreal Engine универсальный и очень гибкий движок, использующий язык программирования C++. Он наполнен множеством бесплатных инструментов разработки. Начинающим разработчикам в работе поможет визуальный редактор Blueprints, позволяющий создать игру без единой строки кода. Его преимуществом является поддержка UltraHD разрешения графики и прямая работа с памятью с помощью встроенных библиотек. Параллельные вычисления можно осуществлять с помощью встроенного в C++ планировщика задач, однако его невозможно применить к чему-либо, кроме логической части игры и физическим вычислениям.

CryEngine

CryEngine на данный момент представляет разработчикам возможность делать в своих проектах фотореалистичную графику. Данный движок предоставляет встроенную поддержку мультипоточных вычислений, однако, так же, как и в Unreal Engine, они применимы лишь на логическую и физическую части игрового проекта.

Godot

Godot развивающийся движок, использующий собственный язык программирования. Присутствует поддержка C++ и C#, в скором времени и Python. Идеально подойдет для знакомства с разработкой игр, если у пользователя уже есть небольшой опыт в программировании. Также имеет встроенный функционал распараллеливания процессов с помощью планировщика задач.

GameMaker Studio

GameMaker Studio — один из самых известных игровых движков наравне с Unity и Unreal Engine. На нем сделаны многие инди-хиты вроде Undertale, а еще у него очень низкий порог вхождения: для GameMaker необязательно умение программировать, и все взаимодействия можно настраивать буквально «перетягиванием».

Впрочем, писать код в нем тоже можно — создатели даже разработали специальный язык, GML (Game Maker Language). У него меньше

возможностей, чем у того же C#, а некоторые его особенности не встречаются в других языках, зато он очень прост в освоении и подойдет как опытным программистам, так и новичкам.

GameMaker Studio не просто хорошо оптимизирован — в нем немало инструментов для работы над играми в любых жанрах. Так что это хороший выбор для тех, кто хочет погрузиться в геймдизайн и не тратить время на «математику» и поиск расширений.

Он не понравится тем, кто хочет работать с 3D: движок не поддерживает его официально, а разработчики не планируют что-то с этим делать. Да и магазин ассетов GameMaker гораздо беднее, чем у конкурентов.

Defold

Defold — один из главных конкурентов Unity, но используют его намного реже. Он рассчитан на двухмерные игры, которые программируются на языке Lua и запускаются почти на всех платформах — от HTML5 (веб-страницы) до Nintendo Switch. Поэтому он быстро работает и компилирует маловесные билды, в отличие от того же Unity.

У Defold нет встроенного визуального программирования, но есть фанатский плагин, который его добавляет. Также в сети опубликовано много обучающего материала по нему, но меньше, чем у конкурентов.

Как и любой молодой движок, Defold постоянно развивается, так что все недостатки могут исправить в будущем. Сейчас же это необычная альтернатива Unity для тех, кто хочет попробовать себя в программировании, но не желает погружаться в трудности семейства C, общепринятого в «большой» индустрии.

1.5 Выбор инструментальных средств

Инструментарий программирования — это совокупность программных продуктов, обеспечивающих технологию разработки, отладки и внедрения создаваемых новых программных продуктов. Они делятся на средства для создания приложений и средства для создания информационных систем (Case-технологии).

Средствами для создания проекта послужили следующие инструменты:

C# (произносится *си-шарп*) — объектно-ориентированный язык программирования. Компилятор с C# входит в стандартную установку самой .NET, поэтому программы на нём можно создавать и компилировать даже без инструментальных средств, вроде Visual Studio. C# относится к семье языков с C-подобным синтаксисом, из них его синтаксис наиболее близок к C++ и Java. Язык имеет статическую типизацию, поддерживает полиморфизм, перегрузку операторов (в том числе операторов явного и неявного приведения типа), делегаты, атрибуты, события, свойства, обобщённые типы и методы, итераторы, анонимные функции с поддержкой замыканий, LINQ, исключения, комментарии в формате XML. Переняв многое от своих предшественников — языков C++, Java, Delphi, Модула и Smalltalk — C#, опираясь на практику их использования, исключает некоторые модели, зарекомендовавшие себя как проблематичные при разработке программных систем: так, C# не поддерживает множественное наследование классов (в отличие от C++).

C# разрабатывался как язык программирования прикладного уровня для CLR и, как таковой, зависит, прежде всего, от возможностей самой CLR. Это касается, прежде всего, системы типов C#, которая отражает BCL. Присутствие или отсутствие тех или иных выразительных особенностей языка диктуется тем, может ли конкретная языковая особенность быть транслирована в соответствующие конструкции CLR. Так, с развитием CLR от версии 1.1 к 2.0 значительно обогатился и сам C#; подобного взаимодействия следует ожидать и в дальнейшем. (Однако эта закономерность была нарушена с выходом C# 3.0, представляющим собой расширения языка, не опирающиеся на расширения платформы .NET.) CLR предоставляет C#, как и всем другим .NET-ориентированным языкам, многие возможности, которых лишены «классические» языки программирования. Например, сборка мусора не реализована в самом C#, а производится CLR для программ, написанных на C# точно так же, как это делается для программ на VB.NET, J# и др.

2. Практическая реализация компьютерной игры “Path in the forest”

2.1 Построение математической модели решения задачи разработки компьютерной игры “Path in the forest”

Математика в дизайне видеоигр помогает программистам создавать прекрасно реализованные миры. Используя базовую геометрию, дизайнеры могут создавать изометрические фоны, создающие иллюзию трехмерного пространства. Они также могут использовать геометрию для создания более сложных трехмерных миров и персонажей.

Помимо геометрии, разработчики игр также используют математику для расчета пространства между объектами и расстояния, которое может пройти игрок. Математика помогает дизайнерам правильно размещать уступы и платформы в нужных местах, чтобы игроки могли до них добраться. Это также позволяет им определять, насколько высокими должны быть стены или как далеко делать промежутки, чтобы заблокировать определенные области мира.

Расположение платформ и их высота должна учитывать силу прыжка игрока, в противном случае игрок может и не допрыгнуть до нужного места. Поэту нужно узнать высоту и возможную длину прыжка. Для этого надо учитывать начальную скорость и время, которое он проводит в воздухе.

Формула для вычисления максимальной высоты прыжка:

$h = (v^2 * \sin^2(\alpha)) / (2 * g)$

где h - максимальная высота прыжка, v - начальная скорость прыжка, α - угол между начальной скоростью и горизонтом (обычно равен 90 градусам для вертикального прыжка), g - ускорение свободного падения (обычно принимается равным 9.8 м/с^2).

Для вычисления времени полета можно использовать формулу:

$t = 2 * v * \sin(\alpha) / g$

где t - время полета.

Зная эти значения, можно также вычислить дальность полета по формуле:

$d = v * \cos(\alpha) * t$

где d - дальность полета

Что бы игрок и/или объект мог передвигаться и совершать прыжок, нужно постоянно делать проверку на наличие поверхности под его ногами, для этого создается переменная типа bool со значением false, потом создается массив коллайдоров. После чего переменной присваивается значение массива коллайдоров, и если длина массива больше единицы, то объект стоит на твердой поверхности.

Что бы игрок мог взаимодействовать физически с объектами (т.е. двигать их, получать от них какое-либо эффект и т.д.) нужно прикрепить к объекту компонент Rigidbode2D (в нашем случае используется компонент, предназначенный для работы с двумерным пространством). И уже в нем настраивать базовые физические свойства.

Математические векторы используются в играх для задания направления объектам, а длина величины будет определять скорость, с которой они будут двигаться

Для передвижения игрока и/или другого какого любо объекта его направления создается вектор, который умножается на время в секундах, которое потребовалось для отрисовки последнего кадра, а оно в свою очередь умножается на переменную скорости, которую задает разработчик.

T = Vector * (deltaTime * speed)

где T – метод transform.position

где deltaTime - время в секундах, которое потребовалось для отрисовки последнего кадра

Скорость персонажа рассчитывается по формуле:

$V = S/t$

Похожий способ используется для создания движение огненного шара, которым, атакует главная героиня монстров. Только тут просто метод transform.right умножается на скорость. И если шар на своем пути в течении 0.4 секунд не сталкивается с каким-либо объектом он просто уничтожается.

Rb = Transform.right * speed

Где rb – твердое тело

Скорость огненного шара рассчитывается, как и скорость персонажа.

Ещё для атаки создается время перезарядки, что бы игрок не мог беспрерывно атаковать. Для этого нужно создать две переменные типа float одна будет отвечать за время перезарядки атаки, а другая за таймер перезарядки и цикл if. В цикле задается условие, что кнопка атаки и таймер перезарядки должны быть больше переменной время перезарядки атаки. Если это так, то переменной отвечающей за таймер перезарядки прибавляется время в секундах, которое потребовалось для отрисовки последнего кадра, затем вызывается метод, отвечающий за анимацию атаки и таймер равен нулю.

Для разворота игрока или другого объекта требуется сделать проверку переменной, отвечающей за передвижение на значение нуля, и если оно равно нулю, то нужно наследовать структуру Quaternion, отвечающие за разворот объекта.

Что бы совершить прыжок, нужно применить к переменной типа Rigidbodey2D функцию AddForce умноженную на метод transform.up умноженный на переменную отвечающий за силу прыжка.

Rb = transform.up * JumpForce

2.2 Организация входных и выходных данных компьютерной игры “Path in the forest”

Входными данными для игры служат:

– Состояние персонажа: его очки здоровья, заполненность слотов в инвентаре, его скорость, нахождение в пространстве (находится ли он воздухе или стоит на земле);

– Расположение других персонажей и противников

– Расположение уровня и платформ

– Нажатые клавиши игроком;

Выходными данными служат:

– Отображение уровня и персонажа на экране

– Анимация объектов

– Изменение состояние персонажа и других объектов в зависимости от действий игрока;

2.3 Описание основных модулей программного продукта “Path in the forest”

Функция для вызова основных методов:

– Функция: void Update()

– Цель: Вызвать методы Run(), Jump(), отображение здоровья персонажа

– Исходные данные: Отсутствуют

– Результат: Вызывание сопутствующих методов

– Вызываемые модули: Run(), Jump()

– Описание алгоритма: Update вызывается раз за кадр. Это главная функция для обновлений кадров.

Функция для бега:

– Функция: void Run()

– Цель: Запуск алгоритма бега

– Исходные данные: Отсутствуют

– Результат: Игрок движется

– Вызываемые модули: Нет

– Описание алгоритма: При нажатии клавиши выбирается “бег” из списка состояний, проигрывается нужная анимация, а так же запускается алгоритм для передвижения объекта

Функция для прыжка:

– Функция: void Jump()

– Цель: Запуск алгоритма прыжка

– Исходные данные: Отсутствуют
– Результат: Игрок прыгает
– Вызываемые модули: Нет
– Описание алгоритма: При нажатии клавиши выбирается “прыжок” из списка состояний, проигрывается нужная анимация, а так же запускается алгоритм для вертикального передвижения объекта.

Функция для получения урона:

– Функция: void GetDamage()
– Цель: Получение урона
– Исходные данные: Нет
– Результат: Игрок либо монстр получает урон
– Вызываемые модули: Die()
– Описание алгоритма: При столкновение с монстром игрок получит урон или монстр получит урон от огненного шара игрока

Функция для создания условий при старте игры:

– Функция: void Awake()
– Цель: Создание условий, заданных разработчиком при старте
– Исходные данные: Жизни игрока, получение компонентов из Rigidbody2D, Animator, SpriteRenderer
– Результат: Получение данных/условий
– Вызываемые модули: Нет
– Описание алгоритма: Эта функция всегда вызывается до любых функций Start и также после того, как префаб был вызван в сцену (если GameObject неактивен на момент старта, Awake не будет вызван, пока GameObject не будет активирован, или функция в каком-нибудь прикреплённом скрипте не вызовет Awake).

Функция для управления диалоговой системой:

– Функция: void StartDialog(Dialog dialog)
– Цель: Вызвать диалог
– Исходные данные: Нет

– Результат: Запуск диалогового окна

– Вызываемые модули: DisplayNextSentence();

– Описание алгоритма: Появление кнопки начать диалог, после нажатия появляется окно диалогом, в котором появляются последовательно предложения.

Функция взаимодействия с предметом:

– Функция: void OnTriggerEnter2D(Collider2D other)

– Цель: Взаимодействие с предметом

– Исходные данные: Нет

– Результат: Предмет попадает в инвентарь

– Вызываемые модули: Нет

– Описание алгоритма: При соприкосновение с игроком предмет попадает в инвентарь и уничтожается на локации.

2.4 Тестовый пример работы компьютерной игры

Наглядный пример работы компьютерной игры “Path in the forest”.

Что бы запустить игру нужно нажать на её ярлык (Рисунок 1).

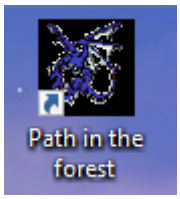

Рисунок 1 – Ярлык игры

На рисунке 2 (Рисунок 2 – Главное меню игры, страница 19) представлено Главное меню игры, где можно начать игру так и выйти из нее, нажав соответствующие кнопки.

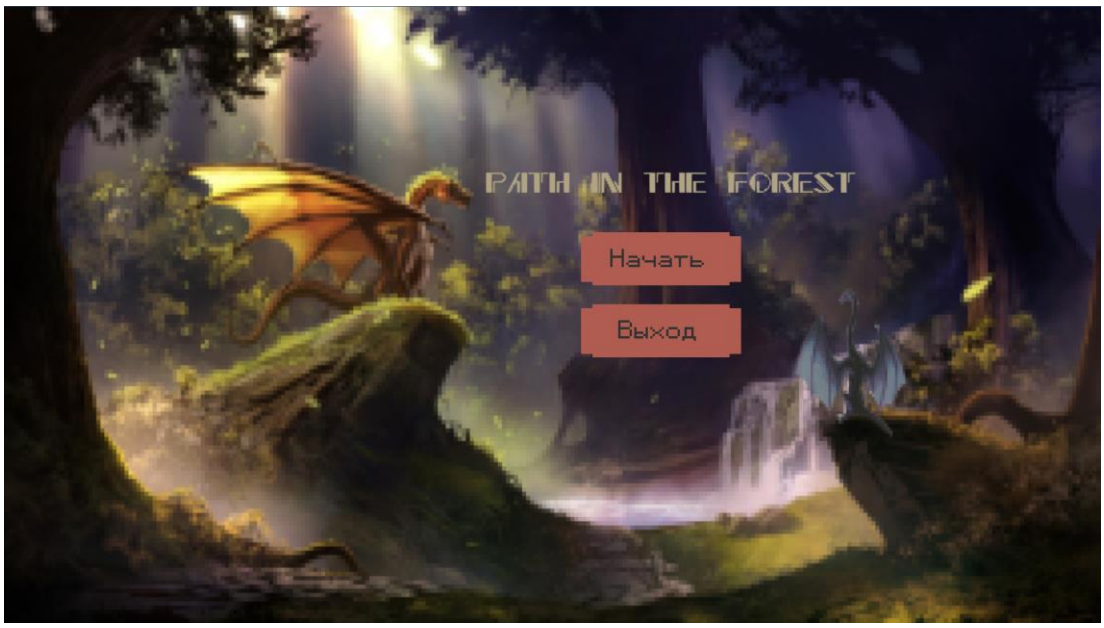

Рисунок 2 – Главное меню игры

Нажав кнопку “Начать”, игрок попадет на локацию, представленную лесистой местностью, на которой расположен главный герой (вернее героиня) - виверна Лорней как показано на рисунке 3

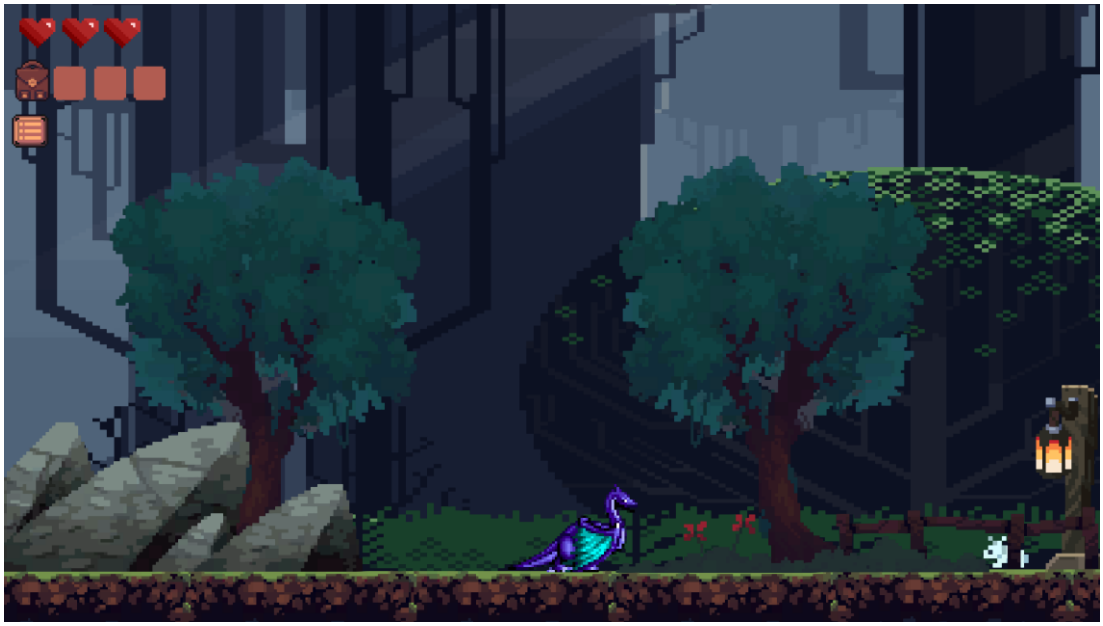

Рисунок 3 - Начало уровня

Нажав клавишу Esc или кнопку “Меню” в правом верхнем углу (Рисунок 4 – Кнопка меню, страница 20) откроется меню паузы. Также над кнопкой расположен инвентарь (Рисунок 5 – Инвентарь, страница 20) и жизни, которые отнимаются при получении урона (Рисунок 6 – Жизни, страница 20). Меню паузы представлено на рисунке 7 (Рисунок 7 – Меню паузы, страница 20).

Рисунок 4 – Кнопка Меню

Рисунок 5 – Инвентарь

Рисунок 6 - Жизни

Рисунок 7 – Меню паузы

Продвигаясь по уровню, игрок встречает первого NPC – золотого дракона Авксентий, с которым нужно начать диалог нажав на соответствующую кнопку (Рисунок 8), далее начнется собственно диалог (Рисунок 9 – Авксентий ведет диалог с Лорней, страница 21).

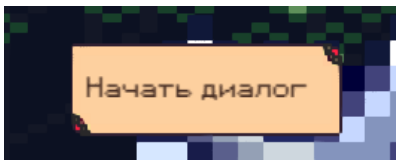

Рисунок 8 – Кнопка “Начать диалог”

Рисунок 9 – Авксентий ведет диалог с Лорней

За драконом расположены не только дружелюбные NPC, но и противники, с которыми игроку следует расправится (Рисунок 10).

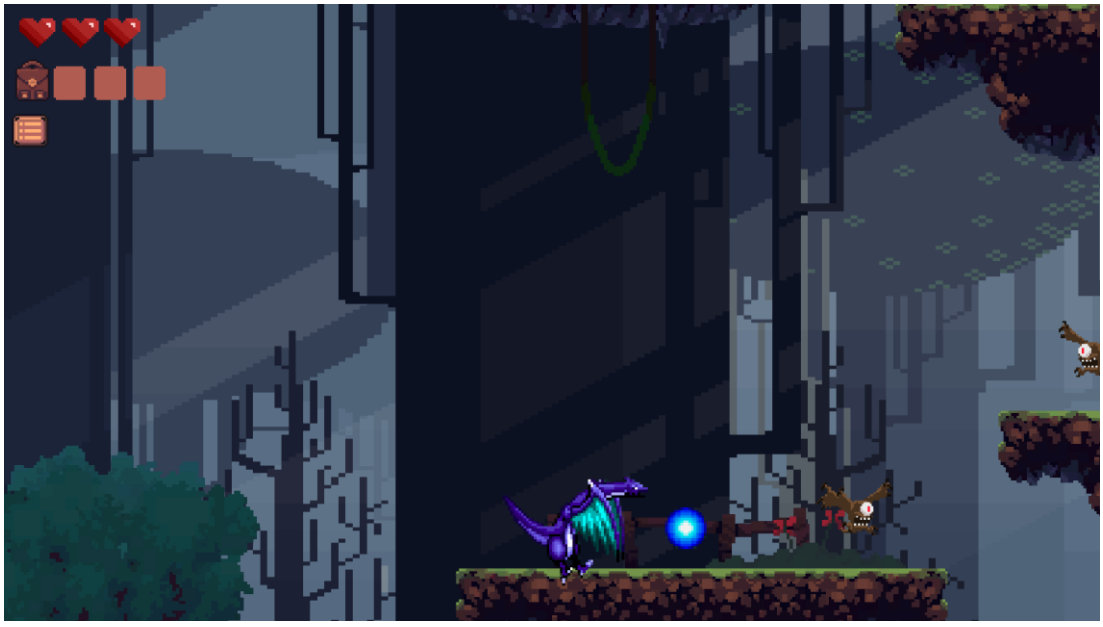

Рисунок 10 - Лорней атакует демонический глаз

По уровню разбросаны три листа с текстом (Рисунок 11 – Нахождение одного листа, страница 22), которые можно собрать: при контакте с игроком они попадают в инвентарь (Рисунок 12 – Лист в инвентаре, страница 22). О их предназначении можно узнать, если найти барда и поговорить с ним (Рисунок 13 Бард рассказывает о своей потери, страница 22).

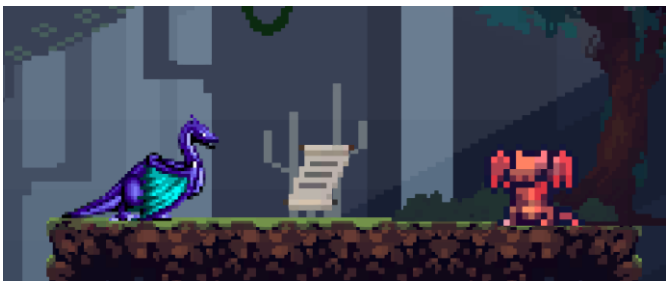

Рисунок 11– Нахождение одного листа

Рисунок 12 – Лист в инвентаре

Рисунок 13 – Бард рассказывает о своей потери

Для того что бы забраться на парящий островок нужно прыгнуть, также прыжком можно миновать монстра как показано на рисунке 14 (Рисунок 14 – Лорней перепрыгивает демонический глаз, страница 23).

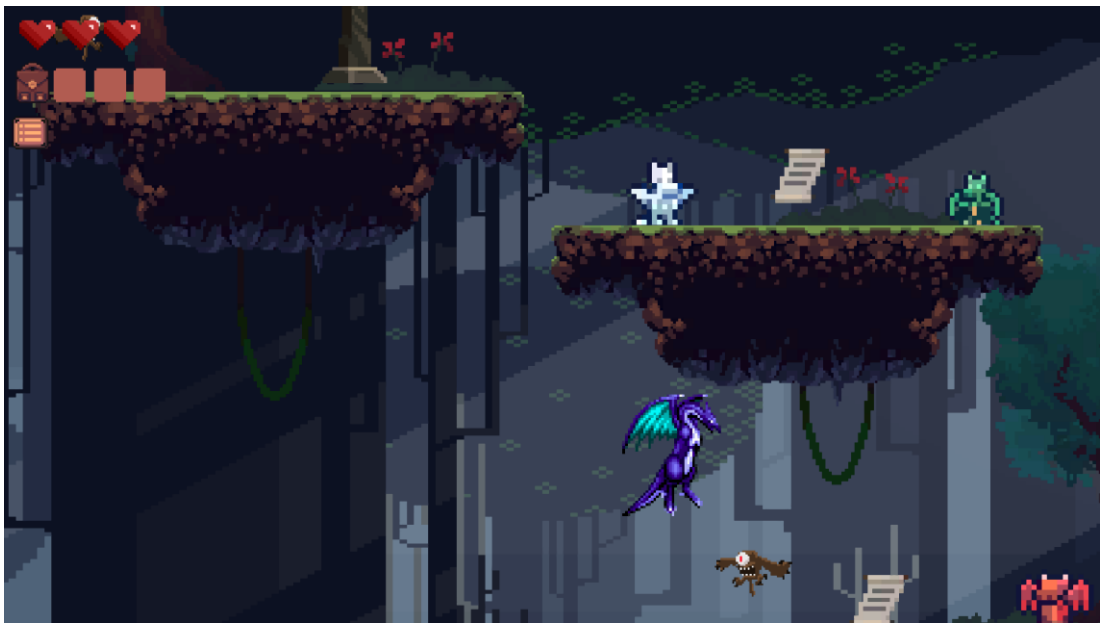

Рисунок 14 – Лорней перепрыгивает демонический глаз

Основной игровой процесс заключатся в исследование уровня (Рисунок 15).

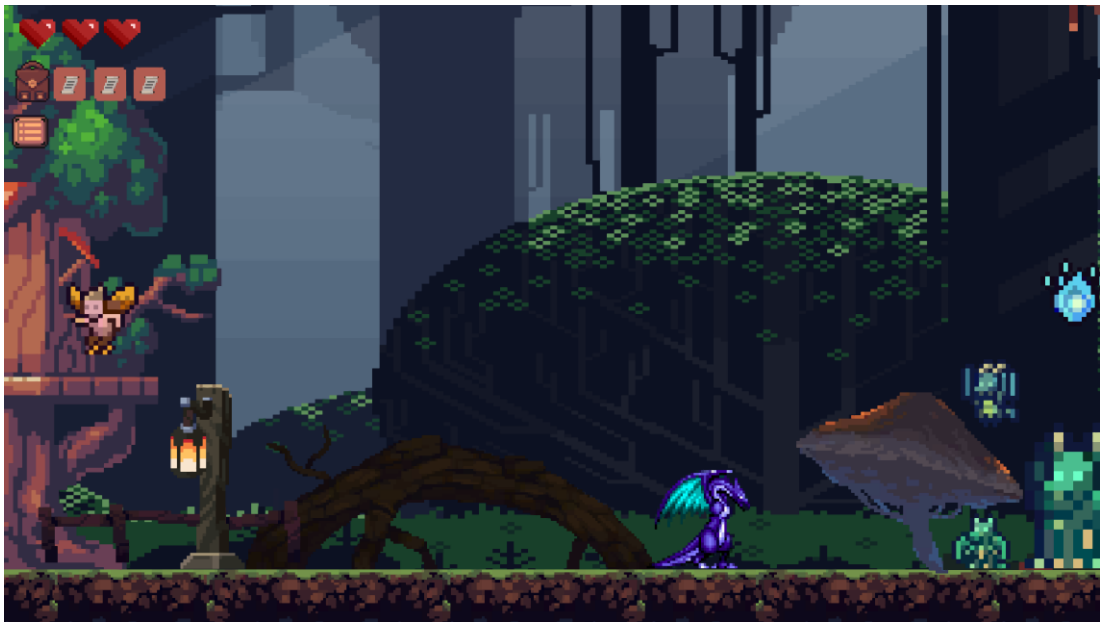

Рисунок 15 – Лорней идет

3. Мероприятия по охране труда

3.1 Требования к организации рабочего места

Стол – это основа игрового места геймера, поверхность, на которой разместятся ваши монитор, колонки, клавиатура и мышь, опционально – корпус ПК, игровая гарнитура и другие аксессуары.

Очевидно, компьютерный стол должен как минимум быть достаточно просторным, исходя из запросов и возможностей.

К выбору компьютерного стола стоит подходить ответственно, не рассчитывать на то, что удастся за любым столом разместиться с комфортом. Не удастся. Рекомендуется тщательно продумать все нюансы размещения мебели в квартире, произвести необходимые замеры, а также подумать над функциями, которые вам нужны от стола, и примериться к возможному обновлению компьютера – на этом часто обжигаются владельцы столов с нишей для монитора, когда решают перейти на дисплей побольше.

На рынке доступно очень много моделей столов, от сугубо «геймерских», с агрессивным дизайном и подсветкой, до классических, строгих и минималистичных моделей. Какой выбрать – дело вкуса и бюджета, но не забывайте обращать внимание на толщину столешницы, устойчивость конструкции и другие параметры. Не лишним окажется наличие дополнительных опций, таких как держатели для чашек и наушников, кабель-каналы для прокладки проводов, регулировка по высоте. Да и подсветка лишней не будет, если пользователь лояльно относитесь к подобным способам украсить свое место.

Кресло – это мебель, в которой проведется немало часов, работая и играя за ПК, поэтому оно должно быть комфортным: не слишком узким, с удобной высотой. Если у него еще и есть дополнительные опции, вроде откидывающейся спинки, настраиваемых подлокотников, подушек под спину и голову – это только прибавит ему бонусов.

На расположение тела на рабочем месте влияет не только мебель, но и используемое оборудование, а точнее, взаимное расположение устройств ввода (мыши, клавиатуры) и вывода (монитора).

Можно выделить две принципиально отличающихся ситуации:

– работа со стационарного компьютера;
– работа с ноутбука.

Стационарный компьютер

Имея все комплектующие по отдельности, можно располагать клавиатуру относительно монитора в соответствии со всеми рекомендациями.

Монитор должен быть:

На уровне глаз. Правда, здесь не все однозначно. Неврологи советуют, чтобы на уровне глаз находился центр монитора, а ортопеды — чтобы там был верхний край монитора. Однако голова точно не должна быть наклонена вниз. Такое неправильное положение дает слишком высокую нагрузку на позвоночник, которая может проявляться дискомфортом в шее или кажущимися головными болями (кажется, что болит голова, а по факту — шея).

Не на свету — чтобы солнце или искусственное освещение не бликовали на экране монитора.

На комфортном расстоянии от глаз (в зависимости от диагонали и прочих параметров).

Два монитора рекомендуют располагать рядом (на одной высоте), устанавливая второй со стороны ведущего глаза. При этом, выбирая диагональ для двух мониторов, стоит учитывать, как сильно придется вертеть головой, чтобы охватить все изображение.

Расположение клавиатуры и мыши относительно монитора зачастую определяет высота стола. Тут надо следить за углами в суставах на руках — в локте должно быть около 90 градусов.

Кстати, при всей банальности такого устройства, как мышь, к ее выбору стоит подойти ответственно. Если в лучезапястном суставе будет излишний изгиб, при полноценной работе вполне можно заработать себе так называемый туннельный синдром. Надо выбирать:

– либо эргономичную мышь, на которой рука не устает;
– либо трекбол;
– либо трекпад.

Правда, однозначных рекомендаций тут дать не получится — многое зависит от личных предпочтений.

Аналогичная ситуация с выбором клавиатуры. Сила нажатия на кнопки не должна быть слишком высокой, чтобы не травмировать суставы в пальцах. Некоторые предпочитают механические клавиатуры, но они работают довольно громко, так что возможность их использования зависит от лояльности домашних.

Ноутбук

По мнению врачей, комфортно использовать ноутбук на длительных промежутках времени можно только вместе с дополнительной клавиатурой и/или монитором. Так, можно либо экран ноутбука разместить на рекомендуемой высоте, используя для ввода текста дополнительную клавиатуру, либо монитор поставить на уровне глаз, задействуя клавиатуру ноутбука. Сочетание этих двух ситуаций — док-станция, к которой можно подключить и внешний монитор, и клавиатуру, фактически с точки зрения эргономики сводит задачу к предыдущей — к стационарному рабочему месту.

Освещение на рабочем месте должно помогать выравнивать контраст изображения на мониторе и окружающего пространства. В поле зрения при работе за монитором не должно быть ярких источников света — солнца или ламп. Если используется настольная лампа, у нее должен быть абажур, защищающий глаза. В темное время суток стоит включать подсветку за

монитором или общий свет, чтобы сам монитор не становился этим ярким источником света.

Хотя напрямую к освещению это и не относится, надо помнить о гимнастике для глаз — делать перерывы каждые 1-1.5 часа. Во время таких перерывов надо перефокусировать взгляд с близкого на дальние расстояния

Необходимо обеспечить нормальное проветривание комнаты, где размещается рабочее место. Те, кто любят автоматику, могут подобрать датчик или установить так называемую альпийскую форточку, которая обеспечит проветривание при закрытых окнах.

ЗАКЛЮЧЕНИЕ

Платформер является одним из старейших игровых жанров. Именно этот жанр и стоял на заре игровой индустрии. Но не смотря на свой возраст (уже больше 40 лет на момент написания дипломного проекта), он не теряет своей актуальности до сих пор.

При написании дипломного проекта была спроектирована и реализована компьютерная игра «Path in the forest». В результате разработки приложения была достигнута поставленная цель, а именно - создана логическая игра в соответствии с техническим заданием. Для этого были разработаны алгоритмы и программы для реализации данной игры с использованием современной технологии программирования. Программное приложение реализовано с использованием движка Unity версии 2021.3.0f1 по лицензии Personal и языка программирования C#.

В ходе выполнения дипломного проекта изучены нормативные документы, регламентирующие состав, содержание и форму технической документации на разрабатываемый программный продукт, среди них ГОСТ 19.701-90 - ЕСПД.

Результатом дипломного проекта является готовый программный продукт: техническая демонстрационная версия компьютерной игры “Path in the forest” в жанре 2D – платформер на Unity. Реализованы подсистемы поиска пути, графического интерфейса пользователя и взаимодействия внутриигровых объектов. Все подсистемы отлажены, оптимизированы, протестированы и интегрированы в единую программную систему.

Таким образом, поставленные цели и задачи дипломного проекта решены, однако данный проект не претендует на глубину теоретического и практического уровней и может быть усовершенствован и продолжен в других аспектах.

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

Книги и учебные пособия

1 Алан Торн. Искусство создания сценариев в Unity ЛитРес 2022

2 Троелсен Э., Джепикс Ф. Язык программирования C# 7 и платформы .NET и .NET Core 8-е издание Диалектика 2018

3 Рихтер Дж. CLR via C# 4-е издание "Издательский дом ""Питер""""2013

4 Джозеф Албахари. C# 9.0. Справочник. Полное описание языка Диалектика 2021

5 Freeman A. - Pro ASP.NET Core 6 Apress 2022

6 Mark J.Price C 10 and NET 6 Modern Cross-Platform Development Sixth Edition Pact 2021

7 Д. В. Лисицин. Объектно-ориентированное программирование. Конспект лекций ЛитРес 2022

8 Алан Торн. Основы анимации в Unity ЛитРес 2022

9 Крис Дикинсон. Оптимизация игр в Unity 5. Советы и методы оптимизации игровых приложений ЛитРес 2022

10 Ферроне Харрисон. Изучаем C# через разработку игр на Unity. 5-е издание "Издательский дом ""Питер""" 2021

Электронные ресурсы

1 Прыжок через пропасть: самые важные платформеры 80-х, изменившие жанр [Электронный ресурс] URL: https://dtf.ru/retro/31060-pryzhok-cherez-propast-samye-vazhnye-platformery-80-h-izmenivshie-zhanr (Дата обращения 20.04.2022.)

2 Unity (игровой движок) [Электронный ресурс] URL: https://ru.wikipedia.org/wiki/Unity_(%D0%B8%D0%B3%D1%80%D0%BE%D0%B2%D0%BE%D0%B9_%D0%B4%D0%B2%D0%B8%D0%B6%D0%BE%D0%BA) (Дата обращения 20.04.2022.)

3 Платформер [Электронный ресурс] URL: https://ru.wikipedia.org/w/index.php?title=%D0%9F%D0%BB%D0%B0%D1%82%D1%84%D0%BE%D1%80%D0%BC%D0%B5%D1%80&stable=1(Дата обращения 20.04.2022.)

4 5 принципов хорошего интерфейса в играх[Электронный ресурс] [сайт :https://skillbox.ru/media/gamedev/5_printsipov_khoroshego_interfeysa_v_igrakh] (Дата обращения 20.04.2022.)

5 Visual Studio Blog [Электронный ресурс] URL : https://devblogs.microsoft.com/visualstudio/(Дата обращения 27.04.2022.)

6 The Top 25 2D Platformers of All Time [Электронный ресурс] URL: https://cultureofgaming.com/25-best-platformers/ (Дата обращения 10.05.2022.)

7 Five Critical Moments in Platform Game History. [Электронный ресурс] URL: https://www.vg247.com/five-critical-moments-in-platform-game-history(Дата обращения 10.05.2022.)

8 Введение в историю геймдизайна. Часть 2: Платформеры . [Электронный ресурс] URL: https://vokigames.com/vvedenie-v-istoriyu-gejmdizajna-chast-2-platformery/ (Дата обращения 10.05.2022.)

9 Этапы разработки компьютерных игр [Электронный ресурс] URL : https://gdjob.pro/stati/soiskatelyam/etapy-razrabotki-kompyuternykh-igr/(Дата обращения 10.05.2022.)

10 Введение в геймдизайн: Основные понятия и принципы проектирования игр [Электронный ресурс] URL: https://vc.ru/flood/10495-gamedev-challenges (Дата обращения 17.05.2022.)

11 Как обустроить себе рабочее место [Электронный ресурс]URL : https://habr.com/ru/companies/maxilect/articles/466333/(Дата обращения 17.05.2022.)

12 Рабочее место геймера: Как обустроить крутой игровой сетап [Электронный ресурс] URL: https://cubiq.ru/rabochee-mesto-geymera/(Дата обращения 17.05.2022.)

13 Интеграция Visual Studio C# [Электронный ресурс] URL: https://docs.unity3d.com/ru/530/Manual/VisualStudioIntegration.html(Дата обращения 23.04.2022.)

ПРИЛОЖЕНИЯ

А Текст программы

Папка – Scripts

Файл - Camera.cs

```
public class Camera : MonoBehaviour
{
    public float FollowSpeed = 2f;
    public float yOffset = 1f;
    public Transform target;

    // Update is called once per frame
    void Update()
    {
        Vector3 newPos = new Vector3(target.position.x, target.position.y + yOffset, -10f);
        transform.position = Vector3.Slerp(transform.position, newPos, FollowSpeed * Time.deltaTime);
    }
    /* [SerializeField] private Transform player;
     private Vector3 pos; // координаты движения

     private void Awake() // в этом методе ищем игрока
     {
        if (!player)
        {
            player = FindObjectOfType<Hero>().transform;
        }
```

```
        }
        private void Update() // в этом методе мы получаем координаты игрока
        {
            pos = player.position;
            pos.z = -10f; // фиксация координаты для того чтобы камера не находилась в одной плосокости с игроком
            transform.position = Vector3.Lerp(transform.position, pos, Time.deltaTime);
        }
       */
    }
```

Файл – Hero.cs

```
    public class Hero : Entity
    {
        [SerializeField] private float speed = 3f; // скорость движения
        [SerializeField] private int health; // кол-во жизней
        [SerializeField] private float jumpForce = 15f; //сила прыжка
        private bool isGrounded = false;

        [SerializeField] private Image[] hearts;
        [SerializeField] private Sprite alivaheart;
        [SerializeField] private Sprite deadheart;

        private Rigidbody2D rb;
        private Animator anim;
        private SpriteRenderer sprite;
        private Vector3 moveVector;

        public static Hero Instance { get; set; } // что бы не создавать экземпляр Hero каждый раз
```

```
        private States State // поля состояний
        {
            get { return (States)anim.GetInteger("state"); } // получаем команды из списка аниматора
            set { anim.SetInteger("state", (int)value); } // контролируем команды из списка
        }
        private void Awake()
        {
            lives = 3;
            health = lives;
            Instance = this;

            rb = GetComponent<Rigidbody2D>();
            anim = GetComponent<Animator>();
            sprite = GetComponentInChildren<SpriteRenderer>();
            Instance = this; // что бы не было ошибки при соприкосновение с игроком
        }
        private void Update() // метод для работы пражка и бега по кнопкам
        {
            if (isGrounded ) State = States.Idle; // для анимации покоя
            if ( Input.GetButton("Horizontal"))
                Run();
            if ( isGrounded && Input.GetButtonDown("Jump")) // благодоря isgrounded прыжок больше не будет бесконечным
                Jump();
            if (health > lives)  // ограничение здоровья
                health = lives;
```

```
            for(int i = 0; i < hearts.Length; i++) // цикл для отаброжения здоровья
        {
                if (i < health)
                    hearts[i].sprite = alivaheart;
                else
                    hearts[i].sprite = deadheart;
                if (i < lives)
                    hearts[i].enabled = true;
                else
                    hearts[i].enabled = false;
        }
         }

         private void FixedUpdate() // что бы небыло проблем с поверхностью :)
         {
             CheckGround();
         }
         private void Run() // метод для бега
         {
             if (isGrounded) State = States.Run; // для анимации бега
             var movement = Input.GetAxis("Horizontal");
             transform.position += new Vector3(movement, 0, 0) * Time.deltaTime * speed;
             if (!Mathf.Approximately(0, movement))
                 transform.rotation = movement < 0 ? Quaternion.Euler(0, 180, 0) : Quaternion.identity;
         }
```

```
        private void Jump() // метод для прыжка
        {
            rb.AddForce(transform.up * jumpForce, ForceMode2D.Impulse);
        }
    private void CheckGround() // метод для проверки поверхности под ногами
        {
            Collider2D[] collider = Physics2D.OverlapCircleAll(transform.position, 0.3f);
            isGrounded = collider.Length > 1;
            if (!isGrounded) State = States.Jump; // для анимации прыжка
        }
    public override void GetDamage()
    {
            health -= 1;
       if (health == 0)
       {
                foreach (var h in hearts)
                    h.sprite = deadheart;
                Die();
       }
    }
}
public enum States // поля состояний
{
    Idle,
    Run,
    Jump
```

Файл – arrao.cs

```
public class arrao : MonoBehaviour
{
    public float speed = 70f; // скорость стрелы
    public Rigidbody2D rb;
    void Start()
    {
        rb.velocity = transform.right * speed; // движение стрелы
        Destroy(gameObject, 0.4f);
    }
    private void OnTriggerEnter2D(Collider2D hitinfo) // метод для уничтоженя врага
    {
        Obstacle enemy = hitinfo.GetComponent<Obstacle>();
        Flight _ = hitinfo.GetComponent<Flight>();
        if (enemy !=null)
        {
            enemy.TakeDamage(1);
        }
        Destroy(gameObject);
    }
}
```

Файл – Entity.cs

```
public class Entity : MonoBehaviour
{
    protected int lives;
    public virtual void GetDamage()
    {
        lives--;
        if (lives <0 )
        {
```

```
            Die();
        }
    }
    public virtual void Die()
    {
        Destroy(this.gameObject);
    }
}
```

Файл – Obctscle.cs

```
public class Obstacle : MonoBehaviour
{
    public int hp = 4; // здоровье
    public GameObject deathEffect;
    public void TakeDamage(int damage) // функция для получения урона
    {
        hp -= damage;
        if (hp <= 0)
        {
            Die();
        }
    }
    void Die() // смерть
    {
        Instantiate(deathEffect, transform.position, Quaternion.identity);
        Destroy(gameObject);
    }

    private void OnCollisionEnter2D(Collision2D collision)
    {
        if (collision.gameObject == Hero.Instance.gameObject)
```

```
        {
            Hero.Instance.GetDamage();
        }
    }
}
```

Файл – Weapon.cs

```
public class Weapon : MonoBehaviour
{
    public Transform arrows;
    public GameObject arrowprefab;
    void Update()
    {
        if (Input.GetButtonDown("Fire1"))
        {
            Shoot();   // при нажатие кнопки будет выполнятся метод стрельба
        }
        void Shoot()  // логика стрельбы
        {
            Instantiate(arrowprefab, arrows.position, arrows.rotation);
        }
    }
}
```

Папка – Menu

Файл – Menu.cs

```
public class Menu : MonoBehaviour
{
    public void PlayGame()
    {

SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex + 1);
```

```
    }
    public void Exit()
    {
        Application.Quit();
        Debug.Log("Игра закрылась");
    }
}
```

Файл – PauseMenu.cs

```
public class PauseMenu : MonoBehaviour
{
    public bool PauseGame;
    public GameObject PauseGameMenu;
    void Update()
    {
        if (Input.GetKeyDown(KeyCode.Escape))
        {
            if (PauseGame)
            {
                Resume();
            }
            else
            {
                Pause();
            }
        }
    }

    public void Resume()
    {
        PauseGameMenu.SetActive(false);
```

```
        Time.timeScale = 1f; // время в нормальном режиме
        PauseGame = false;
    }
    public void Pause()
    {
        PauseGameMenu.SetActive(true);
        Time.timeScale = 0f; // время заморожено
        PauseGame = true;
    }

    public void LoadMenu()
    {
        Time.timeScale = 1f;
        SceneManager.LoadScene("Menu");
    }
}
```

Папка – Dialogs

Файл – DailogAnimator.cs

```
// скрипт для ввызова диалога рядом с персонажем
public class DailogAnimator : MonoBehaviour
{
    public Animator startAnim;
    public DialogManager dm;
    // функция работает, когда игрок вхрдит в зону NPC
    public void OnTriggerEnter2D(Collider2D other)
    {
        startAnim.SetBool("StartOpen", true);
    }

    // функция закончить диалог
```

```
    public void OnTriggerExit2D(Collider2D other)
    {
        startAnim.SetBool("StartOpen", false);
        dm.EndDialog();
    }

}
```

Файл – Dialog.cs

```
[System.Serializable]
public class Dialog
{
    public string name;
    [TextArea(3, 10)] // минимальное и максимальное кол-во строк текста
    public string[] sentences; // массив предложения
}
```

Файл – DailogTrigger.cs

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

public class DialogTrigger : MonoBehaviour
{

    public Dialog dialog;
    // функция вызывающая диалог
    public void DialogueTrigger()
    {
        FindObjectOfType<DialogManager>().StartDialog(dialog);
    }
}
```

Файл – DialogManager.cs

```
public class DialogManager : MonoBehaviour
{
    public Text dialogueText;
    public Text nameText;
    public Animator boxAnim;
    public Animator startAnim;
    private Queue<string> senteces;
    private void Start()
    {
        senteces = new Queue<string>();
    }
    // функция вызова диалога
    public void StartDialog(Dialog dialog)
    {
        boxAnim.SetBool("DialogBox", true);
        startAnim.SetBool("StartOpen", false); // окно начать диалог закрыть
        nameText.text = dialog.name;
        senteces.Clear(); // очитска приложения
        foreach(string sentence in dialog.sentences)
        {
            senteces.Enqueue(sentence); // ставим в очередь предложения
        }
        DisplayNextSentence();
    }
    public void DisplayNextSentence()
    {
        if (senteces.Count == 0)
        {
            EndDialog();
```

```
            return;
        }
        string sentence = senteces.Dequeue(); // строка/предолежние удаляется из очереди
        StopAllCoroutines();
        StartCoroutine(TypeSentence(sentence));
    }
    // интерфейс для последовательного появления букв

    IEnumerator TypeSentence(string sentence)
    {
        dialogueText.text = ""; // текст диалога

        // цикл для каждой буквы в предложении
        foreach(char letter in sentence.ToCharArray())
        {
            dialogueText.text += letter;
            yield return null;
        }
    }
    // функция закрытия диалогового окна
    public void EndDialog()
    {
        boxAnim.SetBool("DialogBox", false);
    }
}
```

Папка – Inventory

Файл – Inventory.cs

```
public class Inventory : MonoBehaviour
{
```

```
    public bool[] isFull;
    public GameObject[] slots;
}
```

Файл – PickUp.cs

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

public class PickUp : MonoBehaviour
{
    private Inventory inventory;
    public GameObject slotButton;
    private void Start()
    {
        inventory = GameObject.FindGameObjectWithTag("Player").GetComponent<Inventory>();
    }
    private void OnTriggerEnter2D(Collider2D other) // Работает когда игрок взаимодействует с предметом
    {
        if (other.CompareTag("Player"))
        {
            for(int i = 0; i < inventory.slots.Length; i++)  // i это слот интвенторя, он пробегается по всем слотам
            {
                if (inventory.isFull[i] == false)  // есть ли предмет в слоте, если нет то идет заполнение
                {
                    inventory.isFull[i] = true;
```

```
                    Instantiate(slotButton, inventory.slots[i].transform); // создание предмета в инвентаре
                    Destroy(gameObject); // уничтожение предмета на локации
                    break;
                }
            }
        }
    }
}
```

Файл – Slot.cs

```
public class Slot : MonoBehaviour
{
    private Inventory inventory;
    public int i;
    private void Start()
    {
        inventory = GameObject.FindGameObjectWithTag("Player").GetComponent<Inventory>();
    }
    private void Update()
    {
        if(transform.childCount <= 0)
        {
            inventory.isFull[i] = false;
        }
    }
```