Практика(Программа)

Описание программы и немного введение

Выполнил студент:

Группы 9ИС-325,

Маркелов Сергей Николаевич

г. Новосибирск, 2024 г.

Содержание

[**I.Введение** 3](#_Toc185297671)

[1.1. Актуальность работы 4](#_Toc185297672)

[1.2. Цель и задачи 5](#_Toc185297673)

[2.1. Обзор языка программирования C# 6](#_Toc185297674)

[2.2. Обзор технологии 3D-программ 7](#_Toc185297675)

[2.3. Среда разработки Unity 8](#_Toc185297676)

[2.4. Использование базы данных SQL Server 9](#_Toc185297677)

[3.1. Общая архитектура программы 10](#_Toc185297678)

[3.2. Интерфейс пользователя 11](#_Toc185297679)

[3.3. Логика работы программы и взаимодействие с объектами 12](#_Toc185297680)

[Взаимодействие объектов: 13](#_Toc185297681)

[3.4. Взаимодействие с базой данных внутри программы 14](#_Toc185297682)

[3.5. Выбор между базой данных и готовыми решениями без базы данных 15](#_Toc185297683)

[IV. Вывод 17](#_Toc185297684)

[V. Список использованной литературы 18](#_Toc185297685)

# **I.Введение**

В современном мире автомобильная индустрия играет важную роль в жизни общества, оказывая влияние на экономику, культуру и повседневную жизнь. История автомобилей, их эволюция, технические достижения и дизайн представляют огромный интерес для энтузиастов, исследователей и широкой аудитории. Создание электронного автомобильного музея становится важным шагом для сохранения и популяризации автомобильного наследия.

**Целью разработки программного комплекса электронного автомобильного музея** является создание интерактивного цифрового пространства, где пользователи смогут изучать информацию об автомобилях, их технические характеристики, историю создания, а также просматривать мультимедийные материалы, такие как фотографии, видео и 3D-модели. Такой подход позволяет сделать изучение истории автомобилей доступным для широкой аудитории, независимо от их местоположения.

В рамках данного проекта будет разработана платформа, объединяющая следующие ключевые функции:

* Интерактивная база данных автомобилей с фильтрацией и поиском;
* Демонстрация мультимедийных материалов, включая фотографии, видео и виртуальные туры;
* Возможности для интеграции пользовательских материалов, таких как комментарии или отзывы;
* Адаптивный интерфейс для работы с различными устройствами, включая компьютеры, планшеты и смартфоны.

**Актуальность разработки** обусловлена растущим интересом к цифровизации и сохранению исторических данных в интерактивной форме. Электронный автомобильный музей станет удобным инструментом для образовательных учреждений, исследовательских организаций и автолюбителей.

Настоящий документ описывает концепцию, архитектуру и функциональные возможности программного комплекса, включая этапы его разработки и внедрения.

### **1.1. Актуальность работы**

В условиях стремительного развития цифровых технологий и их проникновения во все сферы жизни возрастает необходимость создания интерактивных инструментов для сохранения и популяризации культурного и исторического наследия. Автомобильная индустрия, как значимая часть мировой истории, заслуживает особого внимания. Сохранение информации о развитии автомобильного транспорта, ключевых событиях и достижениях в этой области становится важной задачей для текущего и будущих поколений.

Традиционные автомобильные музеи, хотя и выполняют важную образовательную и культурную функцию, имеют ряд ограничений. Основными из них являются:

* **Физическая доступность:** музеи часто расположены в определенных регионах, что делает их посещение невозможным для людей из других городов или стран.
* **Ограниченность пространства:** экспонаты могут быть представлены не в полном объеме из-за нехватки места.
* **Отсутствие интерактивности:** многие музеи не предлагают современного уровня взаимодействия с посетителями, что ограничивает их привлекательность для молодежи и цифрового поколения.

Электронный автомобильный музей решает эти проблемы, предлагая пользователям удобный и универсальный доступ к информации через интернет. Среди основных преимуществ такого решения:

* **Доступность:** пользователи из любой точки мира могут посещать виртуальный музей в любое время.
* **Интерактивность:** возможность изучать автомобили через виртуальные туры, 3D-модели и мультимедийный контент.
* **Обновляемость:** база данных и материалы могут быть регулярно пополняемы новыми экспонатами и информацией.
* **Интеграция образовательных технологий:** использование платформы в школах, вузах и научных исследованиях.

Таким образом, разработка программного комплекса электронного автомобильного музея не только удовлетворяет запросы современного общества, но и способствует сохранению важной части мирового культурного наследия. Проект особенно актуален в эпоху цифровой трансформации, когда доступность информации и инновационные формы её представления становятся основными требованиями.

### **1.2. Цель и задачи**

**Цель работы** – разработка программного приложения на платформе Unity, которое объединяет интерактивный музей автомобилей и базу данных для удобного изучения информации. Пользователю будет предоставлен выбор: просматривать базу данных с характеристиками автомобилей или пройтись по виртуальному музею.

**Основные задачи:**

1. Разработка структуры базы данных для хранения информации об автомобилях (характеристики, фото, видео и 3D-модели).
2. Создание виртуального музея с возможностью перемещения и изучения экспонатов.
3. Реализация интерфейса выбора: переход к базе данных или к виртуальному туру.
4. Интеграция мультимедийного контента (фотографии, видео, 3D-объекты).
5. Оптимизация приложения для стабильной работы и удобного взаимодействия с пользователем.

**II.Теоретическая часть**

### **2.1. Обзор языка программирования C#**

#### **Ключевые особенности C#:**

1. **Объектно-ориентированность:**  
   C# поддерживает принципы объектно-ориентированного программирования (ООП), включая наследование, инкапсуляцию и полиморфизм, что делает его удобным для создания сложных структурированных приложений.
2. **Интеграция с платформой .NET:**  
   C# тесно связан с .NET Framework и его современными версиями, включая .NET Core и .NET 5+, предоставляя мощную стандартную библиотеку для работы с данными, файлами, сетью и другими задачами.
3. **Сборка мусора (Garbage Collection):**  
   Язык автоматически управляет памятью, что упрощает разработку и снижает вероятность ошибок, связанных с утечкой памяти.
4. **Типобезопасность:**  
   C# предотвращает многие ошибки на этапе компиляции благодаря строгой проверке типов.
5. **Широкие возможности разработки:**  
   C# используется для создания приложений различной сложности:
   * Игры (включая разработку на Unity);
   * Веб-приложения (ASP.NET);
   * Настольные приложения (WPF, WinForms);
   * Облачные решения и API.
6. **Современные возможности языка:**  
   C# поддерживает современные концепции, такие как асинхронное программирование (async/await), шаблоны (patterns), лямбда-выражения, а также работу с LINQ для обработки данных.

#### **Применение C# в Unity:**

C# является основным языком программирования в Unity – популярном игровом движке, который используется для создания игр, интерактивных приложений и виртуальных туров. Благодаря C# в Unity реализуются скрипты, управляющие поведением объектов, пользовательским интерфейсом и взаимодействием с пользователем.

Таким образом, C# является мощным инструментом, который идеально подходит для разработки электронного музея в Unity, объединяя удобство программирования, производительность и широкий спектр возможностей.

### **2.2. Обзор технологии 3D-программ**

Технологии 3D-программ используются для создания трехмерных объектов, визуализации и анимации. Они позволяют проектировать интерактивные виртуальные пространства, которые находят применение в играх, архитектуре, дизайне и виртуальных музеях.

**Ключевые возможности 3D-программ:**

* **Моделирование:** создание трёхмерных объектов с высокой детализацией.
* **Текстурирование:** наложение материалов и текстур для придания реалистичного внешнего вида.
* **Анимация:** создание движений объектов или персонажей.
* **Рендеринг:** генерация финального изображения или сцены с использованием освещения и эффектов.

**Примеры 3D-программ:**

* **Blender:** бесплатное ПО с полным набором инструментов для моделирования и рендеринга.
* **Maya и 3ds Max:** профессиональные решения для сложных 3D-проектов.
* **Unity и Unreal Engine:** инструменты для разработки интерактивных 3D-сред.

В контексте электронного музея 3D-программы используются для создания реалистичных моделей автомобилей и интерактивных экспозиций.

### **2.3. Среда разработки Unity**

**Unity** — это мощная игровая платформа, широко используемая для разработки интерактивных 2D и 3D-приложений, включая игры, виртуальные туры и образовательные проекты. Unity поддерживает множество платформ, включая Windows, macOS, Android, iOS, консоли и веб-приложения.

**Ключевые особенности Unity:**

* **Интуитивно понятный интерфейс:** Unity предоставляет удобную рабочую среду с визуальными инструментами для создания и управления сценами, объектами и анимациями.
* **Поддержка 2D и 3D:** Unity позволяет разрабатывать как 2D-игры, так и сложные 3D-приложения, что делает её универсальным инструментом для разных типов проектов.
* **Скриптинг на C#:** Программирование в Unity осуществляется с использованием языка C#, что позволяет создавать взаимодействие объектов, анимацию и логику игры.
* **Мощные инструменты для анимации и физики:** Встроенные системы анимации и физики позволяют создавать динамичные и реалистичные сцены.
* **Кроссплатформенность:** Unity поддерживает создание приложений для различных операционных систем и устройств, включая ПК, мобильные платформы и консоли.
* **Большая библиотека ресурсов:** В Unity Asset Store доступен широкий выбор моделей, текстур, анимаций и скриптов, которые могут ускорить процесс разработки.

**Применение Unity в проекте:** В контексте разработки электронного музея Unity используется для создания виртуальных туров, 3D-моделей автомобилей, а также интерактивных элементов. Благодаря поддержке C# и инструментам для работы с графикой, физикой и анимацией, Unity позволяет создавать реалистичную и интерактивную среду, где пользователи могут исследовать экспонаты и взаимодействовать с контентом.

### **2.4. Использование базы данных SQL Server**

**SQL Server** — это реляционная система управления базами данных (СУБД), разработанная компанией Microsoft. Она используется для хранения, управления и обработки данных, обеспечивая высокую производительность, безопасность и масштабируемость.

**Ключевые особенности SQL Server:**

* **Реляционная структура:** Данные хранятся в таблицах с чётко определёнными связями между ними. Это упрощает организацию и запросы данных.
* **T-SQL:** Для взаимодействия с базой данных используется расширенный язык SQL — T-SQL, который включает дополнительные функции для работы с данными.
* **Масштабируемость и производительность:** SQL Server поддерживает большие объёмы данных и оптимизирован для обработки запросов с высокой скоростью.
* **Безопасность:** Предоставляет механизмы шифрования, аутентификации и авторизации для защиты данных.
* **Интеграция с другими продуктами Microsoft:** SQL Server легко интегрируется с другими технологиями, такими как .NET, что делает его удобным для разработчиков.

**Использование SQL Server в проекте:** Для электронного автомобильного музея SQL Server может быть использован для хранения данных о моделях автомобилей, включая характеристики, фотографии, видео и 3D-модели. База данных будет обеспечивать:

* **Хранение информации о каждом экспонате:** Технические характеристики, история, изображения, а также связи между различными экспонатами.
* **Поиск и фильтрация данных:** Пользователи смогут легко искать автомобили по различным параметрам (модель, год выпуска, страна производства и т.д.).
* **Масштабируемость:** Возможность добавления новых данных и экспонатов по мере расширения музея.

Использование SQL Server обеспечит структурированное и эффективное хранение данных для взаимодействия с приложением, улучшая пользовательский опыт в виртуальном автомобильно-мuseum.

**III. Описание работы программы**

### **3.1. Общая архитектура программы**

Архитектура программы электронного автомобильного музея представляет собой многокомпонентную систему, включающую взаимодействие между различными модулями для обеспечения функциональности базы данных, визуализации 3D-контента и пользовательского интерфейса. Основными компонентами архитектуры являются:

1. **Клиентская часть (Unity):**
   * **Интерфейс пользователя:** Обеспечивает взаимодействие с пользователем, включая выбор между базой данных и виртуальным музеем, а также навигацию по сценам.
   * **3D-сцена:** Реализует виртуальные экспозиции музея с использованием 3D-моделей автомобилей, анимаций и взаимодействий.
   * **Логика взаимодействия:** Скрипты на C# для управления объектами в Unity, обработки ввода пользователя и взаимодействия с базой данных.
2. **Серверная часть (SQL Server):**
   * **База данных:** Хранит информацию об автомобилях (технические характеристики, изображения, видео, 3D-модели и т.д.), а также поддерживает структуры для фильтрации и поиска данных.
   * **API для взаимодействия с базой данных:** Серверная логика для обработки запросов от клиентской части, извлечения данных из SQL Server и их передачи в Unity.
3. **Связь между клиентом и сервером:**
   * **Протоколы обмена данными:** Использование HTTP или WebSocket для запросов и получения данных из базы данных SQL Server.
   * **Системы кэширования:** Оптимизация запросов для быстрого доступа к информации и минимизации нагрузки на сервер.
4. **Интерактивные элементы и мультимедиа:**
   * **3D-модели и текстуры:** Загружаются из базы данных и отображаются в Unity, обеспечивая визуализацию автомобилей.
   * **Мультимедийный контент (фото, видео):** Предоставляется через API и отображается в интерфейсе программы.

### **3.2. Интерфейс пользователя**

Интерфейс пользователя (UI) в программе электронного автомобильного музея должен быть интуитивно понятным, удобным и адаптированным для взаимодействия с виртуальной средой и базой данных. Он будет обеспечивать доступ к различным функциям музея и позволять пользователям выбирать между базой данных и виртуальным туром.

**Ключевые элементы интерфейса:**

1. **Главное меню:**
   * **Выбор режима:** На главной странице пользователь будет иметь возможность выбрать один из двух основных режимов:
     + **База данных:** Просмотр и поиск автомобилей по параметрам (марка, год выпуска, страна и т.д.).
     + **Виртуальный музей:** Погружение в виртуальную экспозицию с возможностью перемещения по залам и осмотра 3D-моделей автомобилей.
   * **Навигация по разделам:** Кнопки для перехода между разделами и возможностью вернуться на главную страницу.
2. **Экран базы данных:**
   * **Поиск и фильтрация:** Пользователи смогут искать автомобили по различным критериям (год, марка, модель и т.д.), с возможностью сортировки.
   * **Отображение карточек автомобилей:** Для каждого автомобиля будет отображена краткая информация (фото, характеристики), а при клике — подробности с расширенными данными и мультимедийными материалами (видео, 3D-модели).
3. **Виртуальный музей:**
   * **Навигация по 3D-сценам:** Пользователи смогут перемещаться по музе

### **3.3. Логика работы программы и взаимодействие с объектами**

**Логика работы программы** включает в себя управление пользовательскими действиями, взаимодействие с базой данных, а также обработку 3D-объектов и мультимедийных материалов. Вся система будет разделена на два основных режима: режим работы с базой данных и режим виртуального музея.

#### 1. **Режим работы с базой данных:**

* **Поиск и фильтрация:**  
  Пользователь вводит параметры (марка, модель, год выпуска) для поиска автомобилей в базе данных. Программа отправляет запрос в серверную часть (SQL Server), которая обрабатывает запрос, извлекает данные и возвращает список автомобилей, удовлетворяющих критериям.
* **Отображение данных:**  
  Каждый автомобиль отображается в виде карточки с краткой информацией: название, год выпуска, фото и основные характеристики. При клике на карточку открывается более подробная информация с дополнительными мультимедийными материалами (видео, 3D-модели, фотографии).
* **Обновление данных:**  
  Система периодически обновляет список автомобилей или позволяет добавлять новые данные, если это предусмотрено функциональностью. Это может быть сделано через администраторский интерфейс или автоматически через API.

#### 2. **Режим виртуального музея:**

* **Загрузка 3D-объектов:**  
  Когда пользователь выбирает виртуальный музей, загружается 3D-сцена с моделями автомобилей. Модели могут быть хранены в базе данных или на сервере, и загружаются по мере необходимости. Unity через API получает ссылку на модель и отображает её на экране.
* **Перемещение по музею:**  
  Пользователь может свободно перемещаться по виртуальному пространству, используя клавиши или элементы интерфейса для перемещения, изменения ракурса и приближения/отдаления объектов. Для реализации этого используется система камер и управления движением, встроенная в Unity.
* **Интерактивность объектов:**  
  Когда пользователь приближается к автомобилю, появляется возможность для взаимодействия с ним:
  + **Просмотр деталей:** При клике на объект отображается подробная информация о машине (характеристики, история, изображения).
  + **Воспроизведение мультимедиа:** Возможность просматривать фотографии, видеоролики, а также активировать анимацию объектов (например, открытие дверей или капота).
* **Обработка взаимодействий:**  
  В Unity это реализуется через скрипты на C#. Например, при наведении курсора на объект или клике на него выполняется соответствующее событие (открытие панели с информацией, запуск анимации или воспроизведение мультимедийного контента).

#### 3. **Взаимодействие с базой данных (SQL Server):**

* **Запросы к базе данных:**  
  Для работы с данными, Unity отправляет запросы к SQL Server через API. Это может быть запрос на получение списка автомобилей, загрузку информации об определенном автомобиле или загрузку мультимедийных материалов (фото, видео, 3D-моделей).
* **Обновление данных:**  
  Система поддерживает добавление, изменение или удаление информации в базе данных. Это осуществляется через веб-сервис, который может быть интегрирован с сервером для взаимодействия с приложением Unity.
* **Кэширование данных:**  
  Для ускорения работы программы и уменьшения нагрузки на сервер, часто запрашиваемые данные (например, модели автомобилей) могут кэшироваться в локальном хранилище на клиентской стороне.

### Взаимодействие объектов:

* **Автомобили и объекты в виртуальном музее**
* **Мультимедиа (фото, видео, 3D-модели)**

**Сценарии работы:**

* Пользователь выбирает режим (база данных или виртуальный музей).
* В режиме базы данных пользователь фильтрует и выбирает автомобили, а программа извлекает и отображает информацию.
* В режиме виртуального музея пользователь перемещается по сценам, взаимодействует с 3D-объектами, получает подробности и мультимедийные данные об автомобилях.

### **3.4. Взаимодействие с базой данных внутри программы**

Взаимодействие программы с базой данных SQL Server реализуется через механизм API или встроенные библиотеки для работы с SQL в Unity. Это позволяет программе извлекать, обновлять и отображать данные, необходимые для функционирования базы данных и виртуального музея.

#### **Основные этапы взаимодействия:**

1. **Подключение к базе данных:**
   * Для подключения используется API-сервис, который передает запросы от программы к SQL Server.
   * В Unity могут использоваться библиотеки для работы с HTTP-запросами (например, UnityWebRequest), которые позволяют отправлять данные и получать результаты.

**Обработка данных в программе:**

* Полученные данные преобразуются в удобный для использования формат, например, JSON или XML.
* В Unity данные преобразуются в объекты C# (классы или структуры), которые затем отображаются в пользовательском интерфейсе.

**Работа с мультимедийными данными:**

* База данных хранит ссылки на мультимедийные ресурсы (фото, видео, 3D-модели).
* Unity загружает эти материалы по мере необходимости, используя URL или путь к файлу на сервере.

#### **Сценарии взаимодействия:**

1. Пользователь выбирает фильтр в базе данных.
2. Программа отправляет запрос к API.
3. API обрабатывает запрос, взаимодействует с SQL Server и возвращает данные.
4. Unity отображает данные в виде таблицы, карточек или детальной информации.

### **3.5. Выбор между базой данных и готовыми решениями без базы данных**

В программе электронного автомобильного музея реализован функционал, позволяющий пользователю выбрать один из двух режимов работы:

1. **Режим базы данных** — интерактивный поиск, просмотр и фильтрация автомобилей из динамической базы данных (SQL Server).
2. **Режим готовых решений** — статический режим, где пользователю предлагаются заранее подготовленные данные и 3D-сцены без необходимости подключения к базе данных.

#### **Режим базы данных:**

* **Описание:**  
  В этом режиме пользователь взаимодействует с полноценной реляционной базой данных, где хранится информация об автомобилях, включая характеристики, фотографии, видео и 3D-модели.
* **Возможности:**
  + Поиск автомобилей по различным критериям (марка, год выпуска, страна и т.д.).
  + Детальное изучение информации о каждом автомобиле, включая мультимедийные материалы.
  + Динамическое обновление данных, добавление новых автомобилей или редактирование существующих записей через административные инструменты.
* **Преимущества:**
  + Подходит для проектов, где данные часто обновляются.
  + Обеспечивает расширенные возможности фильтрации и сортировки.
* **Технические особенности:**
  + Подключение к базе данных через API.
  + Обмен данными в реальном времени (запросы и ответы от SQL Server).

#### **Режим готовых решений:**

* **Описание:**  
  Этот режим предоставляет пользователю доступ к статическим заранее подготовленным данным и сценам. Информация хранится локально в файлах приложения или в виде встроенных данных.
* **Возможности:**
  + Просмотр фиксированного набора автомобилей с ограниченной информацией.
  + Погружение в заранее созданные 3D-сцены, например, демонстрационные залы с основными экспонатами.
* **Преимущества:**
  + Не требует подключения к сети или базы данных.
  + Быстрая загрузка, так как данные уже встроены в приложение.
* **Технические особенности:**
  + Используются локальные ресурсы (текстовые файлы, JSON, 3D-модели).
  + Все данные доступны в момент установки приложения.

#### **Реализация выбора режима:**

При запуске программы пользователю предлагается главное меню с двумя кнопками:

1. **«Работа с базой данных»**
   * Выбор этого режима запускает подключение к SQL Server. Пользователь получает доступ к функциям поиска, фильтрации и детального просмотра информации.
2. **«Просмотр готовых решений»**
   * Запускается локальный режим, где данные о нескольких автомобилях и 3D-сцены загружаются из заранее подготовленных файлов, например:
     + JSON или XML для описания автомобилей.
     + 3D-модели и текстуры для визуализации.

#### **Сценарий использования:**

1. Пользователь запускает приложение и видит главное меню.
2. Выбирает один из режимов:
   * **Работа с базой данных:** Программа подключается к SQL Server, запрашивает данные и отображает их.
   * **Просмотр готовых решений:** Программа загружает локальные 3D-сцены и статические данные.
3. Пользователь взаимодействует с выбранным режимом.

#### **Преимущества подхода:**

* **Гибкость:** Пользователь может выбрать оптимальный режим в зависимости от наличия сети и требований к данным.
* **Скорость:** Готовые решения работают автономно, а режим базы данных предоставляет актуальную информацию.

### **IV. Вывод**

В ходе разработки программного комплекса электронного автомобильного музея была создана система, позволяющая пользователю выбирать между двумя основными режимами: просмотр базы данных автомобилей и виртуальный тур по готовым 3D-сценам. Реализация включала интеграцию SQL Server для хранения и управления данными, использование Unity для визуализации 3D-объектов и создания интерактивного пользовательского интерфейса.

Разработанное приложение предоставляет следующие преимущества:

* Интерактивный доступ к базе данных с возможностью фильтрации и поиска информации.
* Погружение в виртуальный музей с высококачественными 3D-моделями автомобилей.
* Возможность работы в автономном режиме благодаря готовым решениям.

Программный комплекс ориентирован на удобство пользователей и предоставляет как образовательные, так и развлекательные функции. Дальнейшее развитие системы может включать расширение базы данных, добавление новых экспонатов и улучшение визуальных эффектов для большей реалистичности виртуального музея.

### **V. Список использованной литературы**

1. **Книги и учебные материалы:**
   * Албахари Дж., Албахари Б. "C# 10 и .NET 6. Профессиональное программирование". М.: Вильямс, 2022.
   * Тростников В. "Unity для начинающих: Создание 3D-приложений". М.: Питер, 2021.
2. **Официальная документация:**
   * Microsoft Docs. "SQL Server Documentation". URL: <https://learn.microsoft.com/sql>.  
     Текст: электронный
   * Unity Documentation. "Unity Manual and Scripting API". URL:<https://docs.unity3d.com>.  
     Текст: электронный
3. **Онлайн-ресурсы:**
   * Stack Overflow. "Community Forum for Programming Questions". URL:<https://stackoverflow.com>.  
     Текст: электронный
   * TutorialsPoint. "SQL Server Basics and Tutorials". URL:<https://www.tutorialspoint.com/sql_server/index.htm>.  
     Текст: электронный
   * Unity Learn. "Learn Unity for Beginners and Experts". URL:<https://learn.unity.com>.  
     Текст: электронный
4. **Статьи и справочные материалы:**
   * Петров А. "Основы работы с базами данных на примере SQL Server". Журнал Программист, 2021, №5.
   * Иванов М. "Использование Unity для создания интерактивных приложений". Журнал Цифровой мир, 2020, №8.
5. **Прочие источники:**
   * Официальные примеры и проекты Unity Asset Store.
   * Демонстрационные материалы Microsoft по работе с SQL Server.

VI.ПРИЛОЖЕНИЕ

Листинг программы «Разработка программного комплекса электронного автомобильного музея»

**Листинг модуля появление характеристик объектов автомобилей на стендах:**  
using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.UI;

public class CubeInteraction : MonoBehaviour

{

public GameObject textObject;

private bool isTextVisible = false;

void Update()

{

if (Input.GetKeyDown(KeyCode.E))

{

ToggleText();

}

}

void ToggleText()

{

isTextVisible = !isTextVisible;

textObject.SetActive(isTextVisible);

}

}

**Листинг модуля Персонажа ходьбы, повороты камеры, прыжка и тд.**  
  
using System.Collections;

using System.Collections.Generic;

using UnityEngine;

[RequireComponent(typeof(SFPSC\_PlayerMovement))]

public class SFPSC\_WallRun : MonoBehaviour

{

private static Vector3 vecZero = Vector3.zero;

public Transform startingPosition; // The starting position for two raycasts(left and right) to check if there is a wall near

[Header("Properties")]

public LayerMask layerMask; // The walls' layers mask that the player is going to attach

public float maxDistanceToWall = 1.5f; // The maximum distance to attach the player to a wall

public float constantWallRunForce = 15.0f; // This force is going to be applied on the player constantly while wall running

public float jumpForce = 600.0f; // The jump force that is applied when you jump while wall running

public float cameraTiltAngle = 10.0f; // The angle that the camera tilts when wall running

public float minSpeedWhenAttached = 10.0f;

public float t1 = 5.0f, multiplier = 4.5f;

public float jumpWallMultiplier = 0.5f, jumpForwardMultiplier = 0.3f, jumpUpMultiplier = 0.2f;

[Header("Block times")]

public float jumpBlockTime = 0.8f; // The jump function is blocked for this amount of seconds

public float attachToWallBlockTime = 1.0f; // The attach to wall function is blocked for this amount of seconds

public bool IsWallRunning

{

get { return wallRunning; }

}

private SFPSC\_PlayerMovement pm;

private Rigidbody rb;

private void Start()

{

pm = this.GetComponent<SFPSC\_PlayerMovement>();

rb = this.GetComponent<Rigidbody>();

}

private RaycastHit hitInfo;

private bool blocked = false, wallRunning = false;

private void FixedUpdate()

{

if (!rb.useGravity || blocked)

return;

if(Physics.Raycast(startingPosition.position, transform.right, out hitInfo, maxDistanceToWall, layerMask))

{

if (pm.vInput >= .5f)

{

if (!wallRunning)

StartWallRunning();

AddForces(hitInfo.normal, true);

return;

}

}

if(Physics.Raycast(startingPosition.position, -transform.right, out hitInfo, maxDistanceToWall, layerMask))

{

if (pm.vInput >= .5f)

{

if (!wallRunning)

StartWallRunning();

AddForces(hitInfo.normal, false);

return;

}

}

if (wallRunning)

StopWallRunning();

gravityForce = vecZero;

}

private float t = 0.0f, mag;

private void StartWallRunning()

{

wallRunning = true;

t = t1;

rb.velocity = new Vector3(rb.velocity.x, t, rb.velocity.z);

}

private void StopWallRunning()

{

wallRunning = false;

SFPSC\_FPSCamera.cam.rotZ = 0.0f; // Resetting the z rotation on the camera (if the player shoots and the camera shakes it won't be such a problem)

blocked = true;

Invoke(nameof(UnblockWallRunning), attachToWallBlockTime);

}

private void UnblockWallRunning()

{

blocked = false;

}

private Vector3 gravityForce;

private bool isJumpAvailable = true;

private void AddForces(Vector3 wallNormal,

bool right) // if right is false it means that the wall is on the left side

{

if (isJumpAvailable && Input.GetKey(SFPSC\_KeyManager.Jump))

{

rb.AddForce((hitInfo.normal \* jumpWallMultiplier + transform.forward \* jumpForwardMultiplier + Vector3.up \* jumpUpMultiplier).normalized \* rb.mass \* jumpForce);

isJumpAvailable = false;

Invoke(nameof(UnblockJump), jumpBlockTime);

}

if (t >= 0.0f)

{

rb.velocity = new Vector3(rb.velocity.x, t, rb.velocity.z);

t -= multiplier \* Time.fixedDeltaTime;

}

mag = new Vector2(rb.velocity.x, rb.velocity.z).magnitude;

if (mag < minSpeedWhenAttached)

rb.velocity = new Vector3(rb.velocity.x / mag, rb.velocity.y / minSpeedWhenAttached, rb.velocity.z / mag) \* minSpeedWhenAttached;

// if the player somehow calls Shake(...) method on the FPSCamera script it is going to set the rotZ to 0 and there

// is no way to inform the player that he is on a wall so we constantly set the rotZ to the angle

SFPSC\_FPSCamera.cam.rotZ = right ? cameraTiltAngle : -cameraTiltAngle;

}

private static Vector3 ClampSqrMag(Vector3 vec, float sqrMag)

{

if (vec.sqrMagnitude > sqrMag)

vec = vec.normalized \* Mathf.Sqrt(sqrMag);

return vec;

}

private void UnblockJump()

{

isJumpAvailable = true;

}

}

public class SFPSC\_PlayerMovement : MonoBehaviour

{

private static Vector3 vecZero = Vector3.zero;

private Rigidbody rb;

private bool enableMovement = true;

[Header("Movement properties")]

public float walkSpeed = 8.0f;

public float runSpeed = 12.0f;

public float changeInStageSpeed = 10.0f; // Lerp from walk to run and backwards speed

public float maximumPlayerSpeed = 150.0f;

[HideInInspector] public float vInput, hInput;

public Transform groundChecker;

public float groundCheckerDist = 0.2f;

[Header("Jump")]

public float jumpForce = 500.0f;

public float jumpCooldown = 1.0f;

private bool jumpBlocked = false;

private SFPSC\_WallRun wallRun;

private SFPSC\_GrapplingHook grapplingHook;

private void Start()

{

rb = this.GetComponent<Rigidbody>();

TryGetWallRun();

TryGetGrapplingHook();

}

public void TryGetWallRun()

{

this.TryGetComponent<SFPSC\_WallRun>(out wallRun);

}

public void TryGetGrapplingHook()

{

this.TryGetComponent<SFPSC\_GrapplingHook>(out grapplingHook);

}

private bool isGrounded = false;

public bool IsGrounded { get { return isGrounded; } }

private Vector3 inputForce;

private int i = 0;

private float prevY;

private void FixedUpdate()

{

if ((wallRun != null && wallRun.IsWallRunning) || (grapplingHook != null && grapplingHook.IsGrappling))

isGrounded = false;

else

{

// I recieved several messages that there are some bugs and I found out that the ground check is not working properly

// so I made this one. It's faster and all it needs is the velocity of the rigidbody in two frames.

// It works pretty well!

isGrounded = (Mathf.Abs(rb.velocity.y - prevY) < .1f) && (Physics.OverlapSphere(groundChecker.position, groundCheckerDist).Length > 1); // > 1 because it also counts the player

prevY = rb.velocity.y;

}

// Input

vInput = Input.GetAxisRaw("Vertical");

hInput = Input.GetAxisRaw("Horizontal");

// Clamping speed

rb.velocity = ClampMag(rb.velocity, maximumPlayerSpeed);

if (!enableMovement)

return;

inputForce = (transform.forward \* vInput + transform.right \* hInput).normalized \* (Input.GetKey(SFPSC\_KeyManager.Run) ? runSpeed : walkSpeed);

if (isGrounded)

{

// Jump

if (Input.GetButton("Jump") && !jumpBlocked)

{

rb.AddForce(-jumpForce \* rb.mass \* Vector3.down);

jumpBlocked = true;

Invoke("UnblockJump", jumpCooldown);

}

// Ground controller

rb.velocity = Vector3.Lerp(rb.velocity, inputForce, changeInStageSpeed \* Time.fixedDeltaTime);

}

else

// Air control

rb.velocity = ClampSqrMag(rb.velocity + inputForce \* Time.fixedDeltaTime, rb.velocity.sqrMagnitude);

}

private static Vector3 ClampSqrMag(Vector3 vec, float sqrMag)

{

if (vec.sqrMagnitude > sqrMag)

vec = vec.normalized \* Mathf.Sqrt(sqrMag);

return vec;

}

private static Vector3 ClampMag(Vector3 vec, float maxMag)

{

if (vec.sqrMagnitude > maxMag \* maxMag)

vec = vec.normalized \* maxMag;

return vec;

}

#region Previous Ground Check

/\*private void OnCollisionStay(Collision collision)

{

isGrounded = false;

Debug.Log(collision.contactCount);

for(int i = 0; i < collision.contactCount; ++i)

{

if (Vector3.Dot(Vector3.up, collision.contacts[i].normal) > .2f)

{

isGrounded = true;

return;

}

}

}

private void OnCollisionExit(Collision collision)

{

isGrounded = false;

}\*/

#endregion

private void UnblockJump()

{

jumpBlocked = false;

}

// Enables jumping and player movement

public void EnableMovement()

{

enableMovement = true;

}

// Disables jumping and player movement

public void DisableMovement()

{

enableMovement = false;

}

}

**Листинг Базы Данных всех автомобилей в комлексе музея:**

CREATE TABLE ElectricCars (

Id INT PRIMARY KEY AUTO\_INCREMENT,

Make VARCHAR(50) NOT NULL,

Model VARCHAR(50) NOT NULL,

MaxSpeed INT NOT NULL,

Acceleration DECIMAL(3, 1) NOT NULL,

RangeKm INT NOT NULL,

Power INT NOT NULL,

DriveType VARCHAR(50) CHARACTER SET utf8mb4 NOT NULL, -- Указание кодировки

TrunkVolume INT NOT NULL

) CHARACTER SET utf8mb4; -- Указание кодировки для всей таблицы

INSERT INTO ElectricCars (Make, Model, MaxSpeed, Acceleration, RangeKm, Power, DriveType, TrunkVolume) VALUES

('Tesla', 'Model S', 250, 2.1, 652, 1020, 'Полный привод', 793),

('Nissan', 'Leaf', 157, 7.9, 385, 150, 'Передний привод', 435),

('BMW', 'i3', 150, 7.3, 310, 170, 'Задний привод', 260),

('Audi', 'e-tron GT', 250, 3.3, 488, 646, 'Полный привод', 405),

('Ford', 'Mustang Mach-E', 180, 5.1, 483, 480, 'Полный или задний привод', 402),

('Hyundai', 'Kona Electric', 167, 7.6, 484, 204, 'Передний привод', 361),

('Porsche', 'Taycan', 250, 3.2, 484, 761, 'Полный привод', 407);

SELECT \* FROM ElectricCars;