1. **ПРОКОММЕНТИРОВАТЬ КОД ВРАЩЕНИЯ КАМЕРЫ ВОКРУГ УСТАНОВКИ**

**public class Scroll : MonoBehaviour**

**{**

**// Целевая позиция, вокруг которой будет вращаться камера**

**[SerializeField]**

**Transform targetPos;**

**// Чувствительность вращения камеры**

**int sensivity = 3;**

**// Максимальная дистанция от камеры до цели**

**int maxdistance = 20;**

**// Минимальная дистанция от камеры до цели**

**int mindistance = 1;**

**void Update()**

**{**

**// Проверяем, удерживается ли правая кнопка мыши**

**if (Input.GetMouseButton(1))**

**{**

**// Вращаем камеру вокруг целевой позиции по оси Y**

**// Угол вращения определяется движением мыши по оси X, умноженным на чувствительность**

**transform.RotateAround(targetPos.position, Vector3.up, Input.GetAxis("Mouse X") \* sensivity);**

**}**

**}**

**}**

**Класс Scroll**: Позволяет вращать камеру вокруг заданной точки (targetPos) при удерживании правой кнопки мыши. Угол вращения зависит от движения мыши по оси X.  
  
Данный код отвечает за вращение игрового объекта, на котором он висит, вокруг заданной точки.

Точка, вокруг которой будет происходить вращение объявляется в поле targetPos, типа Transform.

Помимо этого, есть приватные поля, отвечающие за некоторую настройку проекта:

Sensitivity – поле, отвечающее за настройку (настройкой это назвать сложно, просто захардкоженая циферка) чувствительность.

maxDistance и minDistance (этот восхитительный нейминг…) конкретно к этому блоку кода не относятся, тем не менее, просто скажу, что это поля отвечающие за минимальную и максимальную дистанцию приближения.

1. **ПРОКОММЕНТИРОВАТЬ КОД ДВИЖЕНИЯ КАМЕРЫ В ГРАНИЦАХ ПОМЕЩЕНИЯ**

**…**

**bool ControlDistance(float distance)**

**{**

**// Проверка, находится ли расстояние до цели в допустимых пределах**

**if (distance > mindistance && distance < maxdistance) return true;**

**return false;**

**}**

**void Update()**

**{**

**float x = Input.GetAxis("Horizontal"); // Получение ввода для горизонтального движения**

**float y = Input.GetAxis("Vertical"); // Получение ввода для вертикального движения**

**// Если есть движение по одной из осей**

**if (x != 0 || y != 0)**

**{**

**// Новая позиция камеры с учетом ввода и чувствительности**

**Vector3 newpos = transform.position + (transform.TransformDirection(new Vector3(x, 0, 0)) + Vector3.up \* y) / sensivity;**

**// Если новая позиция находится в допустимых границах, перемещаем камеру**

**if (ControlDistance(Vector3.Distance(newpos, targetPos.position)))**

**transform.position = newpos;**

**}**

**}**

 **Функция ControlDistance**: Проверяет, находится ли расстояние между камерой и целевой позицией в заданных границах (минимальной и максимальной).

 **Логика в Update**: Управляет движением камеры по горизонтали и вертикали в ответ на ввод пользователя, обновляя позицию только при соблюдении условий расстояния.

Данный участок кода предоставляет метод проверки дистанции на то, чтобы текущий зум был в пределах конкретных рамок (minDistance, maxDistance объявленные и инициализированные выше).

После чего, максимально где и куда вставлен код, который в теории должен быть в Update, но окей. Там мы получаем вертикальный и горизонтальный ввод с клавиатуры, проверяя не равны ли они нулю. В случае же, если хотя бы одна не равна нулю, мы создаем новую позицию с помощью структуры Vector3 где просто к текущей позиции добавляем некоторую сформированный вектор направления передвижения (**transform.TransformDirection(new Vector3(x, 0, 0)) + Vector3.up \* y) / sensivity).**

Дальше просто проверяем, не выходит ли новая позиция за рамки допустимы и

в случае, если не выходит – присваиваем позиции нашего объекта новое положение

1. **ПРОКОММЕНТИРОВАТЬ КОД ПРИБЛИЖЕНИЯ И УДАЛЕНИЯ КАМЕРЫ ОТ УСТАНОВКИ**

**…**

**// Проверка, прокручивается ли колесо мыши**

**if (Input.GetAxis("Mouse ScrollWheel") != 0)**

**{**

**// Вычисляем новую позицию камеры на основе прокрутки колесика**

**Vector3 newpos = transform.position +**

**transform.TransformDirection(Vector3.forward \* Input.GetAxis("Mouse ScrollWheel") \* scrollSpeed);**

**// Если новая позиция в допустимых границах, обновляем позицию камеры**

**if (ControlDistance(Vector3.Distance(newpos, targetPos.position))) transform.position = newpos;**

**}**

**Проверка прокрутки колесика мыши**: Если колесико прокручивается, вычисляется новая позиция камеры, и если она в допустимых границах, позиция обновляется.

Представленный выше код считывает, есть ли прокрутка колеса мыши, после чего формирует вектор новой позиции, проверяет, не выходит ли он за пределы и если не выходит – присваивает новую позицию

1. **КАК СОЗДАТЬ ФАЙЛ ДЛЯ АНИМАЦИИ ОБЪЕКТА В СРЕДЕ UNITY**

 Откройте Unity и выберите объект, для которого хотите создать анимацию.

 Перейдите в окно **Animation** (если его нет, выберите Window > Animation > Animation).

 Нажмите **Create** и сохраните файл анимации в нужной папке.

 Запишите ключевые кадры, изменяя свойства объекта в разные моменты времени.

 После завершения редактирования сохраните анимацию.

**Процесс**: Создается анимация для объекта, задаются ключевые кадры в окне Animation, сохраняется анимационный файл.

1. **КАК СОЗДАТЬ И НАСТРОИТЬ КОНТРОЛЛЕР АНИМАЦИИ ДЛЯ УПРАВЛЕНИЯ АНИМАЦИЕЙ ОБЪЕКТА В СРЕДЕ UNITY** Создайте **Animator Controller** в проекте (щелкните правой кнопкой мыши в папке проекта и выберите Create > Animator Controller).

 Откройте **Animator** (выберите созданный контроллер и откройте окно Animator).

 Перетащите созданную анимацию в окно Animator.

 Настройте переходы между различными состояниями анимации, добавляя условия для перехода.

 Присвойте созданный Animator Controller вашему объекту через компонент **Animator**.

**Контроллер анимации**: Создается Animator Controller, в него добавляются анимации и настраиваются переходы между ними для управления анимациями объекта.

1. **ПРОКОММЕНТИРОВАТЬ СКРИПТ ДЛЯ УПРАВЛЕНИЯ АНИМАЦИЕЙ**

**// Объявляем переменную для хранения компонента Animator**

**Animator anim;**

**void Start()**

**{**

**// Получаем компонент Animator, прикрепленный к объекту**

**anim = GetComponent<Animator>();**

**}**

**void Update()**

**{**

**// Проверяем, нажата ли клавиша Q**

**if (Input.GetKeyDown(KeyCode.Q))**

**{**

**// Устанавливаем параметр "run" в true для начала анимации**

**anim.SetBool("run", true);**

**}**

**// Проверяем, нажата ли клавиша W**

**if (Input.GetKeyDown(KeyCode.W))**

**{**

**// Устанавливаем параметр "run" в false для остановки анимации**

**anim.SetBool("run", false);**

**}**

**}**

**Класс**: Получает компонент Animator и устанавливает параметры анимации в зависимости от нажатия клавиш (Q для запуска анимации и W для её остановки).

Представленный выше код выполняет задачу максимально приметивного контроля над анимациями.  
В методе Start (Вызывается при старте, перед отрисовкой первого кадра) мы получаем ссылку на компонент аниматора с игрового объекта, на котором висит данный скрипт. Далее в Update (КАЖДЫЙ КАДР, НЕ 50 РАЗ В СЕКУНДУ) идет проверка, не нажаты ли кнопки Q или W, в зависимости от которых в аниматоре изменяем состояние параметров типа bool c помощью метода setBool(string, bool), где первый параметр – имя параметра в аниматоре, второй параметр – новое значение.

1. **АЛГОРИТМ ПРОЦЕДУРЫ ПРОГРАММИРОВАНИЯ ИНТЕРАКТИВНОГО UI С ИСПОЛЬЗОВАНИЕМ КНОПОК BUTTON**
2. **Создание канваса**
3. **Создание кнопки на нем**
4. **Создание скрипта с методом, который будет вызываться по нажатию**
5. **Добавление в компонент Button новый слушатель событий**
6. **Тестирование взаимодействия с UI на разных устройствах. Настройте событие, например, изменение текста или активация объекта.**
7. **(Дополнительно) Использование других событий для кнопки (например, нажатие и удержание). При необходимости используйте другие события, такие как OnHover или OnMouseDown.**
8. **ПРОКОММЕНТИРОВАТЬ SCRIPT ВЫВОДА СООБЩЕНИЯ В ТЕКСТОВОЕ ПОЛЕ ПОСЛЕ ЩЕЛЧКА ПО КНОПКЕ НА CANVAS**

**using System.Collections;**

**using UnityEngine;**

**using UnityEngine.UI;**

**public class InText : MonoBehaviour {**

**// Ссылка на текстовое поле, в которое будет выводиться сообщение**

**[SerializeField]**

**Text message;**

**// Метод, вызываемый при щелчке по кнопке**

**public void OnPointer() {**

**// Устанавливаем текст в текстовом поле**

**message.text = "Установить начальные значения";**

**}**

**}**

Данный код просто выводит заданный текст в текстовое поле при вызове метода OnPointer()

**Класс InText**: Устанавливает текст в указанном поле при щелчке кнопки, используя метод OnPointer.

1. **АЛГОРИТМ ПРОЦЕДУРЫ СОЗДАНИЯ «ВСПЛЫВАЮЩЕГО» ТЕКСТОВОГО ОКНА ПРИ НАВЕДЕНИИ КУРСОРА НА КНОПКУ**

**1. Создать UI-объект – окно, которое должно всплывать и расположить его на CANVAS.**

**2. Задать скрипт для всплывающего окна.**

|  |
| --- |
| **using UnityEngine;**  **using System.Collections;**  **public class SettingsPopup : MonoBehaviour {**  **void Start() {           Close();**//  ЗАКРЫТЬ ВСПЛЫВАЮЩЕЕ ОКНО ПРИ ЗАПУСКЕ ПРОГРАММЫ**}       public void Open() {         gameObject.SetActive(true);**//   АКТИВИРОВАТЬ ОБЪЕКТ, ЧТОБЫ ОТКРЫТЬ ОКНО.**}        public void Close() {         gameObject.SetActive(false);**// ДЕАКТИВИРОВАТЬ ОБЪЕКТ, ЧТОБЫ ЗАКРЫТЬ ОКНО.**}**  **}** |

**3. Создать на CANVAS две кнопки, соответственно для открытия и закрытия окна при щелчке по кнопкам мышью.**

**4. Назначить кнопкам общий объект обработки – всплывающее окно и соответственно выбрать в связанном с окном скрипте SettingsPopup в списке доступных в скрипте функций функцию Open() для кнопки открытия окна и Close() для кнопки закрытия окна.**

1. **ПРОКОММЕНТИРОВАТЬ SCRIPT ОТКРЫТИЯ-ЗАКРЫТИЯ «ВСПЛЫВАЮЩЕГО» ОКНА ПРИ НАВЕДЕНИИ-УХОДА КУРСОРА С КНОПКИ НА CANVAS**

**using UnityEngine;**

**using System.Collections;**

**public class SettingsPopup : MonoBehaviour {**

**void Start() {**

**// Закрываем всплывающее окно при старте**

**Close();**

**}**

**// Метод для открытия настроек**

**public void OnOpenSettings() {**

**Open();**

**}**

**// Метод для открытия всплывающего окна**

**public void Open() {**

**gameObject.SetActive(true);** //   АКТИВИРОВАТЬ ОБЪЕКТ, ЧТОБЫ ОТКРЫТЬ ОКНО.

**}**

**// Метод для закрытия всплывающего окна**

**public void Close() {**

**gameObject.SetActive(false);** // ДЕАКТИВИРОВАТЬ ОБЪЕКТ, ЧТОБЫ ЗАКРЫТЬ ОКНО

**}**

**}**

**Класс SettingsPopup**: Управляет видимостью всплывающего окна, открывая его при вызове метода OnOpenSettings и закрывая с помощью метода Close.  
  
Данный компонент содержит три метода, два из которых просто меняют текущее состояние игрового объекта, на котором висит этот компонент – Open, Close.

Метод OnOpenSettings вызывает метод Open.

1. **ПРОКОММЕНТИРОВАТЬ SCRIPT ПОДСВЕТКИ ЭЛЕМЕНТА УСТАНОВКИ ПРИ НАВЕДЕНИИ КУРСОРА МЫШИ НА КНОПКУ С НАЗВАНИЕМ ЭЛЕМЕНТА В СПИСКЕ**

**using System.Collections.Generic;**

**using UnityEngine;**

**public class ChangeColor : MonoBehaviour**

**{**

**// Метод для изменения цвета на красный**

**public void ChangCol()**

**{**

**// Получаем компонент Renderer и меняем цвет материала на красный**

**GetComponent<Renderer>().material.color = new Color(1, 0, 0);**

**}**

**// Метод для изменения цвета обратно на белый**

**public void ChangCol1()**

**{**

**// Получаем компонент Renderer и меняем цвет материала обратно на белый**

**GetComponent<Renderer>().material.color = new Color(1, 1, 1);**

**}**

**}**

**Класс ChangeColor**: Изменяет цвет объекта на красный при наведении и возвращает его к белому при уходе курсора.  
  
Компонент ChangeColor (глаголом называть класс конечно да😊) который управляет цветом материала у игрового объекта. Где метод ChangeCol – отвечает за дефолтный цвет, а ChangeCol1 – за подсветку объекта.

1. **АЛГОРИТМ ПРОЦЕДУРЫ ВЫБОРА УДОБНОГО РАКУРСА ДЛЯ КАМЕРЫ НА СЦЕНЕ**
2. Определение возможных ракурсов для камеры.
3. Создание интерфейса для выбора ракурса (например, кнопки).
4. Реализация логики переключения между ракурсами.
5. Тестирование каждого ракурса на удобство и качество.
6. **ПРОКОММЕНТИРОВАТЬ SCRIPT ПЕРЕДВИЖЕНИЯ КАМЕРЫ В ЗАДАННУЮ ТОЧКУ**

**public class Replacer : MonoBehaviour**

**{**

**// Флаг для контроля движения**

**bool move = true;**

**// Начальная и целевая позиции для перемещения**

**Vector3 startPosition;**

**Vector3 needPosition;**

**float speed = 0.01f; // Скорость перемещения**

**float offset = 0; // Смещение для интерполяции**

**Quaternion startRotation; // Начальная ориентация**

**Quaternion needRotaton; // Целевая ориентация**

**// Метод для инициализации перемещения**

**public void Move1()**

**{**

**move = true; // Включаем движение**

**startPosition = transform.position; // Запоминаем начальную позицию**

**startRotation = transform.rotation; // Запоминаем начальную ориентацию**

**needPosition = new Vector3(277.0f, 251, 6); // Задаем целевую позицию**

**needRotaton = Quaternion.AngleAxis(-180, new Vector3(0, 1, 0)); // Задаем целевую ориентацию**

**}**

**void Update()**

**{**

**// Если движение включено**

**if (move)**

**{**

**offset += speed; // Увеличиваем смещение**

**// Интерполируем позицию и ориентацию между начальной и целевой**

**transform.position = Vector3.Lerp(startPosition, needPosition, offset);**

**transform.rotation = Quaternion.Slerp(startRotation, needRotaton, offset);**

**// Проверяем, достигнуто ли целевое положение**

**if (offset >= 1)**

**{**

**move = false; // Останавливаем движение**

**offset = 0; // Сбрасываем смещение**

**}**

**}**

**}**

**}**

**Класс Replacer**: Инициализирует перемещение камеры к заданной позиции и ориентации, используя интерполяцию для плавного перемещения.

Этот класс необходим для возврата к исходной позиции и вращению (да, начальная позиция захардкожена в методе Move1() ☹)

Пройдемся по полям:

Move – поле, отвечает за то, в движении ли сейчас объект.

startPosition и startRotation – начальная позиция и вращение игрового объекта (начальная в данном контексте – позиция, которая была на момент вызова метода Move1())

speed – некоторая скорость, с которой двигается объект.

Offset – прогресс передвижения от начальной точки к конечной.

needPosition и needRotation – позиция и вращение, в которые необходимо переместить текущий объект.