**КОНТРОЛЬНЫЕ ВОПРОСЫ**

**ПО ТЕМЕ «РАЗРАБОТКА СИМУЛЯТОРА УСТАНОВКИ»**

***(все строки программного кода должны быть прокомментированы)***

1. **Программа для обхода камерой вокруг установки**

using UnityEngine; // Подключение пространства имен Unity для доступа к его функциональности

public class Scroll : MonoBehaviour // Объявление класса Scroll, наследующего от MonoBehaviour

{

[SerializeField]

Transform targetPos; // Цель, вокруг которой будет вращаться камера (устанавливается в инспекторе)

[SerializeField]

int sensitivity = 3; // Чувствительность вращения камеры (настраивается в инспекторе)

void Update() // Метод, вызываемый каждый кадр

{

// Проверяем, удерживается ли правая кнопка мыши

if (Input.GetMouseButton(1))

{

// Получаем значение оси X мыши и умножаем на чувствительность

float y = Input.GetAxis("Mouse X") \* sensitivity;

// Если значение оси X не равно 0, вращаем камеру вокруг цели

if (y != 0)

transform.RotateAround(targetPos.position, Vector3.up, y); // Вращение вокруг позиции цели по оси Y

}

[SerializeField]

float scrollSpeed = 10f; // Скорость прокрутки камеры (настраивается в инспекторе)

[SerializeField]

int maxdistance = 20; // Максимальное расстояние от цели (настраивается в инспекторе)

[SerializeField]

int mindistance = 1; // Минимальное расстояние от цели (настраивается в инспекторе)

// Метод для проверки расстояния от камеры до цели

bool ControlDistance(float distance)

{

// Проверяем, находится ли расстояние между минимальным и максимальным значением

if (distance > mindistance && distance < maxdistance)

return true; // Возвращаем true, если расстояние в пределах

return false; // В противном случае возвращаем false

}

// Получаем ввод с клавиатуры для горизонтального движения (клавиши A, D)

float x = Input.GetAxis("Horizontal") / sensitivity;

// Если значение оси X не равно 0, перемещаем камеру

if (x != 0)

{

// Рассчитываем новую позицию камеры, перемещая её влево или вправо

Vector3 newpos = transform.position + transform.TransformDirection(new Vector3(x, 0, 0));

// Проверяем, находится ли новая позиция в допустимых пределах

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

transform.position = newpos; // Обновляем позицию камеры

}

// Получаем значение прокрутки мыши и умножаем на скорость прокрутки

float z = Input.GetAxis("Mouse ScrollWheel") \* scrollSpeed;

// Если прокрутка не равна 0, изменяем позицию камеры

if (z != 0) // Исправлено: была ошибка в синтаксисе, заменено на "if (z != 0)"

{

// Рассчитываем новую позицию камеры, перемещая её вперед или назад

Vector3 newpos = transform.position + transform.TransformDirection(Vector3.forward \* z);

// Проверяем, находится ли новая позиция в допустимых пределах

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

transform.position = newpos; // Обновляем позицию камеры

}

}

}

1. **Программа сдвига камеры с ограничениями вдоль и вглубь помещения с установкой**

void Update() {

float moveSpeed = 5f; // Определяем скорость перемещения

// Получаем новое значение по оси X с учетом ограничений minX и maxX

float x = Mathf.Clamp(transform.position.x + Input.GetAxis("Horizontal") \* moveSpeed \* Time.deltaTime, minX, maxX);

// Получаем значение по оси Y, ограничивая его между minY и maxY

float y = Mathf.Clamp(transform.position.y, minY, maxY);

// Получаем новое значение по оси Z с учетом ограничений minZ и maxZ

float z = Mathf.Clamp(transform.position.z + Input.GetAxis("Vertical") \* moveSpeed \* Time.deltaTime, minZ, maxZ);

// Обновляем позицию объекта, создавая новый вектор с ограниченными значениями x, y и z

transform.position = new Vector3(x, y, z);

}

1. **Программа приближения и удаления камеры относительно установки**

void Update() {

// Получаем значение прокрутки колесика мыши

float scroll = Input.GetAxis("Mouse ScrollWheel");

// Изменяем поле зрения (field of view) камеры на основе значения прокрутки

// Уменьшаем значение поля зрения для приближения, увеличиваем для удаления

Camera.main.fieldOfView -= scroll \* zoomSpeed; // zoomSpeed - скорость зума

}

1. **Правила создания и размещения на экране элементов интерфейса CANVAS**

При создании интерфейса в Unity используйте систему UI.

У всех объектов UI, размещаемых на холсте, существует **точка привязки**, отображаемая в редакторе в виде наклонного крестика с обводкой. Это инструмент для позиционирования элементов интерфейса.

Привязкой (Anchor) объекта называется точка его присоединения к холсту или экрану, относительно этой точки указывается положение объекта.

1. **Принцип обработки щелчка мышью по кнопке**

В Unity для кнопок используйте компонент Button и создайте обработчик события:

csharp

Copy

public void OnButtonClick() {

// Действия при нажатии

}

1. **Принцип обработки надвижения и ухода курсора мыши с кнопки**

Используйте события OnPointerEnter и OnPointerExit из интерфейса IPointerEnterHandler и IPointerExitHandler:

public void OnPointerEnter(PointerEventData eventData) {

// Действия при наведении

}

public void OnPointerExit(PointerEventData eventData) {

// Действия при уходе

}

1. **Программирование выбора оптимального ракурса размещения камеры при щелчке мышью по кнопке**

public void ChangeCameraPosition(Vector3 newPosition) {

// Устанавливаем новую позицию для главной камеры

// Получаем компонент Transform главной камеры и изменяем его позицию на newPosition

Camera.main.transform.position = newPosition;

}

1. **Программирование подсветки элемента установки при наведении курсора мыши на кнопку**

public void OnPointerEnter(PointerEventData eventData) {

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

}

public void OnPointerEnter(PointerEventData eventData) {

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

}

1. **Программирование появления на экране текстового окна при щелчке по кнопке**

public GameObject textWindow; // Объявление переменной для текстового окна, которое будет отображаться

// Метод, вызываемый при нажатии кнопки

public void OnButtonClick() {

// Активируем текстовое окно, делая его видимым на экране

textWindow.SetActive(true); // Показать текстовое окно

}

1. **Настройка проекта для обработки щелчков мышью по 3D-объектам сцены**

void Update() {

// Проверяем, была ли нажата левая кнопка мыши

if (Input.GetMouseButtonDown(0)) {

// Создаем луч (Ray) из камеры в направлении текущей позиции курсора мыши

Ray ray = Camera.main.ScreenPointToRay(Input.mousePosition);

// Выполняем проверку на пересечение луча с объектами в сцене

// Если луч пересекает объект, информация о столкновении будет сохранена в hit

if (Physics.Raycast(ray, out RaycastHit hit)) {

// Здесь можно выполнять действия с объектом, на который нажали

// Например, можно получить информацию о объекте hit.transform

}

}

}

1. **Правила создания таблицы результатов эксперимента на симуляторе установки**

Вся таблица – заголовки, названия полей и т.п. строится из текстовых объектов UI соответствующего размера с фиксированным текстом, а для ячеек, куда нужно записывать результаты эксперимента, используется символ подчеркивания или минуса, чтобы было проще находить нужные ячейки таблицы при записи в них значений по нажатию кнопки «Записать».

Для записи в текстовые объекты (ячейки таблицы) значений используется, например, для текстового объекта с именем **name1** конструкция

**name1.text.ToString()**,

а для вычислений, использующих записи в текстовых объектах, например, с именем **name1** используется конструкция

**int.Parse(name1.text).**

Ниже приведен пример кода заполнения и очистки таблицы результатов работы

**using System.Collections;  
using System.Collections.Generic;  
using UnityEngine;  
using UnityEngine.UI;  
  
public class TableVal : MonoBehaviour  
{  
//определение используемых переменных**

**[SerializeField]  
    InputField textInput;  
  
    [SerializeField]  
    Text u1;  
    [SerializeField]  
    Text i1;  
    [SerializeField]  
    Text u2;  
    [SerializeField]  
    Text i2;  
    [SerializeField]  
    Text u3;  
    [SerializeField]  
    Text i3;  
    [SerializeField]  
    Text v1;  
    [SerializeField]  
    Text v2;  
    [SerializeField]  
    Text v3;   
    [SerializeField]  
    Text ve;**

**public void WriteValue() //функция для кнопки «Записать»  
    {  
        if (u1.text == "-")  
            u1.text = textInput.text.ToString ();  
        else if (i1.text == "-") {  
            i1.text = textInput.text.ToString ();  
            v1.text = (int.Parse (u1.text) \* int.Parse (i1.text)).ToString ();  
        } else if (u2.text == "-")  
            u2.text = textInput.text.ToString ();  
        else if (i2.text == "-") {  
            i2.text = textInput.text.ToString ();  
            v2.text = (int.Parse (u2.text) \* int.Parse (i2.text)).ToString ();  
        } else if (u3.text == "-")  
            u3.text = textInput.text.ToString ();  
         else if (i3.text == "-") {  
            i3.text = textInput.text.ToString ();  
            v3.text = (int.Parse (u3.text) \* int.Parse (i3.text)).ToString ();  
            ve.text = ((int.Parse (v1.text) + int.Parse (v2.text)+ int.Parse (v3.text))/3).ToString ();  
        }  
        textInput.text="";  
    }  
//функция для кнопки «Очистить»  
    public void Clean()  
    {  
        u1.text = "-";  
        u2.text = "-";  
        u3.text = "-";  
        i1.text = "-";  
        i2.text = "-";  
        i3.text = "-";  
        v1.text = "-";  
        v2.text = "-";  
        v3.text = "-";  
    }  
  
}**

1. **Программирование обработки и расчетов данных, занесенных в таблицу наблюдений**
2. **Как создать файл анимации в Unity**

Создайте анимацию, выбрав объект в иерархии, затем перейдите в **Window > Animation > Animation** и создайте новый файл анимации.

1. **Как добавить к объекту контроллер анимации**

Перейдите в **Inspector** выбранного объекта и добавьте компонент Animator. Укажите созданный файл анимации в качестве контроллера.

1. **Как создать и настроить переход к анимации**
2. **Как задать скорость анимации**

В **Animator** выберите анимацию и измените параметр **Speed** в инспекторе.

1. **Как настроить файл анимации**

В окне **Animation** вы можете редактировать ключевые кадры, добавлять новые и изменять параметры анимации.

1. **Как создать и использовать переменные для управления анимацией**

public Animator animator; // Объявление переменной для компонента Animator, который будет управлять анимациями

void Update() {

// Проверяем, была ли нажата клавиша пробела

if (Input.GetKeyDown(KeyCode.Space)) {

// Устанавливаем триггер "Jump" в Animator, чтобы запустить анимацию прыжка

animator.SetTrigger("Jump");

}

}

1. **Скрипт для переключения между состояниями анимируемого объекта**

public void SwitchState() {

// Получаем текущее значение параметра "isRunning" в Animator

// Устанавливаем его в противоположное значение (true -> false или false -> true)

animator.SetBool("isRunning", !animator.GetBool("isRunning"));

}

1. **Как проиграть анимацию из любого состояния**
2. создать переход от **Any State** к этой анимации, например с именем **Hit**. Для возвращения в состояние ожидания необходимо создать переход от новой анимации **Hit** к анимации по умолчанию.
3. создать еще одну переменную типа **Trigger** (например **hitten**), которая будет отвечать за щелчок мышью для запуска анимации. Добавим к переходу от **Any State** эту переменную, а для обратного перехода оставим галочку **Has Exit Time**, чтобы переход совершался автоматически (без использования переменной) один раз после проигрывания анимации.
4. добавить в метод **Update()** код проигрывания новой пользовательской анимации по клику мыши:

**if (Input.GetMouseButtonDown(0))  {  anim.SetTrigger("hitten"); }**