**ПРОГРАММИРОВАНИЕ**

**ОБРАБОТКИ СОБЫТИЙ**

**ДЛЯ СИМУЛЯТОРА**

**СИСТЕМА ЗАДАНИЙ**

Первое задание при начале работы с симулятором это, как правило, задание «Включить установку». После щелчка по кнопке «РАБОТА УСТАНОВКИ» выполняется событие вывода в текстовое окно сообщения: «Включите установку». Первое задание выполняется при совершении события включения установки, когда пользователь щелкнет мышью на объект «Переключатель», что и приведет к выполнению первого задания.

После выполнения первого задания программа должна выдать в информационное окно следующее задание (см. рисунок), которое должно выполняться в соответствии с появляющимся текстовым сообщением и т. д. вплоть до окончания работы с симулятором.



Результатом работы с симулятором установки являются полученные по ходу выполнения заданий измеренные значения, показания приборов и т. п., которые последовательно заносятся в таблицу с выполнением в ней предусмотренных в каждом конкретном эксперименте вычислений.

**ОБРАБОТКА ЩЕЛЧКОВ МЫШЬЮ ПО 3D-ОБЪЕКТАМ СЦЕНЫ**

1. Для обработки события «щелчок мышью по объекту на сцене» необходимо добавить в иерархию объектов новый не отображаемый на сцене объект **Create/UI/EventSystem**
2. В сценарий для объекта необходимо добавить класс **using UnityEngine.EventSystems**
3. В базовый класс добавить новый интерфейс системы событий **IPointerClickHandler**

**using UnityEngine;**

**using System.Collections;**

**using UnityEngine.EventSystems;**

**public class Script11 : MonoBehaviour, IPointerClickHandler {**

**void Start () {**

**}**

**void Update () {**

**}**

**}**

1. К камере необходимо предварительно добавить компоненту **Physics RayCaster** для согласования щелчков мыши по 2D-экрану со щелчками по 3D-объектам на сцене.

Например, для изменения цвета объекта можно использовать следующий код:

**public void OnPointerClick(PointerEventData eventData) {**

**float red = Random.Range(.0f, 1.0f);**

**float green = Random.Range(.0f, 1.0f);**

**float blue = Random.Range(.0f, 1.0f);**

**Color col1 = new Color(red, green, blue);**

**gameObject.GetComponent<Renderer>().material.color = col1;**

**}**

**СОЗДАНИЕ БЛОК-СХЕМЫ РАБОТЫ УСТАНОВКИ**

Для программирования действий любой установки, прежде всего, необходимо продумать и составить **СХЕМУ ВЗАИМОДЕЙСТВИЯ** **ОБЪЕКТОВ**, отражающую взаимодействие объектов внутри этой установки, при составлении которой необходимо пройти следующие этапы:

* ***Выделить события***, которые могут происходить (включение/выключение установки, поворот или перемещение какого-нибудь объекта, нажатие кнопки и т. д.).
* ***Выделить объекты***, которые должны инициировать эти события (например, нажатие по переключателю инициирует событие по включению/выключению установки).
* ***Выбрать элементы***, которые должны реагировать на то или иное событие (например, на включение/выключение установки может реагировать большое количество элементов, начиная от отображения показаний дисплеев до запуска механизмов).
* ***Указать*,** при каких должны происходить события и какие методы при этом будут вызываться.

Например, в работе симулятора лабораторной установки «НАБЛЮДЕНИЕ КОЛЕЦ НЬЮТОНА» используются следующие события:

* включение установки,
* выбор цвета света вращением барабана,
* переход к окуляру,
* заполнение таблицы результатами наблюдения,
* обработка в таблице результататов наблюдения,
* возврат к выбору нового цвета и повторение наблюдения,
* просмотр таблицы,
* очистка таблицы,
* получение итоговых результатов работы установки.

**ПРОГРАММИРОВАНИЕ РАЗЛИЧНЫХ СОБЫТИЙ КНОПКИ**

По умолчанию кнопка реагирует только на событие **OnClick**, в то время как UI-элементы на CANVAS могут отвечать на разные варианты взаимодействий. Для программирования взаимодействий, отличных от заданных по умолчанию, используется компонент **EventTrigger** - добавим **Add Component** к кнопке новый компонент **Event,** а в меню этого компонента выберем вариант **EventTrigger**.

Событие **OnClick** кнопки отвечает только на полноценный щелчок (кнопка мыши нажимается, а затем отпускается), а мы попробуем запрограммировать реакцию на надвижение курсора мыши. Щелкнув на кнопке **Add New Event Type** раскроем список из 17 различных событий для выбора и добавления их к компоненту **EventTrigger.**

Выберем вариант **PointerEnter**. Появится пустое поле для этого события, полностью аналогичное полю для события OnClick. Щелкните на кнопке со значком + (плюс), чтобы добавить элемент, и перетащите на этот элемент объект-контроллер (объект со скриптом). Последовательность действий будет той же, что и для события OnClick, просто на этот раз мы укажем реакцию на другое событие.

Естественно, для выполнения соответствующих действий при наведении курсора мыши в сценарий обрабатываемого объекта необходимо добавить еще один метод **OnPointerEnter()**:

**[SerializeField]   
    private Text message;  // объявить переменную и связать ее с объектом на сцене**

**public void OnPointerEnter() {**

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

**}**

В заключение, в списке функций, доступных из скрипта объекта-контроллера, необходимо выбрать в меню вариант **OnPointerEnter().**

**АЛГОРИТМ СОЗДАНИЯ ВСПЛЫВАЮЩЕГО ОКНА**

1. Создать UI-объект – окно, которое должно всплывать и расположить его на CANVAS.
2. Задать следующий скрипт для всплывающего окна.

**using UnityEngine;**

**using System.Collections;**

**public class SettingsPopup : MonoBehaviour {**

void Start() {    
**Close();  *//  Закрываем окно в момент начала игры.*  
    }   
  
    public void OnOpenSettings() {    
        Open();  *//  Открываем всплывающее окно.*  
    }  
  
    public void Open() {  
        gameObject.SetActive(true);  *//   Активируем объект, чтобы открыть окно.*  
    }    
    public void Close() {  
        gameObject.SetActive(false);  *// Деактивируем объект, чтобы закрыть окно.*  
    }**

**}**

1. Создать на CANVAS две кнопки, соответственно для открытия и закрытия окна при щелчке по кнопкам мышью.
2. Назначить кнопкам общий объект обработки – всплывающее окно и соответственно выбрать в списке доступных функций **OnOpenSettings()**в связанном скрипте для кнопки открытия окна и **Close()**для кнопки закрытия окна.

**ПЕРЕМЕЩЕНИЕ КАМЕРЫ К ЭЛЕМЕНТУ УСТАНОВКИ ПРИ НАВЕДЕНИИ КУРСОРА МЫШИ НА КНОПКУ**

**АЛГОРИТМ РЕАЛИЗАЦИИ ПЕРЕМЕЩЕНИЯ И ПОВОРОТА КАМЕРЫ К ВЫБРАННОМУ ЭЛЕМЕНТУ УСТАНОВКИ**:

1. Создать на сцене по **пустому объекту** для каждого просматриваемого из заданной точки элемента установки и переместить его в режиме сцены так, чтобы было удобно смотреть из него на один из подсвечиваемых элементов.
2. Выбрать пустой объект и в головном меню во вкладке **GameObject** выполнить команду **Align With View** - пустой объект станет так, как настроен вид того, как вы смотрите на объекты. Это нужно для того, чтобы зафиксировать позицию, с которой должна будет смотреть камера на выбранный объект.
3. Проделать то же самое для каждого элемента установки, после чего проинициализировать в UI-скриптах для соответствующих элементов их **anchorObject** пустыми объектами для просмотра с камеры.

СКРИПТ ПЕРЕДВИЖЕНИЯ КАМЕРЫ В ЗАДАННУЮ ТОЧКУ И ПОВОРОТА ЕЕ К ТРЕБУЕМОМУ ОБЪЕКТУ:

|  |
| --- |
| **public class Replacer : MonoBehaviour**  **{**  **bool move = true;**  **Vector3 startPosition;**  **Vector3 needPosition;**  **float speed = 0.01f;**  **float offset = 0;**  **Quaternion startRotation;**  **Quaternion needRotaton;**    **void Start()**  **{**  **startPosition = transform.position;**  **startRotation = transform.rotation;**  // ЕСЛИ, НАПРИМЕР, ПОЗИЦИЯ И ПОВОРОТ ПУСТОГО ОБЪЕКТА ЗАДАНЫ ЗНАЧЕНИЯМИ  **needPosition = new Vector3(5.45f, 8, -22.6f);       needRotaton = new Quaternion(0.7f, 0, 0, 0.7f);**  **}**    **void FixedUpdate()**  **{**  **if(move)**  **{**  **offset+=speed;**  **transform.position = Vector3.Lerp(startPosition, needPosition, offset);**  **transform.rotation = Quaternion.Slerp(startRotation, needRotaton, offset);**  **if (offset >= 1)**  **{**  **move = false;**  **offset = 0;**  **}**  **}**  **}**  **}** |

**СИСТЕМА ЗАДАНИЙ**

Первое задание при начале работы с симулятором это, как правило, задание «Включить установку». После щелчка по кнопке «РАБОТА УСТАНОВКИ» выполняется событие вывода в текстовое окно сообщения: «Включите установку». Первое задание выполняется при совершении события включения установки, когда пользователь щелкнет мышью на объект «Переключатель», что и приведет к выполнению первого задания.

После выполнения первого задания программа должна выдать в информационное окно следующее задание (см. рисунок), которое должно выполняться в соответствии с появляющимся текстовым сообщением и т. д. вплоть до окончания работы с симулятором.



Результатом работы с симулятором установки являются полученные по ходу выполнения заданий измеренные значения, показания приборов и т. п., которые последовательно заносятся в таблицу с выполнением в ней предусмотренных в каждом конкретном эксперименте вычислений.

**ОБРАБОТКА ЩЕЛЧКОВ МЫШЬЮ ПО 3D-ОБЪЕКТАМ СЦЕНЫ**

1. Для обработки события «щелчок мышью по объекту на сцене» необходимо добавить в иерархию объектов новый не отображаемый на сцене объект **Create/UI/EventSystem**
2. В сценарий для объекта необходимо добавить класс **using UnityEngine.EventSystems**
3. В базовый класс добавить новый интерфейс системы событий **IPointerClickHandler**

**using UnityEngine;**

**using System.Collections;**

**using UnityEngine.EventSystems;**

**public class Script11 : MonoBehaviour, IPointerClickHandler {**

**public void OnPointerClick(PointerEventData eventData) {**

**// … (код необходимых действий с объектами по щелчку по ним курсора мыши)**

**}**

**}**

1. К камере необходимо предварительно добавить компоненту **Physics RayCaster** для согласования щелчков мыши по 2D-экрану со щелчками по 3D-объектам на сцене.

Например, для изменения цвета объекта случайным образом после каждого щелчка щелчка по нему курсором мыши можно использовать следующий код:

**float red = Random.Range(.0f, 1.0f);**

**float green = Random.Range(.0f, 1.0f);**

**float blue = Random.Range(.0f, 1.0f);**

**Color col1 = new Color(red, green, blue);**

**gameObject.GetComponent<Renderer>().material.color = col1;**

**ДВИЖЕНИЕ ПАНЕЛИ УПРАВЛЕНИЯ ПОКАЗАТЬ-СКРЫТЬ**

Для того чтобы панель управления с кнопками не заслоняла общую сцену с симулятором установки и скрывалась-открывалась на экране можно на панель добавить скрипт с кодом обработки событий надвижения-ухода курсора мыши с панели. В этом случае необходимо добавить в код скрипта два новых класса: ***IPointerEnterHandler, IPointerExitHandler.***

В целом код управления плавным движением панели может иметь следующий вид:

**using System.Collections;  
using System.Collections.Generic;  
using UnityEngine;  
using UnityEngine.EventSystems;  
  
public class PanelMove : MonoBehaviour, IPointerEnterHandler, IPointerExitHandler {  
RectTransform UIGameobject;** // трансформация UI Панели **float width;** // ширина панели **float changeX;** // смещение панели  
**float speedPanel;** // скорость закрытия панели **enum states     {** // перечисление состояний панели **open,** //открыта **close,** //закрыта **opening,** //открывается **closing** //закрывается **}  
states state = states.open;** // изначальное состояние закрытое  
 **void Start() {** //инициализация переменных

**UIGameobject = gameObject.GetComponent<RectTransform>();** // объект панели

**width = UIGameobject.sizeDelta.x;** // определение ширины панели **speedPanel = 8;** // скорость движения панели**}**

**void Update()  
    {  
        if (state == states.closing)  
        {  
            float x = UIGameobject.anchoredPosition.x;  
            float y = UIGameobject.anchoredPosition.y;  
            x += speedPanel;  
            changeX += speedPanel;  
            UIGameobject.anchoredPosition = new Vector2(x, y);  
            if (changeX > width)  
            {  
                state = states.close;  
                changeX = 0;  
            }  
        }  
        if (state == states.opening)  
        {  
            float x = UIGameobject.anchoredPosition.x;  
            float y = UIGameobject.anchoredPosition.y;  
            x -= speedPanel;  
            changeX += speedPanel;  
            UIGameobject.anchoredPosition = new Vector2(x, y);  
            if (changeX > width)  
            {  
                state = states.open;  
                changeX = 0;  
            }  
  
        }  
    }**

**public void OnPointerEnter(PointerEventData eventData)  
    {  
        if (state == states.close) state = states.opening;  
    }**

**public void OnPointerExit(PointerEventData eventData)  
    {  
        if (state == states.open) state = states.closing;  
    }  
}**

**СОЗДАНИЕ ВЫПАДАЮЩЕГО МЕНЮ КНОПОК**

Для создания выпадающего меню кнопок при наведению на некоторую кнопку на панели управления необходимо для нее создать скрипт, в котором будут использованы массив выпадающих кнопок и устанавливаться изменяющиеся размеры расположения таких кнопок на двумерной области в ***Canvas*** с использованием двумерных векторов типа ***Vector2***.

Например, для выпадающего списка меню кнопок элементов в симуляторе установки можно использовать следующий код скрипта:

**using System.Collections;  
using System.Collections.Generic;  
using UnityEngine;  
using UnityEngine.EventSystems;  
using UnityEngine.UI;  
  
public class Dropdown : MonoBehaviour, IPointerEnterHandler, IPointerExitHandler {  
  
    public GameObject panelTrigger; // нужно связать переменную с объектом  
    RectTransform triggerTransfrom; // переменная типа прямоугольника   
    public Button button; // объект кнопки  
    RectTransform buttonTransfrom; // прямоугольник для кнопки  
    Vector2 buttonSize; // двумерный вектор для размещения кнопки  
    public GameObject dropdownList; // объект списка меню  
    RectTransform dropdownTransfrom; // прямоугольник для списка меню  
    Vector2 dropdownSize; // двумерный вектор для списка меню  
    public Sprite buttonNormalState; // для накладывания начальной текстуры на кнопку  
    public Sprite buttonHighlightedState; // для накладывания конечной текстуры на кнопку**

**void Start() // инициализация и задание значений переменным   
    {  
        if (dropdownList != null)  
        {  
            triggerTransfrom = panelTrigger.GetComponent<RectTransform>();  
            buttonTransfrom = button.GetComponent<RectTransform>();  
            dropdownTransfrom = dropdownList.GetComponent<RectTransform>();  
            buttonSize = buttonTransfrom.sizeDelta;  
            dropdownSize = dropdownTransfrom.sizeDelta; // размер прямоугольника списка меню  
            dropdownTransfrom.sizeDelta = new Vector2(0, 0);  
            triggerTransfrom.sizeDelta = buttonSize; // размер прямоугольника кнопки  
            dropdownList.SetActive(false); // активизация выпадающего списка меню  
        }**

**}  
  
    public void OnPointerEnter(PointerEventData eventData) // надвинули курсор на кнопку  
    {  
          
        button.GetComponent<Image>().sprite = buttonHighlightedState;  
        dropdownTransfrom.sizeDelta = dropdownSize;  
        triggerTransfrom.sizeDelta = buttonSize + dropdownSize; // общий размер прямоугольника  
        dropdownList.SetActive(true); // активация списка меню   
    }  
  
    public void OnPointerExit(PointerEventData eventData) // убрали курсор с кнопки  
    {  
        if (dropdownList != null)  
        {  
            dropdownTransfrom.sizeDelta = new Vector2(0, 0);  
            triggerTransfrom.sizeDelta = buttonSize; // возврат размера прямоугольника  
            button.GetComponent<Image>().sprite = buttonNormalState; // возврат цвета кнопки  
            dropdownList.SetActive(false); // дезактивация списка меню   
        }  
    }  
}**