**ПРОГРАММИРОВАНИЕ**

**ОБРАБОТКИ СОБЫТИЙ**

**ДЛЯ СИМУЛЯТОРА**

**ПРОГРАММИРОВАНИЕ ИНТЕРАКТИВНОГО UI**

**С ИСПОЛЬЗОВАНИЕМ КНОПОК BUTTON**

Программирование интерактивного взаимодействия пользователя с элементами **UI** сводится к стандартной процедуре, общей для всех элементов:

1. В сцене на основе **CANVAS** создается рабочий **UI**-объект и кнопка **Button**.
2. К созданному **UI**-объекту добавляется сценарий, который будет вызываться при обращении к нему при совершении событий, доступных для кнопок.
3. В сценарий необходимо добавить директиву **using UnityEngine.UI;** для работы с кодом **UI.**
4. Для кнопки выбирается доступное для ее обработки событие и затем создается ее связь с **UI**-объектом, к которому присоединен разработанный предварительно соответствующий **сценарий**.
5. Для кнопки выбирается в качестве функции имя сценария, добавленного для **UI**-объекта, и в раскрывающемся списке функций указывается та функция в сценарии, которая должна выполнить действие при совершении события с кнопкой.

По умолчанию у кнопки доступно событие щелчок мышью - **Click()**, для использования других событий, например, надвижение курсора мыши **PointerEnter()**, необходимо к кнопке добавить компоненту **Event/Event Trigger** и затем выбрать из предложенного списка нужное событие.

**ПРОГРАММИРОВАНИЕ ДРУГИХ СОБЫТИЙ ДЛЯ КНОПКИ**

По умолчанию кнопка реагирует только на событие **OnClick**, в то время как UI-элементы на CANVAS могут отвечать на разные варианты взаимодействий. Событие **OnClick** кнопки отвечает только на полноценный щелчок - кнопка мыши нажимается, а затем отпускается.

Для программирования взаимодействий, отличных от **OnClick**, используется компонент **EventTrigger** - добавим **Add Component** к кнопке новый компонент **Event,** а в меню этого компонента выберем вариант **EventTrigger**. Щелкнув на кнопке **Add New Event Type** раскроем список из 17 различных событий для выбора и добавления их к компоненту **EventTrigger.**

Например, для программирования реакции на надвижение курсора мыши 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 со скриптом. Последовательность действий будет той же, что и для события **OnClick**, просто на этот раз мы укажем реакцию на другое событие – **PointerEnter**.

*Для использования переменных, связываемых с элементами* ***Canvas*** *необходимо в список директив скрипта добавить модуль* ***using UnityEngine.UI;***

Естественно, для выполнения соответствующих действий при наведении курсора мыши в скрипт обрабатываемого объекта необходимо добавить новый метод (функцию), например,с именем **OnPointer()**:

**using System.Collections;  
using System.Collections.Generic;  
using UnityEngine;  
using UnityEngine.UI;                 //  ИМПОРТ МОДУЛЯ ДЛЯ РАБОТЫ С КОДОМ UI  
  
public class qwest1 : MonoBehaviour  
{**

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

**public void OnPointer () {**

**message.text = "Установить начальные значения для всех элементов симулятора установки";**

**}**

**}**

Наконец, в списке функций, доступных из скрипта обрабатываемого объекта, необходимо выбрать функцию (метод) **OnPointer ().**

**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();**//  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.**}**

**}**

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 связанном с окном скрипте **SettingsPopup** в списке доступных в скрипте функций функцию **Open()**для кнопки открытия окна и **Close()**для кнопки закрытия окна.

**ПОДСВЕТКА ЭЛЕМЕНТА УСТАНОВКИ ПРИ НАВЕДЕНИИ КУРСОРА МЫШИ НА КНОПКУ С НАЗВАНИЕМ ЭЛЕМЕНТА В СПИСКЕ**

Скрипт должен быть помещен на элемент установки на сцене, который нужно подсветить при наведении курсора мыши на соответствующую кнопку:

|  |
| --- |
| **using System.Collections; using System.Collections.Generic; using UnityEngine;  public class ChangeColor : MonoBehaviour {          public void ChangCol()     {         GetComponent<Renderer> ().material.color = new Color (1, 0, 0);        }      public void ChangCol1()     {         GetComponent<Renderer> ().material.color = new Color (1, 1, 1);        } }** |

Затем необходимо повторить процедуру вызова из скрипта функций **ChangCol()** и **ChangCol1()** соответственно для событий наведения курсора мыши на кнопку **Pointer Enter** и **Pointer Exit.**

Этот же скрипт можно повторно использовать, добавляя его для всех функциональных элементов установки, а затем повторить процедуру вызова из скриптов всех элементов установки функций **ChangCol()** и **ChangCol1()** соответственно для событий наведения курсора мыши на кнопку **Pointer Enter** и **Pointer Exit.**

**ПЕРЕМЕЩЕНИЕ КАМЕРЫ К ВЫБРАННОМУ ЭЛЕМЕНТУ УСТАНОВКИ**

АЛГОРИТМ РЕАЛИЗАЦИИ ПОВОРОТА КАМЕРЫ И ЕЕ РАКУРСА К ВЫБРАННОМУ ЭЛЕМЕНТУ УСТАНОВКИ.

***1 ВАРИАНТ***

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.

***2 ВАРИАНТ***

1. Активизировать камеру в окне **Hierarchy** и запомнить в окне **Inspector** ее первоначальные положение **Position** и угол поворота **Rotation**.
2. Передвинуть и повернуть камеру для удобного ракурса просмотра выбранного элемента управления ориентируясь на вид сцены в окне предварительного просмотра из камеры **Camera Preview**.
3. Использовать полученные новые координаты и угол поворота камеры в окне **Inspector** в программном коде для поворота камеры.

СКРИПТ ПЕРЕДВИЖЕНИЯ КАМЕРЫ В ЗАДАННУЮ ТОЧКУ ДЛЯ УДОБНОГО ПРОСМОТРА ОБЪЕКТА:

**public class replacer1 : MonoBehaviour  
{  
        bool move = false;  
        float speed = 0.01f;  
        float offset = 0;  
        Vector3 startPosition;  
        Vector3 needPosition;  
        Quaternion startRotation;  
        Quaternion needRotaton;  
  
       public void Move()  
        {  move = true;  
            startPosition = transform.position;  
            startRotation = transform.rotation;  
            needPosition = new Vector3(277.0f, 251, 6);  
            needRotaton = Quaternion.AngleAxis(-180, new Vector3(0,1,0));        }  
        public void Move1()  
        {  move = true;  
            startPosition = transform.position;  
            startRotation = transform.rotation;  
            needPosition = new Vector3(284, 253, 0f);  
            needRotaton = Quaternion.AngleAxis(-90, new Vector3(0,1,0));        }  
        public void Move2()  
        {   move = true;  
            startPosition = transform.position;  
            startRotation = transform.rotation;  
            needPosition = new Vector3(277, 255, 0f);  
            needRotaton = Quaternion.AngleAxis(90, new Vector3(1,0,0));        }  
        public void Move0()  
        {   move = true;  
            startPosition = transform.position;  
            startRotation = transform.rotation;  
            needPosition = new Vector3(276.12f, 252.9f, -6.93f);  
            needRotaton = Quaternion.AngleAxis(0, new Vector3(0,1,0));        }**

**void Update()  
        {   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;      }  
            }  
        }  
    }**