**ПРОГРАММИРОВАНИЕ**

**UI-ИНТЕРФЕЙСА**

**СИМУЛЯТОРА УСТАНОВКИ**

**НА ОСНОВЕ CANVAS**

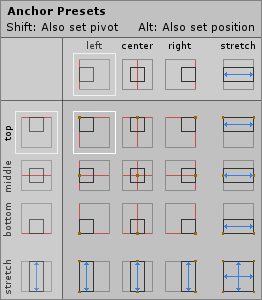
**УПРАВЛЕНИЕ ПОЛОЖЕНИЕМ ЭЛЕМЕНТОВ UI**

Холст (**Canvas**) в редакторе Unity представляет собой 2-мерный прозрачный объект, который накладывается на экран и обозначает границы экрана, а размещаемые на холсте **элементы UI** появляются на этом экране в заданных точках.

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*.

При этом объект остается статичным относительно точки привязки, а сама она может перемещаться относительно холста. Точку привязки можно задать, например, как «центр экрана», и она будет оставаться в центре, когда экран меняет свой размер. Аналогично привязка к правой стороне экрана позволит объекту оставаться справа даже при изменении размеров.



Точки привязки позволяют подстраиваться не только под изменение положения, но и под изменение размера.

Привязку можно отредактировать таким образом, что при увеличении размера экрана изображения будут растягиваться вместе с ним.

Способ наложения элементов **UI** друг на друга определяется порядком их следования на вкладке **Hierarchy**. Чтобы расположить всплывающее окно поверх остальных элементов **UI** необходимо перетащить окно в самый низ иерархического списка дочерних элементов холста, чтобы оно отображалось поверх всего остального.

**ВЫВОД СООБЩЕНИЯ В ТЕКСТОВОЕ ОКНО ПОСЛЕ ЗАПУСКА ПРИЛОЖЕНИЯ**

Создать скрипт для UI-элемента – текстового окна:

**using UnityEngine;**

**using UnityEngine.UI;** //ИМПОРТ МОДУЛЯ ДЛЯ РАБОТЫ С КОДОМ UI

**using System.Collections;**

**public class InText : MonoBehaviour {**

**[SerializeField]**

**private Text message;** //ОБЪЕКТНАЯ ПЕРЕМЕННАЯ ТИПА TEXT

**void Update() {**

**message.text = Time.time.ToString();**

**}**

**}**

В окне Inspector для текстового окна нужно, прежде всего, созданный текстовый объект связать с объектной переменной в поле **Message** скрипта для кнопки.

После этого код сценария с именем, например, **InText** начнет автоматически отображать в текстовом окне значения времени, которое задается в коде конструкцией **Time.time.ToString().**

**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.

**ВЫВОД СООБЩЕНИЯ В ТЕКСТОВОЕ ОКНО ЩЕЛЧКОМ МЫШИ ПО КНОПКЕ**

Для того чтобы вывод сообщения в текстовое окно происходил по событию от пользователя необходимо создать в скрипте с именем, например **Windows\_Up,** функцию для вывода нужного сообщения, например, **OnSettings().** При том объектная переменная **message** типа **Text** должна быть связана с объектом на **Canvas** – с соответствующим текстовым окном.

КОД ДЛЯ ВЫВОДА СООБЩЕНИЯ В ТЕКСТОВОЕ ОКНО ПРИ ЩЕЛЧКЕ КУРСОРОМ МЫШИ ПО КНОПКЕ:

**using UnityEngine;**

**using System.Collections;**

**public class Windows\_Up : MonoBehaviour {**

**[SerializeField]**

**Text message;**

**public void OnSettings() {** // МЕТОД, ВЫЗЫВАЕМЫЙ ПОЛЬЗОВАТЕЛЕМ ЩЕЛЧКОМ ПО КНОПКЕ

**message.text="Установка начальных значений";**

**}**

**…**

**}**

Затем в нижней части панели **Inspector** необходимо активизировать элемент кнопки **On Click()**, щелкнуть на кнопке со значком + (плюс), добавить элемент кнопки (в листинге присутствуют как ячейка для объекта со скриптом , так и меню для вызываемой функции), перетащите на ячейку объект-кнопки, выделите в меню строку с именем скрипта **Windows\_Up** и выбрать в дополнительном меню справа функцию **OnSettings()** имеющуюся в скрипте **Windows\_Up.**

КОД ДЛЯ ВЫВОДА СООБЩЕНИЯ В ТЕКСТОВОЕ ОКНО ПРИ УХОДЕ КУРСОРА МЫШИ С КНОПКИ:

**…**

**public void OnExit() {**// МЕТОД, ВЫЗЫВАЕМЫЙ ПОЛЬЗОВАТЕЛЕМ ПРИ УХОДЕ КУРСОРА МЫШИ С КНОПКИ **message.text = "Симулятор предназначен для проведения лабораторного практикума в виртуальном режиме с установкой, представленной на экране компьютера";   
    }**

**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, которые нужно подсвечивать, 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.**

**АЛГОРИТМ ПЕРЕДВИЖЕНИЯ КАМЕРЫ В ЗАДАННУЮ ТОЧКУ ДЛЯ УДОБНОГО ПРОСМОТРА ОБЪЕКТА:**

Для получения положения камеры в нужном ракурсе для просмотра соответсвующего элемента установки можно поступать следующим образом:

1. Запомнить значения позиции **Position** и поворота **Rotation** камеры в исходном положении.
2. Рассчитать (возможно вручную на отдельном чертеже-схеме) желаемую позицию и поворот камеры для удобного ракурса просмотра соответсвующего элемента установки – сдвиг рассчитывается как разность значений позиций XYZ между установкой и камерой, а поворот камеры на нужный угол определяется в конструкции кватерниона.

Например, для просмотра элемента на обратной стороне установки требуемый угол поворота будет равен:

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

а для просмотра сверху:

**needRotaton = Quaternion.AngleAxis(90, new Vector3(1,0,0));**

1. В скрипте для перемещения камеры в нужные ракурсы задать для каждой кнопки отдельные функции.

Например, функцию **Move()**для просмотра обратной стороны установки:

**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));}**

Здесь булевая переменная **move**используется для запуска движения, а стартовые позиция и угол поворота задаются каждый раз для текущего положения камеры перед ее перемещением.

**СКРИПТ ПЕРЕДВИЖЕНИЯ КАМЕРЫ В ЗАДАННУЮ ТОЧКУ С ПОВОРОТОМ**

**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**() //функция для просмотра камерой обратной стороны установки  
**{  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()** // функция для просмотра камерой установки справа **{  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()** // функция для просмотра камерой установки сверху **{   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()** // функция для возврата камеры к исходномуположению **{   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()** // выполнение поворота камеры в требуемый ракурс

**{   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;      }  
            }  
        }  
    }**