**ПРОГРАММИРОВАНИЕ**

**UI-ИНТЕРФЕЙСА**

**СИМУЛЯТОРА УСТАНОВКИ**

**НА ОСНОВЕ CANVAS**

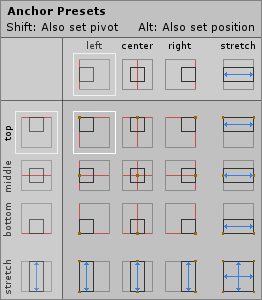
**УПРАВЛЕНИЕ ПОЛОЖЕНИЕМ ЭЛЕМЕНТОВ UI**

Холст (**Canvas**) в редакторе Unity 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 на поле **Message** скрипта для кнопки. После этого код сценария **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";   
    }**

**СОЗДАНИЕ ВСПЛЫВАЮЩЕГО ОКНА**

КОД ДЛЯ ОТКРЫТИЯ-ЗАКРЫТИЯ ВСПЛЫВАЮЩЕГО ТЕКСТОВОГО ОКНА:

**…**

**void Start() {    
        Close();**//  ЗАКРЫТЬ ВСПЛЫВАЮЩЕЕ ОКНО ПРИ ЗАПУСКЕ ПРОГРАММЫ**}   
  
    public void OnOpenSettings() {    
        Open();  //**ОТКРЫТЬ ВСПЛЫВАЮЩЕЕ ТЕКСТОВОЕ ОКНО**}  
  
    public void Open() {  
        gameObject.SetActive(true);**//   АКТИВИРОВАТЬ ОБЪЕКТ, ЧТОБЫ ОТКРЫТЬ ОКНО.**}**

**public void Close() {  
        gameObject.SetActive(false);**// ДЕАКТИВИРОВАТЬ ОБЪЕКТ, ЧТОБЫ ЗАКРЫТЬ ОКНО.**}**

**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:*

|  |
| --- |
| **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);        } }** |

**ПОВОРОТ КАМЕРЫ К ЭЛЕМЕНТУ УСТАНОВКИ, ПРИ ЩЕЛЧКЕ КУРСОРОМ МЫШИ ПО КНОПКЕ С НАЗВАНИЕМ ЭЛЕМЕНТА В СПИСКЕ**

АЛГОРИТМ РЕАЛИЗАЦИИ ПОВОРОТА КАМЕРЫ И ЕЕ РАКУРСА 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 ЗАДАННЫЙ РАКУРС НА СЦЕНЕ:

|  |
| --- |
| **public class Replacer : MonoBehaviour**  **{**  **bool move = true;**  **float speed = 0.01f;**  **float offset = 0;**  **Vector3 startPosition;**  **Vector3 needPosition;**  **Quaternion startRotation;**  **Quaternion needRotaton;**    **void Start()**  **{**  **startPosition = transform.position;**  **startRotation = transform.rotation;**  // ЕСЛИ, НАПРИМЕР, ПОЗИЦИЯ И ПОВОРОТ ПУСТОГО ОБЪЕКТА ЗАДАНЫ ЗНАЧЕНИЯМИ  **needPosition=newVector3(5.45f,8,-22.6f);        needRotaton = new Quaternion(0.7f, 0, 0, 0.7f);**  **}**    **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;**  **}**  **}**  **}**  **}** |