Создание динамических эффектов на html-странице: фиксация заголовка, замена текста на рисунок, увеличение размеров картинки

Для создания адаптивного дизайна Web-страницы необходимо задавать ***ширину*** блоковых элементов на странице в ***процентах***.

Фиксация заголовка позволяет видеть заголовок Web-страницы при прокрутке. Для того, чтобы зафиксировать заголовок, нужно в CSS свойствах блокового элемента установить параметр **position: fixed**.

```
<body>

<div style="position:fixed;  top:0px; width: 100%; z-index:2; background-color: #FFFFFF; font-size: 24px; text-align:center; margin:0px; padding:0px" >

<p><strong>Web-дизайн и разработка мультимедийных изданий</strong></p>

<p></p>

</div>

…
```

Для замены цвета текста при наведении курсора мыши нужно использовать события **onmouseover** и **onmouseout**. Именно эти два события будут менять свойство color на нужный параметр. Т.е. по событию устанавливается новое значение свойства:

```
<body>

Этот <span style="color:blue" onmouseover="this.style.color='red' " onmouseout="this.style.color='blue'">синий текст </span>при наведении мыши становится красным

…
```

Увеличение картинки при наведении курсора мыши также осуществляется через использование событий **onmouseover** и

onmouseout. По событию **onmouseover** устанавливается новая ширина **width** изображения, а по событию **onmouseout** устанавливается изначальная ширина:

```
<body>

<img src="logo.jpg" onmouseover="this.width=this.width*5" onmouseout="this.width=this.width/5" />

…
```

В данном примере при наведении курсора ширина картинки увеличивается в 5 раз, а при выходе курсора вновь уменьшается в 5 раз.

Для замены слова на изображение используются две функции на языке JavaScript: первая функция **toim()** ищет элемент в теле html-документа с помощью метода **getElementById** и перезаписывает его содержимое с помощью метода **innerHTML**. Здесь в качестве параметра передаем html-тег для изображения и ссылку на картинку. Вторая функция **totext()** перезаписывает содержимое элемента на текст.

Эти две функции передаются в качестве параметров событиям **onmousedown** и **onmouseout**, отвечающим за нажатие мыши и уход курсора от цели соответственно. Таким образом, при клике мышкой сработает событие **onmousedown**; оно запустит функцию **toim()**, которая заменит текст на картинку:

```
<script>
function toim() {document.getElementById("t1").innerHTML="<img
src='univer.jpg' />"}
function totext()
{document.getElementById("t1").innerHTML="<span>БГТУ</span>"}
</script>
</head>
<body>
<p>При щелчке по слову <span id="t1" onmousedown="toim()"
onmouseout="totext()">БГТУ</span> оно заменяется на фото
университета</p>
```

Похожим способом можно получить картинки из списка. Для этого при событии **onmousedown** вызывается метод **getElementById**, который находит по идентификатору элемент и изменяет ссылку на изображение:

```
<body>

<p><img src="logo.jpg" id="r1" width="400" height="400" /> </p>

<p
onmousedown="document.getElementById('r1').src='univer.jpg'">университет</p>

<p onmousedown="document.getElementById('r1').src='nebo.jpg'">небо</p>
```

Программа для обхода камерой вокруг установки в помещении

Для управления движения камеры относительно установки необходимо, прежде всего, выбрать центральную точку на установке, для чего в скрипте для управления движением камеры нужно перетянуть в добавленном скрипте в окне ***Inspector*** *для камеры центральный объект установки в поле объектной переменной типа* ***Transform****, например,* ***targetPos*** *, что будет определять точку, вокруг которой будет вращаться и двигаться камера.*

```
public class Scroll : MonoBehaviour
{
    [SerializeField] //позволяет сериализовать поле(использовать потом объект как хранимую структуру)
    Transform targetPos; //точка, вокруг которой движемся
    int sensivity = 3; //чувствительность мыши
    int maxdistance = 20; //максимальное расстояние
    int mindistance = 1; //минимальное расстояние
//  ФУНКЦИЯ ОГРАНИЧЕНИЯ ПРЕДЕЛОВ ДВИЖЕНИЯ КАМЕРЫ
     bool ControlDistance (float distance)
    {
        if (distance > mindistance && distance < maxdistance) return true; //если расстояние больше минимального и меньше максимального
        return false;
    }
    // ВРАЩЕНИЕ ВОКРУГ ЦЕНТРАЛЬНОЙ ТОЧКИ УСТАНОВКИ С ЗАЖАТОЙ ПРАВОЙ КЛАВИШЕЙ МЫШИ
    void Update() //определяет код, перерисовывающий экран 50 раз в секунду
    {
        if (Input.GetMouseButton(1)) //если нажата правая клавиша мыши
        {
            transform.RotateAround(targetPos.position, Vector3.up, Input.GetAxis("Mouse X") * sensivity); //метод RotateAround осуществляет вращение вокруг (targetPos.position - точка вращения, Vector3.up - ось вращения, Input.GetAxis("Mouse X") * sensivity - определяют угол); Input.GetAxis("Mouse X") - значение виртуальной
```

```
оси
        }
        // ДВИЖЕНИЯ КАМЕРЫ В СТОРОНЫ КЛАВИШАМИ
        float x = Input.GetAxis("Horizontal"); //значение
виртуальной оси (горизонтальной) вкладываем в переменную x
        float y = Input.GetAxis("Vertical"); //значение виртуальной
оси (вертикальной) вкладываем в переменную y
        if (x != 0 || y != 0) //если x и y не равны 0
        {
            Vector3 newpos = transform.position +
(transform.TransformDirection(new Vector3(x, 0, 0)) + Vector3.up *
y) / sensivity; //определение новой позиции (Vector3 - тип данных,
хранит xyz координаты; TransformDirection переводит координаты из
локальных в мировые)
            if (ControlDistance(Vector3.Distance(newpos,
targetPos.position))) transform.position = newpos; //если не
достигнуты пределы движения камеры, то устанавливаем новую
позицию
        } } }
```

Программа сдвига камеры с ограничениями вдоль и вглубь помещения с установкой

*Для управления движения камерой относительно установки необходимо, выбрать центральную точку на установке: в скрипте для управления движением камерой нужно перетянуть в добавленном скрипте в окне **Inspector** для камеры центральный объект установки в поле объектной переменной типа **Transform**, например, **targetPos**, что будет определять точку, вокруг которой будет вращаться и двигаться камера.*

```
public class Scroll : MonoBehaviour
{
    [SerializeField] //позволяет сериализовать поле (использовать потом объект как хранимую структуру)
    Transform targetPos; //точка, вокруг которой движемся
    int sensivity = 3; //чувствительность мыши
    int maxdistance = 20; //максимальное расстояние
    int mindistance = 1; //минимальное расстояние

//  ФУНКЦИЯ ОГРАНИЧЕНИЯ ПРЕДЕЛОВ ДВИЖЕНИЯ КАМЕРЫ
     bool ControlDistance (float distance)
    {
        if (distance > mindistance && distance < maxdistance) return true; //если расстояние больше минимального и меньше максимального
        return false;
    }
  // ДВИЖЕНИЯ КАМЕРЫ В СТОРОНЫ КЛАВИШАМИ
float x = Input.GetAxis("Horizontal"); // значение виртуальной горизонтальной оси присваиваем переменной x
float y = Input.GetAxis("Vertical"); // значение горизонтальной горизонтальной оси присваиваем переменной y

  if (x != 0 || y != 0) // если переменные, в которых лежит ввод по горизонтальным или вертикальным осям не равен 0 (что-то произошло, какой-то ввод)
  {
    Vector3 newpos = transform.position + (transform.TransformDirection(new Vector3(x, 0, 0)) + Vector3.up *
```

```
y) / sensivity; //определение новой позиции (Vector3 - тип данных,
хранит xyz координаты; TransformDirection переводит координаты из
локальных в мировые)
      if (ControlDistance(Vector3.Distance(newpos,
targetPos.position)))// если не достигнуты пределы движения камеры
transform.position = newpos; // то устанавливаем новую позицию
   }
}
```

Программа приближения и удаления камеры относительно установки

*Для управления движения камерой относительно установки необходимо, выбрать центральную точку на установке: в скрипте для управления движением камерой нужно перетянуть в добавленном скрипте в окне **Inspector** для камеры центральный объект установки в поле объектной переменной типа **Transform**, например, **targetPos**, что будет определять точку, вокруг которой будет вращаться и двигаться камера.*

```
public class Scroll : MonoBehaviour
{
    [SerializeField] //позволяет сериализовать поле (использовать потом объект как хранимую структуру)
    Transform targetPos; //точка, вокруг которой движемся - тут это не нужно, вроде
    int sensivity = 3; //чувствительность мыши
   int maxdistance = 20; //максимальное расстояние
    int mindistance = 1; //минимальное расстояние

//  ФУНКЦИЯ ОГРАНИЧЕНИЯ ПРЕДЕЛОВ ДВИЖЕНИЯ КАМЕРЫ
     bool ControlDistance (float distance)
    {
        if (distance > mindistance && distance < maxdistance) return true; //если расстояние больше минимального и меньше максимального
        return false;
    }
// ПРИБЛИЖЕНИЕ И УДАЛЕНИЕ КАМЕРЫ ОТ УСТАНОВКИ ПРОКРУТКОЙ КОЛЕСА МЫШИ
if (Input.GetAxis("Mouse ScrollWheel") != 0) // если колесико крутится
   {
      Vector3 newpos = transform.position + transform.TransformDirection(Vector3.forward * Input.GetAxis("Mouse ScrollWheel") * scrollSpeed); // движемся вперед на столько, на сколько крутится колесико со скоростью, которую мы задали ранее
      if (ControlDistance(Vector3.Distance(newpos,
```

```
targetPos.position))) // если не достигнуты пределы движения
камеры
      transform.position = newpos; // то устанавливаем новую
позицию
        }
}
```

Правила создания и размещения на экране элементов интерфейса Canvas

Холст (Canvas) в редакторе Unity - границы экрана, а размещаемые на холсте элементы UI появляются на этом экране в заданных точках.

Создание Canvas (на сцене может быть только один): панель Hierarchy > + > UI > Canvas

Создание элементов UI: панель Hierarchy > + > UI > *выбираем нужный элемент*

Элементы UI должны быть подчинены Canvas в дереве объектов (Hierarchy)

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.

- **Привязка (Anchor) объекта** - точка его присоединения к холсту или экрану, относительно этой точки указывается положение объекта.

Объект остается статичным относительно точки привязки, а сама она может перемещаться относительно холста.

Параметры привязки объекта по вертикали: **top, middle, button**.
Параметры привязки объекта по горизонтали: **left, center, right**.

Комбинируя вертикальную и горизонтальную параметры привязки, можно добиться необходимого позиционирования.

Изменяя размер экрана, Anchor будет “удерживать” элемент на месте, куда его привязали (т.е. **точки привязки подстраиваются под изменение положения**)

Также **точки привязки позволяют подстраиваться и под изменение размера**.

Привязку можно отредактировать таким образом, что при увеличении размера экрана изображения будут растягиваться вместе с ним. Для этого используется **stretch** по вертикали и горизонтали.

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.

Принцип обработки щелчка мышью по кнопке на Canvas

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**.
   p.s. деду так не пишите, но иногда легче просто воспользоваться инспектором, а не делать действия через код
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, наведении курсора мыши **PointerEnter()**, необходимо к кнопке добавить компоненту **Event/Event Trigger** и затем выбрать из предложенного списка нужное событие.

Принцип обработки надвижения и ухода курсора мыши с кнопки на Canvas

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

После чего в коде добавить обработчики **OnPointerEnter()** или **OnPointerExit()**

Процедура создания «всплывающего» текстового окна при наведении курсора на кнопку

Процедура создания «всплывающего» текстового окна при наведении курсора на кнопку

Алгоритм:

1) Создать скрипт с именем TextWindow_Up (Для примера)
2) Создать в скрипте TextWindow_Up функцию для вывода нужного сообщения, у нас будет называться TextShow()
3) В коде должна присутствовать объектная переменная Message типа Text и связана с объектом на Canvas (соответствующим текстотовым окном)
4) 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 **TextWindow_Up** и выбрать в дополнительном меню справа функцию **TextShow()** имеющуюся в скрипте **TextWindow_Up.**

Программа «всплывающего» текстового окна при наведении курсора на кнопку

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.
    }
```

(В inspector добавляем для кнопки код, указываем в качестве префаба тестовое окно)

Алгоритм выбора удобного ракурса для размещения камеры на сцене

Нужно выбрать центральную точку на установке, для чего в скрипте для управления движением камеры нужно перетянуть центральный объект установки в поле объектной переменной типа **Transform**, что будет определять точку, вокруг которой будет вращаться и двигаться камера.

Программа выбора оптимального ракурса размещения камеры при щелчке мышью по кнопке

```
public class CameraScript : 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;
            }
        }
    }
}
```

Программа подсветки элемента установки при наведении курсора мыши на кнопку

Скрипт должен быть помещен на пустой объект канваса

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;

public class ChangeColor : MonoBehaviour
{
    public GameObject gameObj1; //создаем префаб объекта, который
будет подсвечиваться
    public void ChangCol()
    {
//задаем новый цвет объекта для подсвечивания
        gameObj1.GetComponent<Renderer> ().material.color = new
Color (1, 0, 0);
    }

    public void ChangCol1()
    {
//задаем исходный цвет объекта до подсвечивания
        gameObj1.GetComponent<Renderer> ().material.color = new
Color (1, 1, 1);
    }
}
```

В инспекторе необходимо указать объект, который будет подсвечиваться, в качестве префаба.

Чтобы подсветка появлялась при наведении курсора:

Зайти в инспектор для кнопки, по которой будет происходить подсветка, Добавить EventTriger, затем нажать на «+», выбрать TrigerEnter(чтобы подсветка срабатывала при наведении курсора на кнопку), в нем указать функцию ChangCol и пустой объект канваса.

Чтобы подсветка объекта пропадала при уходе курсора с кнопки:

Зайти в инспектор для кнопки, по которой будет происходить подсветка, Добавить EventTriger, затем нажать на «+», выбрать TrigerExit(чтобы подсветка пропадала при уходе курсора с кнопки), в нем указать функцию ChangCol1 и пустой объект канваса.

Программа появления на экране текстового окна при щелчке по кнопке

```
using UnityEngine;
using System.Collections;
public class TextWindow_Up: MonoBehaviour {
[SerializeField]
Text message;
public void TextShow(){                    //  МЕТОД, ВЫЗЫВАЕМЫЙ ПОЛЬЗОВАТЕЛЕМ ЩЕЛЧКОМ ПО КНОПКЕ
      message.text="Установка начальных значений";
  }
...
}
```

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

```
...
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";
    }
```

КОД ДЛЯ ОТКРЫТИЯ-ЗАКРЫТИЯ ВСПЛЫВАЮЩЕГО ТЕКСТОВОГО ОКНА: (ДОПОЛНИТЕЛЬНО)

```
 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.
}
```

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

Программа движения панели управления «показать-скрыть» при наведении-уходе курсора

```
public class PanelMove : MonoBehaviour, IPointerEnterHandler, IPointerExitHandler

{

    RectTransform UIGameobject; // трансформация UI Панели

     // transform – точка; rectTransform – прямоуг-к, внутри к-го мб размещен элемент UI

    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) // закрытие
        {
                        float x = UIGameobject.anchoredPosition.x; // anchoredPosition - позиция центра RectTransform с учетом опорной точки привязки
            float y = UIGameobject.anchoredPosition.y;
            // задаем изменение позиции:
            x += speedPanel;
            changeX += speedPanel;
            // применяем изменение позиции панели
            UIGameobject.anchoredPosition = new Vector2(x, y);
            if (changeX > width)
            {
                state = states.close; // статус меняется на закрыто
                changeX = 0; // смещение прекращается
            }
        }
        if (state == states.opening) // открытие
        {
            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; // статус меняется на открыто
                changeX = 0; // смешение прекращается
            }
        }
    }

        public void OnPointerEnter(PointerEventData eventData) // надвинули курсор на кнопку
    {
          if (state == states.close) state = states.opening; // если панель была закрыта - открываем
    }
     public void OnPointerExit(PointerEventData eventData) // убрали курсор с кнопки
    {
          if (state == states.open) state = states.closing; // если панель была открыта, закрываем
    }
}
```

Программа создания массива выпадающего меню кнопок на Canvas

```
// скрипт создается на кнопку

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); // деактивация списка
меню

        }

    }

}
```

Правила создания анимации в среде Unity

Для создания анимации нужно выбрать объект, который будет анимироваться, открыть для него вкладку **Window/Animation** и создать, прежде всего, отдельный файл анимации в открывающемся окне, нажав кнопку **Create**. После сохранения файла состояния анимации объекта, например **DoorAnimation.anim,** в соответствующей папке проекта создается непосредственно анимация.

Для создания анимации необходимо нажать на красную кнопку «Запись», после чего создать ключевые кадры начального положения объекта, нажав на кнопку «Добавить ключевой кадр». (Ключевые кадры можно создавать например для позиции, размера и вращения объекта). После этого необходимо передвинуть ползунок на несколько кадров вперед и переместить объект на новую позицию (либо повернуть объект, либо изменить его размер). Для завершения анимирования объекта необходимо повторно нажать красную кнопку, для окончания записи анимации.

Далее написано два варианта, как у деда в лекции и как оно работает на самом деле

КАК У ДЕДА В ЛЕКЦИИ

Добавить к пустому объекту, к которому привязана дверь, контроллер анимации **Animator** и объявить триггерную переменную для открытия двери, по аналогии так же как у персонажа.

Для создания контроллера анимации нужно кликнуть правой кнопкой мыши по свободной области в папке **Assets**, выполнить команду **Create->Animator Controller** и дать имя контроллеру. Дважды кликнув на созданный контроллер, перейти в окно редактора.

В начале есть два блока, это **Entry**, с которого будут начинать проигрываться анимации и **Any State** – особое состояние, которое всегда существует и позволяет перейти от любого состояния к конкретному, а также блок выхода **Exit**.

Создать **Empty State** как состояние по умолчанию, щелкнув правой кнопкой мыши в свободной части окна аниматора и выполнив команду **Create State/Empty,** в качестве состояния

покоя у двери и добавить созданную анимацию отдельным состоянием.

Переходы к состоянию анимации открытия двери и обратно должен происходить по булевой переменной **Bolean**, причем обратно - по условию установки режима **Has Exit Time**. При этом после создания анимации двери необходимо *снять зацикливание* в контроллере анимации.

КАК РАБОТАЕТ НА САМОМ ДЕЛЕ

После создания анимации для объекта, автоматически создается контроллер анимации, при помощи которого можно задавать режим работы анимации.

Сперва необходимо создать **Empty State** как состояние по умолчанию, щелкнув правой кнопкой мыши в свободной части окна аниматора и выполнив команду **Create State/Empty,** в качестве состояния покоя объекта и добавить созданную анимацию отдельным состоянием, просто перетащив анимацию из папки Assets на рабочую область аниматора. Далее надо связать состояние покоя и саму анимацию, для этого надо нажать на **Empty State** и далее нажать Make Transition. Переходы к состоянию анимации и обратно должен происходить по булевой переменной **Bolean (Parameters**, тип **Bool)**, затем дать имя переменной**)**, причем обратно - по условию установки режима **Has Exit Time**. При этом после создания анимации необходимо *снять зацикливание* в контроллере анимации. Затем, нажав на переход анимации от состояния покоя к состоянию движения (Это стрелочки «туда-обратно») необходимо добавить созданную переменную в Conditions и установить состояние true. Для перехода анимации опять в состояние покоя нужно также добавить переменную, но с состоянием false и снять галочку **Has exit time**.

Код перехода к запуску анимации по нажатию клавиши

В окне инспектора **Inspector** добавляем **Add Component** для персонажа новый скрипт **New Script** и переходим в редактор кода с шаблоном заготовки.

run - имя булевой переменной в аниматоре

Добавляем в шаблон следующий код:

```
Animator anim;  //переменная типа Animator для ссылки на
анимацию
void Start(){
      anim = GetComponent<Animator>();   //контроллера анимации
}
void Update()
{
      If(Input.GetKeyDown(KeyCode.Q))   //если нажата клавиша q
      {
      anim.SetBool("run", true);       // переменная, отвечающая
за переход имеет значение true
      }
      If(Input.GetKeyDown(KeyCode.W))       //если нажата клавиша
w отпускается
      {
      anim.SetBool("run", false);      // переменная, отвечающая
за переход имеет значение false
```

Правила создания таблицы результатов эксперимента на симуляторе установки

Структурно таблица в unity представляет из себя **массив текстовых полей**, в которые записываются результаты работы с симулятором. Ячейки в таблице могут заполняться как вручную, так и автоматически, вычисляя значения по формулам. В незаполненных ячейках необходимо использовать символ минуса или подчёркивания чтобы было проще ориентироваться в таблице.

Для работы с таблицей в интерфейсе должны присутствовать следующие кнопки:

- кнопка для **записи** значения в таблицу
- кнопка для **отображения** таблицы
- кнопка для **очистки** таблицы

Также должно присутствовать **текстовое поле** для ввода значений. Запись в таблицу значения из этого поля производится по нажатию кнопки “Записать”.

Для записи значений в ячейки таблицы используется, например, для текстового объекта **name** конструкция **name.text**, а для вычислений с записями в текстовых объектах используется конструкция **int.Parse(name.text).**

Программа занесения данных в таблицу наблюдений и проведения расчетов с данными в таблице

```
using UnityEngine;
using UnityEngine.UI;

public class TableVal : MonoBehaviour
{
    [SerializeField]
    InputField textInput;
    [SerializeField]
    Text u1;
    [SerializeField]
    Text u2;
    [SerializeField]
    Text i1;
    [SerializeField]
    Text i2;
    [SerializeField]
    Text v1;
    [SerializeField]
    Text v2;
    [SerializeField]
    Text ve;

// Функция для кнопки «Записать»
public void WriteValue()
{
     if (u1.text == "-")
           u1.text = textInput.text;
    else if (i1.text == "-")
    {
           i1.text = textInput.text;
   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
```

```
();
 ve.text = (int.Parse (v1.text) + int.Parse (v2.text)).ToString();

    }
    textInput.text="";
}
// Функция для кнопки «Очистить»
    public void Clean()
    {
        u1.text = "-";
        u2.text = "-";
        i1.text = "-";
        i2.text = "-";
        v1.text = "-";
        v2.text = "-";
    }
}
```