**СОЗДАНИЕ ИНТЕРФЕЙСА ДЛЯ РАБОТЫ СИМУЛЯТОРА И ПРОГРАММИРОВАНИЕ АНИМАЦИЙ**

**АЛГОРИТМ РАЗРАБОТКИ ИНТЕРФЕЙСА ДЛЯ РАБОТЫ С УСТАНОВКОЙ**

Создать на панели **Panel** списки кнопок для работы с установкой: кнопкой-заголовком «**ПРАКТИКА**» и ниже кнопками для всех этапов выполнения работы с установкой.

Создать всплывающее по щелчку на кнопке «**ПРАКТИКА**» окно внизу сцены со структурой: слева информационное текстовое окно для вывода подсказок о выполняемых этапов работы, а справа добавить на него текстовое окно типа «Поле ввода», а также кнопки «**Запись**», «**Таблица**» и «**Очистка**», подчинив их всех в иерархии всплывающему текстовому окну.

Создать на основе объектов **UI** таблицу из объекта **Image** соответствующего размера и подходящей текстурой фоновое изображение для таблицы и наложенных на него (и подчиненных в иерархии объекту **Image)** ячеек таблицы из объектов **Text** с соответствующей структурой их относительного размещения для записи в эти ячейки результатов работы с установкой.

Добавить на объект **Image** скрипт для обработки таблицы с программным кодом для записи задаваемых значений параметров установки и наблюдаемых в результате значений, а также занесении в таблицу при необходимости вычисляемых результатов поведения установки.

Связать кнопки «**Запись**», «**Таблица**» и «**Очистка**» с соответствующими функциями в скрипте для обработки совершаемых с ними событий, обеспечивающих их функционаьность.

**СОЗДАНИЕ И ЗАПОЛНЕНИЕ ТАБЛИЦЫ РЕЗУЛЬТАТОВ РАБОТЫ**

Вся таблица – заголовки, названия полей и т.п. строится из текстовых объектов UI соответствующего размера с фиксированным текстом, а для ячеек, куда нужно записывать результаты эксперимента, используется символ подчеркивания или минуса, чтобы было проще находить нужные ячейки таблицы при записи в них значений по нажатию кнопки «Записать».

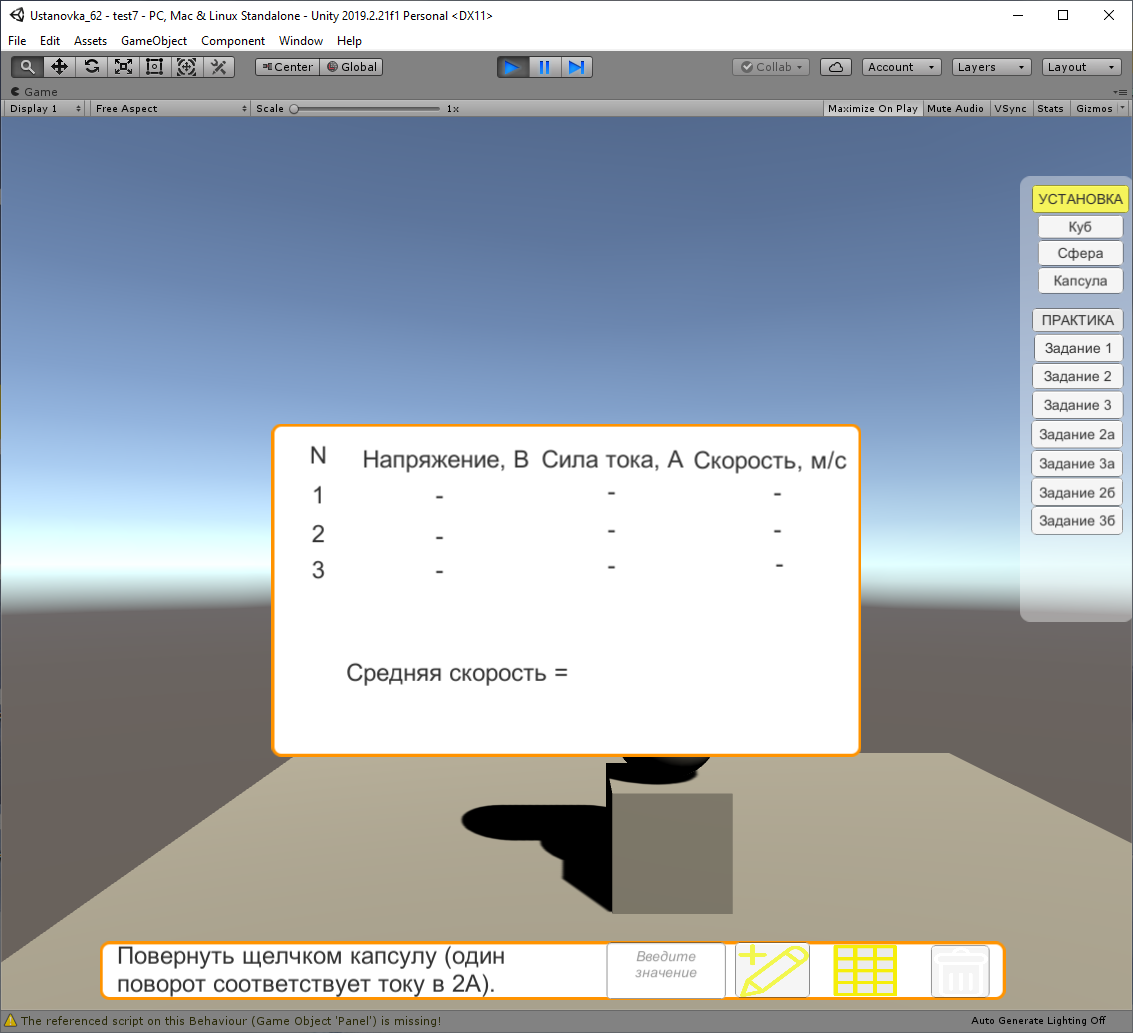
Для записи в текстовые объекты (ячейки таблицы) значений используется, например, для текстового объекта с именем **name1** конструкция

**name1.text.ToString()**,

а для вычислений, использующих записи в текстовых объектах, например, с именем **name1** используется конструкция

**int.Parse(name1.text).**

ПРИМЕРНАЯ СТРУКТУРА ТАБЛИЦЫ



**ПРИМЕР КОДА ЗАПОЛНЕНИЯ И ОЧИСТКИ ТАБЛИЦЫ РЕЗУЛЬТАТОВ РАБОТЫ**

**using System.Collections;  
using System.Collections.Generic;  
using UnityEngine;  
using UnityEngine.UI;  
  
public class TableVal : MonoBehaviour  
{  
//определение используемых переменных**

**[SerializeField]  
    InputField textInput;  
  
    [SerializeField]  
    Text u1;  
    [SerializeField]  
    Text i1;  
    [SerializeField]  
    Text u2;  
    [SerializeField]  
    Text i2;  
    [SerializeField]  
    Text u3;  
    [SerializeField]  
    Text i3;  
    [SerializeField]  
    Text v1;  
    [SerializeField]  
    Text v2;  
    [SerializeField]  
    Text v3;   
    [SerializeField]  
    Text ve;**

**public void WriteValue() //функция для кнопки «Записать»  
    {  
        if (u1.text == "-")  
            u1.text = textInput.text.ToString ();  
        else if (i1.text == "-") {  
            i1.text = textInput.text.ToString ();  
            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 ();  
        } else if (u3.text == "-")  
            u3.text = textInput.text.ToString ();  
         else if (i3.text == "-") {  
            i3.text = textInput.text.ToString ();  
            v3.text = (int.Parse (u3.text) \* int.Parse (i3.text)).ToString ();  
            ve.text = ((int.Parse (v1.text) + int.Parse (v2.text)+ int.Parse (v3.text))/3).ToString ();  
        }  
        textInput.text="";  
    }  
//функция для кнопки «Очистить»  
    public void Clean()  
    {  
        u1.text = "-";  
        u2.text = "-";  
        u3.text = "-";  
        i1.text = "-";  
        i2.text = "-";  
        i3.text = "-";  
        v1.text = "-";  
        v2.text = "-";  
        v3.text = "-";  
    }  
  
}**

**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** *(После добавления на экран элемента* ***Canvas*** *этот объект добавляется автоматически!!!)*
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 {**

…

**}**

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

Например, для изменения цвета объекта на красный при щелчке по нему курсором мыши можно использовать следующий код:

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

**gameObject.GetComponent<Renderer>().material.color = new Color(1,0,0);**

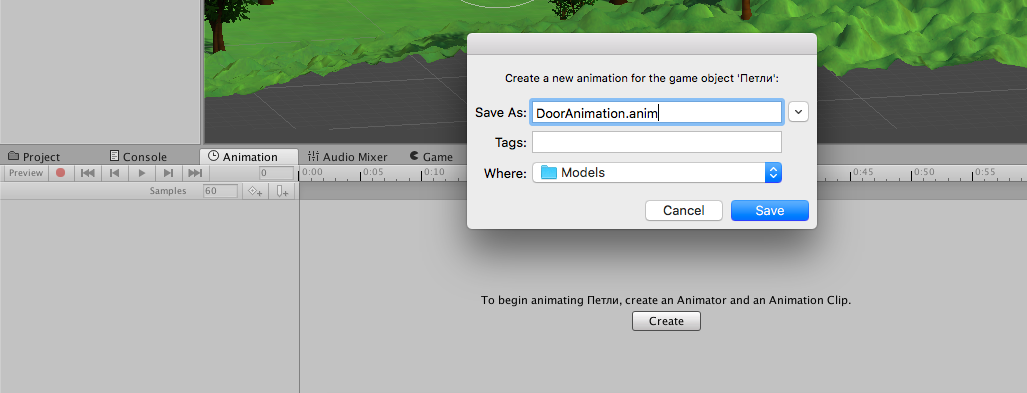
**}**

**СОЗДАНИЕ АНИМАЦИИ ОБЪЕКТА**

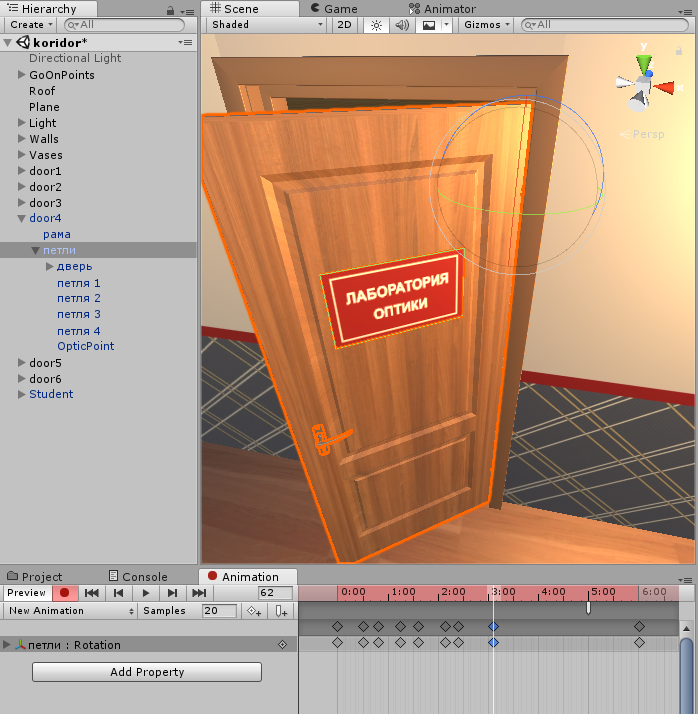
**В СРЕДЕ UNITY**

Для создания анимации в редакторе **Unity** нужно выбрать объект, который будет анимироваться, открыть для него вкладку **Window/Animation/Animation** и создать, прежде всего, отдельный файл анимации в открывающемся окне, нажав кнопку **Create**.

После сохранения файла анимации с типом **\*.anim** в папке проекта **Assets** создается в дальнейшем содержание анимации объекта.



На временной шкале в нижней части окна редактора **Unity** нажать кнопку с красным кружком (на рисунке) и в соответствующих ключевых кадрах записать требуемую анимацию для объекта.

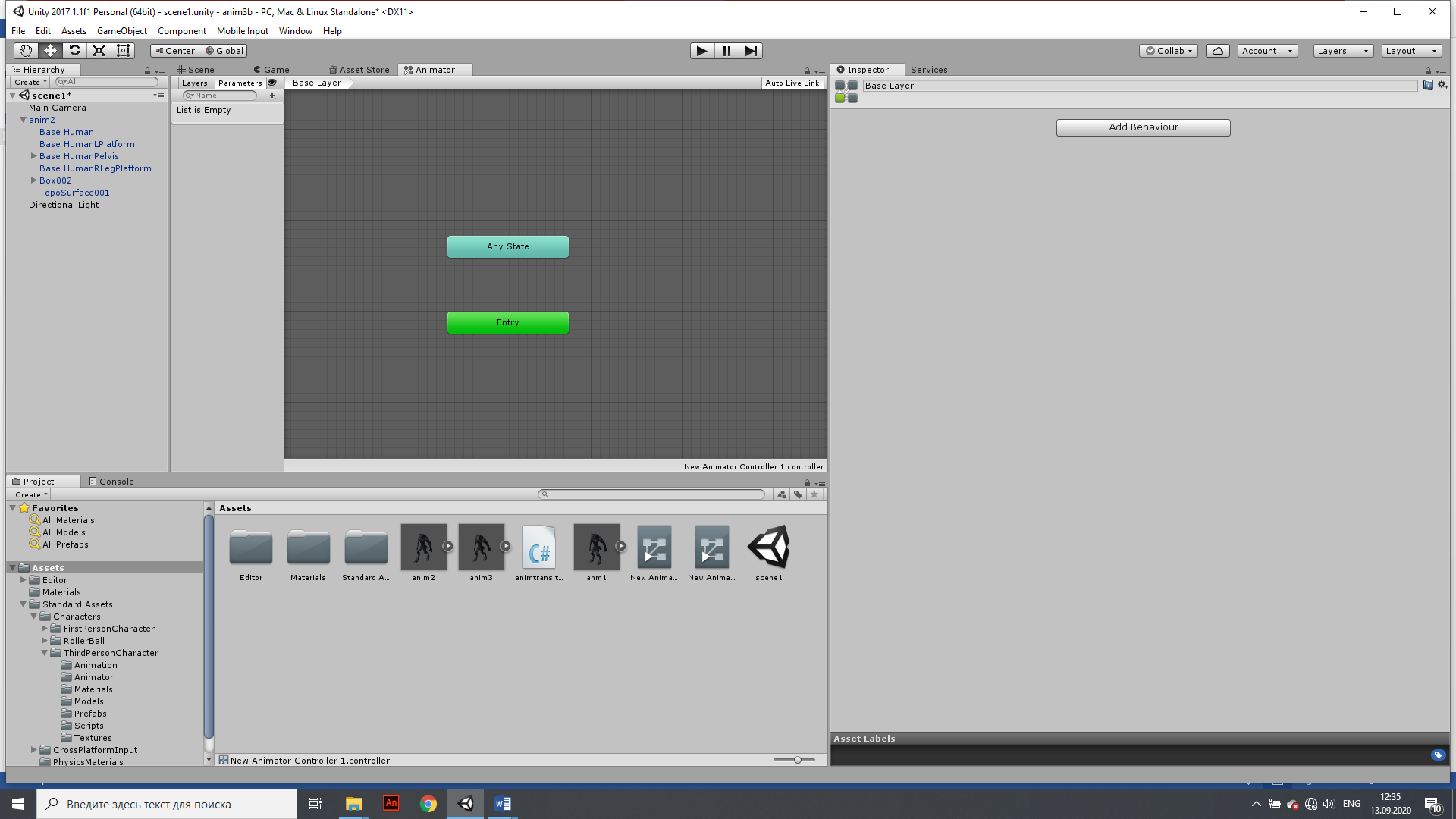


*При создании анимации необходимо учесть, что, если объект создан в среде* ***3dsMAX,*** *то при его анимируемом повороте, например открытии дверки духового шкафа, точка центра вращения* ***Pivot*** *по умолчанию будет расположена по центру. Чтобы объект вращался правильно вокруг края дверки, нужно в* ***Unity*** *создать пустой объект типа* ***Empty Object*** *(невидимый при проигрывании) и в окне* ***Hierarchy*** *сделать анимируемую дверку дочерним элементом этого пустого объекта, при этом разместить предполагаемую точку вращения в центре пустого объекта.*

**СОЗДАНИЕ КОНТРОЛЛЕРА АНИМАЦИИ**

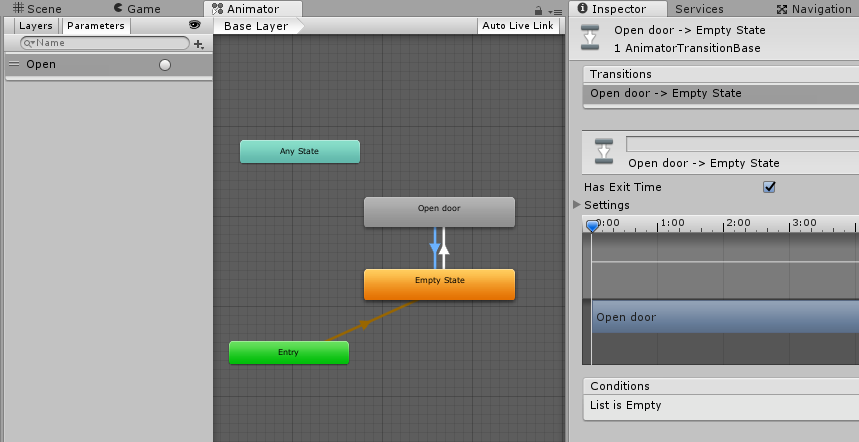
Затем нужно добавить к объекту, контроллер анимации **Animator** – кликнуть правой кнопкой мыши по свободной области в папке **Assets**, выполнить команду **Create->Animator Controller** и дать имя контроллеру. Дважды кликнув на созданный контроллер, перейти в окно редактора.

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.



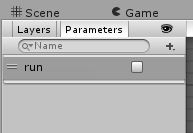
Создать **Empty State** как состояние по умолчанию, щелкнув правой кнопкой мыши в свободной части окна аниматора и выполнив команду **Create State/Empty,** в качестве состояния покоя у двери и добавить созданную анимацию отдельным состоянием.

Переходы к состоянию анимации открытия двери и обратно должны 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.

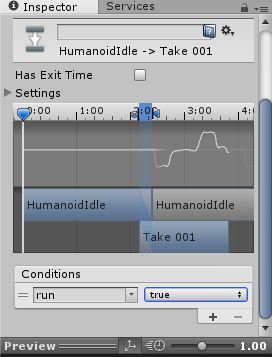


**УПРАВЛЕНИЕ АНИМАЦИЕЙ В UNITY НА ЯЗЫКЕ C#**

Для программного управления анимацией непосредственно в окне редактора необходимо создать программный код на языке C# и, прежде всего, в окне аниматора **создать переменную** открыввкладку **Parameters,** нажать плюс (добавить), выбрать тип создаваемой переменной, затем дать имя переменной, например **run**.



После этого выбрать в окне редактора щелчком мыши переход от анимации ожидания к анимации движения. Затем в Инспекторе для перехода в списке **Conditions** и добавляем в него нашу переменную **run** со значением **true**, нажав кнопку **+**.



Нажать правой кнопкой мыши на анимацию ходьбы и выбрать **Make transition**, чтобы создать обратную связь на анимацию по умолчанию. Выбрать новый переход и опять снять галочку **Has exit time**. Затем в список **Conditions** добавляем нашу переменную **run**, но со значением **false**.

**ДОБАВИТЬ СКРИПТ ДЛЯ ПЕРЕКЛЮЧЕНИЯ МЕЖДУ АНИМАЦИЯМИ**

В окне инспектора **Inspector** добавить **Add Component** для объекта новый скрипт **New Script** и в редакторе кода создатьследующий код:

**Animator anim;** //переменная типа **Animator** для ссылки на анимацию

**void Start(){**

**anim = GetComponent<Animator>();** //инициализация 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**

**}**

**ПРОИГРАТЬ АНИМАЦИЮ ИЗ ЛЮБОГО СОСТОЯНИЯ**

Чтобы проиграть какую-либо анимацию можно было из любого состояния, необходимо:

1. создать переход от **Any State** к этой анимации, например с именем **Hit**. Для возвращения в состояние ожидания необходимо создать переход от новой анимации **Hit** к анимации по умолчанию.
2. создать еще одну переменную типа **Trigger** (например **hitten**), которая будет отвечать за щелчок мышью для запуска анимации. Добавим к переходу от **Any State** эту переменную, а для обратного перехода оставим галочку **Has Exit Time**, чтобы переход совершался автоматически (без использования переменной) один раз после проигрывания анимации.
3. добавить в метод **Update()** код проигрывания новой пользовательской анимации по клику мыши:

**if (Input.GetMouseButtonDown(0))  {  anim.SetTrigger("hitten"); }**