**РАЗРАБОТКА ФУНКЦИОНАЛА, ЗАПИСЬ РЕЗУЛЬТАТОВ РАБОТЫ**

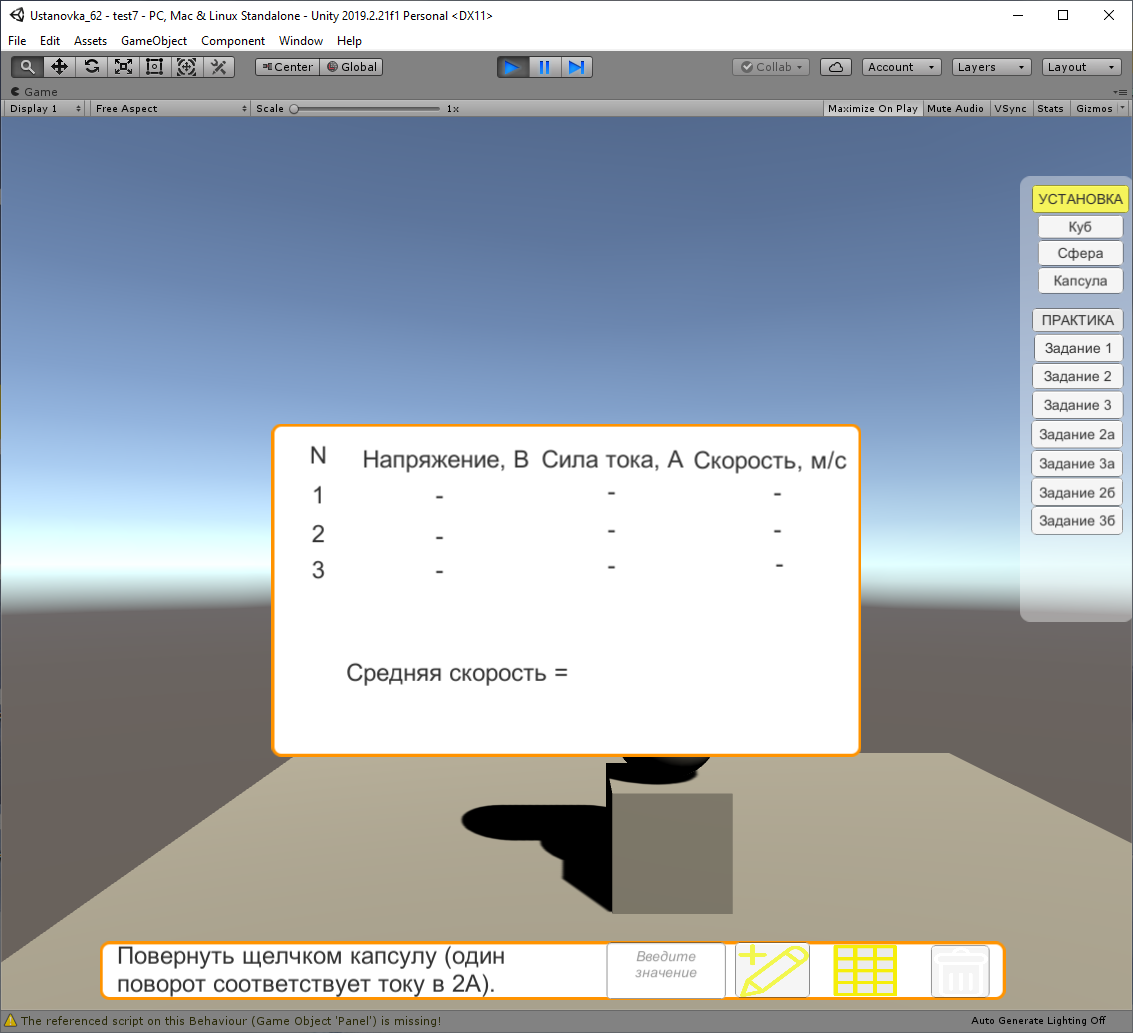
**И**

**ПУБЛИКАЦИЯ ПРИЛОЖЕНИЯ**

**СОЗДАНИЕ ПРОГРАММЫ ВЫПОЛНЕНИЯ ЭТАПОВ РАБОТЫ УСТАНОВКИ**

Для подключения функционала установки при работе с симулятором необходимо создать соответствующие **скрипты**, добавляемые на объект симулятора, с последовательным вызовом из него функций, выполняющихся при нажатии на кнопки выполнения этапов работы с симулятором.

При этом скрипт можно добавить на элемент интерфейса Table, к которому будут обращаться кнопки «ПРАКТИКА» для выполнения заданий, соответствующих этапам работы на симуляторе, выводя соответствующие сообщения в текстовое окно в нижней части экрана левее кнопок для работы с таблицей, как показано на рисунке:



Примерный код такого скрипта приведен ниже:

using System.Collections;  
using System.Collections.Generic;  
using UnityEngine;  
using UnityEngine.UI;   
  
public class Tasks : MonoBehaviour  
{  
    [SerializeField]   
    Text message1;   
    public void spher()   
    {message1.text = "Включить установку, щелкнув сферу"; }  
  
    [SerializeField]   
    Text message2;  
    public void cub()   
    {message2.text =

"Повернуть щелчком куб (один поворот соответствует напряже- нию в 20В). Затем занести заданное напряжение в таблицу и перейти к Заданию 3.";

}

...

}

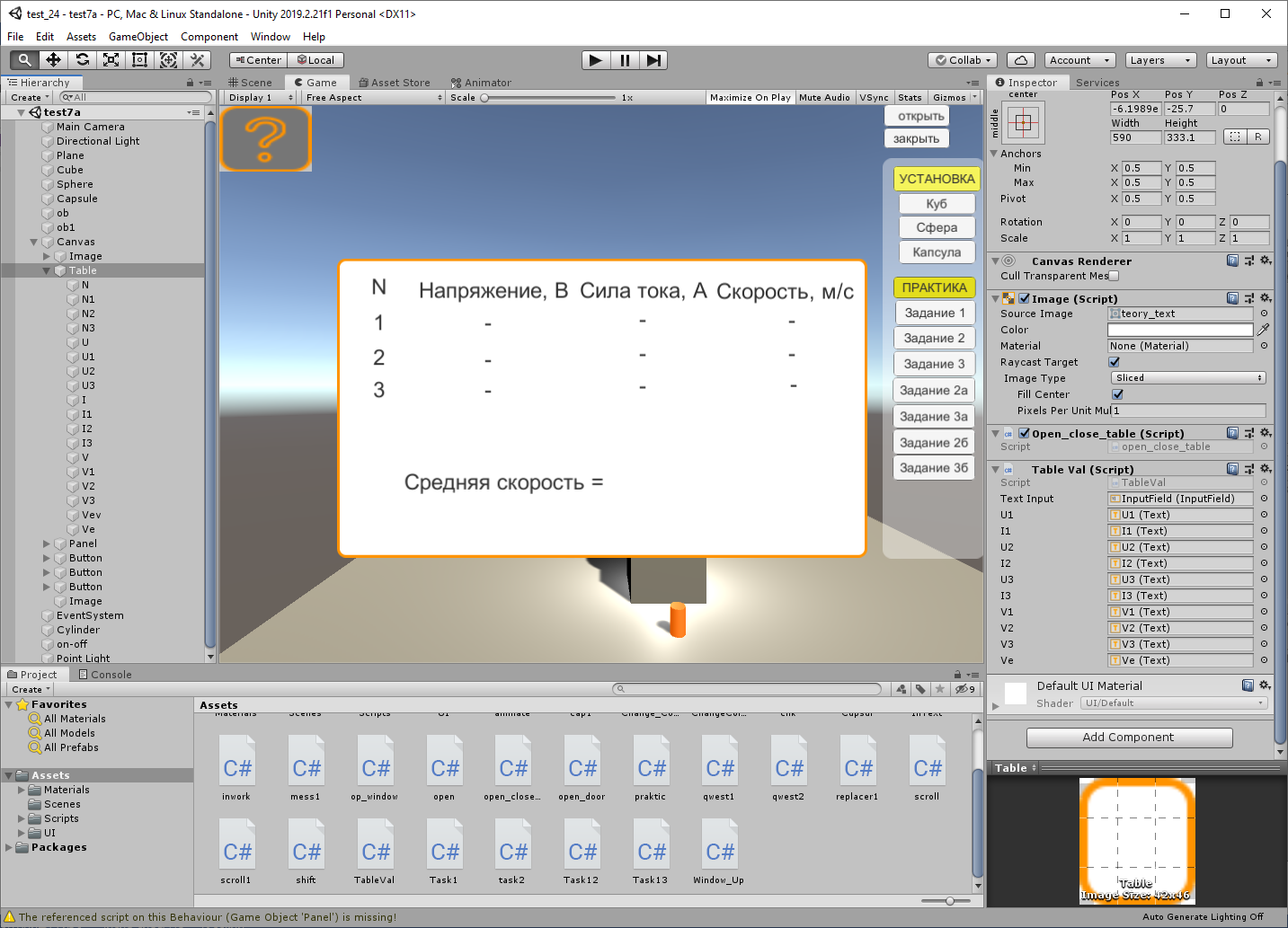
А для того, чтобы выполнить указанные в Задании действия с объектом симулятора на сцене, необходимо добавить скрипт на этот объект. При этом, если используется щелчок мышью по объекту, то необходимо в скрипте учесть правила обработки щелчков мышью с добавлением в скрипт директивы *using UnityEngine.EventSystems* и класса *IpointerClickHandler* к базовому классу *MonoBehaviour.* Например, для поворотов куба при щелчке по нему мышью код может выглядеть следующим образом:

using System.Collections;  
using System.Collections.Generic;  
using UnityEngine;  
using UnityEngine.EventSystems;  
  
public class task2 : MonoBehaviour, IPointerClickHandler  
{  
             
    public void OnPointerClick(PointerEventData eventData)  
    {           
        transform.Rotate (0, 10, 0);  
    }  
      
}

**ВЫВОД ЗНАЧЕНИЙ В ТАБЛИЦУ**

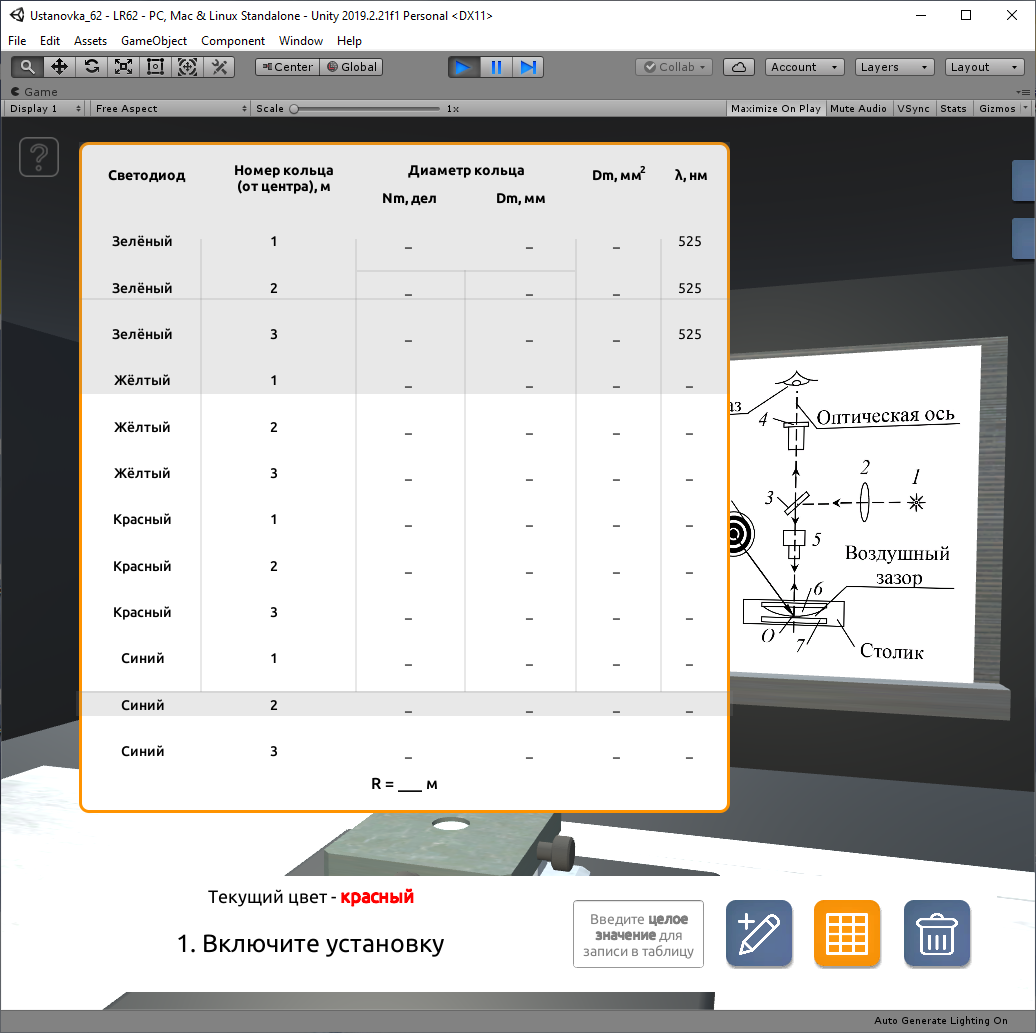
Для сохранения результатов работы с симулятором необходимо создать таблицу, в которую будут заносится результаты измерений показаний приборов установки или значений характеристик величин, получаемых при работе с установкой.

Таблица состоит из массива текстовых полей, в которые записываются результаты работы с симулятором.



Ячейки таблицы заполняются последовательно частично вручную, частично автоматически по формулам.

Для работы с таблицей на информационной панели для практики должны быть предусмотрены кнопки **Button:** кнопка для записи значения в таблицу, кнопка для отображения таблицы и кнопка для очистки таблицы (на рисунке ниже для них использованы соответствующие изображения).



Также на информационной панели должно присутствовать текстовое поле **InputText** для ввода полученных значений со шкалы прибора и занесения его в таблицу.

В ходе выполнения лабораторной работы снятые измерения с установки записываются в нужные ячейки таблицы по нажатию кнопки «Записать», для просмотра содержания таблицы используется событие наведение курсора на кнопку «Таблица», а для очистки таблицы от записей - наведение курсора на кнопку «Очистка».

После осуществления всех необходимых заданий из раздела «ПРАКТИКА» выводится соответствующее сообщение о завершении работы с симулятором установки.

**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

**U1.text.ToString()**,

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

**int.Parse(U1.text).**

Ниже приведен 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 = "-";  
    }  
  
}**

**ЭЛЕМЕНТЫ ИНТЕРФЕЙСА СИМУЛЯТОРА УСТАНОВКИ**

**(МИНИМУМ)**

**Обязательные объекты симулятора установки:**

1. Main Camera.
2. Directional Light и дополнительное освещение.
3. Cabinet (3 стены окружения с тематическими изображениями и стол-подставка для установки).
4. Ustanovka (все статичные и функциональные элементы установки с соответствующими текстурами).
5. Canvas с элементами UI.

**Обязательные скрипты:**

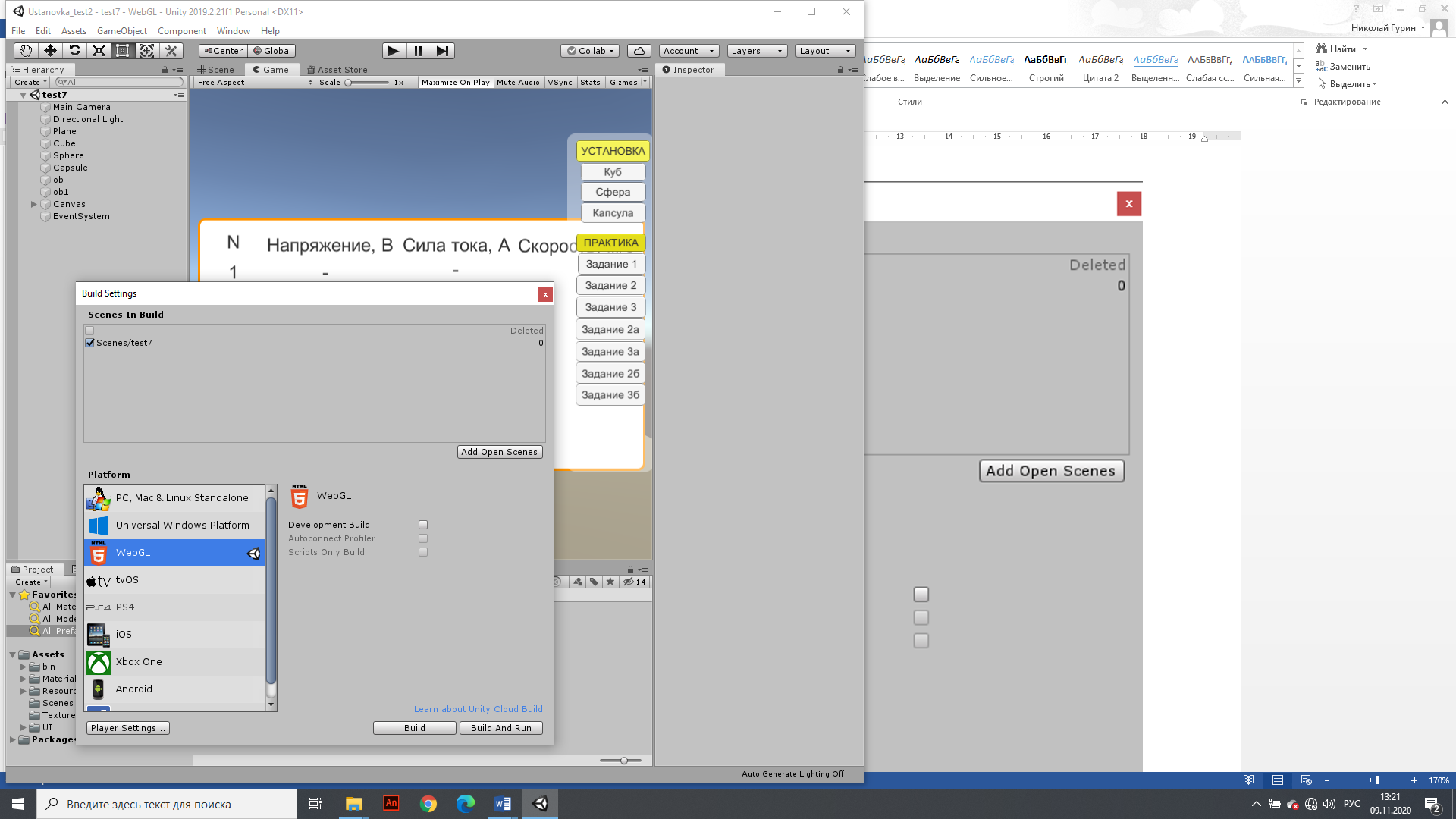
1. Управление камерой.
2. Общая справка-инструкция по работе с установкой.
3. Подсветка выбранного элемента установки.
4. Перемещение и поворот камеры к выбранному элементу установки.
5. Запуск внимации элемента установки
6. Движение панели управления (выдвинуть-задвинуть).
7. Система заданий (этапы работы с учетом функциональных зависимостей между элементами установки.
8. Ввод значений в таблицу результатов работы установки.

**Обязательные функции кнопок:**

1. Вывод подсказок в текстовые окна Text на фоне изображения Image (инструкция по использованию симулятора на полный экран; назначение элементов установки с их подсветкой, очередность выполнения заданий – в информационное окно внизу экрана).
2. Запись результатов работы в таблицу.
3. Показ таблицы.
4. Очистка таблицы.
5. Закончить работу с симулятором.

**ПРОЦЕДУРА ПУБЛИКАЦИИ ПРИЛОЖЕНИЯ UNITY**

1. При публикации обратить внимание, чтобы в путях к проекту, папки для публикации и к файлам **WebGL** не было пробелов и русских букв.
2. Чтобы в публикации отображался русский текст, нужно выбрать нужный шрифт (с расширением **ttf**), скопировать его в папку **Assets** проекта, а затем применить в качестве шрифта на компонент с текстом.
3. Для публикации проекта выполнить команду **File/Build Settings**. Далее выбрать платформу **WebGL** и нажать **Switch Platform** (можно предварительно задать более подробные настройки публикации в **Player Settings**).



1. Если в поле **Scene In Build** сразу не добавилась сцена, нужно нажать кнопку **Add Open Scenes**.
2. Нажать кнопку **Build** и дождаться окончания процедуры построения публикации проекта (возможно потребуется несколько десятков минут).
3. После завершения построения будут созданы необходимые папки и файл **index.html**. На этом этапе уже можно запустить проект в браузере (Google Chrome требует запуска с http сервера), поэтому можно воспользоваться, например, браузером **Firefox**.
4. Для того, чтобы встроить плеер с проектом в уже существующий **html**, достаточно в требуемое место в существующем html-документе скопировать и вставить фрагмент из только что созданного **index.html** (а также созданные при сборке проекта папки в папку сайта):

<**head>**

**<meta http-equiv="Content-Type" content="text/html; charset=utf-8">**

**<link rel="shortcut icon" href="TemplateData/favicon.ico">**

**<link rel="stylesheet" href="TemplateData/style.css">**

**<script src="TemplateData/UnityProgress.js"></script>**

**<script src="Build/UnityLoader.js"></script>**

**<script>**

**var unityInstance = UnityLoader.instantiate("unityContainer", "Build/student.json", {onProgress: UnityProgress});**

**</script>**

**</head>**

**<body>**

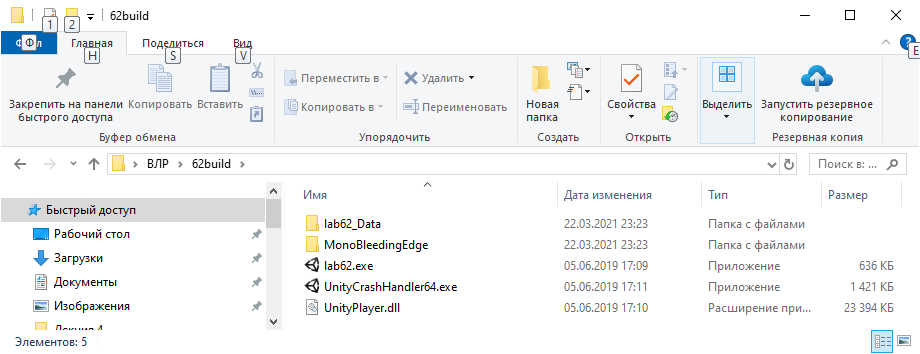
**<div class="webgl-content">**

**<div id="unityContainer" style="width: 960px; height: 600px"></div>**

**</div>**

**</body>**

1. Для публикации разработанного приложения, работающего независимо от браузера, необходимо при выполнении команды **File/Build Settings** (см. рисунок выше) выбрать режим **Universal Windows Platform**. После публикации проекта, например, с именем **lab62** средой **Unity** будет создана папка с необходимыми файлами и папками для запуска приложения **exe-файлом** с соответствующей структурой, как показано на рисунке ниже для проекта:



При этом для запуска приложения необходимо будет выполнить файл **lab62.exe**.