

Лабораторная работа №3: Редактор уровней.

Целью работы является получение навыков создания редакторов трёхмерных сцен.

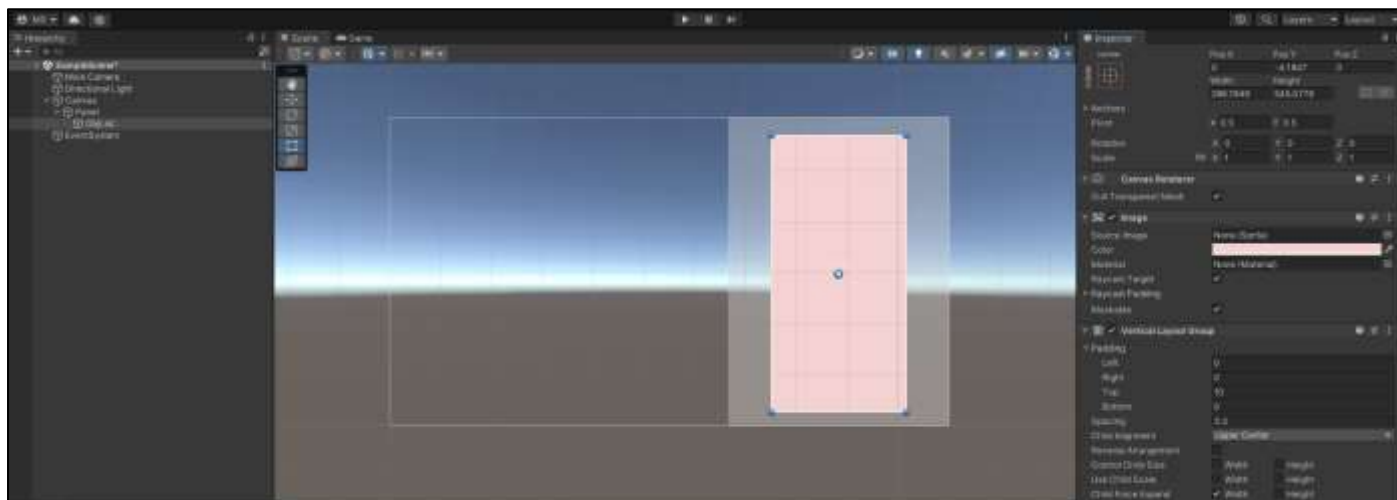
Оглавление

Создание интерфейса добавления объектов в сцену	1
Список объектов	4
Перемещение объектов.....	7
Сохранение и загрузка.....	11
Задания:.....	13
Отчёт.	13
Справка:	13

Создание интерфейса добавления объектов в сцену

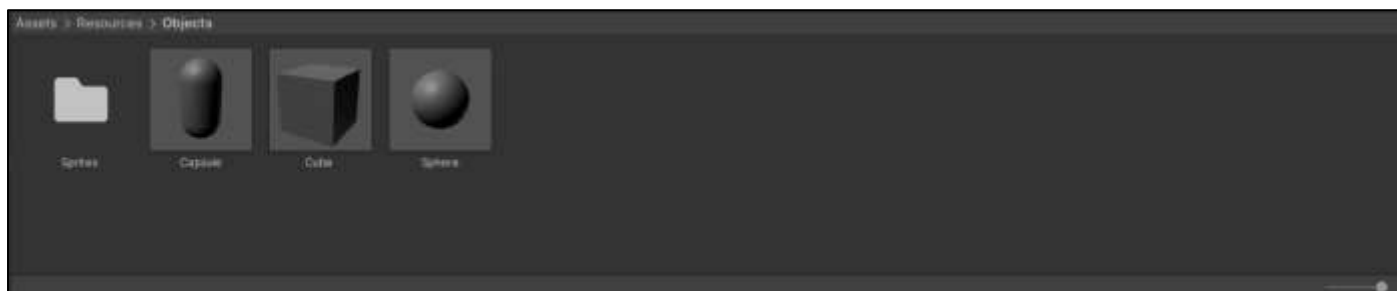
При работе над редакторами трёхмерных сцен, имеет смысл создавать интерфейсы, изменяющиеся автоматически в зависимости от числа шаблонов объектов, доступных в редакторе. Такой подход, позволяет не переделывать интерфейс полностью при добавлении новых шаблонов и категорий объектов.

Для создания одной категории, достаточно добавить в сцену панель, разместить на панели изображение и добавить к изображению компонент VerticalLayoutGroup:

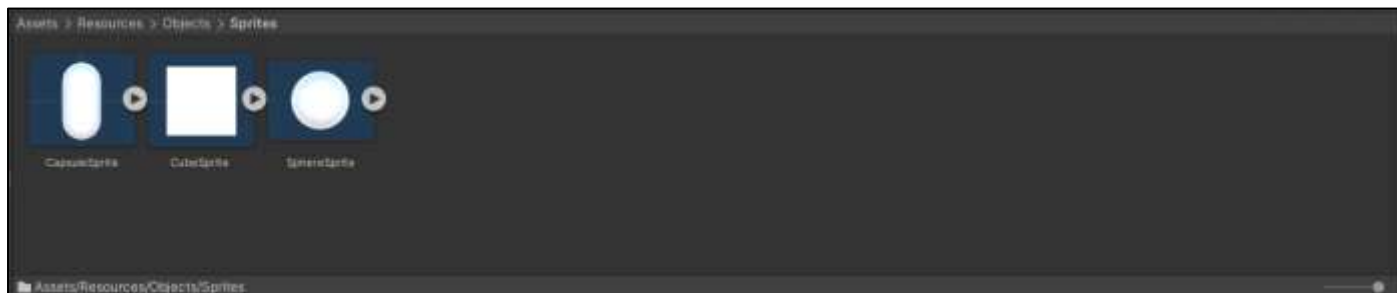


Это позволит добавлять к изображению кнопки, не задавая их позицию и смещение. Так же, можно добавить полосу прокрутки как это было сделано в лабораторной 1, в интерфейсе выбора уровня.

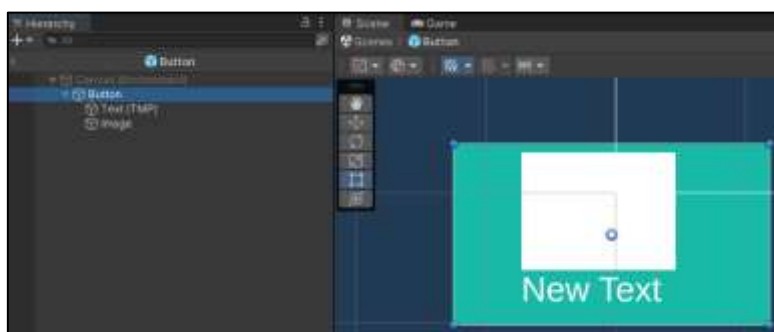
Количество кнопок в элементе интерфейса будет соответствовать количеству доступных шаблонов (prefabs) объектов. Шаблоны объектов, при этом, необходимо добавить в папку, расположенную в директории Resources:



Вместе с шаблонами объектов так же необходимо добавить спрайты, содержащие изображения этих объектов:



После создания области отображения, шаблонов и изображений объектов, необходимо создать шаблон кнопки следующего вида:



Где компонент Text будет служить для вывода названия объекта, а компонент Image — для вывода изображения этого объекта. Так же, необходимо создать и добавить к шаблону кнопки скрипт следующего вида:

```
using UnityEngine;
using UnityEngine.UI;
using TMPro;

Ссылка: 2
public class ButtonScript : MonoBehaviour
{
    public TMP_Text txt; // ссылка на текст внутри кнопки
    public Image img; // ссылка на изображение внутри кнопки

    ссылка:1
    public void setText(string text) // метод установки текста
    {
        txt.text = text;
    }

    ссылка:1
    public void setSprite(Sprite sprite) // метод установки картинки
    {
        img.sprite = sprite;
    }
}
```

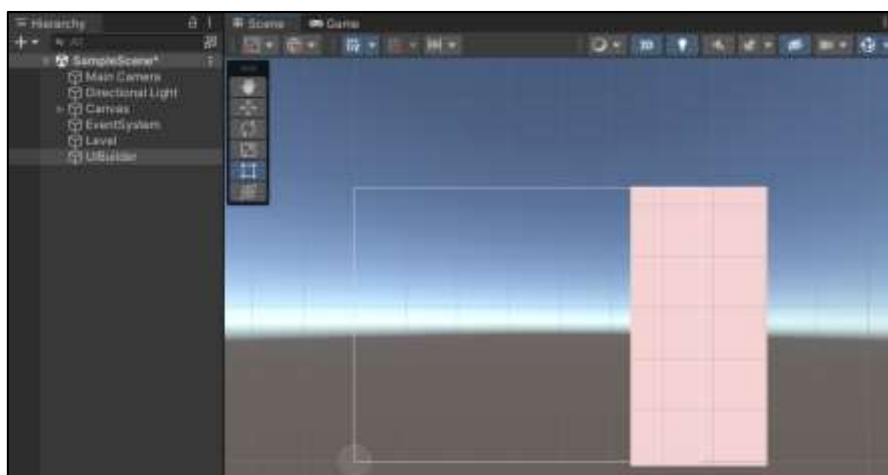
Для хранения загруженных шаблонов и списка объектов, добавленных в сцену, необходимо создать глобальный скрипт следующего вида:

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

Ссылка: 3
public class GlobalScript // глобальный класс, не привязанный ни к одному из объектов сцены
{
    public static Dictionary<string, GameObject> prefs = new Dictionary<string, GameObject>(); // глобальный список префабов

    public static List<GameObject> objects = new List<GameObject>(); // глобальный список объектов на уровне
}
```

Для автоматического добавления кнопок и хранения объектов сцены потребуются пустые объекты UIBuilder и Level:



При этом сам скрипт создания кнопок и добавления объектов может выглядеть следующий образом:

```
using UnityEngine;
using UnityEngine.UI;

[Scriptable]
public class PrefabLoader : MonoBehaviour
{
    public GameObject btn; // ссылка на шаблон кнопки
    public GameObject btnList; // ссылка на список кнопок
    public GameObject level; // ссылка на список объектов уровня

    [Scriptable]
    void Start()
    {
        GameObject[] folderObjects = Resources.LoadAll<GameObject>("Objects"); // получение объектов из директории Resources/Objects

        foreach (GameObject obj in folderObjects) // для каждого объекта
        {
            GlobalScript.prefs.Add(obj.name, obj); // добавление объекта в глобальный словарь

            GameObject t = Instantiate(btn); // создание новой кнопки

            t.transform.SetParent(btnList.transform); // добавление кнопки в список кнопок
            t.GetComponentInChildren<ButtonScript>().setText(obj.name); // установка текста кнопки
            t.GetComponent<Button>().onClick.AddListener(delegate { createObject(obj.name); }); // добавление обработчика нажатия на кнопку
        }

        Object[] textures = Resources.LoadAll("Objects/Sprites", typeof(Sprite)); // получение спрайтов из директории Resources/Objects/Sprites

        for (int j = 0; j < textures.Length; j++) // для каждого спрайта
        {
            Transform tr = btnList.transform.GetChild(j); // получение кнопки
            tr.gameObject.GetComponent<ButtonScript>().setSprite(textures[j] as Sprite); // установка спрайта в качестве картинки на кнопку
        }
    }

    [Scriptable]
    public void createObject(string name) // обработчик нажатия на кнопку добавления объекта
    {
        GameObject t = Instantiate(GlobalScript.prefs[name]); // создание объекта из префаба с соответствующим именем

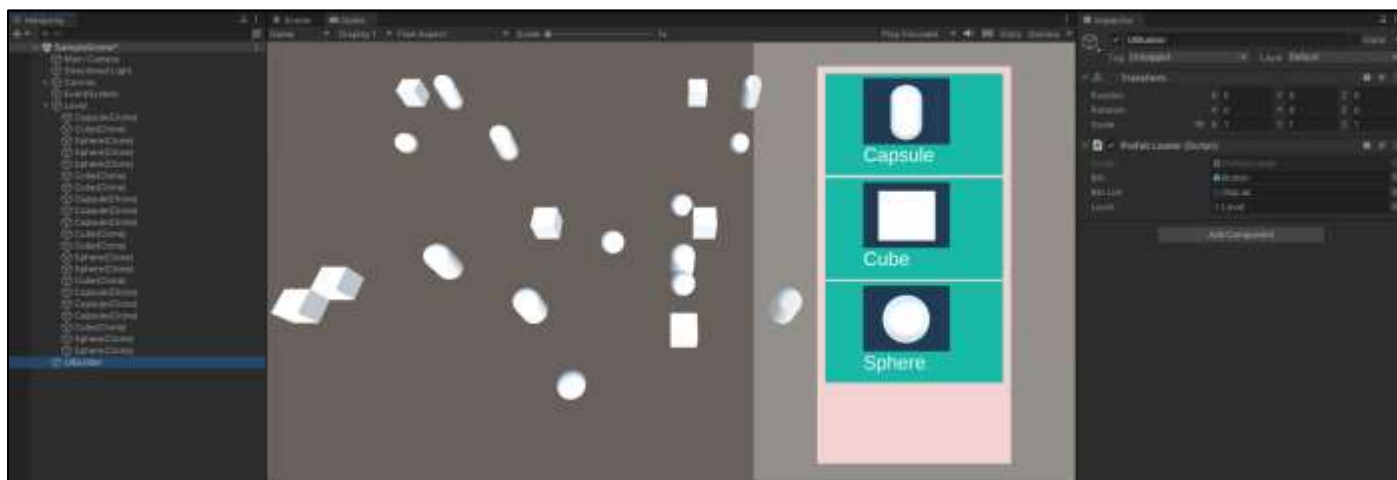
        t.transform.position = new Vector3(Random.Range(-10, 10), 0, Random.Range(-10, 10)); // размещение объекта в случайных координатах

        t.transform.SetParent(level.transform); // добавление объекта в объект уровня

        GlobalScript.objects.Add(t); // добавление объекта в глобальный список объектов
    }
}
```

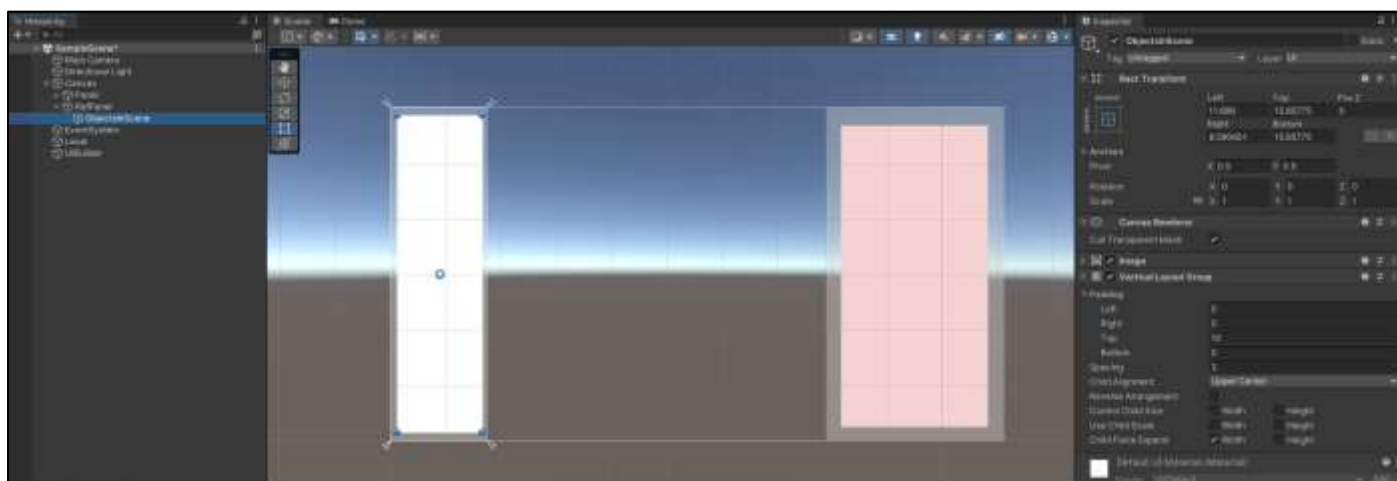
Примечание: Обратите внимание, что в скрипте отсутствуют проверки на некорректный ввод и предполагается, что изображения будут идти в том же порядке что и соответствующие им объекты. При желании, изображения и объекты можно объединить в шаблоны, считывать, как объекты, и “разбирать” на составляющие в скрипте. В примере такой подход не реализован для демонстрации способов загрузки различных объектов.

Если всё было сделано правильно, то при запуске приложения, на панели объектов, появится столько кнопок, сколько шаблонов объектов находится в соответствующей папке. При нажатии на кнопки, в случайной позиции будет появляться объект, соответствующий изображению и тексту на кнопке:



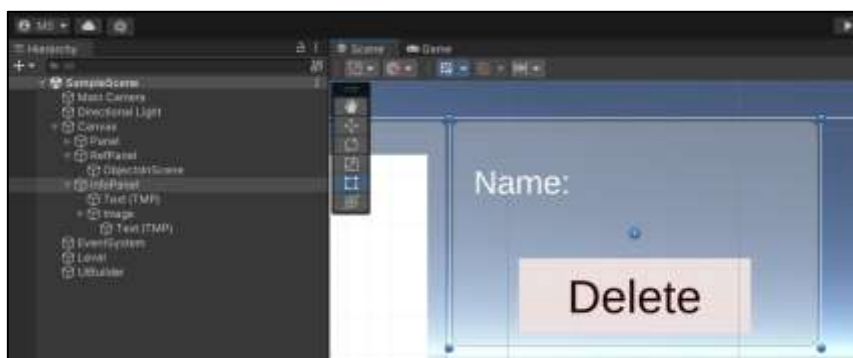
Список объектов

Следующим этапом разработки редактора будет создание списка объектов, добавленных в сцену. Сам по себе список будет выглядеть так же, как и список кнопок:



Для каждого объекта, добавленного в сцену, в этот список будет добавляться связанная с ним кнопка.

Помимо списка кнопок, так же понадобится информационная панель, позволяющая посмотреть информацию о выбранном объекте и совершить с ним какие-либо действия:



В приведённом примере, панель позволяет посмотреть имя выбранного объекта, а также удалить его.

Для реализации связей между кнопками и объектами, глобальный скрипт можно изменить следующим образом:

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

// Ссылка: 7
// класс, описывающий объект сцены
public class COBJect // класс, описывающий объект сцены
{
    public GameObject obj; // ссылка на объект в сцене
    public string name; // имя объекта
    public GameObject button = null; // ссылка на кнопку, связанную с объектом

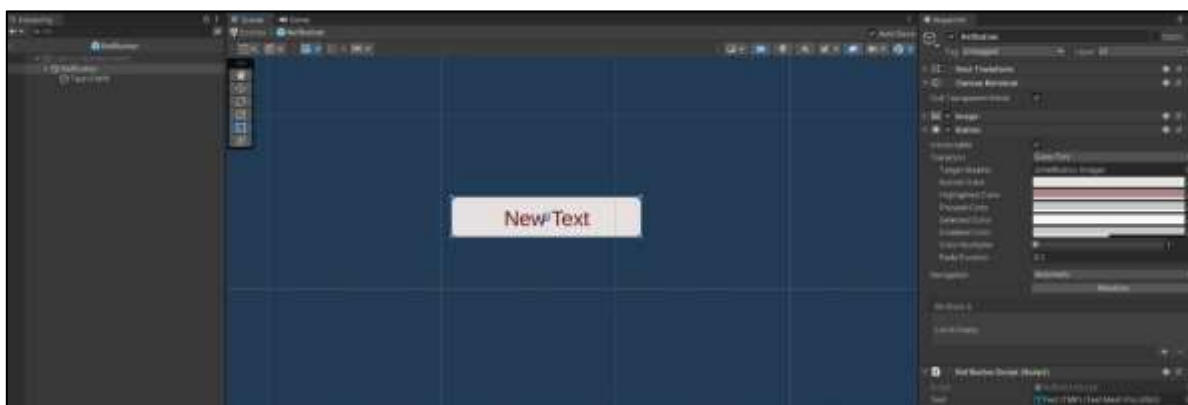
    // Ссылка: 1
    public COBJect(GameObject o, string n) // ссылка на объект и имя объекта устанавливается в конструкторе
    {
        name = n;
        obj = o;
    }

    // Ссылка: 1
    public void setButton(GameObject button) // функция установки ссылки на кнопку
    {
        this.button = button;
    }
}

// Ссылка: 1
public class Cursor // глобальный класс, содержащий ссылку на объект, выбранное в текущий момент
{
    public static COBJect objectReference = null;
}

// Ссылка: 8
public class GlobalScript // глобальный класс, не привязанный ни к одному из объектов сцены
{
    public static Dictionary<string, GameObject> prefs = new Dictionary<string, GameObject>(); // глобальный список префабов
    public static List<COBJect> objects = new List<COBJect>(); // глобальный список объектов в сцене
}
```

Шаблон кнопки, ассоциированной с объектом сцены будет содержать изображение с компонентом Button и текст:



Скрипт будет иметь метод установки текста кнопки:

```
using UnityEngine;
using TMPro;

// Ссылка: 1
public class RefButtonScript : MonoBehaviour
{
    public TMP_Text text; // ссылка на текст внутри кнопки

    // Ссылка: 1
    public void setText(string text) // метод установки текста
    {
        this.text.text = text;
    }
}
```

Скрипт информационной панели будет содержать метод установки имени выбранного объекта, а также, обработчик нажатия клавиши удаления:

```
using UnityEngine;
using UnityEngine.UI;
using TMPro;

[Scriptable]
public class InfoPanelScript : MonoBehaviour
{
    public TMP_Text text; // ссылка на текстовое поле инфо панели

    [ContextMenu]
    public void setText(string text) // метод установки текста
    {
        this.text.text = text;
    }

    [ContextMenu]
    public void delete() // метод, вызываемой при нажатии кнопки удаления
    {
        GlobalScript.objects.Remove(Cursor.objectReference); // исключение выбранного объекта из глобального списка объектов
        Destroy(Cursor.objectReference.obj); // уничтожение объекта в сцене
        Destroy(Cursor.objectReference.button); // уничтожение ассоциированной с объектом кнопки
        Cursor.objectReference = null; // обнуление ссылки на выбранный объект
        transform.gameObject.SetActive(false); // скрываем информационную панель
    }
}
```

Предполагается, что при нажатии кнопки, ссылка на ассоциированный с ней объект передаётся в объект Cursor. Имея ссылку на выбранный объект и ассоциированную с ним кнопку, остаётся удалить их из сцены и списка кнопок соответственно, а очистить Cursor.

Основные изменения будут содержаться в скрипте PrefabLoader. Во-первых, необходимо передать ссылки на новый шаблон кнопки и новые элементы интерфейса:

```
public GameObject btn; // ссылка на шаблон кнопки
public GameObject btnList; // ссылка на список кнопок
public GameObject level; // ссылка на список объектов уровня

public GameObject infoPanel; // ссылка на панель с информацией
public GameObject refBtn; // ссылка на шаблон кнопки, ссылающейся на объект в сцене
public GameObject refBtnList; // ссылка на список кнопок, ссылающихся на объекты в сцене
```

Во-вторых, изменится метод создания объекта:

```
public void createObject(string name) // обработчик нажатия на кнопку добавления объекта
{
    GameObject t = Instantiate(GlobalScript.prefs[name]); // создание объекта из префаба с соответствующим именем

    t.transform.position = new Vector3(Random.Range(-10, 10), 0, Random.Range(-10, 10)); // размещение объекта в случайных координатах

    t.transform.SetParent(level.transform); // добавление объекта в объект уровень

    CObject o = new CObject(t, t.name); // создание объекта, содержащего ссылку на объект в сцене и его имя
    GlobalScript.objects.Add(o); // добавление объекта в глобальный список объектов

    GameObject tRefBtn = Instantiate(refBtn); // создание кнопки, которая будет связана с добавленным объектом
    tRefBtn.GetComponent<RefButtonScript>().setText(o.name); // установка имени объекта в текст кнопки
    tRefBtn.GetComponent<Button>().onClick.AddListener(delegate { selectObject(o); }); // добавление обработчика нажатия на кнопку
    tRefBtn.transform.SetParent(refBtnList.transform); // добавление кнопки в соответствующий список
    o.setButton(tRefBtn); // добавление ссылки на кнопку в объект
}
```

Помимо изменения формата хранения объекта, добавится создание ассоциированной с ним кнопки и назначение ей обработчика нажатия.

Обработчик нажатия на кнопку будет активировать информационную панель и устанавливать ссылку на ассоциированный с кнопкой объект в Cursor:

```
public void selectObject(CObject o) // обработка нажатия на кнопку, связанную с объектом в сцене
{
    infoPanel.SetActive(true); // активация информационной панели
    infoPanel.GetComponent<InfoPanelScript>().setText("Name: " + o.name); // передача в информационную панель имени объекта, связанного с нажатой кнопкой
    Cursor.objectReference = o; // установка объекта ассоциированного с кнопкой в качестве "выбранного"
}
```


Если всё было сделано правильно, то при добавлении объектов в сцену, в список слева будут добавляться ассоциированные с ними кнопки. А при нажатии на кнопку, будет появляться информационная панель, позволяющая удалить ассоциированный с нажатой кнопкой объект:



Перемещение объектов

Следующим этапом развития редактора, будет добавление возможности выбора и перемещения объектов при помощи курсора мыши. При выборе объектов мышью должна появляться информационная панель, а сам объект должен выделяться. При пересечении с другими объектами сцены, перемещаемый объект и объекты, с которыми произошло пересечение должны отмечаться красным цветом.

Для реализации описанного выше функционала, необходимо добавить скрипт объекта, содержащий блок параметров:

```
public class ObjectScript : MonoBehaviour
{
    Object objectRef = null;    // ссылка на объект, внутри которого лежит модель

    public Material normal;    // материал по умолчанию
    public Material selected;  // материал, отображаемый когда модель выбрана
    public Material collision;  // материал, отображаемый когда модель пересекается с другими моделями в сцене

    bool selection = false;    // переменная, определяющая, выбрана ли модель в данный момент

    Renderer ren;              // ссылка на MeshRenderer
}
```

и набор методов для работы с ними:

```
Ссылка: 0
private void Start()
{
    ren = GetComponent<Renderer>();
}

Ссылка: 1
public void select() // метод отвечающий за выделение модели
{
    ren.material = selected;
    selection = true;
}

Ссылка: 1
public void deSelect() // метод, отвечающий за сброс выделения
{
    ren.material = normal;
    selection = false;
}

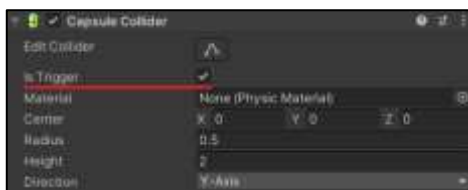
Ссылка: 1
public void setObjectRef(CObject o) // метод, устанавливающий ссылку на объект, содержащий модель
{
    objectRef = o;
}

Ссылка: 3
public CObject getObjectRef() // метод, возвращающий ссылку
{
    return objectRef;
}

Ссылка: 0
private void OnTriggerEnter(Collider other) // метод, срабатывающий при пересечении моделей в сцене
{
    if (other.transform.CompareTag("model"))
        ren.material = this.collision;
}

Ссылка: 0
private void OnTriggerExit(Collider other) // метод, срабатывающий в момент прекращения пересечений в сцене
{
    if (selection == true)
        ren.material = selected;
    else
        ren.material = normal;
}
```

В данном случае, выделение объекта будет происходить при помощи смены его материала. При желании, можно реализовать этот функционал при помощи компонента LineRenderer, через управление прозрачностью базового набора материалов и множеством других способов. Далее следует поместить шаблоны объектов на отдельный слой, отметить isTrigger в их Collider, прикрепить к ним скрипт и компонент Rigidbody:



Поскольку модель теперь может быть выбрана, можно добавить новый функционал в класс Cursor:

```
public class Cursor // глобальный класс, содержащий ссылку на объект, выбранный в текущий момент
{
    public static CObject objectReference = null; // ссылка на выбранный объект
    public static bool move = false; // переменная, определяющая, происходит ли перемещение выбранного объекта

    // ссылка: 3
    public static void select(CObject o) // метод выделения объекта
    {
        objectReference = o;
        objectReference.obj.GetComponent<ObjectScript>().select();
    }

    // ссылка: 2
    public static void deSelect() // метод сброса выделения с объекта
    {
        objectReference.obj.GetComponent<ObjectScript>().deSelect();
        objectReference = null;
    }

    // ссылка: 1
    public static void setPosition(Vector3 position) // метод, изменяющий позицию выбранного объекта
    {
        objectReference.obj.transform.position = position;
    }
}
```

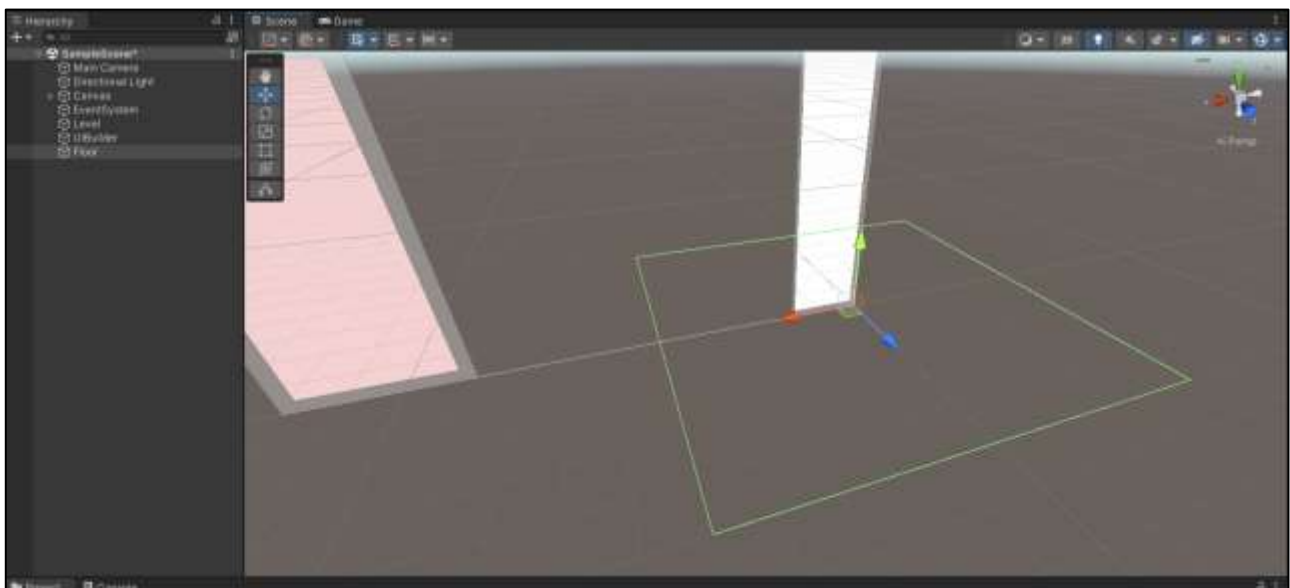
Так же, изменения произойдут в скрипте загрузки шаблонов:

```
public void createObject(string name) // обработчик нажатия на кнопку добавления объекта
{
    GameObject t = Instantiate(GlobalScript.prefs[name]); // создание объекта из префаба с соответствующим именем
    t.transform.position = new Vector3(Random.Range(-10, 10), 0, Random.Range(-10, 10)); // размещение объекта в случайных координатах
    t.transform.SetParent(level.transform); // добавление объекта в объект уровня
    CObject o = new CObject(t, t.name); // создание объекта, содержащего ссылку на объект в сцене и его имя
    GlobalScript.objects.Add(o); // добавление объекта в глобальный список объектов
    t.GetComponent<ObjectScript>().setObjectRef(o); // передача ссылки на объект в модель

    GameObject tRefBtn = Instantiate(refBtn); // создание кнопки, которая будет связана с добавленным объектом
    tRefBtn.GetComponent<RefButtonScript>().setText(o.name); // установка имени объекта в текст кнопки
    tRefBtn.GetComponent<Button>().onClick.AddListener(delegate { selectObject(o); }); // добавление обработчика нажатия на кнопку
    tRefBtn.transform.SetParent(refBtnList.transform); // добавление кнопки в соответствующий список
    o.setButton(tRefBtn); // добавление ссылки на кнопку в объект
}

// ссылка: 1
public void selectObject(CObject o) // обработка нажатия на кнопку, связанную с объектом в сцене
{
    infoPanel.SetActive(true); // активация информационной панели
    infoPanel.GetComponent<InfoPanelScript>().setText("Name: " + o.name); // передача в информационную панель имени объекта, связанного с нажатой кнопкой
    Cursor.select(o); // установка объекта ассоциированного с кнопкой в качестве "выбранного"
}
```

Для удобства позиционирования объектов, предлагается добавить в сцену пустой объект, с добавленным к нему BoxCollider. Объект должен находится на отдельном слое, перекрывать рабочую область сцены и будет служить для позиционирования объектов по осям X и Z:



В завершении, необходимо создать и прикрепить к камере объект Selector, содержащий поля:

```
using UnityEngine.EventSystems;

Скрипт 0
public class Selector : MonoBehaviour
{
    public Camera cam;           // ссылка на камеру
    public LayerMask floor;      // слой рабочей области сцены
    public LayerMask objects;    // слой, содержащий объекты, доступные для выбора и перемещения
    public GameObject infoPanel; // ссылка на информационную панель
}
```

и метод, отвечающий за проверку пересечений:

```
void LateUpdate()
{
    Ray ray = cam.ScreenPointToRay(Input.mousePosition);
    RaycastHit hit;

    if (Input.GetMouseButtonDown(0))
    {
        if (EventSystem.current.IsPointerOverGameObject()) // проверка, попал ли клик в элемент интерфейса,
        { return; } // если да - выход

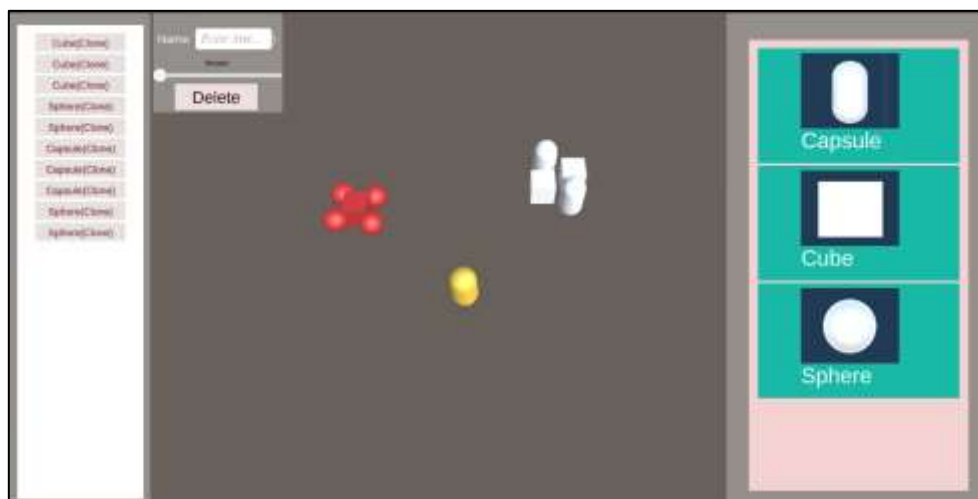
        if (Physics.Raycast(ray, out hit, 1000, objects))
        {
            if (Cursor.objectReference == null) // если в текущий момент объект не выбран
            {
                Cursor.select(hit.transform.GetComponent<ObjectScript>().getObjectRef()); // выбор объекта, с которым произошло пересечение
                infoPanel.SetActive(true); // активация информационной панели
                infoPanel.GetComponent<InfoPanelScript>().setText("Name: " + hit.transform.name); // установка имени выбранного объекта
            }
            else if (hit.transform.GetComponent<ObjectScript>().getObjectRef() != Cursor.objectReference) // если пересечение произошло с объектом,
            { // не выбранным в текущий момент
                Cursor.deSelect(); // сброс выделения с текущего объекта

                Cursor.select(hit.transform.GetComponent<ObjectScript>().getObjectRef()); // выделение нового объекта
                infoPanel.GetComponent<InfoPanelScript>().setText("Name: " + hit.transform.name);
            }
            else
            {
                Cursor.move = true; // пока зажата левая кнопка мыши - выбранный
            } // объект перемещается
        }
        else
        {
            if (Cursor.objectReference != null) // если клик мыши прошёл мимо моделей в сцене
            {
                Cursor.deSelect(); // сброс выделения с текущей выбранной модели
                infoPanel.SetActive(false); // скрывает информационную панель
            }
        }
    }

    if (Input.GetMouseButtonUp(0))
    {
        Cursor.move = false; // отмена перемещения выбранной модели
    } // при отжатии кнопки мыши

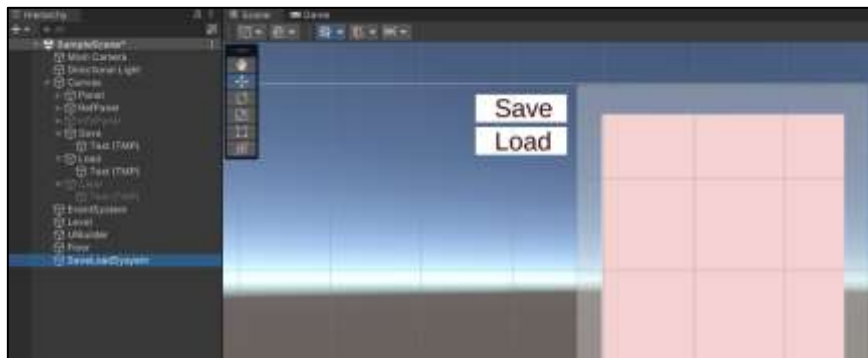
    if (Cursor.move == true) // если включён режим перемещения
    {
        if (Physics.Raycast(ray, out hit, 1000, floor)) // если найдено пересечение с рабочей областью
        {
            if (Cursor.objectReference != null) // если в данный момент выбрана модель
            { Cursor.setPosition(hit.point); } // переместить модель в точку рабочей области
        }
    }
}
```

Результат демонстрирующий выбор и пересечения объектов:



Сохранение и загрузка

Заключительным этапом работы над редактором трёхмерных сцен является создание системы сохранения и загрузки. Для реализации механизма сохранения и загрузки, имеет смысл добавить в сцену объект, который будет содержать соответствующий скрипт, а также две кнопки:



Затем следует описать формат данных, который будет описывать параметры объектов сцены:

```
using System.Collections.Generic;
using System.IO;
using UnityEditor;
using UnityEngine;

[System.Serializable]           // отметка о том, что класс может быть сериализован
// Ссылка: 6
public class CSaveData          // класс, содержащий описание данных объекта сцены, которые будут сохраняться
{
    public string name;          // имя объекта
    public string type;          // тип объекта

    public Vector3 position;      // позиция объекта
    public Quaternion rotation;   // поворот объекта
    public Vector3 scale;         // масштаб объекта
}

// Ссылка: 4
public class CSaveList // список данных которые необходимо сохранить (можно добавить любые данные о сцене)
{
    public List<CSaveData> list = new List<CSaveData>();
}
```

Обратите внимание – структура, которая будет отвечать за сохранение, должна быть отмечена как Serializable.

Сам класс, описывающий систему сохранения/загрузки будет содержать ссылку на скрипт, отвечающий за создание объектов в сцене:

```
public class SaveLoadSystem : MonoBehaviour
{
    public GameObject prefabLoader; // ссылка на объект, содержащий скрипт создания объектов сцены
    PrefabLoader pl;                // ссылка на скрипт создания объектов сцены
}
```

А метод сохранения может выглядеть следующим образом:

```
public void saveScene() // метод сохранения сцены
{
    CSaveList list = new CSaveList(); // создание объекта, содержащего параметры объектов сцены
    string json = "";

    foreach (CObject o in GlobalScript.objects) // для каждого объекта в сцене
    {
        CSaveData cfo = new CSaveData(); // создание структуры описывающей параметры объекта
        cfo.name = o.name;
        cfo.type = o.type;
        cfo.position = o.obj.transform.position;
        cfo.rotation = o.obj.transform.rotation;
        cfo.scale = o.obj.transform.localScale;
        list.list.Add(cfo); // добавление структуры в список
    }

    json = JsonUtility.ToJson(list); // преобразование объекта в формат JSON
    var path = EditorUtility.SaveFilePanel("Save scene as JSON", "", "scene" + ".json", "json"); // диалог сохранения файла
    File.WriteAllText(path, json); // запись JSON строки в файл
}
```

Для того, чтобы реализовать метод загрузки сцены из файла, можно модифицировать уже имеющийся метод добавления объектов в сцену. Во-первых, появится разделение понятий типа и имени объекта, во-вторых, второй параметр, отвечающий за наличие сохранённых данных:

```
public void createObject(string type, CSaveData sd = null) // обработчик нажатия на кнопку добавления объекта
{
    GameObject t = Instantiate(GlobalScript.prefs[type]); // создание объекта из префаба с соответствующим именем
    t.transform.SetParent(level.transform); // добавление объекта в объект уровень
    CObject o = new CObject(t, t.name); // создание объекта, содержащего ссылку на объект в сцене и его имя
    GlobalScript.objects.Add(o); // добавление объекта в глобальный список объектов
    o.type = type; // сохранение типа объекта

    if (sd == null) // если отсутствуют сохранённые данные
    {
        t.transform.position = new Vector3(Random.Range(-10, 10), 0, Random.Range(-10, 10)); // размещение объекта в случайных координатах
    }
    else // если объект создаётся по сохранённым данным
    {
        t.transform.position = sd.position;
        t.transform.rotation = sd.rotation;
        t.transform.localScale = sd.scale;
        o.name = sd.name;
    }

    t.GetComponent<ObjectScript>().setObjectRef(o); // передача ссылки на объект в модель

    GameObject tRefBtn = Instantiate(refBtn); // создание кнопки, которая будет связана с добавленным объектом
    tRefBtn.GetComponent<RefButtonScript>().setText(o.name); // установка имени объекта в текст кнопки
    tRefBtn.GetComponent<Button>().onClick.AddListener(delegate { selectObject(o); }); // добавление обработчика нажатия на кнопку
    tRefBtn.transform.SetParent(refBtnList.transform); // добавление кнопки в соответствующий список
    o.setButton(tRefBtn); // добавление ссылки на кнопку в объект
}
```

Кроме того, необходимо добавить метод для очистки сцены:

```
public void clearLevel() // метод очистки сцены
{
    infoPanel.SetActive(false); // отключение информационной панели, на случай если она была активна
    Cursor.objectReference = null; // обнуление ссылки на выбранный объект, на случай если был выбран объект

    foreach (CObject o in GlobalScript.objects) // для каждого объекта в сцене
    {
        Destroy(o.obj); // уничтожение объекта в сцене
        Destroy(o.button); // уничтожение ассоциированной с объектом кнопки
    }
    GlobalScript.objects.Clear(); // очистка списка объектов
}
```

При наличии описанных выше методов, загрузку данных из файла сохранения можно реализовать следующим способом:

```
public void loadScene() // метод загрузки сцены
{
    string path = EditorUtility.OpenFilePanel("Load scene from json", "", "json"); // диалог открытия файла
    string json = File.ReadAllText(path); // получение JSON строки

    CSaveList list = JsonUtility.FromJson<CSaveList>(json); // преобразование JSON строки в класс, содержащий список параметров объектов

    pl.clearLevel(); // очистка сцены

    foreach (CSaveData sd in list.list) // добавление в сцену объектов из загруженного списка
    {
        pl.createObject(sd.type, sd);
    }
}
```

Задания:

1. Создать 2 или более панелей объектов. Осуществить переключение между панелями при помощи элемента интерфейса Dropdown.
2. Добавить на информационную панель возможность переименовать выбранный объект.
3. Добавить на информационную панель возможность вращать выбранный объект.
4. Добавить возможность перемещать выбранные объекты по вертикали.
5. Реализовать следующий функционал: при создании нового объекта, он перемещается вслед за курсором мыши до первого нажатия левой кнопки мыши.

Отчёт.

В качестве отчёта вы должны предоставить документ, содержащий следующие пункты:

1. Цель и задачи лабораторной работы.
2. Краткое описание действий, которые вы выполняли для решения поставленных в лабораторной работе заданий с картинками, представляющими работу приложения.
3. Краткий вывод о проделанной вами работе
4. Ссылка на облачный диск/github с архивом в котором находится **написанные вами скрипты**.

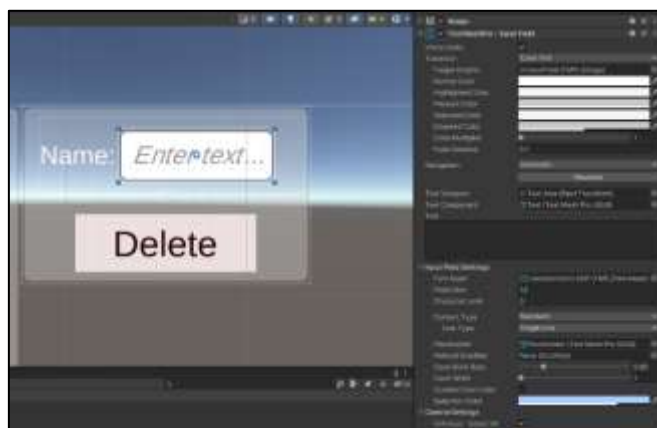
Справка:

Компонент Dropdown содержит список вариантов выбора (Options) и событие “Изменение значения”, параметром которого является номер выбранной опции:



Реализовать **задание 1**, можно, например, создав несколько панелей, заполняемых кнопками на основе шаблонов объектов из разных папок, а затем переключая их свойство active при смене значения компонента Dropdown.

Для выполнения **задания 2**, может быть использован компонент InputField:



Для выполнения задания 3, можно использовать компонент Slider:

