Санкт-Петербургский государственный морской технический университет

Факультет морского приборостроения

Кафедра систем автоматического управления и

бортовой вычислительной техники

Отчёт по теме:

Объектно-ориентированное программирование

По дисциплине:

«Программирование на языках высокого уровня»

Выполнил:

Николенко Михаил Андреевич

Студент группы 3270

Проверил:

Сакович Сергей Юрьевич

Санкт-Петербург 2020 г.

Оглавление:

Введение

В рамках курсовой работы, которую мы выполняли в команде, состоящей из трёх человек, нам удалось создать прототип игры, в которой игрок управляет небольшой подводной лодкой и сам составляет алгоритм её движения. Каждый из нас выполнял определённую задачу и в процессе реализации мы соединяли наши наработки в один общий продукт, дорабатывали, исправляли ошибки. В рамках курсовой работы каждый из нас опишет всё, что было сделано лично автором той же курсовой работы. Все элементы, которые создаются в пару кликов или имеют сравнительно небольшую реализацию будут описаны очень кратко.

Участники команды:

1. Агринский Никита Андреевич - основная задача: логика движения подводной лодки, создание 3d моделей.
2. Газизов Айнур Айдарович - основная задача: интерактивная панель логической схемы для управления подводной лодкой.
3. Николенко Михаил Андреевич - основная задача: конструктор игрового пространства и загрузка сохранённого пространства на другой сцене.

Общая цель:

Создать прототип небольшой игры. Основной игры должна стать морская тематика, а именно: подводная лодка, как главный объект игры, различные препятствия, такие как мины, элементы естественного ландшафта.
Так-же нужно дать игроку возможность управления подводной лодкой, используя Интерактивную Панель Составления Логической Схемы. И последняя задача - редактор карты, по которой игрок будет передвигаться. Для выполнения поставленных задач, нами было принято решение использовать Unity — межплатформенная среда разработки компьютерных игр на C#.

Моя роль в проекте:

Моя главная задача в данном проекте - создать удобный конструктор карт и создать инструменты для дальнейшего взаимодействия с ним других участников.

Конструктор игрового пространства

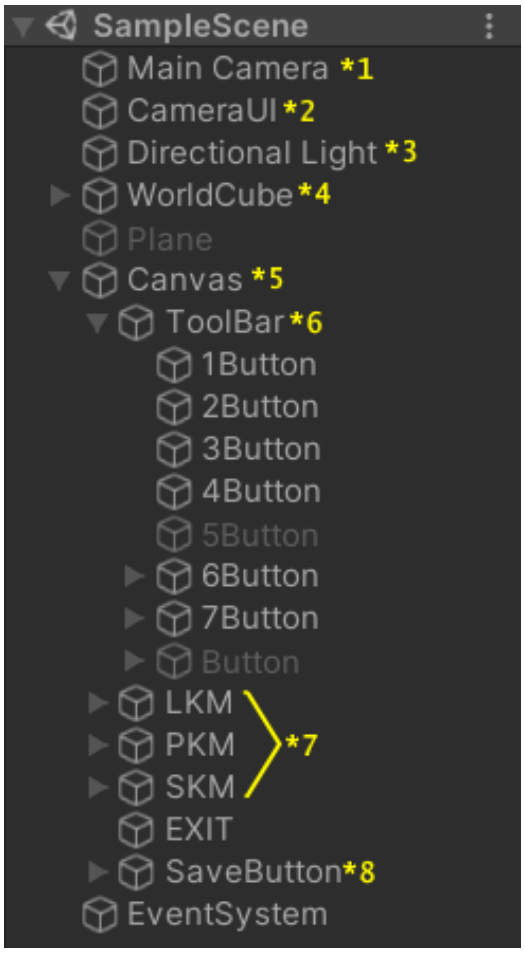
Раздел 1. Функции будущей программы.

Для комфортного использования конструктора игрового пространства конечным пользователем и для дальнейших манипуляций с картой, нужно реализовать следующие пункты:

1. Игровое пространство будет представлено в виде кубов и записано в трёхмерный массив. В каждый элемент этого массива записываем тип объекта, находящегося в этом кубе(лодка, мина и тд).
2. Ограничение игрового пространства посредством выбора игроком границ: длина, ширина, глубина.
3. Пользовательский интерфейс для выбора объектов и размещения их на игровом пространстве.
4. Сохранение построенного пространства в текстовый файл.

Раздел 2. Сцена редактирование игрового пространства.

Для начала создаём сцену. По умолчанию на сцене присутствует только: Main Camera, Directional Light и EventSystem. Для выполнения поставленной задачи нужно создать ещё ряд объектов, и итого на сцене будут следующие объекты:



1. Main Camera - главная камера, которой пользователь сможет управлять.

2. CameraUI - вторичная камера, благодаря которой мы можем видеть объекты пользовательского интерфейса(далее UI).

3. Directional Light - освещение на сцене.

4. WorldCube - главный объект на сцене. Тут будет находится скрипт WorldLogic.cs

5. Canvas - обязательный элемент UI. На нём находятся все элементы UI, такие как текст, кнопки, картинки.

6. ToolBar - панель объектов для размещения и кнопки: режим без объекта и режим удаления объектов.

7. LKM, PKM, SKM - подсказки для пользователя.

8. SaveButton - кнопка сохранение со скриптом SaveButton.cs

Раздел 3. WorldLogic.cs

В скрипте WorldLogic.cs реализовано редактирование пользователем игрового пространства и запись этого пространства в трёхмерный массив.

Текст скрипта с комментариями

```
using UnityEngine;
using UnityEngine.EventSystems;

public class WorldLogic : MonoBehaviour
{
    public GameObject submarine, mine, chainLink, fixOnTheGround, lighthouse; // 3d модели

    // Материалы для 3d моделей
    public Material generalMaterial, invisibleMaterial, sandMaterial;
    public Mesh generalMesh;
    public static Material materialForMiniCube;

    private GameObject model3D; // используемая в данный момент 3d модель
    private byte modelID; // индификационный номер(далее ID) используемой 3d модели

    private static int step = MapSizeEditor.step; // step - размер одной клетки. Пр. 1 кл = 10 у.е

    public Camera camera; // камера на сцене, через которую смотрит игрок. Нужна для использования Raycast

    private GameObject newMiniCube; // временный объект, созданный игроком при нажатии на экран

    GameObject[,,] TypeOfObjectOnMap; // массив с объектами для удобного удаления их с сцены
    public static byte[,,] TypeOfObjectOnMapInt; // массив с ID объектов

    private byte MainX,MainY,MainZ; // Координаты главной лодки
    public static byte rotation;
    private byte EndX,EndY,EndZ; // Координаты конечного пункта пути

    private float mouseScrollValue; // для редактирования высоты с помощью колёсика мышки
```

```
    // Временные переменные, чтобы не создавать их каждый frame
    private float positionX, positionY, positionZ;
    private float differenceX, differenceY, differenceZ;
    int xCoord, yCoord, zCoord;
    int tempxCoord = 0, tempzCoord = 0;

    bool Boolfas, deleteMode;
    WorldLogic()
    {
        // инициализируем массивы с размерами, указанными пользователем
        TypeOfObjectOnMap = new GameObject[MapSizeEditor.countX, MapSizeEditor.countY, MapSizeEditor.countZ];
        TypeOfObjectOnMapInt = new byte[MapSizeEditor.countX, MapSizeEditor.countY, MapSizeEditor.countZ];
    }
    public void IsTapped(int num) // вызывается при нажатии на панель (рис.1)
    {
        if (newMiniCube == null)
        {
            modelID = (byte)num;
            switch (num)
            {
                case 1:
                    model3D = submarine;
                    deleteMode = false;
                    break;
                case 2:
                    model3D = mine;
                    deleteMode = false;
                    break;
                case 3:
                    model3D = lighthouse;
                    deleteMode = false;
                    break;
                case 6:
                    model3D = null;
                    deleteMode = false;
                    break;
                case 7:
                    deleteMode = true;
                    break;
                case 9:
                    model3D = null;
                    deleteMode = false;
                    break;
                default:
                    deleteMode = false;
                    model3D = null;
                    break;
            }
        }
    }

    private void Start()
    {
        gameObject.transform.localScale = new Vector3(MapSizeEditor.countX * step, MapSizeEditor.countY * step,
MapSizeEditor.countZ * step);

        for(int x = 0; x < MapSizeEditor.countX; x++)
        {
            for (int y = 0; y<1; y++)
            {
                for (int z = 0; z < MapSizeEditor.countZ; z++) // создаём самый нижний слой кубов, который нельзя
будет удалять
                {
                    xCoord = (int)(step * (x));
                    yCoord = (int)(step * (y));
                    zCoord = (int)(step * (z));
                    newMiniCube = new GameObject(); // создаём новый объект
                    newMiniCube.transform.position = new Vector3(xCoord, yCoord, zCoord); // устанавливаем
местоположение
                    newMiniCube.transform.localScale = new Vector3(step, step, step); // устанавливаем размер
                    newMiniCube.transform.parent = transform; // делаем дочерним объектом Объекта, на котором
висит данный скрипт
                    newMiniCube.AddComponent<MeshCollider>().sharedMesh = generalMesh; // создаём коллайдер,
чтобы объект имел границы
```

```
                    newMiniCube.AddComponent<MeshFilter>().mesh = generalMesh;  // задаём объект как куб
                    newMiniCube.AddComponent<MeshRenderer>().material = generalMaterial;  // вешаем на куб материал

                    TypeOfObjectOnMap[x, y, z] = newMiniCube; // записываем объект в массив с объектами
                    TypeOfObjectOnMapInt[x, y, z] = 9; // записываем тип объекта для дальшейшего сохранения в файл и для других манипуляций
                    newMiniCube.tag = "ground"; // присваиваем тег для того, чтобы в дальнейшем самый нижний слой нельзя было удалить
                }
            }
        }
        // обнуляем значения
        newMiniCube = null;
        xCoord = 0;
        yCoord = 0;
        zCoord = 0;
        materialForMiniCube = sandMaterial;
    }

    /// <summary> Метод, отвечающий за инициализацию
    /// <see cref="UnityEngine.GameObject"/>
    ///  на сцене
    ///  <para> Является методом класса <seealso cref="WorldLogic"/></para>
    /// </summary>
    void SpawnerControl(GameObject model3D)
    {
        if (Input.GetMouseButtonDown(0)) // вызывается, когда игрок ставит объект с помощью ЛКМ
        {
            if (newMiniCube == null)
            {
                if (model3D != null || modelID == 9)
                {
                    newMiniCube = new GameObject(); // создаём новый объект
                    newMiniCube.transform.localScale = new Vector3(step + 2, step + 2 , step + 2); // задаём размер объекта чуть больше, чем все объекты, чтобы визуально было удобнее его размещать
                    newMiniCube.AddComponent<MeshFilter>().mesh = generalMesh; // задаём объект как куб

                    if (modelID == 9) newMiniCube.AddComponent<MeshRenderer>().material = generalMaterial; // вешаем на куб материал
                    else Instantiate(model3D, newMiniCube.transform); // загружаем модельку в объект,

                    if (model3D == mine) newMiniCube.tag = "mine";
                }
            }
            else
            {
                if (xCoord >= 0 && xCoord < MapSizeEditor.countX) // проверяем допустимость координат
                {
                    if (yCoord >= 0 && yCoord < MapSizeEditor.countY) // проверяем допустимость координат
                    {
                        if (zCoord >= 0 && zCoord < MapSizeEditor.countZ) // проверяем допустимость координат
                        {
                            if (TypeOfObjectOnMap[xCoord, yCoord, zCoord] == null) // если на данных координатах нет объекта, то размещаем его
                            {
                                newMiniCube.transform.localScale = new Vector3(step , step , step );
                                newMiniCube.transform.parent = transform;
                                newMiniCube.AddComponent<BoxCollider>().center = new Vector3(0, 0, 0);
                                TypeOfObjectOnMap[xCoord, yCoord, zCoord] = newMiniCube;
                                TypeOfObjectOnMapInt[xCoord, yCoord, zCoord] = modelID;

                                if (modelID == 1)
                                {
                                    if (!(MainX == 0 && MainY == 0 && MainZ == 0)) // удаляем повторения главной лодки
                                    {
                                        Destroy(TypeOfObjectOnMap[MainX, MainY, MainZ]);
                                        TypeOfObjectOnMapInt[MainX, MainY, MainZ] = 0;
                                    }
                                    MainX = (byte)xCoord; MainY = (byte)yCoord; MainZ = (byte)zCoord;
                                    rotation = (byte)((newMiniCube.transform.localEulerAngles.y)/90); // сохраняем поворот лодки (от 0 до 3)
                                }
                                else if (modelID == 3)
```

```
                                {
                                    if (!(EndX == 0 && EndY == 0 && EndZ == 0)) // удаляем повторения маячка с
конечной точкой маршрута
                                    {
                                        Destroy(TypeOfObjectOnMap[EndX, EndY, EndZ]);
                                        TypeOfObjectOnMapInt[EndX, EndY, EndZ] = 0;
                                    }
                                    EndX = (byte)xCoord; EndY = (byte)yCoord; EndZ = (byte)zCoord;
                                }
                                if (model3D == mine) // если объект Мина - то размещаем под ним цепь, если это
возможно, если нет, сразу крепление к поверхности
                                {
                                    for (int i = yCoord; i>= 1; i--)
                                    {
                                        if (TypeOfObjectOnMap[xCoord, i - 1, zCoord] != null)
                                        {
                                            if (TypeOfObjectOnMap[xCoord, i - 1, zCoord].tag == "mine")
                                            {
                                                Destroy(TypeOfObjectOnMap[xCoord, i - 1, zCoord]);
                                                TypeOfObjectOnMap[xCoord, i - 1, zCoord] = null;
                                                TypeOfObjectOnMapInt[xCoord, i - 1, zCoord] = 0;
                                            }
                                        }
                                        if (TypeOfObjectOnMap[xCoord, i - 1, zCoord] != null)
                                        {
                                            if (TypeOfObjectOnMap[xCoord, i - 1, zCoord].tag != "chain")
                                            {
                                                if (TypeOfObjectOnMap[xCoord, i, zCoord].tag != "mine")
                                                {
                                                    newMiniCube = new GameObject();
                                                    newMiniCube.transform.localScale = new Vector3(step, step,
step);
                                                    newMiniCube.AddComponent<MeshFilter>().mesh = generalMesh;
                                                    newMiniCube.transform.position = new Vector3(xCoord * step, i
* step, zCoord * step);
                                                    newMiniCube.transform.parent = transform;
                                                    newMiniCube.AddComponent<BoxCollider>().center = new
Vector3(0, 0, 0);
                                                    Destroy(TypeOfObjectOnMap[xCoord, i, zCoord]);
                                                    Instantiate(fixOnTheGround, newMiniCube.transform);
                                                    Instantiate(chainLink, newMiniCube.transform);
                                                    newMiniCube.tag = "chain";
                                                    TypeOfObjectOnMap[xCoord, i, zCoord] = newMiniCube;
                                                    newMiniCube = null;
                                                    break;
                                                }
                                                else
                                                {
                                                    Instantiate(fixOnTheGround, newMiniCube.transform);
                                                    TypeOfObjectOnMap[xCoord, i, zCoord] = newMiniCube;
                                                    newMiniCube = null;
                                                    break;
                                                }
                                            }
                                            else
                                            {
                                                break;
                                            }
                                        }
                                        yCoord = i - 1;
                                        xCoord = tempxCoord;
                                        zCoord = tempzCoord;
                                        newMiniCube = new GameObject();
                                        newMiniCube.transform.localScale = new Vector3(step, step, step);
                                        newMiniCube.AddComponent<MeshFilter>().mesh = generalMesh;
                                        newMiniCube.transform.position = new Vector3(xCoord * step, yCoord *
step, zCoord * step);
                                        newMiniCube.transform.parent = transform;
                                        newMiniCube.AddComponent<BoxCollider>().center = new Vector3(0, 0, 0);
                                        Instantiate(chainLink, newMiniCube.transform);
                                        newMiniCube.tag = "chain";
                                        TypeOfObjectOnMap[xCoord, yCoord, zCoord] = newMiniCube;
                                        TypeOfObjectOnMapInt[xCoord, yCoord, zCoord] = 8;
                                    }
                                }
```

```
                                newMiniCube = null;
                                mouseScrollValue = 0;
                            }
                        }
                    }
                }
            }
        }
        if (newMiniCube != null)
        {
            if (Mathf.Abs(Input.mouseScrollDelta.y) > 0) mouseScrollValue += Input.mouseScrollDelta.y; // смена
высоты колёсиком мышки

            Ray ray = camera.ScreenPointToRay(Input.mousePosition); // используем Raycast для определения
местоположения мышки на 3d пространстве
            RaycastHit hit;
            if (Physics.Raycast(ray, out hit, 1000 * step))
            {
                // получаем координаты, куда попал луч и переводим их в удобный для нас вид
                differenceX = Mathf.Abs(hit.point.x - hit.collider.gameObject.transform.position.x) < step / 2 -
0.0001f ? 0 : hit.point.x - hit.collider.gameObject.transform.position.x;
                differenceY = Mathf.Abs(hit.point.y - hit.collider.gameObject.transform.position.y) < step / 2 -
0.0001f ? 0 : hit.point.y - hit.collider.gameObject.transform.position.y;
                differenceZ = Mathf.Abs(hit.point.z - hit.collider.gameObject.transform.position.z) < step / 2 -
0.0001f ? 0 : hit.point.z - hit.collider.gameObject.transform.position.z;

                // сопоставляем эти координаты с координатами объекта, в который попал луч
                positionX = hit.collider.gameObject.transform.position.x + differenceX * 2;
                positionY = hit.collider.gameObject.transform.position.y + differenceY * 2 + mouseScrollValue *
step;
                positionZ = hit.collider.gameObject.transform.position.z + differenceZ * 2;


                if (positionY >= 0 && positionY / step + 0.1 < MapSizeEditor.countY) // Проверяем, выходит ли Y
(в unity это высота) за установленные пользователем границы
                {
                    yCoord = (int)(positionY / step + 0.1);
                    if (positionX >= 0 && positionX / step + 0.1 < MapSizeEditor.countX) // Проверяем, выходит ли
X за установленные пользователем границы
                    {
                        tempxCoord = (int)(positionX / step + 0.1);
                    }
                    if (positionZ >= 0 && positionZ/ step + 0.1 < MapSizeEditor.countZ) // Проверяем, выходит ли
Z за установленные пользователем границы
                    {
                        tempzCoord = (int)(positionZ / step + 0.1);
                    }
                }
                else mouseScrollValue = 0;

                Boolfas = false; // Найдена ли свободная клетка
                for (int i = yCoord; i < MapSizeEditor.countY; i++) // если клетка по высоте занята, то смотрим
свободное место сверху
                {
                    if (TypeOfObjectOnMap[tempxCoord, i, tempzCoord] == null)
                    {
                        xCoord = tempxCoord;
                        yCoord = i;
                        zCoord = tempzCoord;
                        Boolfas = true;
                        break;
                    }
                }
                if(!Boolfas) // Если свободного места сверху нет, то ищем снизу
                {
                    for (int i = MapSizeEditor.countY-1; i >= 0; i--)
                    {
                        if (TypeOfObjectOnMap[tempxCoord, i, tempzCoord] == null)
                        {
                            xCoord = tempxCoord;
                            yCoord = i;
                            zCoord = tempzCoord;
                            break;
                        }
                    }
```

```
                }
                if (TypeOfObjectOnMap[xCoord, yCoord, zCoord] == null)
                {
                    // если место не занято, то перемещаем редактируемый объект туда
                    newMiniCube.transform.position = new Vector3(xCoord * step, yCoord * step, zCoord * step);
                }
            }
        }
    }
    /// <summary>
    /// Метод, который удаляет <see cref="UnityEngine.GameObject"/> со сцены
    /// </summary>
    /// <remarks><see cref="UnityEngine.GameObject"/> выбирается с помощью <see
cref="UnityEngine.Physics.Raycast(Ray, out RaycastHit, float)"/>
    /// </remarks>
    void DeleteControl()
    {
        if (Input.GetMouseButtonDown(0) && deleteMode) // Пользователь нажал ЛКМ при Delete режиме
        {
            Ray ray = camera.ScreenPointToRay(Input.mousePosition); // используем Raycast для определения
местоположения мышки на 3d пространстве
            RaycastHit hit;
            if(Physics.Raycast(ray, out hit, 100 * step))
            {
                if (hit.collider.gameObject.tag != "ground") // запрещаем удаление самого нижнего слоя
                {
                    if (hit.collider.gameObject.tag != "chain") // запрещаем удаление цепи - чтобы пользователь
случайно не удалил её
                    {
                        Destroy(hit.collider.gameObject);
                        TypeOfObjectOnMapInt[(int)(hit.collider.gameObject.transform.position.x / step + 0.1),
(int)(hit.collider.gameObject.transform.position.y / step + 0.1),
(int)(hit.collider.gameObject.transform.position.z / step + 0.1)] = 0;
                    }
                    if (hit.collider.gameObject.tag == "mine") // удаляем цепь под миной, если пользователь всё
таки хотел удалить мину
                    {
                        for (int i = (int)(hit.collider.gameObject.transform.position.y / step + 0.1); i >= 0;
i--)
                        {
                            if (TypeOfObjectOnMap[(int)(hit.collider.gameObject.transform.position.x / step +
0.1), i, (int)(hit.collider.gameObject.transform.position.z / step + 0.1)].tag == "chain")
                            {
                                Destroy(TypeOfObjectOnMap[(int)(hit.collider.gameObject.transform.position.x /
step + 0.1), i, (int)(hit.collider.gameObject.transform.position.z / step + 0.1)]);
                                TypeOfObjectOnMapInt[(int)(hit.collider.gameObject.transform.position.x / step +
0.1), i, (int)(hit.collider.gameObject.transform.position.z / step + 0.1)] = 0;
                            }
                        }

                    }
                }
            }
        }
    }
    void objectRot(GameObject model)
    {
        if (Input.GetButtonDown("T"))
        {
            model.transform.Rotate(0,90,0);
        }
        else if (Input.GetButtonDown("R"))
        {
            model.transform.Rotate(0, -90, 0);
        }
    }
    private void Update()
    {
        if (!EventSystem.current.IsPointerOverGameObject() && !deleteMode)
        {
            SpawnerControl(model3D);
        }
        if (!EventSystem.current.IsPointerOverGameObject())
        {
            DeleteControl();
```

```
        }
        if(model3D != null && newMiniCube != null && model3D != mine)
        {
            objectRot(newMiniCube);
        }
    }
}
```

Раздел 4. SaveButton.cs

В данном скрипте реализовано сохранение трёхмерного массива с объектами в файл, для дальнейшего использования.

Текст скрипта с комментариями

```
using UnityEngine;
using System.IO;

public class SaveButton : MonoBehaviour
{
    string nameOfSaveFile = "/save1.txt", // имя файла
        nameDirectiry = "/saves"; // имя папки для этого файла
    private void Start()
    {
        if (!Directory.Exists(Directory.GetCurrentDirectory() + nameDirectiry)) // проверяем существует ли папка
        {
            Directory.CreateDirectory(Directory.GetCurrentDirectory() + nameDirectiry); // создаём папку
        }
        if (!File.Exists(Directory.GetCurrentDirectory() + nameDirectiry + nameOfSaveFile)) // проверяем
существует ли файл
        {
            File.Create(Directory.GetCurrentDirectory() + nameDirectiry + nameOfSaveFile); // создаём файл
        }
    }
    public void IsTapped()
    {
        if(File.Exists(Directory.GetCurrentDirectory() + nameDirectiry + nameOfSaveFile)) // проверяем существует
ли файл
        {
            string sizeOfMas = MapSizeEditor.countX + "\n" + MapSizeEditor.countY + "\n" + MapSizeEditor.countZ
+"\n"; // сохраняем размеры карты построчно
            string mass = "";
            for (int x = 0; x<MapSizeEditor.countX; x++)
            {
                for (int y = 0; y < MapSizeEditor.countY; y++)
                {
                    for (int z = 0; z < MapSizeEditor.countZ; z++)
                    {
                        mass += WorldLogic.TypeOfObjectOnMapInt[x, y, z].ToString(); // сохраняем типы всех
объектов подряд
                        if (WorldLogic.TypeOfObjectOnMapInt[x, y, z] == 1) mass +=
WorldLogic.rotation.ToString();
                    }

                }

            }
            File.WriteAllText(Directory.GetCurrentDirectory() + nameDirectiry + nameOfSaveFile, sizeOfMas +
mass); // записываем string в файл
        }
    }
}
```

Загрузка игрового пространства

Раздел 1. Функции будущей программы.

Для реализации загрузки игрового пространства нам нужно также создать сцену, как и в редакторе игрового пространства, и реализовать загрузку и интерпретацию данных из текстового файла.

Раздел 2. Сцена загрузки игрового пространства.

Для начала создаём сцену. По умолчанию на сцене присутствует только: Main Camera, Directional Light и EventSystem. Для выполнения поставленной задачи нужно создать ещё ряд объектов, и итого на сцене будут следующие объекты:

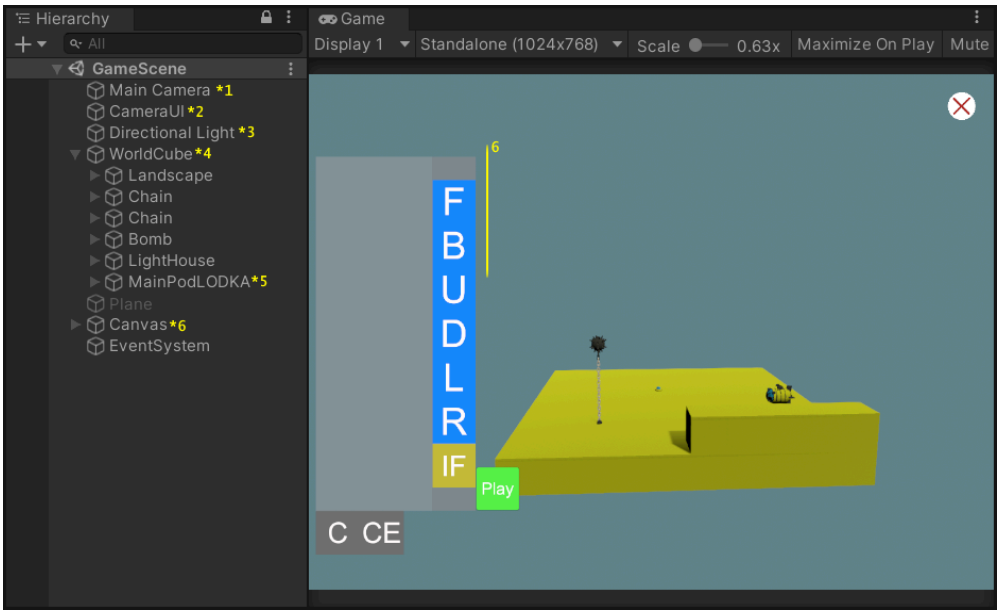
1. Main Camera - главная камера, которой пользователь сможет управлять.

2. CameraUI - вторичная камера, благодаря которой мы можем видеть объекты пользовательского интерфейса(далее UI).

3. Directional Light - освещение на сцене.

4. WorldCube - главный объект на сцене. Тут будет находится скрипт GameWorldLogic.cs

5. MainPodLODKA - объект, которым управляет игрок с помощью алгоритма. На это объекте находится скрипт MoveLogic.cs, подробнее о нём в курсовой работе Агринского Никиты Андреевича.



6. Canvas - обязательный элемент UI. На нём находятся все элементы UI, а так же тут расписано управление подводной лодкой. Подробнее об этом в курсовой работе Газизова Айнура Айдаровича.

Раздел 3. GameWorldLogic.cs

В скрипте GameWorldLogic.cs реализована загрузка игрового пространства из файла.

Текст скрипта с комментариями

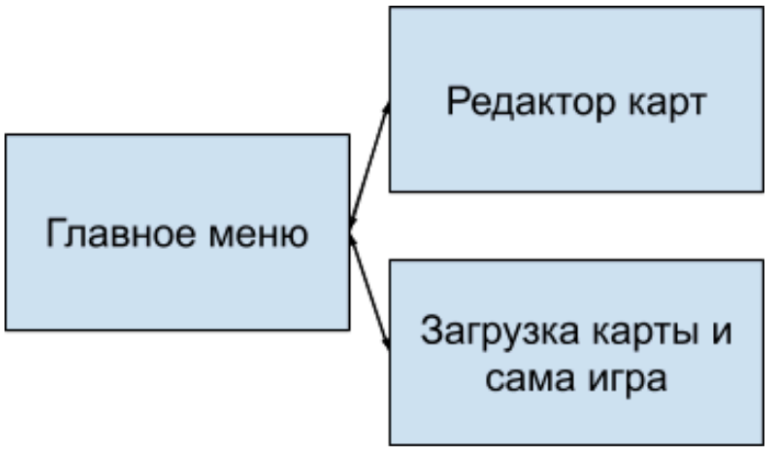
```
using System.IO;
using UnityEngine;

public class GameWorldLogic : MonoBehaviour
{
    public GameObject submarine, mine, chainLink, fixOnTheGround, lighthouse; // 3d модели

    private static int step = MapSizeEditor.step; // step - размер одной клетки. Пр. 1 кл = 10 у.е

    public Camera camera; // камера на сцене, через которую смотрит игрок. Нужна для использования Raycast

    // Материалы для 3d моделей
    public Material generalMaterial, invisibleMaterial,sandMaterial;
    public Mesh generalMesh;
    public static Material materialForMiniCube;

    private GameObject newMiniCube; // временный объект для размещения его на карте
    private GameObject landscape; // главный объект для иерархии ландшафта

    public static byte[,,] TypeOfObjectOnMapInt; // массив с ID объектов

    private void Start()
    {
        // загружаем игровое пространство
        landscape = new GameObject(); // создаём родительский объект, куда будем записывать все объекты, типа
9(поверхность куб)
        landscape.name = "Landscape";
        landscape.transform.parent = transform;
        string nameOfSaveFile = "/save1.txt", nameDirectiry = "/saves";
        string[] readedLines = File.ReadAllLines(Directory.GetCurrentDirectory() + nameDirectiry +
nameOfSaveFile); // чтение файла
        string sepLine = readedLines[0]; // читаем первую линию из файла
        if (byte.TryParse(sepLine, out byte outValue)) MapSizeEditor.countX = outValue; // она отвечает за размер
карты по X
        sepLine = readedLines[1]; // читаем вторую линию из файла
        if (byte.TryParse(sepLine, out outValue)) MapSizeEditor.countY = outValue; // она отвечает за размер карты
по Y
        sepLine = readedLines[2]; // читаем третью линию из файла
        if (byte.TryParse(sepLine, out outValue)) MapSizeEditor.countZ = outValue; // она отвечает за размер карты
по Z
        sepLine = readedLines[3]; // читаем четвёртую линию из файла, которая хранит типы объектов расположенных
подряд
        int count = 0;
        TypeOfObjectOnMapInt = new byte[MapSizeEditor.countX, MapSizeEditor.countY, MapSizeEditor.countZ]; //
инициализируем массив с размерами, указанными в файле
        for (byte x = 0; x < MapSizeEditor.countX; x++)
        {
            for (byte y = 0; y < MapSizeEditor.countY; y++)
            {
                for (byte z = 0; z < MapSizeEditor.countZ; z++)
                {
                    if (byte.TryParse(sepLine[count++].ToString(), out outValue)) // конвертируем символ в число
                    {
                        TypeOfObjectOnMapInt[x, y, z] = outValue;
                        if (outValue != 1) CreateObjectType(x, y, z, outValue);
                        else
                        {
                            byte.TryParse(sepLine[count++].ToString(), out byte outValue2);
                            CreateObjectType(x, y, z, outValue,outValue2);
                        }
                    }
                }
            }
```

```
        }
    }
    void CreateObjectType(byte x, byte y, byte z, byte type, byte rotation = 0)
    {
        if(type != 0)
        {
            newMiniCube = new GameObject();
            newMiniCube.transform.localScale = new Vector3(step, step, step);
            newMiniCube.AddComponent<MeshFilter>().mesh = generalMesh;
            newMiniCube.transform.position = new Vector3(x * step, y * step, z * step);
            newMiniCube.AddComponent<BoxCollider>().center = new Vector3(0, 0, 0);
            if(type != 9) newMiniCube.transform.parent = transform;
            switch (type)
            {
                case 1:
                    Instantiate(submarine, newMiniCube.transform);
                    newMiniCube.AddComponent<MoveLogic>();
                    newMiniCube.name = "MainPodLODKA";
                    CommandReader.submarine = newMiniCube;
                    if(rotation != 0)
                    {
                        newMiniCube.transform.Rotate(0, 90*(rotation), 0);
                    }
                    break;
                case 2:
                    Instantiate(mine, newMiniCube.transform);
                    if (TypeOfObjectOnMapInt[x, y - 1, z] != 8) // если под бомбой снизу нет цепи, то создаём
объект крепёж
                    {
                        Instantiate(fixOnTheGround, newMiniCube.transform);
                    }
                    newMiniCube.name = "Bomb";
                    break;
                case 3:
                    Instantiate(lighthouse, newMiniCube.transform);
                    newMiniCube.name = "LightHouse";
                    break;
                case 8:
                    Instantiate(chainLink, newMiniCube.transform);
                    if(y!=0)
                    {
                        if (TypeOfObjectOnMapInt[x, y - 1, z] != 8) // если под цепью снизу нет цепи, то создаём
объект крепёж
                        {
                            Instantiate(fixOnTheGround, newMiniCube.transform);
                        }
                    }
                    newMiniCube.name = "Chain";
                    break;
                case 9:
                    newMiniCube.AddComponent<MeshRenderer>().material = generalMaterial;
                    newMiniCube.name = "Box["+x+":"+y+":"+z+"]";
                    newMiniCube.transform.parent = landscape.transform;
                    break;
                default:
                    break;
            }
        }
    }
}
```

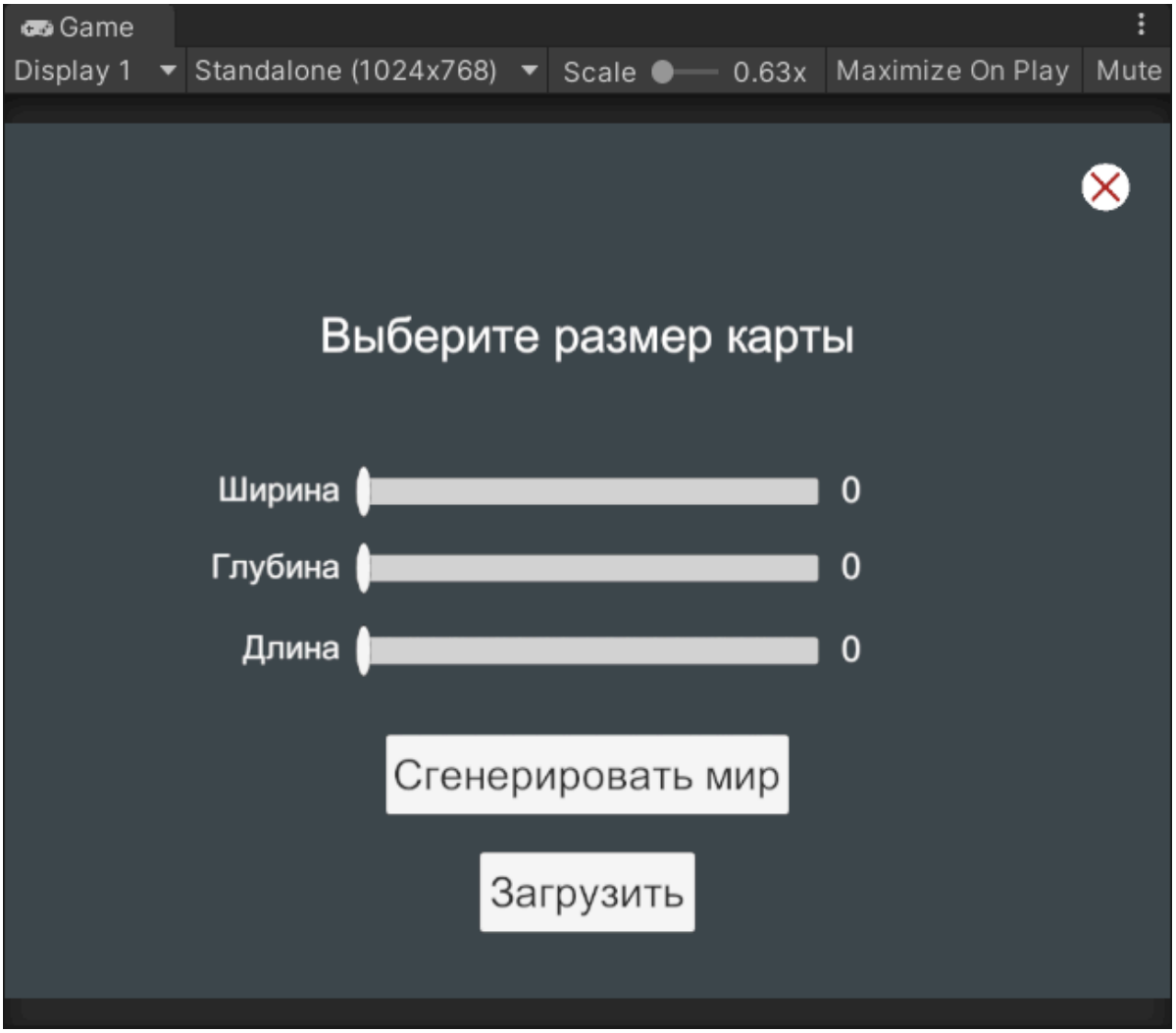
Совместная работа с командой

Соединение скриптов и сцен в общую картину

Для того, чтобы соединить все наши программы, нужно решить, по какой схеме будут запускаться сцены. По общему решению выбрана такая схема:

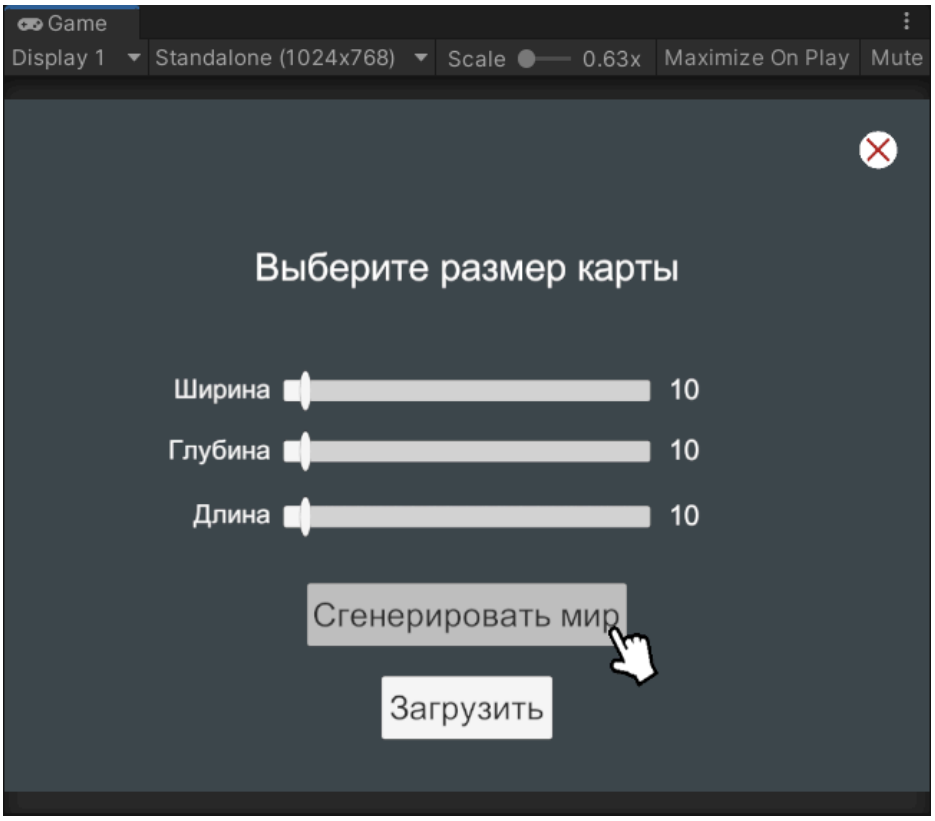


Из главного меню игрок сможет настраивать размер создаваемой пустой карты, которую далее он сможет редактировать. Также игрок может загрузить ранее сохранённую карту.

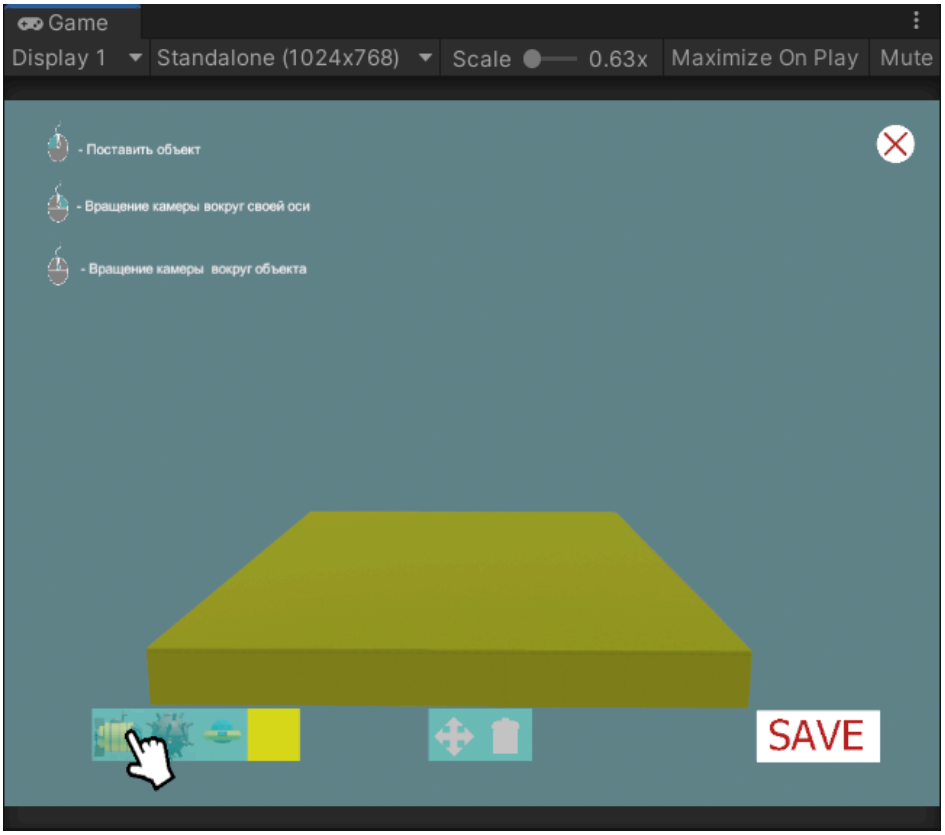


Примеры работы игры

Создаём пустое игровое пространство

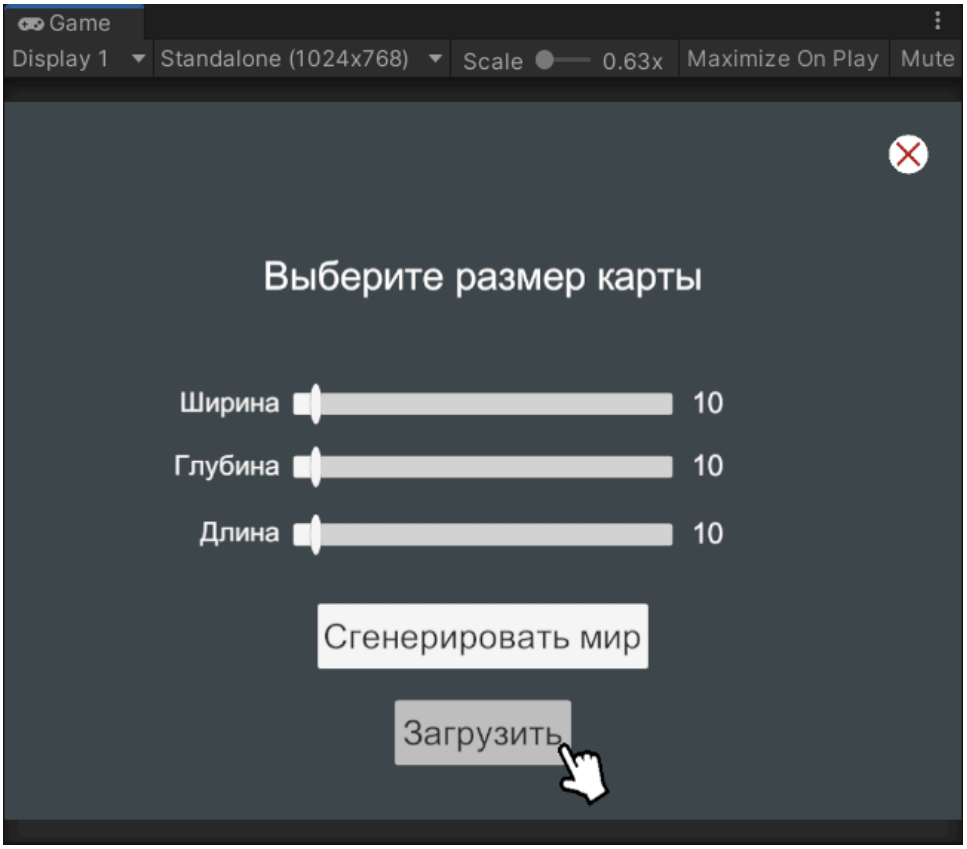


Теперь можем с помощью панели с объектами начать редактировать карту

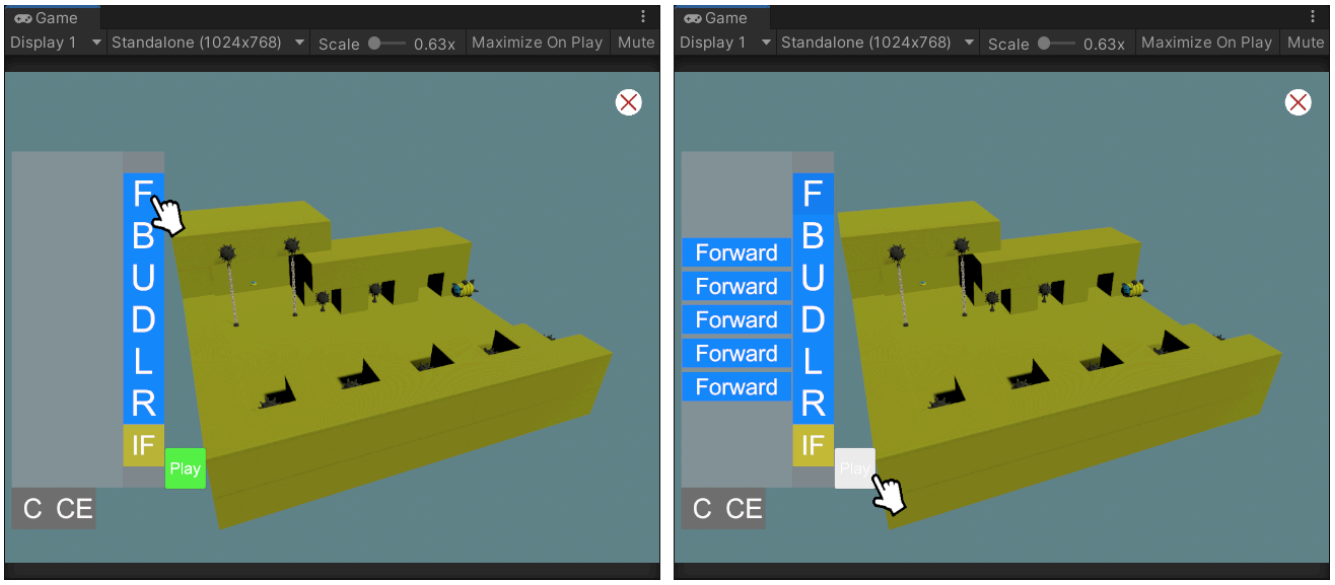


Когда построили игровое
пространство, можно его сохранить

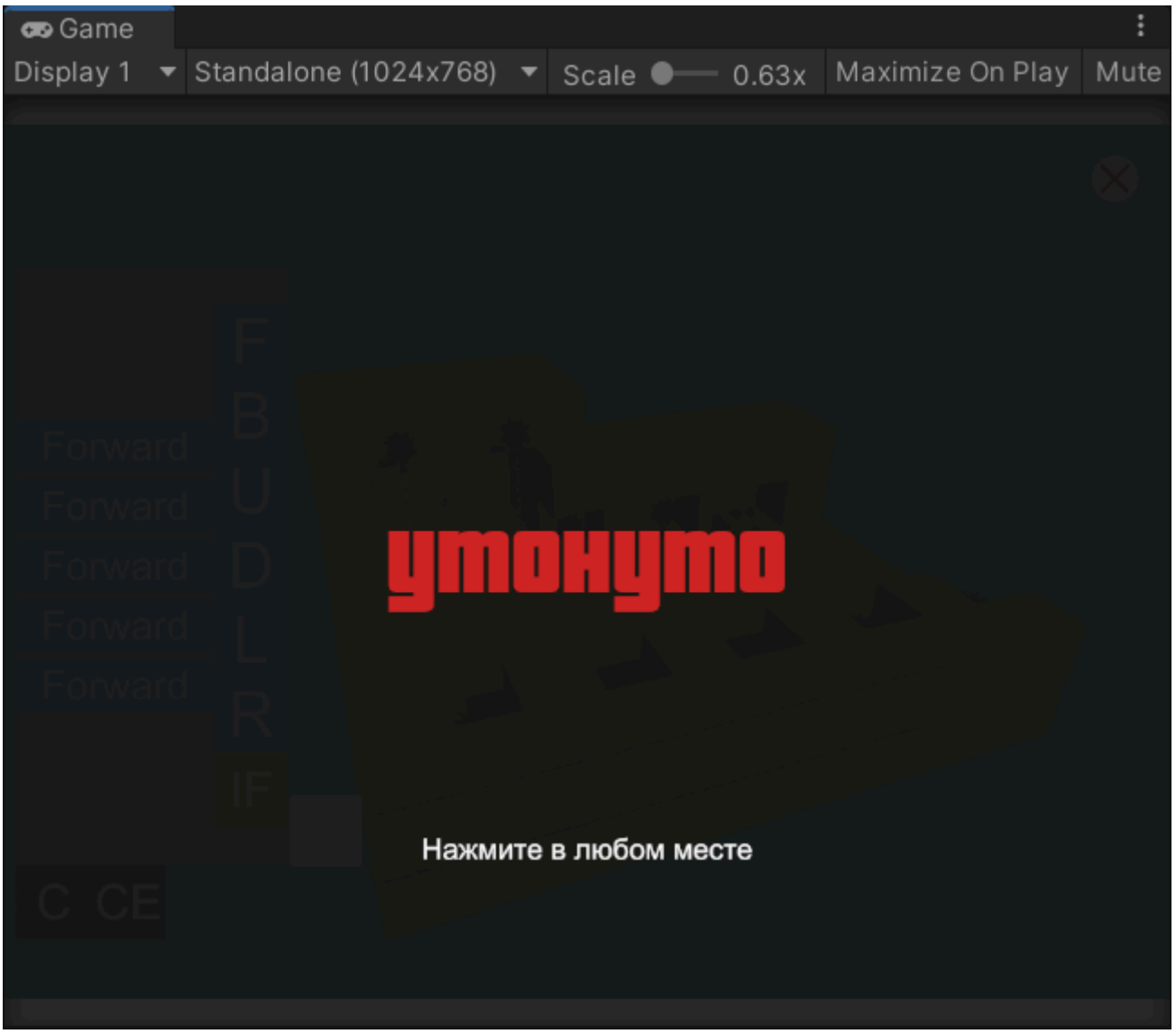
Закрываем редактор на крестик в правом верхнем
углу и попав на главное меню, нажимаем кнопку загрузить



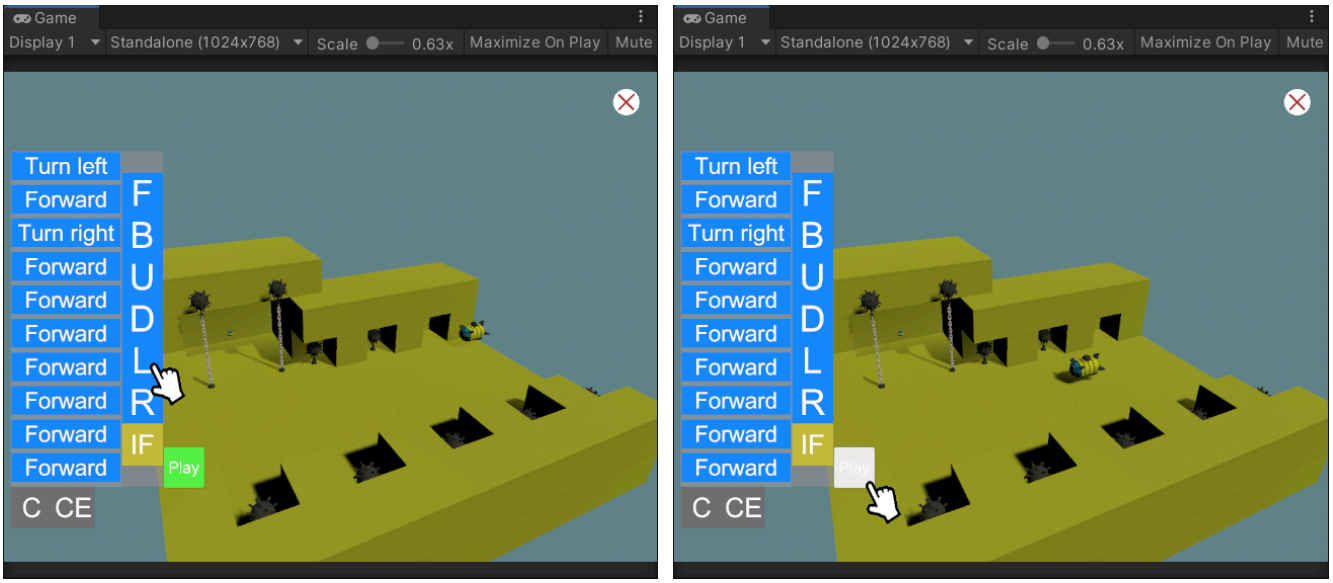
Попробуем создать путь только вперёд и нажимаем Play



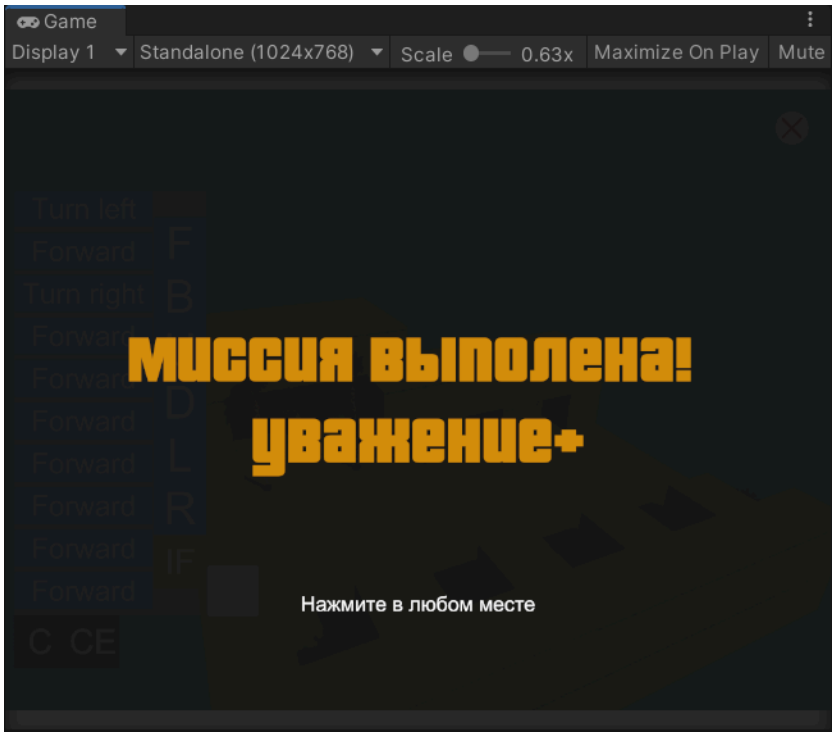
В итоге мы наткнулись на мину, которая действует на все соседние клетки, поэтому нужно придумать другой путь, который будет лежать не рядом с минами.



Такой путь найден. L F R F(x7) R F(x4)



Конечная точка пути достигнута



Заключение

В ходе данной курсовой работы мне успешно удалось выполнить поставленные задачи: я создал конструктор игрового пространства, запись его в трёхмерный массив, сохранение и загрузку этого пространства из файла. Также общими усилиями мы соединили все наши реализованные задачи в один общий продукт и получили прототип некой игры про подводную лодку. C# несколько отличается от C++, но в ходе разработки с большими трудностями я не столкнулся, так как это всё таки семейство языков Си.