МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по учебной практике

Тема: Алгоритм Дейкстры

Студент гр. 9303 _____ Алексеенко Б. С

Студент гр. 9303 _____ Махаличев Н. А.

Студент гр. 9304 _____ Шуняев А. В.

Руководитель _____ Фиалковский М. С.

Санкт-Петербург
2021

ЗАДАНИЕ

НА УЧЕБНУЮ ПРАКТИКУ

Студент Алексеенко Б., гр. 9303

Студент Махаличев Н., гр. 9303

Студент Шуняев А., гр. 9304

Тема работы: Алгоритм Дейкстры

Задание на практику:

Разработка игры, которая интерактивно демонстрирует алгоритм Дейкстры.

Предполагаемый объем пояснительной записки:

Не менее 10 страниц.

Сроки проведения практики: 01.07.2021-14.07.2021

Дата сдачи отчета: 14.07.2021

Студент	_____	Алексеенко Б. С
Студент	_____	Махаличев Н. А.
Студент	_____	Шуняев А. В.
Руководитель	_____	.Фиалковский М. С.

АННОТАЦИЯ

Целью учебной практики является разработка игры в жанре Tower Defense, в которой игрок сможем наглядно пронаблюдать за тем, как работает алгоритм.

Игра разрабатывается на движке Unity командой из 3-х человек. Роли были распределены среди участников бригады согласно поставленным задачам. В отчете представлена информация о разработке проекта.

SUMMARY

The aim of the training practice is to develop a tower defense game in which the player will be able to visually observe how the algorithm works.

The game is being developed on the Unity engine by a team of 3 people. The roles were distributed among the team members according to the assigned tasks. The report provides information on the development of the project.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Целью практической работы является разработка интерактивной визуализации алгоритма Дейкстры, т. е. создание игры, с помощью которой игрок сможет в легкой форме понять то, как работает данный алгоритм. Для достижения поставленной цели были решены следующие задачи:

1. Согласовать с руководителем спецификацию проекта. По итогу, для разработки игры был выбран игровой движок Unity, который базируется на языке программирования C#, собственно проект был реализован, с помощью инструментов этого языка.
2. Распределить роли между членами бригады и составить примерный план разработки.
3. Изучить базовые принципы и инструменты Unity, освоить базу C#, освежить память об алгоритме Дейкстры.
4. Реализовать задуманные игровые механики и отладить их.
5. Разработать нативный UI интерфейс, в котором пользователь сможет быстро разобраться.
6. Протестировать игру на заранее подготовленных тестах и на случайно генерированных.

1. ТРЕБОВАНИЯ К ПРОГРАММЕ

1.1.1 Требования к игровой части проекта.

- Игра должна концептуально относиться к жанру игр Tower Defense.
- Игровые механики и интерфейс игры должны быть легко поняты играющим.
- Игра не должна быть слишком тяжелой и должна оставаться проходимой (ведь ее смысл — дать играющему представление об алгоритме Дейкстры в легкой обучающей форме).
- Сам игровой процесс разбит на волны. Когда заканчивается одна волна, граф должен перестраиваться и маршрут прохода врагов должен быть пересчитан.

1.1.2 Требования к интерфейсу.

- Начало игры и управление ее состоянием во время игрового процесса должно происходить через интерфейс меню.
- Взаимодействие с игрой должно происходить с помощью интерфейса установки башен.
- Интерфейс должен подсказывать игроку на сколько пройден уровень.

1.1.3 Наброски интерфейса.

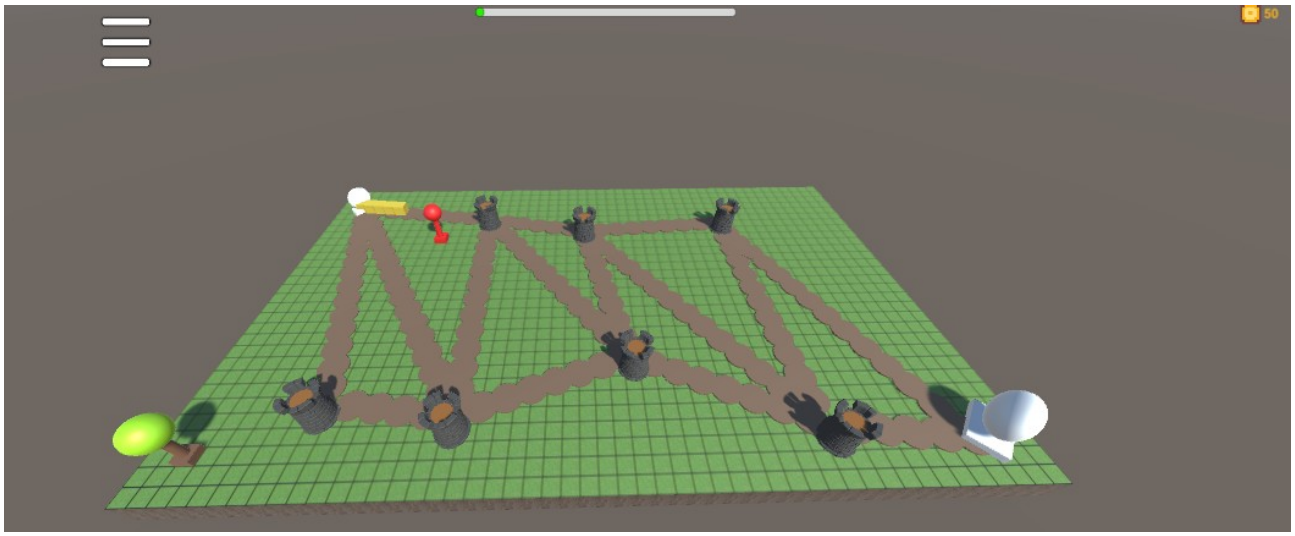

Рисунок 1–Интерфейс во время игрового процесса

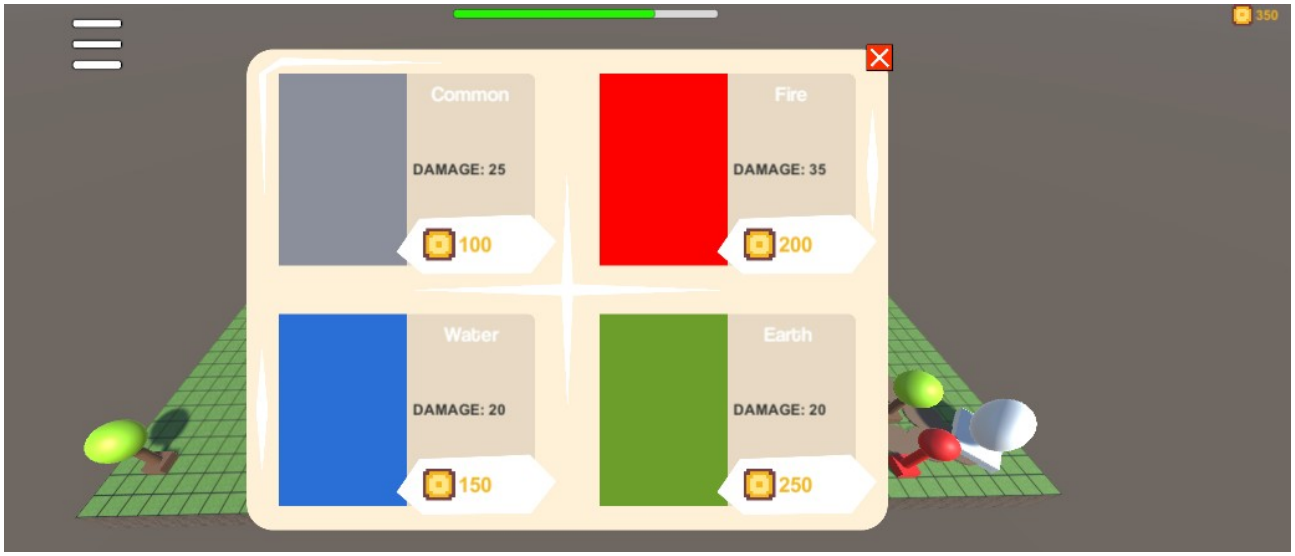

Рисунок 2–Интерфейс выбора башни

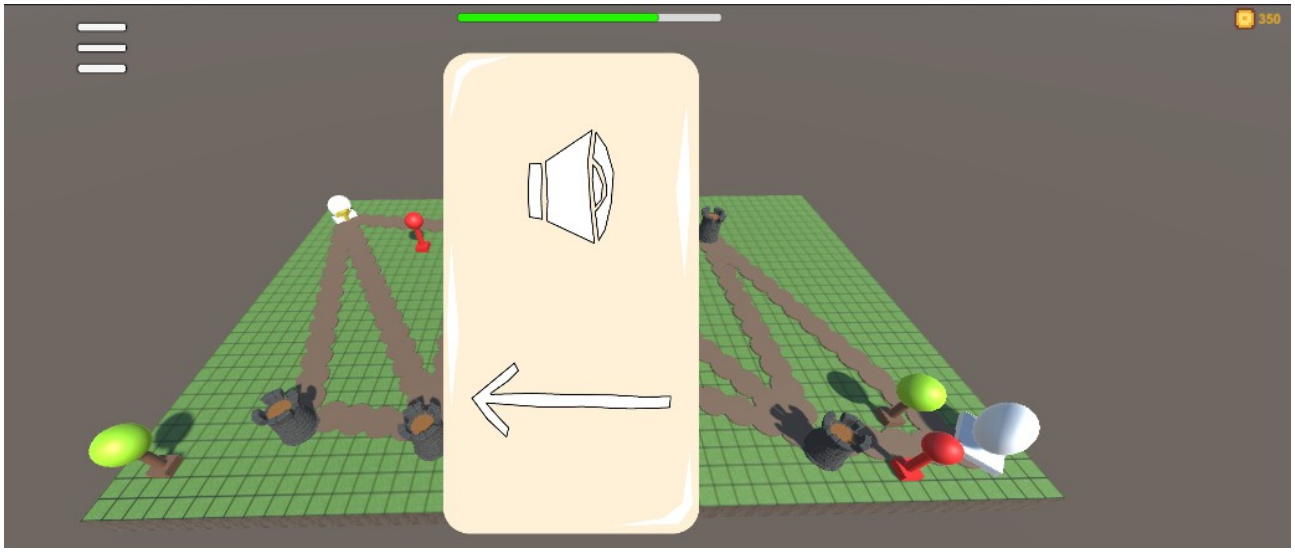

Рисунок 3–Интерфейс игры на паузе

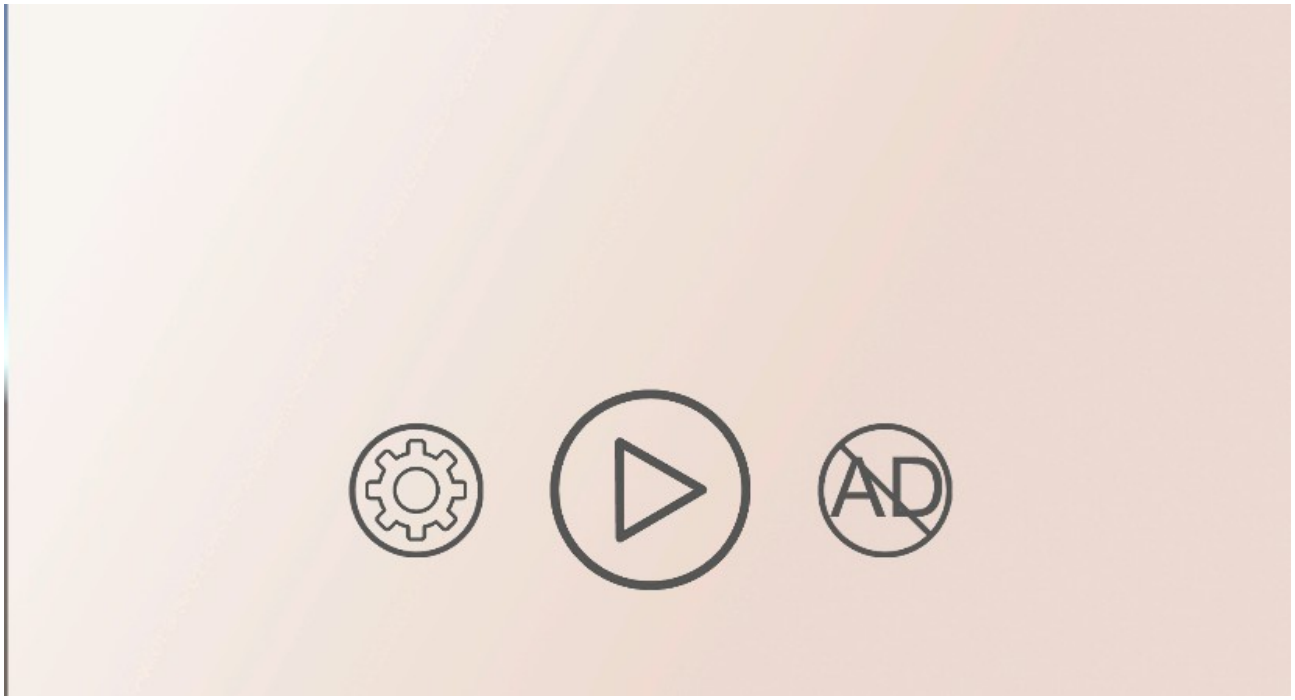

Рисунок 4–Главное меню игры

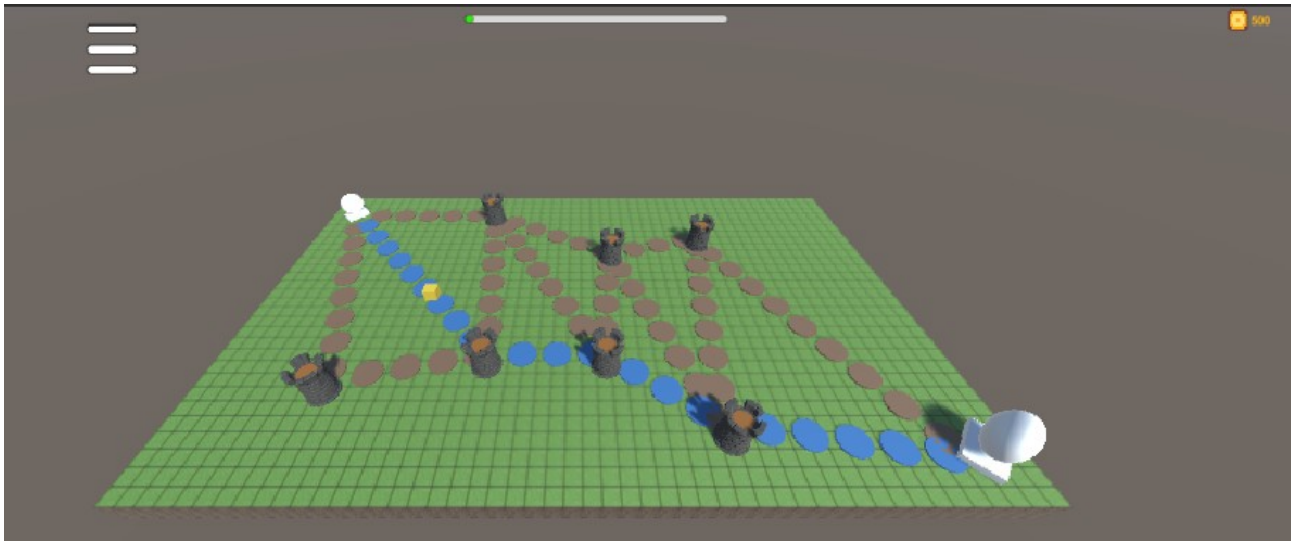

Рисунок 5–Отображение построенного пути.

1.2 Уточнения после сдачи первой версии.

- Необходимо разработать мануал, где будет описано за что отвечают объекты на уровне и как, в принципе, играть в игру.

2. ОРГАНИЗАЦИЯ РАБОТЫ

2.1. План разработки

Задачи работы: изучить основы C#, для того чтобы построить архитектуру базируясь на инструментах данного языка, изучить основы движка Unity, разработать игровые механики, внедрить алгоритм, протестировать.

1. К 02.07.2021 изучить основы C# для дальнейшей работы над проектом, создать репозиторий проекта.
2. К 03.07.2021 обсудить основную концепцию игры, параллельно разобраться в Unity.
3. К 06.07.2021 реализовать основные классы для работы с графом (генерация баз, построение над ними графа).
4. К 09.07.2021 доработать UI для взаимодействия игрока с игрой.
5. К 10.07.2021 реализовать алгоритм Дейкстры и класс врагов, которые будут идти по отработанному пути.
6. К 13.07.2021 протестировать и отладить игру, если необходимо, то произвести рефакторинг кода.

2.2. Распределение ролей в команде

Обязанности между участниками команды распределены следующим образом:

Шуняев А. В. – лидер команды, алгоритмист и геймдизайн

Махаличев Н. А. – реализация UI, алгоритмист.

Алексеенко Б. С. – алгоритмист, написание отчета.

3. ОПИСАНИЕ КОДА ПРОГРАММЫ

3.1. Описание алгоритма и используемых структур данных

Основной класс, который связывает все остальные – `GameCore`. Глобально структуру классов проекта можно разделить на несколько частей, а именно: система раундов (`RoundManager`), система врагов (`Enemy`, `EnemySpawn`), система генерации (`Cell`, `Map`, `Base`, `RoadSpawn`), система, которая реализует графовое представление карты (`Edge`, `Graph`), система, которая реализует алгоритм Дейкстры (`Dijkstra`), система башней (`Tower` и `Attack`), система интерфейсов. Можно сказать, что глобально логика игры состоит из алгоритма, логики интерфейсов и игровой логики.

Алгоритм Дейкстры – поиск минимального пути в графе, с помощью очереди с приоритетами.

Сначала присваиваем приоритет -1 ко всем вершинам, кроме стартовой, ей присваиваем 0. После чего на каждой итерации алгоритма сортируем список по величине приоритета. Далее, в зависимости от суммы веса ребра и приоритета вершины от которой мы идем, строится маршрут, после чего очередь с приоритетом перестраивается. На следующей итерации мы возьмем вершину с наименьшим приоритетом.

Алгоритм реализуется метод `Algorithm` класса `Dijkstra`.

Для представления графа используются классы `Edge` и `Graph`, где первый репрезентирует компонент графа, как ребро, а второй хранит словарь вершин и рёбер. Вершины представлены классом `GameObject`, который является инструментом Unity. `GameObject`, который используется для работы с вершинами, связан с классами семейства `Base`, которые уже связаны с игровой логикой напрямую. В множество вершин входят обычные базы, место появления противников и главная база. Маршрут строится от места появления противников до главной базы.

В конце раунда для перестройки графа происходит перестройка графа, которая реализуется методом `RemakeGraph()`,

3.2. Описание основных классов для реализации игровой логики

Как было указано выше, основной класс, который связывает остальные системы – `GameCore` (паттерн Фасад).

Если разбирать игру поэтапно, то сначала с помощью классов `Map` и `Cell` происходит генерация уровня. Когда уровень сгенерирован, метод класса `Map` `SpawnBases()` создает множество баз (вершин графа), которые после связываются ребрами методом `CreateGraph()`. Карта уровня условно разделяется на несколько секторов, в которых случайным образом выбирается координата базы. На уровне есть одно место появления противников и одна главная база. Противники могу разрушать базы, в таком случае, после окончания волны, будет происходить перестроение графа и нахождение нового пути для врагов. Классы `EnemySpawn` и `RoadSpawn` нужны для того, чтобы отобразить в самой игре модели врагов и дорог (ребер графа), отображение баз и башней не требует логики, поэтому для этого используется метод `PlaceMapUnit()` класса `Map`.

Класс `Tower` и `AttackUnit` реализуют логику механики башней. `Tower` – класс, который представляет саму башню, а `AttackUnit` – снаряд, который выпускает башня во врагов. Эти классы являются абстрактными, от них унаследованы классы, которые представляют разные виды башней и соответствующие им виды снарядов (`EarthTower`, `FireTower`, `WaterTower`, `CommonTower`).

3.3. Описание основных классов для реализации интерфейса

Класс Audio отвечает за использование звуков, за их громкость и за часть UI, которая с этим связана (кнопка громкости). Класс Coin отвечает за отображение счетчика монет. Класс MainUI – класс фасад, который связывает все элементы UI. Класс Pause отображает меню паузы. Класс ProgressBar отображает полосу прогресса. Класс SceneManager меняет сцены (то, что отображает Unity игроку в качестве перспективы).

4. ТЕСТИРОВАНИЕ ПРОГРАММЫ

Для начала был протестирован интерфейс. Переходы от сцены к сцене работают во всех случаях корректно. Меню установки башни работает корректно. Звук при нажатии на кнопки так же стабильно правильно проигрывается.

Построение графа было протестировано вручную. Игра была запущена множество раз, ошибок не было выявлено. Во время перестройки, конфигурация графа изменяется, так что были протестированы и разные конфигурации графа. Ошибок самой перестройки тоже не было выявлено.

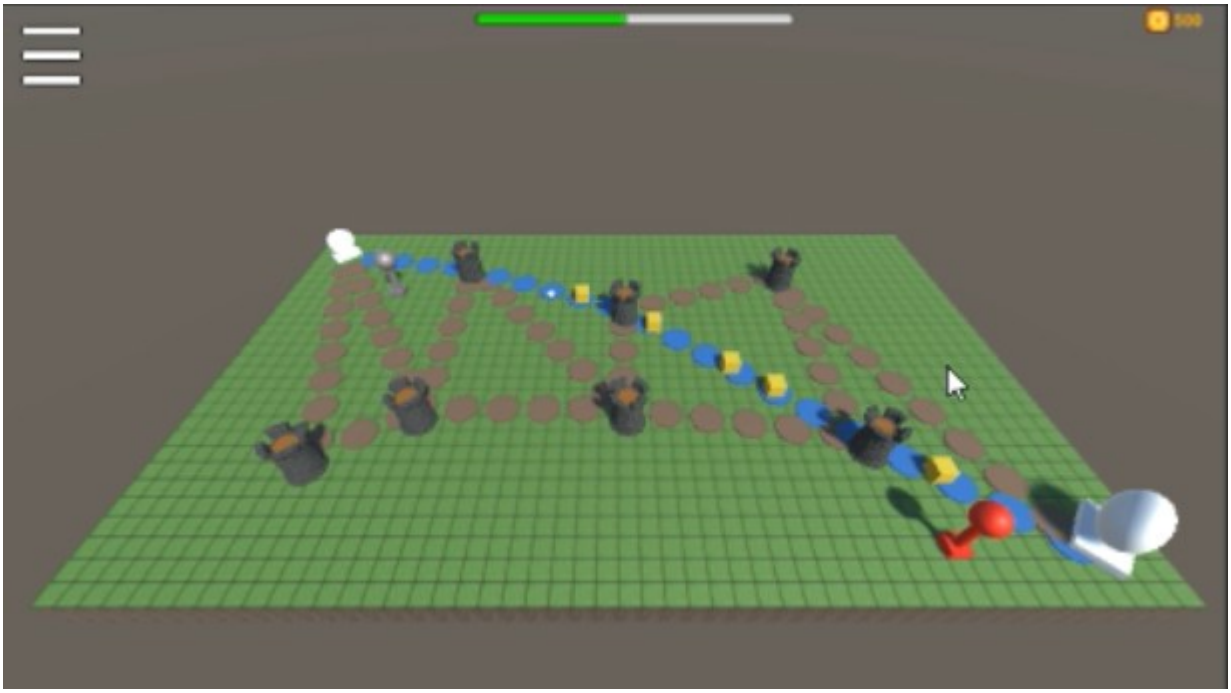

Рисунок 6–На данном экранном снимке продемонстрированно, что враги действительно идут по кратчайшему пути

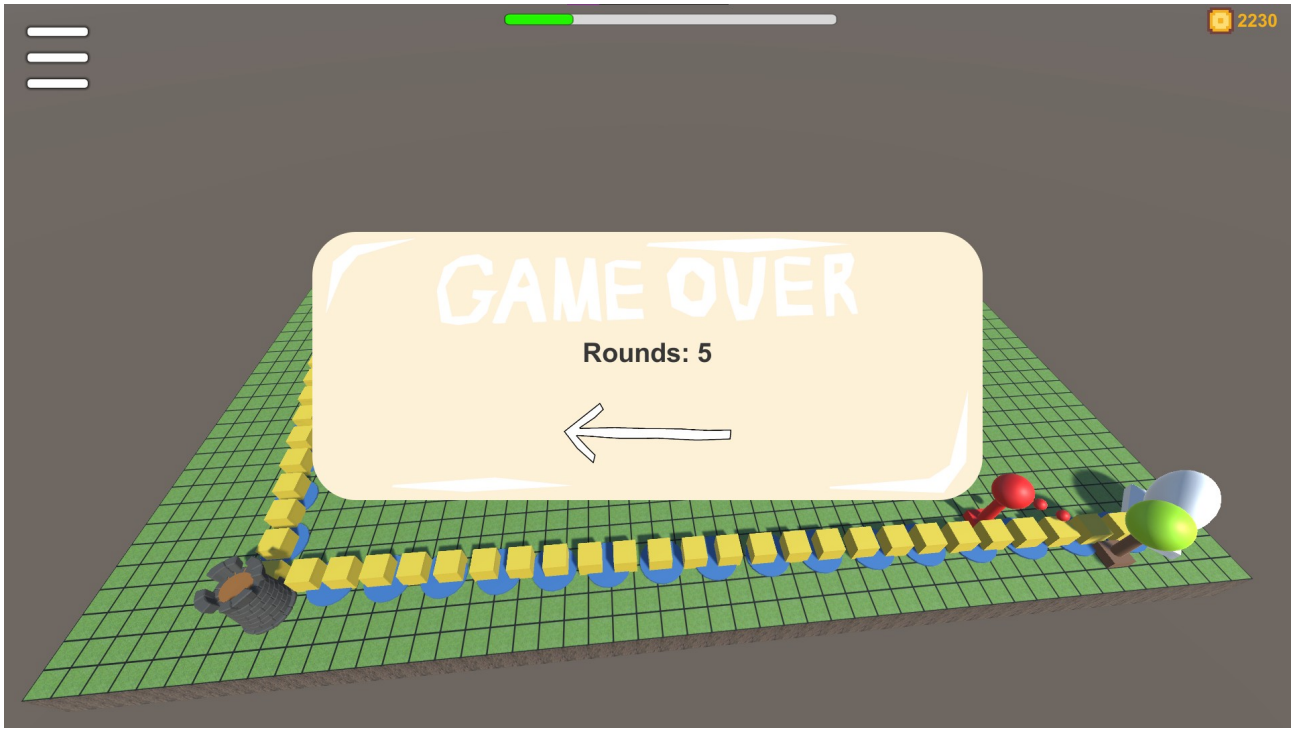

Рисунок 7–Меню поражения

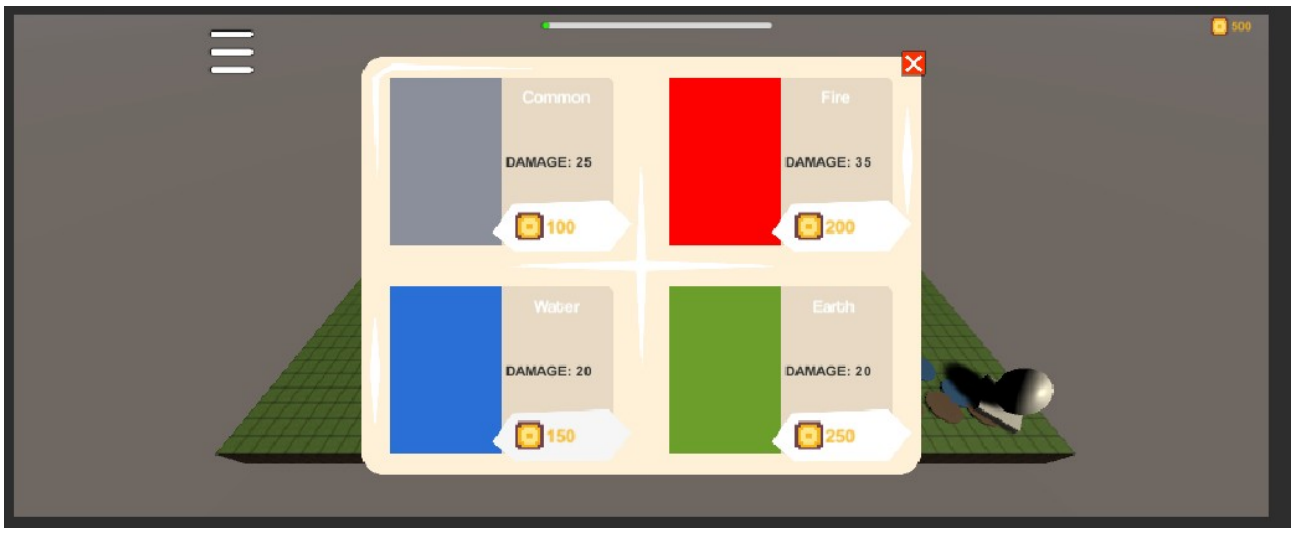

Рисунок 8–При нажатии на клетку открывается меню выбора башни

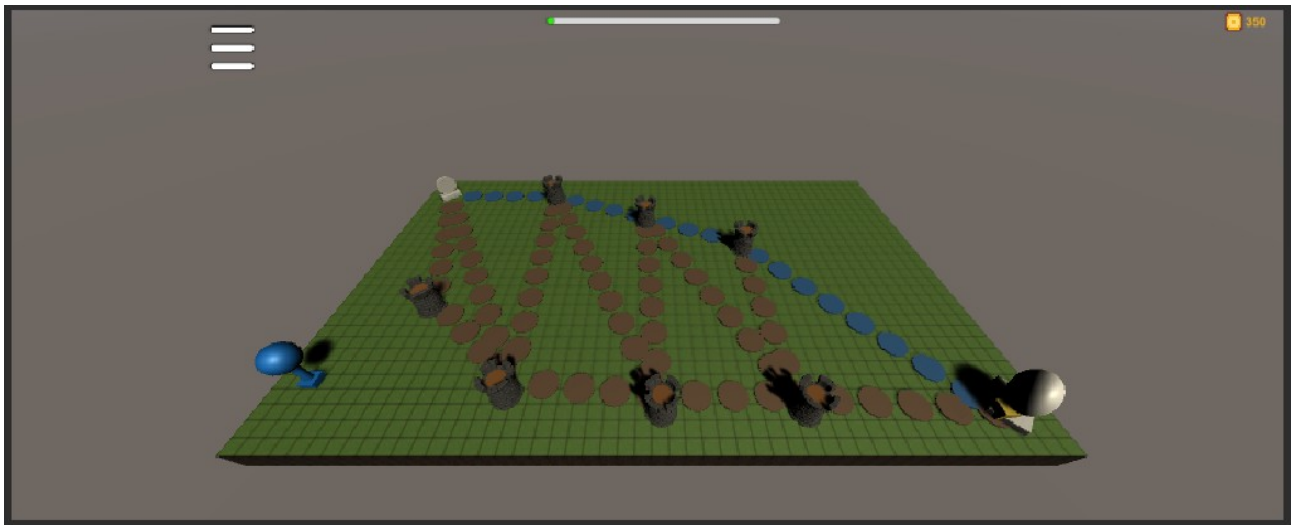

Рисунок 9–При нажатии на стоимость выбранной башни, она устанавливается на поле.

ЗАКЛЮЧЕНИЕ

В ходе работы были выполнены следующие задачи:

- Была собрана бригада для разработки проекта, были поставлены цели, составлен план и распределены роли. Были изучены основы языка C# и игрового движка Unity. Взаимодействие членов команды было реализовано с помощью Unity Hub. Были преодолены трудности с организацией рабочего процесса и получен опыт работы в команде.
- Был разработан концепт игры, изучены некоторые принципы гейм-дизайна.
- Была разработана игра, которая удовлетворяет установленным требованиям.
- Игра была отлажена и были улучшены некоторые аспекты кода.
- Было проведено тестирование работы алгоритма для построения маршрута врагов.

Обобщив все вышесказанное, можно сказать, что была достигнута цель учебной практики: разработка интерактивной визуализации алгоритма Дейкстры, т. е. создание игры.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. https://metanit.com/sharp/
2. Andrew Troelsen Philip Japikse Pro C# 7 With .NET and .NET Core Eighth Edition. (2018 г.)
3. Документация .NET и C# https://docs.microsoft.com/en-us/dotnet/csharp/
4. Документация Unity https://docs.unity3d.com/Manual/index.html

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Файл Base.cs

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

public abstract class Base : MonoBehaviour
{
    [SerializeField] protected int _health;
    protected int _number;

    protected GameCore _core;

    protected virtual void TakeDamage(int damage)
    {
        _health -= damage;
    }

    protected virtual void DeleteBase()
    {
        _core.DeleteBase();
    }

    public virtual void Destroy()
    {

    }
    public virtual int GetHealth()
    {
        return _health;
    }
}
```

Файл MainBase.cs

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

public class MainBase : Base
{
    protected override void TakeDamage(int damage)
    {
        base.TakeDamage(damage);
    }
}
```

Файл MinorBase.cs

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

public class MinorBase : Base
{
    private List<GameObject> _roads;
    private void Awake()
    {
        _core = FindObjectOfType<GameCore>();
        _roads = new List<GameObject>();
    }

    protected override void TakeDamage(int damage)
    {
        base.TakeDamage(damage);
    }

    private void OnTriggerEnter(Collider other)
    {
        Enemy enemy = other.gameObject.GetComponent<Enemy>();
        if(enemy != null)
        {


enemy.SetTarget(_core.GetEnemyTarget(enemy.GetTargetNumber()));
            TakeDamage(1);
        }
    }

    protected override void DeleteBase()
    {
        base.DeleteBase();
    }
    public override void Destroy()
    {
        Destroy(gameObject);
    }
}
```

Файл EnemySpawn.cs

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

public class EnemySpawn : Base
{
    private Transform _transform;
```

```
    private Vector3 _spawnPoint;

    private void Awake()
    {
        _transform = GetComponent<Transform>();
        _spawnPoint = new Vector3(_transform.position.x,
_transform.position.y + 2, _transform.position.z);
    }
    public List<GameObject> SpawnEnemy(int amount, GameObject
_enemyPrefab, Vector3 target)
    {
        List<GameObject> enemies = new List<GameObject>();
        for (int i = 0; i < amount; i++)
        {
            GameObject currentEnemy = Instantiate(_enemyPrefab,
_spawnPoint, Quaternion.identity);
            enemies.Add(currentEnemy);
        }

        StartCoroutine(SetTargets(enemies, target));
        return enemies;
    }

    private IEnumerator SetTargets(List<GameObject> enemies,
Vector3 target)
    {
        for (int i = 0; i < enemies.Count; i++)
        {
            yield return new WaitForSeconds(1);
            enemies[i].GetComponent<Enemy>().SetTarget(target);

        }
    }
}
```

Файл CommonEnemy.cs

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

public class CommonEnemy : Enemy
{
    private void Awake()
    {
        _transform = GetComponent<Transform>();
        _core = FindObjectOfType<GameCore>();
        _target = _transform.position;
    }
    private void FixedUpdate()
    {
```

```
        if(_health <= 0)
        {
            Destroy();
        }
        if (_target != _transform.position)
        {
            _transform.position =
Vector3.MoveTowards(_transform.position, _target, _speed *
Time.deltaTime);
        }
    }

    public override void TakeDamage(float damage)
    {
        _health -= damage;
    }
    public override void Destroy()
    {
        _core.AddMoney(_goldForDeath);
        _core.DeleteEnemy(gameObject);
        Destroy(gameObject);
    }
    public override int GetTargetNumber()
    {
        return ++_targetNumber;
    }
    public override void SetTarget(Vector3 target)
    {
        _target = target;
    }
    public override void MakeSlower(float value)
    {
        _speed -= value;
    }
}
```

Файл Enemy.cs

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

public abstract class Enemy : MonoBehaviour
{
    [SerializeField] protected float _health;
    [SerializeField] protected float _speed;
    [SerializeField] protected int _goldForDeath;

    protected int _targetNumber = 0;
    protected Vector3 _target;
    protected Transform _transform;
```

```
    protected GameCore _core;

    public abstract void TakeDamage(float damage);
    public abstract int GetTargetNumber();
    public abstract void SetTarget(Vector3 target);
    public abstract void MakeSlower(float value);
    public abstract void Destroy();
}
```

Файл CameraMovement.cs

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

public class CameraMovement : MonoBehaviour
{
    [SerializeField] private Transform _target;
    [SerializeField] private int _scrollSpeed = 1;
    [SerializeField] private float _mouseSensitivity = 1.0f;
    [SerializeField] private float _distanceFromTarget = 0.0f;

    private Camera _camera;
    private float _rotation;

    private void Awake()
    {
        _camera = gameObject.GetComponent<Camera>();
    }

    void Update()
    {
        if (Time.timeScale != 0)
        {
            if (Input.GetMouseButton(0))
            {
                _rotation += Input.GetAxis("Mouse X") *
_mouseSensitivity;

                transform.localEulerAngles = new Vector3(45,
_rotation, 0);

                transform.position = _target.position -
transform.forward * _distanceFromTarget;
            }
            _camera.fieldOfView -= Input.GetAxis("Mouse
ScrollWheel") * _scrollSpeed;
            _camera.fieldOfView = Mathf.Clamp(_camera.fieldOfView,
20, 90);
        }
```

```
    }
}
```

Файл GameCore.cs

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

public class GameCore : MonoBehaviour
{
    private MainUI _mainUI;
    private Map _map;
    private RoundManager _round;

    private void Awake()
    {
        _mainUI = GetComponent<MainUI>();
        _round = GetComponent<RoundManager>();

        _map = FindObjectOfType<Map>();

        StartGame();
    }

    //-------------------------------------------------------
    //MainUI
    public void EnableBuyUI(int x, int z)
    {
        _mainUI.EnableBuyUI(x, z);
    }
    public void DisableBuyUI()
    {
        _mainUI.DisableBuyUI();
    }

    public void AddMoney(int value)
    {
        _mainUI.AddMoney(value);
    }
    public void SpendMoney(int value)
    {
        _mainUI.SpendMoney(value);
    }
    public bool IsEnoughtMoney(int value)
    {
        return _mainUI.IsEnoughMoney(value);
    }

    public void IncreaseProgress(float value)
```

```
    {
        _mainUI.IncreaseProgress(value);
    }

    //-------------------------------------------------------
    //Map
    public void PlaceTower(int x, int z, GameObject prefab)
    {
        _map.PlaceMapUnit(x,z,prefab);
    }
    public void SpawnEnemy(int amount, GameObject _enemyPrefab,
Vector3 target)
    {
        _map.SpawnEnemy(amount, _enemyPrefab, target);
    }
    public Dictionary<GameObject, List<Edge>> GetGraph()
    {
        return _map.GetGraph();
    }
    public List<GameObject> GetVertexes()
    {
        return _map.GetVertexes();
    }
    public void DeleteBase()
    {
        _map.DeleteBase();
    }

    //-------------------------------------------------------
    //RoundManager
    public Vector3 GetEnemyTarget(int number)
    {
        return _round.GetEnemyTarget(number);
    }
    public void DeleteEnemy(GameObject enemy)
    {
        _round.DeleteEnemy(enemy);
    }

    //-------------------------------------------------------
    //GameCore
    public void StartGame()
    {
        StartCoroutine(StartGameCoroutine());
    }
    private IEnumerator StartGameCoroutine()
    {
        //_map.CreateMap();
        yield return new WaitForSeconds(3);
        _round.CreateWay();
        yield return new WaitForSeconds(5);
```

```
        _round.Enemies =
_map.SpawnEnemy(_round.GetAmountEnemies(),
_round.GetEnemyPrefab(), _round.GetEnemyTarget());
        _round.StartWaitingEndRound();
    }

    public void StartNewRound()
    {
        StartCoroutine(NewRoundCoroutine());
    }
    private IEnumerator NewRoundCoroutine()
    {
        DeleteRoads();
        yield return new WaitForSeconds(3);
        _map.RemakeGraph();
        yield return new WaitForSeconds(3);
        _round.CreateWay();
        yield return new WaitForSeconds(3);
        _round.Enemies =
_map.SpawnEnemy(_round.GetAmountEnemies(),
_round.GetEnemyPrefab(), _round.GetEnemyTarget());
        _round.StartWaitingEndRound();
    }
    private void DeleteRoads()
    {
        GameObject[] roads =
GameObject.FindGameObjectsWithTag("road");
        for (int i = 0; i < roads.Length; i++)
        {
            Destroy(roads[i]);
        }
    }
}
```

Файл RoundManager.cs

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

public class RoundManager : MonoBehaviour
{
    [SerializeField] private int _amountEnemies;
    [SerializeField] private GameObject _enemyPrefab;

    private GameCore _core;

    public List<GameObject> Enemies;
    private List<GameObject> _way;
```

```
    private void Awake()
    {
        _core = GetComponent<GameCore>();

        Enemies = new List<GameObject>();
        _way = new List<GameObject>();
    }

    public void StartWaitingEndRound()
    {
        StartCoroutine(WaitEndRoundCoroutine());
    }
    private IEnumerator WaitEndRoundCoroutine()
    {
        while(Enemies.Count > 0)
        {
            yield return new WaitForEndOfFrame();
        }
        _amountEnemies += 10;
        _core.IncreaseProgress(0f);
        _core.DeleteBase();
        _way = new List<GameObject>();
        _core.StartNewRound();
    }

    public void CreateWay()
    {
        Dijkstra way = new Dijkstra(_core.GetGraph(),
_core.GetVertexes());
        way.Algorithm();
        _way = way.GetAnswer();
    }

    public int GetAmountEnemies()
    {
        return _amountEnemies;
    }
    public GameObject GetEnemyPrefab()
    {
        return _enemyPrefab;
    }
    public Vector3 GetEnemyTarget(int number = 0)
    {
        return _way[number].transform.position;
    }

    public void DeleteEnemy(GameObject enemy)
    {
        for (int i = 0; i < Enemies.Count; i++)
        {
            if(Enemies[i] == enemy)
            {
```

```
                Enemies.RemoveAt(i);
                break;
            }
        }
        _core.IncreaseProgress((1.0f/(float)_amountEnemies));
    }
}
```

Файл Cell.cs

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

public class Cell : MonoBehaviour, IPointerClickHandler
{
    [SerializeField] private int _x;
    [SerializeField] private int _z;
    [SerializeField] private GameCore _core;

    private void Awake()
    {
        _x = (int)GetComponent<Transform>().position.x;
        _z = (int)GetComponent<Transform>().position.z;

        _core = FindObjectOfType<GameCore>();
    }

    public void OnPointerClick(PointerEventData eventData)
    {
        _core.EnableBuyUI(_x, _z);
    }
}
```

Файл Dijkstra.cs

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

public class Dijkstra
{
    struct Priority
    {
        public GameObject Vertex;
        public float PriorityValue;
    }
    private int Comparator(Priority a, Priority b)
    {
```

```
        if (a.PriorityValue == -1)
        {
            return 1;
        }
        if (b.PriorityValue == -1)
        {
            return -1;
        }
        if (a.PriorityValue > b.PriorityValue)
        {
            return 1;
        }
        else
        {
            return -1;
        }
    }
    private Dictionary<GameObject, List<Edge>> _graph;
    private List<Priority> _priorityList;
    private List<GameObject> _vertexes;
    private Dictionary<GameObject, GameObject> _theWay;
    private List<GameObject> _answer;
    private GameObject _start;
    private GameObject _end;

    public Dijkstra(Dictionary<GameObject, List<Edge>> graph,
List<GameObject> vertexes)
    {
        _theWay = new Dictionary<GameObject, GameObject>();
        _graph = new Dictionary<GameObject, List<Edge>>();
        _vertexes = new List<GameObject>();
        _vertexes = vertexes;
        _graph = graph;
        _start = vertexes[0];
        _end = vertexes[vertexes.Count - 1];
        _answer = new List<GameObject>();
        _priorityList = new List<Priority>();
        _priorityList.Add(new Priority { Vertex = vertexes[0],
PriorityValue = 0 });
        for (int i = 1; i < _vertexes.Count; i++)
        {
            _priorityList.Add(new Priority { Vertex =
_vertexes[i], PriorityValue = -1 });
        }
    }

    public bool Algorithm()
    {
        while (_priorityList.Count != 0)
        {
            _priorityList.Sort(Comparator);
```

```
            if (_priorityList[0].Vertex == _end)
            {
                GameObject key = _end;
                while (key != _start)
                {
                    _answer.Add(key);
                    key = _theWay[key];
                };
                _answer.Reverse();
                return true;
            }
            else
            {
                List<Edge> edges =
_graph[_priorityList[0].Vertex];
                foreach (Edge edge in edges)
                {
                    for (int i = 1; i < _priorityList.Count; i++)
                    {
                        if (edge.GetEnd() ==
_priorityList[i].Vertex)
                        {
                            if (((edge.GetDistance() +
_priorityList[0].PriorityValue) < _priorityList[i].PriorityValue)
|| (_priorityList[i].PriorityValue == -1))
                            {
                                _priorityList[i] = new Priority
{ Vertex = edge.GetEnd(), PriorityValue = edge.GetDistance() +
_priorityList[0].PriorityValue};
                                _theWay[_priorityList[i].Vertex] =
_priorityList[0].Vertex;
                            }
                        }
                    }
                }
                _priorityList.RemoveAt(0);
            }
        }
        return false;
    }

    public List<GameObject> GetAnswer()
    {
        return _answer;
    }
}
```

Файл Edge.cs

```
using System.Collections;
```

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

public class Edge
{
    private GameObject _start;
    private GameObject _end;
    private float _distance;

    public Edge(GameObject start, GameObject end, float distance)
    {
        _start = start;
        _end = end;
        _distance = distance;
    }

    public GameObject GetStart() {
        return _start;
    }

    public GameObject GetEnd()
    {
        return _end;
    }

    public float GetDistance() {
        return _distance;
    }
}
```

Файл Graph.cs

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

public class Graph : MonoBehaviour
{
    private Dictionary<GameObject, List<Edge>> _graph;
    private GameObject _start;
    private GameObject _end;

    private void Awake()
    {
        _graph = new Dictionary<GameObject, List<Edge>>();
    }

    public void Add(GameObject vertex, List<Edge> edges)
    {
        _graph[vertex] = new List<Edge>();
        _graph.Add(vertex, edges);
```

```
    }
}
```

Файл Map.cs

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


public class Map : MonoBehaviour
{
    [SerializeField] private GameObject _cellPrefab;
    [SerializeField] private GameObject _mainBasePrefab;
    [SerializeField] private GameObject _minorBasePrefab;
    [SerializeField] private GameObject _enemySpawnPrefab;
    [SerializeField] private GameObject _roadSpawn;

    [SerializeField] private int _width;
    [SerializeField] private int _height;
    [SerializeField] private int _amountLines;

    private Dictionary<GameObject, List<Edge>> _graph;
    private List<List<GameObject>> _map;
    private List<GameObject> _graphVertexes;


    private void Awake()
    {
        _map = new List<List<GameObject>>();
        _graphVertexes = new List<GameObject>();
        _graph = new Dictionary<GameObject, List<Edge>>();

        for (int i = 0; i < _height; i++)
        {
            _map.Add(new List<GameObject>());
            for (int j = 0; j < _width; j++)
            {
                _map[i].Add(Instantiate(_cellPrefab, new
Vector3(j, 0, i), Quaternion.identity));
            }
        }

        SpawnBases();
        PlaceEnemySpawn();
        GameObject temp = _graphVertexes[0];
        _graphVertexes[0] = _graphVertexes[_graphVertexes.Count -
1];
        _graphVertexes[_graphVertexes.Count - 1] = temp;
        //_graph = CreateGraph();
        RemakeGraph();
```

```
    }

    public void PlaceMapUnit(int x, int z, GameObject unit)
    {
        DeleteMapUnit(x, z);
        _map[z].Insert(x, Instantiate(unit, new Vector3(x, 0, z),
unit.transform.rotation));
    }
    private void PlaceMainBase()
    {
        _graphVertexes.Add(Instantiate(_mainBasePrefab, new
Vector3(_width - 4, 0, 2), _mainBasePrefab.transform.rotation));
    }
    private void PlaceMinorBase(int minXPos, int minZPos)
    {
        int x = Random.Range(minXPos + 3, (minXPos + _width / 4) -
3);
        int z = Random.Range(minZPos + 3, (minZPos + _height / 2)
- 3);
        _graphVertexes.Add(Instantiate(_minorBasePrefab, new
Vector3(x, 0, z), _minorBasePrefab.transform.rotation));
    }
    private void PlaceEnemySpawn()
    {
        _graphVertexes.Add(Instantiate(_enemySpawnPrefab, new
Vector3(2, 0, _height - 4),
_enemySpawnPrefab.transform.rotation));
    }
    private void DeleteMapUnit(int x, int z)
    {
        Destroy(_map[z][x]);
        _map[z].RemoveAt(x);
    }

    private void SpawnBases()
    {
        int stepForIterationX = _width / (_width / 10);
        int stepForIterationY = _height / 2;
        PlaceMainBase();
        for (int i = 0; i < _width - stepForIterationX; i +=
stepForIterationX)
        {
            for (int j = 0; j < _height; j += stepForIterationY)
            {
                if (i == 0 && j == stepForIterationY)
                {
                    continue;
                }
                PlaceMinorBase(i, j);
            }
        }
    }
```

```
    public List<GameObject> SpawnEnemy(int amount, GameObject
_enemyPrefab, Vector3 target)
    {
        return
_graphVertexes[0].GetComponent<EnemySpawn>().SpawnEnemy(amount,
_enemyPrefab, target);
    }


    public void RemakeGraph()
    {
        _graph = CreateGraph();
    }
    public void DeleteBase()
    {
        for (int i = 0; i < _graphVertexes.Count; i++)
        {
            if(_graphVertexes[i].GetComponent<Base>().GetHealth()
<= 0 )
            {
                _graphVertexes[i].GetComponent<Base>().Destroy();
                _graphVertexes.RemoveAt(i);
                --i;
            }
        }
    }
    public Dictionary<GameObject, List<Edge>> CreateGraph()
    {
        Dictionary<GameObject, List<Edge>> temp = new
Dictionary<GameObject, List<Edge>>();
        List<Edge> edges = new List<Edge>();
        if (_graphVertexes.Count > 4)
        {
            edges = new List<Edge>();
            for (int j = 1; j < 4; j++)
            {
                AddEdge(0, j, edges);
            }
            temp.Add(_graphVertexes[0], edges);
        }
        else
        {
            edges = new List<Edge>();
            for (int j = 1; j < _graphVertexes.Count-1; j++)
            {
                AddEdge(0, j, edges);
            }
            temp.Add(_graphVertexes[0], edges);
        }
        for (int i = 2; i < _graphVertexes.Count - 2; i++)
        {
            edges = new List<Edge>();
```

```
            AddEdge(i, i + 1, edges);
            AddEdge(i, i + 2, edges);
            temp.Add(_graphVertexes[i], edges);
        }
        edges = new List<Edge>();
        AddEdge(1, 2, edges);
        temp.Add(_graphVertexes[1], edges);
       if (_graphVertexes[1] != _graphVertexes[_graphVertexes.Count
- 2])
        {
            edges = new List<Edge>();
            AddEdge(_graphVertexes.Count - 2, _graphVertexes.Count
- 1, edges);
            temp.Add(_graphVertexes[_graphVertexes.Count - 2],
edges);
        }
        return temp;
    }
    private void AddEdge(int index1, int index2, List<Edge> edges)
    {
        Edge temp = new Edge(_graphVertexes[index1],
_graphVertexes[index2], GetDistance(_graphVertexes[index1],
_graphVertexes[index2]));
        edges.Add(temp);
        PrintRoad(_graphVertexes[index1].transform.position,
_graphVertexes[index2].transform.position);
    }
    private void PrintRoad(Vector3 vertex1, Vector3 vertex2)
    {
        GameObject temp = Instantiate(_roadSpawn, vertex1,
Quaternion.identity);
        temp.GetComponent<RoadSpawn>().SetTarget(vertex2);
    }


    private float GetDistance(GameObject vertex1, GameObject
vertex2)
    {
        return Vector3.Distance(vertex1.transform.position,
vertex2.transform.position);
    }
    public Dictionary<GameObject, List<Edge>> GetGraph()
    {
        return _graph;
    }
    public List<GameObject> GetVertexes()
    {
        return _graphVertexes;
    }
}
```

Файл RoadSpawn.cs

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

public class RoadSpawn : MonoBehaviour
{
    [SerializeField] private GameObject _roadPrefab;

    private Vector3 _target;
    private Transform _transform;

    private float _timer = 0;
    private float _time = 0.12f;


    private void Awake()
    {
        _transform = GetComponent<Transform>();
    }

    private void FixedUpdate()
    {
        if(_target != null)
        {
            if(_transform.position != _target)
            {
                _transform.position =
Vector3.MoveTowards(_transform.position, _target, 0.2f);
                if(_timer >= _time)
                {
                    Instantiate(_roadPrefab, _transform.position,
Quaternion.identity);
                    _timer = 0;
                }
                else
                {
                    _timer += Time.deltaTime;
                }

            }
            else
            {

                Destroy(gameObject);
            }
        }
    }

    public void SetTarget(Vector3 target)
    {
```

```
        _target = target;
    }
}
```

Файл AttackUnit.cs

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

public abstract class AttackUnit : MonoBehaviour
{
    [SerializeField] protected float _defualtSpeed;
    [SerializeField] protected float _speedBoost;
    [SerializeField] protected float _damage;

    protected float _speed;

    protected Vector3 _startPosition;
    protected Transform _transform;
    protected Enemy _target;

    public abstract void SetTarget(Enemy target);
    public abstract void Upgrade(float upgradeScale);
    protected abstract void UnitDestroy();

}
```

Файл CommonAttackUnit.cs

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

public class CommonAttackUnit : AttackUnit
{
    private void Awake()
    {
        _transform = GetComponent<Transform>();

        _startPosition = _transform.position;
        _speed = _defualtSpeed;
    }
    private void FixedUpdate()
    {
        if(_target != null)
        {
```

```
            _transform.position =
Vector3.MoveTowards(_transform.position,
_target.transform.position, _speed*Time.deltaTime);
            _speed += _speedBoost;
        }
        else
        {
            if (this.gameObject.activeInHierarchy)
            {
                UnitDestroy();
            }
        }
    }
    private void OnTriggerEnter(Collider other)
    {
        if(other.gameObject.GetComponent<Enemy>() == _target)
        {
            _target.TakeDamage(_damage);
            UnitDestroy();
        }
    }

    public override void SetTarget(Enemy target)
    {
        _target = target;
    }
    public override void Upgrade(float upgradeScale)
    {
        _damage *= upgradeScale;
        _speed *= upgradeScale;
    }
    protected override void UnitDestroy()
    {
        _target = null;
        _speed = _defualtSpeed;
        _transform.position = _startPosition;
        gameObject.SetActive(false);
    }
}
```

Файл EarthAttackUnit.cs

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

public class EarthAttackUnit : AttackUnit
{
    private void Awake()
    {
        _transform = GetComponent<Transform>();
```

```
        _startPosition = _transform.position;
        _speed = _defualtSpeed;
    }
    private void FixedUpdate()
    {
        if (_target != null)
        {
            _transform.position =
Vector3.MoveTowards(_transform.position,
_target.transform.position, _speed * Time.deltaTime);
            _speed += _speedBoost;
        }
        else
        {
            if (this.gameObject.activeInHierarchy)
            {
                UnitDestroy();
            }
        }
    }
    private void OnTriggerEnter(Collider other)
    {
        Enemy temp = other.gameObject.GetComponent<Enemy>();
        if (temp != null)
        {
            temp.TakeDamage(_damage);
            if (temp == _target)
            {
                UnitDestroy();
            }

        }
    }

    public override void SetTarget(Enemy target)
    {
        _target = target;
    }
    public override void Upgrade(float upgradeScale)
    {
        _damage *= upgradeScale;
        _speed *= upgradeScale;
    }
    protected override void UnitDestroy()
    {
        _target = null;
        _speed = _defualtSpeed;
        _transform.position = _startPosition;
        gameObject.SetActive(false);
    }
}
```

Файл FireAttackUnit.cs

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

public class FireAttackUnit : AttackUnit
{
    private void Awake()
    {
        _transform = GetComponent<Transform>();

        _startPosition = _transform.position;
        _speed = _defualtSpeed;
    }
    private void FixedUpdate()
    {
        if (_target != null)
        {
            _transform.position =
Vector3.MoveTowards(_transform.position,
_target.transform.position, _speed * Time.deltaTime);
            _speed += _speedBoost;
        }
        else
        {
            if (this.gameObject.activeInHierarchy)
            {
                UnitDestroy();
            }
        }
    }
    private void OnTriggerEnter(Collider other)
    {
        if (other.gameObject.GetComponent<Enemy>() == _target)
        {
            _target.TakeDamage(_damage);
            UnitDestroy();
        }
    }

    public override void SetTarget(Enemy target)
    {
        _target = target;
    }
    public override void Upgrade(float upgradeScale)
    {
        _damage *= upgradeScale;
        _speed *= upgradeScale;
    }
```

```
    protected override void UnitDestroy()
    {
        _target = null;
        _speed = _defualtSpeed;
        _transform.position = _startPosition;
        gameObject.SetActive(false);
    }
}
```

Файл WaterAttackUnit.cs

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

public class WaterAttackUnit : AttackUnit
{
    [SerializeField] private float _deceleration;
    private void Awake()
    {
        _transform = GetComponent<Transform>();

        _startPosition = _transform.position;
        _speed = _defualtSpeed;
    }
    private void FixedUpdate()
    {
        if (_target != null)
        {
            _transform.position =
Vector3.MoveTowards(_transform.position,
_target.transform.position, _speed * Time.deltaTime);
            _speed += _speedBoost;
        }
        else
        {
            if (this.gameObject.activeInHierarchy)
            {
                UnitDestroy();
            }
        }
    }
    private void OnTriggerEnter(Collider other)
    {
        Enemy temp = other.gameObject.GetComponent<Enemy>();
        if (temp != null)
        {
            temp.TakeDamage(_damage);
            temp.MakeSlower(_deceleration);
            if(temp == _target)
            {
```

```
                UnitDestroy();
            }

        }
    }

    public override void SetTarget(Enemy target)
    {
        _target = target;
    }
    public override void Upgrade(float upgradeScale)
    {
        _damage *= upgradeScale;
        _speed *= upgradeScale;
    }
    protected override void UnitDestroy()
    {
        _target = null;
        _speed = _defualtSpeed;
        _transform.position = _startPosition;
        gameObject.SetActive(false);
    }
}
```

Файл CommonTower.cs

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


public class CommonTower : Tower
{
    private void Awake()
    {
        _transform = GetComponent<Transform>();
        _collider = GetComponent<SphereCollider>();
        _attackUnits = new List<AttackUnit>();

        _collider.radius = _range;

        for (int i = 0; i < _amountAttackUnits; i++)
        {
            _attackUnits.Add(Instantiate(_prefab,
                                         new
Vector3(_transform.position.x, _transform.position.y + 3,
_transform.position.z),
                                         Quaternion.identity));

            _attackUnits[i].gameObject.SetActive(false);
        }
```

```
    }
    private void Update()
    {
        if(_target != null)
        {
            if(_attackTimer <= 0f)
            {
                Attack();
            }
            else
            {
                _attackTimer -= Time.deltaTime;
            }
        }

    }
    private void OnTriggerStay(Collider other)
    {
        if (_target == null &&
other.gameObject.GetComponent<Enemy>())
        {
            _target = other.gameObject.GetComponent<Enemy>();
        }
    }
    private void OnTriggerExit(Collider other)
    {
        if (_target == other.gameObject.GetComponent<Enemy>())
        {
            _target = null;
        }
    }

    protected override void Attack()
    {
        for (int i = 0; i < _amountAttackUnits; i++)
        {
            if (!(_attackUnits[i].gameObject.activeInHierarchy))
            {
                _attackUnits[i].gameObject.SetActive(true);
                _attackUnits[i].SetTarget(_target);
                _attackTimer = _attackSpeed;
                break;
            }
            else if(i == _amountAttackUnits - 1)
            {
                _attackUnits.Add(Instantiate(_prefab,
                                             new
Vector3(_transform.position.x, _transform.position.y + 3,
_transform.position.z),

Quaternion.identity));
                ++_amountAttackUnits;
```

```
                _attackUnits[_amountAttackUnits -
1].gameObject.SetActive(false);
            }
        }

    }
    protected override void Upgrade()
    {
        _range *= _upgradeScale;
        for (int i = 0; i < _amountAttackUnits; i++)
        {
            _attackUnits[i].Upgrade(_upgradeScale);
        }
    }
}
```

Файл EarthTower.cs

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

public class EarthTower : Tower
{
    private void Awake()
    {
        _transform = GetComponent<Transform>();
        _collider = GetComponent<SphereCollider>();
        _attackUnits = new List<AttackUnit>();

        _collider.radius = _range;

        for (int i = 0; i < _amountAttackUnits; i++)
        {
            _attackUnits.Add(Instantiate(_prefab,
                                         new
Vector3(_transform.position.x, _transform.position.y + 3,
_transform.position.z),
                                         Quaternion.identity));

            _attackUnits[i].gameObject.SetActive(false);
        }
    }
    private void Update()
    {
        if (_target != null)
        {
            if (_attackTimer <= 0f)
            {
                Attack();
            }
```

```
            else
            {
                _attackTimer -= Time.deltaTime;
            }
        }

    }
    private void OnTriggerStay(Collider other)
    {
        if (_target == null &&
other.gameObject.GetComponent<Enemy>() && other != null)
        {
            _target = other.gameObject.GetComponent<Enemy>();
        }
    }
    private void OnTriggerExit(Collider other)
    {
        if (_target != null && _target ==
other.gameObject.GetComponent<Enemy>())
        {
            _target = null;
        }
    }

    protected override void Attack()
    {
        for (int i = 0; i < _amountAttackUnits; i++)
        {
            if (!(_attackUnits[i].gameObject.activeInHierarchy))
            {
                _attackUnits[i].gameObject.SetActive(true);
                _attackUnits[i].SetTarget(_target);
                _attackTimer = _attackSpeed;
                break;
            }
            else if (i == _amountAttackUnits - 1)
            {
                _attackUnits.Add(Instantiate(_prefab,
                                             new
Vector3(_transform.position.x, _transform.position.y + 3,
_transform.position.z),

Quaternion.identity));
                ++_amountAttackUnits;
                _attackUnits[_amountAttackUnits -
1].gameObject.SetActive(false);
            }
        }

    }
    protected override void Upgrade()
    {
```

```
        _range *= _upgradeScale;
        for (int i = 0; i < _amountAttackUnits; i++)
        {
            _attackUnits[i].Upgrade(_upgradeScale);
        }
    }
}
```

Файл FireTower.cs

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

public class FireTower : Tower
{
    private void Awake()
    {
        _transform = GetComponent<Transform>();
        _collider = GetComponent<SphereCollider>();
        _attackUnits = new List<AttackUnit>();

        _collider.radius = _range;

        for (int i = 0; i < _amountAttackUnits; i++)
        {
            _attackUnits.Add(Instantiate(_prefab,
                                         new
Vector3(_transform.position.x, _transform.position.y + 3,
_transform.position.z),
                                         Quaternion.identity));

            _attackUnits[i].gameObject.SetActive(false);
        }
    }
    private void Update()
    {
        if (_target != null)
        {
            if (_attackTimer <= 0f)
            {
                Attack();
            }
            else
            {
                _attackTimer -= Time.deltaTime;
            }
        }

    }
    private void OnTriggerStay(Collider other)
```

```
    {
        if (_target == null &&
other.gameObject.GetComponent<Enemy>())
        {
            _target = other.gameObject.GetComponent<Enemy>();
        }
    }
    private void OnTriggerExit(Collider other)
    {
        if (_target == other.gameObject.GetComponent<Enemy>())
        {
            _target = null;
        }
    }

    protected override void Attack()
    {
        for (int i = 0; i < _amountAttackUnits; i++)
        {
            if (!(_attackUnits[i].gameObject.activeInHierarchy))
            {
                _attackUnits[i].gameObject.SetActive(true);
                _attackUnits[i].SetTarget(_target);
                _attackTimer = _attackSpeed;
                break;
            }
            else if (i == _amountAttackUnits - 1)
            {
                _attackUnits.Add(Instantiate(_prefab,
                                             new
Vector3(_transform.position.x, _transform.position.y + 3,
_transform.position.z),

Quaternion.identity));
                ++_amountAttackUnits;
                _attackUnits[_amountAttackUnits -
1].gameObject.SetActive(false);
            }
        }

    }
    protected override void Upgrade()
    {
        _range *= _upgradeScale;
        for (int i = 0; i < _amountAttackUnits; i++)
        {
            _attackUnits[i].Upgrade(_upgradeScale);
        }
    }
}
```

Файл Tower.cs

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

public abstract class Tower : MonoBehaviour
{
    [SerializeField] protected float _range;
    [SerializeField] protected float _attackSpeed;
    [SerializeField] protected float _upgradeScale;

    [SerializeField] protected int _amountAttackUnits;

    [SerializeField] protected AttackUnit _prefab;

    protected float _attackTimer = 0f;
    protected Enemy _target;
    protected Transform _transform;
    protected SphereCollider _collider;
    protected List<AttackUnit> _attackUnits;

    protected abstract void Attack();
    protected abstract void Upgrade();
}
```

Файл WaterTower.cs

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

public class WaterTower : Tower
{
    private void Awake()
    {
        _transform = GetComponent<Transform>();
        _collider = GetComponent<SphereCollider>();
        _attackUnits = new List<AttackUnit>();

        _collider.radius = _range;

        for (int i = 0; i < _amountAttackUnits; i++)
        {
            _attackUnits.Add(Instantiate(_prefab,
                                         new
Vector3(_transform.position.x, _transform.position.y + 3,
_transform.position.z),
                                         Quaternion.identity));
```

```
            _attackUnits[i].gameObject.SetActive(false);
        }
    }
    private void Update()
    {
        if (_target != null)
        {
            if (_attackTimer <= 0f)
            {
                Attack();
            }
            else
            {
                _attackTimer -= Time.deltaTime;
            }
        }

    }
    private void OnTriggerStay(Collider other)
    {
        if (_target == null &&
other.gameObject.GetComponent<Enemy>())
        {
            _target = other.gameObject.GetComponent<Enemy>();
        }
    }
    private void OnTriggerExit(Collider other)
    {
        if (_target == other.gameObject.GetComponent<Enemy>())
        {
            _target = null;
        }
    }

    protected override void Attack()
    {
        for (int i = 0; i < _amountAttackUnits; i++)
        {
            if (!(_attackUnits[i].gameObject.activeInHierarchy))
            {
                _attackUnits[i].gameObject.SetActive(true);
                _attackUnits[i].SetTarget(_target);
                _attackTimer = _attackSpeed;
                break;
            }
            else if (i == _amountAttackUnits - 1)
            {
                _attackUnits.Add(Instantiate(_prefab,
                                             new
Vector3(_transform.position.x, _transform.position.y + 3,
_transform.position.z),
```

```
Quaternion.identity));
                ++_amountAttackUnits;
                _attackUnits[_amountAttackUnits -
1].gameObject.SetActive(false);
            }
        }

    }
    protected override void Upgrade()
    {
        _range *= _upgradeScale;
        for (int i = 0; i < _amountAttackUnits; i++)
        {
            _attackUnits[i].Upgrade(_upgradeScale);
        }
    }
}
```

Файл Audio.cs

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

public class Audio : MonoBehaviour
{
    [SerializeField] private Sprite[] _switchSprites;

    private Image _switchImage;
    private int _audioState;

    void Start()
    {
        _audioState = 1;
        _switchImage = gameObject.GetComponent<Button>().image;
        _switchImage.sprite = _switchSprites[_audioState];

gameObject.GetComponent<Button>().onClick.AddListener(ChangeAudioS
tate);
    }

    private void ChangeAudioState()
    {
        _audioState = 1 - _audioState;
        AudioListener.volume = _audioState;
        _switchImage.sprite = _switchSprites[_audioState];
    }
}
```

Файл Coins.cs

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

public class Coins : MonoBehaviour
{
    [SerializeField] private int _startValue;

    private Text _text;
    private int _currentValue = 0;

    private void Awake()
    {
        _text = gameObject.GetComponent<Text>();
        _currentValue = _startValue;
    }
    private void Start()
    {
        _text.text = _startValue.ToString();
    }

    public void AddMoney(int value)
    {
        _currentValue += value;
        _text.text = _currentValue.ToString();
    }
    public void SpendMoney(int value)
    {
        _currentValue -= value;
        _text.text = _currentValue.ToString();
    }
    public bool IsEnoughMoney(int value)
    {
        return value <= _currentValue;
    }
}
```

Файл Main.cs

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

public class MainUI : MonoBehaviour
{
    private ProgressBar _progressBar;
    private BuyUI _buyUI;
```

```
    private Coins _coins;

    private void Awake()
    {
        _progressBar = FindObjectOfType<ProgressBar>();
        _buyUI = FindObjectOfType<BuyUI>();
        _coins = FindObjectOfType<Coins>();
        _buyUI.gameObject.SetActive(false);
    }

    public void IncreaseProgress(float newProgress)
    {
        _progressBar.IncreaseProgress(newProgress);
    }

    public void EnableBuyUI(int x, int z)
    {
        _buyUI.gameObject.SetActive(true);
        _buyUI.SetPosition(x, z);
    }
    public void DisableBuyUI()
    {
        _buyUI.gameObject.SetActive(false);
    }

    public void AddMoney(int value)
    {
        _coins.AddMoney(value);
    }
    public void SpendMoney(int value)
    {
        _coins.SpendMoney(value);
    }
    public bool IsEnoughMoney(int value)
    {
        return _coins.IsEnoughMoney(value);
    }
}
```

Файл Pause.cs

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

public class Pause : MonoBehaviour
{
    [SerializeField] private GameObject _menuUI;

    private bool _isPaused = false;
```

```
    private void Start()
    {
        Time.timeScale = 1;
    }
    public void PauseGame()
    {
        if (_isPaused)
        {
            _menuUI.SetActive(false);
            Time.timeScale = 1;
            _isPaused = false;
        }
        else
        {
            _menuUI.SetActive(true);
            Time.timeScale = 0;
            _isPaused = true;
        }
    }
}
```

Файл ProgressBar.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
public class ProgressBar : MonoBehaviour
{
    [SerializeField] private float _fillSpeed;

    private ParticleSystem _particleSys;
    private Slider _slider;

    private void Awake()
    {
        _slider = GetComponent<Slider>();
        _particleSys = FindObjectOfType<ParticleSystem>();
    }


    public void IncreaseProgress(float newProgress)
    {
        if(newProgress == 0f)
        {
            _slider.value = 0f;
        }
        _slider.value += newProgress;
   }
}
```

Файл SceneManager.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;
public class ScenesManager : MonoBehaviour
{
    public void ChangeScene(int sceneIndex)
    {
        SceneManager.LoadScene(sceneIndex);
    }
}
```