Урок 6

Урок 6

Сохранение данных. Миникарта

Сохранение данных. Создание миникарты.

На этом уроке:

1. Научимся сохранять данные различными способами
2. Научимся шифровать данные
3. Познакомитесь с паттерном «Репозиторий»
4. Добавим в игру мини карту и радар
5. Научимся создавать ScreenShot игрового экрана

Сохранение данных

Подготовка

Для начала определимся, где будет находиться наш файл:

```
Application.dataPath;              // Содержит путь к папке игровых данных
Application.persistentDataPath;    // Путь к постоянной директории данных
Application.streamingAssetsPath;   // Путь к папке StreamingAssets
Application.temporaryCachePath;    // Путь к временным данным/директории кэша
```

Отрефакторм класс SavedData. Добавим структуру Vector3Serializable для сохранения позиции:

```
using System;
using UnityEngine;

namespace Geekbrains
{
    [Serializable]
    public sealed class SavedData
    {
        public string Name;
        public Vector3Serializable Position;
        public bool IsEnabled;

        public override string ToString() => $"Name {Name} Position {Position}
IsVisible {IsEnabled}";
    }

    [Serializable]
    public struct Vector3Serializable
    {
        public float X;
        public float Y;
        public float Z;

        private Vector3Serializable(float valueX, float valueY, float valueZ)
        {
            X = valueX;
            Y = valueY;
            Z = valueZ;
        }
```

```
        public static implicit operator Vector3(Vector3Serializable value)
        {
            return new Vector3(value.X, value.Y, value.Z);
        }

        public static implicit operator Vector3Serializable(Vector3 value)
        {
            return new Vector3Serializable(value.x, value.y, value.z);
        }

        public override string ToString() => $" (X = {X} Y = {Y} Z = {Z})";
    }
}
```

Данный класс и структура помечены атрибутом Serializable. Это необходимо для сохранения с помощью сериализации. Более подробно мы разберем атрибуты на 8 занятии.

Интерфейс, который будут реализовывать классы, отвечающие за сохранение и загрузку данных:

```
namespace Geekbrains
{
   public interface IData<T>
   {
      void Save(T data, string path = null);
      T Load(string path = null);
   }
}
```

Текстовый документ

Для работы с файлами подключаем библиотеку System.IO:

```
using System.IO;

namespace Geekbrains
{
   public sealed class StreamData : IData<SavedData>
   {
      public void Save(SavedData data, string path = null)
      {
         if (path == null) return;
         using (var sw = new StreamWriter(path))
         {
            sw.WriteLine(data.Name);
            sw.WriteLine(data.Position.X);
            sw.WriteLine(data.Position.Y);
            sw.WriteLine(data.Position.Z);
            sw.WriteLine(data.IsEnabled);
         }
      }

      public SavedData Load(string path = null)
      {
         var result = new SavedData();
```

```
        using (var sr = new StreamReader(path))
        {
            while (!sr.EndOfStream)
            {
                result.Name = sr.ReadLine();
                result.Position.X = sr.ReadLine().TrySingle();
                result.Position.Y = sr.ReadLine().TrySingle();
                result.Position.Z = sr.ReadLine().TrySingle();
                result.IsEnabled = sr.ReadLine().TryBool();
            }
        }
        return result;
    }
  }
}
```

XML

Рассмотрим один из способов записи и чтения данных в XML:

```
using System.IO;
using System.Xml;
using UnityEngine;

namespace Geekbrains
{
   public sealed class XMLData : IData<SavedData>
   {
      public void Save(SavedData player, string path = "")
      {
         var xmlDoc = new XmlDocument();

         XmlNode rootNode = xmlDoc.CreateElement("Player");
         xmlDoc.AppendChild(rootNode);

         var element = xmlDoc.CreateElement("Name");
         element.SetAttribute("value", player.Name);
         rootNode.AppendChild(element);

         element = xmlDoc.CreateElement("PosX");
         element.SetAttribute("value", player.Position.X.ToString());
         rootNode.AppendChild(element);

         element = xmlDoc.CreateElement("PosY");
         element.SetAttribute("value", player.Position.Y.ToString());
         rootNode.AppendChild(element);

         element = xmlDoc.CreateElement("PosZ");
         element.SetAttribute("value", player.Position.Z.ToString());
         rootNode.AppendChild(element);

         element = xmlDoc.CreateElement("IsEnable");
         element.SetAttribute("value", player.IsEnabled.ToString());
         rootNode.AppendChild(element);

         XmlNode userNode = xmlDoc.CreateElement("Info");
```

```
            var attribute = xmlDoc.CreateAttribute("Unity");
            attribute.Value = Application.unityVersion;
            userNode.Attributes.Append(attribute);
            userNode.InnerText = "System Language: " +
                                 Application.systemLanguage;
            rootNode.AppendChild(userNode);

            xmlDoc.Save(path);
        }

        public SavedData Load(string path = "")
        {
            var result = new SavedData();
            if (!File.Exists(path)) return result;
            using (var reader = new XmlTextReader(path))
            {
                while (reader.Read())
                {
                    var key = "Name";
                    if (reader.IsStartElement(key))
                    {
                        result.Name = reader.GetAttribute("value");
                    }
                    key = "PosX";
                    if (reader.IsStartElement(key))
                    {
                        result.Position.X = reader.GetAttribute("value").TrySingle();
                    }
                    key = "PosY";
                    if (reader.IsStartElement(key))
                    {
                        result.Position.Y = reader.GetAttribute("value").TrySingle();
                    }
                    key = "PosZ";
                    if (reader.IsStartElement(key))
                    {
                        result.Position.Z = reader.GetAttribute("value").TrySingle();
                    }
                    key = "IsEnable";
                    if (reader.IsStartElement(key))
                    {
                        result.IsEnabled = reader.GetAttribute("value").TryBool();
                    }
                }
            }

            return result;
        }
    }
}
```

Для использования списка подключим библиотеку **System.Collections.Generic**, для работы с XML – библиотеку **System.Xml**, а для сериализации объекта – **System.Xml.Serialization**.
Содержимое файла:

```
<Player>
  <Name value="Roman" />
```

```
  <PosX value="7,86" />
  <PosY value="1" />
  <PosZ value="-11,01" />
  <IsEnable value="True" />
  <Info Unity="2019.3.0f6">System Language: Russian</Info>
</Player>
```

Этот способ очень гибок: в файле можно создать несколько групп и сохранять любые параметры.

BinarySerialization

Напишем класс для бинарной сиреализации объекта.

```
using System;
using System.IO;
using System.Runtime.Serialization.Formatters.Binary;

namespace Geekbrains
{
   public class BinarySerializationData<T> : IData<T>
   {
      private static BinaryFormatter _formatter;

      public BinarySerializationData()
      {
         _formatter = new BinaryFormatter();
      }

      public void Save(T data, string path = null)
      {
         if (data == null && !String.IsNullOrEmpty(path)) return;
         if (!typeof(T).IsSerializable) return;
         using (var fs = new FileStream(path, FileMode.Create))
         {
            _formatter.Serialize(fs, data);
         }
      }

      public T Load(string path)
      {
         T result;
         if (!File.Exists(path)) return default(T);
         using (var fs = new FileStream(path, FileMode.Open))
         {
            result = (T)_formatter.Deserialize(fs);
         }
         return result;
      }
   }
}
```

XmlSerializer

Напишем класс для xml сиреализации объекта.

```
using System;
using System.IO;
using System.Xml.Serialization;

namespace Geekbrains
{
   public class SerializableXMLData<T> : IData<T>
   {
      private static XmlSerializer _formatter;

      public SerializableXMLData()
      {
         _formatter = new XmlSerializer(typeof(T));
      }

      public void Save(T data, string path = null)
      {
         if (data == null && !String.IsNullOrEmpty(path)) return;
         using (var fs = new FileStream(path, FileMode.Create))
         {
            _formatter.Serialize(fs, data);
         }
      }

      public T Load(string path)
      {
         T result;
         if (!File.Exists(path)) return default;
         using (var fs = new FileStream(path, FileMode.Open))
         {
            result = (T)_formatter.Deserialize(fs);
         }
         return result;
      }
   }
}
```

PlayerPrefs

PlayerPrefs – это класс для работы с сохранением данных игровой сессии.

```
using UnityEngine;

namespace Geekbrains
{
   public class PlayerPrefsData : IData<SavedData>
   {
      public void Save(SavedData data, string path = null)
      {
         PlayerPrefs.SetString("Name", data.Name);
         PlayerPrefs.SetFloat("PosX", data.Position.X);
         PlayerPrefs.SetFloat("PosY", data.Position.Y);
         PlayerPrefs.SetFloat("PosZ", data.Position.Z);
         PlayerPrefs.SetString("IsEnable", data.IsEnabled.ToString());

         //----------------------------
         PlayerPrefs.Save();
```

```
    }

    public SavedData Load(string path = null)
    {
        var result = new SavedData();

        var key = "Name";
        if (PlayerPrefs.HasKey(key))
        {
            result.Name = PlayerPrefs.GetString(key);
        }

        key = "PosX";
        if (PlayerPrefs.HasKey(key))
        {
            result.Position.X = PlayerPrefs.GetFloat(key);
        }

        key = "PosY";
        if (PlayerPrefs.HasKey(key))
        {
            result.Position.Y= PlayerPrefs.GetFloat(key);
        }

        key = "PosZ";
        if (PlayerPrefs.HasKey(key))
        {
            result.Position.Z = PlayerPrefs.GetFloat(key);
        }

        key = "IsEnable";
        if (PlayerPrefs.HasKey(key))
        {
            result.IsEnabled = PlayerPrefs.GetString(key).TryBool();
        }
        return result;
    }

    public void Clear()
    {
        PlayerPrefs.DeleteAll();
    }
  }
}
```

Эта система сохранения проста в применении, но ее не рекомендуется использовать по ряду причин:

1. В операционной системе Windows **PlayerPrefs** пишет данные в реестр;
2. **PlayerPrefs** имеет лимит по объему записанной информации – 1 мб.

Json

```
using System.IO;
using UnityEngine;
```

```
namespace Geekbrains
{
   public class JsonData<T> : IData<T>
   {
      public void Save(T data, string path = null)
      {
         var str = JsonUtility.ToJson(data);
         File.WriteAllText(path, Crypto.CryptoXOR(str));
      }

      public T Load(string path = null)
      {
         var str = File.ReadAllText(path);
         return JsonUtility.FromJson<T>(Crypto.CryptoXOR(str));
      }
   }
}
```

```
{"Name":"Roman","Position":{"X":7.860000133514404,"Y":1.0,"Z":-11.010000228881836},"IsEnabled":true}
```

Паттерн «Репозиторий»

Создадим класс, который будет добавлять нужные нам объекты:

```
using System.IO;
using UnityEngine;

namespace Geekbrains
{
   public sealed class SaveDataRepository
   {
      private readonly IData<SavedData> _data;

      private const string _folderName = "dataSave";
      private const string _fileName = "data.bat";
      private readonly string _path;

      public SaveDataRepository()
      {
         if (Application.platform == RuntimePlatform.WebGLPlayer)
         {
            _data = new PlayerPrefsData();
         }
         else
         {
            _data = new JsonData<SavedData>();
         }
         _path = Path.Combine(Application.dataPath, _folderName);

      }

      public void Save(PlayerBase player)
      {
         if (!Directory.Exists(Path.Combine(_path)))
         {
```

```
            Directory.CreateDirectory(_path);
        }
        var savePlayer = new SavedData
        {
            Position = player.transform.position,
            Name = "Roman",
            IsEnabled = true
        };

        _data.Save(savePlayer, Path.Combine(_path, _fileName));
    }

    public void Load(PlayerBase player)
    {
        var file = Path.Combine(_path, _fileName);
        if (!File.Exists(file)) return;
        var newPlayer = _data.Load(file);
        player.transform.position = newPlayer.Position;
        player.name = newPlayer.Name;
        player.gameObject.SetActive(newPlayer.IsEnabled);

        Debug.Log(newPlayer);
    }
  }
}
```

Вызов сохранения будет выглядеть так:

```
using UnityEngine;

namespace Geekbrains
{
    public sealed class InputController : IExecute
    {
        private readonly PlayerBase _playerBase;
        private readonly SaveDataRepository _saveDataRepository;
        private readonly KeyCode _savePlayer = KeyCode.C;
        private readonly KeyCode _loadPlayer = KeyCode.V;

        public InputController(PlayerBase player)
        {
            _playerBase = player;

            _saveDataRepository = new SaveDataRepository();
        }

        public void Execute()
        {
            _playerBase.Move(Input.GetAxis("Horizontal"), 0.0f,
Input.GetAxis("Vertical"));

            if (Input.GetKeyDown(_savePlayer))
            {
                _saveDataRepository.Save(_playerBase);
            }
            if (Input.GetKeyDown(_loadPlayer))
```

```
            {
                _saveDataRepository.Load(_playerBase);
            }
        }
    }
}
```

Простой пример шифрования данных:

```
using System;

namespace Geekbrains
{
   public static class Crypto
   {
      public static string CryptoXOR(string text, int key = 42)
      {
         var result = String.Empty;
         foreach (var simbol in text)
         {
            result += (char)(simbol ^ key);
         }
         return result;
      }
   }
}
```

Пример использование:

```
using System.IO;
using UnityEngine;

namespace Geekbrains
{
   public class JsonData<T> : IData<T>
   {
      public void Save(T data, string path = null)
      {
         var str = JsonUtility.ToJson(data);
         File.WriteAllText(path, Crypto.CryptoXOR(str));
      }

      public T Load(string path = null)
      {
         var str = File.ReadAllText(path);
         return JsonUtility.FromJson<T>(Crypto.CryptoXOR(str));
      }
   }
}
```

После сохранения данные будут выглядит так

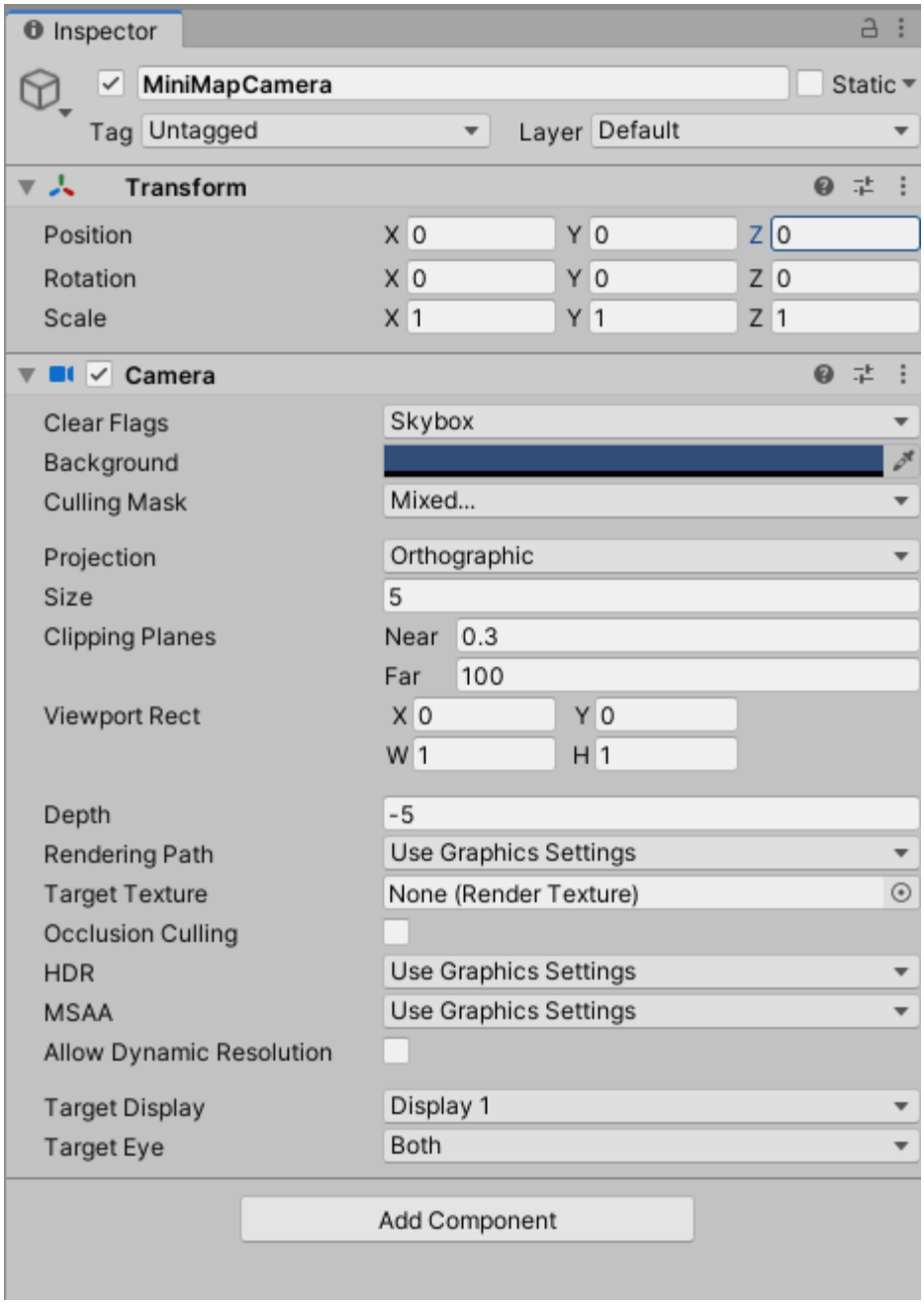
```
QdKGOxEGKDzEYC^CEDQr--sp-WcYoDKHFON^X_OW
```

Создание миникарты

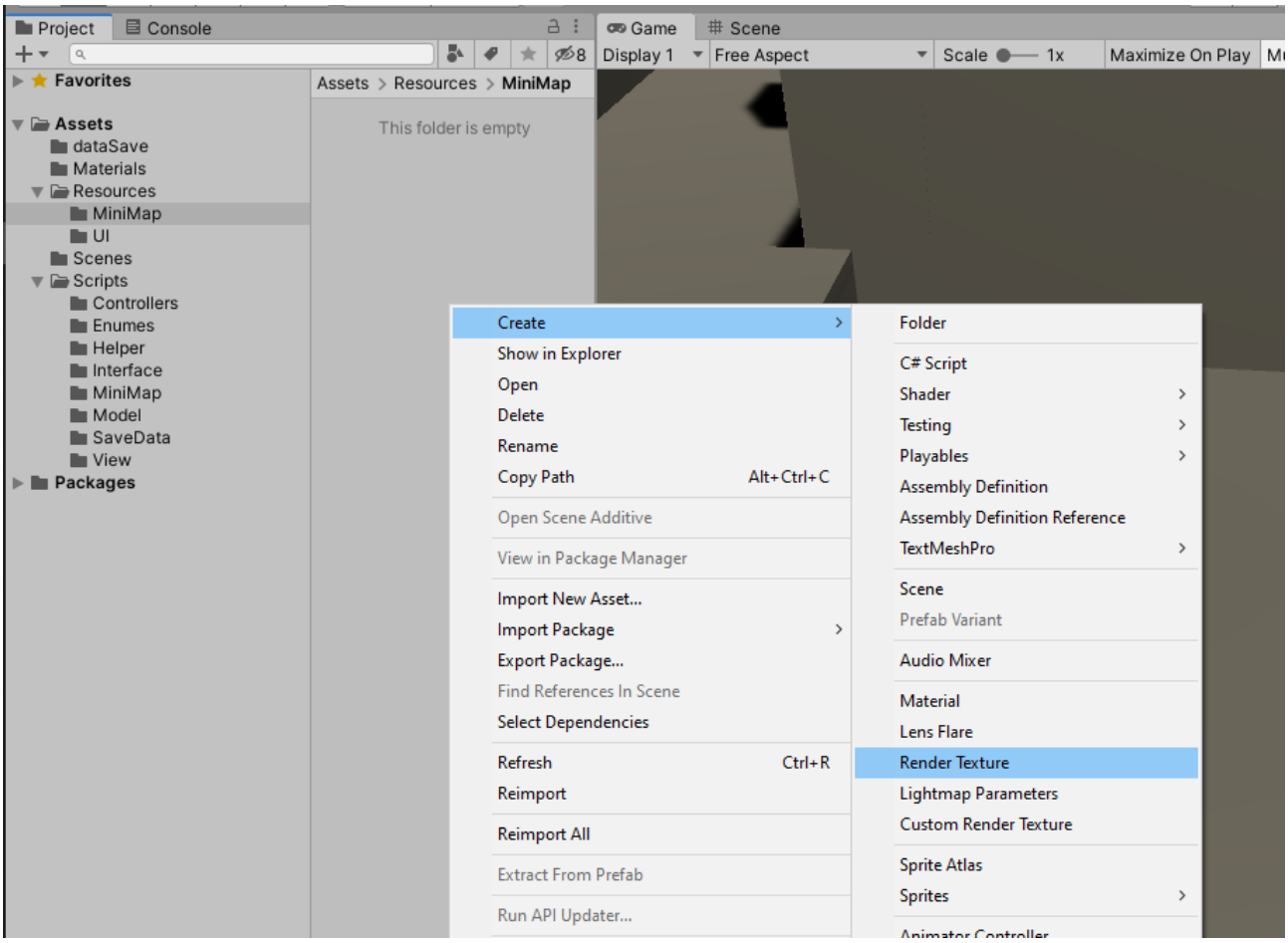
Рассмотрим 2 способа создания миникарты.

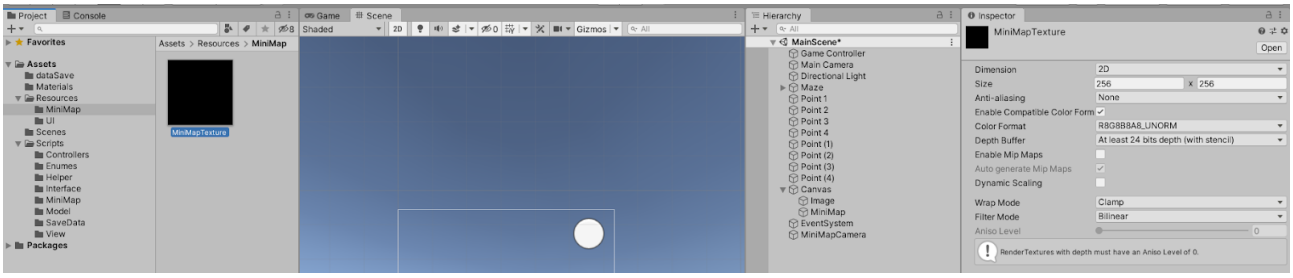
Начнем с самого простого. Добавим на сцену еще одну камеру и выставим ей координаты так, чтобы она смотрела сверху на наш персонаж. Определим ей приоритет*ниже, чем у основной камеры, и удалим с нее все лишние компоненты.

**«depth» переводится как «глубина», но мы прочтем как приоритет.*

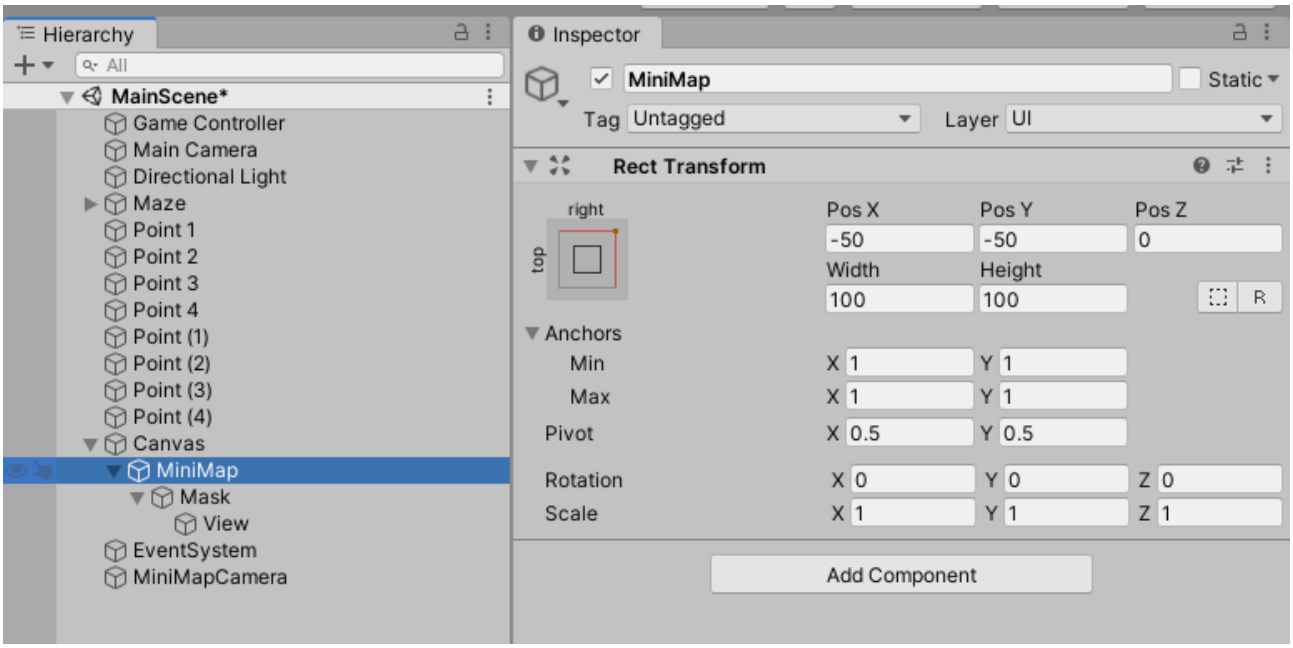


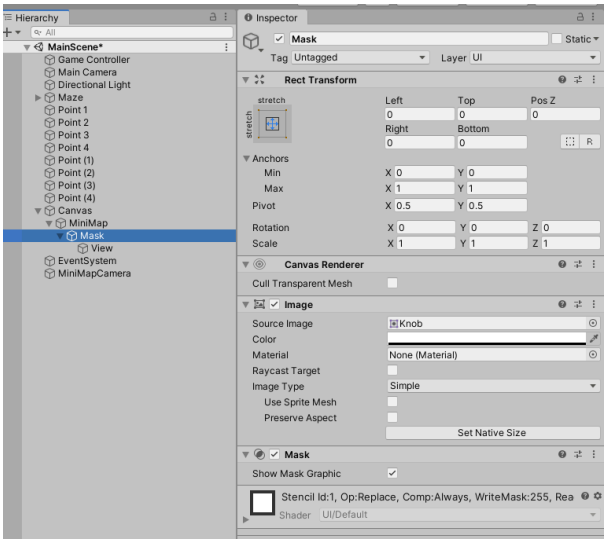
Создаем текстуру, на которую будет проецироваться изображение с камеры. Помещаем ее в папке **Resources**:

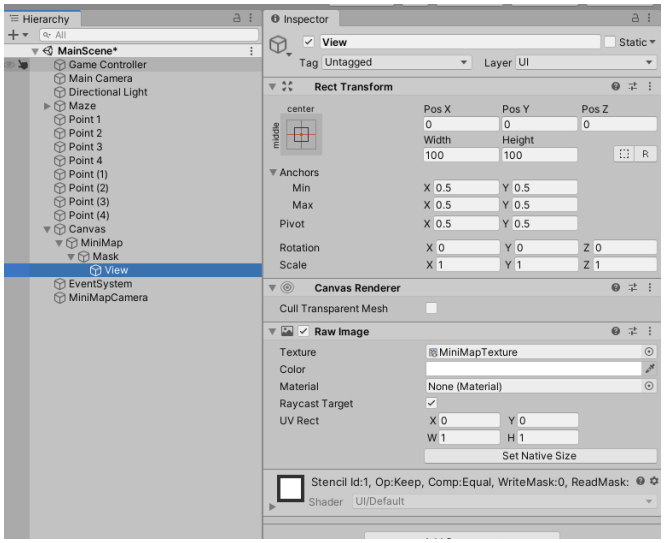


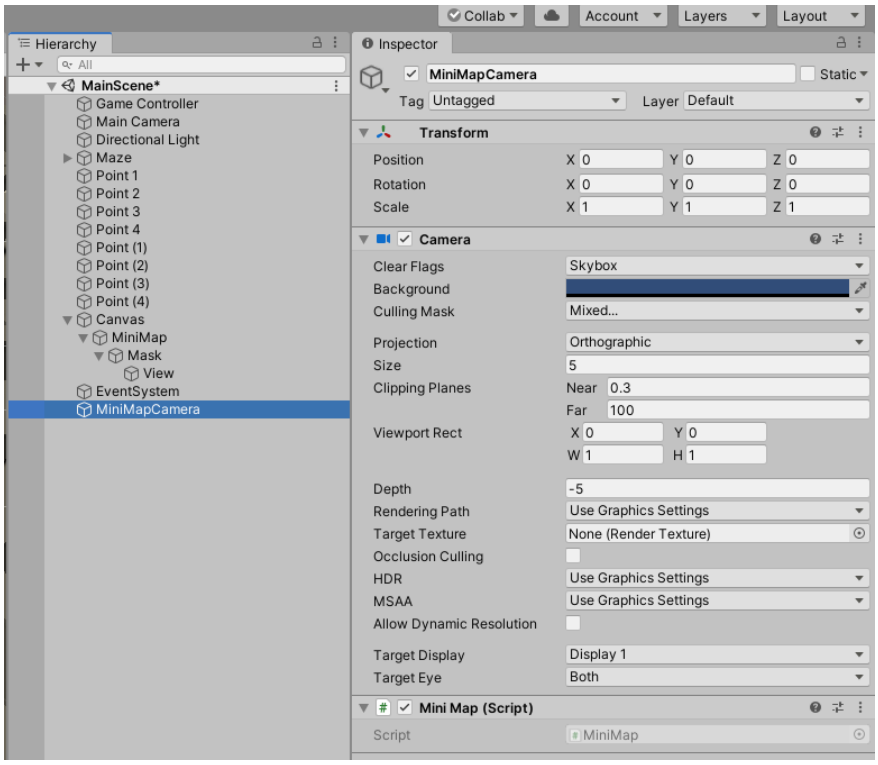


Создадим объекты для миникарты и настроим их:


Hierarchy
All
MainScene*
Game Controller
Main Camera
Directional Light
Maze
Point 1
Point 2
Point 3
Point 4
Point (1)
Point (2)
Point (3)
Point (4)
Canvas
MiniMap
Mask
View
EventSystem
MiniMapCamera
Inspector
MiniMap
Static
Tag Untagged
Layer UI
Rect Transform
right
top
Pos X
Pos Y
Pos Z
-50
-50
0
Width
Height
100
100
Anchors
Min
X 1
Y 1
Max
X 1
Y 1
Pivot
X 0.5
Y 0.5
Rotation
X 0
Y 0
Z 0
Scale
X 1
Y 1
Z 1
Add Component



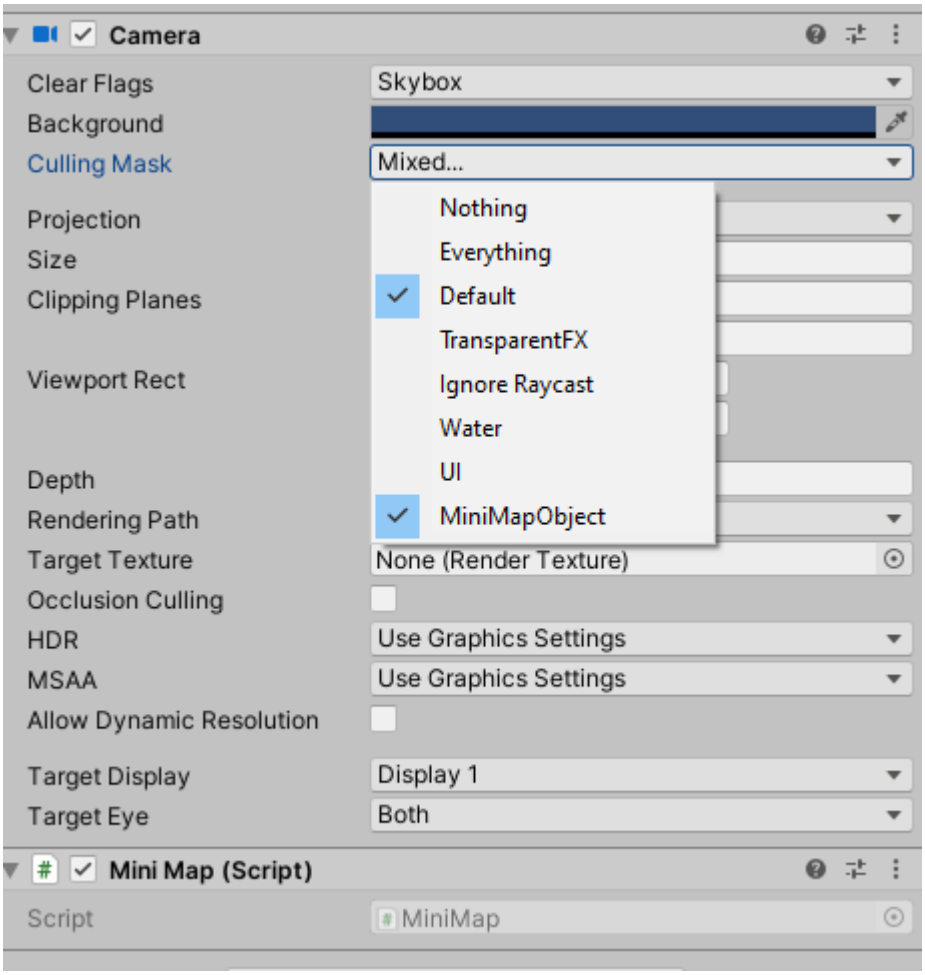




Класс для слежения камеры за игроком:

```
using UnityEngine;

namespace Geekbrains
{
    public sealed class MiniMap : MonoBehaviour
    {
        private Transform _player;
        private void Start()
        {
            _player = Camera.main.transform;
            transform.parent = null;
            transform.rotation = Quaternion.Euler(90.0f, 0, 0);
            transform.position = _player.position + new Vector3(0, 5.0f, 0);

            var rt = Resources.Load<RenderTexture>("MiniMap/MiniMapTexture");

            GetComponent<Camera>().targetTexture = rt;
        }

        private void LateUpdate()
        {
            var newPosition = _player.position;
            newPosition.y = transform.position.y;
            transform.position = newPosition;
            transform.rotation = Quaternion.Euler(90, _player.eulerAngles.y, 0);
        }
    }

}
```

Первый вариант миникарты готов. Но этот способ – один из самых ресурсоемких. Можно сократить затрачиваемые ресурсы: например, рендерить объекты только на определенном слое.

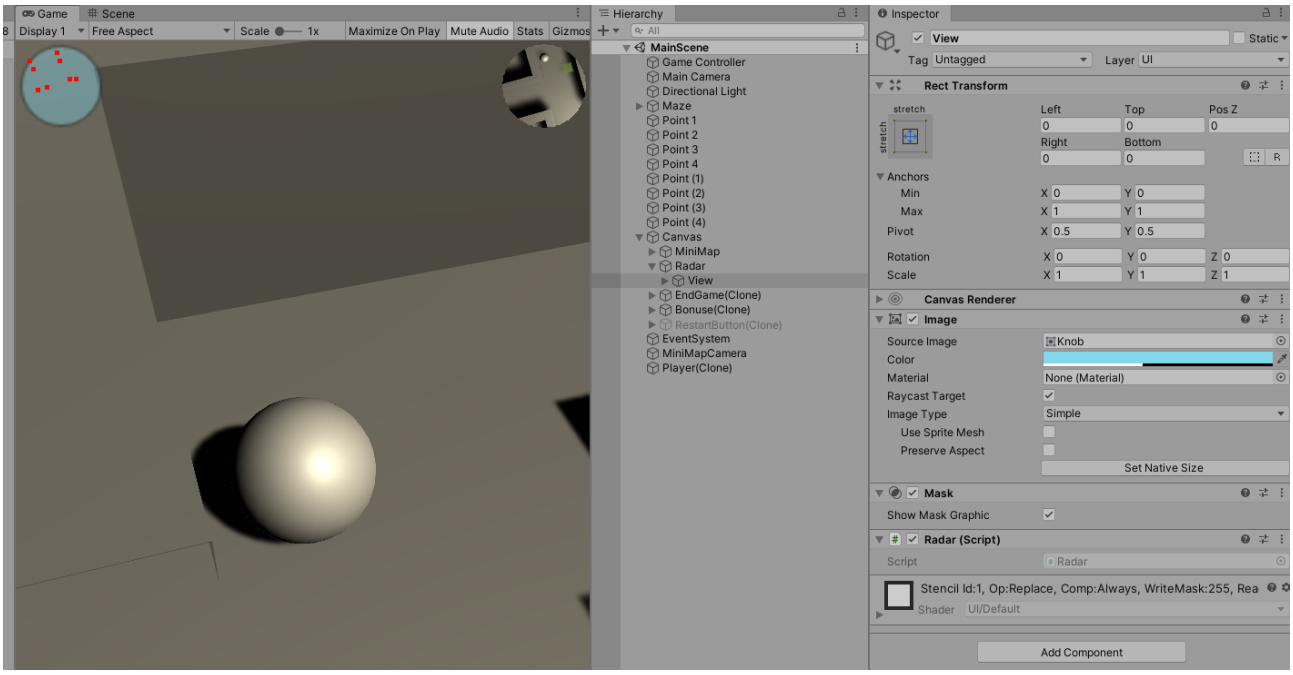


Рассмотрим еще один вариант миникарты. Создадим новый класс и назовем его **Radar**. Он будет отвечать за вывод и синхронизацию с реальными объектами значков на миникарте.

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

namespace Geekbrains
{
   public sealed class Radar : MonoBehaviour
   {
      private Transform _playerPos; // Позиция главного героя
      private readonly float _mapScale = 2;
      public static List<RadarObject> RadObjects = new List<RadarObject>();

      private void Start()
      {
         _playerPos = Camera.main.transform;
      }

      public static void RegisterRadarObject(GameObject o, Image i)
      {
```

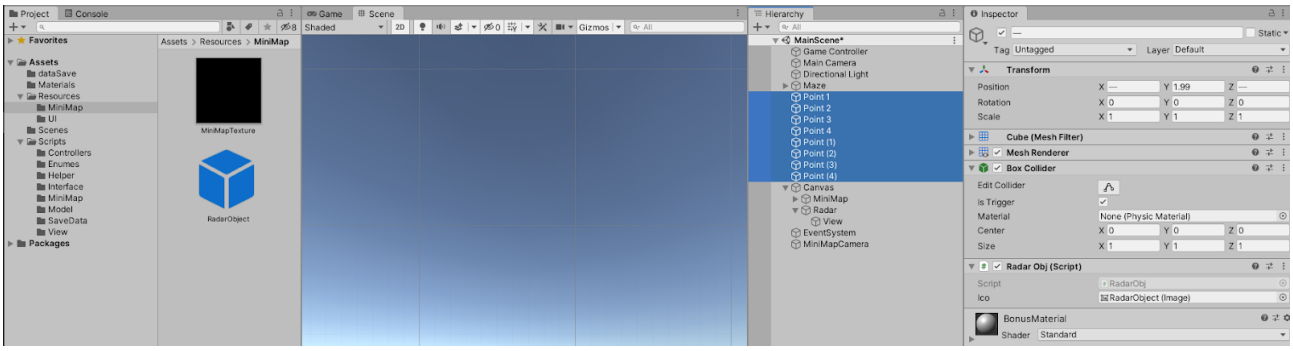
```
        Image image = Instantiate(i);
        RadObjects.Add(new RadarObject { Owner = o, Icon = image });
    }

    public static void RemoveRadarObject(GameObject o)
    {
        List<RadarObject> newList = new List<RadarObject>();
        foreach (RadarObject t in RadObjects)
        {
            if (t.Owner == o)
            {
                Destroy(t.Icon);
                continue;
            }
            newList.Add(t);
        }
        RadObjects.RemoveRange(0, RadObjects.Count);
        RadObjects.AddRange(newList);
    }

    private void DrawRadarDots() // Синхронизирует значки на миникарте с реальными
объектами
    {
        foreach (RadarObject radObject in RadObjects)
        {
            Vector3 radarPos = (radObject.Owner.transform.position -
                                _playerPos.position);
            float distToObject = Vector3.Distance(_playerPos.position,
                                    radObject.Owner.transform.position) * _mapScale;
            float deltay = Mathf.Atan2(radarPos.x, radarPos.z) * Mathf.Rad2Deg -
                           270 - _playerPos.eulerAngles.y;
            radarPos.x = distToObject* Mathf.Cos(deltay* Mathf.Deg2Rad) * -1;
            radarPos.z = distToObject* Mathf.Sin(deltay* Mathf.Deg2Rad);
            radObject.Icon.transform.SetParent(transform);
            radObject.Icon.transform.position = new Vector3(radarPos.x,
                                                    radarPos.z, 0) +
transform.position;
        }
    }

    private void Update()
    {
        if (Time.frameCount % 2 == 0)
        {
            DrawRadarDots();
        }
    }
}

public sealed class RadarObject
{
    public Image Icon;
    public GameObject Owner;
}
```

Теперь создадим объект пользовательского интерфейса **Image** – это и будет миникарта. Добавляем компоненты **Mask** и класс **Radar**.



Создадим класс **MakeRadarObject**. Добавляем его как компонент на все объекты, которые будут отображаться на миникарте:

```
using UnityEngine;
using UnityEngine.UI;

namespace Geekbrains
{
   public sealed class RadarObj : MonoBehaviour
   {
      [SerializeField] private Image _ico;

        private void OnValidate()
        {
            _ico = Resources.Load<Image>("MiniMap/RadarObject");
        }

        private void OnDisable()
      {
         Radar.RemoveRadarObject(gameObject);
      }

      private void OnEnable()
      {
         Radar.RegisterRadarObject(gameObject, _ico);
      }
   }
}
```



ScreenShot в игре

Снять снимок с экрана полезная вещь в играх, особенно если на его основе будет создана миникарта

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

namespace Geekbrains
{
   public sealed class PhotoController
   {
      private bool _isProcessed;
      private readonly string _path;
      private int _layers = 5;
      private int _resolution = 5;
      private Camera _camera;

      public PhotoController()
      {
         _path = Application.dataPath;
         _camera = Camera.main;
      }

      private IEnumerator DoTapExampleAsync()
      {
         _isProcessed = true;
         _camera.cullingMask = ~(1 << _layers);
         var sw = Screen.width;
         var sh = Screen.height;
         yield return new WaitForEndOfFrame();
         var sc = new Texture2D(sw, sh, TextureFormat.RGB24, true);
         sc.ReadPixels(new Rect(0, 0, sw, sh), 0, 0);
         var bytes = sc.EncodeToPNG();
         var filename = String.Format("{0:ddMMyyyy_HHmmssfff}.png",
            DateTime.Now);
         File.WriteAllBytes(Path.Combine(_path, filename), bytes);
         yield return new WaitForSeconds(2.3f);
         _camera.cullingMask |= 1 << _layers;
         _isProcessed = false;
      }

      public void FirstMethod()
      {
         var filename = string.Format("{0:ddMMyyyy_HHmmssfff}.png", DateTime.Now);
```

```
            ScreenCapture.CaptureScreenshot(Path.Combine(_path, filename),
                _resolution);
        }
    }
}
```

Практическое задание

1. Реализовать сохранениие сохранение положительных и отрицательных бонусов на карте.
2. Добавить в игру миникарту
3. *Добавить к сохранению данных их шифрование.
4. * Реализовать в игре скриншот карты и вставить её под радар

Дополнительные материалы

1. https://habrahabr.ru/post/187394/ – Unity + MySQL;
2. https://habrahabr.ru/post/181239/ – Unity + SQLite;
3. https://habrahabr.ru/post/163071/ – хорошие примеры по сериализации данных;
4. https://unity3d.com/ru/learn/tutorials/topics/best-practices/guide-assetbundles-and-resources?playlist=30089
5. https://docs.unity3d.com/Manual/AssetBundlesIntro.html

Используемая литература

Для подготовки данного методического пособия были использованы следующие ресурсы:

1. https://docs.unity3d.com/ScriptReference/PlayerPrefs.html
2. https://docs.unity3d.com/Manual/JSONSerialization.html
3. https://docs.unity3d.com/Manual/StreamingAssets.html
4. https://docs.unity3d.com/Manual/class-ScriptableObject.html