Урок 4

MVC

mvc.

На этом уроке:

1. Вы узнаете что такое архитектура MVC
2. Научимся загружать префабы из ресурсов

Разделение логики и данных

Самое сложное в программировании крупных проектов, это структурирование кода и проработка зависимостей классов. Существуют различные паттерны, которые реализуют разделение ответственностей (MVC, MVP, MVVM и т.д.). Они разделяют данные приложения, пользовательского интерфейса и управляющей логики на три отдельных компонента, чтобы каждый из них можно было независимо модифицировать.

Ресурсы игры

Мы будем загружать некоторые объекты из кода. В Unity3d есть несколько способов динамически, во время игры, загружать ресурсы: префабы, объекты, изображения, текстуры и другие.

Один из способов воспользоваться классом Resources. Для этого необходимо в папке Assets создать специализированную папку Resources, это сделано для удобства, но вообще папка Resources может находится в любой иерархии папок. Объекты будут загружаться достаточно быстро, так как они индексируются. И все объекты из данной папки попадут в build. Поэтому не стоит помещать сюда все подряд. Объект загружается по имени! Также можно создавать подпапки и обращение будет происходить так: имя_папки/название_объекта. Вложенность может быть достаточно большой, но у данной папки есть ограничение в 4 Gb.

Пример:

```
using UnityEngine;

namespace Geekbrains
{
    public class ExampleClass : MonoBehaviour
    {
        void Start()
        {
            GameObject go = GameObject.CreatePrimitive(PrimitiveType.Plane);
            Renderer rend = go.GetComponent<Renderer>();
            rend.material.mainTexture = Resources.Load("glass") as Texture;
        }
    }
}
```

Один Update

Все классы, которые должны выполнять логику каждый кадр, будут реализовывать интерфейс IExecute

```
namespace Geekbrains
{
    public interface IExecute
    {
```

```
        void Execute();
    }
}
```

Создадим класс для хранения всех обновляемых объектов.

```
using System;
using System.Collections;
using Object = UnityEngine.Object;

namespace Geekbrains
{
    public sealed class ListExecuteObject : IEnumerator, IEnumerable
    {
        private IExecute[] _interactiveObjects;
        private int _index = -1;
        private InteractiveObject _current;

        public ListExecuteObject()
        {
            var interactiveObjects = Object.FindObjectsOfType<InteractiveObject>();
            for (var i = 0; i < interactiveObjects.Length; i++)
            {
                if (interactiveObjects[i] is IExecute interactiveObject)
                {
                    AddExecuteObject(interactiveObject);
                }
            }
        }

        public void AddExecuteObject(IExecute execute)
        {
            if (_interactiveObjects == null)
            {
                _interactiveObjects = new[] {execute};
                return;
            }
            Array.Resize(ref _interactiveObjects, Length + 1);
            _interactiveObjects[Length-1] = execute;
        }

        public IExecute this [int index]
        {
            get => _interactiveObjects[index];
            private set => _interactiveObjects[index] = value;
        }

        public int Length => _interactiveObjects.Length;

        public bool MoveNext()
        {
            if (_index == _interactiveObjects.Length - 1)
            {
                Reset();
                return false;
            }

            _index++;
            return true;
        }
```

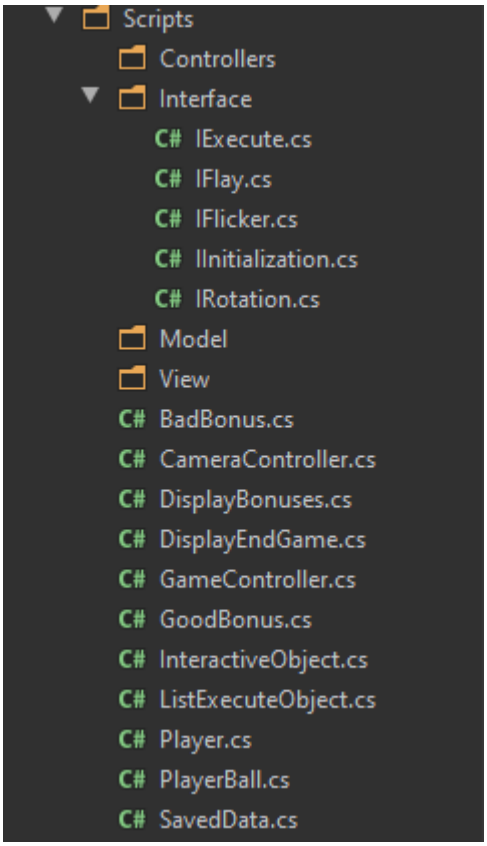
```
        public void Reset() => _index = -1;

        public object Current => _interactiveObjects[_index];

        public IEnumerator GetEnumerator()
        {
            return this;
        }

        IEnumerator IEnumerable.GetEnumerator()
        {
            return GetEnumerator();
        }
    }
}
```

В данном классе есть метод AddExecuteObject для добавления обновляемых объектов из вне.

```
using UnityEngine;

namespace Geekbrains
{
    public sealed class GameController : MonoBehaviour
    {
        private ListExecuteObject _interactiveObject;

        private void Awake()
        {
            _interactiveObject = new ListExecuteObject();
        }

        private void Update()
        {
            for (var i = 0; i < _interactiveObject.Length; i++)
            {
                var interactiveObject = _interactiveObject[i];

                if (interactiveObject == null)
                {
                    continue;
                }
                interactiveObject.Execute();
            }
        }
    }
}
```

Рефакторинг

Прежде чем переходить к разбиению функционала отрефакторим существующий код.



Класс InteractiveObject реализует интерфейс IExecute

```
using UnityEngine;
using static UnityEngine.Random;

namespace Geekbrains
{
    public abstract class InteractiveObject : MonoBehaviour, IExecute
    {
        protected Color _color;

        private bool _isInteractable;

        protected bool IsInteractable
        {
             get { return _isInteractable; }
             private set
             {
                 _isInteractable = value;
                 GetComponent<Renderer>().enabled = _isInteractable;
                 GetComponent<Collider>().enabled = _isInteractable;
             }
        }

        private void OnTriggerEnter(Collider other)
        {
            if (!IsInteractable || !other.CompareTag("Player"))
            {
                return;
            }
            Interaction();
```

```
            IsInteractable = false;
        }

        protected abstract void Interaction();
        public abstract void Execute();

        private void Start()
        {
            IsInteractable = true;
            _color = ColorHSV();
            if (TryGetComponent(out Renderer renderer))
            {
                renderer.material.color = _color;
            }
        }
    }
}
```

Наследники класса InteractiveObject GoodBonus и BadBonus. В них добавлены события, которые будут срабатывать при столкновении с этими объектами. И методы для движения объектов вызываются в методе Execute.

```
using System;
using UnityEngine;
using static UnityEngine.Random;

namespace Geekbrains
{
    public sealed class GoodBonus : InteractiveObject, IFlay, IFlicker
    {
        public int Point;
        public event Action<int> OnPointChange = delegate(int i) {  };
        private Material _material;
        private float _lengthFlay;

        private void Awake()
        {
            _material = GetComponent<Renderer>().material;
            _lengthFlay = Range(1.0f, 5.0f);
        }

        protected override void Interaction()
        {
            OnPointChange.Invoke(Point);
        }

        public override void Execute()
        {
            if(!IsInteractable){return;}
            Flay();
            Flicker();
        }

        public void Flay()
        {
            transform.localPosition = new Vector3(transform.localPosition.x,
                Mathf.PingPong(Time.time, _lengthFlay),
                transform.localPosition.z);
        }
```

```
        public void Flicker()
        {
            _material.color = new Color(_material.color.r, _material.color.g,
_material.color.b,
                Mathf.PingPong(Time.time, 1.0f));
        }
    }
}
```

При инициализации события пустым делегатом, можно не проверять его на null

```
using System;
using UnityEngine;
using static UnityEngine.Random;

namespace Geekbrains
{
    public sealed class BadBonus : InteractiveObject, IFlay, IRotation
    {
        public event Action<string, Color> OnCaughtPlayerChange = delegate(string
str, Color color) {  };
        private float _lengthFlay;
        private float _speedRotation;

        private void Awake()
        {
            _lengthFlay = Range(1.0f, 5.0f);
            _speedRotation = Range(10.0f, 50.0f);
        }

        protected override void Interaction()
        {
            OnCaughtPlayerChange.Invoke(gameObject.name, _color);
        }

        public override void Execute()
        {
            if(!IsInteractable){return;}
            Flay();
            Rotation();
        }

        public void Flay()
        {
            transform.localPosition = new Vector3(transform.localPosition.x,
                Mathf.PingPong(Time.time, _lengthFlay),
                transform.localPosition.z);
        }

        public void Rotation()
        {
            transform.Rotate(Vector3.up * (Time.deltaTime * _speedRotation),
Space.World);
        }
    }
}
```

Класс DisplayBonuses

```
using UnityEngine.UI;
namespace Geekbrains
{
    public sealed class DisplayBonuses
    {
        private Text _text;
        public DisplayBonuses()
        {
            // будем грузить из кода
        }

        public void Display(int value)
        {
            _text.text = $"Вы набрали {value}";
        }
    }
}
```

Класс DisplayEndGame

```
using UnityEngine;
using UnityEngine.UI;

namespace Geekbrains
{
    public sealed class DisplayEndGame
    {
        private Text _finishGameLabel;

        public DisplayEndGame()
        {
            // будем грузить из кода
        }

        public void GameOver(string name, Color color)
        {
            _finishGameLabel.text = $"Вы проиграли. Вас убил {name} {color} цвета";
        }
    }
}
```

Класс GameController

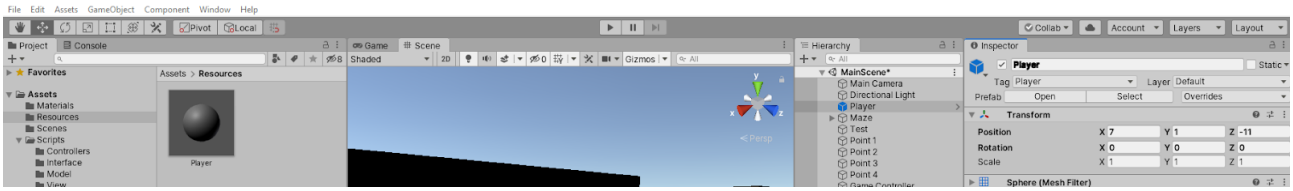
```
using System;
using UnityEngine;

namespace Geekbrains
{
    public sealed class GameController : MonoBehaviour, IDisposable
    {
        private ListExecuteObject _interactiveObject;
        private DisplayEndGame _displayEndGame;
        private DisplayBonuses _displayBonuses;
        private int _countBonuses;

        private void Awake()
        {
            _interactiveObject = new ListExecuteObject();
            _displayEndGame = new DisplayEndGame();
            _displayBonuses = new DisplayBonuses();
            foreach (var o in _interactiveObject)
```

```
            {
                if (o is BadBonus badBonus)
                {
                    badBonus.OnCaughtPlayerChange += CaughtPlayer;
                    badBonus.OnCaughtPlayerChange += _displayEndGame.GameOver;
                }

                if (o is GoodBonus goodBonus)
                {
                    goodBonus.OnPointChange += AddBonuse;
                }
            }
        }

        private void CaughtPlayer(string value, Color args)
        {
            Time.timeScale = 0.0f;
        }

        private void AddBonuse(int value)
        {
            _countBonuses += value;
            _displayBonuses.Display(_countBonuses);
        }

        private void Update()
        {
            for (var i = 0; i < _interactiveObject.Length; i++)
            {
                var interactiveObject = _interactiveObject[i];

                if (interactiveObject == null)
                {
                    continue;
                }
                interactiveObject.Execute();
            }
        }

        public void Dispose()
        {
            foreach (var o in _interactiveObject)
            {
                if (o is BadBonus badBonus)
                {
                    badBonus.OnCaughtPlayerChange -= CaughtPlayer;
                    badBonus.OnCaughtPlayerChange -= _displayEndGame.GameOver;
                }

                if (o is GoodBonus goodBonus)
                {
                    goodBonus.OnPointChange -= AddBonuse;
                }
            }
        }
    }
}
```

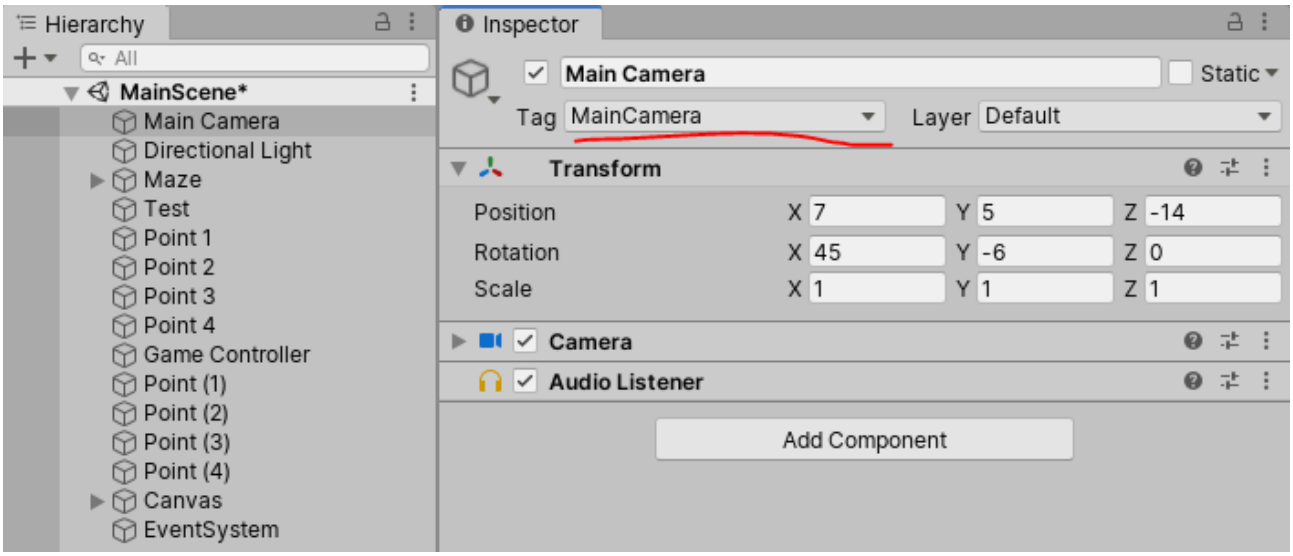
Классы CameraController, Player, PlayerBall и SavedData пока остаются без изменений.

Класс для управления камерой

Создадим папку и поместим туда игровой объект сделав из него префаб. Удаляем этот объект со сцены предварительно запоминая координаты, где находился объект.



С объекта Main Camera удаляем скрипт CameraController и проверяем что у данного объекта выставлен тэг MainCamera



Наша игра не особо большая, поэтому **в нашем случае** можно использовать один класс для получения ссылок на объекты.

```
using UnityEngine;

namespace Geekbrains
{
    public class Reference
    {
        private PlayerBall _playerBall;
        private Camera _mainCamera;

        public PlayerBall PlayerBall
        {
            get
            {
                if (_playerBall == null)
                {
                    var gameObject = Resources.Load<PlayerBall>("Player");
                    _playerBall = Object.Instantiate(gameObject);
                }

                return _playerBall;
            }
```

```
        }

        public Camera MainCamera
        {
            get
            {
                if (_mainCamera == null)
                {
                    _mainCamera = Camera.main;
                }
                return _mainCamera;
            }
        }
    }
}
```

Удаляем наследование CameraController от MonoBehaviour. Добавляем реализацию интерфейса IExecute. В конструкторе пробрасываем зависимость от класса Reference.

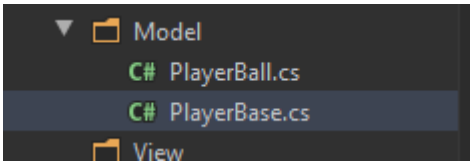
```
using UnityEngine;

namespace Geekbrains
{
    public sealed class CameraController : IExecute
    {
        private Transform _player;
        private Transform _mainCamera;
        private Vector3 _offset;

        public CameraController(Transform player, Transform mainCamera)
        {
            _player = player;
            _mainCamera = mainCamera;
            _mainCamera.LookAt(_player);
            _offset = _mainCamera.position - _player.position;
        }

        public void Execute()
        {
            _mainCamera.position = _player.position + _offset;
        }
    }
}
```

В классе GameController создаем экземпляр класса CameraController и добавляем его в список объектов, которые должны обновляться каждый кадр.

```
private void Awake()
{
    _interactiveObject = new ListExecuteObject();

    var reference = new Reference();

    _cameraController = new CameraController(reference.PlayerBall.transform, reference.MainCamera.transform);
    _interactiveObject.AddExecuteObject(_cameraController);
```

Перемещение игрока

Класс Player переименовываем в класс PlayerBase и вместе с классом PlayerBall переносим в папку Model.



Модифицируем класс PlayerBase

```
using UnityEngine;

namespace Geekbrains
{
    public abstract class PlayerBase : MonoBehaviour
    {
        public float Speed = 3.0f;

        public abstract void Move(float x, float y, float z);
    }
}
```

И соответственно изменим логику дочернего класса PlayerBall

```
using UnityEngine;

namespace Geekbrains
{
    public sealed class PlayerBall : PlayerBase
    {
        private Rigidbody _rigidbody;

        private void Start()
        {
            _rigidbody = GetComponent<Rigidbody>();
        }

        public override void Move(float x, float y, float z)
        {
            _rigidbody.AddForce(new Vector3(x, y, z) * Speed);
        }
    }
}
```

Добавим контроллер управления игроком

```
using UnityEngine;

namespace Geekbrains
{
    public sealed class InputController : IExecute
    {
        private readonly PlayerBase _playerBase;

        public InputController(PlayerBase player)
        {
            _playerBase = player;
```

```
        }

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

И добавим класс InputController в обновляемые объекты

```
        private CameraController _cameraController;
        private InputController _inputController;
        private int _countBonuses;

        Event function
        private void Awake()
        {
            _interactiveObject = new ListExecuteObject();

            var reference = new Reference();

            _cameraController = new CameraController(reference.PlayerBall.transform, reference.MainCamera.transform);
            _interactiveObject.AddExecuteObject(_cameraController);

            _inputController = new InputController(reference.PlayerBall);
            _interactiveObject.AddExecuteObject(_inputController);

            _displayEndGame = new DisplayEndGame();
```

Теперь мы можем динамически заменять игроков и управление, например, под другую платформу

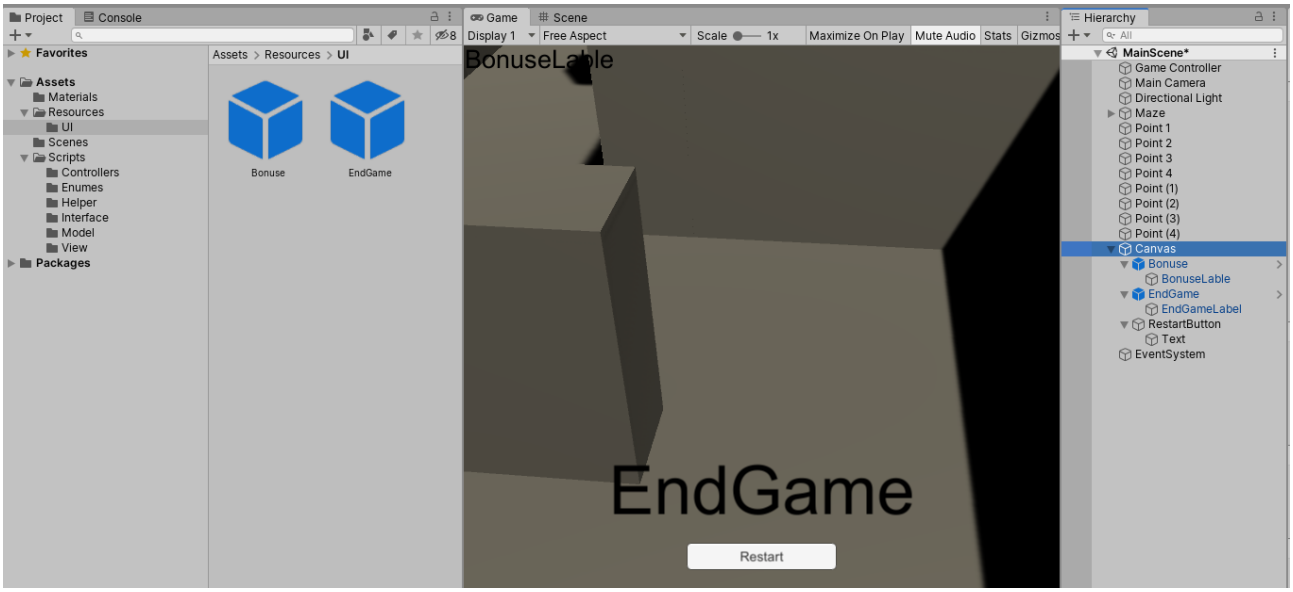
Пример:

```
namespace Geekbrains
{
    public enum PlayerType
    {
        None = 0,
        Ball = 1,
        Cube = 2
    }
}
```

```
public PlayerType PlayerType = PlayerType.Ball;  Unchanged
private ListExecuteObject _interactiveObject;
private DisplayEndGame _displayEndGame;
private DisplayBonuses _displayBonuses;
private CameraController _cameraController;
private InputController _inputController;
private int _countBonuses;

 Event function
private void Awake()
{
    _interactiveObject = new ListExecuteObject();

    var reference = new Reference();

    PlayerBase player = null;
    if (PlayerType == PlayerType.Ball)
    {
        player = reference.PlayerBall;
    }

    _cameraController = new CameraController(player.transform, reference.MainCamera.transform);
    _interactiveObject.AddExecuteObject(_cameraController);

    if (Application.platform == RuntimePlatform.WindowsEditor)
    {
        _inputController = new InputController(player);
        _interactiveObject.AddExecuteObject(_inputController);
    }
```

Пользовательский интерфейс

Разобьем объекты по группам и уберем в префаб



Добавим в класс с ресурсами наши объекты

```
public sealed class Reference
{
    private PlayerBall _playerBall;
    private Camera _mainCamera;
    private GameObject _bonuse;
    private GameObject _endGame;
    private Canvas _canvas;

    public Canvas Canvas
    {
        get
        {
            if (_canvas == null)
            {
                _canvas = Object.FindObjectOfType<Canvas>();
            }
            return _canvas;
        }
    }

    public GameObject Bonuse
    {
        get
        {
            if (_bonuse == null)
            {
                var gameObject = Resources.Load<GameObject>("UI/Bonuse");
                _bonuse = Object.Instantiate(gameObject, Canvas.transform);
            }

            return _bonuse;
        }
    }

    public GameObject EndGame
    {
        get
        {
            if (_endGame == null)
            {
                var gameObject = Resources.Load<GameObject>("UI/EndGame");
                _endGame = Object.Instantiate(gameObject, Canvas.transform);
            }

            return _endGame;
        }
    }

    //....
}
```

```
using System;
using UnityEngine;
using UnityEngine.UI;

namespace Geekbrains
{
    public sealed class DisplayBonuses
```

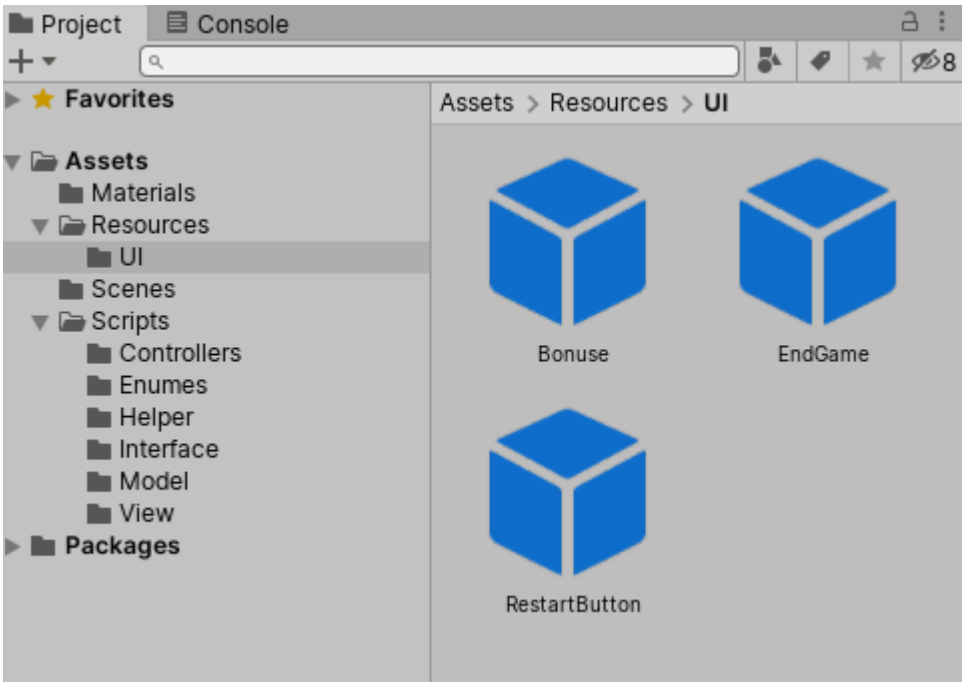
```
    {
        private Text _bonuseLable;

        public DisplayBonuses(GameObject bonus)
        {
            _bonuseLable = bonus.GetComponentInChildren<Text>();
            _bonuseLable.text = String.Empty;
        }

        public void Display(int value)
        {
            _bonuseLable.text = $"Вы набрали {value}";
        }
    }
}
```

```
using System;
using UnityEngine;
using UnityEngine.UI;

namespace Geekbrains
{
    public sealed class DisplayEndGame
    {
        private Text _finishGameLabel;

        public DisplayEndGame(GameObject endGame)
        {
            _finishGameLabel = endGame.GetComponentInChildren<Text>();
            _finishGameLabel.text = String.Empty;
        }

        public void GameOver(string name, Color color)
        {
            _finishGameLabel.text = $"Вы проиграли. Вас убил {name} {color} цвета";
        }
    }
}
```

В классе GameController в методе Awake передаем ссылки на окна

```
_displayEndGame = new DisplayEndGame(reference.EndGame);
_displayBonuses = new DisplayBonuses(reference.Bonuse);
```

Кнопку перезапуска игры также переносим в ресурсы




```
using UnityEngine;
using UnityEngine.UI;

namespace Geekbrains
{
    public sealed class Reference
    {
        private PlayerBall _playerBall;
        private Camera _mainCamera;
        private GameObject _bonuse;
        private GameObject _endGame;
        private Canvas _canvas;
        private Button _restartButton;

        public Button RestartButton
        {
            get
            {
                if (_restartButton == null)
                {
                    var gameObject = Resources.Load<Button>("UI/RestartButton");
                    _restartButton = Object.Instantiate(gameObject,
Canvas.transform);
                }

                return _restartButton;
            }
        }

        //....
    }
}
```

```
using System;
using UnityEngine;
```

```
using UnityEngine.SceneManagement;

namespace Geekbrains
{
    public sealed class GameController : MonoBehaviour, IDisposable
    {
        public PlayerType PlayerType = PlayerType.Ball;
        private ListExecuteObject _interactiveObject;
        private DisplayEndGame _displayEndGame;
        private DisplayBonuses _displayBonuses;
        private CameraController _cameraController;
        private InputController _inputController;
        private int _countBonuses;
        private Reference _reference;

        private void Awake()
        {
            _interactiveObject = new ListExecuteObject();

            _reference = new Reference();

            PlayerBase player = null;
            if (PlayerType == PlayerType.Ball)
            {
                player = _reference.PlayerBall;
            }

            _cameraController = new CameraController(player.transform,
_reference.MainCamera.transform);
            _interactiveObject.AddExecuteObject(_cameraController);

            if (Application.platform == RuntimePlatform.WindowsEditor)
            {
                _inputController = new InputController(player);
                _interactiveObject.AddExecuteObject(_inputController);
            }

            _displayEndGame = new DisplayEndGame(_reference.EndGame);
            _displayBonuses = new DisplayBonuses(_reference.Bonuse);
            foreach (var o in _interactiveObject)
            {
                if (o is BadBonus badBonus)
                {
                    badBonus.OnCaughtPlayerChange += CaughtPlayer;
                    badBonus.OnCaughtPlayerChange += _displayEndGame.GameOver;
                }

                if (o is GoodBonus goodBonus)
                {
                    goodBonus.OnPointChange += AddBonuse;
                }
            }

            _reference.RestartButton.onClick.AddListener(RestartGame);
            _reference.RestartButton.gameObject.SetActive(false);
        }

        private void RestartGame()
        {
```

```
            SceneManager.LoadScene(sceneBuildIndex: 0);
            Time.timeScale = 1.0f;
        }

        private void CaughtPlayer(string value, Color args)
        {
            _reference.RestartButton.gameObject.SetActive(true);
            Time.timeScale = 0.0f;
        }

        private void AddBonuse(int value)
        {
            _countBonuses += value;
            _displayBonuses.Display(_countBonuses);
        }

        private void Update()
        {
            for (var i = 0; i < _interactiveObject.Length; i++)
            {
                var interactiveObject = _interactiveObject[i];

                if (interactiveObject == null)
                {
                    continue;
                }
                interactiveObject.Execute();
            }
        }

        public void Dispose()
        {
            foreach (var o in _interactiveObject)
            {
                if (o is BadBonus badBonus)
                {
                    badBonus.OnCaughtPlayerChange -= CaughtPlayer;
                    badBonus.OnCaughtPlayerChange -= _displayEndGame.GameOver;
                }

                if (o is GoodBonus goodBonus)
                {
                    goodBonus.OnPointChange -= AddBonuse;
                }
            }
        }
    }
}
```

Практическое задание

1. Добавить кнопку перезапуска уровня
2. Добавить экран победы
3. *Реализовать разделение на MVC бонусов и ловушек

Дополнительные материалы

1. https://github.com/neuecc/UniRx – еще один вариант проектирования игры, основанной на событиях, без использования стандартной функции Update.
2. https://github.com/sschmid/Entitas-CSharp – хорошие примеры оптимизации программного кода.
3. Джозеф Хокинг, Unity в действии. Мультиплатформенная разработка на C# – очень интересная и полезная книга по основам Unity3D.
4. https://habrahabr.ru/post/303562/ – о кэшировании данных.
5. http://netcoder.ru/blog/csharp/240.html – и еще статья про кэширование.
6. https://msdn.microsoft.com/ru-ru/library/ff926074.aspx – правила написания кода c#.
7. https://habrahabr.ru/post/26077/ – рекомендации по написанию кода.
8. https://habr.com/ru/post/215605/
9. https://habr.com/ru/post/338518/
10. https://habr.com/ru/post/319652/
11. https://habr.com/ru/post/276593/
12. https://habr.com/ru/post/270547/
13. https://blogs.unity3d.com/ru/2015/12/23/1k-update-calls/
14. https://habr.com/ru/post/322258/
15. https://ru.stackoverflow.com/questions/391369/mvc-%D0%B2-unity-%D0%9D%D0%B0%D0%B4%D0%BE-%D0%BB%D0%B8-%D0%95%D1%81%D0%BB%D0%B8-%D0%B4%D0%B0-%D1%82%D0%BE-%D0%BA%D0%B0%D0%BA

Используемая литература

Для подготовки данного методического пособия были использованы следующие ресурсы:

1. э