ПРИЛОЖЕНИЕ А

AttachToPlayersBody.cs

```
using UnityEngine;

/**
Скрипт, прикрепляющий объект к якорю весящему на игроке

Данный класс используется для того, чтобы объекты, которые прикрепляются к игроку не приходилось долго искать на самом игроке.
Идея в следующем:
1. В нужную(-ые) части тела игрока мы прикрепляем EmptyObject якорь;
2. Создаем нужный нам объект, где нам будет угодно;
3. Добавляем на объект данный скрипт и в его полях указываем якорь, к которому объект будет прикреплен при запуске приложения;
4. *Если объект прикрепляется к рукам, то необходимо указать отдельный якорь для контроллеров и для рук*

@attention Для корректной работы данный класс требует, что бы в сцене присутствовали скрипты:
- ControllerEvents;

@param catcher ComponentCatcher данной сцены.
@param controllerBodyPart Якорь, к которому прикрепляется объект.
Если объект прикрепляется к рукам, то в данный параметр указывается якорь для прикрепления к контроллерам
@param handBodyPart (*Не обязательно*) Если объект прикрепляется к рукам, то в данный параметр указывается якорь для прикрепления
↪  к рукам
@see ComponentCatcher; ControllerEvents
 */
public class AttachToPlayersBody : MonoBehaviour
{
    [SerializeField] private Transform _controllerBodyPart;
    [Header("[Optional]")]
    [SerializeField] private Transform _handBodyPart;

    [SerializeField] private ComponentCatcher _catcher;

    private Vector3 _currentPosition;
    private Vector3 _currentRotation;

    private ControllerEvents _controllerEvents;

    private void Start()
    {
        _controllerEvents = _catcher.GetControllerEvents();
        if (_controllerEvents != null)
        {
            _controllerEvents.ControllerTypeChange += OnAttachChange;
        }

        _currentPosition = new Vector3(transform.localPosition.x, transform.localPosition.y, transform.localPosition.z);
        _currentRotation = new Vector3(transform.localRotation.eulerAngles.x, transform.localRotation.eulerAngles.y,
        ↪  transform.localRotation.eulerAngles.z);

        gameObject.transform.parent = _controllerBodyPart;

        OnAttachChange(!OVRPlugin.GetHandTrackingEnabled());

        if (_handBodyPart == null)
        {
            _handBodyPart = _controllerBodyPart;
        }
    }

    private void OnDestroy()
    {
        if (_controllerEvents != null)
        {
            _controllerEvents.ControllerTypeChange -= OnAttachChange;
        }
    }

    private void OnAttachChange(bool isAttachToController)
    {
        if (isAttachToController)
        {
            gameObject.transform.parent = _controllerBodyPart;
            RestoreLocalTransform();
        }
        else
        {
            gameObject.transform.parent = _handBodyPart;
            RestoreLocalTransform();
        }
    }

    private void RestoreLocalTransform()
    {
        transform.localPosition = new Vector3(_currentPosition.x, _currentPosition.y, _currentPosition.z);
        transform.localRotation = Quaternion.Euler(new Vector3(_currentRotation.x, _currentRotation.y, _currentRotation.z));
    }
}
```

AudioController.cs

```
using UnityEngine;

/**
```

```
Класс, манипулирующий звуками сцены

@param controlledSources Массив AudioSource сцены, над которыми будут производиться манипуляции.
 */
public class AudioController : MonoBehaviour
{
    [SerializeField] private AudioSource[] _controlledSources;

    private bool _isMute;

    private void Start()
    {
        _isMute = false;
        SetMuteToAll();
    }

    /**
     Вкл/выкл конкретный аудио источник.
    @param [in] index Индекс переключаемого источника
     */
    public void SwitchMute(int index)
    {
        if (index >= 0 && index < _controlledSources.Length)
        {
            _controlledSources[index].mute = !_controlledSources[index].mute;
        }
    }

    /// Вкл/выкл все контролируемые аудио источники.
    public void SwitchMuteToAll()
    {
        _isMute = !_isMute;
        SetMuteToAll();
    }

    private void SetMuteToAll()
    {
        foreach (AudioSource audioSource in _controlledSources)
        {
            audioSource.mute = _isMute;
        }
    }
}
```

AvatarInfo.cs

```
using UnityEngine;

/**
 Класс, хранящий информацию об аватаре.

Данный класс необходим для поиска prefab-ов аватаров в папке Resources.
Так же данный класс используется для отображения аватаров в UI.

@param isActive Если аватар не активен, он не будет добавлятся в список аватаров и не будет отображаться.
Это нужно, если какой-то аватар еще не настроен, но необходимо вести работу с другими аватарами.
@param avatarImage Изображение отображаемое в UI.
@param avatarName Путь до аватара в папке Resources/Avatars. По данному пути NetworkManager будет искать данный аватар.
 */
public class AvatarInfo : MonoBehaviour
{
    [SerializeField] private bool _isActive;
    [SerializeField] private Sprite _avatarImage;
    [SerializeField] private string _avatarName;

    /**
     Геттер активности аватара.
    @return bool Активен ли данный аватар.
     */
    public bool IsAvatarActive()
    {
        return _isActive;
    }

    /**
     Геттер изображения аватара.
    @return Sprite Изображение аватара.
     */
    public Sprite GetAvatarImage()
    {
        return _avatarImage;
    }

    /**
      Геттер имени аватара.

     Имя аватара – это путь до prefab-а с аватаром в папке Resources/Avatars.
     @return string Путь до prefab-а с аватаром в папке Resources/Avatars.
      */
    public string GetAvatarName()
    {
        return _avatarName;
    }
}
```

MapAvatarBody.cs

```
using UnityEngine;

/// Класс для установки положения и поворота кости скелета, в положение цели.
[System.Serializable]
public class MapRigTransform
{
    /// Положение, в которое будет установлена указанная кость.
    public Transform Target;
    /// Кость скелета, которая будет установлена в указанное положение.
    public Transform Rig;

    /// Отступ положения кости от цели.
    public Vector3 TrackingPositionOffset;
    /// Отступ поворота кости от цели
    public Vector3 TrackingRotationOffset;

    /// Установить кость в указанное положение с учетом отступов.
    public void MapRig()
    {
        Rig.position = Target.TransformPoint(TrackingPositionOffset);
        Rig.rotation = Target.rotation * Quaternion.Euler(TrackingRotationOffset);
    }
}

/**
 Класс, отвечающий за синхронизацию положения тела аватара, с управляющими элементами.

Данный класс связывает отображаемую модель аватара и положение шлема, контроллеров и рук.
Без данного класса модель заспавнится в случайном месте и будет тянуться к скелету.

@param head MapRigTransform для контроллера головы и кости головы.
@param rightHand MapRigTransform для контроллера правой руки и кости правой руки.
@param leftHand MapRigTransform для контроллера левой руки и кости левой руки.
@param bodyOffset Отступ отображаемого тела от центральной камеры шлема.
@param turningSmoothness Плавность поворота тела вслед за головой.
@see MapRigTransform
*/
public class MapAvatarBody : MonoBehaviour
{
    [SerializeField] private MapRigTransform _head;
    [SerializeField] private MapRigTransform _rightHand;
    [SerializeField] private MapRigTransform _leftHand;

    [SerializeField] private float _turningSmoothness;
    [SerializeField] private Vector3 _bodyOffset;

    private void LateUpdate()
    {
        transform.position = _head.Rig.position + _bodyOffset;

        transform.forward = Vector3.Lerp(transform.forward, Vector3.ProjectOnPlane(_head.Rig.forward, Vector3.up).normalized,
        ↪  Time.deltaTime * _turningSmoothness);

        _head.MapRig();
        _rightHand.MapRig();
        _leftHand.MapRig();
    }
}
```

RPMAvatarInfo.cs

```
using UnityEngine;

/**
 Класс, хранящий информацию об аватаре, использующим модель Ready Player Me.

@param redyPlayerMeAvatar Prefab аватара, созданный через Ready Player Me.
@param gender Пол модели. В зависимости от данного параметра будет выбран скелет,
к которому будет прикреплена модель.
@see AvatarInfo
 */
public class RPMAvatarInfo : AvatarInfo
{
    [SerializeField] private GameObject _redyPlayerMeAvatar;
    [SerializeField] private Gender _gender;

    private void Awake()
    {
        RPMAvatarParser[] avatarSkeleton = GetComponentsInChildren<RPMAvatarParser>();
        foreach(RPMAvatarParser skeleton in avatarSkeleton)
        {
            if(skeleton.GetSkeletonGender()!= _gender)
            {
                skeleton.gameObject.SetActive(false);
            }
        }
    }

    /**
     Геттер модели Ready Player Me.
    @return Prefab аватара, созданный через Redy Player Me
     */
    public GameObject GetRedyPlayerMeAvatar()
    {
        return _redyPlayerMeAvatar;
```

```
    }
}
```

RPMAvatarParser.cs

```
using UnityEngine;

/// Класс для замены меша и материала оригинальной модели на указанную
[System.Serializable]
public class MeshSwapper
{
    /// Нужно ли использовать данную модель.
    public bool IsActive;
    /// SkinnedMeshRenderer оригинальной модели. В данной модели меш и материал будут заменены.
    public SkinnedMeshRenderer Original;
    /**
     SkinnedMeshRenderer, из которого будут взяты меш и материалы.
    @attention Данная модель должна присутствовать в сцене.
    То есть перед ее использованием ее необходимо заспавнить при помощи Instantiate.
     */
    public SkinnedMeshRenderer NewMesh;

    /// Метод замены меша и материалов в оригинальной модели на меш и материалы из указанной.
    public void SwapMesh()
    {
        Original.sharedMesh = NewMesh.sharedMesh;
        Original.materials = NewMesh.materials;
    }
}

/// Пол скелета модели
public enum Gender
{
    Male, ///< Мужской
    Female ///< Женский
};


/**
 Компонент для парсинга модели Ready Player Me в аватар.

Данный компонент берет модель из RPMAvatarInfo и прикрепляет выбранные ее части к скелету,
на котором используется данный компонент.
@param skeletonGender Пол скелета, с которым работает данный компонент.
@param rpmAvatarInfo RPMAvatarInfo аватара, в котором работает данный компонент.
Из данного RPMAvatarInfo будет взята модель, прикрепляемая к скелету.
@param eyeLeft Использовать ли в аватаре модель левого глаза.
@param eyeRight Использовать ли в аватаре модель правого глаза.
@param head Использовать ли в аватаре модель головы.
@param teeth Использовать ли в аватаре модель зубов.
@param body Использовать ли в аватаре модель тела.
@param outfitBottom Использовать ли в аватаре модель штанов.
@param outfitFootwear Использовать ли в аватаре модель ботинок.
@param outfitTop Использовать ли в аватаре модель верхней одежды.
@param hair Использовать ли в аватаре модель волос.
@param beard Использовать ли в аватаре модель бороды.
@param glasses Использовать ли в аватаре модель очков.
@see RPMAvatarInfo
 */
public class RPMAvatarParser : MonoBehaviour
{
    [SerializeField] private Gender _skeletonGender;
    [SerializeField] private RPMAvatarInfo _rpmAvatarInfo;

    [Header("Active avatar parts")]
    [SerializeField] private bool _eyeLeft;
    [SerializeField] private bool _eyeRight;
    [SerializeField] private bool _head;
    [SerializeField] private bool _teeth;
    [SerializeField] private bool _body;
    [SerializeField] private bool _outfitBottom;
    [SerializeField] private bool _outfitFootwear;
    [SerializeField] private bool _outfitTop;
    [SerializeField] private bool _hair;
    [SerializeField] private bool _beard;
    [SerializeField] private bool _glasses;

    private MeshSwapper _eyeLeftMS;
    private MeshSwapper _eyeRightMS;
    private MeshSwapper _headMS;
    private MeshSwapper _teethMS;
    private MeshSwapper _bodyMS;
    private MeshSwapper _outfitBottomMS;
    private MeshSwapper _outfitFootwearMS;
    private MeshSwapper _outfitTopMS;
    private MeshSwapper _hairMS;
    private MeshSwapper _beardMS;
    private MeshSwapper _glassesMS;

    private void Start()
    {
        _eyeLeftMS = new MeshSwapper();
        _eyeLeftMS.IsActive = _eyeLeft;
        _eyeRightMS = new MeshSwapper();
        _eyeRightMS.IsActive = _eyeRight;
        _headMS = new MeshSwapper();
        _headMS.IsActive = _head;
        _teethMS = new MeshSwapper();
```

```
            _teethMS.IsActive = _teeth;
            _bodyMS = new MeshSwapper();
            _bodyMS.IsActive = _body;
            _outfitBottomMS = new MeshSwapper();
            _outfitBottomMS.IsActive = _outfitBottom;
            _outfitFootwearMS = new MeshSwapper();
            _outfitFootwearMS.IsActive = _outfitFootwear;
            _outfitTopMS = new MeshSwapper();
            _outfitTopMS.IsActive = _outfitTop;
            _hairMS = new MeshSwapper();
            _hairMS.IsActive = _hair;
            _beardMS = new MeshSwapper();
            _beardMS.IsActive = _beard;
            _glassesMS = new MeshSwapper();
            _glassesMS.IsActive = _glasses;

            SkinnedMeshRenderer[] children = GetComponentsInChildren<SkinnedMeshRenderer>();
            GameObject tempObject = Instantiate(_rpmAvatarInfo.GetRedyPlayerMeAvatar());
            SkinnedMeshRenderer[] rpmChildrens = tempObject.GetComponentsInChildren<SkinnedMeshRenderer>();
            for (int i = 0; i < children.Length; i++)
            {
                switch (children[i].name)
                {
                    case "Renderer_EyeLeft":
                        SetMeshes("Renderer_EyeLeft", ref _eyeLeftMS, ref children[i], rpmChildrens);
                        break;
                    case "Renderer_EyeRight":
                        SetMeshes("Renderer_EyeRight", ref _eyeRightMS, ref children[i], rpmChildrens);
                        break;
                    case "Renderer_Head":
                        SetMeshes("Renderer_Head", ref _headMS, ref children[i], rpmChildrens);
                        break;
                    case "Renderer_Teeth":
                        SetMeshes("Renderer_Teeth", ref _teethMS, ref children[i], rpmChildrens);
                        break;
                    case "Renderer_Body":
                        SetMeshes("Renderer_Body", ref _bodyMS, ref children[i], rpmChildrens);
                        break;
                    case "Renderer_Outfit_Bottom":
                        SetMeshes("Renderer_Outfit_Bottom", ref _outfitBottomMS, ref children[i], rpmChildrens);
                        break;
                    case "Renderer_Outfit_Footwear":
                        SetMeshes("Renderer_Outfit_Footwear", ref _outfitFootwearMS, ref children[i], rpmChildrens);
                        break;
                    case "Renderer_Outfit_Top":
                        SetMeshes("Renderer_Outfit_Top", ref _outfitTopMS, ref children[i], rpmChildrens);
                        break;
                    case "Renderer_Hair":
                        SetMeshes("Renderer_Hair", ref _hairMS, ref children[i], rpmChildrens);
                        break;
                    case "Renderer_Beard":
                        SetMeshes("Renderer_Beard", ref _beardMS, ref children[i], rpmChildrens);
                        break;
                    case "Renderer_Glasses":
                        SetMeshes("Renderer_Glasses", ref _glassesMS, ref children[i], rpmChildrens);
                        break;
                }
            }
            Destroy(tempObject);
        }

        private void SetMeshes(string name, ref MeshSwapper meshSwapper, ref SkinnedMeshRenderer child, SkinnedMeshRenderer[]
        ↪  rpmChildrens)
        {
            if (meshSwapper.IsActive)
            {
                meshSwapper.Original = child;
                foreach (SkinnedMeshRenderer rpmChildren in rpmChildrens)
                {
                    if (rpmChildren.name == name)
                    {
                        meshSwapper.NewMesh = rpmChildren;
                        break;
                    }
                }
                if (meshSwapper.NewMesh == null)
                {
                    child.gameObject.SetActive(false);
                }
                else
                {
                    meshSwapper.SwapMesh();
                }
            }
            else
            {
                child.gameObject.SetActive(false);
            }
        }

        /**
         Геттер пола скелета.
        @return Gender используемого скелета.
         */
        public Gender GetSkeletonGender()
        {
            return _skeletonGender;
        }
    }
```

ControllerAnimationUpdater.cs

```
using UnityEngine;

/**
Класс обеспечивающий проигрывание анимации контроллеров Oculus на сервере

Данный класс обновляет анимацию моделей контроллера на сервере, в зависимости от нажатых на локальном контроллере
↪  кнопок/стиков/триггеров.
@param animator Аниматор, который используется для анимации модели контроллера.
 */
public class ControllerAnimationUpdater : ControllerModel
{
    [SerializeField] private Animator _animator;

    private void Update()
    {
        UpdateAnimation(_animator);
    }

    private void UpdateAnimation(Animator animator)
    {
        if (animator != null && _myPhotonView.IsMine)
        {
            animator.SetFloat("Button 1", OVRInput.Get(OVRInput.Button.One, _controllerType) ? 1.0f : 0.0f);
            animator.SetFloat("Button 2", OVRInput.Get(OVRInput.Button.Two, _controllerType) ? 1.0f : 0.0f);
            animator.SetFloat("Button 3", OVRInput.Get(OVRInput.Button.Start, _controllerType) ? 1.0f : 0.0f);

            animator.SetFloat("Joy X", OVRInput.Get(OVRInput.Axis2D.PrimaryThumbstick, _controllerType).x);
            animator.SetFloat("Joy Y", OVRInput.Get(OVRInput.Axis2D.PrimaryThumbstick, _controllerType).y);

            animator.SetFloat("Trigger", OVRInput.Get(OVRInput.Axis1D.PrimaryIndexTrigger, _controllerType));
            animator.SetFloat("Grip", OVRInput.Get(OVRInput.Axis1D.PrimaryHandTrigger, _controllerType));
        }
    }
}
```

ControllerEvents.cs

```
using UnityEngine;
using UnityEngine.Events;

/**
Класс, отслеживающий переключение с контроллеров на руки
 */
public class ControllerEvents : MonoBehaviour
{
    /// Событие переключения контроллеров на руки или наоборот.
    public UnityAction<bool> ControllerTypeChange;

    private bool _isAttachToController;

    private void Start()
    {
        _isAttachToController = OVRPlugin.GetHandTrackingEnabled();
        ControllerTypeChange?.Invoke(_isAttachToController);
    }

    private void Update()
    {
        if (OVRPlugin.GetHandTrackingEnabled())
        {
            if (_isAttachToController)
            {
                _isAttachToController = false;
                ControllerTypeChange?.Invoke(_isAttachToController);
            }
        }
        else
        {
            if (!_isAttachToController)
            {
                _isAttachToController = true;
                ControllerTypeChange?.Invoke(_isAttachToController);
            }
        }
    }

    /**
    Геттер текущего состояния.
    @return bool Если true, значит в данный момент используются контроллеры.
    Если false, значит в данный момент используются руки.
     */
    public bool IsAttachToControllerNow()
    {
        return _isAttachToController;
    }
}
```

ControllerModel.cs

```
using UnityEngine;
using Photon.Pun;

/// Тип внешнего вида контроллеров.
public enum ControllerType
{
    OculusController, ///< Контроллеры Oculus.
    HandsPrefabs ///< Руки игрока.
}

/**
Суперкласс хранящий информацию о отображаемом на сервере контроллере
 */
public class ControllerModel : MonoBehaviour
{
    /// PhotonView отвечающий за синхронизацию данного объекта.
    [SerializeField] protected PhotonView _myPhotonView;

    /// Отображаемая модель контроллера.
    [SerializeField] protected GameObject _objectModel;

    /** ControllerType отображаемого контроллера.
     - OculusController;
     - HandsPrefabs;
     */
    [SerializeField] protected ControllerType _type;
    /**
     Тип контроллера.
    - OVRInput.Controller.LTouch;
    - OVRInput.Controller.RTouch;
    - OVRInput.Controller.LHand;
    - OVRInput.Controller.RHand;
     */
    [SerializeField] protected OVRInput.Controller _controllerType;

    /**
     Геттер модели контроллера.
    @return GameObject используемая модель контроллера.
     */
    public GameObject GetObjectModel()
    {
        return _objectModel;
    }

    /**
     Геттер типа контроллера.
    @return ControllerType данного контроллера
     */
    public ControllerType GetControllerType()
    {
        return _type;
    }

    /**
     Установка существования данного объекта.

     Когда мы переключаемся на другое отображение контроллеров, мы должны отключить предыдущее и включить новое отображение.
    @param [in] isActive Существует ли данный объект.
     */
    public void SetActive(bool isActive)
    {
        gameObject.SetActive(isActive);
    }
}
```

ControllerTypeController.cs

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

/**
Класс, отвечающий за переключение типов контроллеров на сервере

Данный класс используется в prefab-е игрока на сервере.
Когда локальный игрок меняет тип контроллера (с контроллеров на руки или наоборот),
данный класс изменяет меняет отображаемый тип контроллера на сервере.
@param controllers Список переключаемых контроллеров.
@see ControllerModel
 */
public class ControllerTypeController : MonoBehaviour
{
    [SerializeField] private List<ControllerModel> _controllers;

    private ControllerType _currentControllerType;
    private ControllerModel _currentController;

    private void Start()
    {
        SwitchControllerView(_currentControllerType);
    }

    private void OnDestroy()
    {
        _controllers.Clear();
    }
```

```
    public void SwitchControllerView(ControllerType controllerType)
    {
        _currentControllerType = controllerType;
        if (_currentController == null || _currentController.GetControllerType() != controllerType)
        {
            foreach (ControllerModel controller in _controllers)
            {
                if (controller.GetControllerType() == controllerType)
                {
                    controller.SetActive(true);
                    _currentController = controller;
                }
                else
                {
                    controller.SetActive(false);
                }
            }
        }
    }
}
```

GostHandTransformUpdater.cs

```
using UnityEngine;
using Photon.Pun;

/**
Класс, отвечающий за синхронизацию положения рук на сервере

Данный класс синхронизирует руки из prefab-а LeftHandSynthetic/RightHandSynthetic.

Класс ориентируется на название костей в руках. А именно он ищет в сцене и в переданном объекте объекты с названиями:
- b_l_wrist;
- b_r_wrist;
- b_l_index1;
- b_r_index1;
- b_l_index2;
- b_r_index2;
- b_l_index3;
- b_r_index3;
- b_l_middle1;
- b_r_middle1;
- b_l_middle2;
- b_r_middle2;
- b_l_middle3;
- b_r_middle3;
- b_l_pinky0;
- b_r_pinky0;
- b_l_pinky1;
- b_r_pinky1;
- b_l_pinky2;
- b_r_pinky2;
- b_l_pinky3;
- b_r_pinky3;
- b_l_ring1;
- b_r_ring1;
- b_l_ring2;
- b_r_ring2;
- b_l_ring3;
- b_r_ring3;
- b_l_thumb0;
- b_r_thumb0;
- b_l_thumb1;
- b_r_thumb1;
- b_l_thumb2;
- b_r_thumb2;
- b_l_thumb3;
- b_r_thumb3;
@param handType Тип руки:
    - None;
    - Right;
    - Left;
*/
[RequireComponent(typeof(PhotonView))]
public class GostHandTransformUpdater : ControllerModel
{
    [SerializeField] private HandType _handType;

    private string _prefix;

    private Transform _wrist;

    private Transform _index1;
    private Transform _index2;
    private Transform _index3;

    private Transform _middle1;
    private Transform _middle2;
    private Transform _middle3;

    private Transform _pinky0;
    private Transform _pinky1;
    private Transform _pinky2;
    private Transform _pinky3;

    private Transform _ring1;
    private Transform _ring2;
```

```
private Transform _ring3;

private Transform _thumb0;
private Transform _thumb1;
private Transform _thumb2;
private Transform _thumb3;

private Transform _serverWrist;

private Transform _serverIndex1;
private Transform _serverIndex2;
private Transform _serverIndex3;

private Transform _serverMiddle1;
private Transform _serverMiddle2;
private Transform _serverMiddle3;

private Transform _serverPinky0;
private Transform _serverPinky1;
private Transform _serverPinky2;
private Transform _serverPinky3;

private Transform _serverRing1;
private Transform _serverRing2;
private Transform _serverRing3;

private Transform _serverThumb0;
private Transform _serverThumb1;
private Transform _serverThumb2;
private Transform _serverThumb3;

private void Start()
{
    _prefix = "";
    switch (_handType)
    {
        case HandType.Left:
            _prefix = "b_l";
            break;
        case HandType.Right:
            _prefix = "b_r";
            break;
    }
    if (_prefix != "")
    {
        CreateHand();
    }
}

private void Update()
{
    if (_myPhotonView.IsMine)
    {
        MapPosition(_serverWrist, _wrist);

        MapPosition(_serverThumb0, _thumb0);
        MapPosition(_serverThumb1, _thumb1);
        MapPosition(_serverThumb2, _thumb2);
        MapPosition(_serverThumb3, _thumb3);

        MapPosition(_serverIndex1, _index1);
        MapPosition(_serverIndex2, _index2);
        MapPosition(_serverIndex3, _index3);

        MapPosition(_serverMiddle1, _middle1);
        MapPosition(_serverMiddle2, _middle2);
        MapPosition(_serverMiddle3, _middle3);

        MapPosition(_serverRing1, _ring1);
        MapPosition(_serverRing2, _ring2);
        MapPosition(_serverRing3, _ring3);

        MapPosition(_serverPinky0, _pinky0);
        MapPosition(_serverPinky1, _pinky1);
        MapPosition(_serverPinky2, _pinky2);
        MapPosition(_serverPinky3, _pinky3);
    }
}

private void CreateHand()
{
    ParseServerHand();
    FindLocalHand();
}

private void FindLocalHand()
{
    _wrist = GameObject.Find(_prefix + "_wrist").transform;

    _index1 = _wrist.Find(_prefix + "_index1").transform;
    _index2 = _index1.Find(_prefix + "_index2").transform;
    _index3 = _index2.Find(_prefix + "_index3").transform;

    _middle1 = _wrist.Find(_prefix + "_middle1").transform;
    _middle2 = _middle1.Find(_prefix + "_middle2").transform;
    _middle3 = _middle2.Find(_prefix + "_middle3").transform;

    _pinky0 = _wrist.Find(_prefix + "_pinky0").transform;
    _pinky1 = _pinky0.Find(_prefix + "_pinky1").transform;
    _pinky2 = _pinky1.Find(_prefix + "_pinky2").transform;
    _pinky3 = _pinky2.Find(_prefix + "_pinky3").transform;

    _ring1 = _wrist.Find(_prefix + "_ring1").transform;
```

```
        _ring2 = _ring1.Find(_prefix + "_ring2").transform;
        _ring3 = _ring2.Find(_prefix + "_ring3").transform;

        _thumb0 = _wrist.Find(_prefix + "_thumb0").transform;
        _thumb1 = _thumb0.Find(_prefix + "_thumb1").transform;
        _thumb2 = _thumb1.Find(_prefix + "_thumb2").transform;
        _thumb3 = _thumb2.Find(_prefix + "_thumb3").transform;
    }

    private void ParseServerHand()
    {
        _serverWrist = _objectModel.transform.Find(_prefix + "_wrist");

        _serverIndex1 = _serverWrist.Find(_prefix + "_index1").transform;
        _serverIndex2 = _serverIndex1.Find(_prefix + "_index2").transform;
        _serverIndex3 = _serverIndex2.Find(_prefix + "_index3").transform;

        _serverMiddle1 = _serverWrist.Find(_prefix + "_middle1").transform;
        _serverMiddle2 = _serverMiddle1.Find(_prefix + "_middle2").transform;
        _serverMiddle3 = _serverMiddle2.Find(_prefix + "_middle3").transform;

        _serverPinky0 = _serverWrist.Find(_prefix + "_pinky0").transform;
        _serverPinky1 = _serverPinky0.Find(_prefix + "_pinky1").transform;
        _serverPinky2 = _serverPinky1.Find(_prefix + "_pinky2").transform;
        _serverPinky3 = _serverPinky2.Find(_prefix + "_pinky3").transform;

        _serverRing1 = _serverWrist.Find(_prefix + "_ring1").transform;
        _serverRing2 = _serverRing1.Find(_prefix + "_ring2").transform;
        _serverRing3 = _serverRing2.Find(_prefix + "_ring3").transform;

        _serverThumb0 = _serverWrist.Find(_prefix + "_thumb0").transform;
        _serverThumb1 = _serverThumb0.Find(_prefix + "_thumb1").transform;
        _serverThumb2 = _serverThumb1.Find(_prefix + "_thumb2").transform;
        _serverThumb3 = _serverThumb2.Find(_prefix + "_thumb3").transform;
    }

    private void MapPosition(Transform target, Transform rigTransform)
    {
        target.position = rigTransform.position;
        target.rotation = rigTransform.rotation;
    }
}
```

HandsAnimaionUpdater.cs

```
using UnityEngine;
using Photon.Realtime;

/**
Скрипт обеспечивающий проигрывание анимации моделей рук на сервере

Данный класс обновляет анимацию моделей рук на сервере, в зависимости от жестов распознанных системой GestureAnimation.

@attention Для корректной работы данный класс требует, что бы в сцене присутствовали скрипты:
- ComponentCatcher;
- NetworkVariables;
- GestureAnimation;
@see ControllerModel; GestureAnimation; GestureProperties; NetworkVariables; ComponentCatcher
 */
public class HandsAnimaionUpdater : ControllerModel
{
    /// Аниматор, который используется для анимации модели контроллера.
    [SerializeField] private Animator _animator;

    private GestureAnimation _gestureAnimation;

    private GestureProperties _fingers;

    private NetworkVariables _networkVariables;

    private void Start()
    {
        ComponentCatcher catcher = FindAnyObjectByType<ComponentCatcher>();

        _gestureAnimation = catcher?.GetGestureAnimator();
        if (_gestureAnimation)
        {
            _fingers = new GestureProperties();
            if (_controllerType == OVRInput.Controller.LTouch || _controllerType == OVRInput.Controller.LHand)
            {
                _gestureAnimation.LeftGestChange += ChangeHandPose;
                _fingers.Type = HandType.Left;
            }
            if (_controllerType == OVRInput.Controller.RTouch || _controllerType == OVRInput.Controller.RHand)
            {
                _gestureAnimation.RightGestChange += ChangeHandPose;
                _fingers.Type = HandType.Right;
            }
        }

        _networkVariables = catcher?.GetNetworkVariables();
        if (_networkVariables)
        {
            _networkVariables.OnNetworkVariablesUpdate += OnPlayerPropertiesUpdate;
        }
    }

    private void Update()
```

```
    {
        UpdateAnimation(_animator);
    }

    private void UpdateAnimation(Animator animator)
    {
        animator.SetBool("isThumbClosed", _fingers.Thumb);
        animator.SetBool("isIndexClosed", _fingers.Index);
        animator.SetBool("isMiddleClosed", _fingers.Middle);
        animator.SetBool("isRingClosed", _fingers.Ring);
        animator.SetBool("isPinkyClosed", _fingers.Pinky);
    }

    private void OnDestroy()
    {
        if (_gestureAnimation)
        {
            if (_controllerType == OVRInput.Controller.LTouch || _controllerType == OVRInput.Controller.LHand)
            {
                _gestureAnimation.LeftGestChange -= ChangeHandPose;
            }
            if (_controllerType == OVRInput.Controller.RTouch || _controllerType == OVRInput.Controller.RHand)
            {
                _gestureAnimation.RightGestChange -= ChangeHandPose;
            }
        }

        if (_networkVariables)
        {
            _networkVariables.OnNetworkVariablesUpdate -= OnPlayerPropertiesUpdate;
        }
    }

    private void ChangeHandPose(bool[] fingers)
    {
        if (_myPhotonView.IsMine)
        {
            ChangeLocalHandPose(fingers);
            NetworkVariables.SendPropertyToServer(PhotonServerActions.GESTURE_FINGERS, fingers);
        }
    }

    private void ChangeLocalHandPose(bool[] fingers)
    {
        _fingers.SetFromBoolArray(fingers);
    }

    private void OnPlayerPropertiesUpdate(Player targetPlayer, ExitGames.Client.Photon.Hashtable changedProps)
    {
        if (changedProps.ContainsKey(PhotonServerActions.GESTURE_FINGERS))
        {
            if (!_myPhotonView.IsMine)
            {
                if (targetPlayer == _myPhotonView.Owner)
                {
                    ChangeLocalHandPose((bool[])changedProps[PhotonServerActions.GESTURE_FINGERS]);
                }
            }
        }
    }
}
```

HandView.cs

```
using UnityEngine;

/**
 Компонент якорь для отображения контроллеров на сервере

К данному компоненту привязывается отображение соответствующего контроллера на сервере.
То есть его необходимо расположить в месте с контроллером локального игрока.
@param handType HandType контроллера, который будет привязан к этому компоненту.
@param catcher ComponentCatcher находящийся в данной сцене.
 */
public class HandView : MonoBehaviour
{
    [SerializeField] private HandType _handType;
    [SerializeField] private ComponentCatcher _catcher;

    /**
     Геттер HandType.
    @return HandType
    - None;
    - Right;
    - Left;
     */
    public HandType GetHandType()
    {
        return _handType;
    }

    /**
     Геттер ControllerEvents.
    @return ControllerEvents данного игрока.
     */
    public ControllerEvents GetControllerSwitcher()
    {
        if (_catcher == null)
```

```
        {
            return null;
        }

        return _catcher.GetControllerEvents();
    }
}
```

EasySingleton.cs

```
using UnityEngine;

/**
Компонент обеспечивающий паттерн Singleton для того объекта, к которому он применен

Паттерн Singleton - порождающий паттерн, который гарантирует,
что для определенного класса будет создан только один объект,
а также предоставит к этому объекту точку доступа.
 */
public class EasySingleton : MonoBehaviour
{
    /// Единственный статичный экземпляр данного объекта
    public static EasySingleton Instance = null;

    private void Start()
    {
        SingleToneOnStart();
    }

    private void SingleToneOnStart()
    {
        if (Instance == null)
        {
            Instance = this;
        }
        else
        {
            if (Instance != this)
            {
                Destroy(this.gameObject);
            }
        }
        DontDestroyOnLoad(gameObject);
    }
}
```

MicrophoneController.cs

```
using UnityEngine;
using UnityEngine.UI;
using Photon.Voice.Unity;
using Photon.Realtime;

/**
 Класс отвечающий за взаимодействие с локальным микрофоном

@attention Для корректной работы данный класс требует, что бы в сцене присутствовали скрипты:
- NetworkVariables;

@param catcher ComponentCatcher находящийся в данной сцене.
@param microphoneOnImage Изображение включенного микрофона.
@param microphoneOffImage Изображение выключенного микрофона.
@param microphone Recorder микрофона из Photon.Voice.Unity.
@see ComponentCatcher; NetworkVariables;
 */
public class MicrophoneController : MonoBehaviour
{
    [SerializeField] private ComponentCatcher _catcher;
    [SerializeField] private Image _microphoneOnImage;
    [SerializeField] private Image _microphoneOffImage;
    private Recorder _microphone;

    private bool isMicrophoneActive;
    private float _currentMicrophoneVolume;
    private NetworkVariables _networkVariables;

    private void Awake()
    {
        _currentMicrophoneVolume = 1.0f;
    }

    private void Start()
    {
        _microphone = _catcher.GetRecorder();
        _networkVariables = _catcher.GetNetworkVariables();
        if (_networkVariables)
        {
            _networkVariables.OnNetworkVariablesUpdate += OnPlayerPropertiesUpdate;
        }
        isMicrophoneActive = true;
        _microphoneOnImage.enabled = isMicrophoneActive;
        _microphoneOffImage.enabled = !_microphoneOnImage.enabled;
```

```
        if (_microphone != null)
        {
            _microphone.RecordingEnabled = isMicrophoneActive;
        }
    }

    private void OnDestroy()
    {
        if (_networkVariables)
        {
            _networkVariables.OnNetworkVariablesUpdate -= OnPlayerPropertiesUpdate;
        }
    }

    /// Вкл/выкл микрофон.
    public void SwitchMicrophoneActivity()
    {
        isMicrophoneActive = !isMicrophoneActive;
        _microphoneOnImage.enabled = isMicrophoneActive;
        _microphoneOffImage.enabled = !_microphoneOnImage.enabled;
        if (_microphone != null)
        {
            _microphone.RecordingEnabled = isMicrophoneActive;
        }
    }

    /**
    Изменить громкость микрофона через UI компонент Slider.
    @param [in] slider Slider, в соответствии со значением которого устанавливается текущая громкость микрофона.
     */
    public void ChangeMicrophoneVolume(Slider slider)
    {
        _currentMicrophoneVolume = (float)slider.value;
        NetworkVariables.SendPropertyToServer(PhotonServerActions.MICROPHONE_VOLUME, _currentMicrophoneVolume);
    }

    private void OnPlayerPropertiesUpdate(Player targetPlayer, ExitGames.Client.Photon.Hashtable changedProps)
    {
        if (changedProps.ContainsKey(PhotonServerActions.UPDATE_STATUS))
        {
            NetworkVariables.SendPropertyToServer(PhotonServerActions.MICROPHONE_VOLUME, _currentMicrophoneVolume);
        }
    }
}
```

MicrophoneNetworkSettings.cs

```
using UnityEngine;
using Photon.Voice.Unity;
using Photon.Pun;
using Photon.Realtime;

/**
Класс взаимодействия с компонентами воспроизводящими звук с микрофона на сервере

Данный класс дополняет класс MicrophoneController. Он обрабатывает запросы микрофона пришедшие на сервер.

@attention Для корректной работы данный класс требует, что бы в сцене присутствовали скрипты:
- ComponentCatcher;
- NetworkVariables;
@see ComponentCatcher; NetworkVariables
 */
[RequireComponent(typeof(Speaker)), RequireComponent(typeof(AudioSource))]
public class MicrophoneNetworkSettings : MonoBehaviour
{
    [SerializeField] private PhotonView _myPhotonView;

    private AudioSource _audioSource;
    private NetworkVariables _networkVariables;

    private void Start()
    {
        ComponentCatcher catcher = FindObjectOfType<ComponentCatcher>();
        _networkVariables = catcher?.GetNetworkVariables();
        if (_networkVariables)
        {
            _networkVariables.OnNetworkVariablesUpdate += OnPlayerPropertiesUpdate;
        }
        _audioSource = GetComponent<AudioSource>();
    }

    private void OnDestroy()
    {
        if (_networkVariables)
        {
            _networkVariables.OnNetworkVariablesUpdate -= OnPlayerPropertiesUpdate;
        }
    }

    private void ChangeVolume(float newVolume)
    {
        _audioSource.volume = Mathf.Clamp(newVolume, 0.0f, 1.0f);
    }

    private void OnPlayerPropertiesUpdate(Player targetPlayer, ExitGames.Client.Photon.Hashtable changedProps)
    {
        if (changedProps.ContainsKey(PhotonServerActions.MICROPHONE_VOLUME))
        {
```

```
            if (!_myPhotonView.IsMine && targetPlayer == _myPhotonView.Owner)
            {
                ChangeVolume((float)changedProps[PhotonServerActions.MICROPHONE_VOLUME]);
            }
        }
    }
}
```

ObjectToMainCameraRotator.cs

```
using UnityEngine;

/**
 Компонент разварачивающий объект лицом к активной камере

@param camera Камера, к которой будет разворачиваться объект.
 */
public class ObjectToMainCameraRotator : MonoBehaviour
{
    [Header("[Optional]")]
    [SerializeField] private Camera _camera;

    private void Update()
    {
        if (_camera == null)
        {
            Camera[] sceneCameras = new Camera[10];
            Camera.GetAllCameras(sceneCameras);
            foreach (Camera camera in sceneCameras)
            {
                if (camera.enabled)
                {
                    _camera = camera;
                    break;
                }
            }
            if (_camera == null)
            {
                Debug.LogWarning("[" + this.name + "] Scene does not contain a camera");
                return;
            }
        }

        transform.LookAt(_camera.transform);
        transform.Rotate(Vector3.up * 180);
    }
}
```

PlayAudioWhenObjectMove.cs

```
using UnityEngine;


/**
 Класс, проигрывающий указанный AudioSource при изменении положения/поворота объекта

@param soundOfMoving AudioSource воспроизводимый при изменении положения объекта
@param soundOfRotation AudioSource воспроизводимый при изменении поворота объекта
@param positionSoundDistance Расстояние, через которое будет проигрываться звук.
Каждый раз, когда объект проходит данное расстояние, относительно предыдущего положения воспроизведения звука или начального
↪  положения, воспроизводится звук soundOfMoving.
@param rotationSoundAngle Угол, при повороте на который воспроизводится звук.
Каждый раз, когда объект совершает поворот, относительно предыдущего положения воспроизведения звука или начального положения,
↪  воспроизводится звук rotationSoundAngle.
 */
public class PlayAudioWhenObjectMove : MonoBehaviour
{
    [SerializeField] private AudioSource _soundOfMoving;
    [SerializeField] private AudioSource _soundOfRotation;

    [SerializeField] float _positionSoundDistance;
    [SerializeField] float _rotationSoundAngle;

    private Vector3 _previousPosition;
    private Vector3 _previousRotation;

    private void Start()
    {
        _previousPosition = new Vector3(transform.localPosition.x, transform.localPosition.y, transform.localPosition.z);
        _previousRotation = transform.rotation.eulerAngles;
    }

    private void Update()
    {
        if (_soundOfMoving)
        {
            float distance = Vector3.Distance(transform.localPosition, _previousPosition);
            if (distance > _positionSoundDistance)
            {
                _soundOfMoving.PlayOneShot(_soundOfMoving.clip);
                _previousPosition = new Vector3(transform.localPosition.x, transform.localPosition.y, transform.localPosition.z);
            }
```

```
        }

        if (_soundOfRotation)
        {
            Vector3 currentRotation = transform.rotation.eulerAngles;
            if (Mathf.Abs(_previousRotation.x - currentRotation.x) > _rotationSoundAngle)
            {
                _soundOfRotation.PlayOneShot(_soundOfRotation.clip);
                _previousRotation.x = currentRotation.x;
            }
            else
            {
                if (Mathf.Abs(_previousRotation.y - currentRotation.y) > _rotationSoundAngle)
                {
                    _soundOfRotation.PlayOneShot(_soundOfRotation.clip);
                    _previousRotation.y = currentRotation.y;
                }
                else
                {
                    if (Mathf.Abs(_previousRotation.z - currentRotation.z) > _rotationSoundAngle)
                    {
                        _soundOfRotation.PlayOneShot(_soundOfRotation.clip);
                        _previousRotation.z = currentRotation.z;
                    }
                }
            }
        }
    }
}
```

UIDisplayPlayerName.cs

```
using UnityEngine;
using Photon.Pun;
using TMPro;
/**
 Компонент, отображающий имя игрока в текстовом поле.

@param photonView Компонент PhotonView.
@param playerNameText Текстовое поле, в котором будет отображаться имя игрока.
 */
public class UIDisplayPlayerName : MonoBehaviour
{
    [SerializeField] private PhotonView _photonView;
    [SerializeField] private TMP_Text _playerNameText;

    private void Start()
    {
        _playerNameText.text = _photonView.Owner.NickName;
        if (_photonView.IsMine)
        {
            _playerNameText.GetComponentInParent<Canvas>().enabled = false;
        }
    }
}
```

BackFromFloor.cs

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


/**
 Класс возвращающий объект в стартовое положение, если объект упал на пол.

Полом является объект имеющий компонент Floor.
@attention Объект или его потомки должны иметь коллайдер.
@param returnSpeed Скорость, с которой объект вернется в исходное положение.
@param bindObject Объекты BackFromFloor, которые должны вернуться на свои места вместе с данным объектом.
Это нужно, если у нас сложный объект, в котором несколько независимых/частично зависимых частей.
И если одна из таких частей упала на пол - значит необходимо вернуть весь объект в исходное положение.
А значит необходимо вернуть в исходное положение все связанные с этим объектом части.
 */
[RequireComponent(typeof(Rigidbody))]
public class BackFromFloor : MonoBehaviour
{
    [SerializeField] private float _returnSpeed = 1f;
    [SerializeField] private List<BackFromFloor> _bindObjects;

    private Vector3 _startPosition;
    private Vector3 _startRotation;
    private Rigidbody _rigidBody;
    private List<Collider> _colliders;

    private bool _isBackToStartPosition;

    private void Start()
    {
        _isBackToStartPosition = false;
        _startPosition = new Vector3(transform.position.x, transform.position.y, transform.position.z);
        _startRotation = new Vector3(transform.rotation.x, transform.rotation.y, transform.rotation.z);
        _rigidBody = GetComponent<Rigidbody>();
```

```
        _colliders = new List<Collider>();
        _colliders.AddRange(GetComponentsInChildren<Collider>());
        if (TryGetComponent(out Collider collider))
        {
            _colliders.Add(collider);
        }
    }

    private void OnCollisionEnter(Collision collision)
    {
        if (collision.gameObject.TryGetComponent(out Floor floor))
        {
            BackToStartPosition();
        }
    }

    /// Метод для возвращения объекта в исходное положение
    public void BackToStartPosition()
    {
        if (!_isBackToStartPosition)
        {
            _isBackToStartPosition = true;
            foreach (BackFromFloor bindObject in _bindObjects)
            {
                bindObject.BackToStartPosition();
            }
            SetColliderState(false);
            _rigidBody.isKinematic = true;
            transform.LeanMove(_startPosition, _returnSpeed).setOnComplete(() =>
            {
                _isBackToStartPosition = false;
                SetColliderState(true);
                _rigidBody.isKinematic = false;
            });
            transform.LeanRotate(_startRotation, _returnSpeed);
        }
    }

    private void SetColliderState(bool isActive)
    {
        foreach (Collider collider in _colliders)
        {
            collider.enabled = isActive;
        }
    }
}
```

ElectricalCircuit.cs

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

/**
Класс электрической цепи.

Данный класс должен быть прикреплен к источнику электрического тока PowerSupply.
Т.к. электрическая цепь начинается и заканчивается на источнике тока.

Данный класс просчитывает электрическую цепь из элементов, последовательно подключенных друг к другу,
начиная с источника электрического тока PowerSupply. В цепи просчитывается общее сопротивление и напряжение.
Если цепь замыкается, то есть последовательное соединение из более чем 2-х ElectricalElement,
начинается в PowerSupply и заканчивается в этом же PowerSupply, то происходит расчет силы тока в цепи,
согласно закону Ома: I=U/R.
@see PowerSupply; ElectricalElement
 */
[RequireComponent(typeof(PowerSupply))]
public class ElectricalCircuit : MonoBehaviour
{
    private List<ElectricalElement> _activeElements;
    private Dictionary<ElectricalElement, List<ElectricalElement>> _elementGraph;
    private float _amperage;
    private float _voltage;
    private float _resistance;
    private bool _isClosed;
    private PowerSupply _startElement;

    private void Start()
    {
        _activeElements = new List<ElectricalElement>();
        _elementGraph = new Dictionary<ElectricalElement, List<ElectricalElement>>();
        _startElement = GetComponent<PowerSupply>();
        _startElement.SetElectricCircuit(this);
        _activeElements.Add(_startElement);
    }

    /**
     Геттер состояния замкнутости электрической цепи.
    @return bool Замкнута ли электрическая цепь.
     */
    public bool IsCircuitClosed()
    {
        return _isClosed;
    }

    /**
     Получить силу тока в электрической цепи.
    @return float Сила тока в электрической цепи.
     */
```

```
public float GetAmperage()
{
    if (_isClosed)
    {
        return _amperage;
    }
    return 0;
}

/**
 Получить напряжение в электрической цепи.
@return float Напряжение в электрической цепи.
 */
public float GetVoltage()
{
    if (_isClosed)
    {
        return _voltage;
    }
    return 0;
}

/**
 Пересчитать электрическую цепь.

Данный метод обходит все элементы ElectricalElement, последовательно подключенные друг к другу,
начиная с minus выхода PowerSupply. Во всей цепи происходит подсчет общего сопротивления.
И если по окончании обхода цепи, метод вышел к стартовому элементу PowerSupply,
то цепь считается замкнутой и происходит перерасчет силы тока в цепи.
Если цепь не замкнута, то сила тока в цепи считается 0.
 */
public void RecalculateCircuit()
{
    _resistance = 0;
    _voltage = 0;
    _amperage = 0;

    foreach (KeyValuePair<ElectricalElement, List<ElectricalElement>> element in _elementGraph)
    {
        element.Key.SetElectricCircuit(null);
    }
    _elementGraph.Clear();
    CreateElementGraph(_startElement, _startElement.GetMinusConnectedWire());
    _activeElements.Clear();
    if (!_elementGraph.ContainsKey(_startElement))
    {
        _isClosed = false;
    }
    else
    {
        _isClosed = CalculateElectricalCircuit(_elementGraph[_startElement][0]);
        if (_isClosed)
        {
            _amperage = _voltage / _resistance;
        }
    }
}

private void CreateElementGraph(ElectricalElement from, ElectricalElement to)
{
    if (to == null)
    {
        return;
    }

    if (!_elementGraph.ContainsKey(from))
    {
        _elementGraph.Add(from, new List<ElectricalElement>());
    }
    else
    {
        if (_elementGraph[from].Contains(to))
        {
            return;
        }
    }

    to.SetElectricCircuit(this);
    _elementGraph[from].Add(to);

    if (to.GetConnectedElectricalElements().Count > 0)
    {
        ElectricityTransfer electricityTransfer = to as ElectricityTransfer;
        if (electricityTransfer == null)
        {
            foreach (ElectricalElement element in to.GetConnectedElectricalElements())
            {
                if (element != from)
                {
                    CreateElementGraph(to, element);
                }
            }
        }
        else
        {
            foreach (ElectricalElement element in electricityTransfer.GetMinusWireElements())
            {
                if (element != from)
                {
                    CreateElementGraph(to, element);
                }
            }
        }
```

```
        }
    }

    private bool CalculateElectricalCircuit(ElectricalElement currentElement)
    {
        if (currentElement == null || !_elementGraph.ContainsKey(currentElement))
        {
            return false;
        }

        if (currentElement == _startElement)
        {
            PowerSupply powerSupply = currentElement as PowerSupply;
            if (!_activeElements.Contains(currentElement))
            {
                _voltage += powerSupply.GetVoltage();
                _activeElements.Add(currentElement);
            }
            return true;
        }

        bool isCircuitClose = false;
        foreach (ElectricalElement electricalElement in _elementGraph[currentElement])
        {
            isCircuitClose = isCircuitClose || CalculateElectricalCircuit(electricalElement);
        }

        ConfigureCircuitWithElement(currentElement, isCircuitClose);

        return isCircuitClose;
    }

    private void ConfigureCircuitWithElement(ElectricalElement element, bool isCircuitClose)
    {
        switch (element.GetName())
        {
            case ElectricalElement.RESISTOR_AND_RHEOSTAT:
                Resistor resistor = element as Resistor;
                if (isCircuitClose)
                {
                    _resistance += resistor.GetResistence();
                }
                break;
            case ElectricalElement.POWER_SUPPLY:
                PowerSupply powerSupply = element as PowerSupply;
                if (isCircuitClose)
                {
                    _voltage += powerSupply.GetVoltage();
                }
                break;
            case ElectricalElement.VOLTMETER:
                Voltmeter voltmeter = element as Voltmeter;

                break;
            case ElectricalElement.AMPERMETER:
                Milliammeter ampermeter = element as Milliammeter;

                break;
            case ElectricalElement.WIRE:
                WireElement wire = element as WireElement;

                break;
        }
    }
}
```

ElectricalElement.cs

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

/**
 Интерфейс элементов, которые в реальном времени определяют направление тока внутри себя.

WireElement не определяет направление в реальном времени.
@see WireElement
 */
public interface ElectricityTransfer
{
    /**
     Получить провода, в которые утекает ток.
    @return Список WireElement, в которые утекает ток.
     */
    List<WireElement> GetMinusWireElements();
}

/**
 Суперкласс всех элементов электрической схемы ElectricalCircuit
@see ElectricalCircuit; ElectricityTransfer;
 */
public class ElectricalElement : MonoBehaviour
{
    /// Имя элемента Источник питания
    public const string POWER_SUPPLY = "PowerSupply";
    /// Имя элемента Вольтметр
    public const string VOLTMETER = "Voltmeter";
    /// Имя элемента Амперметр
    public const string AMPERMETER = "Ampermeter";
```

```
    /// Имя элементов Резистор и Реостат
    public const string RESISTOR_AND_RHEOSTAT = "Resistor";
    /// Имя элемента Мультиклемма
    public const string MULTY_CLEMA = "MultiClema";
    /// Имя элемента Провод
    public const string WIRE = "WireElement";

    /// Электрическая цепь, в которой находится данный элемент
    protected ElectricalCircuit _electricalCircuit;
    /// Элементы, которые подключены к данному элементу
    private List<ElectricalElement> _connectedElements;
    /// Имя данного элемента
    protected string _name;

    protected void Awake()
    {
        _connectedElements = new List<ElectricalElement>();
        _electricalCircuit = null;
    }

    /**
     Геттер имени данного элемента.
    @return string Имя данного элемента.
     */
    public string GetName()
    {
        return _name;
    }

    /**
     Геттер элементов, подключенных к данному элементу
    @return List<ElectricalElement> подключенных к данному элементу.
     */
    public List<ElectricalElement> GetConnectedElectricalElements()
    {
        return _connectedElements;
    }

    /**
     Сеттер электрической цепи, в которой находится данный элемент.
    @param [in] electricalCircuit ElectricalCircuit , в которой находится данный элемент.
     */
    public void SetElectricCircuit(ElectricalCircuit electricalCircuit)
    {
        _electricalCircuit = electricalCircuit;
    }

    /**
     Добавление элемента, подключенного к данному элементу.
    @param [in] electricalElement Добавляемый ElectricalElement.
     */
    public void AddConnectedElement(ElectricalElement electricalElement)
    {
        if (!_connectedElements.Contains(electricalElement))
        {
            _connectedElements.Add(electricalElement);
            _electricalCircuit?.RecalculateCircuit();
        }
    }

    /**
     Удаление элемента из списка элементов, подключенных к данному элементу.
    @param [in] electricalElement Удаляемый ElectricalElement.
     */
    public void RemoveConnectedElement(ElectricalElement electricalElement)
    {
        if (_connectedElements.Contains(electricalElement))
        {
            _connectedElements.Remove(electricalElement);
            _electricalCircuit?.RecalculateCircuit();
        }
    }

    /// Удалить ссылку на электрическую цепь, в которой находится данный элемент.
    public void RemoveEctricalCircuitLink()
    {
        _electricalCircuit = null;
    }

    /**
     Геттер электрической цепи, в которой находится данный элемент.
    @return ElectricalCircuit , в которой находится данный элемент.
     */
    public ElectricalCircuit GetelEctricalCircuit()
    {
        return _electricalCircuit ? _electricalCircuit : null;
    }
}
```

Floor.cs

```
using UnityEngine;

/**
 Компонент определяющий объект как пол.

Данный компонент необходим для работы класса BackFromFloor.
@see BackFromFloor
```

```
 */
public class Floor : MonoBehaviour
{
}
```

Milliammeter.cs

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

/**
Класс миллиамперметра.

По данному классу элемент в электронной цепи определяется как Амперметр.

Миллиамперметр выводит значение силы тока в замкнутой электронной цепи, к которой он подключен,
на шкалу ScaleWithPointer данного миллиамперметра.
@note Что бы миллиамперметр работал, он должен присутствовать в замкнутой электронной цепи ElectricalCircuit.
@param pluse WireInput миллиамперметра, из которого должен выходить ток.
Если в данный вход будет входить ток, то миллиамперметра будет показывать отрицательные значения.
@param minus WireInput миллиамперметра, в который должен входить ток.
Если из данного входа будет выходить ток, то миллиамперметра будет показывать отрицательные значения.
@param pointer Шкала измерений данного миллиамперметра ScaleWithPointer.
@see ScaleWithPointer; WireInput; WireElement; ElectricalElement; ElectricityTransfer; ElectricalCircuit
*/
public class Milliammeter : ElectricalElement, ElectricityTransfer
{
    [SerializeField] private WireInput _pluse;
    [SerializeField] private WireInput _minus;
    [SerializeField] private ScaleWithPointer _pointer;

    private int _multiplier;
    /// Множитель для перевода миллиампер в амперы
    private const int _valueScaler = 1000;


    /**
     Эта функция всегда вызывается до начала любых функций, а также сразу после инициализации prefab-а.
    В данной функции происходит инициализация параметров электрического элемента и специфических для вольтметра миллиамперметра.
     */
    protected new void Awake()
    {
        base.Awake();
        _name = AMPERMETER;
        _pluse.SetParent(this);
        _minus.SetParent(this);
        _multiplier = 0;
    }

    private void Update()
    {
        if (_electricalCircuit != null && _electricalCircuit.IsCircuitClosed())
        {
            _pointer.SetCurrentValue(_electricalCircuit.GetAmperage() * _valueScaler * _multiplier);
        }
        else
        {
            _pointer.SetCurrentValue(0);
        }

        UpdateElectricityStatus();
    }

    /**
     Установить множитель показаний миллиамперметра.

    Если ток течет от minus к pluse, то множитель равен (1).
    Если ток течет в обратном направлении, множитель равен (-1).
    @param [in] multiplier Множитель.
     */
    public void SetMultiplier(int multiplier)
    {
        _multiplier = multiplier;
    }

    /// Реализация метода интерфейса.
    public List<WireElement> GetMinusWireElements()
    {
        UpdateElectricityStatus();
        List<WireElement> wireElements = new List<WireElement>();
        if (_minus.GetConnectedWire() != null && _minus.GetConnectedWire().GetOutputType() == OutputType.Minuse)
        {
            wireElements.Add(_minus.GetConnectedWire().GetWire());
        }
        if (_pluse.GetConnectedWire() != null && _pluse.GetConnectedWire().GetOutputType() == OutputType.Minuse)
        {
            wireElements.Add(_pluse.GetConnectedWire().GetWire());
        }

        return wireElements;
    }

    private void UpdateElectricityStatus()
    {
        if (_minus.GetConnectedWire() == null)
        {
            if (_pluse.GetConnectedWire() != null && _pluse.GetConnectedWire().GetOutputType() == OutputType.Minuse)
            {
```

```
                _pluse.GetConnectedWire().SetOutputTypeForWholeWire(OutputType.None);
            }
        }
        else
        {
            if (_pluse.GetConnectedWire() == null)
            {
                if (_minus.GetConnectedWire().GetOutputType() == OutputType.Minuse)
                {
                    _minus.GetConnectedWire().SetOutputTypeForWholeWire(OutputType.None);
                }
            }
            else
            {
                if (_minus.GetConnectedWire().GetOutputType() == OutputType.Pluse)
                {
                    _multiplier = 1;
                    _pluse.GetConnectedWire().SetOutputTypeForWholeWire(OutputType.Minuse);
                }
                else
                {
                    if (_pluse.GetConnectedWire().GetOutputType() == OutputType.Pluse)
                    {
                        _multiplier = -1;
                        _minus.GetConnectedWire().SetOutputTypeForWholeWire(OutputType.Minuse);
                    }
                }
            }
        }
    }
}
```

MoveCloserIfDistanceIsLarge.cs

```
using UnityEngine;
using Oculus.Interaction;

/**
 Компонент для синхронизации относительного положения концов провода

Данный класс используется для разрешения ситуаций, когда оба конца провода isKinematic.
Это может произойти, когда один конец провода у нас в руках, а другой прикреплен к клемме.
В таком случае необходимо открепить второй конец провода от клеммы и притянуть его поближе к первому концу.

Данный класс так же поддерживает определенное расстояние maxDistance между данным объектом и объектом dependentObject.
Это необходимо для избегания глитчей провода. Если текущее расстояние между объектами больше указанного,
и данный объект игрок держит в руке, то dependentObject будет притягиваться к данному объекту со скоростью speed.

@note Данный компонент стоит применять на оба объекта, которые предполагается притягивать.

@note Притяжение к данному объекту будет действовать только если игрок взял этот объект в руки.

@param dependentObject Объект, который будет притягиваться к данному объекту.
@param maxDistance Расстояние, превышая которое, dependentObject начинает притягиваться к данному объекту.
@param speed Скорость, с которой dependentObject притягивается к данному объекту
@see WireOutput; WireInput
 */
[RequireComponent(typeof(Grabbable))]
public class MoveCloserIfDistanceIsLarge : MonoBehaviour
{
    [SerializeField] Transform _dependentObject;
    [SerializeField] private float _maxDistance = -1f;
    [SerializeField] private float _speed = 1f;

    private bool _isGrab;
    private Grabbable _grabbable;

    private void Start()
    {
        if (_maxDistance < 0f)
        {
            _maxDistance = Vector3.Distance(transform.position, _dependentObject.position);
        }
        _grabbable = GetComponent<Grabbable>();
        _grabbable.WhenPointerEventRaised += OnObjectGrabChange;
        _isGrab = false;
    }

    private void Update()
    {
        if (_isGrab)
        {
            if (Vector3.Distance(transform.position, _dependentObject.position) > _maxDistance)
            {
                if (_dependentObject.TryGetComponent(out WireOutput output))
                {
                    output.GetConnectedObject()?.RemoveWireFromInput();
                }
                _dependentObject.position = Vector3.MoveTowards(_dependentObject.position, transform.position, _speed *
                ↪  Time.deltaTime);
            }
        }
    }

    private void OnDestroy()
    {
        _grabbable.WhenPointerEventRaised -= OnObjectGrabChange;
    }
```

```
    private void OnObjectGrabChange(PointerEvent grabEvent)
    {
        switch (grabEvent.Type)
        {
            case PointerEventType.Select:
                if (!_isGrab)
                {
                    _isGrab = true;
                }
                break;
            case PointerEventType.Unselect:
                if (_isGrab)
                {
                    _isGrab = false;
                }
                break;
        }
    }
}
```

MultiClema.cs

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

/**
 Класс переходника на 3 входа.

Данный переходник необходим для подключения вольтметра (Voltmeter) в электрическую цепь.
Ведь вольтметр должен включаться параллельно.
@param leftInput Левый вход WireInput переходника.
@param rightInput Правый вход WireInput переходника.
@param parallelInput Вход WireInput находящийся по середине переходника.
К данному входу должен подключаться вольтметр (Voltmeter).
@see WireInput; WireElement; ElectricalElement; ElectricityTransfer
 */
public class MultiClema : ElectricalElement, ElectricityTransfer
{
    [SerializeField] private WireInput _leftInput;
    [SerializeField] private WireInput _rightInput;
    [SerializeField] private WireInput _parallelInput;

    /**
     Эта функция всегда вызывается до начала любых функций, а также сразу после инициализации prefab-а.
    В данной функции происходит инициализация параметров электрического элемента и специфических для переходника параметров.
     */
    protected new void Awake()
    {
        base.Awake();
        _name = MULTY_CLEMA;
        _leftInput.SetParent(this);
        _rightInput.SetParent(this);
        _parallelInput.SetParent(this);
    }

    private void Update()
    {
        UpdateElectricityStatus();
    }

    /**
     Геттер провода, подключенного к левому входу переходника.
    @return WireElement, подключенный к левому входу переходника.
     */
    public WireElement GetLeftConnectedWire()
    {
        return _leftInput.GetConnectedWire()?.GetWire();
    }

    /**
     Геттер провода, подключенного к правому входу переходника.
    @return WireElement, подключенный к правому входу переходника.
     */
    public WireElement GetRightConnectedWire()
    {
        return _rightInput.GetConnectedWire()?.GetWire();
    }

    /**
     Геттер провода, подключенного к находящемуся по середине входу переходника.
    @return WireElement, подключенный к находящемуся по середине входу переходника.
     */
    public WireElement GetParallelConnectedWire()
    {
        return _parallelInput.GetConnectedWire()?.GetWire();
    }

    /// Реализация метода интерфейса.
    public List<WireElement> GetMinusWireElements()
    {
        UpdateElectricityStatus();

        List<WireElement> wireElements = new List<WireElement>();

        if (_leftInput.GetConnectedWire() != null && _leftInput.GetConnectedWire().GetOutputType() == OutputType.Minuse)
        {
            wireElements.Add(_leftInput.GetConnectedWire().GetWire());
```

```
        }
        if (_rightInput.GetConnectedWire() != null && _rightInput.GetConnectedWire().GetOutputType() == OutputType.Minuse)
        {
            wireElements.Add(_rightInput.GetConnectedWire().GetWire());
        }
        if (_parallelInput.GetConnectedWire() != null && _parallelInput.GetConnectedWire().GetOutputType() == OutputType.Minuse)
        {
            wireElements.Add(_parallelInput.GetConnectedWire().GetWire());
        }

        return wireElements;
    }

    private void UpdateElectricityStatus()
    {
        if (_leftInput.GetConnectedWire() == null)
        {
            if (_rightInput.GetConnectedWire() != null && _rightInput.GetConnectedWire().GetOutputType() == OutputType.Minuse)
            {
                _rightInput.GetConnectedWire().SetOutputTypeForWholeWire(OutputType.None);
            }
        }
        else
        {
            if (_rightInput.GetConnectedWire() == null)
            {
                if (_leftInput.GetConnectedWire().GetOutputType() == OutputType.Minuse)
                {
                    _leftInput.GetConnectedWire().SetOutputTypeForWholeWire(OutputType.None);
                }
            }
            else
            {
                if (_leftInput.GetConnectedWire().GetOutputType() == OutputType.Pluse)
                {
                    _rightInput.GetConnectedWire().SetOutputTypeForWholeWire(OutputType.Minuse);
                }
                else
                {
                    if (_rightInput.GetConnectedWire().GetOutputType() == OutputType.Pluse)
                    {
                        _leftInput.GetConnectedWire().SetOutputTypeForWholeWire(OutputType.Minuse);
                    }
                }
            }
        }

        if (_parallelInput.GetConnectedWire() != null)
        {
            if (_parallelInput.GetConnectedWire().GetOutputType() == OutputType.None)
            {
                _parallelInput.GetConnectedWire().SetOutputTypeForWholeWire(OutputType.Minuse);
            }
        }
    }
}
```

PowerSupply.cs

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

/**
 Класс электрического тока в цепи.
 */
[System.Serializable]
public class Electricity
{
    /// Сила тока.
    public float Amperage;
    /// Напряжение.
    public float Voltage;

    /**
     Скопировать параметры другого класса электричества.
    @param [in] electricity Класс электрического тока параметры которого необходимо скопировать.
     */
    public void CopyElectricity(Electricity electricity)
    {
        Amperage = electricity.Amperage;
        Voltage = electricity.Voltage;
    }
}

/**
 Класс источника электрического тока.

Источник тока постоянно посылает сгенерированный им ток на minus выход.
@param generatedElectricity Electricity порождаемое данным источником тока.
@param pluse WireInput источника тока, в который должен входить ток.
@param minus WireInput источника тока, из которого выходит ток.
@see Electricity; WireInput; ElectricalElement; ElectricityTransfer; WireElement;
 */
public class PowerSupply : ElectricalElement, ElectricityTransfer
{
    [SerializeField] private Electricity _generatedElectricity;
    [SerializeField] private WireInput _pluse;
    [SerializeField] private WireInput _minus;
```

```
    /**
     Эта функция всегда вызывается до начала любых функций, а также сразу после инициализации prefab-а.
    В данной функции происходит инициализация параметров электрического элемента и специфических для источника электрического тока
    ↪  параметров.
     */
    protected new void Awake()
    {
        base.Awake();
        _name = POWER_SUPPLY;
        _pluse.SetParent(this);
        _minus.SetParent(this);
    }

    private void Update()
    {
        UpdateElectricityStatus();
    }

    /**
     Получить силу тока, генерируемую данным источником тока.
    @return float Сила тока, генерируемая данным источником тока.
     */
    public float GetAmperage()
    {
        return _generatedElectricity.Amperage;
    }

    /**
     Получить напряжение, генерируемое данным источником тока.
    @return float Напряжение, генерируемое данным источником тока.
     */
    public float GetVoltage()
    {
        return _generatedElectricity.Voltage;
    }

    /**
     Получить провод, подключенный к minus выходу источника электрического тока.
    @return WireElement, подключенный к minus выходу источника электрического тока.
     */
    public WireElement GetMinusConnectedWire()
    {
        return _minus.GetConnectedWire()?.GetWire();
    }

    /// Реализация метода интерфейса.
    public List<WireElement> GetMinusWireElements()
    {
        List<WireElement> wireElements = new List<WireElement>();
        if (_minus.GetConnectedWire() != null)
        {
            wireElements.Add(_minus.GetConnectedWire().GetWire());
        }

        return wireElements;
    }

    private void UpdateElectricityStatus()
    {
        _minus.GetConnectedWire()?.SetOutputTypeForWholeWire(OutputType.Minuse);
    }
}
```

Resistor.cs

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

/**
Класс резистора

По данному классу элемент в электронной цепи определяется как Резистор.

Резистор - пассивный элемент электрических цепей, обладающий определённым или переменным значением электрического сопротивления.

@param leftInput Условно левый вход WireInput резистора.
@param rightInput Условно правый вход WireInput резистора.
@param resistance Сопротивление резистора
@see WireInput; WireElement; ElectricalElement; ElectricityTransfer
 */
public class Resistor : ElectricalElement, ElectricityTransfer
{
    [SerializeField] private WireInput _leftInput;
    [SerializeField] private WireInput _rightInput;

    [SerializeField] protected float _resistance;

    /**
     Эта функция всегда вызывается до начала любых функций, а также сразу после инициализации prefab-а.
    В данной функции происходит инициализация параметров электрического элемента и специфических для резистора параметров.
     */
    protected new void Awake()
    {
        base.Awake();
        _name = RESISTOR_AND_RHEOSTAT;
        _leftInput.SetParent(this);
        _rightInput.SetParent(this);
```

```
    }

    /**
     Метод вызываемый один раз за кадр.
    В данном методе в реальном времени определяется направление тока внутри резистора.
     */
    protected void Update()
    {
        UpdateElectricityStatus();
    }

    /**
     Получить сопротивление резистора.
    @return float Сопротивление резистора.
     */
    public float GetResistence()
    {
        return _resistance;
    }

    /// Реализация метода интерфейса.
    public List<WireElement> GetMinusWireElements()
    {
        UpdateElectricityStatus();
        List<WireElement> wireElements = new List<WireElement>();
        if (_leftInput.GetConnectedWire() != null && _leftInput.GetConnectedWire().GetOutputType() == OutputType.Minuse)
        {
            wireElements.Add(_leftInput.GetConnectedWire().GetWire());
        }
        if (_rightInput.GetConnectedWire() != null && _rightInput.GetConnectedWire().GetOutputType() == OutputType.Minuse)
        {
            wireElements.Add(_rightInput.GetConnectedWire().GetWire());
        }

        return wireElements;
    }

    private void UpdateElectricityStatus()
    {
        if (_leftInput.GetConnectedWire() == null)
        {
            if (_rightInput.GetConnectedWire() != null && _rightInput.GetConnectedWire().GetOutputType() == OutputType.Minuse)
            {
                _rightInput.GetConnectedWire().SetOutputTypeForWholeWire(OutputType.None);
            }
        }
        else
        {
            if (_rightInput.GetConnectedWire() == null)
            {
                if (_leftInput.GetConnectedWire().GetOutputType() == OutputType.Minuse)
                {
                    _leftInput.GetConnectedWire().SetOutputTypeForWholeWire(OutputType.None);
                }
            }
            else
            {
                if (_leftInput.GetConnectedWire().GetOutputType() == OutputType.Pluse)
                {
                    _rightInput.GetConnectedWire().SetOutputTypeForWholeWire(OutputType.Minuse);
                }
                else
                {
                    if (_rightInput.GetConnectedWire().GetOutputType() == OutputType.Pluse)
                    {
                        _leftInput.GetConnectedWire().SetOutputTypeForWholeWire(OutputType.Minuse);
                    }
                }
            }
        }
    }
}
```

Rheostat.cs

```
using UnityEngine;

/**
Класс реостата

По данному классу элемент в электронной цепи определяется как Реостат.

Реостат - электрический аппарат для регулирования и ограничения тока или напряжения в электрической цепи,
основная часть которого — проводящий элемент с переменным электрическим сопротивлением.

@param positionAxis Ось CoordinateAxis, по которой движется указатель реостата.
@param pointer Указатель реостата.
@param minDistance Минимальное положение указателя реостата. В этом положении сопротивление реостата будет минимальным.
@param maxDistance Максимальное положение указателя реостата. В этом положении сопротивление реостата будет максимальным.
@param step Шаг пересчета сопротивления. Когда указатель проходит данное расстояние, происходит пересчет сопротивления в
↪  соответствии с новым положением указателя.
Данный параметр позволяет не пересчитывать постоянно всю электронную цепь, при неизменном сопротивлении.
@param minValue Минимальное значение сопротивления. При нахождении указателя в позиции minDistance, реостат будет иметь данное
↪  сопротивление.
@param maxValue Максимальное значение сопротивления. При нахождении указателя в позиции maxDistance, реостат будет иметь данное
↪  сопротивление.
@see Resistor
 */
```

```
public class Rheostat : Resistor
{
    [Header("Pointer object")]
    [SerializeField] private CoordinateAxis _positionAxis;
    [SerializeField] private Transform _pointer;
    [SerializeField] private float _minDistance;
    [SerializeField] private float _maxDistance;
    [SerializeField] private float _step;
    [Header("Measurand")]
    [SerializeField] private float _minValue;
    [SerializeField] private float _maxValue;

    private Vector3 _previousPosition;
    private float _distance;
    private float _valueRange;
    private float _currentDistance;

    /**
     Эта функция всегда вызывается до начала любых функций, а также сразу после инициализации prefab-а.
    В данной функции происходит инициализация параметров электрического элемента и специфических для реостата параметров.
     */
    protected new void Awake()
    {
        base.Awake();
        _distance = _maxDistance - _minDistance;
        _valueRange = _maxValue - _minValue;
        _currentDistance = _minDistance;
        ReaclculateCurrentDistance();
        _resistance = Mathf.Clamp(_minValue + (_currentDistance / _distance) * _valueRange, _minValue, _maxValue);
    }

    /**
     Метод вызываемый один раз за кадр.
    В данном методе в реальном времени определяется направление тока внутри реостата.
    А также происходит подсчет сопротивления в зависимости от положения указателя.
     */
    protected new void Update()
    {
        base.Update();
        ReaclculateCurrentDistance();

        float distance = Vector3.Distance(_pointer.localPosition, _previousPosition);
        if (distance > _step)
        {
            _resistance = Mathf.Clamp(_minValue + (_currentDistance / _distance) * _valueRange, _minValue, _maxValue);
            _previousPosition = new Vector3(_pointer.localPosition.x, _pointer.localPosition.y, _pointer.localPosition.z);
            _electricalCircuit?.RecalculateCircuit();
        }
    }

    private void ReaclculateCurrentDistance()
    {
        switch (_positionAxis)
        {
            case CoordinateAxis.X:
                _currentDistance = Mathf.Clamp(_pointer.localPosition.x, _minDistance, _maxDistance) - _minDistance;
                break;
            case CoordinateAxis.Y:
                _currentDistance = Mathf.Clamp(_pointer.localPosition.y, _minDistance, _maxDistance) - _minDistance;
                break;
            case CoordinateAxis.Z:
                _currentDistance = Mathf.Clamp(_pointer.localPosition.z, _minDistance, _maxDistance) - _minDistance;
                break;
        }
    }
}
```

ScaleWithPointer.cs

```
using UnityEngine;

/// Ось Эйлеровых координат
public enum CoordinateAxis
{
    X, ///< Ось X
    Y, ///< Ось Y
    Z ///< Ось Z
}

/**
Класс шкалы с указателем.

Приборы на подобии амперметра имеют шкалу измерений.
Текущее значение указывается указателем.
Шкала представляет собой полукруг. А указатель стрелку, идущую из центра круга.
Что бы изменить значение показателя, нам нужно изменить поворот указателя.
Данный компонент управляет поворотом указателя, в зависимости от текущего значения измерений.
@param rotationAxis Ось поворота указателя CoordinateAxis.
@param pointer Объект указателя.
@param minAngle Минимальное значение угла поворота.
В данном значении указатель должен указывать на минимальное значение шкалы.
@param maxAngle Максимальное значение угла поворота.
В данном значении указатель должен указывать на максимальное значение шкалы.
@param speed Скорость изменения поворота указателя.
@param minValue Минимальное значение шкалы. На данное значение указывает указатель при повороте на minAngle.
@param maxValue Максимальное значение шкалы. На данное значение указывает указатель при повороте на maxAngle.
@see CoordinateAxis
 */
```

```
public class ScaleWithPointer : MonoBehaviour
{
    [Header("Pointer object")]
    [SerializeField] private CoordinateAxis _rotationAxis;
    [SerializeField] private Transform _pointer;
    [SerializeField] private float _minAngle;
    [SerializeField] private float _maxAngle;
    [SerializeField] private float _speed;
    [Header("Measurand")]
    [SerializeField] private float _minValue;
    [SerializeField] private float _maxValue;

    private float _currentValue;
    private float _angleDistance;
    private float _valueRange;
    private bool isPointerMoving;
    private float _lastValue;

    private void Start()
    {
        _angleDistance = _maxAngle - _minAngle;
        _valueRange = _maxValue - _minValue;
        isPointerMoving = false;
        _lastValue = 0;
    }

    private void Update()
    {
        if (!isPointerMoving)
        {
            SetPointer(_currentValue);
        }
    }

    /**
     Установить текущее значение шкалы.
    @param [in] value Текущее значение шкалы.
     */
    public void SetCurrentValue(float value)
    {
        _currentValue = value;
    }

    private void SetPointer(float value)
    {
        value = Mathf.Clamp(value, _minValue, _maxValue);
        if (value != _lastValue)
        {
            _lastValue = value;
            isPointerMoving = true;
            float rotationAngle = (value / _valueRange) * _angleDistance;
            switch (_rotationAxis)
            {
                case CoordinateAxis.X:
                    _pointer.LeanRotateX(rotationAngle, _speed).setOnComplete(() => isPointerMoving = false);
                    break;
                case CoordinateAxis.Y:
                    _pointer.LeanRotateY(rotationAngle, _speed).setOnComplete(() => isPointerMoving = false);
                    break;
                case CoordinateAxis.Z:
                    _pointer.LeanRotateZ(rotationAngle, _speed).setOnComplete(() => isPointerMoving = false);
                    break;
            }
        }
    }
}
```

Voltmeter.cs

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

/**
Класс вольтметра.

По данному классу элемент в электронной цепи определяется как Вольтметр.

Логика работы.

К вольтметру должны быть подключены провода WireElement.
Данные провода одним концом должны быть подключены к входам вольтметра,
другим концом к parallelInput MultiClema.

\image html VoltmetrSchema.png width=500
<div style = "text-align: center;" >
    Схема подключения вольтметра
</div>
Вольтметр будет подсчитывать сопротивление в цепи следующим образом.
Он начнет обходить провод по направлению тока, и суммировать сопротивление на резисторах (Resistor),
до тех пор, пока не найдет первую MultiClema.Дальше он пойдет по проводу подключенному к leftInput MultiClema.
Он так же суммирует сопративление на резисторах (Resistor), до тех пор, пока не найдет вторую MultiClema.
Теперь он пойдет по проводу подключенному к parallelInput MultiClema, т.к.вольтметр должен крепиться к этому входу MultiClema.
Вольтметр продолжает суммировать сопративление, пока не найдет себя. После этого он подсчитывает напряжение по закону Ома: U=
↪  A\*R.
Полученное напряжение выводится на шкалу ScaleWithPointer вольтметра.
@note Что бы вольтметр работал, он должен присутствовать в замкнутой электронной цепи ElectricalCircuit.
@param pluse WireInput вольтметра, из которого должен выходить ток.
Если в данный вход будет входить ток, то вольтметр будет показывать отрицательные значения.
```

```
@param minus WireInput вольтметра, в который должен входить ток.
Если из данного входа будет выходить ток, то вольтметр будет показывать отрицательные значения.
@param pointer Шкала измерений данного вольтметра ScaleWithPointer.
@see WireInput; ScaleWithPointer; ElectricalElement; ElectricityTransfer; WireElement; MultiClema; Resistor; ElectricalCircuit
*/
public class Voltmeter : ElectricalElement, ElectricityTransfer
{
    [SerializeField] private WireInput _pluse;
    [SerializeField] private WireInput _minus;
    [SerializeField] private ScaleWithPointer _pointer;
    private int _multiplier;
    private float _circuitCectionResistance;

    /**
     Эта функция всегда вызывается до начала любых функций, а также сразу после инициализации prefab-а.
    В данной функции происходит инициализация параметров электрического элемента и специфических для вольтметра параметров.
     */
    protected new void Awake()
    {
        base.Awake();
        _name = VOLTMETER;
        _pluse.SetParent(this);
        _minus.SetParent(this);
        _multiplier = 0;
    }

    private void Update()
    {
        if (_electricalCircuit != null && _electricalCircuit.IsCircuitClosed())
        {
            _circuitCectionResistance = 0;
            FindFirstMultiClema(this, _pluse.GetConnectedWire().GetWire());
        }
        else
        {
            _pointer.SetCurrentValue(0);
        }
        UpdateElectricityStatus();
    }

    private void FindFirstMultiClema(ElectricalElement from, ElectricalElement to)
    {
        if (to == null)
        {
            return;
        }
        MultiClema multiClema = to as MultiClema;
        if (multiClema != null)
        {
            FindLastMultiClema(multiClema, multiClema.GetLeftConnectedWire());
        }
        else
        {
            Resistor resistor = to as Resistor;
            if (resistor != null)
            {
                _circuitCectionResistance += resistor.GetResistence();
            }
            foreach (ElectricalElement electricalElement in to.GetConnectedElectricalElements())
            {
                if (electricalElement != from)
                {
                    FindFirstMultiClema(to, electricalElement);
                    break;
                }
            }
        }
    }

    private void FindLastMultiClema(ElectricalElement from, ElectricalElement to)
    {
        if (to == null)
        {
            return;
        }
        MultiClema multiClema = to as MultiClema;
        if (multiClema != null)
        {
            FindMyself(multiClema, multiClema.GetParallelConnectedWire());
        }
        else
        {
            Resistor resistor = to as Resistor;
            if (resistor != null)
            {
                _circuitCectionResistance += resistor.GetResistence();
            }
            foreach (ElectricalElement electricalElement in to.GetConnectedElectricalElements())
            {
                if (electricalElement != from)
                {
                    FindLastMultiClema(to, electricalElement);
                    break;
                }
            }
        }
    }

    private void FindMyself(ElectricalElement from, ElectricalElement to)
    {
        if (to == null)
        {
            return;
```

```
        }
        if (to == this)
        {
            _pointer.SetCurrentValue(_electricalCircuit.GetAmperage() * _circuitCectionResistance * _multiplier);
        }
        else
        {
            Resistor resistor = to as Resistor;
            if (resistor != null)
            {
                _circuitCectionResistance += resistor.GetResistence();
            }
            foreach (ElectricalElement electricalElement in to.GetConnectedElectricalElements())
            {
                if (electricalElement != from)
                {
                    FindMyself(to, electricalElement);
                    break;
                }
            }
        }
    }

    /**
     Установить множитель показаний вольтметра.

    Если ток течет от minus к pluse, то множитель равен (1).
    Если ток течет в обратном направлении, множитель равен (-1).
    @param [in] multiplier Множитель.
     */
    public void SetMultiplier(int multiplier)
    {
        _multiplier = multiplier;
    }

    /// Реализация метода интерфейса.
    public List<WireElement> GetMinusWireElements()
    {
        UpdateElectricityStatus();
        List<WireElement> wireElements = new List<WireElement>();
        if (_minus.GetConnectedWire() != null && _minus.GetConnectedWire().GetOutputType() == OutputType.Minuse)
        {
            wireElements.Add(_minus.GetConnectedWire().GetWire());
        }
        if (_pluse.GetConnectedWire() != null && _pluse.GetConnectedWire().GetOutputType() == OutputType.Minuse)
        {
            wireElements.Add(_pluse.GetConnectedWire().GetWire());
        }

        return wireElements;
    }

    private void UpdateElectricityStatus()
    {
        if (_minus.GetConnectedWire() == null)
        {
            if (_pluse.GetConnectedWire() != null && _pluse.GetConnectedWire().GetOutputType() == OutputType.Minuse)
            {
                _pluse.GetConnectedWire().SetOutputTypeForWholeWire(OutputType.None);
            }
        }
        else
        {
            if (_pluse.GetConnectedWire() == null)
            {
                if (_minus.GetConnectedWire().GetOutputType() == OutputType.Minuse)
                {
                    _minus.GetConnectedWire().SetOutputTypeForWholeWire(OutputType.None);
                }
            }
            else
            {
                if (_minus.GetConnectedWire().GetOutputType() == OutputType.Pluse)
                {
                    _multiplier = 1;
                    _pluse.GetConnectedWire().SetOutputTypeForWholeWire(OutputType.Minuse);
                }
                else
                {
                    if (_pluse.GetConnectedWire().GetOutputType() == OutputType.Pluse)
                    {
                        _multiplier = -1;
                        _minus.GetConnectedWire().SetOutputTypeForWholeWire(OutputType.Minuse);
                    }
                }
            }
        }
    }
}
```

WireElement.cs

```
using UnityEngine;

/**
 Класс провода

@param output1 Первый конец провода.
```

```
@param output1 Второй конец провода.
@see WireOutput; ElectricalElement; OutputType
 */
public class WireElement : ElectricalElement
{
    [SerializeField] private WireOutput _output1;
    [SerializeField] private WireOutput _output2;

    protected new void Awake()
    {
        base.Awake();
        _name = WIRE;
        _output1.SetWire(this);
        _output2.SetWire(this);
    }

    /**
     Обновить направление тока в проводе.

    Данный метод вызывается одним из концов провода WireOutput.
    Метод устанавливает второму концу провода направление,
    противоположное направлению в конце провода, вызвавшего данный метод.
    @param [in] outputType OutputType конца провода, который вызвал данный метод.
    @param [in] sender WireOutput вызвавший данный метод.
     */
    public void UpdeteWireType(OutputType outputType, WireOutput sender)
    {
        if (outputType == OutputType.None)
        {
            _output1.SetOutputTypeToOneOutput(OutputType.None);
            _output2.SetOutputTypeToOneOutput(OutputType.None);
        }
        else
        {
            if (sender == _output1)
            {
                if (outputType == OutputType.Pluse)
                {
                    _output2.SetOutputTypeToOneOutput(OutputType.Minuse);
                }
                else
                {
                    _output2.SetOutputTypeToOneOutput(OutputType.Pluse);
                }
            }
            else
            {
                if (outputType == OutputType.Pluse)
                {
                    _output1.SetOutputTypeToOneOutput(OutputType.Minuse);
                }
                else
                {
                    _output1.SetOutputTypeToOneOutput(OutputType.Pluse);
                }
            }
        }
    }
}
```

WireInput.cs

```
using UnityEngine;

/**
Компонент, обозначающий данный объект местом для крепления провода

Концы провода содержат компонент WireOutput. Именно он и прикрепляется к данному объекту.

Объекты, содержащие данный компонент, используются для проведения тока, идущего по проводу WireElement
к ElectricalElement и наоборот.

@param plugPosition Позиция, в которую будет прикреплен провод.
@param unplugPosition Позиция, в которой окажется провод, при откреплении.
Данную позицию нужно выбрать так, чтобы при попадании в нее коллайдер провода не соприкасался с коллайдером данного объекта.
@param breakingDistance Расстояние, на которое должен отклониться провод от plugPosition, чтобы считать, что он откреплен.
@param connectedAudio Опциональный параметр. AudioSource воспроизводимый при прикреплении провода.
@param disconnectedAudio Опциональный параметр. AudioSource воспроизводимый при откреплении провода.
@see WireOutput; ElectricalElement
 */
public class WireInput : MonoBehaviour
{
    [SerializeField] private Transform _plugPosition;
    [SerializeField] private Transform _unplugPosition;
    [SerializeField] private float _breakingDistance = 0.02f;
    [Header("[Optional]")]
    [SerializeField] private AudioSource _connectedAudio;
    [Header("[Optional]")]
    [SerializeField] private AudioSource _disconnectedAudio;

    private bool _isConnected = false;
    private WireOutput _connectedWire;
    private ElectricalElement _parent;

    private void Update()
    {
        if (_connectedWire != null)
        {
```

```
            if (Vector3.Distance(_connectedWire.transform.position, _plugPosition.position) > _breakingDistance)
            {
                RemoveWireFromInput();
            }
            else
            {
                _connectedWire.transform.position = new Vector3(_plugPosition.position.x, _plugPosition.position.y,
                ↪  _plugPosition.position.z);
            }
        }
    }

    private void OnTriggerEnter(Collider other)
    {
        if (!_isConnected)
        {
            if (other.gameObject.TryGetComponent(out WireOutput wire))
            {
                ConnectWire(wire);
            }
        }
    }

    /// Открепить провод от объекта
    public void RemoveWireFromInput()
    {
        if (_isConnected && _connectedWire)
        {
            DisconnectWire(_connectedWire);
        }
    }

    /**
     Получить WireOutput прикрепленный к данному объекту
    @return WireOutput прикрепленный к данному объекту либо null,
    если к данному объекту не прикреплен провод.
     */
    public WireOutput GetConnectedWire()
    {
        return _connectedWire ? _connectedWire : null;
    }

    private void ConnectWire(WireOutput wire)
    {
        _isConnected = true;
        _connectedWire = wire;
        if (_connectedWire.gameObject.TryGetComponent(out Rigidbody rigidbody))
        {
            rigidbody.isKinematic = _isConnected;
            rigidbody.useGravity = !_isConnected;
        }
        wire.SetConnectedObject(this);
        if (_connectedAudio != null && _connectedAudio.enabled)
        {
            _connectedAudio?.PlayOneShot(_connectedAudio?.clip);
        }
        _parent?.AddConnectedElement(wire.GetWire());
        wire.GetWire().AddConnectedElement(_parent);
        _connectedWire.transform.position = new Vector3(_plugPosition.position.x, _plugPosition.position.y,
        ↪  _plugPosition.position.z);
    }

    private void DisconnectWire(WireOutput wire)
    {
        _isConnected = false;
        wire.SetConnectedObject(null);
        _connectedWire.transform.position = new Vector3(_unplugPosition.position.x, _unplugPosition.position.y,
        ↪  _unplugPosition.position.z);
        if (wire.gameObject.TryGetComponent(out Rigidbody rigidbody))
        {
            rigidbody.isKinematic = _isConnected;
            rigidbody.useGravity = !_isConnected;
        }
        _connectedWire = null;
        if (_disconnectedAudio != null && _disconnectedAudio.enabled)
        {
            _disconnectedAudio?.PlayOneShot(_disconnectedAudio?.clip);
        }
        if (wire.GetOutputType() == OutputType.Minuse)
        {
            wire.SetOutputTypeForWholeWire(OutputType.None);
        }
        wire.GetWire().RemoveConnectedElement(_parent);
        _parent?.RemoveConnectedElement(wire.GetWire());
    }

    /**
     Сеттер ElectricalElement, который использует данное место крепления провода.

    @param [in] electricalElement ElectricalElement, который использует данное место крепления провода
     */
    public void SetParent(ElectricalElement electricalElement)
    {
        _parent = electricalElement;
    }
}
```

WireOutput.cs

```
using UnityEngine;

/// Направление тока на данном выходе провода
public enum OutputType
{
    None, ///< Ток не идет по проводу
    Pluse, ///< Ток выходит из данного конца провода
    Minuse ///< Ток входит в данный конец провода
};

/**
 Компонент конца провода

Данный компонент крепиться на концы провода WireElement. WireElement управляет данным компонентом.
Данный компонент прикрепляется к WireInput.
@see WireInput; WireElement; OutputType
 */
public class WireOutput : MonoBehaviour
{
    private WireInput _connectedObject;
    private WireElement _wire;
    private OutputType _outputType;

    /**
     Установить объект, к которому подключен данный конец провода.
    @param [in] connectedObject WireInput к которому подключен данный конец провода.
     */
    public void SetConnectedObject(WireInput connectedObject)
    {
        _connectedObject = connectedObject;
    }

    /**
    Установить провод, одним из концов которого является данный объект.
    @param [in] wire WireElement, одним из концов которого является данный объект.
     */
    public void SetWire(WireElement wire)
    {
        _wire = wire;
    }

    /**
     Установить течение тока для всего провода.
     @param [in] outputType OutputType данного конца провода.
    Второй конец провода будет установлен в противоположное направление.
     */
    public void SetOutputTypeForWholeWire(OutputType outputType)
    {
        _outputType = outputType;
        _wire.UpdeteWireType(outputType, this);
    }

    /**
     Установить течение тока только для данного конца провода.
     @param [in] outputType OutputType данного конца провода.
    Второй конец провода останется неизменным.
     */
    public void SetOutputTypeToOneOutput(OutputType outputType)
    {
        _outputType = outputType;
    }

    /**
     Получить WireInput к которому прикреплен данный объект
    @return WireInput к которому прикреплен данный объект либо null,
    если данный объект не прикреплен к WireInput.
     */
    public WireInput GetConnectedObject()
    {
        return _connectedObject ? _connectedObject : null;
    }

    /**
     Получить провод, концом которого является данный объект
    @return WireElement, концом которого является данный объект.
     */
    public WireElement GetWire()
    {
        return _wire;
    }

    /**
     Получить направление тока на данном конце провода.
    @return OutputType, данного конца провода.
     */
    public OutputType GetOutputType()
    {
        return _outputType;
    }
}
```

GestureAnimation.cs

```
using Oculus.Interaction;
using UnityEngine;
```

```
using UnityEngine.Events;

/**
Класс, вызывающий события смены жестов левой и правой руки

Данный класс используется для анимации моделей рук на сервере.

Принцип работы.

Жесты, используемые для анимации моделей рук, имеют компонент GestureProperties.
В данном компоненте указано, какие пальцы загнуты у данного жеста.
Данный класс отлавливает жесты с компонентом GestureProperties, переводит его в массив bool и отправляет его скрипту
↪  HandsAnimaionUpdater.
@see GestureDetector
 */
public class GestureAnimation : GestureDetector
{
    /// Событие изменение жеста левой руки. Используется в HandsAnimaionUpdater.
    public UnityAction<bool[]> LeftGestChange;
    /// Событие изменение жеста правой руки. Используется в HandsAnimaionUpdater.
    public UnityAction<bool[]> RightGestChange;

    /**
    Метод, вызываемый при распознавании жеста системой GestureDetector.
    @param [in] gesture Распознанный жест.
     */
    protected override void GestureSelected(ActiveStateSelector gesture)
    {
        if (gesture.gameObject.TryGetComponent(out GestureProperties gestureProperties))
        {

            switch (gestureProperties.GetHandType())
            {
                case HandType.Left:
                    LeftGestChange?.Invoke(gestureProperties.GetFingersState());
                    break;
                case HandType.Right:
                    RightGestChange?.Invoke(gestureProperties.GetFingersState());
                    break;
                default:
                    Debug.LogWarning("[" + this.name + "] Жесту не назначен тип руки.");
                    break;
            }
        }
    }

    /**
    Метод, вызываемый по окончании распознавания жеста системой GestureDetector.
    @param [in] gesture Распознанный жест.
     */
    protected override void GestureUnselected(ActiveStateSelector gesture)
    {
        if (gesture.gameObject.TryGetComponent(out GestureProperties gestureProperties))
        {
            switch (gestureProperties.GetHandType())
            {
                case HandType.Left:
                    break;
                case HandType.Right:
                    break;
                default:
                    Debug.LogWarning("[" + this.name + "] Жесту не назначен тип руки.");
                    break;
            }
        }
    }
}
```

GestureDetector.cs

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

/**
 Абстрактный суперкласс для распознавания жестов

Данный класс распознает жесты указанные в его списке распознаваемых жестов.
 */
public abstract class GestureDetector : MonoBehaviour
{
    /// Список жестов, распознаваемых данной системой.
    [SerializeField] protected List<ActiveStateSelector> _gestures;

    private void Start()
    {
        foreach (ActiveStateSelector gesture in _gestures)
        {
            if (gesture != null)
            {
                gesture.WhenSelected += () => GestureSelected(gesture);
                gesture.WhenUnselected += () => GestureUnselected(gesture);
            }
            else
            {
                Debug.LogWarning("[" + gameObject.name + "] В массиве жестов есть null.");
            }
        }
```

```
    }

    /**
    Метод, вызываемый при распознавании жеста.
    @param [in] gesture Распознанный жест.
     */
    protected abstract void GestureSelected(ActiveStateSelector gesture);

    /**
    Метод, вызываемый по окончании распознавания жеста.
    @param [in] gesture Распознанный жест.
     */
    protected abstract void GestureUnselected(ActiveStateSelector gesture);
}
```

GestureProperties.cs

```
using UnityEngine;

/// Тип руки.
public enum HandType
{
    None = 0, ///< Не назначен.
    Right = 1, ///< Правая рука.
    Left = 2 ///< Левая рука.
};

/**
Компонент, хранящий в себе свойства жеста.

Данный компонент, инкапсулирующий работу с информацией о загнутых пальцах в жесте.
Данный компонент используется для анимации рук на сервере.
Каждый жест, используемый для анимации модели руки на сервере, имеет данный компонент.
@param gestureFingers GestureFingers данного жеста.
 */
public class GestureProperties : MonoBehaviour
{
    /** Тип руки данного жеста:
    - None;
    - Right;
    - Left;
     */
    public HandType Type;
    /// Загнут ли большой палец.
    public bool Thumb;
    /// Загнут ли указательный палец.
    public bool Index;
    /// Загнут ли средний палец.
    public bool Middle;
    /// Загнут ли безымянный палец.
    public bool Ring;
    /// Загнут ли мизинец.
    public bool Pinky;

    private bool[] _areFingersClosed;

    /// Конструктор по умолчанию. Ни один палец не загнут.
    public GestureProperties()
    {
        _areFingersClosed = new bool[5];
        Thumb = false;
        Index = false;
        Middle = false;
        Ring = false;
        Pinky = false;
        RefreshFingersArray();
    }

    /**
     Сеттер загнутых пальцев из другого GestureFingers.
     @param [in] fingers Копируемый GestureProperties.
     */
    public void SetFromAnotherGestureProperties(GestureProperties fingers)
    {
        this.Type = fingers.Type;
        this.Thumb = fingers.Thumb;
        this.Index = fingers.Index;
        this.Middle = fingers.Middle;
        this.Ring = fingers.Ring;
        this.Pinky = fingers.Pinky;
        RefreshFingersArray();
    }

    /**
     Сеттер загнутых пальцев из массива bool.
     @param [in] fingers Массив bool загнутых пальцев.
     */
    public void SetFromBoolArray(bool[] fingers)
    {
        if (fingers.Length >= 5)
        {
            this.Thumb = fingers[0];
            this.Index = fingers[1];
            this.Middle = fingers[2];
            this.Ring = fingers[3];
            this.Pinky = fingers[4];
            RefreshFingersArray();
        }
```

```
    }

    /**
     Геттер массива загнутых пальцев.
    @return bool[] Массив загнутых пальцев.
     */
    public bool[] GetFingersState()
    {
        RefreshFingersArray();
        return _areFingersClosed;
    }

    /**
    Геттер HandType данного жеста.
    @return HandType:
    - None;
    - Right;
    - Left;
     */
    public HandType GetHandType()
    {
        return Type;
    }

    private void RefreshFingersArray()
    {
        _areFingersClosed[0] = Thumb;
        _areFingersClosed[1] = Index;
        _areFingersClosed[2] = Middle;
        _areFingersClosed[3] = Ring;
        _areFingersClosed[4] = Pinky;
    }
}
```

PermissionRequester.cs

```
using UnityEngine;
using UnityEngine.Android;

/**
Компонент, запрашивающий разрешения шлема виртуальной реальности.

Данный компонент стоит располагать в стартовой сцене.
В свойствах данного компонента указывается какие разрешения необходимо запросить.
При включении данного компонента, он запросит все указанные разрешения и отключиться.
@param microphone Запрашивать ли разрешение пользоваться микрофоном.
 */
public class PermissionRequester : MonoBehaviour
{
    [Header("Request permissions")]
    [SerializeField] private bool _microphone;

    private void OnEnable()
    {
        if (_microphone && !Permission.HasUserAuthorizedPermission(Permission.Microphone))
        {
            Permission.RequestUserPermission(Permission.Microphone);
        }

        enabled = false;
    }
}
```

SceneChanger.cs

```
using UnityEngine;
using UnityEngine.SceneManagement;

/**
Скрипт, отвечающий за смену сцен в приложении

Данный класс инкапсулирует логику переключения между сценами у локального игрока
@param startSceneName Имя стартовой сцены. На данную сцену возвращаемся из всех других сцен.
*/
public class SceneChanger : MonoBehaviour
{
    [SerializeField] private string _startSceneName;

    /**
    Загрузить указанную сцену
    @param [in] sceneName Имя загружаемой сцены
    */
    public void Load(string sceneName)
    {
        SceneManager.LoadScene(sceneName);
    }

    /**
    Загрузить стартовую сцену.
    */
    public void LoadStartScene()
```

```
    {
        SceneManager.LoadScene(_startSceneName);
    }

    /**
    Метод завершающий работу приложения.

    @note Данный метод завершает только собранное приложение.
    В режиме отладки в консоль будет выведен Warning о завершении,
    но отладка продолжится
    */
    public void ExitFromApplication()
    {
        Debug.LogWarning("Завершение работы приложения.");
        Application.Quit();
    }
}
```

ChangeNetworkName.cs

```
using UnityEngine;
using TMPro;
using Photon.Pun;

/**
 Класс, изменяющий имя игрока в сети на значение, записанное в поле ввода.

@attention Для корректной работы данный класс требует, что бы в сцене присутствовали скрипты:
- ComponentCatcher;
- NetworkManager;

@param inputText Поле ввода, в котором записывается новое имя игрока в сети.
@param catcher ComponentCatcher данной сцены.
@see ComponentCatcher; NetworkManager
 */
public class ChangeNetworkName : MonoBehaviour
{
    [SerializeField] private TMP_InputField _inputText;
    [SerializeField] private ComponentCatcher _catcher;

    private NetworkManager _networkManager;

    private void Start()
    {
        _networkManager = _catcher.GetNetworkManager();
        if (_networkManager != null)
        {
            _networkManager.NetworConnectionEvent += OnNetworConnection;
        }
        _inputText.enabled = false;
    }

    private void OnDestroy()
    {
        if (_networkManager != null)
        {
            _networkManager.NetworConnectionEvent -= OnNetworConnection;
        }
    }

    /// Метод изменяющий имя игрока в сети на то, что записано в inputText.
    public void OnUsernameInputFieldChanged()
    {
        PhotonNetwork.NickName = _inputText.text;
    }

    private void OnNetworConnection(NetworkCode code)
    {
        switch (code)
        {
            case NetworkCode.CONNECT_TO_LOBBY_COMPLETE:
                _inputText.enabled = true;
                _inputText.text = PhotonNetwork.NickName;
                break;
            case NetworkCode.DISCONNECT_FROM_SERVER_COMPLETE:
                _inputText.enabled = false;
                break;
        }
    }
}
```

ComponentCatcher.cs

```
using System.Collections.Generic;
using UnityEngine;
using Photon.Voice.Unity;

/**
Класс, отлавливающий определенные компоненты в текущей сцене

@attention Данный класс не должен быть Singletone.
Для каждой отдельной сцены должен быть свой экземпляр данного класса, если он в ней нужен.
```

```
@attention В сцене достаточно одного экземпляра такого класса.

Данный класс находит в сцене указанные компоненты.
Это компоненты, запрашиваемые другими скриптами в ходе работы приложения.
Так же компоненты пришедшие из других сцен.

Логика работы следующая. При старте сцены ComponentCatcher отлавливает все важные компоненты.
В дальнейшем, если какому-то скрипту понадобиться один из этих компонентов он будет обращаться к ComponentCatcher.

Локальные компоненты запрашивают данный класс через свои свойства.
А компоненты спавнемые сервером будут искать ComponentCatcher в сцене.

@note ComponentCatcher безусловно отлавливает компоненты:
 - VRLoggersManager;
 - NetworkManager;
 - ControllerEvents;

@param catchVirtualKeyboardController Отлавливать ли в текущей сцене VirtualKeyboardController.
@param catchGestureAnimation Отлавливать ли в текущей сцене GestureAnimation.
@param catchNetworkVariables Отлавливать ли в текущей сцене NetworkVariables.
@param catchRecorder Отлавливать ли в текущей сцене Recorder.
@see VRLoggersManager; NetworkManager; ControllerEvents; GestureAnimation; NetworkVariables
 */
public class ComponentCatcher : MonoBehaviour
{
    /// Словарь типов классов
    private static readonly Dictionary<System.Type, string> typeToString =
       new Dictionary<System.Type, string>
       {
            { typeof(string), "string" },
            { typeof(bool), "bool" },
            { typeof(byte), "byte" },
            { typeof(char), "char" },
            { typeof(decimal), "decimal" },
            { typeof(double), "double" },
            { typeof(short), "short" },
            { typeof(int), "int" },
            { typeof(long), "long" },
            { typeof(sbyte), "sbyte" },
            { typeof(float), "float" },
            { typeof(ushort), "ushort" },
            { typeof(uint), "uint" },
            { typeof(ulong), "ulong" },
            { typeof(void), "void" },
            { typeof(VRLoggersManager), "VRLoggersManager" },
            { typeof(NetworkManager), "NetworkManager" },
            { typeof(ControllerEvents), "ControllerEvents" },
            { typeof(GestureAnimation), "GestureAnimation" },
            { typeof(NetworkVariables), "NetworkVariables" },
            { typeof(VirtualKeyboardController), "VirtualKeyboardController" },
            { typeof(Recorder), "Recorder" },

            { typeof(object), "object" }
       };

    [Header("Catchable Components")]
    [SerializeField] private bool _catchVirtualKeyboardController;
    [SerializeField] private bool _catchGestureAnimation;
    [SerializeField] private bool _catchNetworkVariables;
    [SerializeField] private bool _catchRecorder;

    private NetworkManager _networkManager;
    private VRLoggersManager _vrLogger;
    private ControllerEvents _controllerEvents;
    private GestureAnimation _gestureAnimator;
    private NetworkVariables _networkVariables;
    private VirtualKeyboardController _virtualKeyboardController;
    private Recorder _recorder;


    private void Start()
    {
        RefreshComponents();
    }

    private void RefreshComponents()
    {
        if (_vrLogger == null)
        {
            _vrLogger = FindObjectOfType<VRLoggersManager>();
            if (_vrLogger == null)
            {
                Debug.LogWarning("[" + this.name + "] Не удалось поймать VRLoggersManager");
            }
        }
        if (_networkManager == null)
        {
            _networkManager = FindObjectOfType<NetworkManager>();
            CheckComponentState(_networkManager);
        }
        if (_controllerEvents == null)
        {
            _controllerEvents = FindObjectOfType<ControllerEvents>();
            CheckComponentState(_controllerEvents);
        }

        if (_catchVirtualKeyboardController && _virtualKeyboardController == null)
        {
            _virtualKeyboardController = FindObjectOfType<VirtualKeyboardController>();
            CheckComponentState(_virtualKeyboardController);
        }
```

```
    if (_catchGestureAnimation && _gestureAnimator == null)
    {
        _gestureAnimator = FindObjectOfType<GestureAnimation>();
        CheckComponentState(_gestureAnimator);
    }

    if (_catchNetworkVariables && _networkVariables == null)
    {
        _networkVariables = FindObjectOfType<NetworkVariables>();
        CheckComponentState(_networkVariables);
    }

    if (_catchRecorder && _recorder == null)
    {
        _recorder = FindObjectOfType<Recorder>();
        CheckComponentState(_recorder);
    }

}

private void CheckComponentState<T>(T component)
{
    if (component == null)
    {
        _vrLogger?.Log("[" + this.name + "] Не удалось поймать " + typeToString[typeof(T)]);
        Debug.LogWarning("[" + this.name + "] Не удалось поймать " + typeToString[typeof(T)]);
    }
}

private void TryToGetComponent<T>(ref T fild)
{
    if (fild == null)
    {
        RefreshComponents();
        if (fild == null)
        {
            _vrLogger?.Log("[" + this.name + "] Catcher не содержит " + typeToString[typeof(T)]);
            Debug.LogWarning("[" + this.name + "] Catcher не содержит " + typeToString[typeof(T)]);
        }
    }
}

/**
 Геттер NetworkManager
@return NetworkManager, если он был найден в сцене.
 */
public NetworkManager GetNetworkManager()
{
    TryToGetComponent(ref _networkManager);

    return _networkManager;
}

/**
 Геттер ControllerEvents
@return ControllerEvents, если он был найден в сцене.
 */
public ControllerEvents GetControllerEvents()
{
    TryToGetComponent(ref _controllerEvents);

    return _controllerEvents;
}

/**
 Геттер VirtualKeyboardController
@return VirtualKeyboardController, если он был найден в сцене.
 */
public VirtualKeyboardController GetVirtualKeyboard()
{
    TryToGetComponent(ref _virtualKeyboardController);

    return _virtualKeyboardController;
}

/**
 Геттер GestureAnimation
@return GestureAnimation, если он был найден в сцене.
 */
public GestureAnimation GetGestureAnimator()
{
    TryToGetComponent(ref _gestureAnimator);

    return _gestureAnimator;
}

/**
 Геттер NetworkVariables
@return NetworkVariables, если он был найден в сцене.
 */
public NetworkVariables GetNetworkVariables()
{
    TryToGetComponent(ref _networkVariables);

    return _networkVariables;
}

/**
 Геттер Recorder
@return Recorder, если он был найден в сцене.
 */
public Recorder GetRecorder()
{
```

```
        TryToGetComponent(ref _networkVariables);

        return _recorder;
    }


    /**
     Геттер VRLoggersManager
    @return VRLoggersManager, если он был найден в сцене.
     */
    public VRLoggersManager GetVRLoggersManager()
    {
        TryToGetComponent(ref _vrLogger);

        return _vrLogger;
    }

}
```

NetworkManager.cs

```
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.Events;
using Photon.Pun;
using Photon.Realtime;

/**
Класс, хранящий настройки комнаты
 */
[System.Serializable]
public class RoomSettings
{
    /**
     Ключ для поля CustomProperties в RoomInfo.
     Поле с данным ключем хранит ID сцены, из которой будет создана комната.
     */
    public const string SCENE_ID = "SceneID";
    /**
     Ключ для поля CustomProperties в RoomInfo.
     Поле с данным ключем хранит индекс сцены в списке defaultRooms
     */
    public const string ROOM_INDEX = "RoomIndex";

    /// Название комнаты.
    public string name;
    /// ID сцены, из которой будет создана комната.
    public int sceneID;
    /// Количество игроков в комнате.
    public byte playersInRoom;
    /// Будет ли комната видима.
    public bool isRoomVisible;
}

/// Коды состояний сервера
public enum NetworkCode
{
    NO_CODE = 619, ///< Отсутствие кода (необходимо, если действие должно произойти не зависимо от состояния сервера).
    CONNECT_TO_SERVER_IN_PROGRESS = 100, ///< В процессе подключения к серверу.
    CONNECT_TO_SERVER_COMPLETE = 200, ///< Подключение к серверу завершено.
    CONNECT_TO_LOBBY_IN_PROGRESS = 101, ///< В процессе подключения к лобби.
    CONNECT_TO_LOBBY_COMPLETE = 201, ///< Подключение к лобби завершено.
    CONNECT_TO_ROOM_IN_PROGRESS = 102, ///< В процессе подключения к комнате.
    CONNECT_TO_ROOM_COMPLETE = 202, ///< Подключение к комнате завершено.
    CONNECT_TO_ROOM_FAILD = 402, ///< Не удалось подключиться к комнате.
    PLAYER_ENTER_THE_ROOM = 203, ///< К комнате подключился новый игрок.
    DISCONNECT_FROM_SERVER_IN_PROGRESS = 104, ///< В процессе отключения от сервера.
    DISCONNECT_FROM_SERVER_COMPLETE = 204, ///< Отключение от сервера завершено.
    ROOM_LIST_UPDATE = 105 ///< Список комнат обновился.
}

/**
Класс, отвечающий за взаимодействие с сервером Photon

Данный класс отвечает за:
 - Подключение к серверу;
 - Отключение от сервера;
 - Создание и подключение к лобби;
 - Создание и подключение к комнате;
 - Отключение от комнаты;
 - Отключение от лобби;
 - Отключение от сервера;
 - Обновление списка созданных комнат;
 - Спавн игрока на сервере.

@param vrLogger VRLoggersManager для вывода логов внутри игры.
@param sceneChanger SceneChanger для перехода между сценами.
@param defaultRooms Список RoomSettings комнат, к которым будет производиться подключение.
@param autoStartTestRoom bool Параметр для отладки.
Если true, то автоматически подключает в первую комнату при запуске приложения.
@see VRLoggersManager; SceneChanger; RoomSettings; NetworkCode
 */
public class NetworkManager : MonoBehaviourPunCallbacks
{
    /// Событие сервера.
    public event UnityAction<NetworkCode> NetworConnectionEvent;
    /// Событие обновление списка комнат.
    public event UnityAction<List<RoomInfo>> RoomListUpdate;
```

```
[SerializeField] private VRLoggersManager _vrLogger;
[SerializeField] private SceneChanger _sceneChanger;
[SerializeField] private List<RoomSettings> _defaultRooms;
[SerializeField] private bool _autoStartTestRoom;

private string _playersPrefabName;

private bool _quitFromApplication;
private bool _isConnectedToServer;
private GameObject _spawnedPlayerPrefab;

private void Start()
{
    _quitFromApplication = false;
    _isConnectedToServer = false;
    _vrLogger.SetNetworkManager(this);
    if (_autoStartTestRoom)
    {
        ConnectToServer();
    }
}

/// Осуществить подключение к серверу.
public void ConnectToServer()
{
    NetworConnectionEvent?.Invoke(NetworkCode.CONNECT_TO_SERVER_IN_PROGRESS);
    _vrLogger.Log("[" + this.name + "] Conecting to server...");
    PhotonNetwork.ConnectUsingSettings();
}

/// Метод, выполняемый при подключении к серверу.
public override void OnConnectedToMaster()
{
    base.OnConnectedToMaster();
    SetRandomName();
    NetworConnectionEvent?.Invoke(NetworkCode.CONNECT_TO_SERVER_COMPLETE);
    _isConnectedToServer = true;
    _vrLogger.Log("[" + this.name + "] Connected to master server.");
    NetworConnectionEvent?.Invoke(NetworkCode.CONNECT_TO_LOBBY_IN_PROGRESS);
    PhotonNetwork.AutomaticallySyncScene = true;
    PhotonNetwork.JoinLobby();
}

/**
Создание/подключение к комнате.

Если комнаты не существует - она будет создана.
Иначе произойдет подключение к существующей комнате.
@param [in] roomIndex Индекс комнаты в defaultRooms, к которой мы хотим подключиться.
@param [in] roomID Индекс комнаты. При создании нескольких комнат из одного шаблона мы хотим различать их по ID
 */
public void InitRoom(int roomIndex, int roomID)
{
    if (roomIndex >= 0 && roomIndex < _defaultRooms.Count)
    {
        NetworConnectionEvent?.Invoke(NetworkCode.CONNECT_TO_ROOM_IN_PROGRESS);
        RoomSettings defaultRoom = _defaultRooms[roomIndex];

        PhotonNetwork.LoadLevel(defaultRoom.sceneID);

        RoomOptions roomOptions = new RoomOptions();
        roomOptions.MaxPlayers = defaultRoom.playersInRoom;
        roomOptions.IsVisible = defaultRoom.isRoomVisible;
        roomOptions.IsOpen = true;
        roomOptions.CustomRoomProperties = new ExitGames.Client.Photon.Hashtable();
        roomOptions.CustomRoomProperties.Add(RoomSettings.ROOM_INDEX, roomIndex);
        roomOptions.CustomRoomProperties.Add(RoomSettings.SCENE_ID, defaultRoom.sceneID);
        PhotonNetwork.JoinOrCreateRoom(defaultRoom.name + " " + roomID, roomOptions, TypedLobby.Default);
    }
    else
    {
        _vrLogger.Log("[" + this.name + "] Room index " + roomIndex + " is not correct.");
    }
}

/**
Подключение к комнате.

@param [in] roomInfo Информация о комнате, к которой производится подключение.
@note Внутри roomInfo.CustomProperties должен храниться ключ RoomSettings.SCENE_ID, хранящий значением id сцены, к которой
↪  необходимо подключиться.
Иначе подключение к комнате не произойдет.
 */
public void JoinRoom(RoomInfo roomInfo)
{
    if (roomInfo.CustomProperties.ContainsKey(RoomSettings.SCENE_ID))
    {
        PhotonNetwork.LoadLevel((int)roomInfo.CustomProperties[RoomSettings.SCENE_ID]);
        PhotonNetwork.JoinRoom(roomInfo.Name);
    }
    else
    {
        _vrLogger.Log("[" + this.name + "] Room Info has not contain custom property: \"" + RoomSettings.SCENE_ID + "\".");
    }
}

/// Покинуть текущую комнату
public void LeaveRoom()
{
    PhotonNetwork.LeaveRoom();
}
```

```
/**
Отключиться от сервера.
@param [in] quitFromApplication Если true, то после отключения от сервера произойдет выход из приложения.
 */
public void DisconnectedFromServer(bool quitFromApplication = false)
{
    _vrLogger.Log("[" + this.name + "] Disconnecting from server.");
    NetworConnectionEvent?.Invoke(NetworkCode.DISCONNECT_FROM_SERVER_IN_PROGRESS);
    _quitFromApplication = quitFromApplication;
    if (quitFromApplication && !_isConnectedToServer)
    {
        _sceneChanger.ExitFromApplication();
    }
    PhotonNetwork.Disconnect();
}

/// Метод, выполняемый при подключении к комнате.
public override void OnJoinedRoom()
{
    base.OnJoinedRoom();
    NetworConnectionEvent?.Invoke(NetworkCode.CONNECT_TO_ROOM_COMPLETE);
    _vrLogger.Log("[" + this.name + "] You are join to the room.");
    SpawnPlayerPrefab();
    NetworkVariables.SendPropertyToServer(PhotonServerActions.UPDATE_STATUS, "Update");
}

/// Метод, выполняемый при отключении от комнаты.
public override void OnLeftRoom()
{
    string destroyedPlayerName = "#destroyed#";
    if (_spawnedPlayerPrefab != null)
    {
        destroyedPlayerName = _spawnedPlayerPrefab.name;
    }
    base.OnLeftRoom();
    PhotonNetwork.Destroy(_spawnedPlayerPrefab);
    _vrLogger.Log("[" + this.name + "] Player " + destroyedPlayerName + " is destroy");
    _sceneChanger.LoadStartScene();
}

/**
Метод, выполняемый при отключении от сервера.
@param [in] cause Причина отключения от сервера.
 */
public override void OnDisconnected(DisconnectCause cause)
{
    base.OnDisconnected(cause);
    _isConnectedToServer = false;
    NetworConnectionEvent?.Invoke(NetworkCode.DISCONNECT_FROM_SERVER_COMPLETE);
    _vrLogger.Log("[" + this.name + "] You was disconnected from server.");
    if (_quitFromApplication)
    {
        _sceneChanger.ExitFromApplication();
    }
}

/// Метод, выполняемый при подключении к лобби.
public override void OnJoinedLobby()
{
    base.OnJoinedLobby();

    NetworConnectionEvent?.Invoke(NetworkCode.CONNECT_TO_LOBBY_COMPLETE);
    _vrLogger.Log("[" + this.name + "] Some user is join to the lobby.");
    if (_autoStartTestRoom)
    {
        _autoStartTestRoom = false;
        if (_defaultRooms.Count > 0)
        {
            InitRoom(0, 0);
        }
        else
        {
            _vrLogger.Log("[" + this.name + "] No rooms to connect.");
        }
    }
}

/**
Метод выполняемый, когда другой игрок подключился к комнате.
@param [in] newPlayer Данные подключившегося игрока.
 */
public override void OnPlayerEnteredRoom(Player newPlayer)
{
    base.OnPlayerEnteredRoom(newPlayer);
    _vrLogger.Log("[" + this.name + "] " +
       newPlayer.NickName == null || newPlayer.NickName == ""
        ? "Some unknown user"
        : newPlayer.NickName
        + "is join to the room.");
}

/**
Метод выполняемый при обновлении списка комнат. При добавлении/удалении комнаты.
@param [in] roomList Список комнат RoomInfo, существующих на данный момент.
 */
public override void OnRoomListUpdate(List<RoomInfo> roomList)
{
    NetworConnectionEvent?.Invoke(NetworkCode.ROOM_LIST_UPDATE);
    foreach (RoomInfo room in roomList)
    {
        for (int i = 0; i < _defaultRooms.Count; i++)
        {
            if (_defaultRooms[i].name == room.Name.TrimEnd(new char[] { ' ', '0', '1', '2', '3', '4', '5', '6', '7', '8', '9'
            ↪  }))
```

```
                {
                    room.CustomProperties.Add(RoomSettings.ROOM_INDEX, i);
                    room.CustomProperties.Add(RoomSettings.SCENE_ID, _defaultRooms[i].sceneID);
                    break;
                }
            }
        }
        RoomListUpdate?.Invoke(roomList);
    }

    /**
     Сеттер названия prefab-а игрока.

    @param [in] playersPrefabName string название prefab-а игрока лежащего в Assets/Resources/Avatars.
     */
    public void SetPlayersPrefabName(string playersPrefabName)
    {
        _playersPrefabName = playersPrefabName;
    }

    private void SetRandomName()
    {
        if (PhotonNetwork.NickName.Length == 0)
        {
            char[] lettersArray = { 'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S',
            ↪  'T', 'U', 'V', 'W', 'X', 'Y', 'Z' };
            string randomName = ""
                + lettersArray[Random.Range(0, lettersArray.Length - 1)]
                + ""
                + lettersArray[Random.Range(0, lettersArray.Length - 1)]
                + "-"
                + Random.Range(0, 99)
                + " "
                + lettersArray[Random.Range(0, lettersArray.Length - 1)]
                + lettersArray[Random.Range(0, lettersArray.Length - 1)]
                + lettersArray[Random.Range(0, lettersArray.Length - 1)]
                + lettersArray[Random.Range(0, lettersArray.Length - 1)]
                + "-"
                + lettersArray[Random.Range(0, lettersArray.Length - 1)];

            PhotonNetwork.NickName = randomName;
        }
    }

    private void SpawnPlayerPrefab()
    {
        Vector3 playerPosition = new Vector3(transform.position.x + Random.Range(-10, 10), transform.position.y +
        ↪  Random.Range(-10, 10));
        _spawnedPlayerPrefab = PhotonNetwork.Instantiate(_playersPrefabName, playerPosition, transform.rotation);
        _vrLogger.Log("[" + this.name + "] Player " + _spawnedPlayerPrefab.GetComponent<PhotonView>().Owner.NickName + " spawned
        ↪  at position(" + playerPosition.x + "," + playerPosition.y + "," + playerPosition.z + ")");
    }
}
```

NetworkManagerProvider.cs

```
using UnityEngine;

/**
Класс, обеспечивающий взаимодействие UI с NetworkManager.

Взаимодействуя с ComponentCatcher, данный класс предоставляет элементам UI взаимодействовать с NetworkManager.

@attention Для корректной работы данный класс требует, что бы в сцене присутствовали скрипты:
- NetworkManager;

@param catcher ComponentCatcher, который используется для получения доступа к NetworkManager.
@see ComponentCatcher; NetworkManager
 */
public class NetworkManagerProvider : MonoBehaviour
{
    [SerializeField] private ComponentCatcher _catcher;

    private NetworkManager _networkManager;

    private void Start()
    {
        if (_catcher != null)
        {
            _networkManager = _catcher.GetNetworkManager();
        }
    }

    /// Подключиться к серверу.
    public void ConnectToServer()
    {
        _networkManager?.ConnectToServer();
    }

    /// Покинуть комнату.
    public void LeaveRoom()
    {
        _networkManager?.LeaveRoom();
    }

    /**
    Создание/подключение к комнате нулевой комнате.
```

```
    Если комнаты не существует - она будет создана.
    Иначе произойдет подключение к существующей комнате.
    @param [in] roomIndex Индекс комнаты в defaultRooms, к которой мы хотим подключиться.
     */
    public void InitDefaultRoom(int roomIndex)
    {
        _networkManager?.InitRoom(roomIndex, 0);
    }

    /// Отключиться от сервера.
    public void DisconnectedFromServer()
    {
        _networkManager?.DisconnectedFromServer();
    }

    /**
    Выйти из приложения.
    @note Данный метод завершает только собранное приложение.
    В режиме отладки в консоль будет выведен Warning о завершении,
    но отладка продолжится
     */
    public void ExitFromApplication()
    {
        _networkManager?.DisconnectedFromServer(true);
    }
}
```

NetworkPlayer.cs

```
using UnityEngine;
using Photon.Pun;
using Photon.Realtime;

/**
 Класс, отвечающий за синхронизацию локального игрока и его отображения на сервере

Данный класс используется в prefab-е игрока, находящегося по пути Assets/Resources/Avatars.


Данный класс отвечает за:
- синхронизацию положения головы (шлема oculus);
- синхронизацию положения рук (контроллеров oculus/отслеживаемых человеческих рук);
- вид контроллеров (контроллеры oculus/отслеживаемые человеческие руки);

@attention Для корректной работы данный класс требует, что бы в сцене присутствовали скрипты:
- ComponentCatcher;
- NetworkVariables;
- HandView;
- OVRCameraRig;
- ControllerEvents;

@param head Объект отображения головы на сервере (голова в prefab-е игрока).
@param leftHand Объект отображения левой руки на сервере (левая рука в prefab-е игрока).
@param rightHand Объект отображения правой руки на сервере (правая рука в prefab-е игрока).
@param controllertTypeController Массив ControllerTypeController, которыми управляет данный класс.
@param dublicateMainPlayer Если true, то пользователь будет видеть как его контроллеры отображаются на сервере.
@see ControllerTypeController; HandView; ControllerEvents; NetworkVariables; ComponentCatcher
 */
[RequireComponent(typeof(PhotonView))]
public class NetworkPlayer : MonoBehaviour
{
    [SerializeField] private Transform _head;
    [SerializeField] private Transform _leftHand;
    [SerializeField] private Transform _rightHand;

    [SerializeField] private ControllerTypeController[] _controllertTypeController;

    [SerializeField] private bool _dublicateMainPlayer;

    private bool _isAttachToController;

    /// Объект синхронизации Photon
    private PhotonView _photonView;
    private Transform _headRig;
    private Transform _leftHandRig;
    private Transform _rightHandRig;


    private HandView[] _handViews;
    private ControllerEvents _controllerChangeTypeEvent;

    private NetworkVariables _networkVariables;

    private void Awake()
    {
        _photonView = GetComponent<PhotonView>();
    }

    private void Start()
    {
        CreatePlayer();
        ComponentCatcher catcher = FindObjectOfType<ComponentCatcher>();
        _networkVariables = catcher?.GetNetworkVariables();
        if (_networkVariables)
        {
            _networkVariables.OnNetworkVariablesUpdate += OnPlayerPropertiesUpdate;
        }
    }
```

```
private void Update()
{
    if (_photonView.IsMine)
    {
        MapPosition(_head, _headRig);
        MapPosition(_leftHand, _leftHandRig);
        MapPosition(_rightHand, _rightHandRig);
    }
}

private void OnDestroy()
{
    if (_controllerChangeTypeEvent)
    {
        _controllerChangeTypeEvent.ControllerTypeChange -= OnControllerChange;
    }

    if (_networkVariables)
    {
        _networkVariables.OnNetworkVariablesUpdate -= OnPlayerPropertiesUpdate;
    }
}

private void CreatePlayer()
{
    OVRCameraRig ovrCameraRig = FindObjectOfType<OVRCameraRig>();
    _headRig = ovrCameraRig.transform.Find("TrackingSpace/CenterEyeAnchor");

    _handViews = FindObjectsOfType<HandView>();
    foreach (HandView handView in _handViews)
    {
        if (handView.GetHandType() == HandType.Left)
        {
            _leftHandRig = handView.transform;
        }
        else
        {
            _rightHandRig = handView.transform;
        }
        if (_controllerChangeTypeEvent == null)
        {
            _controllerChangeTypeEvent = handView.GetControllerSwitcher();
            if (_controllerChangeTypeEvent != null)
            {
                _controllerChangeTypeEvent.ControllerTypeChange += OnControllerChange;
                OnControllerChange(_controllerChangeTypeEvent.IsAttachToControllerNow());
            }
        }

    }

    if (_photonView.IsMine)
    {
        if (!_dublicateMainPlayer)
        {
            foreach (Renderer renderer in GetComponentsInChildren<Renderer>())
            {
                renderer.enabled = false;
            }
        }
    }
}

/**
 Метод для установки положения и поворота одного объекта, согласно соответствующим характеристикам другого объекта.
@param [in] target Объект, которому устанавливаются положение и поворот.
@param [in] rigTransform Объект, из которого берутся положение и поворот для первого объекта.
 */
protected void MapPosition(Transform target, Transform rigTransform)
{
    target.position = rigTransform.position;
    target.rotation = rigTransform.rotation;
}

private void OnControllerChange(bool isAttachToController)
{
    if (_photonView.IsMine)
    {
        _isAttachToController = isAttachToController;
        if (isAttachToController)
        {
            ChangeControllerView(ControllerType.OculusController);
        }
        else
        {
            ChangeControllerView(ControllerType.HandsPrefabs);
        }
    }
}

private void ChangeControllerView(ControllerType type)
{
    // Обновляем локального player-а
    ChangeLocalControllerView(type);
    // Обновляем player-а на сервере через свойства photon
    NetworkVariables.SendPropertyToServer(PhotonServerActions.CONTROLLER_TYPE, type);
}

private void ChangeLocalControllerView(ControllerType type)
{
    foreach (ControllerTypeController myControllerrPrefab in _controllertTypeController)
    {
```

```
            myControllerrPrefab.SwitchControllerView(type);
        }
    }

    private void OnPlayerPropertiesUpdate(Player targetPlayer, ExitGames.Client.Photon.Hashtable changedProps)
    {
        if (changedProps.ContainsKey(PhotonServerActions.CONTROLLER_TYPE))
        {
            if (!_photonView.IsMine)
            {
                if (targetPlayer == _photonView.Owner)
                {
                    ChangeLocalControllerView((ControllerType)changedProps[PhotonServerActions.CONTROLLER_TYPE]);
                }
            }
        }

        if (changedProps.ContainsKey(PhotonServerActions.UPDATE_STATUS))
        {
            OnControllerChange(_isAttachToController);
        }
    }
}
```

NetworkVariables.cs

```
using UnityEngine.Events;
using Photon.Pun;
using Photon.Realtime;
using Hashtable = ExitGames.Client.Photon.Hashtable;

/// Класс, хранящий константы действий, совершаемых на сервере.
public class PhotonServerActions
{
    /// Изменить тип контроллера.
    public const string CONTROLLER_TYPE = "ControllerType";
    /// Изменить анимацию отображения руки в соответствии с жестом.
    public const string GESTURE_FINGERS = "GestureFingers";
    /// Обновить состояние всех переменных.
    public const string UPDATE_STATUS = "UpdateStatus";
    /// Изменить громкость микрофона.
    public const string MICROPHONE_VOLUME = "MicrophoneVolume";
    /// Изменить kinematic объекта
    public const string CHANGE_KINEMATIC = "ChangeKinematic";
    /// Изменить видимость объекта InterfaceHider
    public const string CHANGE_HIDER = "ChangeHider";
}

/**
 Класс работы с переменными сервера

Данный класс предоставляет возможность отправлять переменные на сервер
и обрабатывать переменные пришедшие на сервер.
 */
public class NetworkVariables : MonoBehaviourPunCallbacks
{
    /// Событие пришествия на сервер новой переменной.
    public UnityAction<Player, Hashtable> OnNetworkVariablesUpdate;

    /**
    Метод отправки на сервер переменной.
    @param [in] propertyName string имя переменной из класса PhotonServerActions.
    @param [in] property Переменная, отправляемая на сервер.
     */
    public static void SendPropertyToServer<T>(string propertyName, T property)
    {
        Hashtable hash = new Hashtable();
        hash.Add(propertyName, property);
        PhotonNetwork.LocalPlayer.SetCustomProperties(hash);
    }

    /**
    Метод, вызываемый при пришествии новых переменных на сервер.
    @param [in] targetPlayer Игрок, приславший переменные.
    @param [in] changedProps Хеш таблица Photon, содержащая переменные, присланные на сервер.
     */
    public override void OnPlayerPropertiesUpdate(Player targetPlayer, Hashtable changedProps)
    {
        OnNetworkVariablesUpdate?.Invoke(targetPlayer, changedProps);
    }
}
```

OwnershipTransfer.cs

```
using UnityEngine;
using Oculus.Interaction;
using Photon.Pun;
using Photon.Realtime;

/**
Скрипт, передающий права на объект
```

```
/*
В Photon у интерактивных объектов есть владелец. И Photon синхронизирует с сервером только данные владельца объекта.
Владельцем становится первый схвативший объект.

Но нам необходимо, что бы все пользователи могли взаимодействовать с объектами.

Для этого необходимо передавать владение объектом тому пользователю, который взял объект.
Этим и занимается данный скрипт.

Данный скрипт вешается на объект. Если объект кто-то берет, то этот кто-то становится его владельцем,
независимо от того есть ли у объекта сейчас другой владелец.

@attention Для корректной работы данный класс требует, что бы в сцене присутствовали скрипты:
- ComponentCatcher;
- VRLoggersManager;
- NetworkVariables;
@see VRLoggersManager; ComponentCatcher
 */
[RequireComponent(typeof(Grabbable))]
public class OwnershipTransfer : MonoBehaviourPun, IPunOwnershipCallbacks
{
    private VRLoggersManager _vrLogger;
    private bool _isGrab;
    private Grabbable _grabbable;

    private void Awake()
    {
        PhotonNetwork.AddCallbackTarget(this);
        _grabbable = GetComponent<Grabbable>();
        _grabbable.WhenPointerEventRaised += OnObjectGrabChange;
        _isGrab = false;
    }

    private void Start()
    {
        ComponentCatcher catcher = FindObjectOfType<ComponentCatcher>();
        if (catcher == null)
        {
            Debug.LogWarning("[" + this.name + "] Can not find ComponentCatcher in scene");
        }
        else
        {
            _vrLogger = catcher.GetVRLoggersManager();
        }
    }

    private void OnDestroy()
    {
        PhotonNetwork.RemoveCallbackTarget(this);
        _grabbable.WhenPointerEventRaised -= OnObjectGrabChange;
    }

    private void OnObjectGrabChange(PointerEvent grabEvent)
    {
        switch (grabEvent.Type)
        {
            case PointerEventType.Select:
                if (!_isGrab)
                {
                    _isGrab = true;
                    base.photonView.RequestOwnership();
                }
                break;
            case PointerEventType.Unselect:
                if (_isGrab)
                {
                    _isGrab = false;
                }
                break;
        }
    }

    /**
     Метод, вызываемый при запросе на смену владельца.
    @param [in] targetView PhotonView данного объекта.
    @param [in] requestingPlayer Player, который хочет стать владельцем.
     */
    public void OnOwnershipRequest(PhotonView targetView, Player requestingPlayer)
    {
        if (targetView != base.photonView)
        {
            return;
        }

        base.photonView.TransferOwnership(requestingPlayer);
    }

    /**
     Метод, вызываемый по окончании передачи прав на объект.
    @param [in] targetView PhotonView данного объекта.
    @param [in] previousOwner Player предыдущего владельца.
     */
    public void OnOwnershipTransfered(PhotonView targetView, Player previousOwner)
    {
        _vrLogger?.Log(this.name + " is change owner to " + targetView.name);
        Debug.Log(this.name + " is change owner to " + targetView.name);
        if (targetView != base.photonView)
        {
            return;
        }
    }

    /**
     Метод, вызываемый при ошибке передачи прав на объект.
```

```
    @param [in] targetView PhotonView данного объекта.
    @param [in] senderOfFailedRequest Player, который пытался стать владельцем.
     */
    public void OnOwnershipTransferFailed(PhotonView targetView, Player senderOfFailedRequest)
    {
        _vrLogger?.Log("Ownership Transfer Failed");
        Debug.LogWarning("Ownership Transfer Failed");
    }
}
```

RigidBodySycn.cs

```
using UnityEngine;
using Photon.Pun;
using Photon.Realtime;

/**
 Скрипт для синхронизации RigidBody на сервере

Данный скрипт синхронизирует с сервером параметры RigidBody объекта:
- isKinematic;

@attention Для корректной работы данный класс требует, что бы в сцене присутствовали скрипты:
- ComponentCatcher;
- NetworkVariables;
@param photonView PhotonView данного объекта
@see NetworkVariables; ComponentCatcher
 */
[RequireComponent(typeof(Rigidbody))]
public class RigidBodySycn : MonoBehaviour
{
    [SerializeField] private PhotonView _photonView;
    private Rigidbody _rigidbody;
    private NetworkVariables _networkVariables;
    private bool _isKinematic;
    private bool _isInit;

    private void Awake()
    {
        _rigidbody = GetComponent<Rigidbody>();
        _isKinematic = _rigidbody.isKinematic;
        _isInit = false;
    }

    private void Start()
    {
        ComponentCatcher catcher = FindObjectOfType<ComponentCatcher>();
        if (catcher == null)
        {
            Debug.LogWarning("[" + this.name + "] Can not find ComponentCatcher in scene");
        }
        else
        {
            _networkVariables = catcher.GetNetworkVariables();
            if (_networkVariables)
            {
                _networkVariables.OnNetworkVariablesUpdate += OnPlayerPropertiesUpdate;
            }
        }
    }

    private void OnDestroy()
    {
        if (_networkVariables)
        {
            _networkVariables.OnNetworkVariablesUpdate -= OnPlayerPropertiesUpdate;
        }
    }

    private void Update()
    {
        if (_rigidbody.isKinematic != _isKinematic)
        {
            _isKinematic = _rigidbody.isKinematic;
            NetworkVariables.SendPropertyToServer(PhotonServerActions.CHANGE_KINEMATIC, _isKinematic);
            _isInit = true;
        }
    }

    private void ChangeKinematic(bool isKinematic)
    {
        _rigidbody.isKinematic = isKinematic;
    }

    private void OnPlayerPropertiesUpdate(Player targetPlayer, ExitGames.Client.Photon.Hashtable changedProps)
    {
        if (changedProps.ContainsKey(PhotonServerActions.CHANGE_KINEMATIC))
        {
            if (!_photonView.IsMine)
            {
                if (targetPlayer == _photonView.Owner)
                {
                    ChangeKinematic((bool)changedProps[PhotonServerActions.CHANGE_KINEMATIC]);
                    _isInit = true;
                }
            }
        }
```

```
        if (changedProps.ContainsKey(PhotonServerActions.UPDATE_STATUS))
        {
            if (_isInit)
            {
                NetworkVariables.SendPropertyToServer(PhotonServerActions.CHANGE_KINEMATIC, _isKinematic);
            }
        }
    }
}
```

AvatarList.cs

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

/**
Класс, контролирующий все элементы списка доступных аватаров.

Данный класс ищет в папке ресурсов аватары AvatarInfo и из них составляет список аватаров и отображает его в указанном UI.

Так же данный класс связывается с NetworkManager для указания ему выбранного в текущий момент аватара.


@attention Для корректной работы данный класс требует, что бы в сцене присутствовали скрипты:
- ComponentCatcher;
- NetworkManager;

@param catcher ComponentCatcher данной сцены.
@param avatarsListContent UI поле, в которое будет отображен список аватаров.
@param avatarListItemPrefab Prefab отображаемого элемента списка. Данный prefab должен содержать компонент AvatarListItem.
@see ComponentCatcher; NetworkManager; AvatarListItem; AvatarInfo
 */
public class AvatarList : MonoBehaviour
{
    [SerializeField] private ComponentCatcher _catcher;
    [SerializeField] private Transform _avatarsListContent;
    [SerializeField] private GameObject _avatarListItemPrefab;
    [Tooltip("All avatars should be stored in a subfolder of the Resources folder.")]
    [SerializeField] private string _avatarsFolder;


    private List<AvatarListItem> _avatarListItems;
    private NetworkManager _networkManager;

    private void Start()
    {
        _networkManager = _catcher.GetNetworkManager();
        if (_networkManager != null)
        {
            _networkManager.NetworConnectionEvent += OnNetworConnection;
        }
        UpdateAvatarList();
    }

    private void OnDestroy()
    {
        if (_networkManager != null)
        {
            _networkManager.NetworConnectionEvent -= OnNetworConnection;
        }
    }

    /**
     Метод изменяющий выбранный в данный момент элемент списка.

    Данный метод делает не выбранными все элементы списка, кроме переданного ему во входном параметре.
    Так же в данном методе устанавливается новое имя спавнемого prefab-а игрока(аватара).
    @param [in] newSelectedElement AvatarListItem выбранный в данный момент.
     */
    public void ChangeCurrentElement(AvatarListItem newSelectedElement)
    {
        foreach (AvatarListItem item in _avatarListItems)
        {
            if (item == newSelectedElement)
            {
                item.SetSelected(true);
                _networkManager?.SetPlayersPrefabName(item.GetAvatarName());
            }
            else
            {
                item.SetSelected(false);
            }
        }
    }

    private void UpdateAvatarList()
    {
        foreach (Transform trans in _avatarsListContent)
        {
            Destroy(trans.gameObject);
        }

        if (_avatarListItems == null)
        {
            _avatarListItems = new List<AvatarListItem>();
        }
        _avatarListItems.Clear();
```

```
        AvatarInfo[] avatars = Resources.LoadAll<AvatarInfo>(_avatarsFolder);
        foreach (AvatarInfo avatar in avatars)
        {
            if (avatar.IsAvatarActive())
            {
                AvatarListItem item = Instantiate(_avatarListItemPrefab, _avatarsListContent).GetComponent<AvatarListItem>();
                item.SetControllerList(this);
                item.SetSelected(false);
                item.SetAvatarImage(avatar.GetAvatarImage());
                item.SetAvatarName(avatar.GetAvatarName());
                _avatarListItems.Add(item);
            }
        }

        if (_avatarListItems.Count > 0)
        {
            _avatarListItems[0].SetSelected(true);
        }
    }

    private void OnNetworConnection(NetworkCode code)
    {
        switch (code)
        {
            case NetworkCode.CONNECT_TO_LOBBY_COMPLETE:
                foreach (AvatarListItem item in _avatarListItems)
                {
                    if (item.IsSelected())
                    {
                        _networkManager?.SetPlayersPrefabName(item.GetAvatarName());
                        break;
                    }
                }
                break;
        }
    }
}
```

AvatarListItem.cs

```
using UnityEngine;
using UnityEngine.UI;
/**
Класс пункта списка аватаров

В приложении есть список возможных аватаров.
Список состоит из ряда объектов данного класса.
Данный класс хранит информацию об аватаре, который будет отображаться в виртуальной комнате.
Данный элемент так же отвечает за отображение аватара в списке AvatarList.

@param avatarImage Image, в котором будет выведено изображение аватара.
@param selectCheckboxImage Image выбранного checkbox-а.
@param unselectCheckboxImage Image невыбранного checkbox-а.
@param selectFrameImage Image рамки вокруг изображения аватара.
@see AvatarList
 */
public class AvatarListItem : MonoBehaviour
{
    [SerializeField] private Image _avatarImage;
    [SerializeField] private Image _selectCheckboxImage;
    [SerializeField] private Image _unselectCheckboxImage;
    [SerializeField] private Image _selectFrameImage;

    private AvatarList _controllerList;

    private bool _isSelected;
    private string _avatarName;

    private void Update()
    {
        _selectCheckboxImage.enabled = _isSelected;
        _selectFrameImage.enabled = _isSelected;
        _unselectCheckboxImage.enabled = !_isSelected;
    }

    /**
     Сеттер изображения аватара.
    @param [in] image Sprite изображения аватара.
     */
    public void SetAvatarImage(Sprite image)
    {
        _avatarImage.sprite = image;
    }

    /**
     Сеттер названия аватара.
    @param [in] name Название аватара.
     */
    public void SetAvatarName(string name)
    {
        _avatarName = name;
    }

    /**
     Сеттер состояния элемента списка.
    @param [in] isSelected Выбран ли элемент.
     */
    public void SetSelected(bool isSelected)
```

```
    {
        _isSelected = isSelected;
    }

    /**
     Сеттер списка управляющего данным элементом.
    @param [in] controllerList AvatarList управляющий данным элементом.
     */
    public void SetControllerList(AvatarList controllerList)
    {
        _controllerList = controllerList;
    }

    /**
     Геттер имени аватара.
    @return string имя аватара, загружаемое NetworkManager-ом.
     */
    public string GetAvatarName()
    {
        return _avatarName;
    }

    /**
     Геттер состояния элемента.
    @return bool выбран ли данный элемент.
     */
    public bool IsSelected()
    {
        return _isSelected;
    }

    /// Метод выбирающий данный элемент, если он не выбран.
    public void OnClick()
    {
        if (!_isSelected)
        {
            _controllerList?.ChangeCurrentElement(this);
        }
    }
}
```

ChangeUILayer.cs

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

/// Класс связывающий код состояния сервера и отображаемый в этом состоянии слой.
[System.Serializable]
public class ChangeLayers
{
    /// Код состояния сервера.
    public NetworkCode Code;
    /// Отображаемый Canvas при поступлении данного кода от сервера.
    public CanvasGroup Layer;
}

/**
 Класс отвечающий за плавную смену слоев на UI экране

@attention Для корректной работы данный класс требует, что бы в сцене присутствовали скрипты:
- NetworkManager;

@param catcher ComponentCatcher находящийся на данной сцене;
@param startLayer Canvas отображаемый изначально;
@param layers Список ChangeLayers, которые будут меняться в соответствии с указанными кодами;
@param fadeDuration Продолжительность затухания слоя;
@param fadeSmoothness Плавность затухания слоя;
@see ComponentCatcher; ChangeLayers; NetworkManager.
 */
public class ChangeUILayer : MonoBehaviour
{
    [SerializeField] private ComponentCatcher _catcher;
    [SerializeField] private CanvasGroup _startLayer;
    [SerializeField] private List<ChangeLayers> _layers;
    [Range(1, 120)]
    [SerializeField] private float _fadeDuration;
    [Range(0.00001f, 0.1f)]
    [SerializeField] private float _fadeSmoothness;

    private NetworkManager _networkManager;
    private Queue<ChangeLayers> _changeQueue;
    private float _currentFrame;
    private CanvasGroup _currentLayer;

    private void Awake()
    {
        _changeQueue = new Queue<ChangeLayers>();
        _currentFrame = 0;
    }

    private void Start()
    {
        _networkManager = _catcher.GetNetworkManager();
        if (_networkManager != null)
        {
            _networkManager.NetworConnectionEvent += OnNetworConnection;
        }
```

```
        if (_layers.Count > 0)
        {
            for (int i = 0; i < _layers.Count; i++)
            {
                _layers[i].Layer.alpha = 0;
                _layers[i].Layer.gameObject.SetActive(false);
            }
        }
        _startLayer.gameObject.SetActive(true);
        _startLayer.alpha = 1;
        _currentLayer = _startLayer;
    }

    private void Update()
    {
        _currentFrame += Time.deltaTime * 10;
        if (_currentFrame / _fadeDuration > 1)
        {
            _currentFrame = 0;
            if (_changeQueue.Count > 0)
            {
                ChangeLayers changeLayer = _changeQueue.Peek();
                if (changeLayer.Layer == _currentLayer)
                {
                    _changeQueue.Dequeue();
                }
                else
                {
                    changeLayer.Layer.gameObject.SetActive(true);
                    if (_currentLayer.alpha == 0 && changeLayer.Layer.alpha == 1)
                    {
                        _currentLayer.gameObject.SetActive(false);
                        _currentLayer = changeLayer.Layer;
                        _changeQueue.Dequeue();
                    }
                    else
                    {
                        _currentLayer.alpha = Mathf.Max(0, _currentLayer.alpha - _fadeSmoothness);
                        changeLayer.Layer.alpha = Mathf.Min(1, changeLayer.Layer.alpha + _fadeSmoothness);
                    }
                }
            }
        }

    }

    private void OnDestroy()
    {
        if (_networkManager != null)
        {
            _networkManager.NetworConnectionEvent -= OnNetworConnection;
        }
    }

    private void OnNetworConnection(NetworkCode code)
    {
        foreach (ChangeLayers layer in _layers)
        {
            if (layer.Code == code)
            {
                _changeQueue.Enqueue(layer);
                break;
            }
        }
    }

    /**
     Метод для произвольной смены слоя, не зависимо от состояния сервера.
    @param [in] layer Canvas, который мы хотим отобразить. Данный Canvas должен быть указан в layers.
     */
    public void ChangeLayer(CanvasGroup layer)
    {
        foreach (ChangeLayers changeLayer in _layers)
        {
            if (changeLayer.Layer == layer)
            {
                _changeQueue.Enqueue(changeLayer);
                break;
            }
        }
    }
}
```

KeyboardButton.cs

```
using UnityEngine;
using TMPro;

/**
 Класс кнопки виртуальной клавиатуры.
@param buttonText Текстовое поле кнопки.
@see VirtualKeyboardController
 */
public class KeyboardButton : MonoBehaviour
{
    [SerializeField] private TMP_Text _buttonText;

    private string _text;
```

```
    private VirtualKeyboardController _keyboardController;

    private bool _isSpecialButton;

    private void Awake()
    {
        _text = _buttonText.text;
        _isSpecialButton =
            _text == VirtualKeyboardController.BACKSPACE
            || _text == VirtualKeyboardController.SHIFT
            || _text == VirtualKeyboardController.CAPS_LOCK
            || _text == VirtualKeyboardController.TO_NUMBERS
            || _text == VirtualKeyboardController.BROWSE
            || _text == VirtualKeyboardController.ENTER
            || _text == VirtualKeyboardController.ESCAPE;
    }

    /**
     Метод устанавливающий контроллер клавиатуры для данной кнопки.

    Все кнопки на клавиатуре контролируются классом VirtualKeyboardController.
    Когда данный класс собирает кнопки, которые он контролирует,
    он так же должен через данный метод установить себя, как контролирующий класс для данной кнопки.
    @param [in] keyboardController Контроллер клавиатуры данной кнопки VirtualKeyboardController.
     */
    public void SetKeyboardController(VirtualKeyboardController keyboardController)
    {
        _keyboardController = keyboardController;
    }

    /**
     Метод устанавливающий символ в заглавный или в строчный.
    Не производит действия на специальные кнопки.
    @param [in] isUpperCase Если true, то установит символ в заглавный.
    Если false, то установит символ в строчный.
     */
    public void ChangeUpperCase(bool isUpperCase)
    {
        if (!_isSpecialButton)
        {
            if (isUpperCase)
            {
                if (_text != null)
                {
                    _text = _text.ToUpper();
                }
            }
            else
            {
                if (_text != null)
                {
                    _text = _text.ToLower();
                }
            }
            _buttonText.text = _text;
        }
    }

    /// Метод нажатия на кнопку
    public void OnClick()
    {
        _keyboardController?.ButtonAction(_text);
    }
}
```

KeyboardProvider.cs

```
using UnityEngine;
using TMPro;
using Oculus.Interaction;

/**
Класс, обеспечивающий взаимодействие UI с VirtualKeyboardController.

Взаимодействуя с ComponentCatcher, данный класс предоставляет элементам UI взаимодействовать с VirtualKeyboardController.

@attention Для корректной работы данный класс требует, что бы в сцене присутствовали скрипты:
- VirtualKeyboardController;

@param catcher ComponentCatcher, который используется для получения доступа к VirtualKeyboardController.
@see ComponentCatcher; VirtualKeyboardController; InterfaceHider

 */
public class KeyboardProvider : MonoBehaviour
{
    [SerializeField] private ComponentCatcher _catcher;

    private VirtualKeyboardController _keyboard;
    private GameObject _keyboardSurface;
    private InterfaceHider _keyboardHider;

    private void Start()
    {
        _keyboard = _catcher.GetVirtualKeyboard();

        if (_keyboard)
        {
```

```
            _keyboardHider = _keyboard.GetComponent<InterfaceHider>();
            _keyboardHider.OnInterfaseHide += DisableKeyboard;
            _keyboardSurface = _keyboard.GetComponentInParent<Cylinder>().gameObject;
            _keyboard.CloseVirtualKeyboard();
        }
    }

    private void OnDestroy()
    {
        _keyboardHider.OnInterfaseHide -= DisableKeyboard;
    }

    public void CreateVirtualKeyboard(TMP_InputField inputText)
    {
        _keyboardSurface?.SetActive(true);
        _keyboard?.CreateVirtualKeyboard(inputText);
    }

    public void CloseVirtualKeyboard()
    {
        _keyboard?.CloseVirtualKeyboard();
    }

    private void DisableKeyboard()
    {
        _keyboardSurface?.SetActive(false);
    }
}
```

VirtualKeyboardController.cs

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

/**
 Класс контроллер виртуальной клавиатуры.

Данный класс контролирует виртуальную клавиатуру.
В качестве параметров ему передаются объекты разных раскладок клавиатур.
Данный класс собирает все клавиши с данных раскладок и обеспечивает их работу.

Когда текстовому полю ввода нужно вызвать клавиатуру, это текстовое поле вызывает метод CreateVirtualKeyboard(TMP_InputField)
и передает себя в качестве параметра данного метода. В дальнейшем, до закрытия клавиатуры клавишей Esc,
все кнопки будут вводить символы в указанное текстовое поле.
@param enKeyboard Объект английской раскладки клавиатуры. Данный объект должен содержать компоненты KeyboardButton.
@param ruKeyboard Объект русской раскладки клавиатуры. Данный объект должен содержать компоненты KeyboardButton.
@param numberKeyboard Объект раскладки клавиатуры, содержащей спецсимволы. Данный объект должен содержать компоненты
↪  KeyboardButton.
@see InterfaceHider; KeyboardButton
 */
[RequireComponent(typeof(InterfaceHider))]
public class VirtualKeyboardController : MonoBehaviour
{
    /// Константа имени специальной кнопки
    public const string BACKSPACE = "←";
    /// Константа имени специальной кнопки
    public const string SHIFT = "↑";
    /// Константа имени специальной кнопки
    public const string CAPS_LOCK = "Caps\nLock";
    /// Константа имени специальной кнопки
    public const string TO_NUMBERS = "123";
    /// Константа имени специальной кнопки
    public const string TO_LETTERS = "abc";
    /// Константа имени специальной кнопки
    public const string BROWSE = "(#)";
    /// Константа имени специальной кнопки
    public const string ENTER = "Enter";
    /// Константа имени специальной кнопки
    public const string ESCAPE = "Esc";

    [SerializeField] private Transform _enKeyboard;
    [SerializeField] private Transform _ruKeyboard;
    [SerializeField] private Transform _numberKeyboard;

    private List<KeyboardButton> _buttons;

    private TMP_InputField _inputText;
    private InterfaceHider _hider;

    private bool _isShift;
    private bool _isCapsLock;
    private bool _isEnKeyboard;
    private bool _isNumberKey;

    private void Awake()
    {
        _hider = GetComponent<InterfaceHider>();
    }

    private void Start()
    {
        _isShift = true;
        _isCapsLock = false;
        _isEnKeyboard = true;
        _isNumberKey = false;

        _buttons = new List<KeyboardButton>();
```

```
    FiilButtonsArray(_enKeyboard);
    FiilButtonsArray(_ruKeyboard);
    FiilButtonsArray(_numberKeyboard);

    ChangeLetterKeyboard();
    transform.localScale = Vector3.zero;
}

/**
 Метод, показывающий виртуальную клавиатуру.

Данный метод вызывается текстовым полем ввода.
Входным параметром является поле ввода, в которое будет вводить текст данная клавиатура.
@param [in] inputText TMP_InputField, в которое будет вводить текст данная клавиатура.
 */
public void CreateVirtualKeyboard(TMP_InputField inputText)
{
    _hider.ShowInterface();
    _inputText = inputText;
}

/**
 Метод, закрывающий виртуальную клавиатуру.

Данный метод скроет виртуальную клавиатуру
и отвяжет ее от текстового поля, в которое осуществлялся ввод.
 */
public void CloseVirtualKeyboard()
{
    _inputText = null;
    _hider.HideInterface();
}

/**
 Метод, вызываемый кнопкой клавиатуры.

В качестве входного параметра, кнопка вызвавшая данный метод, передает свой текст.

Если данный метод вызывается обычной кнопкой,
то произойдет ввод текста кнопки в текстовое поле.
Если данный метод вызывается специальной кнопкой,
то произойдет действие соответствующие специальной кнопке.
@param [in] buttonName Имя кнопки, вызывающей данный метод.
 */
public void ButtonAction(string buttonName)
{
    switch (buttonName)
    {
        case BACKSPACE:
            Backspace();
            break;
        case SHIFT:
            _isShift = !_isShift;
            ChangeUpperCase(_isShift);
            break;
        case CAPS_LOCK:
            _isCapsLock = !_isCapsLock;
            ChangeUpperCase(_isCapsLock);
            break;
        case TO_NUMBERS:
        case TO_LETTERS:
            ChangeNumberKey();
            break;
        case BROWSE:
            _isEnKeyboard = !_isEnKeyboard;
            ChangeLetterKeyboard();
            break;
        case ENTER:
            if (_inputText as TMP_InputField != null)
            {
                CloseVirtualKeyboard();
            }
            else
            {
                EnterText("\n");
            }
            break;
        case ESCAPE:
            CloseVirtualKeyboard();
            break;
        default:
            EnterText(buttonName);
            break;
    }
}

private void FiilButtonsArray(Transform keyboard)
{
    foreach (KeyboardButton button in keyboard.gameObject.GetComponentsInChildren<KeyboardButton>())
    {
        button.SetKeyboardController(this);
        _buttons.Add(button);
    }

}

private void ChangeUpperCase(bool isUpperCase)
{
    foreach (KeyboardButton button in _buttons)
    {
        button.ChangeUpperCase(isUpperCase);
    }
```

```
    }

    private void ChangeNumberKey()
    {
        _isNumberKey = !_isNumberKey;
        if (_isNumberKey)
        {
            _enKeyboard.gameObject.SetActive(false);
            _ruKeyboard.gameObject.SetActive(false);
            _numberKeyboard.gameObject.SetActive(true);
        }
        else
        {
            ChangeLetterKeyboard();
        }
    }

    private void ChangeLetterKeyboard()
    {
        if (_isEnKeyboard)
        {
            _enKeyboard.gameObject.SetActive(true);
            _ruKeyboard.gameObject.SetActive(false);
            _numberKeyboard.gameObject.SetActive(false);
        }
        else
        {
            _enKeyboard.gameObject.SetActive(false);
            _ruKeyboard.gameObject.SetActive(true);
            _numberKeyboard.gameObject.SetActive(false);
        }
        ChangeUpperCase(_isCapsLock || _isShift);
    }

    private void Backspace()
    {
        if (_inputText != null && _inputText.text.Length > 0)
        {
            _inputText.text = _inputText.text.Substring(0, _inputText.text.Length - 1);
        }
    }

    private void EnterText(string text)
    {
        if (_inputText != null)
        {
            _inputText.text = _inputText.text + text;
            if (_isShift)
            {
                _isShift = !_isShift;
                ChangeUpperCase(_isShift);
            }
        }
    }
}
```

InterfaceHider.cs

```
using UnityEngine;
using UnityEngine.Events;

/**
 Скрипт для плавного появления/скрытия интерфейса

Данный скрипт используется, если по нажатию на кнопку нам нужно показать/скрыть интерфейс.

@param hideDuration Скорость появления/скрытия интерфейса.
 */
public class InterfaceHider : MonoBehaviour
{
    /// Событие скрытия интерфейса.
    public UnityAction OnInterfaseHide;
    /// Событие появление интерфейса.
    public UnityAction OnInterfaseShow;

    [SerializeField] private float _hideDuration = 1f;

    private Vector3 _currentScale;
    private bool _isHide;

    private void Awake()
    {
        _isHide = false;
        _currentScale = new Vector3(transform.localScale.x, transform.localScale.y, transform.localScale.z);
    }

    /// Метод скрытия интерфейса.
    public void HideInterface()
    {
        if (!_isHide)
        {
            _isHide = true;
            transform.LeanScale(Vector3.zero, _hideDuration)
                .setEaseInBack()
                .setOnComplete(() => OnInterfaseHide?.Invoke());
        }
    }
```

```
    /// Метод появления интерфейса.
    public void ShowInterface()
    {
        if (_isHide)
        {
            _isHide = false;
            transform.LeanScale(_currentScale, _hideDuration).setOnComplete(() => OnInterfaseShow?.Invoke());
        }
    }

    /// Метод для переключения видимости интерфейса на противоположное.
    public void SwitchInterfaceHiding()
    {
        if (_isHide)
        {
            ShowInterface();
        }
        else
        {
            HideInterface();
        }
    }
}
```

MenuSwapper.cs

```
using UnityEngine;

/**
Скрипт для переключения различных UI через InterfaceHider

Переключение вкладок меню может быть реализовано различными способами.
В данном случае, у нас есть множество UI, каждый из которых это отдельная вкладка меню.
Данный скрипт отображает одну из этих вкладок и скрывает все остальные.
@param menuUIs Массив InterfaceHider, которым управляет данный скрипт.
@see InterfaceHider
 */
public class MenuSwapper : MonoBehaviour
{
    [SerializeField] private InterfaceHider[] _menuUIs;

    private void Start()
    {
        HideAll();
    }

    /**
     Метод переключения на другую вкладку меню.
    @param [in] menuNumber Индекс вкладки, на которую необходимо переключиться.
     */
    public void SwitchMenu(int menuNumber)
    {
        HideAll();
        if (menuNumber >= 0 && menuNumber < _menuUIs.Length)
        {
            _menuUIs[menuNumber].ShowInterface();
        }
    }

    /// Метод для скрытия всех вкладок меню.
    public void HideAll()
    {
        foreach (InterfaceHider hider in _menuUIs)
        {
            hider.HideInterface();
        }
    }
}
```

ButtonTextChanger.cs

```
using UnityEngine;
using UnityEngine.EventSystems;
using TMPro;

/**
Компонент меняющий текст в текстовом поле при наведении курсора на кнопку

@param textField TextMeshPro, в котором будет меняться текст.
@param text string текст, который появиться в textField при наведении на кнопку.
 */
public class ButtonTextChanger : MonoBehaviour, IPointerEnterHandler
{
    [SerializeField] private TMP_Text _textField;
    [TextArea(2,4)]
    [SerializeField] private string _text;

    /// Реализация метода интерфейса. Метод вызываемый при наведении курсора на кнопку.
    public void OnPointerEnter(PointerEventData eventData)
    {
        _textField.text = _text;
    }
```

```
}
```

GifAnimation.cs

```
using UnityEngine;
using UnityEngine.UI;

/**
Скрипт для воспроизведения gif анимации в UI

@param frameRate Количество кадров в секунду (30 оптимально).
@param frames Массив кадров gif анимации.
 */
[RequireComponent(typeof(RawImage))]
public class GifAnimation : MonoBehaviour
{
    [Range(1, 120)]
    [SerializeField] private float _frameRate;
    [SerializeField] private Texture2D[] _frames;

    private RawImage _image;
    private float _index;

    private void Awake()
    {
        _image = GetComponent<RawImage>();
        _index = 0;
    }

    private void Update()
    {
        _index += Time.deltaTime * _frameRate;
        _index = _index % _frames.Length;
        _image.texture = _frames[(int)_index];
    }
}
```

SetupRoomName.cs

```
using UnityEngine;
using Photon.Pun;
using TMPro;

/**
Скрипт для установки имени текущей комнаты в указанное текстовое поле

@attention Для корректной работы данный класс требует, что бы в сцене присутствовали скрипты:
- ComponentCatcher;
- NetworkManager;

@param roomNameText Текстовое поле в которое будет записано "Имя комнаты: " + имя_текущей_комнаты.
@see NetworkManager; ComponentCatcher
 */
public class SetupRoomName : MonoBehaviour
{
    [SerializeField] private TMP_Text _roomNameText;

    private NetworkManager _networkManager;

    private void Start()
    {
        ComponentCatcher catcher = FindObjectOfType<ComponentCatcher>();
        _networkManager = catcher?.GetNetworkManager();
        if (_networkManager)
        {
            _networkManager.NetworConnectionEvent += OnNetworConnection;
        }

        _roomNameText.text = "Имя комнаты: Вы не находитесь в комнате";
    }

    private void OnDestroy()
    {
        if (_networkManager != null)
        {
            _networkManager.NetworConnectionEvent -= OnNetworConnection;
        }
    }

    private void OnNetworConnection(NetworkCode code)
    {
        switch (code)
        {
            case NetworkCode.CONNECT_TO_ROOM_COMPLETE:
                if (PhotonNetwork.CurrentRoom != null || PhotonNetwork.CurrentRoom.Name == "")
                {
                    _roomNameText.text = "Имя комнаты: " + PhotonNetwork.CurrentRoom?.Name;
                }
                else
                {
                    _roomNameText.text = "Имя комнаты: Вы не находитесь в комнате";
                }
```

```
                break;
            case NetworkCode.CONNECT_TO_LOBBY_COMPLETE:
                _roomNameText.text = "Имя комнаты: Вы находитесь в лобби";
                break;
        }
    }
}
```

RoomListItem.cs

```
using Photon.Realtime;
using UnityEngine;
using TMPro;

/**
Класс пункта списка комнат

В приложении есть список комнат, к которым мы можем подключаться.
Список состоит из ряда объектов данного класса.
Данный класс хранит информацию о комнате, к которой мы можем подключиться через данный пункт списка.

@param buttonText Текстовое поле, в которое будет вписано название пункта списка.
@see UIRoomListController
 */
public class RoomListItem : MonoBehaviour
{
    [SerializeField] private TMP_Text _buttonText;

    private RoomInfo _roomInfo;
    private UIRoomListController _roomConnector;

    /**
     Настройка пункта меню.

    После создания пункта списка, в него нужно занести информацию, которую он будет в себе хранить.
    Так же через данный метод передается UIConnectToRoom, отвечающий за работу с пунктами списка.
    @param [in] roomInfo Информация о комнате, к которой можно подключиться через данный пункт списка.
    @param [in] roomConnector UIRoomListController управляющий данным пунктом списка.
     */
    public void SetUp(RoomInfo roomInfo, UIRoomListController roomConnector)
    {
        _roomInfo = roomInfo;
        _buttonText.text = "     " + roomInfo.Name;
        _roomConnector = roomConnector;
    }

    /**
     Геттер Информации о комнате, к которой можно подключиться через данный пункт списка.
    @return RoomInfo Информации о комнате, к которой можно подключиться через данный пункт списка.
     */
    public RoomInfo GetRoomInfo()
    {
        return _roomInfo;
    }

    /// Метод нажатия на пункт списка.
    public void OnClick()
    {
        _roomConnector?.ChangeCurrentLoadRoom(gameObject);
    }
}
```

UIRoomListController.cs

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

/**
Класс, контролирующий все элементы списка комнат, к которым можно подключиться.

Данный класс хранит информацию о всех существующий на данный момент комнатах.
Он отображает эту информацию через элементы списка комнат, к которым можно подключиться.

Так же данный класс связывается с NetworkManager для получения информации о текущих комнатах.
И отправляет запрос на подключение к выбранной комнате.

Данному классу, через метод SetRoomType(), передается тип комнаты.
После чего контроллер выведет на экран список из элементов RoomListItem, для существующий комнат данного типа.
При выборе одного из элементов списка, можно подключиться к указанной в нем комнате, при помощи метода ConnectToSelectedRoom().
Так же данный класс создает новые комнаты с уникальным именем для комнат данного типа: имя_комнаты + ID_в_списке, через метод
↪  CreateNewRoom().

@attention Для корректной работы данный класс требует, что бы в сцене присутствовали скрипты:
- NetworkManager;

@param catcher ComponentCatcher данной сцены.
@param roomListContent UI слой, в который будут отображаться элементы списка.
@param roomListItemPrefab Prefab элемента списка. Prefab должен содержать компонент RoomListItem.
@param connectToRoomButton Кнопка подключения к комнате, которая сейчас выбрана в списке.
@param defaultButtonColor Цвет не выбранного элемента списка.
```

```
@param selectedButtonColor Цвет выбранного элемента списка.
@see ComponentCatcher; NetworkManager; RoomListItem
 */
public class UIRoomListController : MonoBehaviour
{
    [SerializeField] private ComponentCatcher _catcher;
    [SerializeField] private Transform _roomListContent;
    [SerializeField] private GameObject _roomListItemPrefab;
    [SerializeField] private GameObject _connectToRoomButton;
    [SerializeField] private Color _defaultButtonColor;
    [SerializeField] private Color _selectedButtonColor;

    private NetworkManager _networkManager;

    private int _roomType;
    private List<RoomInfo> _rooms;
    private List<RoomListItem> _roomButtons;
    private RoomListItem _currentLoadRoom;

    private void Start()
    {
        _networkManager = _catcher.GetNetworkManager();
        if (_networkManager)
        {
            _networkManager.RoomListUpdate += OnRoomListUpdate;
        }
        _connectToRoomButton.SetActive(false);
    }

    private void OnEnable()
    {
        UpdateUI();
    }

    private void OnDestroy()
    {
        if (_networkManager)
        {
            _networkManager.RoomListUpdate -= OnRoomListUpdate;
        }
    }

    /**
     Метод установки типа комнаты.

    @param [in] roomType Индекс типа комнаты в поле defaultRooms компонента NetworkManager.
     */
    public void SetRoomType(int roomType)
    {
        _roomType = roomType;
        UpdateUI();
    }

    /**
     Метод изменения текущего выбранного компонента списка.

    Данный метод вызовет элемент списка, когда на него кликнут и в качестве входного параметра он передаст себя.
    @param [in] selectedButton Новый выбранный элемент списка.
     */
    public void ChangeCurrentLoadRoom(GameObject selectedButton)
    {
        foreach (Transform trans in _roomListContent)
        {
            Image buttonImage = trans.gameObject.GetComponent<Image>();
            if (buttonImage)
            {
                if (trans.gameObject == selectedButton)
                {
                    buttonImage.color = _selectedButtonColor;
                    _currentLoadRoom = selectedButton.GetComponent<RoomListItem>();
                    _connectToRoomButton.SetActive(true);
                }
                else
                {
                    buttonImage.color = _defaultButtonColor;
                }
            }
        }
    }

    /**
     Метод, делающий все элементы списка не выбранными.
     */
    public void ClearCurrentLoadRoom()
    {
        foreach (Transform trans in _roomListContent)
        {
            Image buttonImage = trans.gameObject.GetComponent<Image>();
            if (buttonImage)
            {
                buttonImage.color = _defaultButtonColor;
            }
        }
        _currentLoadRoom = null;
        _connectToRoomButton.SetActive(false);
    }

    /// Метод, осуществляющий попытку подключения к выбранной в текущей момент комнате.
    public void ConnectToSelectedRoom()
    {
        if (_currentLoadRoom)
        {
            _networkManager.JoinRoom(_currentLoadRoom.GetRoomInfo());
```

```
        }
    }

    /// Метод, создающий новую комнату с ID равным количеству уже существующих комнат данного типа.
    public void CreateNewRoom()
    {
        _networkManager.InitRoom(_roomType, _rooms.Count);
    }

    private void OnRoomListUpdate(List<RoomInfo> roomList)
    {
        _rooms = roomList;
        UpdateUI();
    }

    private void UpdateUI()
    {
        foreach (Transform trans in _roomListContent)
        {
            Destroy(trans.gameObject);
        }

        if (_roomButtons == null)
        {
            _roomButtons = new List<RoomListItem>();
        }
        _roomButtons.Clear();
        _currentLoadRoom = null;
        _connectToRoomButton.SetActive(false);

        if (_rooms != null)
        {
            foreach (RoomInfo room in _rooms)
            {
                if ((int)room.CustomProperties[RoomSettings.ROOM_INDEX] == _roomType)
                {
                    RoomListItem item = Instantiate(_roomListItemPrefab, _roomListContent).GetComponent<RoomListItem>();
                    if (item)
                    {
                        item.SetUp(room, this);
                        _roomButtons.Add(item);

                        item.gameObject.GetComponent<Image>().color = _defaultButtonColor;
                    }
                }
            }
        }
    }
}
```

VRLogger.cs

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

/**
Скрипт для вывода логов внутри игры

@param logField TextMeshPro, в который будут выводиться логи.
@param maxLine Максимальное число строк в логах.
 */
public class VRLogger : MonoBehaviour
{
    [SerializeField] private TMP_Text _logField;
    [Range(1, 1000)]
    [SerializeField] private int _maxLine;

    private List<string> _lines;

    private void Awake()
    {
        _logField.text = "";
        _lines = new List<string>();
    }

    /**
     Метод для вывода лога.
    @param [in] text string текст выводимого лога.
     */
    public void Log(string text)
    {
        if (_lines.Count >= _maxLine)
        {
            _lines.RemoveAt(0);
        }
        _lines.Add(text);
        FillLog();
    }

    /// Геттер текста логов.
    public string GetTextFromLogger()
    {
        return _logField.text;
    }

    /// Геттер текста логов в виде списка строк.
    public List<string> GetLoggerLines()
```

```
    {
        return _lines;
    }

    /**
     Сеттер множества строк логов, через список строк логов.
     @param [in] lines Список строк логов.
     */
    public void SetLoggerLines(List<string> lines)
    {
        this._lines = lines;
    }

    /// Метод для очистки логов.
    public void ClearLog()
    {
        _lines.Clear();
        FillLog();
    }

    private void FillLog()
    {
        _logField.text = "";
        foreach (string line in _lines)
        {
            _logField.text += line + "\n";
        }
    }
}
```

VRLoggersManager.cs

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

/**
Скрипт, отвечающий за вывод логов во все VRLogger на сцене

@attention Для корректной работы vrLogger-а необходимо перед началом работы установить ему NetworkManager,
воспользовавшись методом SetNetworkManager(NetworkManager networkManager).
@see NetworkManager; VRLogger
 */
public class VRLoggersManager : MonoBehaviour
{
    private List<VRLogger> _vrLoggers;
    private NetworkManager _networkManager;

    private void Awake()
    {
        RefreshVrLogger();
    }

    private void OnDestroy()
    {
        if (_networkManager != null)
        {
            _networkManager.NetworConnectionEvent -= OnNetworConnection;
        }
    }

    private void RefreshVrLogger()
    {
        if (_vrLoggers == null)
        {
            _vrLoggers = new List<VRLogger>();
        }
        List<string> oldLines = new List<string>();
        if (_vrLoggers.Count > 0)
        {
            oldLines.AddRange(_vrLoggers[0].GetLoggerLines());
            foreach (VRLogger vrLogge in _vrLoggers)
            {
                vrLogge.ClearLog();
            }
        }
        _vrLoggers.Clear();
        VRLogger[] vrLoggers = FindObjectsByType<VRLogger>(FindObjectsSortMode.None);
        if (vrLoggers.Length > 0)
        {
            foreach (VRLogger vrLogger in vrLoggers)
            {
                vrLogger.SetLoggerLines(oldLines);
                _vrLoggers.Add(vrLogger);

            }
        }
        else
        {
            Debug.LogWarning("No VRLogger found in scene.");
        }
    }

    /**
     Метод для вывода лога.
    @param [in] log string текст выводимого лога.
     */
    public void Log(string log)
    {
```

```
        if (_vrLoggers != null)
        {
            foreach (VRLogger vrLogger in _vrLoggers)
            {
                vrLogger.Log(log);
            }
        }
        else
        {
            Debug.LogWarning("No vrLogger found in scene.");
            Debug.Log(log);
        }
    }

    /**
     Сеттер NetworkManager-а.
    @attention Для корректной работы vrLogger-а необходимо перед началом работы установить ему NetworkManager.
    @param [in] networkManager NetworkManager.
     */
    public void SetNetworkManager(NetworkManager networkManager)
    {
        if (_networkManager != null)
        {
            _networkManager.NetworConnectionEvent -= OnNetworConnection;
        }
        this._networkManager = networkManager;
        networkManager.NetworConnectionEvent += OnNetworConnection;
    }

    private void OnNetworConnection(NetworkCode code)
    {
        // Если произошло завершение какого-то подключения
        // коды завершений 2**
        if (((int)code) / 100 == 2)
        {
            RefreshVrLogger();
        }
    }
}
```