**ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ**

**НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ**

**«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»**

Факультет компьютерных наук

Образовательная программа бакалавриата «Программная инженерия»

|  |  |  |
| --- | --- | --- |
| СОГЛАСОВАНО  Научный руководитель,  младший научный сотрудник  МНУЛ ИССА ФКН, к. т. н.  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ О. В. Максименкова  «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_ 2020 г. |  | УТВЕРЖДАЮ  Академический руководитель образовательной программы «Программная инженерия»  профессор департамента программной инженерии, канд. техн. наук  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ В.В. Шилов  «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_ 2020 г. |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | | ***Подп. и дата*** |  | | ***Инв. № дубл.*** |  | | ***Взам. инв. №*** |  | | ***Подп. и дата*** |  | | ***Инв. № подл*** |  | | **Браузерная мультиплейерная игра "Minigames"**  **Текст программы**  **ЛИСТ УТВЕРЖДЕНИЯ**  **RU.17701729.04.01-01 12 01-1-ЛУ** | | |
|  |  | |
| Исполнитель  студент группы БПИ 196  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ /М.С. Шестаков /  «\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2020 г. | |
|  | | |
|  | |  |

**Москва 2020**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| УТВЕРЖДЕН  RU.17701729.04.01-01 12 01-1-ЛУ |  | |  | |
| |  |  | | --- | --- | | ***Подп. и дата*** |  | | ***Инв. № дубл.*** |  | | ***Взам. инв. №*** |  | | ***Подп. и дата*** |  | | ***Инв. № подл*** |  | | **Браузерная мультиплейерная игра "Minigames"**  **Текст программы**  **RU.17701729.04.01-01 12 01-1**  **Листов 191** | | | | |
|  | |  | | |
|  | | |
|  | | | | |
|  | | | |  |

**Москва 2020**

**АННОТАЦИЯ**

В данном документе представлен текст программы “Браузерная мультиплейерная игра “Minigames””. Программа была написана на языке C# с помощью среды Jetbrains Rider 2020.1.2. Также текст программы доступен в облачном хранилище [13].

Оформление данного документа ведётся согласно требованиям соответствующих ГОСТ.

**СОДЕРЖАНИЕ**

Оглавление

[1. ТЕКСТ ПРОГРАММЫ КЛИЕНТА 7](#_Toc41217720)

[1.1 BulletTrailRenderer.cs 7](#_Toc41217721)

[1.2 Client.cs 8](#_Toc41217722)

[1.3 ObjectId.cs 11](#_Toc41217723)

[1.4 sClient.cs 17](#_Toc41217724)

[1.5 Triangle.cs 20](#_Toc41217725)

[1.6 ActionController.cs 22](#_Toc41217726)

[1.7 AIController.cs 24](#_Toc41217727)

[1.8 CameraFollower.cs 25](#_Toc41217728)

[1.9 CharacterAnimator.cs 25](#_Toc41217729)

[1.10 CharacterController.cs 28](#_Toc41217730)

[1.11 HandAnimationBlender.cs 28](#_Toc41217731)

[1.12 MotionController.cs 30](#_Toc41217732)

[1.13 PlayerAnimationState.cs 34](#_Toc41217733)

[1.14 PlayerController.cs 34](#_Toc41217734)

[1.15 IAction.cs 36](#_Toc41217735)

[1.16 PushAction.cs 36](#_Toc41217736)

[1.17 ShootAction.cs 38](#_Toc41217737)

[1.18 ShootPistolAction.cs 39](#_Toc41217738)

[1.19 ShootSemiautoAction.cs 40](#_Toc41217739)

[1.20 BombGun.cs 41](#_Toc41217740)

[1.21 GunState.cs 43](#_Toc41217741)

[1.22 IGun.cs 43](#_Toc41217742)

[1.23 Pistol.cs 43](#_Toc41217743)

[1.24 ReloadingGun.cs 45](#_Toc41217744)

[1.25 SemiautoGun.cs 50](#_Toc41217745)

[1.26 ShootSystem.cs 51](#_Toc41217746)

[1.27 ShotGun.cs 54](#_Toc41217747)

[1.28 BombGunController.cs 56](#_Toc41217748)

[1.29 GunController.cs 56](#_Toc41217749)

[1.30 PistolController.cs 57](#_Toc41217750)

[1.31 SemiautoController.cs 58](#_Toc41217751)

[1.32 ShotgunController.cs 58](#_Toc41217752)

[1.33 DamageSource.cs 58](#_Toc41217753)

[1.34 HPChange.cs 60](#_Toc41217754)

[1.35 HPController.cs 60](#_Toc41217755)

[1.36 CommandsSystem.cs 63](#_Toc41217756)

[1.37 ICommand.cs 71](#_Toc41217757)

[1.38 AddOrChangeInstance.cs 71](#_Toc41217758)

[1.39 AddPlayerToGame.cs 72](#_Toc41217759)

[1.40 ApplyForceCommand.cs 72](#_Toc41217760)

[1.41 ChangeHPCommand.cs 73](#_Toc41217761)

[1.42 ChangePlayerProperty.cs 74](#_Toc41217762)

[1.43 ChangePlayerScore.cs 74](#_Toc41217763)

[1.44 CreateChatMessageCommand.cs 75](#_Toc41217764)

[1.45 DrawPositionTracerCommand.cs 76](#_Toc41217765)

[1.46 DrawTargetedTracerCommand.cs 76](#_Toc41217766)

[1.47 ExplodeBombCommand.cs 77](#_Toc41217767)

[1.48 PickCoinCommand.cs 78](#_Toc41217768)

[1.49 PickUpGunCommand.cs 79](#_Toc41217769)

[1.50 PlayerPushCommand.cs 80](#_Toc41217770)

[1.51 SetAfterShowResultsCommand.cs 81](#_Toc41217771)

[1.52 SetGameMode.cs 81](#_Toc41217772)

[1.53 SetPlatformStateCommand.cs 82](#_Toc41217773)

[1.54 SpawnParabolaFlyingCommand.cs 82](#_Toc41217774)

[1.55 SpawnPlayerCommand.cs 83](#_Toc41217775)

[1.56 SpawnPrefabCommand.cs 85](#_Toc41217776)

[1.57 StartGameCommand.cs 86](#_Toc41217777)

[1.58 TakeOwnCommand.cs 86](#_Toc41217778)

[1.59 ClientEditor.cs 87](#_Toc41217779)

[1.60 WebGLEditorScript.cs 87](#_Toc41217780)

[1.61 EventsHandler.cs 88](#_Toc41217781)

[1.62 EventsManager.cs 90](#_Toc41217782)

[1.63 GameManager.cs 91](#_Toc41217783)

[1.64 Instance.cs 97](#_Toc41217784)

[1.65 InstanceManager.cs 98](#_Toc41217785)

[1.66 MatchesManager.cs 99](#_Toc41217786)

[1.67 MatchInfo.cs 104](#_Toc41217787)

[1.68 Player.cs 105](#_Toc41217788)

[1.69 PlayersManager.cs 107](#_Toc41217789)

[1.70 PlayerStorage.cs 109](#_Toc41217790)

[1.71 Bomb.cs 110](#_Toc41217791)

[1.72 BombOnCollisionExploder.cs 112](#_Toc41217792)

[1.73 BombTriggerHPExploder.cs 113](#_Toc41217793)

[1.74 Coin.cs 113](#_Toc41217794)

[1.75 MovingPlatform.cs 114](#_Toc41217795)

[1.76 ParabolaFlyingObject.cs 117](#_Toc41217796)

[1.77 GameModeFunctions.cs 119](#_Toc41217797)

[1.78 IGameMode.cs 121](#_Toc41217798)

[1.79 PickCoinsGameMode.cs 121](#_Toc41217799)

[1.80 ShooterGameMode.cs 123](#_Toc41217800)

[1.81 InterpolationFunctions.cs 126](#_Toc41217801)

[1.82 ManagedGameObject.cs 130](#_Toc41217802)

[1.83 UnmanagedGameObject.cs 131](#_Toc41217803)

[1.84 PlayerManagedGameObject.cs 135](#_Toc41217804)

[1.85 PlayerUnmanagedGameObject.cs 135](#_Toc41217805)

[1.86 GameObjectProperty.cs 136](#_Toc41217806)

[1.87 IGameObjectProperty.cs 138](#_Toc41217807)

[1.88 PlayerProperty.cs 139](#_Toc41217808)

[1.89 Request.cs 141](#_Toc41217809)

[1.90 RequestsManager.cs 143](#_Toc41217810)

[1.91 Response.cs 144](#_Toc41217811)

[1.92 ClientCommandsRoom.cs 146](#_Toc41217812)

[1.93 CommandsHandler.cs 150](#_Toc41217813)

[1.94 WebSocketHandler.cs 152](#_Toc41217814)

[1.95 Server.cs 155](#_Toc41217815)

[1.96 AutoRotateToCamera.cs 156](#_Toc41217816)

[1.97 DebugUI.cs 157](#_Toc41217817)

[1.98 MainUIController.cs 158](#_Toc41217818)

[1.99 MultiImagePanel.cs 166](#_Toc41217819)

[1.100 StartUIController.cs 167](#_Toc41217820)

[1.101 AutoDisableRendererOnStart.cs 169](#_Toc41217821)

[1.102 AutoHideOnStart.cs 170](#_Toc41217822)

[1.103 AutoID.cs 170](#_Toc41217823)

[1.104 AutoMatchJoiner.cs 171](#_Toc41217824)

[1.105 gUtil.cs 172](#_Toc41217825)

[1.106 IOwnedEventHandler.cs 172](#_Toc41217826)

[1.107 RotatingItem.cs 173](#_Toc41217827)

[2. ТЕКСТ ПРОГРАММЫ СЕРВЕРА 175](#_Toc41217828)

[2.1 data.js 175](#_Toc41217829)

[2.2 httpserver.js 175](#_Toc41217830)

[2.3 index.js 176](#_Toc41217831)

[2.4 JsonRequest.js 177](#_Toc41217832)

[2.5 logger.js 177](#_Toc41217833)

[2.6 match\_maker.js 178](#_Toc41217834)

[2.7 MessageFlags.js 181](#_Toc41217835)

[2.8 messages\_handler.js 182](#_Toc41217836)

[2.9 MessageType.js 184](#_Toc41217837)

[2.10 Room.js 184](#_Toc41217838)

[2.11 Util.js 187](#_Toc41217839)

[2.12 websocketserver.js 188](#_Toc41217840)

[ЛИСТ РЕГИСТРАЦИИ ИЗМЕНЕНИЙ 191](#_Toc41217841)

# ТЕКСТ ПРОГРАММЫ КЛИЕНТА

## BulletTrailRenderer.cs

﻿using UnityEngine;  
  
/// <summary>  
/// Компонента для отрисовки следа от пули  
/// </summary>  
public class BulletTrailRenderer : MonoBehaviour {  
  
 /// <summary>  
 /// Является ли данная компонента главной, от которой все копируются  
 /// </summary>  
 private bool main = true;  
 /// <summary>  
 /// Скорость  
 /// </summary>  
 public float speed = 10;  
 /// <summary>  
 /// Первая позиция  
 /// </summary>  
 public Vector3 v1;  
 /// <summary>  
 /// Вторая позиция  
 /// </summary>  
 public Vector3 v2;  
  
 /// <summary>  
 /// Создаёт след от пули  
 /// </summary>  
 /// <param name="v1">Стартовая позиция</param>  
 /// <param name="v2">Конечная позиция</param>  
 public void MoveFromTo(Vector3 v1,   
 Vector3 v2) {  
   
 var go = Instantiate(gameObject, v1, Quaternion.identity);  
 go.GetComponent<BulletTrailRenderer>().\_moveTo(v2);  
 }  
  
 /// <summary>  
 /// Устанавливает цель, к которой должен следовать данный след  
 /// </summary>  
 /// <param name="v2"></param>  
 private void \_moveTo(Vector3 v2) {  
 this.v1 = transform.position;  
 this.v2 = v2;  
 main = false;  
 }  
   
 /// <summary>  
 /// Передвигает след от пули  
 /// </summary>  
 void Update() {  
 if (main) return;  
 transform.position = Vector3.MoveTowards(transform.position, v2, speed \* Time.deltaTime);  
 if (transform.position == v2) {  
 Destroy(gameObject);  
 }  
 }  
}

## Client.cs

﻿  
using System;  
using UnityEngine;  
using Random = UnityEngine.Random;  
using UnityEngine.Assertions;  
using System.Collections.Generic;  
using System.Reflection;  
using Character.Guns;  
using CommandsSystem.Commands;  
using GameMode;  
using Networking;  
  
using Debug = UnityEngine.Debug;  
  
/// <summary>  
/// Класс с функциями для работы с игровым полем  
/// </summary>  
public class Client : MonoBehaviour  
{  
 /// <summary>  
 /// Статическая ссылка на Client (синглтон)  
 /// </summary>  
 public static Client client { get; private set; }  
  
 /// <summary>  
 /// Рендерер следов от пуль  
 /// </summary>  
 public BulletTrailRenderer bulletTrailRenderer;  
  
 /// <summary>  
 /// Объект, хранящиц границу внутри которой можно создавать объекты  
 /// </summary>  
 public GameObject spawnBorder = null;  
   
   
 /// <summary>  
 /// Граница игрового поля  
 /// </summary>  
 public TrianglePolygon spawnPolygon;  
   
 /// <summary>  
 /// Главный персонаж   
 /// </summary>  
 public GameObject mainPlayerObj;  
 /// <summary>  
 /// Камера  
 /// </summary>  
 public GameObject cameraObj;  
   
 /// <summary>  
 /// Слоаврь префабов, которые можно создавать на игровом поле  
 /// </summary>  
 private Dictionary<string, GameObject> prefabs = new Dictionary<string, GameObject>();  
 /// <summary>  
 /// Список префабов, которые можно создавать на игровом поле  
 /// </summary>  
 public List<GameObject> prefabsList = new List<GameObject>();  
  
  
  
  
 /// <summary>  
 /// Инициализирует переменные  
 /// </summary>  
 private void Awake() {  
 client = this;  
  
 sClient.Init();  
  
 if (spawnBorder != null) {  
 List<Vector3> points = new List<Vector3>();  
 for (int i = 0; i < spawnBorder.transform.childCount; i++) {  
 points.Add(spawnBorder.transform.GetChild(i).transform.position);  
 }  
  
 spawnPolygon = new TrianglePolygon(points);  
 }  
  
 foreach (var prefab in prefabsList) {  
 prefabs.Add(prefab.name, prefab);  
 }  
  
  
 var c = new SpawnPrefabCommand("123123", Vector3.back, Quaternion.identity, 123, 4, 778);  
 var f = c.Serialize();  
 var d = SpawnPrefabCommand.Deserialize(f);  
  
 Debug.LogError("test");  
 Assembly.Load("Assembly-CSharp").GetType("AIController");  
 Debug.LogError(Type.GetType("AIController"));  
 Debug.Log("CLIENT starting");  
 }  
  
   
  
  
 /// <summary>  
 /// Удаляет объект с игрового поля  
 /// </summary>  
 /// <param name="gameObject">Объект</param>  
 public void RemoveObject(GameObject gameObject) {  
 ObjectID.RemoveObject(gameObject);  
 Destroy(gameObject);  
 }  
  
 /// <summary>  
 /// Создаёт объект на игровом поле  
 /// </summary>  
 /// <param name="command">Команда для создание объекта</param>  
 /// <returns>Созданный объект</returns>  
 public GameObject SpawnObject(SpawnPrefabCommand command)  
 {  
 if (!prefabs.ContainsKey(command.prefabName)) {  
 throw new ArgumentException($"not found prefab '{command.prefabName}' in Client.prefabs");  
 }  
 GameObject prefab = prefabs[command.prefabName];  
 var gameObject = Instantiate(prefab, command.position, command.rotation);  
 ObjectID.StoreObject(gameObject, command.id, command.owner, command.creator);  
 Debug.Log($"Spawned {gameObject}({gameObject.GetInstanceID()}). id: {command.id}");  
 return gameObject;  
 }  
  
 /// <summary>  
 /// Создаёт объект на игровом поле  
 /// </summary>  
 /// <param name="name">Название префаба</param>  
 /// <param name="position">Положение объекта</param>  
 /// <param name="rotation">Поворот объекта</param>  
 /// <returns>Созданный объект</returns>  
 public GameObject SpawnPrefab(string name, Vector3 position = new Vector3(),  
 Quaternion rotation = new Quaternion()) {  
 return Instantiate(prefabs[name], position, rotation);  
 }  
}

## ObjectId.cs

﻿using System;  
using System.Collections.Generic;  
using UnityEngine;  
using UnityEngine.Assertions;  
  
/// <summary>  
/// Класс для хранения объектов по ID  
/// </summary>  
public class ObjectID: MonoBehaviour  
{  
 /// <summary>  
 /// Класс для хранения объекта с данными  
 /// </summary>  
 class ObjectData {  
 /// <summary>  
 /// Переменная для хранения ссылки на объект  
 /// </summary>  
 private WeakReference<GameObject> weakReference;  
  
 /// <summary>  
 /// Ссылка на объект  
 /// </summary>  
 public GameObject GameObject {  
 get {  
 GameObject res;  
 if (!weakReference.TryGetTarget(out res)) return null;  
 return res;  
 }  
 }  
  
 /// <summary>  
 /// Владелец объекта  
 /// </summary>  
 public int owner;  
 /// <summary>  
 /// Игрок, создавший объект  
 /// </summary>  
 public int creator;  
  
 /// <summary>  
 /// Конструктор   
 /// </summary>  
 /// <param name="gameObject">Объект</param>  
 /// <param name="owner">Владелец</param>  
 /// <param name="creator">Создатель</param>  
 public ObjectData(GameObject gameObject, int owner, int creator) {  
 this.weakReference = new WeakReference<GameObject>(gameObject);  
 this.owner = owner;  
 this.creator = creator;  
 }  
 }  
  
 /// <summary>  
 /// Слоаврь объектов по ID  
 /// </summary>  
 private static Dictionary<int, ObjectData> IDToObject = new Dictionary<int, ObjectData>();  
 /// <summary>  
 /// Словарь ID по UnityID  
 /// </summary>  
 private static Dictionary<int, int> UnityIDtoObjectID = new Dictionary<int, int>();  
  
 /// <summary>  
 /// Генератор случайныъ чисел  
 /// </summary>  
 private static System.Random random = new System.Random();  
 /// <summary>  
 /// Случаный ID  
 /// </summary>  
 public static int RandomID => random.Next();  
   
 /// <summary>  
 /// Сохраняет объект  
 /// </summary>  
 /// <param name="gameObject">Объект</param>  
 /// <param name="id">ID объекта</param>  
 /// <param name="owner">Владелец объекта</param>  
 /// <param name="creator">Создатель объекта</param>  
 public static void StoreObject(GameObject gameObject, int id, int owner, int creator) {  
 Assert.IsFalse(IDToObject.ContainsKey(id), $"ID#{id} already exists");  
 Assert.IsFalse(UnityIDtoObjectID.ContainsKey(gameObject.GetInstanceID()), $"InstanceID#{gameObject.GetInstanceID()} already exists");  
 IDToObject.Add(id, new ObjectData(gameObject, owner, creator));  
 UnityIDtoObjectID.Add(gameObject.GetInstanceID(), id);  
 }  
  
 /// <summary>  
 /// Сохраняет объект, созданный локально  
 /// </summary>  
 /// <param name="gameObject">Объект</param>  
 /// <param name="creator">Создатель</param>  
 public static void StoreOwnedObject(GameObject gameObject, int creator) {  
 StoreObject(gameObject, RandomID, sClient.ID, creator);  
 }  
  
 /// <summary>  
 /// Получает ID объекта  
 /// </summary>  
 /// <param name="gameObject">Объект</param>  
 /// <returns>ID объекта</returns>  
 public static int GetID(GameObject gameObject) {  
 int result;  
 if (!UnityIDtoObjectID.TryGetValue(gameObject.GetInstanceID(), out result)) {  
 throw new KeyNotFoundException($"Gameobject {gameObject.name}#{gameObject.GetInstanceID()} not found in ObjectID");  
 }  
 return result;  
 }  
  
 /// <summary>  
 /// Пытается получить ID объекта  
 /// </summary>  
 /// <param name="gameObject">Объект</param>  
 /// <param name="result">ID объекта</param>  
 /// <returns>true, если данный объект суещствует. иначе false </returns>  
 public static bool TryGetID(GameObject gameObject, out int result) {  
 return UnityIDtoObjectID.TryGetValue(gameObject.GetInstanceID(), out result);  
 }  
  
 /// <summary>  
 /// Пытается получить объект по ID  
 /// </summary>  
 /// <param name="id">ID</param>  
 /// <param name="gameObject">Объект</param>  
 /// <returns>true, если объект был найден. иначе false</returns>  
 public static bool TryGetObject(int id, out GameObject gameObject) {  
 ObjectData go;  
 var res = IDToObject.TryGetValue(id, out go);  
 if (!res) {  
 gameObject = null;  
 return false;  
 } else {  
 gameObject = go.GameObject;  
 }  
 return gameObject != null;  
 }  
   
 /// <summary>  
 /// Получает объект по ID  
 /// </summary>  
 /// <param name="id">ID</param>  
 /// <returns>Объект</returns>  
 public static GameObject GetObject(int id)  
 {  
 if (!IDToObject.ContainsKey(id))  
 {  
 Debug.LogWarning($"Cannot find object with id: {id}");  
 return null;  
 }  
 return IDToObject[id].GameObject;  
 }  
  
  
 /// <summary>  
 /// Удаляет объект  
 /// </summary>  
 /// <param name="gameObject">Объект</param>  
 public static void RemoveObject(GameObject gameObject) {  
 int id;  
 if (!TryGetID(gameObject, out id)) return;  
 Assert.IsTrue(IDToObject.Remove(id));  
 Assert.IsTrue(UnityIDtoObjectID.Remove(gameObject.GetInstanceID()));  
 }  
  
 /// <summary>  
 /// Выводит информацию об объектах в строку  
 /// </summary>  
 /// <returns>Строку с информацией об объектах</returns>  
 public new static string ToString() {  
 var text = "";  
 foreach (var id\_obj in IDToObject) {  
 var id = id\_obj.Key;  
 var pair = id\_obj.Value;  
 var go = pair.GameObject;  
 var unityid = go.GetInstanceID();  
 text += $"{go.name}#{unityid} -- {id} (owned {pair.owner})\n";  
 }  
  
 return text;  
 }  
  
 /// <summary>  
 /// Получает данные об объекте по ID  
 /// </summary>  
 /// <param name="id">id</param>  
 /// <returns>Данные об объекте</returns>  
 private static ObjectData GetObjectData(int id) {  
 if (!IDToObject.ContainsKey(id))  
 {  
 throw new KeyNotFoundException($"Gameobject {id} not found in ObjectID");  
 }  
 return IDToObject[id];  
 }  
   
  
 /// <summary>  
 /// Получает данные об объекте  
 /// </summary>  
 /// <param name="gameObject">Объект</param>  
 /// <returns>Данные об объекте</returns>  
 private static ObjectData GetObjectData(GameObject gameObject) {  
 return GetObjectData(GetID(gameObject));  
 }  
   
 /// <summary>  
 /// Получает владельца объекта  
 /// </summary>  
 /// <param name="id">ID объекта</param>  
 /// <returns>Владельца объекта</returns>  
 public static int GetOwner(int id) {  
 return GetObjectData(id).owner;  
 }  
   
 /// <summary>  
 /// Получает владельца объекта  
 /// </summary>  
 /// <param name="gameObject">Объект</param>  
 /// <returns>Владельца объекта</returns>  
 public static int GetOwner(GameObject gameObject) {  
 return GetObjectData(gameObject).owner;  
 }  
   
 /// <summary>  
 /// Пытается получить владельца объекта  
 /// </summary>  
 /// <param name="id">ID объекта</param>  
 /// <param name="owner">Владелец</param>  
 /// <returns>true, если объект был найден. Иначе false</returns>  
 public static bool TryGetOwner(int id, out int owner) {  
 ObjectData res;  
 if (!IDToObject.TryGetValue(id, out res) || res == null) {  
 owner = 0;  
 return false;  
 }  
  
 owner = res.owner;  
 return true;  
 }  
   
 /// <summary>  
 /// Проверяет, владеет ли данный клиент объектом  
 /// </summary>  
 /// <param name="id">ID объекта</param>  
 /// <returns>true, если владеет. Иначе false</returns>  
 public static bool IsOwned(int id) {  
 return GetOwner(id) == sClient.ID;  
 }  
   
 /// <summary>  
 /// Проверяет, владеет ли данный клиент объектом  
 /// </summary>  
 /// <param name="go">Объект</param>  
 /// <returns>true, если владеет. Иначе false</returns>  
 public static bool IsOwned(GameObject go) {  
 return GetOwner(go) == sClient.ID;  
 }  
   
 /// <summary>  
 /// Изменяет владельца объекта  
 /// </summary>  
 /// <param name="id">ID объекта</param>  
 /// <param name="owner">Новый владелец</param>  
 public static void SetOwner(int id, int owner) {  
 IDToObject[id].owner = owner;  
 }  
   
   
 /// <summary>  
 /// Получает создателя объекта  
 /// </summary>  
 /// <param name="id">ID объекта</param>  
 /// <returns>ID создателя</returns>  
 public static int GetCreator(int id) {  
 return GetObjectData(id).creator;  
 }  
   
 /// <summary>  
 /// Получает создателя объекта  
 /// </summary>  
 /// <param name="gameObject">Объект</param>  
 /// <returns>ID создателя</returns>  
 public static int GetCreator(GameObject gameObject) {  
 return GetObjectData(gameObject).creator;  
 }  
   
 /// <summary>  
 /// Пытается получить создателя объекта  
 /// </summary>  
 /// <param name="id">ID объекта</param>  
 /// <param name="creator">Создатель объекта</param>  
 /// <returns>true, если объект был найден. Иначе false</returns>  
 public static bool TryGetCreator(int id, out int creator) {  
 ObjectData res;  
 if (!IDToObject.TryGetValue(id, out res) || res == null) {  
 creator = 0;  
 return false;  
 }  
 creator = res.creator;  
 return true;  
 }  
   
   
   
 /// <summary>  
 /// Очищает значения переменных  
 /// </summary>   
 public static void Clear() {  
 IDToObject.Clear();  
 UnityIDtoObjectID.Clear();  
 }  
}

## sClient.cs

﻿using System;  
using System.Collections;  
using System.Collections.Generic;  
using CommandsSystem;  
using CommandsSystem.Commands;  
using Events;  
using Game;  
using GameMode;  
using JsonRequest;  
using Networking;  
using UnityEngine;  
using UnityEngine.SceneManagement;  
using Util2;  
  
/// <summary>  
/// Главный класс, управляющий игрой, выбором матча  
/// </summary>  
public class sClient : MonoBehaviour {  
 /// <summary>  
 /// Состояние  
 /// </summary>  
 public enum STATE {  
 START\_SCREEN,  
 FIND\_MATCH,  
 IN\_GAME  
 }  
  
 /// <summary>  
 /// Набирает ли пользователь сейчас текст  
 /// </summary>  
 public static bool isTyping = false;  
 /// <summary>  
 /// Количество сообщений в секунду, отправляемых по сети для синхронизации  
 /// </summary>  
 public const int NETWORK\_FPS = 20;  
  
 /// <summary>  
 /// ID клиента  
 /// </summary>  
 public static int ID => InstanceManager.ID;  
 /// <summary>  
 /// Генератор случайных чисел  
 /// </summary>  
 public static System.Random random = new System.Random();  
  
 /// <summary>  
 /// Время, когда началась игра  
 /// </summary>  
 private static float gameStartTime;  
  
 /// <summary>  
 /// Текущее состояния  
 /// </summary>  
 public static STATE state = STATE.START\_SCREEN;  
   
 /// <summary>  
 /// Переключает состояние на начало игры  
 /// </summary>  
 public static void SetGameStarted() {  
 if (state != STATE.FIND\_MATCH)   
 Debug.LogError("Called SetGameStarted but sClient state is " + state);  
 gameStartTime = Time.time;  
 state = STATE.IN\_GAME;  
 MatchesManager.SetMatchIsPlaying();  
 }  
  
 /// <summary>  
 /// Время, прошедшее с начала игры  
 /// </summary>  
 public static float GameTime => Time.time - gameStartTime;  
  
 /// <summary>  
 /// Был ли инициализирован этот класс  
 /// </summary>  
 private static bool initialized = false;  
  
 /// <summary>  
 /// Инициализирует переменные  
 /// </summary>  
 public static void Init() {  
 if (initialized) return;  
 initialized = true;  
 InstanceManager.Init();  
 PlayersManager.mainPlayer = new Player(sClient.ID, sClient.ID, 0);  
 }  
  
 /// <summary>  
 /// Сбрасывает значения переменных  
 /// </summary>  
 public static void Reset() {  
   
 CommandsHandler.Reset();  
 MatchesManager.Reset();  
 PlayersManager.Reset();  
 InstanceManager.Reset();  
 GameManager.Reset();  
 ObjectID.Clear();  
 state = STATE.START\_SCREEN;  
 if (AutoMatchJoiner.isRunning) {  
 sClient.LoadScene("empty\_scene");  
 } else {  
 sClient.LoadScene("start\_scene");  
 }  
 }  
  
 /// <summary>  
 /// Переключает состояние в поиск матча  
 /// </summary>  
 public static void StartFindingMatch() {  
 state = STATE.FIND\_MATCH;  
 }  
   
 /// <summary>  
 /// Инициалилизирует переменные при запуске игры  
 /// </summary>  
 private void Awake() {  
 Init();  
 }  
   
  
 /// <summary>  
 /// Обновляет состояние  
 /// </summary>  
 void Update() {  
 if (isTyping)  
 Input.ResetInputAxes();  
  
 CommandsHandler.Update();  
 RequestsManager.Update();  
 switch (state) {  
 case STATE.START\_SCREEN:  
 break;  
 case STATE.FIND\_MATCH:  
 MatchesManager.Update();  
 break;  
 case STATE.IN\_GAME:  
 GameManager.Update();  
 break;  
 }  
 }  
   
 /// <summary>  
 /// Обрабатывает выход из приложения  
 /// </summary>  
 private void OnApplicationQuit() {  
 CommandsHandler.Stop();  
 }  
   
 /// <summary>  
 /// Загружает сцену  
 /// </summary>  
 /// <param name="sceneName"></param>  
 public static void LoadScene(string sceneName) {  
 UberDebug.LogChannel("Client", "Loading scene " + sceneName);  
 SceneManager.LoadScene(sceneName);  
 }  
}

## Triangle.cs

﻿using System;  
using System.Collections.Generic;  
using UnityEngine;  
using Random = UnityEngine.Random;  
  
/// <summary>  
/// Класс для треугольника  
/// </summary>  
public class Triangle {  
 /// <summary>  
 /// Координаты треугольника  
 /// </summary>  
 public Vector3 a, b, c;  
  
 /// <summary>  
 /// Конструктор треугольника  
 /// </summary>  
 public Triangle() { }  
 /// <summary>  
 /// Конструктор треугольника  
 /// </summary>  
 /// <param name="a">Первая точка</param>  
 /// <param name="b">Вторая точка</param>  
 /// <param name="c">Третья точка</param>  
 public Triangle(Vector3 a, Vector3 b, Vector3 c) {  
 this.a = a;  
 this.b = b;  
 this.c = c;  
 }  
   
 /// <summary>  
 /// Получает случайную точку в треугольнике  
 /// </summary>  
 /// <returns>Случаная точка</returns>  
 public Vector3 RandomPoint() {  
 float r1 = Random.value;  
 float r2 = Random.value;  
 return (1 - Mathf.Sqrt(r1)) \* a + (Mathf.Sqrt(r1) \* (1 - r2)) \* b + (Mathf.Sqrt(r1) \* r2) \* c;  
 }  
   
 /// <summary>  
 /// Вычисляет площадь треугольника  
 /// </summary>  
 public float Area => Vector3.Cross(a - b, a - c).magnitude / 2;  
}  
  
/// <summary>  
/// Класс, представляющий выпуклый многоугольник, составленный из треугольников  
/// </summary>  
public class TrianglePolygon {  
 /// <summary>  
 /// Список треугольников  
 /// </summary>  
 private List<Triangle> triangles;  
 /// <summary>  
 /// Суммарная площадь всех треугольников  
 /// </summary>  
 private float areaSum;  
 /// <summary>  
 /// Конструктор  
 /// </summary>  
 /// <param name="points">Список точек</param>  
 public TrianglePolygon(List<Vector3> points) {  
 triangles = new List<Triangle>();  
 for (int i = 2; i < points.Count; i++) {  
 triangles.Add(new Triangle(points[0], points[i - 1], points[i]));  
 }  
  
 areaSum = 0;  
 foreach (var triangle in triangles) {  
 areaSum += triangle.Area;  
 }  
 }  
  
 /// <summary>  
 /// Находит случайную точку внутри многоугольника  
 /// </summary>  
 /// <returns>Случайную точку</returns>  
 public Vector3 RandomPoint() {  
 float r = Random.Range(0, areaSum);  
 int i = 0;  
  
 while (r >= triangles[i].Area)  
 {  
 r -= triangles[i].Area;  
 i++;  
 }  
  
 return triangles[i].RandomPoint();  
 }  
}

## ActionController.cs

﻿using System;  
using Character.Actions;  
using Character.Guns;  
using UnityEngine;  
  
namespace Character {  
  
  
   
 /// <summary>  
 /// Компонента для обработки действий, которые может делать игрок  
 /// </summary>  
 [RequireComponent(typeof(CharacterAnimator))]  
 [RequireComponent(typeof(PushAction))]  
 [RequireComponent(typeof(ShootPistolAction))]  
 [RequireComponent(typeof(ShootSemiautoAction))]  
 public class ActionController : MonoBehaviour {  
   
 /// <summary>  
 /// Ссылка на CharacterAnimator игрока  
 /// </summary>  
 private CharacterAnimator animator;  
   
 /// <summary>  
 /// Инициализирует переменны  
 /// </summary>  
 void Start() {  
 animator = GetComponent<CharacterAnimator>();  
 }  
  
 /// <summary>  
 /// Текущее действие  
 /// </summary>  
 private IAction currentAction = null;  
 /// <summary>  
 /// Координата, в которую направлен прицел игрока  
 /// </summary>  
 public Vector3 Target;  
   
 /// <summary>  
 /// Прекращает выполнение текущего действия  
 /// </summary>  
 private void StopCurrent() {  
 if (currentAction != null) {  
 DoAction = false;  
 (currentAction as MonoBehaviour).enabled = false;  
 }  
 }  
  
 /// <summary>  
 /// Изменяет действие персонажа  
 /// </summary>  
 /// <param name="setup">Функция для инициализации нового действия</param>  
 /// <typeparam name="T">Тип нового действия, должен быть MonoBehaivour и IAction</typeparam>  
 public void SetAction<T>(System.Action<T> setup)  
 where T: MonoBehaviour, IAction {  
 StopCurrent();  
 var action = gameObject.GetComponent<T>();  
  
 setup(action);  
   
 action.enabled = true;  
 currentAction = action;  
 }  
   
  
 /// <summary>  
 /// Изменяет действие перонажа на пустое  
 /// </summary>  
 public void SetNothing() {  
 StopCurrent();  
 currentAction = null;  
 }  
  
 /// <summary>  
 /// Переменная для хранения DoAction  
 /// </summary>  
 private bool \_actionDoing = false;  
   
 /// <summary>  
 /// Автоматическая переменная, устанавливающая выполняет ли сейчас действие игрок  
 /// </summary>  
 public bool DoAction {  
 get => \_actionDoing;  
 set {  
 if (value == \_actionDoing) return;  
 \_actionDoing = value;  
 if (value) {  
 currentAction?.OnStartDoing();  
 } else {  
 currentAction?.OnStopDoing();  
 }  
 }  
 }  
 }  
}

## AIController.cs

﻿using UnityEngine;  
using CharacterController = Character.CharacterController;  
  
/// <summary>  
/// Класс для игрока, управлемого искуственным интеллектом  
/// </summary>  
public class AIController : CharacterController {  
  
 /// <summary>  
 /// Изменяет параметры игрока на основе ИИ. Автоматически вызывается Unity каждый кадр  
 /// </summary>  
 void Update()  
 {  
 if (false && Random.value < 0.01f) {  
 var dir = new Vector3(Random.value \* 2 - 1, 0, Random.value \* 2 - 1);  
 motionController.TargetDirection = dir;  
 var rot = Random.rotation \* Vector3.forward;  
 rot.y = 0;  
 motionController.TargetRotation = rot;  
  
 actionController.DoAction = Random.value < 0.2f;  
 }  
 }  
}

## CameraFollower.cs

﻿using UnityEngine;  
  
namespace Character {  
 /// <summary>  
 /// Компонента для следования камеры за игроком  
 /// </summary>  
 public class CameraFollower : MonoBehaviour {  
 /// <summary>  
 /// Игрок, за которым должна следовать камера  
 /// </summary>  
 public GameObject character;  
  
 /// <summary>  
 /// Высота относительно игрока, на которой должна располагаться камера  
 /// </summary>  
 public float yLevel = 0;  
  
 /// <summary>  
 /// Перемещает камеру в позицию над игроком. Автоматически вызывается Unity каждый кадр  
 /// </summary>  
 void LateUpdate() {  
 if (character == null) return;  
 var position = character.transform.position;  
 var vec = new Vector3(position.x, yLevel, position.z);  
 transform.position = vec;  
 }  
 }  
}

## CharacterAnimator.cs

﻿using UnityEngine;  
  
namespace Character {  
 /// <summary>  
 /// Класс для анимации персонажа  
 /// </summary>  
 [RequireComponent(typeof(Animator))]  
 [RequireComponent(typeof(HandAnimationBlender))]  
 public class CharacterAnimator : MonoBehaviour {  
 /// <summary>  
 /// Ссылка на Animator у персонажа  
 /// </summary>  
 private Animator animator;  
 /// <summary>  
 /// Ссылка на HandAnimationBlender у персонажа  
 /// </summary>  
 private HandAnimationBlender handAnimationBlender;  
  
  
 /// <summary>  
 /// Константа с хэшем от idle  
 /// </summary>  
 private static readonly int Idle = Animator.StringToHash("idle");  
 /// <summary>  
 /// Константа с хэшем от speed  
 /// </summary>  
 private static readonly int Speed = Animator.StringToHash("speed");  
 /// <summary>  
 /// Константа с хэшем от push  
 /// </summary>  
 private static readonly int Push = Animator.StringToHash("push");  
 /// <summary>  
 /// Константа с хэшем от rotationSpeed  
 /// </summary>  
 private static readonly int RotationSpeed = Animator.StringToHash("rotationSpeed");  
  
 /// <summary>  
 /// Переменная для хранения состояния анимации персонажа  
 /// </summary>  
 private PlayerAnimationState \_animationState = new PlayerAnimationState();  
  
 /// <summary>  
 /// Текущее состояние анимации персонажа  
 /// </summary>  
 public PlayerAnimationState animationState {  
 get => \_animationState;  
 set {  
 \_animationState = value;  
 SetIdle(value.idle);  
 SetSpeed(value.speed);  
 SetRotationSpeed(value.rotationSpeed);  
 }  
 }  
   
 /// <summary>  
 /// Инициализирует переменные  
 /// </summary>  
 void Start() {  
 animator = GetComponent<Animator>();  
 handAnimationBlender = GetComponent<HandAnimationBlender>();  
 }  
  
 /// <summary>  
 /// Управлет анимацией стояния  
 /// </summary>  
 /// <param name="idle">true, если сейчас должна работать анимация стояния. иначе -- false</param>  
 public void SetIdle(bool idle) {  
 animator.SetBool(Idle, idle);  
 \_animationState.idle = idle;  
 }  
   
 /// <summary>  
 /// Передает в аниматор скорость персонажа  
 /// </summary>  
 /// <param name="speed">Скорость персонажа</param>  
 public void SetSpeed(float speed) {  
 animator.SetFloat(Speed, speed);  
 \_animationState.speed = speed + 10;  
 }  
  
 /// <summary>  
 /// Показывает анимацию толкания  
 /// </summary>  
 public void SetPush() {  
 animator.SetBool(Push, true);  
 }  
  
 /// <summary>  
 /// Передает в аниматор скорость поворота игрока  
 /// </summary>  
 /// <param name="rotationSpeed"></param>  
 public void SetRotationSpeed(float rotationSpeed) {  
 animator.SetFloat(RotationSpeed, 0);  
 \_animationState.rotationSpeed = rotationSpeed;  
 }  
 }  
}

## CharacterController.cs

﻿using UnityEngine;  
  
namespace Character {  
 /// <summary>  
 /// Базовый класс для компоненты, управляющей персонажем  
 /// </summary>  
 public class CharacterController : MonoBehaviour {  
 /// <summary>  
 /// Объект, которым управляет данный контроллер  
 /// </summary>  
 public GameObject target;  
   
 /// <summary>  
 /// Ссылка на motionController у управляемого персонажа  
 /// </summary>  
 protected MotionController motionController;  
 /// <summary>  
 /// Ссылка на actionController у управляемого персонажа  
 /// </summary>  
 protected ActionController actionController;  
  
 /// <summary>  
 /// Инициализирует переменные  
 /// </summary>  
 protected virtual void Start() {  
 motionController = target.GetComponent<MotionController>();  
 actionController = target.GetComponent<ActionController>();  
 }  
 }  
}

## HandAnimationBlender.cs

﻿using UnityEngine;  
using UnityEngine.Assertions;  
  
  
 /// <summary>  
 /// Класс для анимации рук персонажа  
 /// </summary>  
 public class HandAnimationBlender : MonoBehaviour {  
 /// <summary>  
 /// Скорость переключения между анимациями  
 /// </summary>  
 public float blendSpeed = 1.5f;  
   
 /// <summary>  
 /// Началось ли производить переключение между анимациями  
 /// </summary>  
 private bool blendStart = false;  
 /// <summary>  
 /// Началось ли производить обратное переключение между анимациями  
 /// </summary>  
 private bool blendStop = false;  
 /// <summary>  
 /// Текущий коэффициент смешивания слоев анимации  
 /// </summary>  
 private float blendCoeff = 0.1f;  
 /// <summary>  
 /// Индекс слоя с анимациями рук  
 /// </summary>  
 private int layerIndex;  
   
 /// <summary>  
 /// Ссылка на Animator персонажа  
 /// </summary>  
 private Animator animator;  
   
 /// <summary>  
 /// Инициализирует переменные  
 /// </summary>  
 void Start() {  
 animator = GetComponent<Animator>();  
 layerIndex = animator.GetLayerIndex("hands");  
 Assert.AreNotEqual(layerIndex, -1);  
 }  
  
 /// <summary>  
 /// Обновляет анимацию рук (если нужно). Автоматически вызывается Unity каждый кадр  
 /// </summary>  
 void Update() {  
 if (blendStart) {  
   
 blendCoeff += blendSpeed \* Time.deltaTime;  
 if (blendCoeff >= 1) {  
 blendStart = false;  
 blendCoeff = 1;  
 }  
 }  
  
 if (blendStop) {  
 blendCoeff -= blendSpeed \* Time.deltaTime;  
 if (blendCoeff <= 0) {  
 blendStop = false;  
 blendCoeff = 0.1f;  
 }  
 }  
   
 animator.SetLayerWeight(layerIndex, blendCoeff);  
 }  
  
 /// <summary>  
 /// Начинает анимацию рук. Автоматически вызывается Unity при запуске анимации  
 /// </summary>  
 public void StartHandAnimation() {  
 blendStart = true;  
 blendStop = false;  
 animator.SetBool("push", false);  
 }  
  
 /// <summary>  
 /// Прекращает анимацию рук. Автоматически вызывается Unity по окончанию анимации  
 /// </summary>  
 public void StopHandAnimation() {  
 blendStart = false;  
 blendStop = true;  
 }  
 }  
}

## MotionController.cs

﻿using System.Collections.Generic;  
using Character.HP;  
using UnityEngine;  
  
namespace Character {  
   
 /// <summary>  
 /// Компонента для передвижения персонажа по игровому полю  
 /// </summary>  
 [RequireComponent(typeof(Rigidbody))]  
 [RequireComponent(typeof(CharacterAnimator))]  
 public class MotionController : MonoBehaviour {  
 /// <summary>  
 /// Ссылка на Rigidbody персонажа  
 /// </summary>  
 private new Rigidbody rigidbody;  
 /// <summary>  
 /// Ссылка на CapsuleCollider персонажа  
 /// </summary>  
 private CapsuleCollider capsuleCollider;  
 /// <summary>  
 /// Ссылка на CharacterAnimator персонажа  
 /// </summary>  
 private CharacterAnimator animator;  
   
 /// <summary>  
 /// Сила, которая изменяет скорость персонажа  
 /// </summary>  
 public float moveForce = 4000;  
 /// <summary>  
 /// Максимальная скорость персонажа  
 /// </summary>  
 public float speed = 6.0f;  
 /// <summary>  
 /// Скорость поворота персонажа  
 /// </summary>  
 public float rotationSpeed = 700.0f;  
   
  
 /// <summary>  
 /// Инициализирует переменные  
 /// </summary>  
 void Start() {  
 rigidbody = GetComponent<Rigidbody>();  
 capsuleCollider = GetComponent<CapsuleCollider>();  
 animator = GetComponent<CharacterAnimator>();  
 }  
  
 private List<GameObject> groundCollisions = new List<GameObject>();  
  
 /// <summary>  
 /// Провряет, что персонаж всё еще стоит на земле.  
 /// </summary>  
 /// <param name="gameObject">Объект, на котором персонаж прекратил стоять</param>  
 private void DeGround(GameObject gameObject) {  
 if (!groundCollisions.Remove(gameObject)) return;  
 isGrounded = groundCollisions.Count != 0;  
 }  
  
 /// <summary>  
 /// Сообщает, что персонаж всё ещё стоит на данном коллайдере. Автоматически вызывается Unity  
 /// </summary>  
 /// <param name="other">Коллайдер</param>  
 private void OnTriggerStay(Collider other) {  
 if (!groundCollisions.Contains(other.gameObject)) {  
 groundCollisions.Add(other.gameObject);  
 isGrounded = true;  
 }  
 }  
  
 /// <summary>  
 /// Сообщает, что персонаж больше не стоит на данном коллайдере . Автоматически вызывается Unity  
 /// </summary>  
 /// <param name="other">Коллайдер</param>  
 private void OnTriggerExit(Collider other) {  
 DeGround(other.gameObject);  
 }  
   
 /// <summary>  
 /// Находится ли персонаж на земле  
 /// </summary>  
 private bool isGrounded = true;  
   
 /// <summary>  
 /// Направление, в котором должен двигаться персонаж  
 /// </summary>  
 public Vector3 TargetDirection { get; set; }  
 /// <summary>  
 /// Направление, в котором смотрит персонаж  
 /// </summary>  
 public Vector3 TargetRotation { get; set; }  
  
   
 /// <summary>  
 /// Производит передвижение персонажа. Автоматически вызывается Unity при каждой обработке физики  
 /// </summary>  
 void FixedUpdate() {  
 for (int i = 0; i < groundCollisions.Count; i++) {  
 if (!groundCollisions[i]) {  
 DeGround(groundCollisions[i]);  
 break;  
 }  
 }  
  
 if (transform.position.y < -15) {  
 gameObject.GetComponent<HPController>().TakeDamage(100000, DamageSource.InstaKill(), true);  
 }  
  
 if (isGrounded) {  
 var targetSpeed = speed \* TargetDirection;  
 var bodySpeed = rigidbody.velocity;  
  
 var vec = bodySpeed - targetSpeed;  
 vec.y = 0;  
 if (vec.magnitude > 1) {  
 vec.Normalize();  
 if (rigidbody.velocity.sqrMagnitude > 5f) {// && //Mathf.Abs(rigidbody.velocity.sqrMagnitude - speed \* speed) > 0.5f)  
 rigidbody.velocity = targetSpeed;   
 } else {  
 rigidbody.AddForce(-vec \* moveForce);  
 Debug.Log("addforce");   
 }  
 } else {  
 animator.SetRotationSpeed(0);  
 }  
 }  
  
 if (TargetDirection != Vector3.zero)  
 transform.rotation = Quaternion.RotateTowards(transform.rotation,  
 Quaternion.LookRotation(TargetDirection),  
 rotationSpeed \* Time.deltaTime);  
  
  
 float linearSpeed = TargetDirection.magnitude;  
 animator.SetIdle(linearSpeed == 0.0f);  
 animator.SetSpeed(linearSpeed);  
 }  
  
 /// <summary>  
 /// Отрисовывает отладочную информацию о состоянии игрока. Автоматически вызывается средой Unity  
 /// </summary>  
 private void OnDrawGizmos() {  
  
 if (!Application.isPlaying) return;  
  
 var pos = transform.position + capsuleCollider.center;  
 DebugExtension.DebugCircle(pos,  
 Color.green, radius: 0.1f, depthTest: false);  
  
 DebugExtension.DebugCircle(transform.position + Vector3.up \* 2,  
 isGrounded ? Color.red : Color.blue, radius: 0.1f);  
   
 pos.y += 0.5f;  
  
 DebugExtension.DebugArrow(pos, TargetDirection \* 1f, Color.blue);  
 DebugExtension.DebugArrow(pos, TargetRotation.normalized / 2, Color.red);  
 }  
 }  
}

## PlayerAnimationState.cs

﻿namespace Character {  
 /// <summary>  
 /// Структура, хранящая состояние анимации персонажа  
 /// </summary>  
 public struct PlayerAnimationState {  
 /// <summary>  
 /// Должна ли сейчас быть анимация стояния на месте  
 /// </summary>  
 public bool idle;  
 /// <summary>  
 /// Скорость персонажа  
 /// </summary>  
 public float speed;  
 /// <summary>  
 /// Скорость поворота персонажа  
 /// </summary>  
 public float rotationSpeed;  
 }  
}

## PlayerController.cs

﻿using System;  
using UnityEngine;  
using CharacterController = Character.CharacterController;  
  
/// <summary>  
/// Класс с дополнительными функциями для Vector2  
/// </summary>  
public static class Vector2Extension {  
 /// <summary>  
 /// Поворачиает вектор  
 /// </summary>  
 /// <param name="v">Вектор</param>  
 /// <param name="degrees">Угол в градусах</param>  
 /// <returns>Повернутый вектор</returns>  
 public static Vector2 Rotate(this Vector2 v, float degrees) {  
 float radians = degrees \* Mathf.Deg2Rad;  
 float sin = Mathf.Sin(radians);  
 float cos = Mathf.Cos(radians);  
   
 float tx = v.x;  
 float ty = v.y;  
   
 return new Vector2(cos \* tx - sin \* ty, sin \* tx + cos \* ty);  
 }  
}  
  
/// <summary>  
/// Компонента для персонажа, управляемого человеком  
/// </summary>  
public class PlayerController : CharacterController {  
 /// <summary>  
 /// Переменная для внутреннего использования (нужна для уменьшения нагрузки на сборщик мусора)  
 /// </summary>  
 private Plane plane;  
  
 /// <summary>  
 /// Инициализирует переменные  
 /// </summary>  
 protected override void Start()  
 {  
 base.Start();  
 plane = new Plane(Vector3.up, 0);  
 transform.parent = null;  
 }  
  
 /// <summary>  
 /// Управляет персонажем в соответвии с положением мышки и нажатыми клавишами. Автоматически вызывается Unity каждый кадр  
 /// </summary>  
 void Update() {  
 if (sClient.isTyping)  
 Input.ResetInputAxes();  
  
 Vector2 vec = new Vector2(Input.GetAxis("Horizontal"), Input.GetAxis("Vertical"));  
 var len = 1;  
   
 var vec3 = new Vector3(vec.x, 0, vec.y);  
 vec3 = Camera.main.transform.rotation \* vec3;  
 vec3.y = 0;  
 vec3 = vec3.normalized \* len;  
   
 motionController.TargetDirection = vec3;  
  
 var ray = Camera.main.ScreenPointToRay(Input.mousePosition);  
 var target\_pos = target.transform.position + Vector3.up \* 1.5f;  
 plane.SetNormalAndPosition(Vector3.up, target\_pos);  
  
 float distance;  
 plane.Raycast(ray, out distance);  
 var pos = ray.GetPoint(distance);  
 motionController.TargetRotation = pos - target\_pos;  
   
   
 plane.SetNormalAndPosition(Vector3.up, target.transform.position);  
 plane.Raycast(ray, out distance);  
 actionController.Target = ray.GetPoint(distance);  
  
 actionController.DoAction = Input.GetMouseButton(0);  
 }  
}

## IAction.cs

﻿namespace Character.Actions {  
 /// <summary>  
 /// Интерфейс действия пользователя  
 /// </summary>  
 public interface IAction {  
 /// <summary>  
 /// Функция, которая вызывается, когда игрок начинает делать действие  
 /// </summary>  
 void OnStartDoing();  
   
 /// <summary>  
 /// Функция, которая вызывается, когда игрок прекращает делать действие  
 /// </summary>  
 void OnStopDoing();  
 }  
}

## PushAction.cs

﻿using CommandsSystem.Commands;  
using Networking;  
using RotaryHeart.Lib.PhysicsExtension;  
using UnityEngine;  
  
namespace Character.Actions {  
 /// <summary>  
 /// Действие персонажа, при котором он толкает предметы перед собой с определенной силой  
 /// </summary>  
 [RequireComponent(typeof(CharacterAnimator))]  
 public class PushAction : MonoBehaviour, IAction {  
 /// <summary>  
 /// Объект, внутри котрого нужно толькать предметы  
 /// </summary>  
 public GameObject pushCollider;  
 /// <summary>  
 /// Сила с которой нужно толкать предметы  
 /// </summary>  
 public float force = 3300;  
   
 /// <summary>  
 /// Ссылка на CharacterAnimator у игрока  
 /// </summary>  
 private CharacterAnimator animator;  
  
 /// <summary>  
 /// Инициализирует переменные  
 /// </summary>  
 void Start() {  
 animator = gameObject.GetComponent<CharacterAnimator>();  
 }  
  
 /// <summary>  
 /// Начинает выполнять данное действие  
 /// </summary>  
 public void OnStartDoing() {  
 animator.SetPush();  
 CommandsHandler.gameRoom.RunSimpleCommand(new PlayerPushCommand(ObjectID.GetID(gameObject)), MessageFlags.NONE);  
 }  
  
 /// <summary>  
 /// Толкает предметы перед собой. Автоматически вызывается Unity в конце анимации рук персонажа  
 /// </summary>  
 public void pushEnd() {  
 var center = pushCollider.transform.position;  
 var scale = pushCollider.transform.localScale;  
 var rotation = pushCollider.transform.rotation;  
  
 var delta = rotation \* Vector3.up \* scale.y;  
 //delta.Scale(scale);  
   
 var radius = scale.x \* transform.lossyScale.x / 2;  
  
 delta -= rotation \* (radius \* Vector3.up);  
   
 var start = center - delta;  
 var stop = center + delta;  
   
 // DebugExtension.DebugCapsule(start, stop, Color.red, radius, 1);  
  
 var f =   
 RotaryHeart.Lib.PhysicsExtension.Physics.OverlapCapsule(start, stop, radius, PreviewCondition.Both, drawDuration: 1);  
  
 var force = rotation \* Vector3.up \* this.force;  
 foreach (var v in f) {  
 if (v.gameObject == gameObject) continue;  
   
 if (v.gameObject.CompareTag("Unmanagable")) {  
 var command = new ApplyForceCommand(v.gameObject, force);  
 CommandsHandler.gameRoom.RunSimpleCommand(command, MessageFlags.NONE);  
 } else {  
 var rig = v.gameObject.GetComponent<Rigidbody>();  
 if (rig != null) {  
 rig.AddForce(force, ForceMode.Impulse);  
 }  
 }  
 }  
 }  
  
 /// <summary>  
 /// Заканчивает выполнять данное действие  
 /// </summary>  
 public void OnStopDoing() { }   
 }  
}

## ShootAction.cs

﻿using Character.Guns;  
using Events;  
using GameMode;  
using UnityEngine;  
  
namespace Character.Actions {  
 /// <summary>  
 /// Действие персонажа, при котором он стреляет по врагам  
 /// </summary>  
 /// <typeparam name="T">Оружие, из которго игрок стреляет</typeparam>  
 public abstract class ShootAction<T> : MonoBehaviour, IAction   
 where T: IGun {  
   
 /// <summary>  
 /// Ссылка на CharacterAnimator у игрока  
 /// </summary>  
 private CharacterAnimator animator;  
  
 /// <summary>  
 /// Оружие, из которого игрок стреляет  
 /// </summary>  
 public T gun;  
  
 /// <summary>  
 /// Обрабатывает подбор оружия игроком. Вызывается при включении скрипта  
 /// </summary>  
 void OnEnable() {  
 if (gun != null) {  
 gun.OnPickedUp(gameObject);  
 EventsManager.handler.OnPlayerPickedUpGun(gameObject, gun);  
 }  
 }  
  
 /// <summary>  
 /// Обрабатывает потерю оружия игроком. Вызывается при выключении скрипта  
 /// </summary>  
 private void OnDisable() {  
 if (GameManager.sceneReloaded) return;  
 if (gun != null) {  
 gun.OnDropped();  
 EventsManager.handler.OnPlayerDroppedGun(gameObject, gun);  
 }  
 }  
   
 /// <summary>  
 /// Инициализирует переменные  
 /// </summary>  
 void Start() {  
 animator = gameObject.GetComponent<CharacterAnimator>();  
 }  
  
 /// <summary>  
 /// Функция, которая вызывается, когда игрок начинает делать действие  
 /// </summary>  
 public abstract void OnStartDoing();  
  
 /// <summary>  
 /// Функция, которая вызывается, когда игрок прекращает делать действие  
 /// </summary>  
 public abstract void OnStopDoing();  
 }  
}

## ShootPistolAction.cs

﻿using Character.Guns;  
using UnityEngine;  
  
namespace Character.Actions {  
 /// <summary>  
 /// Действие для стрельбы из пистолета (делает один выстрел за клик)  
 /// </summary>  
 public class ShootPistolAction : ShootAction<ReloadingGun> {  
 /// <summary>  
 /// Время, когда была отдана последняя команда стрелять.  
 /// Нужно для возможности отдать команду стрелять, если до перезарядки орудия осталось меньше 150 мсек  
 /// </summary>  
 private float needShoot = -100;  
   
 /// <summary>  
 /// Отдает команду стрелять  
 /// </summary>  
 public override void OnStartDoing() {  
 needShoot = Time.time;  
 }  
   
 /// <summary>  
 /// Вызывается при прекращении действия  
 /// </summary>  
 public override void OnStopDoing() {}  
  
 /// <summary>  
 /// Обновляет состояние оружия. Если была отдана команда стрелять и оружие заряжено -- стреляет  
 /// </summary>  
 void LateUpdate() {  
 gun.Update(Time.deltaTime);  
 if (Time.time - needShoot < 0.15f && gun.state == GunState.READY) {  
 gun.Shoot();  
 needShoot = -100;  
 }  
 }  
 }  
}

## ShootSemiautoAction.cs

﻿using Character.Guns;  
using UnityEngine;  
  
namespace Character.Actions {  
 /// <summary>  
 /// Действие для стрельбы из автомата (непрерывно производит выстрелы, пока действие выполняется)  
 /// </summary>  
 public class ShootSemiautoAction : ShootAction<ReloadingGun> {  
 /// <summary>  
 /// Переенная показывающая, нужно ли производить выстрел  
 /// </summary>  
 private bool needShoot = false;  
   
 /// <summary>  
 /// Начинает стрельбу из оружия  
 /// </summary>  
 public override void OnStartDoing() {  
 needShoot = true;  
 }  
   
 /// <summary>  
 /// Прекращает стрельбу из оружия  
 /// </summary>  
 public override void OnStopDoing() {  
 needShoot = false;  
 }  
  
 /// <summary>  
 /// Обновляет состояние оружия. Если была отдана команда стрелять и оружие заряжено -- стреляет  
 /// </summary>  
 void LateUpdate() {  
 gun.Update(Time.deltaTime);  
 if (needShoot && gun.state == GunState.READY) {  
 gun.Shoot();  
 }  
 }  
 }  
}

## BombGun.cs

﻿using System;  
using Character.Guns;  
using CommandsSystem.Commands;  
using UnityEngine;  
  
namespace Character.Guns {  
 /// <summary>  
 /// Класс для гранотомета  
 /// </summary>  
 [Serializable]  
 public partial class BombGun : ReloadingGun {  
 /// <summary>  
 /// Время перезарядки магазина  
 /// </summary>  
 public float \_fullReloadTime = 10.0f;  
 /// <summary>  
 /// Время перезарядки между выстрелами  
 /// </summary>  
 public float \_reloadTime = 0.3f;  
 /// <summary>  
 /// Количество выстрелов в одном боекомплекте  
 /// </summary>  
 public int \_bulletsInMagazine = 10;  
  
   
 /// <summary>  
 /// Возвращает время перезарядки магазина  
 /// </summary>  
 /// <returns>Время перезарядки магазина</returns>  
 public override float GetFullReloadTime() => \_fullReloadTime;  
 /// <summary>  
 /// Возвращает время перезарядки между выстрелами  
 /// </summary>  
 /// <returns>Время перезарядки между выстрелами</returns>  
 public override float GetReloadTime() => \_reloadTime;  
 /// <summary>  
 /// Возвращает количество выстрелов в одном боекомплекте  
 /// </summary>  
 /// <returns>Количество выстрелов в одном боекомплекте</returns>  
 public override int GetBulletsInMagazine() => \_bulletsInMagazine;  
  
 // переменные для корректной работы сериализации  
 /// public int \_bulletsCount;  
 /// public int \_magazinesCount;  
 /// public Vector3 position;  
 /// public int id;  
 /// public int \_state;  
  
   
 /// <summary>  
 /// Производит выстрел из гранатомета  
 /// </summary>  
 protected override void DoShoot() {  
 ShootSystem.ShootWithBomb(player.gameObject, player.GetComponent<ActionController>().Target, "bomb");  
 }  
  
 /// <summary>  
 /// Создает гранотомет на игровом поле  
 /// </summary>  
 public void Run() {  
 var go = Client.client.SpawnObject(new SpawnPrefabCommand("bombgun", position, Quaternion.identity, id, 0, 0));  
 go.GetComponent<BombGunController>().gun = this;  
 }  
 }  
}

## GunState.cs

﻿namespace Character.Guns {  
 /// <summary>  
 /// Перечесление с возможными состояниями оружия  
 /// </summary>  
 public enum GunState {  
 READY,  
 RELOADING\_BULLET,  
 RELOADING\_MAGAZINE  
 }  
}

## IGun.cs

﻿using UnityEngine;  
  
namespace Character.Guns {  
 /// <summary>  
 /// Интерфейс оружия  
 /// </summary>  
 public interface IGun {  
 /// <summary>  
 /// Состояние оружия  
 /// </summary>  
 GunState state { get; }  
 /// <summary>  
 /// Обработчик. Вызывается при подборе игроком оружия  
 /// </summary>  
 /// <param name="player">Игрок, который подобрал оружие</param>  
 void OnPickedUp(GameObject player);  
 /// <summary>  
 /// Обработчик. Вызывается при потере игроком оружия  
 /// </summary>  
 void OnDropped();  
 }  
}

## Pistol.cs

﻿  
using System;  
using CommandsSystem.Commands;  
using UnityEngine;  
  
namespace Character.Guns {  
   
 /// <summary>  
 /// Класс для пистолета  
 /// </summary>  
 [Serializable]  
 public partial class Pistol : ReloadingGun {  
 /// <summary>  
 /// Время перезарядки магазина  
 /// </summary>  
 public float \_fullReloadTime = 4.0f;  
 /// <summary>  
 /// Время перезарядки между выстрелами  
 /// </summary>  
 public float \_reloadTime = 0.3f;  
 /// <summary>  
 /// Количество выстрелов в одном боекомплекте  
 /// </summary>  
 public int \_bulletsInMagazine = 5;  
  
 /// <summary>  
 /// Урон от выстрела  
 /// </summary>  
 public float damage = 25;  
   
 /// <summary>  
 /// Возвращает время перезарядки магазина  
 /// </summary>  
 /// <returns>Время перезарядки магазина</returns>  
 public override float GetFullReloadTime() => \_fullReloadTime;  
 /// <summary>  
 /// Возвращает время перезарядки между выстрелами  
 /// </summary>  
 /// <returns>Время перезарядки между выстрелами</returns>  
 public override float GetReloadTime() => \_reloadTime;  
 /// <summary>  
 /// Возвращает количество выстрелов в одном боекомплекте  
 /// </summary>  
 /// <returns>Количество выстрелов в одном боекомплекте</returns>  
 public override int GetBulletsInMagazine() => \_bulletsInMagazine;  
   
 // переменные для корректной работы сериализации  
 /// public int \_bulletsCount;  
 /// public int \_magazinesCount;  
 /// public Vector3 position;  
 /// public int id;  
 /// public int \_state;  
  
 /// <summary>  
 /// Производит выстрел из пистолета  
 /// </summary>  
 protected override void DoShoot() {  
 ShootSystem.ShootWithDamage(player.gameObject, Quaternion.LookRotation(player.TargetRotation), Vector3.zero, damage);  
 }  
  
 /// <summary>  
 /// Создает пистолет на игровом поле  
 /// </summary>  
 public void Run() {  
 var go = Client.client.SpawnObject(new SpawnPrefabCommand("pistol", position, Quaternion.identity, id, 0, 0));  
 go.GetComponent<PistolController>().gun = this;  
 }  
 }  
}

## ReloadingGun.cs

﻿using System;  
using CommandsSystem;  
using Events;  
using Interpolation.Managers;  
using Networking;  
using UnityEngine;  
using Random = UnityEngine.Random;  
  
namespace Character.Guns {  
 /// <summary>  
 /// Базовый класс для оружия, которое может перезаряжаться  
 /// </summary>  
 [Serializable]  
 public abstract class ReloadingGun : IGun {  
 /// <summary>  
 /// Состояние оружия в виде целого числа (нужно для сериализации)  
 /// </summary>  
 public int \_state = (int)GunState.READY;  
 /// <summary>  
 /// Состояние оружия  
 /// </summary>  
 public GunState state {  
 get => (GunState) \_state;  
 set => \_state = (int) value;  
 }  
   
 /// <summary>  
 /// Позиция на игровом поле, на котором должно появиться оружие  
 /// </summary>  
 public Vector3 position;  
 /// <summary>  
 /// id оружия  
 /// </summary>  
 public int id;  
  
 /// <summary>  
 /// Возвращает время перезарядки магазина  
 /// </summary>  
 /// <returns>Время перезарядки магазина</returns>  
 public abstract float GetFullReloadTime();  
 /// <summary>  
 /// Возвращает время перезарядки между выстрелами  
 /// </summary>  
 /// <returns>Время перезарядки между выстрелами</returns>  
 public abstract float GetReloadTime();  
 /// <summary>  
 /// Возвращает количество выстрелов в одном боекомплекте  
 /// </summary>  
 /// <returns>Количество выстрелов в одном боекомплекте</returns>  
 public abstract int GetBulletsInMagazine();  
  
 /// <summary>  
 /// Переменная для хранения bulletsCount  
 /// </summary>  
 public int \_bulletsCount;  
 /// <summary>  
 /// Текущее количество патронов, которое заряжено (включая тот, которым будет произведен выстрел)  
 /// </summary>  
 public int bulletsCount {   
 get => \_bulletsCount;  
 private set {  
 \_bulletsCount = value;  
 if (player != null) {  
 EventsManager.handler.OnPlayerBulletsCountChanged(player.gameObject, \_bulletsCount);  
 }  
 }   
 }  
  
 /// <summary>  
 /// Переенная для хранения magazinesCount  
 /// </summary>  
 public int \_magazinesCount = 1;  
 /// <summary>  
 /// Количество полных магазинов с патронами, которое осталось  
 /// </summary>  
 public int magazinesCount {  
 get => \_magazinesCount;  
 private set {  
 \_magazinesCount = value;  
 if (player != null) {  
 EventsManager.handler.OnPlayerMagazinesCountChanged(player.gameObject, \_magazinesCount);  
 }  
 }  
 }  
  
 /// <summary>  
 /// Конструктор оружия, которое может перезаряжаться. Инициализирует переменные  
 /// </summary>  
 public ReloadingGun() {  
 bulletsCount = GetBulletsInMagazine();  
 state = GunState.READY;  
 id = ObjectID.RandomID;  
 }  
  
  
 /// <summary>  
 /// Ссылка на MotionController игрока, который держит оружие  
 /// </summary>  
 protected MotionController player;  
 /// <summary>  
 /// Обрабатывает событие, когда игрок подбирает оружие  
 /// </summary>  
 /// <param name="player">Игрок, подобравший оружие</param>  
 public void OnPickedUp(GameObject player) {  
 this.player = player.GetComponent<MotionController>();  
 if (bulletsCount == 0) {  
 SetReloadMagazine();  
 }   
 }  
  
 /// <summary>  
 /// Проверяет, остались ли патроны или магазины в оружии  
 /// </summary>  
 /// <returns>true, если не осталось ни патронов, ни магазинов. Иначе возвращает false</returns>  
 public bool IsEmpty() => bulletsCount == 0 && magazinesCount == 0;  
  
 /// <summary>  
 /// Создает на игрвом поле данное оружие  
 /// </summary>  
 private void Spawn() {  
 Vector3 dir;  
 var rig = player.GetComponent<Rigidbody>();  
 dir = rig.velocity;  
 if (dir.sqrMagnitude < 0.01f)  
 dir = new Vector3(Random.value - 0.5f, 0, Random.value - 0.5f);//player.transform.forward;  
  
 id = ObjectID.RandomID;  
 this.position = player.transform.position - dir.normalized \* 2 + Vector3.up;  
 CommandsHandler.gameRoom.RunSimpleCommand(this as ICommand, MessageFlags.IMPORTANT);  
 }  
   
 /// <summary>  
 /// Обрабатывает выбрасывание игроком оружия  
 /// </summary>  
 public void OnDropped() {  
 /\* if (!IsEmpty()) // just destroy it  
 {  
 Spawn();  
 }\*/  
   
 this.player = null;  
 /\*   
 // drop reloading state  
 if (state == GunState.RELOADING\_MAGAZINE)  
 state = GunState.READY;\*/  
 }  
  
 /// <summary>  
 /// Время, которое осталось до перезарядки патрона или магазина  
 /// </summary>  
 private float needTime = 0;  
   
 /// <summary>  
 /// Обновляет состояние оружия. Перезаряжает патроны в оружии  
 /// </summary>  
 /// <param name="dt">Время, которое прошло с последнего обновления состояния</param>  
 public void Update(float dt) {  
 if (state == GunState.RELOADING\_BULLET) {  
 needTime -= dt;  
 if (needTime <= 0)  
 state = GunState.READY;  
 }  
 if (state == GunState.RELOADING\_MAGAZINE) {  
 needTime -= dt;  
 if (needTime <= 0) {  
 state = GunState.READY;  
 bulletsCount = GetBulletsInMagazine();  
 magazinesCount--;  
 }  
 }  
 }  
  
 /// <summary>  
 /// Выпускает пулю из оружия  
 /// </summary>  
 protected abstract void DoShoot();  
   
   
 /// <summary>  
 /// Производит выстрел из оружия  
 /// </summary>  
 /// <exception cref="InvalidOperationException">Вызывает исключение, если в оружии не было заряжено патрона</exception>  
 public void Shoot() {  
 if (state == GunState.READY) {  
 DoShoot();  
 bulletsCount--;  
 SetReloadBullet();  
   
 if (bulletsCount > 0) {  
 SetReloadBullet();  
 } else if (magazinesCount > 0) {  
 SetReloadMagazine();  
 } else {  
 //throw new Exception("werwerwererw");  
 player.GetComponent<ActionController>().SetNothing();  
 }  
 } else {  
 throw new InvalidOperationException("Cannot shoot without bullet");  
 }  
 }  
  
 /// <summary>  
 /// Устанавливает состояние на перезарядку патрона  
 /// </summary>  
 private void SetReloadBullet() {  
 state = GunState.RELOADING\_BULLET;  
 needTime = GetReloadTime();  
 }  
   
 /// <summary>  
 /// Устанавливает состояние на перезарядку магазина  
 /// </summary>  
 public void SetReloadMagazine() {  
 if (magazinesCount == 0) return;  
 bulletsCount = 0;  
 state = GunState.RELOADING\_MAGAZINE;  
 needTime = GetFullReloadTime();  
 }  
 }  
}

## SemiautoGun.cs

﻿using System;  
using CommandsSystem.Commands;  
using UnityEngine;  
  
namespace Character.Guns {  
 /// <summary>  
 /// Класс для автомата  
 /// </summary>  
 [Serializable]  
 public partial class SemiautoGun : ReloadingGun {  
 /// <summary>  
 /// Время перезарядки магазина  
 /// </summary>  
 public float \_fullReloadTime = 7.0f;  
 /// <summary>  
 /// Время перезарядки магазина  
 /// </summary>  
 public float \_reloadTime = 0.06f;  
 /// <summary>  
 /// Количество выстрелов в одном боекомплекте  
 /// </summary>  
 public int \_bulletsInMagazine = 50;  
  
 /// <summary>  
 /// Урон от выстрела  
 /// </summary>  
 public float damage = 3;  
 /// <summary>  
 /// Точность (вычисляется по Гауссу)  
 /// </summary>  
 public float accurancy = 50f;  
   
 /// <summary>  
 /// Возвращает время перезарядки магазина  
 /// </summary>  
 /// <returns>Время перезарядки магазина</returns>  
 public override float GetFullReloadTime() => \_fullReloadTime;  
 /// <summary>  
 /// Возвращает время перезарядки между выстрелами  
 /// </summary>  
 /// <returns>Время перезарядки между выстрелами</returns>  
 public override float GetReloadTime() => \_reloadTime;  
 /// <summary>  
 /// Возвращает количество выстрелов в одном боекомплекте  
 /// </summary>  
 /// <returns>Количество выстрелов в одном боекомплекте</returns>  
 public override int GetBulletsInMagazine() => \_bulletsInMagazine;  
   
 // переменные для корректной работы сериализации  
 /// public int \_bulletsCount;  
 /// public int \_magazinesCount;  
 /// public Vector3 position;  
 /// public int id;  
 /// public int \_state;  
  
   
 /// <summary>  
 /// Производит выстрел из автомата  
 /// </summary>  
 protected override void DoShoot() {  
 Vector3 random\_delta = ShootSystem.RandomDelta(1 / accurancy);  
 ShootSystem.ShootWithDamage(player.gameObject, Quaternion.LookRotation(player.TargetRotation), random\_delta, damage);  
 }  
   
 /// <summary>  
 /// Создает автомат на игровом поле  
 /// </summary>  
 public void Run() {  
 var go = Client.client.SpawnObject(new SpawnPrefabCommand("semiauto", position, Quaternion.identity, id, 0, 0));  
 go.GetComponent<SemiautoController>().gun = this;  
 }  
 }  
}

## ShootSystem.cs

﻿using Character.HP;  
using CommandsSystem;  
using CommandsSystem.Commands;  
using Networking;  
using UnityEngine;  
using Util2;  
using Physics = UnityEngine.Physics;  
  
namespace Character.Guns {  
 /// <summary>  
 /// Класс с функциями для стрельбы  
 /// </summary>  
 public static class ShootSystem {  
   
 /// <summary>  
 /// Находит раположение оружия у игрока  
 /// </summary>  
 /// <param name="characterPosition">Координаты игрока</param>  
 /// <returns>Расположение оружия</returns>  
 public static Vector3 GetGunPosition(Vector3 characterPosition) {  
 return characterPosition + Vector3.up \* 1.5f;  
 }  
   
 /// <summary>  
 /// Рисует трассер от выстрела  
 /// </summary>  
 /// <param name="start">Координата, откуда был произведен выстрел</param>  
 /// <param name="stop">Координата, куда попал выстрел</param>  
 public static void DrawTracer(Vector3 start, Vector3 stop) {  
 Client.client.bulletTrailRenderer.MoveFromTo(start, stop);  
 // DebugExtension.DrawArrow(start, stop - start);  
 /\* RotaryHeart.Lib.PhysicsExtension.DebugExtensions.DebugCapsule(start, stop, Color.magenta,  
 drawDuration: 100f, preview: PreviewCondition.Both);\*/  
 }  
   
 /// <summary>  
 /// Выпускает пулю из заданной позиции и проверяет в какой объект она попала  
 /// Отрисовывает выстрел.  
 /// </summary>  
 /// <param name="transform">Положение игрока, выпустившего пулю</param>  
 /// <param name="rotation">Направление, в котором дожна лететь пуля</param>  
 /// <param name="directionDelta">Насколько нужно отклонить пулю от правильного направления</param>  
 /// <param name="raycastRes">Информация о попаданнии пули<param>  
 /// <param name="command">Команда отрисовки трассера от выстрела.</param>  
 /// <returns>true если пуля попала в какой-то объект, иначе - false</returns>  
 public static bool SimpleRaycast(Transform transform, Quaternion rotation, Vector3 directionDelta, out RaycastHit raycastRes, out ICommand command) {  
 var position = GetGunPosition(transform.position);  
 var direction = rotation \* (Vector3.forward + directionDelta);  
  
 /\*RotaryHeart.Lib.PhysicsExtension.Physics.Raycast(position, direction, drawDuration: 0.1f,  
 hitColor: Color.red, noHitColor: Color.white, preview: PreviewCondition.Both);\*/  
 bool rres = Physics.Raycast(position, direction, out raycastRes);  
 if (rres && ObjectID.TryGetID(raycastRes.collider.gameObject, out int targetID)) {  
 var t = ObjectID.GetID(raycastRes.collider.gameObject);  
 DrawTracer(position, raycastRes.point);  
 command = new DrawTargetedTracerCommand(ObjectID.GetID(transform.gameObject), targetID, new HPChange());  
 } else if (rres) {  
 Vector3 target = raycastRes.point;  
 DrawTracer(position, target);  
 command = new DrawPositionTracerCommand(ObjectID.GetID(transform.gameObject), target);  
 } else {  
 Vector3 target = position + direction.normalized \* 100;  
 DrawTracer(position, target);  
 command = new DrawPositionTracerCommand(ObjectID.GetID(transform.gameObject), target);  
 }  
 return rres;  
 }  
  
 /// <summary>  
 /// Переменнаая для хранения результата от попадания пули  
 /// </summary>  
 private static RaycastHit \_raycastHit;  
 /// <summary>  
 /// Производит выстрел с уроном  
 /// </summary>  
 /// <param name="gameObject">Игрок, производящий выстрел</param>  
 /// <param name="rotation">Направление, в котором нужно произвести выстрел</param>  
 /// <param name="directionDelta">Отклонение от направления выстрела</param>  
 /// <param name="damage">Урон от выстрела</param>  
 /// <returns>true, если в результате выстрела был нанесен урон. Иначе false</returns>  
 public static bool ShootWithDamage(GameObject gameObject, Quaternion rotation, Vector3 directionDelta, float damage) {  
 ICommand command;  
 var raycastRes = SimpleRaycast(gameObject.transform, rotation, directionDelta, out \_raycastHit, out command);  
   
 if (raycastRes != false) {  
 var other = \_raycastHit.collider.gameObject;  
 var hp = other.GetComponent<HPController>();  
  
 if (hp != null) {  
 float realDamage = hp.TakeDamage(damage, DamageSource.Player(gameObject), false);  
 if (command is DrawTargetedTracerCommand c) {  
 c.HpChange.delta = -realDamage;  
 c.HpChange.source = DamageSource.Player(c.player);  
 }  
   
 CommandsHandler.gameModeRoom.RunSimpleCommand(command, MessageFlags.NONE);  
 return true;  
 }  
 }  
   
 CommandsHandler.gameModeRoom.RunSimpleCommand(command, MessageFlags.NONE);  
 return false;  
 }  
  
 /// <summary>  
 /// Создает случайное отклонение на основе распределения Гаусса  
 /// </summary>  
 /// <param name="sigma">Коэффициент в распределении Гаусса</param>  
 /// <returns>Случайное отклонение</returns>  
 public static Vector3 RandomDelta(double sigma) {  
 float x = (float) sClient.random.NextGaussian(0, sigma);  
 float y = (float) sClient.random.NextGaussian(0, sigma);  
 return new Vector3(x, y, 0);  
 }  
  
 /// <summary>  
 /// Производит выстрел бомбой  
 /// </summary>  
 /// <param name="gameObject">Игрок, который произвел выстрел</param>  
 /// <param name="target">Координата, в которую игрок производит выстрел</param>  
 /// <param name="bombPrefab">Название префаба бомбы</param>  
 public static void ShootWithBomb(GameObject gameObject, Vector3 target, string bombPrefab) {  
 Vector3 start = GetGunPosition(gameObject.transform.position);  
 start += (target - start).normalized \* 0.7f;  
   
 float len = (target - start).magnitude;  
 Vector3 medium = (start + target) / 2 + Vector3.up \* len;  
  
  
 float totalTime = len / 10f;  
   
 CommandsHandler.gameModeRoom.RunSimpleCommand(new SpawnParabolaFlyingCommand(  
 new SpawnPrefabCommand(bombPrefab, start, Quaternion.identity, ObjectID.RandomID, sClient.ID, ObjectID.GetID(gameObject)),  
 medium, target, totalTime), MessageFlags.IMPORTANT);  
 }  
 }  
}

## ShotGun.cs

﻿using System;  
using CommandsSystem.Commands;  
using UnityEngine;  
  
namespace Character.Guns {  
 /// <summary>  
 /// Класс для дробовика  
 /// </summary>  
 [Serializable]  
 public partial class ShotGun : ReloadingGun {  
 /// <summary>  
 /// Время перезарядки магазина  
 /// </summary>  
 public float \_fullReloadTime = 5.0f;  
 /// <summary>  
 /// Время перезарядки магазина  
 /// </summary>  
 public float \_reloadTime = 0.3f;  
 /// <summary>  
 /// Количество выстрелов в одном боекомплекте  
 /// </summary>  
 public int \_bulletsInMagazine = 3;  
  
 /// <summary>  
 /// Урон от выстрела  
 /// </summary>  
 public float damage = 18;  
 /// <summary>  
 /// Количество дроби, вылетающее за один выстрел  
 /// </summary>  
 public int shootsCount = 5;  
 /// <summary>  
 /// Точность (вычисляется по Гауссу)  
 /// </summary>  
 public float accurancy = 12f;  
   
 /// <summary>  
 /// Возвращает время перезарядки магазина  
 /// </summary>  
 /// <returns>Время перезарядки магазина</returns>  
 public override float GetFullReloadTime() => \_fullReloadTime;  
 /// <summary>  
 /// Возвращает время перезарядки между выстрелами  
 /// </summary>  
 /// <returns>Время перезарядки между выстрелами</returns>  
 public override float GetReloadTime() => \_reloadTime;  
 /// <summary>  
 /// Возвращает количество выстрелов в одном боекомплекте  
 /// </summary>  
 /// <returns>Количество выстрелов в одном боекомплекте</returns>  
 public override int GetBulletsInMagazine() => \_bulletsInMagazine;  
   
   
 // переменные для корректной работы сериализации  
 /// public int \_bulletsCount;  
 /// public int \_magazinesCount;  
 /// public Vector3 position;  
 /// public int id;  
 /// public int \_state;  
  
  
 /// <summary>  
 /// Производит выстрел из дробовика  
 /// </summary>  
 protected override void DoShoot() {  
 for (int i = 0; i < shootsCount; i++) {  
 Vector3 random\_delta = ShootSystem.RandomDelta(1 / accurancy);  
 ShootSystem.ShootWithDamage(player.gameObject, Quaternion.LookRotation(player.TargetRotation), random\_delta, damage);  
 }  
 }  
   
 /// <summary>  
 /// Создает дробовик на игровом поле  
 /// </summary>  
 public void Run() {  
 var go = Client.client.SpawnObject(new SpawnPrefabCommand("shotgun", position, Quaternion.identity, id, 0, 0));  
 go.GetComponent<ShotgunController>().gun = this;  
 }  
 }  
}

## BombGunController.cs

﻿using UnityEngine;  
  
namespace Character.Guns {  
 /// <summary>  
 /// Класс для гранатомета, расположенного в виде игрового объекта на поле  
 /// </summary>  
 public class BombGunController : GunController<BombGun> {}  
}

## GunController.cs

﻿using CommandsSystem.Commands;  
using Networking;  
using UnityEngine;  
  
namespace Character.Guns {  
 /// <summary>  
 /// Класс для оружия, которое расположено на игровом поле и которое можно подобрать  
 /// </summary>  
 /// <typeparam name="T">Оружие</typeparam>  
 public class GunController<T> : MonoBehaviour  
 where T: IGun {  
 public T gun;  
   
 /// <summary>  
 /// Время, когда была последний раз проведена попытка подобрать оружие.  
 /// Нужно для предотвращения спама командами подобрать оружие  
 /// </summary>  
 private float picked = float.MaxValue;  
  
 /// <summary>  
 /// Автоматически вызывается Unity при столкновении с другими объектами  
 /// </summary>  
 /// <param name="other">Другой объект</param>  
 private void OnTriggerEnter(Collider other) {  
 if (picked - Time.time < 5) return;  
   
 if (other.CompareTag("Player")) {  
 picked = Time.time;  
 var command = new PickUpGunCommand(ObjectID.GetID(other.gameObject), ObjectID.GetID(this.gameObject));  
   
 CommandsHandler.gameRoom.RunUniqCommand(command, UniqCodes.PICK\_UP\_GUN, command.gun, MessageFlags.IMPORTANT);  
 }  
 }  
 }  
}

## PistolController.cs

﻿namespace Character.Guns {  
 /// <summary>  
 /// Класс для пистолета, расположенного в виде игрового объекта на поле  
 /// </summary>  
 public class PistolController : GunController<Pistol> { }  
}

## SemiautoController.cs

﻿namespace Character.Guns {  
 /// <summary>  
 /// Класс для автомата, расположенного в виде игрового объекта на поле  
 /// </summary>  
 public class SemiautoController : GunController<SemiautoGun> { }  
}

## ShotgunController.cs

﻿namespace Character.Guns {  
 /// <summary>  
 /// Класс для дробовика, расположенного в виде игрового объекта на поле  
 /// </summary>  
 public class ShotgunController : GunController<ShotGun> { }  
}

## DamageSource.cs

﻿using GameMode;  
using UnityEngine;  
  
namespace Character.HP {  
 /// <summary>  
 /// Класс для работы с истониками урона  
 /// </summary>  
 public static class DamageSource {  
  
 /// <summary>  
 /// Возвращает числовое значение, соотвествующее отсутствию источнику урона  
 /// </summary>  
 /// <returns>Числовое значение, соотвествующее отсутствию источнику урона</returns>  
 public static int None() {  
 return 0;  
 }  
  
 /// <summary>  
 /// Возвращает числовое значение, соотвествующее бесконечно сильному источнику урона  
 /// </summary>  
 /// <returns>Числовое значение, соотвествующее бесконечно сильному источнику урона</returns>  
 public static int InstaKill() {  
 return 1;  
 }  
  
 /// <summary>  
 /// Возвращает числовое значение, соотвествующее игроку, который является источником урона  
 /// </summary>  
 /// <param name="id">id игрока, являющегося источником урона</param>  
 /// <returns>Числовое значение, соотвествующее игроку, который является источником урона</returns>  
 public static int Player(int id) {  
 return id;  
 }  
  
 /// <summary>  
 /// Возвращает числовое значение, соотвествующее источнику урона от игрока  
 /// </summary>  
 /// <param name="gameObject">Объект игрока, являющегося источником урона</param>  
 /// <returns>Числовое значение, соотвествующее источнику урона от игрока</returns>  
 public static int Player(GameObject gameObject) {  
 return Player(ObjectID.GetID(gameObject));  
 }  
  
 /// <summary>  
 /// Возвращает числовое значение, соотвествующее источнику урона от бомбы  
 /// </summary>  
 /// <returns>Числовое значение, соотвествующее источнику урона от бомбы</returns>  
 public static int Bomb() {  
 return 2;  
 }  
  
 /// <summary>  
 /// Получает объект, от которого был получен урон  
 /// </summary>  
 /// <param name="damageSource">Числовое значение, соответствующее источнику урона</param>  
 /// <returns>Объект, от которого был получен урон. null, если урон был получен не от объекта или этот объект уже не существует</returns>  
 public static GameObject GetSourceGO(int damageSource) {  
 GameObject res;  
 if (ObjectID.TryGetObject(damageSource, out res))  
 return res;  
 return null;  
 }   
 }  
}

## HPChange.cs

﻿namespace Character.HP {  
 /// <summary>  
 /// Структура, представляющая собой изменение здоровья игрового объекта  
 /// </summary>  
 public struct HPChange {  
 /// <summary>  
 /// Дельта, на которую изменилось здоровье  
 /// </summary>  
 public float delta;  
 /// <summary>  
 /// Источник изменения здоровья  
 /// </summary>  
 public int source;  
  
 /// <summary>  
 /// Конструктор изменения здоровья  
 /// </summary>  
 /// <param name="delta">Дельта, на которую изменилось здоровье</param>  
 /// <param name="source">Источник изменения здоровья</param>  
 public HPChange(float delta, int source) {  
 this.delta = delta;  
 this.source = source;  
 }  
 }  
}

## HPController.cs

﻿using CommandsSystem.Commands;  
using Events;  
using Networking;  
using UnityEngine;  
using UnityEngine.UI;  
  
namespace Character.HP {  
 /// <summary>  
 /// Класс для компоненты здоровья игрового объекта  
 /// </summary>  
 public class HPController : MonoBehaviour {  
 /// <summary>  
 /// Максимальное количество здоровья у объекта  
 /// </summary>  
 public float MaxHP = 100f;  
 /// <summary>  
 /// Скорость анимирования изменения здоровья у объекта  
 /// </summary>  
 public float HPAnimationSpeed = 140;  
  
 /// <summary>  
 /// Изображение, на котором рисуется полоска со здоровьем  
 /// </summary>  
 public Image hpImage;  
   
 /// <summary>  
 /// Переменная, хранящая был ли убит данный объект  
 /// </summary>  
 private bool dead = false;  
  
 /// <summary>  
 /// Текущее количество здоровья  
 /// </summary>  
 public float currentHp;  
  
 /// <summary>  
 /// Иництализирует переменные  
 /// </summary>  
 void Start() {  
 currentHp = MaxHP;  
 hpOnBar = currentHp;  
 }  
  
 /// <summary>  
 /// Переменная для хранения hpOnBar  
 /// </summary>  
 private float \_hpOnBar;  
 /// <summary>  
 /// Количество здоровья, которое сейчас отображается на полоске здоровья  
 /// </summary>  
 private float hpOnBar {  
 get => \_hpOnBar;  
 set {  
 \_hpOnBar = value;  
 hpImage.fillAmount = value / MaxHP;  
 }  
 }  
  
 /// <summary>  
 /// Анимирует полоску здоровья. Автоматически вызывается Unity каждый кадр  
 /// </summary>  
 void Update() {  
 if (hpOnBar != currentHp) {  
 hpOnBar = Mathf.MoveTowards(hpOnBar, currentHp, HPAnimationSpeed \* Time.deltaTime);  
 }  
 }  
  
 /// <summary>  
 /// Наносит урон по здоровью данного объекта  
 /// </summary>  
 /// <param name="damage">Наносимый урон</param>  
 /// <param name="source">Источник урона</param>  
 /// <param name="autoSendChange">Нужно ли отослать команду с изменением здоровья на сервер. Если false, то изменения будут применены локально</param>  
 /// <returns>Урон, который был нанесён</returns>  
 public float TakeDamage(float damage, int source, bool autoSendChange) {  
 float realDamage = Mathf.Min(currentHp, damage);  
  
 if (autoSendChange) {  
 CommandsHandler.gameModeRoom.RunSimpleCommand(new ChangeHPCommand(ObjectID.GetID(gameObject),   
 new HPChange(-realDamage, source)), MessageFlags.IMPORTANT);  
 } else {  
 \_applyHpChange(new HPChange(-realDamage, source));  
 }  
   
   
  
 return realDamage;  
 }  
  
 /// <summary>  
 /// Применяет изменения здоровья  
 /// </summary>  
 /// <param name="hpChange">Изменение здровья</param>  
 public void \_applyHpChange(HPChange hpChange) {  
 currentHp += hpChange.delta;  
 if (currentHp > MaxHP)  
 currentHp = MaxHP;  
  
 EventsManager.handler.OnObjectChangedHP(gameObject, hpChange.delta, hpChange.source);  
   
 if (!dead && currentHp <= 0) {  
 dead = true;  
 EventsManager.handler.OnObjectDead(gameObject, hpChange.source);  
 }  
 }  
   
   
 /// <summary>  
 /// Применяет изменение здоровья к объекту  
 /// </summary>  
 /// <param name="target">Объект</param>  
 /// <param name="hpChange">Изменение здоровья</param>  
 public static void ApplyHPChange(GameObject target, HPChange hpChange) {  
 if (hpChange.source == DamageSource.None()) return;  
 var hp = target.GetComponent<HPController>();  
 hp.\_applyHpChange(hpChange);  
 }  
 }  
  
  
}

## CommandsSystem.cs

using System;  
using System.IO;  
using System.Linq;  
using System.Text;  
using Character;  
using Character.Guns;  
using CommandsSystem.Commands;  
using JsonRequest;  
  
namespace CommandsSystem {  
   
 /// <summary>  
 /// Класс для сериализации и десериализации команд  
 /// </summary>  
 public class CommandsSystem {  
 /// <summary>  
 /// Кодирует заданную команду в бинарный вид и записывает в stream.  
 /// Данный метод использует кодогенерацию  
 /// </summary>  
 /// <param name="command">Команда</param>  
 /// <param name="stream">Поток, в который нужно записать команду</param>  
 /// <typeparam name="T">Тип команды</typeparam>  
 private void EncodeCommand<T>(T command, Stream stream) where T : ICommand {  
/\*BEGIN2\*/  
 if (command is AddOrChangeInstance addorchangeinstance) {  
 stream.WriteByte((byte)0);  
 var buf = addorchangeinstance.Serialize();  
 stream.Write(buf, 0, buf.Length);  
 } else  
 if (command is AddPlayerToGame addplayertogame) {  
 stream.WriteByte((byte)1);  
 var buf = addplayertogame.Serialize();  
 stream.Write(buf, 0, buf.Length);  
 } else  
 if (command is ApplyForceCommand applyforcecommand) {  
 stream.WriteByte((byte)2);  
 var buf = applyforcecommand.Serialize();  
 stream.Write(buf, 0, buf.Length);  
 } else  
 if (command is ChangeHPCommand changehpcommand) {  
 stream.WriteByte((byte)3);  
 var buf = changehpcommand.Serialize();  
 stream.Write(buf, 0, buf.Length);  
 } else  
 if (command is ChangePlayerProperty changeplayerproperty) {  
 stream.WriteByte((byte)4);  
 var buf = changeplayerproperty.Serialize();  
 stream.Write(buf, 0, buf.Length);  
 } else  
 if (command is ChangePlayerScore changeplayerscore) {  
 stream.WriteByte((byte)5);  
 var buf = changeplayerscore.Serialize();  
 stream.Write(buf, 0, buf.Length);  
 } else  
 if (command is CreateChatMessageCommand createchatmessagecommand) {  
 stream.WriteByte((byte)6);  
 var buf = createchatmessagecommand.Serialize();  
 stream.Write(buf, 0, buf.Length);  
 } else  
 if (command is DrawPositionTracerCommand drawpositiontracercommand) {  
 stream.WriteByte((byte)7);  
 var buf = drawpositiontracercommand.Serialize();  
 stream.Write(buf, 0, buf.Length);  
 } else  
 if (command is DrawTargetedTracerCommand drawtargetedtracercommand) {  
 stream.WriteByte((byte)8);  
 var buf = drawtargetedtracercommand.Serialize();  
 stream.Write(buf, 0, buf.Length);  
 } else  
 if (command is ExplodeBombCommand explodebombcommand) {  
 stream.WriteByte((byte)9);  
 var buf = explodebombcommand.Serialize();  
 stream.Write(buf, 0, buf.Length);  
 } else  
 if (command is PickCoinCommand pickcoincommand) {  
 stream.WriteByte((byte)10);  
 var buf = pickcoincommand.Serialize();  
 stream.Write(buf, 0, buf.Length);  
 } else  
 if (command is PickUpGunCommand pickupguncommand) {  
 stream.WriteByte((byte)11);  
 var buf = pickupguncommand.Serialize();  
 stream.Write(buf, 0, buf.Length);  
 } else  
 if (command is PlayerPushCommand playerpushcommand) {  
 stream.WriteByte((byte)12);  
 var buf = playerpushcommand.Serialize();  
 stream.Write(buf, 0, buf.Length);  
 } else  
 if (command is SetAfterShowResultsCommand setaftershowresultscommand) {  
 stream.WriteByte((byte)13);  
 var buf = setaftershowresultscommand.Serialize();  
 stream.Write(buf, 0, buf.Length);  
 } else  
 if (command is SetGameMode setgamemode) {  
 stream.WriteByte((byte)14);  
 var buf = setgamemode.Serialize();  
 stream.Write(buf, 0, buf.Length);  
 } else  
 if (command is SetPlatformStateCommand setplatformstatecommand) {  
 stream.WriteByte((byte)15);  
 var buf = setplatformstatecommand.Serialize();  
 stream.Write(buf, 0, buf.Length);  
 } else  
 if (command is SpawnParabolaFlyingCommand spawnparabolaflyingcommand) {  
 stream.WriteByte((byte)16);  
 var buf = spawnparabolaflyingcommand.Serialize();  
 stream.Write(buf, 0, buf.Length);  
 } else  
 if (command is SpawnPlayerCommand spawnplayercommand) {  
 stream.WriteByte((byte)17);  
 var buf = spawnplayercommand.Serialize();  
 stream.Write(buf, 0, buf.Length);  
 } else  
 if (command is SpawnPrefabCommand spawnprefabcommand) {  
 stream.WriteByte((byte)18);  
 var buf = spawnprefabcommand.Serialize();  
 stream.Write(buf, 0, buf.Length);  
 } else  
 if (command is StartGameCommand startgamecommand) {  
 stream.WriteByte((byte)19);  
 var buf = startgamecommand.Serialize();  
 stream.Write(buf, 0, buf.Length);  
 } else  
 if (command is TakeOwnCommand takeowncommand) {  
 stream.WriteByte((byte)20);  
 var buf = takeowncommand.Serialize();  
 stream.Write(buf, 0, buf.Length);  
 } else  
 if (command is Pistol pistol) {  
 stream.WriteByte((byte)21);  
 var buf = pistol.Serialize();  
 stream.Write(buf, 0, buf.Length);  
 } else  
 if (command is ShotGun shotgun) {  
 stream.WriteByte((byte)22);  
 var buf = shotgun.Serialize();  
 stream.Write(buf, 0, buf.Length);  
 } else  
 if (command is SemiautoGun semiautogun) {  
 stream.WriteByte((byte)23);  
 var buf = semiautogun.Serialize();  
 stream.Write(buf, 0, buf.Length);  
 } else  
 if (command is BombGun bombgun) {  
 stream.WriteByte((byte)24);  
 var buf = bombgun.Serialize();  
 stream.Write(buf, 0, buf.Length);  
 } else  
/\*END2\*/  
 {  
 throw new ArgumentException("Unknown command: " + command.GetType());  
 }  
   
   
 }  
   
 /// <summary>  
 /// MemoryStream для внутреннего использования (нужен для уменьшения нагрузки на сборщик мусора)  
 /// </summary>  
 private static MemoryStream \_stream = new MemoryStream();  
 /// <summary>  
 /// BinaryWriter для внутреннего использования (нужен для уменьшения нагрузки на сборщик мусора)  
 /// </summary>  
 private static BinaryWriter \_writer = new BinaryWriter(\_stream);  
  
 /// <summary>  
 /// Очищает потоки для записи команд от данных  
 /// </summary>  
 private void ResetWriteStreams() {  
 \_stream.SetLength(0);  
 \_writer.Seek(0, SeekOrigin.Begin);  
 }  
   
 /// <summary>  
 /// Кодирует обычную команду  
 /// </summary>  
 /// <param name="command">Команда</param>  
 /// <param name="room">Комната, в которую нужно отправить команду</param>  
 /// <param name="flags">Флаги отправки</param>  
 /// <typeparam name="T">Тип команды</typeparam>  
 /// <returns>Закодировннаую в бинарный вид команду</returns>  
 public byte[] EncodeSimpleCommand<T>(T command, int room, MessageFlags flags) where T : ICommand {  
 ResetWriteStreams();  
 \_writer.Write((byte) MessageType.SimpleMessage);   
 \_writer.Write((int)room);  
 \_writer.Write((byte)flags);  
   
 EncodeCommand(command, \_stream);  
 return \_stream.ToArray();  
 }  
  
 /// <summary>  
 /// Кодирует уникальную команду  
 /// </summary>  
 /// <param name="command">Команда</param>  
 /// <param name="room">Комната, в которую нужно отправить команду</param>  
 /// <param name="code1">Первая половина уникального кода</param>  
 /// <param name="code2">Вторая половина уникального кода</param>  
 /// <param name="flags">Флаги отправки</param>  
 /// <typeparam name="T">Тип команды</typeparam>  
 /// <returns>Закодировннаую в бинарный вид команду</returns>  
 public byte[] EncodeUniqCommand<T>(T command, int room, int code1, int code2, MessageFlags flags) where T : ICommand {  
 ResetWriteStreams();  
 \_writer.Write((byte) MessageType.UniqMessage);  
 \_writer.Write(room);  
 \_writer.Write((byte)flags);  
 \_writer.Write(code1);  
 \_writer.Write(code2);  
   
 EncodeCommand(command, \_stream);  
 return \_stream.ToArray();  
 }  
  
 /// <summary>  
 /// Кодирует команду запроса сообщений с сервера  
 /// </summary>  
 /// <param name="room">Комната из которой запрашиваются сообщения</param>  
 /// <param name="firstIndex">Индекс первого сообщения, которое нужно отправить</param>  
 /// <param name="lastIndex">Индекс последнего сообщения, которое нужно отправить</param>  
 /// <param name="flags">Флаги</param>  
 /// <returns>Закодированную в бинарный вид команду</returns>  
 public byte[] EncodeAskMessage(int room, int firstIndex, int lastIndex, MessageFlags flags) {  
 ResetWriteStreams();  
   
 \_writer.Write((byte) MessageType.AskMessage);  
 \_writer.Write(room);  
 \_writer.Write((byte)flags);  
 \_writer.Write(firstIndex);  
 \_writer.Write(lastIndex);  
  
 return \_stream.ToArray();  
 }  
  
 /// <summary>  
 /// Кодирует команду присоединения к игровой комнате  
 /// </summary>  
 /// <param name="room">Комната, к которой нужно присоединиться</param>  
 /// <param name="flags">Флаги</param>  
 /// <returns>Закодированную в бинарный вид команду</returns>  
 public byte[] EncodeJoinGameRoomMessage(int room, MessageFlags flags) {  
 ResetWriteStreams();  
   
 \_writer.Write((byte) MessageType.JoinGameRoom);  
 \_writer.Write(room);  
 \_writer.Write((byte)flags);  
  
 return \_stream.ToArray();  
 }  
  
 /// <summary>  
 /// Кодирует команду покидания игровой комнаты  
 /// </summary>  
 /// <param name="room">Комната, которую нужно покинуть</param>  
 /// <param name="flags">Флаги</param>  
 /// <returns>Закодированную в бинарный вид команду</returns>  
 public byte[] EncodeLeaveGameRoomMessage(int room, MessageFlags flags) {  
 ResetWriteStreams();  
   
 \_writer.Write((byte) MessageType.LeaveGameRoom);  
 \_writer.Write(room);  
 \_writer.Write((byte)flags);  
   
 return \_stream.ToArray();  
 }  
  
 /// <summary>  
 /// Кодирует JSON-сообщение на сервер  
 /// </summary>  
 /// <param name="json">JSON строка</param>  
 /// <param name="room">Комната, в которую нужно отправить сообщение</param>  
 /// <param name="flags">Флаги</param>  
 /// <returns>Закодированное в бинарный вид сообщение</returns>  
 public byte[] EncodeJsonMessage(string json, int room, MessageFlags flags) {  
 ResetWriteStreams();  
   
 \_writer.Write((byte)MessageType.JSON);  
 \_writer.Write(room);  
 \_writer.Write((byte)flags);  
 var bytes = Encoding.UTF8.GetBytes(json);  
 \_writer.Write(bytes);  
   
 return \_stream.ToArray();  
 }  
   
 /// <summary>  
 /// Декодирует команду с сервера  
 /// В данном методе используется кодогенерация  
 /// </summary>  
 /// <param name="array">Закодированная команда</param>  
 /// <param name="num">Номер команды</param>  
 /// <param name="room">Комната, в которую пришла команда</param>  
 /// <returns>Декодированную команду</returns>  
 public ICommand DecodeCommand(byte[] array, out int num, out int room) {  
 var stream = new MemoryStream(array);  
   
 var reader = new BinaryReader(stream);  
   
 num = reader.ReadInt32();  
 room = reader.ReadInt32();  
   
 byte commandType = (byte) stream.ReadByte();  
 byte[] arr = array.Skip(9).ToArray(); // TODO: fix performance  
 switch (commandType) {  
/\*BEGIN1\*/  
 case 0:  
 return AddOrChangeInstance.Deserialize(arr);  
 case 1:  
 return AddPlayerToGame.Deserialize(arr);  
 case 2:  
 return ApplyForceCommand.Deserialize(arr);  
 case 3:  
 return ChangeHPCommand.Deserialize(arr);  
 case 4:  
 return ChangePlayerProperty.Deserialize(arr);  
 case 5:  
 return ChangePlayerScore.Deserialize(arr);  
 case 6:  
 return CreateChatMessageCommand.Deserialize(arr);  
 case 7:  
 return DrawPositionTracerCommand.Deserialize(arr);  
 case 8:  
 return DrawTargetedTracerCommand.Deserialize(arr);  
 case 9:  
 return ExplodeBombCommand.Deserialize(arr);  
 case 10:  
 return PickCoinCommand.Deserialize(arr);  
 case 11:  
 return PickUpGunCommand.Deserialize(arr);  
 case 12:  
 return PlayerPushCommand.Deserialize(arr);  
 case 13:  
 return SetAfterShowResultsCommand.Deserialize(arr);  
 case 14:  
 return SetGameMode.Deserialize(arr);  
 case 15:  
 return SetPlatformStateCommand.Deserialize(arr);  
 case 16:  
 return SpawnParabolaFlyingCommand.Deserialize(arr);  
 case 17:  
 return SpawnPlayerCommand.Deserialize(arr);  
 case 18:  
 return SpawnPrefabCommand.Deserialize(arr);  
 case 19:  
 return StartGameCommand.Deserialize(arr);  
 case 20:  
 return TakeOwnCommand.Deserialize(arr);  
 case 21:  
 return Pistol.Deserialize(arr);  
 case 22:  
 return ShotGun.Deserialize(arr);  
 case 23:  
 return SemiautoGun.Deserialize(arr);  
 case 24:  
 return BombGun.Deserialize(arr);  
 /\*END1\*/  
 case 255:  
 return null;  
 case 254:  
 return Response.Deserialize(arr);  
 default:  
 throw new ArgumentOutOfRangeException("Command type is " + commandType);  
 }  
 }  
 }  
   
}

## ICommand.cs

﻿namespace CommandsSystem {  
 /// <summary>  
 /// Интерфейс для команды  
 /// </summary>  
 public interface ICommand {  
 /// <summary>  
 /// Выполняет команду  
 /// </summary>  
 void Run();  
 }  
}

## AddOrChangeInstance.cs

﻿using GameMode;  
  
namespace CommandsSystem.Commands {  
 /// <summary>  
 /// Команда, сообщаяющая о изменении одного из Instance в текущей игре  
 /// </summary>  
 public partial class AddOrChangeInstance {  
 /// <summary>  
 /// Изменившийся Instance  
 /// </summary>  
 public Instance instance;  
  
 /// <summary>  
 /// Применяет изменения  
 /// </summary>  
 public void Run() {  
 if (instance.id == InstanceManager.currentInstance.id) return;  
 for (int i = 0; i < InstanceManager.instances.Count; i++) {  
 if (InstanceManager.instances[i].id == instance.id) {  
 InstanceManager.instances[i] = instance;  
 return;  
 }  
 }  
 InstanceManager.instances.Add(instance);  
 }  
 }  
}

## AddPlayerToGame.cs

﻿using Events;  
using GameMode;  
  
namespace CommandsSystem.Commands {  
 /// <summary>  
 /// Добавляет игрока в игру  
 /// </summary>  
 public partial class AddPlayerToGame {  
 /// <summary>  
 /// Добавляемый игрок  
 /// </summary>  
 public Player player;  
  
 /// <summary>  
 /// Добавляет игрока в игру  
 /// </summary>  
 public void Run() {  
 if (PlayersManager.GetPlayerById(player.id) != null) return;  
 PlayersManager.players.Add(player);  
   
 EventsManager.handler.OnPlayerScoreChanged(player, player.score);  
 }  
 }  
}

## ApplyForceCommand.cs

﻿using UnityEngine;  
  
namespace CommandsSystem.Commands {  
 /// <summary>  
 /// Команда, сообщающая, что к игровому объекту нужно применить силу  
 /// </summary>  
 public partial class ApplyForceCommand {  
 /// <summary>  
 /// Id игрового объекта, к которму нужно применить силу  
 /// </summary>  
 public int objectId;  
 /// <summary>  
 /// Сила  
 /// </summary>  
 public Vector3 force;  
   
 /// <summary>  
 /// Конструктор команды  
 /// </summary>  
 /// <param name="gameObject">Игровой объект, к которму нужно применить силу</param>  
 /// <param name="force">Сила</param>  
 public ApplyForceCommand(GameObject gameObject, Vector3 force) :  
 this(ObjectID.GetID(gameObject), force) {}  
  
 /// <summary>  
 /// Применяет силу, если объект обрабатывается на данном клиенте  
 /// </summary>  
 public void Run() {  
 var gameObject = ObjectID.GetObject(objectId);  
 if (gameObject == null) {  
 Debug.LogError($"Not found gameobject#{objectId} for applying force");  
 return;  
 }  
 var rigidBody = gameObject.GetComponent<Rigidbody>();  
 if (rigidBody == null) return; // means we dont control this gameobject, so just skip it  
 rigidBody.AddForce(force, ForceMode.Impulse);  
 }  
 }  
}

## ChangeHPCommand.cs

﻿using Character.HP;  
  
namespace CommandsSystem.Commands {  
 /// <summary>  
 /// Команда, сообщающая что нужно изменить здоровье объекта  
 /// </summary>  
 public partial class ChangeHPCommand {  
 /// <summary>  
 /// id объекта, у которого нужно изменить здоровье  
 /// </summary>  
 public int id;  
 /// <summary>  
 /// Изменение здоровья  
 /// </summary>  
 public HPChange HpChange;  
  
 /// <summary>  
 /// Применяет изменение здоровья  
 /// </summary>  
 public void Run() {  
 HPController.ApplyHPChange(ObjectID.GetObject(id), HpChange);  
 }  
 }  
}

## ChangePlayerProperty.cs

﻿using Interpolation;  
using UnityEngine;  
  
namespace CommandsSystem.Commands {  
 /// <summary>  
 /// Класс для хранения и синхронизации состояния игрока  
 /// </summary>  
 public partial class ChangePlayerProperty {  
 /// <summary>  
 /// Состояние игрока  
 /// </summary>  
 public PlayerProperty property;  
 /// <summary>  
 /// Время, которое прошло с последнего изменения состояния  
 /// </summary>  
 public float deltaTime;  
   
 /// <summary>  
 /// Применяет изменение состояния  
 /// </summary>  
 public void Run() {  
 GameObject gameObject;  
 if (!ObjectID.TryGetObject(property.id, out gameObject))  
 {  
 return;  
 }  
  
 var controller = gameObject.GetComponent<UnmanagedGameObject<PlayerProperty>>();  
 if (controller is null) return;  
 controller.SetStateAnimated(property, deltaTime);  
#if DEBUG\_INTERPOLATION   
 DebugExtension.DebugPoint(property.position, Color.red, 0.1f, 3);  
#endif  
 }  
 }  
}

## ChangePlayerScore.cs

﻿using Events;  
using GameMode;  
  
namespace CommandsSystem.Commands {  
 /// <summary>  
 /// Команда для изменения очков игрока  
 /// </summary>  
 public partial class ChangePlayerScore {  
 /// <summary>  
 /// Игрок, у которого нужно изменить очки  
 /// </summary>  
 public int player;  
 /// <summary>  
 /// Новое количество очков  
 /// </summary>  
 public int newScore;  
  
 /// <summary>  
 /// Применяет изменения  
 /// </summary>  
 public void Run() {  
 var player = PlayersManager.GetPlayerById(this.player);  
 player.score += newScore;  
  
 EventsManager.handler.OnPlayerScoreChanged(player, player.score);  
 }  
 }  
}

## CreateChatMessageCommand.cs

﻿using GameMode;  
  
namespace CommandsSystem.Commands {  
 /// <summary>  
 /// Команда для отправки сообщения в чат  
 /// </summary>  
 public partial class CreateChatMessageCommand {  
 /// <summary>  
 /// Игрок, отправивший сообщение  
 /// </summary>  
 public int playerid;  
 /// <summary>  
 /// Сообщение  
 /// </summary>  
 public string message;  
  
 /// <summary>  
 /// Отображает сообщение в чате  
 /// </summary>  
 public void Run() {  
 var player = PlayersManager.GetPlayerById(playerid);  
 MainUIController.mainui.AddChatMessage(player, message);  
 }  
 }  
}

## DrawPositionTracerCommand.cs

﻿using Character.Guns;  
using Interpolation.Managers;  
using UnityEngine;  
  
namespace CommandsSystem.Commands {  
 /// <summary>  
 /// Команда рисования следа от пули между персонажем и координатой  
 /// </summary>  
 public partial class DrawPositionTracerCommand {  
 /// <summary>  
 /// id персонажа, выпустивший пулю  
 /// </summary>  
 public int player;  
 /// <summary>  
 /// Координата, в которую попала пуля  
 /// </summary>  
 public Vector3 target;  
   
 /// <summary>  
 /// Выполняет команду  
 /// </summary>  
 public void Run() {  
 var player = ObjectID.GetObject(this.player);  
 if (player.GetComponent<PlayerManagedGameObject>() != null) return;  
 ShootSystem.DrawTracer(ShootSystem.GetGunPosition(player.transform.position),   
 target);  
 }  
 }  
}

## DrawTargetedTracerCommand.cs

﻿using Character.Guns;  
using Character.HP;  
using Interpolation.Managers;  
  
namespace CommandsSystem.Commands {  
 /// <summary>  
 /// Команда рисования следа от пули между персонажами  
 /// </summary>  
 public partial class DrawTargetedTracerCommand {  
 /// <summary>  
 /// Id персонажа, выпустившего пулю  
 /// </summary>  
 public int player;  
 /// <summary>  
 /// Id персонажа, в которого попала пуля  
 /// </summary>  
 public int target;  
  
 /// <summary>  
 /// Изменение здоровья персонажа, в которого попала пуля  
 /// </summary>  
 public HPChange HpChange;  
  
 /// <summary>  
 /// Выполняет команду  
 /// </summary>  
 public void Run() {  
 var target = ObjectID.GetObject(this.target);  
  
  
 var player = ObjectID.GetObject(this.player);  
 if (player.GetComponent<PlayerManagedGameObject>() != null) return;  
   
 ShootSystem.DrawTracer(ShootSystem.GetGunPosition(player.transform.position),  
 ShootSystem.GetGunPosition(target.transform.position));  
   
 HPController.ApplyHPChange(target, HpChange);  
 }  
 }  
}

## ExplodeBombCommand.cs

﻿using GameMechanics;  
  
namespace CommandsSystem.Commands {  
 /// <summary>  
 /// Команда для взрыва указанной бомбы  
 /// </summary>  
 public partial class ExplodeBombCommand {  
 /// <summary>  
 /// id бомбы, которую нужно взорвать  
 /// </summary>  
 public int bombId;  
  
 /// <summary>  
 /// Взрывает указанную бомбу  
 /// </summary>  
 public void Run() {  
 var bomb = ObjectID.GetObject(bombId);  
 if (bomb == null) return;  
 bomb.GetComponent<Bomb>().RealExplode();  
 Client.client.RemoveObject(bomb);  
 }  
 }  
}

## PickCoinCommand.cs

﻿using Events;  
using UnityEngine;  
  
namespace CommandsSystem.Commands {  
 /// <summary>  
 /// Команда, сообщающая, что игрок подобрал монетку  
 /// </summary>  
 public partial class PickCoinCommand {  
 /// <summary>  
 /// Id игрока, подобравшего монетку  
 /// </summary>  
 public int player;  
 /// <summary>  
 /// Id подобранной монетки  
 /// </summary>  
 public int coin;  
  
 /// <summary>  
 /// Подбирает монетку  
 /// </summary>  
 public void Run()  
 {  
 var player = ObjectID.GetObject(this.player);  
 var coin = ObjectID.GetObject(this.coin);  
 if (player == null || coin == null)  
 {  
 Debug.LogWarning("Player or coin null.");  
 return;  
 }  
  
 EventsManager.handler.OnPlayerPickedUpCoin(player, coin);  
  
 Client.client.RemoveObject(coin);  
 }  
   
 }  
}

## PickUpGunCommand.cs

﻿using Character;  
using Character.Actions;  
using Character.Guns;  
using UnityEngine;  
using UnityEngine.Assertions;  
  
namespace CommandsSystem.Commands {  
 /// <summary>  
 /// Команда, сообщающая, что персонаж подобрал оружие  
 /// </summary>  
 public partial class PickUpGunCommand {  
 /// <summary>  
 /// ID персонажа  
 /// </summary>  
 public int player;  
 /// <summary>  
 /// ID оружия  
 /// </summary>  
 public int gun;  
  
   
 /// <summary>  
 /// Подбирает оружие  
 /// </summary>  
 public void Run()  
 {  
 var player = ObjectID.GetObject(this.player);  
 var gunObject = ObjectID.GetObject(this.gun);  
 if (player == null || gunObject == null)  
 {  
 Debug.LogWarning("Player or gun null.");  
 return;  
 }  
  
 var managed = player.GetComponent<ActionController>();  
 if (managed == null) {  
 Client.client.RemoveObject(gunObject);  
 return;  
 };  
  
 ReloadingGun gun = gunObject.GetComponent<PistolController>()?.gun ??   
 gunObject.GetComponent<SemiautoController>()?.gun ??  
 gunObject.GetComponent<ShotgunController>()?.gun ??  
 gunObject.GetComponent<BombGunController>()?.gun as ReloadingGun;/\*??  
 managed.GetComponent<SemiautoController>()?.gun as ReloadingGun;\*/  
  
 Assert.IsNotNull(gun);  
 if (gun is Pistol || gun is ShotGun || gun is BombGun) {  
 managed.SetAction<ShootPistolAction>(action => action.gun = gun);  
 } else {  
 managed.SetAction<ShootSemiautoAction>(action => action.gun = gun);  
 }  
   
 Client.client.RemoveObject(gunObject);  
 // managed.SetAction<();  
 }  
 }  
}

## PlayerPushCommand.cs

﻿using Character;  
using Interpolation.Managers;  
using UnityEngine;  
  
namespace CommandsSystem.Commands {  
 /// <summary>  
 /// Команда сообщающая, что для данного игрока нужно показать анимацию толкания  
 /// </summary>  
 public partial class PlayerPushCommand {  
 /// <summary>  
 /// id игрока, для которого нужно показать анимацию  
 /// </summary>  
 public int playerId;  
   
   
 /// <summary>  
 /// Показывает анимацию толкания для соотвествующего игрока  
 /// </summary>  
 public void Run() {  
 var player = ObjectID.GetObject(playerId);  
 if (player == null) {  
 Debug.LogWarning($"Player#{playerId} was null ");  
 return;  
 }  
  
 var manager = player.GetComponent<PlayerUnmanagedGameObject>();  
 if (manager == null) return; // means we own this player  
 var characterAnimator = player.GetComponent<CharacterAnimator>();  
 characterAnimator.SetPush();  
 }  
 }  
}

## SetAfterShowResultsCommand.cs

﻿using GameMode;  
  
namespace CommandsSystem.Commands {  
 /// <summary>  
 /// Команда, сообщающая что GameManager должен перестать показывать результаты игры  
 /// </summary>  
 public partial class SetAfterShowResultsCommand {  
 /// <summary>  
 /// Переменная для корректной работы сериализации  
 /// </summary>  
 public int \_;  
   
 /// <summary>  
 /// Изменяет состояние GameManager на соотвестующее команде  
 /// </summary>  
 public void Run() {  
 GameManager.SetAfterShowResults();  
 }  
 }  
}

## SetGameMode.cs

﻿using GameMode;  
  
namespace CommandsSystem.Commands {  
 /// <summary>  
 /// Команда, сообщающая, что нужно запускать заданный игровой режим  
 /// </summary>  
 public partial class SetGameMode {  
 /// <summary>  
 /// Код игрового режима  
 /// </summary>  
 public int gamemodeCode;  
 /// <summary>  
 /// Номер комнаты, в которой будет проводиться данный игровой режим  
 /// </summary>  
 public int roomId;  
 /// <summary>  
 /// Номер игры по порядку  
 /// </summary>  
 public int currentGameNum;  
  
 /// <summary>  
 /// Загружает заданный игровой режим  
 /// </summary>  
 public void Run() {  
 GameManager.SetGameMode(gamemodeCode, roomId, currentGameNum);  
 }  
 }  
}

## SetPlatformStateCommand.cs

﻿using GameMechanics;  
  
namespace CommandsSystem.Commands {  
 /// <summary>  
 /// Команда для синхронизации состояния подвижной платформы  
 /// </summary>  
 public partial class SetPlatformStateCommand {  
 /// <summary>  
 /// Id подвижной платформы  
 /// </summary>  
 public int id;  
 /// <summary>  
 /// Направление, в котором должна двигаться платформа сейчас  
 /// </summary>  
 public int direction;  
   
 /// <summary>  
 /// Синхронизирует состояние платформы с заданным  
 /// </summary>  
 public void Run() {  
 var platform = ObjectID.GetObject(id);  
 platform.GetComponent<MovingPlatform>().SetMoveState(direction);  
 }  
 }  
}

## SpawnParabolaFlyingCommand.cs

﻿using GameMechanics;  
using UnityEngine;  
  
namespace CommandsSystem.Commands {  
 /// <summary>  
 /// Команда для создания объекта с компонентой ParabolaFlyingObject  
 /// </summary>  
 public partial class SpawnParabolaFlyingCommand {  
 /// <summary>  
 /// Команда для создания объекта  
 /// </summary>  
 public SpawnPrefabCommand command;  
 /// <summary>  
 /// Среднее положение объекта  
 /// </summary>  
 public Vector3 medium;  
 /// <summary>  
 /// Конечное положение объекта  
 /// </summary>  
 public Vector3 target;  
 /// <summary>  
 /// Время полёта объекта  
 /// </summary>  
 public float totalTime;  
  
 /// <summary>  
 /// Создает на игровом поле объект с хаданными парамаетрами  
 /// </summary>  
 public void Run() {  
 var go = Client.client.SpawnObject(command);  
 var flying = go.AddComponent<ParabolaFlyingObject>();  
 flying.start = command.position;  
 flying.medium = medium;  
 flying.stop = target;  
 flying.totalTime = totalTime;  
 }  
 }  
}

## SpawnPlayerCommand.cs

﻿  
using Character;  
using Events;  
using GameMode;  
using UnityEngine;  
  
namespace CommandsSystem.Commands {  
 /// <summary>  
 /// Команда для создания на игровом поле персонажа  
 /// </summary>  
 public partial class SpawnPlayerCommand {  
 /// <summary>  
 /// Базовая команда для создания объекта   
 /// </summary>  
 public SpawnPrefabCommand command;  
   
 /// <summary>  
 /// Id игрока, который будет управлять данным персонажем  
 /// </summary>  
 public int playerId;  
   
 /// <summary>  
 /// Создает персонажа с заданными параметрами  
 /// </summary>  
 public void Run() {  
 Player player = PlayersManager.GetPlayerById(playerId);  
  
 GameObject go;  
 if (command.owner == sClient.ID) {  
 if (player.controllerType == 0) {  
 command.prefabName += "WithPlayer";  
 go = Client.client.SpawnObject(command);  
 Client.client.cameraObj.GetComponent<CameraFollower>().character = go;  
 Client.client.mainPlayerObj = go;  
 } else {  
 command.prefabName += "WithAI";  
 go = Client.client.SpawnObject(command);  
 }  
  
 } else {  
 command.prefabName += "Ghost";  
 go = Client.client.SpawnObject(command);  
 }  
  
 go.GetComponent<PlayerStorage>().Player = player;  
  
 EventsManager.handler.OnSpawnedPlayer(go, player);  
 /\* if (ObjectID.IsOwned(go)) {  
 switch (controllerType) {  
 case 0:  
   
 break;  
 case 1:  
   
 break;  
 }  
 }  
 \*/  
  
 }  
 }  
}

## SpawnPrefabCommand.cs

﻿using UnityEngine;  
  
namespace CommandsSystem.Commands {  
 /// <summary>  
 /// Команда для создания объекта на игровом поле  
 /// </summary>  
 public partial class SpawnPrefabCommand {  
 /// <summary>  
 /// Название префаба, в которой нужно создать  
 /// </summary>  
 public string prefabName;  
 /// <summary>  
 /// Позиция, в которой нужно создать объект  
 /// </summary>  
 public Vector3 position;  
 /// <summary>  
 /// Поворот, на который должен быть развернут объект  
 /// </summary>  
 public Quaternion rotation;  
 /// <summary>  
 /// Id объекта  
 /// </summary>  
 public int id;  
 /// <summary>  
 /// Владелец объекта  
 /// </summary>  
 public int owner;  
 /// <summary>  
 /// Игрок, создавший объект (если объект создан не игроком следует указать -1)  
 /// </summary>  
 public int creator;  
   
 private static System.Random random = new System.Random();  
  
 /// <summary>  
 /// Создает объект с заданными параметрами  
 /// </summary>  
 public void Run() {  
 var go = Client.client.SpawnObject(this);  
 }  
   
 }  
}

## StartGameCommand.cs

﻿namespace CommandsSystem.Commands {  
   
 /// <summary>  
 /// Команда начать игру  
 /// </summary>  
 public partial class StartGameCommand {  
 public int \_;  
  
 /// <summary>  
 /// Начинает игру  
 /// </summary>  
 public void Run() {  
 sClient.SetGameStarted();  
 }  
 }  
}

## TakeOwnCommand.cs

﻿  
namespace CommandsSystem.Commands {  
 /// <summary>  
 /// Команда для смены владельца у объекта  
 /// </summary>  
 public partial class TakeOwnCommand {  
 /// <summary>  
 /// Id объекта, у которого меняется владелец  
 /// </summary>  
 public int objectId;  
 /// <summary>  
 /// Id нового владельца объекта  
 /// </summary>  
 public int owner;  
  
 /// <summary>  
 /// Применяет изменения владельца  
 /// </summary>  
 public void Run() {  
 int curOwner;  
 if (!ObjectID.TryGetOwner(objectId, out curOwner)) return;  
  
 if (curOwner != 0) {  
 return;  
 }  
   
 ObjectID.SetOwner(objectId, owner);  
 foreach (var component in ObjectID.GetObject(objectId).GetComponents<IOwnedEventHandler>()) {  
 component.HandleOwnTaken(owner);  
 }  
 }  
 }  
}

## ClientEditor.cs

﻿using UnityEditor;  
using UnityEngine;  
  
namespace Editor {  
 /// <summary>  
 /// Класс для показа отладочной информации в инспекторе Unity  
 /// </summary>  
 [CustomEditor(typeof(Client))]  
 public class ClientEditor : UnityEditor.Editor {  
 /// <summary>  
 /// Выводит отладочную информацию в инспектор Unity  
 /// </summary>  
 public override void OnInspectorGUI() {  
 base.OnInspectorGUI();  
  
 if (Application.isPlaying) {  
 EditorGUILayout.TextArea("ID: " + sClient.ID);  
 EditorGUILayout.TextArea(ObjectID.ToString());  
 }  
 }  
 }  
}

## WebGLEditorScript.cs

﻿using UnityEditor;  
using UnityEngine;  
  
/// <summary>  
/// Класс для установки настроек сборки в webgl  
/// </summary>  
public class WebGLEditorScript  
{  
 /// <summary>  
 /// Устанавливает настройки сборки в webgl  
 /// </summary>  
 [MenuItem("Tools/Setup webgl settings")]  
 public static void DisableErrorMessageTesting() {  
 Debug.Log(PlayerSettings.WebGL.threadsSupport);  
 PlayerSettings.WebGL.threadsSupport = false;  
 PlayerSettings.SetIncrementalIl2CppBuild(BuildTargetGroup.WebGL, true);  
 Debug.Log(PlayerSettings.WebGL.memorySize);  
 PlayerSettings.WebGL.memorySize = 512;  
 //PlayerSettings.SetPropertyBool("useEmbeddedResources", true, BuildTargetGroup.WebGL);  
 }  
}

## EventsHandler.cs

﻿using Character.Guns;  
using Game;  
using GameMode;  
using UnityEngine;  
  
namespace Events {  
 /// <summary>  
 /// Класс для обработки событий  
 /// </summary>  
 public class EventsHandler {  
 /// <summary>  
 /// Делегат для события об изменении параметра персонажа  
 /// </summary>  
 /// <param name="player">Персонаж</param>  
 /// <param name="parameter">Изменившийся параметр</param>  
 /// <typeparam name="T">Тип параметра</typeparam>  
 public delegate void PlayerObjParameterChanged<T>(GameObject player, T parameter);  
  
   
 /// <summary>  
 /// Событие изменения числа патронов в оружии персонажа  
 /// </summary>  
 public PlayerObjParameterChanged<int> OnPlayerBulletsCountChanged = delegate { };  
 /// <summary>  
 /// Событие изменения числа магазинов у персонажа  
 /// </summary>  
 public PlayerObjParameterChanged<int> OnPlayerMagazinesCountChanged = delegate { };  
 /// <summary>  
 /// Событие, когда персонаж подбирает оружие  
 /// </summary>  
 public PlayerObjParameterChanged<IGun> OnPlayerPickedUpGun = delegate { };  
 /// <summary>  
 /// Событие, когда персонаж теряет оружие  
 /// </summary>  
 public PlayerObjParameterChanged<IGun> OnPlayerDroppedGun = delegate { };  
  
 /// <summary>  
 /// Событие, когда персонаж подбирает монетку  
 /// </summary>  
 public PlayerObjParameterChanged<GameObject> OnPlayerPickedUpCoin = delegate { };  
   
   
   
 /// <summary>  
 /// Делегат события об изменении параметра игрока  
 /// </summary>  
 /// <param name="player">Игрок</param>  
 /// <param name="parameter">Изменившийся параметр</param>  
 /// <typeparam name="T">Тип параметра</typeparam>  
 public delegate void PlayerParameterChanged<T>(Player player, T parameter);  
  
 /// <summary>  
 /// Событие, когда игрок поу  
 /// </summary>  
 public PlayerParameterChanged<int> OnPlayerScoreChanged = delegate { };  
  
 /// <summary>  
 /// Делегат для события смерти объекта  
 /// </summary>  
 /// <param name="go">Умерший объект</param>  
 /// <param name="killSource">Источник урона</param>  
 public delegate void ObjectDead(GameObject go, int killSource);  
 /// <summary>  
 /// Событие, когда объект умирает  
 /// </summary>  
 public ObjectDead OnObjectDead = delegate { };  
  
 /// <summary>  
 /// Делегат для события о получении урона объектом  
 /// </summary>  
 /// <param name="go">Объект, получивший урон</param>  
 /// <param name="delta">Изменение здоровья объекта</param>  
 /// <param name="damageSource">Источник урона</param>  
 public delegate void ObjectGotDamage(GameObject go, float delta, int damageSource);  
 /// <summary>  
 /// Событие о получении урона объектом  
 /// </summary>  
 public ObjectGotDamage OnObjectChangedHP = delegate { };  
  
  
 /// <summary>  
 /// Делегат для события об изменении информации о текущем матче  
 /// </summary>  
 /// <param name="last">Старая информация о матче</param>  
 /// <param name="current">Новая информация о матче</param>  
 public delegate void CurrentMatchChanged(MatchInfo last, MatchInfo current);  
 /// <summary>  
 /// Событие об изменении информации о текущем матче  
 /// </summary>  
 public CurrentMatchChanged OnCurrentMatchChanged = delegate { };  
  
 /// <summary>  
 /// Делегат для события о появлении персонажа на игровом поле  
 /// </summary>  
 /// <param name="gameObject">Персонаж</param>  
 /// <param name="player">Игрок</param>  
 public delegate void SpawnedPlayer(GameObject gameObject, Player player);  
 /// <summary>  
 /// Событие о появлении персонажа на игровом поле  
 /// </summary>  
 public SpawnedPlayer OnSpawnedPlayer = delegate { };  
 }  
}

## EventsManager.cs

﻿using UnityEngine;  
  
namespace Events {  
 /// <summary>  
 /// Класс для управления обработкой событии  
 /// </summary>  
 public class EventsManager : MonoBehaviour {  
  
 /// <summary>  
 /// Текущий обработчик событий  
 /// </summary>  
 public static EventsHandler handler;  
  
 /// <summary>  
 /// Инициализирует переменные. Автоматически вызывается Unity  
 /// </summary>  
 private void Awake() {  
 handler = new EventsHandler();  
 }  
  
 /// <summary>  
 /// Устанавливает обработчики событий  
 /// </summary>  
 private void Start() {  
 MainUIController.mainui.SetupHandlers();  
 }  
 }  
}

## GameManager.cs

﻿using System;  
using System.Collections.Generic;  
using System.Diagnostics;  
using System.Linq;  
using Character;  
using CommandsSystem.Commands;  
using Events;  
using Networking;  
using UI;  
using UnityEngine;  
using UnityEngine.Assertions;  
using UnityEngine.SceneManagement;  
using Util2;  
using Debug = UnityEngine.Debug;  
using Random = UnityEngine.Random;  
  
namespace GameMode {  
 /// <summary>  
 /// Класс для управления игрой  
 /// </summary>  
 public static class GameManager {  
 /// <summary>  
 /// Количество игр, которое нужно сыграть  
 /// </summary>  
 private const int TOTAL\_GAMES\_COUNT = 2;  
   
 /// <summary>  
 /// Перечисление возможных состояний игры  
 /// </summary>  
 private enum STATE {  
 INIT,  
 WAIT\_OTHERS,  
 CHOOSE\_GAMEMODE,  
 WAIT\_CHOOSING\_GAMEMODE,  
 WAIT\_FOR\_ALL\_LOAD\_GAMEMODE,  
 UPDATE\_GAMEMODE,  
 STOP\_GAMEMODE,  
 SHOW\_RESULTS,  
 WAIT\_SHOW\_RESULTS,  
 WAIT\_AFTER\_SHOW\_RESULTS,  
 AFTER\_SHOW\_RESULTS,  
 FINISH  
 }  
  
 /// <summary>  
 /// Переменная, хранящая была ли загружена новая сцена на предыдущем кадре  
 /// </summary>  
 public static bool sceneReloaded = false;  
 /// <summary>  
 /// Количество сыгранных игр  
 /// </summary>  
 public static int gamesCount = 0;  
   
 /// <summary>  
 /// Переменная для хранения состояния игры  
 /// </summary>  
 private static STATE \_state;  
 /// <summary>  
 /// Текущее состояние игры  
 /// </summary>  
 private static STATE state {  
 get => \_state;  
 set {  
 \_state = value;  
 DebugUI.debugText[1] = $"GameManager.State: {state}. {gamesCount}. {PlayersManager.mainPlayer.id}";  
 }  
 }  
  
 /// <summary>  
 /// Текущий игровой режим  
 /// </summary>  
 public static IGameMode gameMode;  
   
 /// <summary>  
 /// Время, когда должен закончиться игровой режим  
 /// </summary>  
 public static float timeEnd = -1;  
  
 /// <summary>  
 /// Изменяет игровой режим  
 /// </summary>  
 /// <param name="gamemodeCode">Код игрового режима</param>  
 /// <param name="roomId">Номер комнаты, в которой будет проводиться данный игровой режим</param>  
 /// <param name="currentGameNum">Номер игры по порядку</param>  
 public static void SetGameMode(int gamemodeCode, int roomId, int currentGameNum) {  
 CommandsHandler.gameModeRoom = new ClientCommandsRoom(roomId);  
   
   
 foreach (var player in PlayersManager.players) {  
 player.score = 0;  
 EventsManager.handler.OnPlayerScoreChanged(player, player.score);  
 }  
   
   
 var ttest = new Stopwatch();  
 ObjectID.Clear();  
 sceneReloaded = true;  
 sClient.LoadScene("new\_scene2");  
 Debug.LogError("Loaded scene in " + ttest.ElapsedMilliseconds);  
  
  
 switch (gamemodeCode) {  
 case 0:  
 gameMode = new ShooterGameMode();  
 break;  
 case 1:  
 gameMode = new PickCoinsGameMode();  
 break;  
 default:  
 throw new ArgumentException($"Unknown gamemodeCode: {gamemodeCode}");  
 }  
  
 availableGameModes.Remove(gamemodeCode);  
   
   
 InstanceManager.currentInstance.currentLoadedGamemodeNum++;  
 InstanceManager.currentInstance.Send();  
   
   
 MainUIController.mainui.HideTotalScore();  
  
   
   
   
 state = STATE.WAIT\_FOR\_ALL\_LOAD\_GAMEMODE;  
   
 Assert.AreEqual(InstanceManager.currentInstance.currentLoadedGamemodeNum, currentGameNum);  
  
 }  
  
   
 /// <summary>  
 /// Перестаёт показывать результаты игры  
 /// </summary>  
 public static void SetAfterShowResults() {  
 if (state == STATE.WAIT\_AFTER\_SHOW\_RESULTS)  
 state = STATE.AFTER\_SHOW\_RESULTS;  
 MainUIController.mainui.HideTotalScore();  
 }  
   
 /// <summary>  
 /// Время, которое осталось до конца показа результатов  
 /// </summary>  
 private static float showResultsWaitTime;  
  
 /// <summary>  
 /// Сбрасывает состояние переменных  
 /// </summary>  
 public static void Reset() {  
 state = STATE.INIT;  
 gamesCount = 0;  
 gameMode = null;  
 }  
   
 /// <summary>  
 /// Игровые режимы, которые можно запустить  
 /// </summary>  
 private static List<int> availableGameModes = new List<int>();  
  
 /// <summary>  
 /// Выбирает один из возможных режимов  
 /// </summary>  
 /// <returns>Выбраыннй игровой режим</returns>  
 private static int ChooseGameMode() {  
 if (availableGameModes.Count == 0) {  
 Debug.LogError("no available game modes. choosing random one");  
 return Random.Range(0, 2);  
 }  
  
 var index = Random.Range(0, availableGameModes.Count);  
 return availableGameModes[index];  
 }  
  
 /// <summary>  
 /// Обновляет состояние игры  
 /// </summary>  
 /// <exception cref="Exception"></exception>  
 public static void Update() {  
 switch (state) {  
 case STATE.INIT:  
 // CommandsHandler.gameRoom = new ClientCommandsRoom(137);  
 // CommandsHandler.gameRoom.RunUniqCommand(new StartGameCommand(), 1, 1, MessageFlags.IMPORTANT);  
  
 availableGameModes.Clear();  
 availableGameModes.Add(0);  
 availableGameModes.Add(1);  
   
 InstanceManager.currentInstance.Send();  
  
  
  
 CommandsHandler.gameRoom.RunSimpleCommand(new AddPlayerToGame(PlayersManager.mainPlayer),  
 MessageFlags.IMPORTANT);  
 int cnt = 2;  
 if (AutoMatchJoiner.isRunning && !AutoMatchJoiner.sneedWaitOtherPlayers)  
 cnt = 3;  
 for (int i = 0; i < cnt; i++) {  
 var ai = new Player(ObjectID.RandomID, sClient.ID, 1);  
 CommandsHandler.gameRoom.RunUniqCommand(new AddPlayerToGame(ai), UniqCodes.ADD\_AI\_PLAYER, i,  
 MessageFlags.IMPORTANT);  
 }  
  
 state = STATE.WAIT\_OTHERS;  
 break;  
 case STATE.WAIT\_OTHERS:  
 if (PlayersManager.playersCount == 4)  
 state = STATE.CHOOSE\_GAMEMODE;  
 break;  
 case STATE.CHOOSE\_GAMEMODE:  
 int gamemode = ChooseGameMode();  
 CommandsHandler.gameRoom.RunUniqCommand(new SetGameMode(gamemode, sClient.random.Next(),  
 InstanceManager.currentInstance.currentLoadedGamemodeNum + 1),  
 UniqCodes.CHOOSE\_GAMEMODE, InstanceManager.currentInstance.currentLoadedGamemodeNum + 1,  
 MessageFlags.IMPORTANT);  
 state = STATE.WAIT\_CHOOSING\_GAMEMODE;  
 break;  
 case STATE.WAIT\_CHOOSING\_GAMEMODE:  
 break;  
 case STATE.WAIT\_FOR\_ALL\_LOAD\_GAMEMODE:  
 bool allLoaded = true;  
 foreach (var instance in InstanceManager.instances) {  
 if (instance.currentLoadedGamemodeNum !=  
 InstanceManager.currentInstance.currentLoadedGamemodeNum)  
 allLoaded = false;  
 }  
  
 if (allLoaded) {  
 timeEnd = Time.time + gameMode.TimeLength;  
 state = STATE.UPDATE\_GAMEMODE;  
 }  
  
 break;  
 case STATE.UPDATE\_GAMEMODE:  
 sceneReloaded = false;  
  
 var res = gameMode.Update();  
 if (!res) {  
 state = STATE.STOP\_GAMEMODE;  
 }  
   
 MainUIController.mainui.SetTimerTime((int) (timeEnd - Time.time));  
  
 if (timeEnd < Time.time) {  
 state = STATE.STOP\_GAMEMODE;  
 }  
  
 break;  
 case STATE.STOP\_GAMEMODE:  
 var res2 = gameMode.Stop();  
 if (!res2) {  
 state = STATE.SHOW\_RESULTS;  
 }  
  
 break;  
 case STATE.SHOW\_RESULTS:   
 var players2 = PlayersManager.playersSortedByScore;  
  
 for (int i = 0; i < players2.Count; i++) {  
 players2[i].placeInLastGame = i + 1;  
 players2[i].totalScore += players2.Count - i;  
 }  
  
 gamesCount++;  
 if (gamesCount == TOTAL\_GAMES\_COUNT) {  
 MainUIController.mainui.ShowFinalResults();  
 state = STATE.FINISH;  
 break;  
 }  
  
 showResultsWaitTime = 8;  
 MainUIController.mainui.ShowTotalScore(TOTAL\_GAMES\_COUNT - gamesCount,  
 (int) Math.Ceiling(showResultsWaitTime));  
 state = STATE.WAIT\_SHOW\_RESULTS;  
 break;  
  
 case STATE.WAIT\_SHOW\_RESULTS:  
 showResultsWaitTime -= Time.deltaTime;  
 MainUIController.mainui.SetTotalScoreTimeRemaining((int) Math.Ceiling(showResultsWaitTime));  
 if (showResultsWaitTime < 0) {  
 state = STATE.WAIT\_AFTER\_SHOW\_RESULTS;  
 CommandsHandler.gameRoom.RunSimpleCommand(new SetAfterShowResultsCommand(),  
 MessageFlags.IMPORTANT);  
 }  
  
 break;  
 case STATE.WAIT\_AFTER\_SHOW\_RESULTS:  
 break;  
  
 case STATE.AFTER\_SHOW\_RESULTS:  
 if (gamesCount >= TOTAL\_GAMES\_COUNT) {  
 state = STATE.FINISH;  
 UberDebug.Log("finish");  
 } else {  
 state = STATE.CHOOSE\_GAMEMODE;  
 }  
  
 break;  
 case STATE.FINISH:  
 // ???  
 return;  
 default:  
 throw new Exception($"Unknown GameManager state: {state}");  
 }  
 }  
  
 }  
}

## Instance.cs

﻿using System.Collections.Generic;  
using CommandsSystem.Commands;  
using Networking;  
using UnityEngine;  
  
namespace GameMode {  
 /// <summary>  
 /// Класс для хранения и синхронизации информации о клиенте  
 /// </summary>  
 public class Instance {  
 /// <summary>  
 /// id данного instance  
 /// </summary>  
 public int id;  
  
 /// <summary>  
 /// Отображаемое имя у данного instance  
 /// </summary>  
 public string name;  
 /// <summary>  
 /// Текущий загруженный игровой режим  
 /// </summary>  
 public int currentLoadedGamemodeNum;  
 /// <summary>  
 /// Конструктор для сериализации  
 /// </summary>  
 public Instance() {}  
  
 /// <summary>  
 /// Конструктор  
 /// </summary>  
 /// <param name="id">id создаваемого instance</param>  
 public Instance(int id) {  
 this.id = id;  
 this.currentLoadedGamemodeNum = -1;  
 this.name = "Instance#" + id;  
 }  
  
 /// <summary>  
 /// Отправляет информацию об instance другим игрокам  
 /// </summary>  
 public void Send() {  
 CommandsHandler.gameRoom.RunSimpleCommand(new AddOrChangeInstance(this), MessageFlags.IMPORTANT);  
 }  
 }  
}

## InstanceManager.cs

﻿using System.Collections.Generic;  
  
namespace GameMode {  
 /// <summary>  
 /// Класс для управления instance  
 /// </summary>  
 public static class InstanceManager {  
 /// <summary>  
 /// Список instance в текущей игре  
 /// </summary>  
 public static List<Instance> instances = new List<Instance>();  
 /// <summary>  
 /// Текущий instance  
 /// </summary>  
 public static Instance currentInstance;  
 /// <summary>  
 /// Id текущего instance  
 /// </summary>  
 public static int ID => currentInstance.id;  
  
 /// <summary>  
 /// Инициализирует переменные  
 /// </summary>  
 public static void Init() {  
 currentInstance = new Instance(ObjectID.RandomID);  
 instances.Add(currentInstance);  
 }  
  
 /// <summary>  
 /// Сбрасывает состояние переменных  
 /// </summary>  
 public static void Reset() {  
 instances.Clear();  
 if (currentInstance != null) {  
 instances.Add(currentInstance);  
 currentInstance.currentLoadedGamemodeNum = -1;  
 }  
 }  
 }  
}

## MatchesManager.cs

﻿using System.Collections.Generic;  
using CommandsSystem;  
using CommandsSystem.Commands;  
using Events;  
using GameMode;  
using JsonRequest;  
using LightJson;  
using Networking;  
using UI;  
using UnityEngine;  
  
namespace Game {  
 /// <summary>  
 /// Класс для управления и присоединения к игровым матчам  
 /// </summary>  
 public static class MatchesManager {  
 /// <summary>  
 /// Перечисление возможных состояний менеджера матчей  
 /// </summary>  
 public enum STATE {  
 START\_FIND,  
 WAIT\_MATCHES\_INFO,  
 WAIT\_PLAYERS\_IN\_MATCH,  
 WAIT\_STARTING\_MATCH,  
 PLAYING\_MATCH  
 }  
   
 /// <summary>  
 /// Переменная для хранения состояния менеджера матчей  
 /// </summary>   
 private static STATE \_state = STATE.START\_FIND;  
  
 /// <summary>  
 /// Состояние менеджера матчей  
 /// </summary>  
 private static STATE state {  
 get => \_state;  
 set {  
 \_state = value;  
 DebugUI.debugText[1] = $"MatchesManager.State: {state}.";  
 }  
 }  
  
 /// <summary>  
 /// Информация о текущем матче  
 /// </summary>  
 public static MatchInfo currentMatch;  
  
 /// <summary>  
 /// Создаёт матч с заданными параметрами  
 /// </summary>  
 /// <param name="matchInfo">Параметры создаваемого матча</param>  
 private static void CreateMatch(MatchInfo matchInfo) {  
 UberDebug.LogChannel("Matchmaking", "Creating match with params: " + matchInfo.ToJson().ToString());  
 RequestsManager.Send(new Request(CommandsHandler.matchmakingRoom, RequestType.CreateMatch,   
 matchInfo.ToJson(), response => {  
 if (response["result"] != "success") {  
 UberDebug.LogErrorChannel("Matchmaking", "Error in json request: " + response.ToString());  
 state = STATE.START\_FIND;  
 return;  
 }  
 UberDebug.LogChannel("Matchmaking", "Created match");  
 JoinMatch(matchInfo.roomid);  
 } ));  
 }  
  
 /// <summary>  
 /// Присоединяется к заданному матчу  
 /// </summary>  
 /// <param name="matchid">ID матча, к которому нужно присоединиться</param>  
 private static void JoinMatch(int matchid) {  
 UberDebug.LogChannel("Matchmaking", "Joining match#" + matchid);  
 var json = new JsonObject();  
 json["matchid"] = matchid;  
 json["name"] = PlayersManager.mainPlayer.name;  
 RequestsManager.Send(new Request(CommandsHandler.matchmakingRoom, RequestType.JoinMatch,  
 json, response => {  
 if (response["result"] != "success") {  
 UberDebug.LogErrorChannel("Matchmaking", "Error in json request: " + response.ToString());  
 state = STATE.START\_FIND;  
 return;  
 }  
 UberDebug.LogChannel("Matchmaking", "Joined match#" + matchid);  
   
 state = STATE.WAIT\_PLAYERS\_IN\_MATCH;  
   
 CommandsHandler.gameRoom = new ClientCommandsRoom(matchid);  
   
 HandleJsonMatchChanged(response);  
   
 }));  
 }  
  
 /// <summary>  
 /// Получает список матчей и автоматически присоединяется к одному из возможных  
 /// </summary>  
 private static void GetMatchesList() {  
 RequestsManager.Send(new Request(CommandsHandler.matchmakingRoom, RequestType.GetMatchesList, new JsonObject(),   
 response => {  
 if (response["result"] != "success") {  
 UberDebug.LogErrorChannel("Matchmaking", "Error in json request: " + response.ToString());  
 state = STATE.START\_FIND;  
 return;  
 }  
  
 UberDebug.LogChannel("Matchmaking", "Available matches: " + response["matches"].ToString());  
 foreach (var matchJson in response["matches"].AsJsonArray) {  
 var match = MatchInfo.FromJson(matchJson);  
 if (match.players.Count < match.maxPlayersCount) {  
 JoinMatch(match.roomid);  
 return;  
 }  
 }  
  
 int mid = ObjectID.RandomID;  
 var matchInfo = new MatchInfo("Match#" + mid, mid, 2, new List<string>(), 0);  
 CreateMatch(matchInfo);  
 }));  
 }  
  
 /// <summary>  
 /// Посылает сообщение о старте матча  
 /// </summary>  
 public static void SendStartGame() {  
 if (state == STATE.WAIT\_STARTING\_MATCH) return;  
 if (state == STATE.PLAYING\_MATCH) return;  
 if (state != STATE.WAIT\_PLAYERS\_IN\_MATCH) {  
 UberDebug.LogErrorChannel("Matchmaking", $"Invalid state of MatchesManager: {state} while trying to start match!");  
 return;  
 }  
  
 state = STATE.WAIT\_STARTING\_MATCH;  
   
 var json = new JsonObject();  
 json["matchid"] = currentMatch.roomid;  
 json["state"] = 1;  
 UberDebug.LogChannel("Matchmaking", "Starting game");  
 RequestsManager.Send(new Request(CommandsHandler.matchmakingRoom, RequestType.ChangeMatchState,  
 json, response => {  
 if (response["result"] != "success") {  
 UberDebug.LogErrorChannel("Matchmaking", "Error in json request: " + response.ToString());  
 GameManager.Reset();  
 return;  
 }  
 UberDebug.LogChannel("Matchmaking", "Started match: " + response["match"].ToString());  
 CommandsHandler.gameRoom.RunUniqCommand(new StartGameCommand(123), UniqCodes.START\_GAME, 0,  
 MessageFlags.NONE);  
 }));  
  
 }  
  
 /// <summary>  
 /// Устанавливает локальное состояние матча на PLAYING  
 /// </summary>  
 public static void SetMatchIsPlaying() {  
 state = STATE.PLAYING\_MATCH;  
 }  
   
 /// <summary>  
 /// Обрабатывает JSON-сообщение, что текущий матч изменился  
 /// </summary>  
 /// <param name="json">JSON-сообщение</param>  
 public static void HandleJsonMatchChanged(JsonValue json) {  
 UberDebug.LogChannel("Matchmaking", "Match changed " + json.ToString());  
 var mi = MatchInfo.FromJson(json["match"]);  
  
 var last = currentMatch;  
 currentMatch = mi;  
  
 EventsManager.handler.OnCurrentMatchChanged(last, currentMatch);  
 }  
   
 /// <summary>  
 /// Обновляет состояние менеджера матчей  
 /// </summary>  
 public static void Update() {  
 switch (state) {  
 case STATE.START\_FIND:  
 CommandsHandler.matchmakingRoom = new ClientCommandsRoom(42);  
 GetMatchesList();  
   
 state = STATE.WAIT\_MATCHES\_INFO;  
 break;  
 case STATE.WAIT\_MATCHES\_INFO:  
 break;  
 case STATE.WAIT\_PLAYERS\_IN\_MATCH:  
 break;  
 case STATE.WAIT\_STARTING\_MATCH:  
 break;  
 case STATE.PLAYING\_MATCH:  
 break;  
 }  
 }  
  
 public static void Reset() {  
 \_state = STATE.START\_FIND;  
 }  
 }  
}

## MatchInfo.cs

﻿using System.Collections.Generic;  
using LightJson;  
  
namespace Game {  
 /// <summary>  
 /// Класс для информации о матче  
 /// </summary>  
 public class MatchInfo {  
 /// <summary>  
 /// id комнаты, в которой проводится матч  
 /// </summary>  
 public int roomid;  
 /// <summary>  
 /// Имя матча  
 /// </summary>  
 public string name;  
 /// <summary>  
 /// Максимальное количество игроков в матче  
 /// </summary>  
 public int maxPlayersCount;  
 /// <summary>  
 /// Список игроков в матче  
 /// </summary>  
 public List<string> players;  
  
 /// <summary>  
 /// Текущее состояние матча. 0 означает, что матч ещё не начался. 1 -- начался.  
 /// </summary>  
 public int state;  
  
 /// <summary>  
 /// Конструктор информации о матче  
 /// </summary>  
 /// <param name="name">Имя матча</param>  
 /// <param name="roomid">id комнаты, в которой проводится матч</param>  
 /// <param name="maxPlayersCount">Максимальное количество игроков в матче</param>  
 /// <param name="players">Список игроков в матче</param>  
 /// <param name="state">Текущее состояние матча</param>  
 public MatchInfo(string name, int roomid, int maxPlayersCount, List<string> players, int state) {  
 this.maxPlayersCount = maxPlayersCount;  
 this.roomid = roomid;  
 this.name = name;  
 this.players = players;  
 this.state = state;  
 }  
  
 /// <summary>  
 /// Записывает информацию о матче в формате JSON  
 /// </summary>  
 /// <returns>Информациюю о матче в формате JSON</returns>  
 public JsonValue ToJson() {  
 //return $"{{'roomid':{roomid}, 'name':'{name}', 'maxPlayersCount':{maxPlayersCount}, 'playersCount':{playersCount} }}";  
 var res = new JsonObject();  
 res["roomid"] = roomid;  
 res["name"] = name;  
 res["maxPlayersCount"] = maxPlayersCount;  
 res["state"] = state;  
 return res;  
 }  
  
 /// <summary>  
 /// Создаёт информацию о матче из JSON-объекта  
 /// </summary>  
 /// <param name="json">JSON-объект</param>  
 /// <returns>Информацию о матче</returns>  
 public static MatchInfo FromJson(JsonValue json) {  
 var players = new List<string>();  
 foreach (var value in json["players"].AsJsonArray) {  
 players.Add(value.AsString);  
 }  
 return new MatchInfo(json["name"].AsString, json["roomid"].AsInteger,   
 json["maxPlayersCount"].AsInteger, players, json["state"]);  
 }  
 }  
}

## Player.cs

﻿  
using UnityEngine;  
  
namespace GameMode {  
 /// <summary>  
 /// Класс для игрока  
 /// </summary>  
 public class Player {  
 /// <summary>  
 /// ID игрока  
 /// </summary>  
 public int id;  
 /// <summary>  
 /// Количество очков у игрока  
 /// </summary>  
 public int score;  
 /// <summary>  
 /// Имя игрока  
 /// </summary>  
 public string name;  
 /// <summary>  
 /// Владелец игрока (id instance, который управляет игроком)  
 /// </summary>  
 public int owner;  
  
 /// <summary>  
 /// Кем управляется игрок. 0 -- человеком, 1 -- компьютером  
 /// </summary>  
 public int controllerType;  
  
 /// <summary>  
 /// Суммарное количество очков за все игровые режимы  
 /// </summary>  
 public int totalScore;  
 /// <summary>  
 /// Место, занятое в последней игре  
 /// </summary>  
 public int placeInLastGame;  
   
   
 /// <summary>  
 /// Конструктор для сериализации  
 /// </summary>  
 public Player() {}  
  
 /// <summary>  
 /// Конструктор игрока  
 /// </summary>  
 /// <param name="id">ID игрока</param>  
 /// <param name="owner">Владелец игрока</param>  
 /// <param name="controllerType">Кем управляется игрок. 0 -- человеком, 1 -- компьютером</param>  
 public Player(int id, int owner, int controllerType) {  
 this.id = id;  
 this.score = 0;  
 this.name = "Player#" + Random.Range(0, 100);  
 this.owner = owner;  
 this.controllerType = controllerType;  
 this.placeInLastGame = -1;  
 this.totalScore = 0;  
 }  
  
 /// <summary>  
 /// Записывает информацию об игроке в строку  
 /// </summary>  
 /// <returns>Строку с информацией об игроке</returns>  
 public override string ToString() {  
 string controllerName;  
 if (controllerType == 0) {  
 controllerName = "player";  
 } else if (controllerType == 1) {  
 controllerName = "AI";  
 } else {  
 controllerName = "unknown: " + controllerType;  
 }  
 return $"Player#{id} name:{name} score:{score} owner:{owner} controller: {controllerName}";  
 }  
 }  
}

## PlayersManager.cs

﻿using System.Collections.Generic;  
using System.Linq;  
using CommandsSystem.Commands;  
using Networking;  
using UnityEngine;  
  
namespace GameMode {  
 /// <summary>  
 /// Класс для хранения информации об игроках  
 /// </summary>  
 public static class PlayersManager {  
 /// <summary>  
 /// Список игроков в текущем матче  
 /// </summary>  
 public static List<Player> players = new List<Player>();  
  
 /// <summary>  
 /// Возвращает список игроков, отсортированный по очкам  
 /// </summary>  
 public static List<Player> playersSortedByScore {  
 get {  
 var players2 = players.ToList();  
 players2.Sort((player1, player2) =>  
 player2.score.CompareTo(player1.score));  
 return players2;  
 }  
 }  
   
 /// <summary>  
 /// Возвращает список игроков, отсортированный по суммрному количеству очков  
 /// </summary>  
 public static List<Player> playersSortedByTotalScore {  
 get {  
 var players2 = players.ToList();  
 players2.Sort((player1, player2) =>  
 player2.totalScore.CompareTo(player1.totalScore));  
 return players2;  
 }  
 }  
   
 /// <summary>  
 /// Главный игрок, которым упраляет человек  
 /// </summary>  
 public static Player mainPlayer;  
  
 /// <summary>  
 /// Количество игроков в игре  
 /// </summary>  
 public static int playersCount => players.Count;  
  
 /// <summary>  
 /// Возвращает игрока с заданным ID  
 /// </summary>  
 /// <param name="id">ID игрока</param>  
 /// <returns>Игрока с заданным ID</returns>  
 public static Player GetPlayerById(int id) {  
 for (int i = 0; i < players.Count; i++) {  
 if (players[i].id == id) {  
 return players[i];  
 }  
 }  
  
 return null;  
 }  
  
 /// <summary>  
 /// Начисляет очки игроку  
 /// </summary>  
 /// <param name="player">Персонаж игрока</param>  
 /// <param name="score">Количество очков</param>  
 public static void AddScoreToPlayer(GameObject player, int score) {  
 AddScoreToPlayer(player.GetComponent<PlayerStorage>().Player, score);  
 }   
   
 /// <summary>  
 /// Начисляет очки игроку  
 /// </summary>  
 /// <param name="player">Игрок</param>  
 /// <param name="score">Количество очков</param>  
 public static void AddScoreToPlayer(Player player, int score) {  
 CommandsHandler.gameRoom.RunSimpleCommand(new ChangePlayerScore(player.id, score), MessageFlags.IMPORTANT);  
 }  
  
 /// <summary>  
 /// Проверяет, является ли данный игрок главным  
 /// </summary>  
 /// <param name="player">Игрок</param>  
 /// <returns>true, если является главным. Иначе false</returns>  
 public static bool IsMainPlayer(Player player) {  
 return PlayersManager.mainPlayer != null && player.id == PlayersManager.mainPlayer.id;  
 }  
  
 /// <summary>  
 /// Сбрасывает значения переменных  
 /// </summary>  
 public static void Reset() {  
 players.Clear();  
 if (mainPlayer != null) {  
 mainPlayer.totalScore = 0;  
 players.Add(mainPlayer);  
 }  
 }  
 }  
}

## PlayerStorage.cs

﻿using TMPro;  
using UnityEngine;  
  
namespace GameMode {  
 /// <summary>  
 /// Компонента для хранения ссылки игрока внутри персонажа, которым он управляет  
 /// </summary>  
 public class PlayerStorage : MonoBehaviour {  
 /// <summary>  
 /// Переменная для хранения ссылки на игрока  
 /// </summary>  
 private Player \_player;  
  
 /// <summary>  
 /// Панель, на которой должно отображаться имя игрока  
 /// </summary>  
 public TextMeshProUGUI namePanel;  
   
 /// <summary>  
 /// Ссылка на игрока  
 /// </summary>  
 public Player Player {  
 get => \_player;  
 set {  
 \_player = value;  
 if (\_player.id == PlayersManager.mainPlayer.id) {  
 namePanel.text = $"<color=green>{\_player.name}</color>";  
 } else {  
 namePanel.text = $"<color=red>{\_player.name}</color>";  
 }  
 }  
 }  
 }  
}

## Bomb.cs

﻿using System;  
using System.Collections.Generic;  
using Character.HP;  
using CommandsSystem.Commands;  
using Networking;  
using RotaryHeart.Lib.PhysicsExtension;  
using UnityEngine;  
using UnityEngine.Jobs;  
using Object = System.Object;  
  
namespace GameMechanics {  
 /// <summary>  
 /// Класс для бобмы  
 /// </summary>  
 public class Bomb : MonoBehaviour {  
 /// <summary>  
 /// Урон от попадания  
 /// </summary>  
 public float damage;  
 /// <summary>  
 /// Сила взрыва  
 /// </summary>  
 public float explosionForce = 400;  
 /// <summary>  
 /// Сфера, внутри которой взрывается бобма  
 /// </summary>  
 public SphereCollider area;  
  
 /// <summary>  
 /// Должна ли бомба наносить урон игроку, который её создал (false, если должна)  
 /// </summary>  
 public bool noDamageToCreator = false;  
  
 /// <summary>  
 /// Радиус взрыва  
 /// </summary>  
 [NonSerialized]  
 public float radius;  
  
 /// <summary>  
 /// Инициализирует переменные  
 /// </summary>  
 public void Start() {  
 radius = transform.lossyScale.x \* area.radius;  
 }  
  
 /// <summary>  
 /// Синхронно взрывает бомбу  
 /// </summary>  
 public void Explode() {  
 int id = ObjectID.GetID(gameObject);  
 CommandsHandler.gameModeRoom.RunUniqCommand(new ExplodeBombCommand(id), UniqCodes.EXPLODE\_BOMB, id, MessageFlags.IMPORTANT);  
 }  
   
 /// <summary>  
 /// Взрывает бомбу локально  
 /// </summary>  
 public void RealExplode() {  
 var creatorId = ObjectID.GetCreator(gameObject);  
  
 // Physics.OverlapSphere(transform.position, radius);  
 var colliders = RotaryHeart.Lib.PhysicsExtension.Physics.OverlapSphere(transform.position, radius, PreviewCondition.Both, 2,  
 Color.red, Color.green);  
 HashSet<HPController> gameObjects = new HashSet<HPController>();  
 foreach (var collider in colliders) {  
  
 if (ObjectID.TryGetID(collider.gameObject, out var iid)) {  
 Vector3 force = collider.transform.position - transform.position;  
 float len = force.magnitude;  
 force = force.normalized \* explosionForce / Mathf.Sqrt(len);  
 CommandsHandler.gameModeRoom.RunSimpleCommand(new ApplyForceCommand(iid, force ), MessageFlags.NONE);  
 }  
   
 var hp = collider.GetComponent<HPController>();  
 if (hp == null) continue;  
 int hpid = ObjectID.GetID(hp.gameObject);  
 if (ObjectID.IsOwned(hpid)) {  
 if (noDamageToCreator && (hpid == creatorId ||  
 ObjectID.TryGetCreator(hpid, out var hpCreator) && hpCreator == creatorId))  
 continue;  
 gameObjects.Add(hp);  
 }  
 }  
  
 foreach (var hp in gameObjects) {  
   
 hp.TakeDamage(damage, DamageSource.Bomb(), true);  
 }  
 }  
 }  
}

## BombOnCollisionExploder.cs

﻿using System;  
using UnityEngine;  
  
namespace GameMechanics {  
 /// <summary>  
 /// Компонента для взрыва бомбы при столкновении с объектом  
 /// </summary>  
 [RequireComponent(typeof(Bomb))]  
 public class BombOnCollisionExploder : MonoBehaviour {  
 /// <summary>  
 /// Настраивает физику, чтобы бомба не сталкивалась с персонажем, создавшем её  
 /// </summary>  
 public void Start() {  
 var creator = ObjectID.GetCreator(gameObject);  
 if (creator != 0) {  
 // to prevent lags dont collide with player  
 foreach (var col1 in GetComponents<Collider>()) {  
 foreach (var col2 in ObjectID.GetObject(creator).GetComponents<Collider>()) {  
 Physics.IgnoreCollision(col1, col2);  
 }  
 }  
   
 }  
   
 }  
   
 /// <summary>  
 /// Взрывает бомбу. Автоматически вызывается Unity при столкновении с другим объектом  
 /// </summary>  
 /// <param name="other">Информация о столкновении</param>  
 public void OnCollisionEnter(Collision other) {  
 GetComponent<Bomb>().Explode();  
 }  
 }  
}

## BombTriggerHPExploder.cs

﻿using System;  
using Character.HP;  
using UnityEngine;  
  
namespace GameMechanics {  
 /// <summary>  
 /// Компонента для взрыва бомбы при столкновении с объектом, имеющим здоровье  
 /// </summary>  
 [RequireComponent(typeof(Bomb))]  
 public class BombTriggerHPExploder : MonoBehaviour {  
 /// <summary>  
 /// Взрывает бомбу, если объект имеет здоровье. Автоматически вызывается Unity при столкновении  
 /// </summary>  
 /// <param name="other">Коллайдер объекта, с которым столкнулась бомба</param>  
 private void OnTriggerEnter(Collider other) {  
 var hp = other.GetComponent<HPController>();  
 if (hp != null) {  
 GetComponent<Bomb>().Explode();  
 }  
 }  
 }  
}

## Coin.cs

﻿using CommandsSystem.Commands;  
using GameMode;  
using Networking;  
using UnityEngine;  
  
/// <summary>  
/// Класс для монетки  
/// </summary>  
public class Coin : MonoBehaviour {  
 /// <summary>  
 /// Время, когда последний раз была отправлена команда подобрать монетку на сервер  
 /// </summary>  
 private float picked = -100;  
  
 /// <summary>  
 /// Подбирает монетку при столкновении с игроком. Автоматически вызывается Unity  
 /// </summary>  
 /// <param name="other">Коллайдер объекта, с которым столкнулась монетка</param>  
 private void OnTriggerEnter(Collider other) {  
 if (Time.time - picked < 5) return;  
   
 if (other.CompareTag("Player")) {  
 picked = Time.time;  
 var command = new PickCoinCommand(ObjectID.GetID(other.gameObject), ObjectID.GetID(this.gameObject));  
   
 CommandsHandler.gameModeRoom.RunSimpleCommand(command, MessageFlags.IMPORTANT);  
 }  
 }  
  
}

## MovingPlatform.cs

﻿using CommandsSystem.Commands;  
using GameMode;  
using Networking;  
using UnityEngine;  
using Util2;  
  
namespace GameMechanics {  
   
 /// <summary>  
 /// Класс для подвижной платформы  
 /// </summary>  
 public class MovingPlatform : MonoBehaviour, IOwnedEventHandler {  
 /// <summary>  
 /// Позиция, между которой должна перемещаться платформа  
 /// </summary>  
 public Transform nextTransform;  
  
 /// <summary>  
 /// Предыдущая позиция платформы  
 /// </summary>  
 private Vector3 lastPosition;  
 /// <summary>  
 /// Следующая позиция  
 /// </summary>  
 private Vector3 nextPosition;  
   
 /// <summary>  
 /// Скорость платформы  
 /// </summary>  
 public float speed = 10f;  
 /// <summary>  
 /// Время, на которое платформа останавливается  
 /// </summary>  
 public float stayTime = 2;  
   
 /// <summary>  
 /// Состояние платформы  
 /// </summary>  
 public int state = STAY\_STATE;  
 /// <summary>  
 /// Состояние платформы, при котором она двигается  
 /// </summary>  
 private const int MOVE\_STATE = 0;  
 /// <summary>  
 /// Состояние платформы, при котором она стоит на месте  
 /// </summary>  
 private const int STAY\_STATE = 1;  
 /// <summary>  
 /// Состояние плафтформы, при котором она ожидает, когда ей дадут команду передвигаться  
 /// </summary>  
 private const int WAITING\_FOR\_COMMAND = 2;  
   
 /// <summary>  
 /// Направление, в котором двигается платформа  
 /// </summary>  
 public int direction = DIRECTION\_LAST\_TO\_NEXT;  
 /// <summary>  
 /// Направление от предыдушего к следующему  
 /// </summary>  
 private const int DIRECTION\_LAST\_TO\_NEXT = 0;  
 /// <summary>  
 /// Направление от следующего к предыдущему  
 /// </summary>  
 private const int DIRECTION\_NEXT\_TO\_LAST = 1;  
   
 /// <summary>  
 /// ID платформы  
 /// </summary>  
 private int id;  
  
 /// <summary>  
 /// Инициализирует переменные  
 /// </summary>  
 private void Start() {  
 lastPosition = transform.position;  
 nextPosition = nextTransform.position;  
 id = ObjectID.GetID(gameObject);  
 CommandsHandler.gameModeRoom.RunSimpleCommand(new TakeOwnCommand(id, sClient.ID),   
 MessageFlags.IMPORTANT);  
 }  
  
 /// <summary>  
 /// Обрабатывает событие, когда какой-либо объект попадает на платформу. Автоматически вызывается Unity при столкновении  
 /// </summary>  
 /// <param name="other">Информация о столкновении</param>  
 private void OnCollisionEnter(Collision other) {  
 if (other.gameObject.GetComponent<Rigidbody>() != null && other.transform.position.y > transform.position.y)  
 other.transform.parent = transform;  
 }  
  
 /// <summary>  
 /// Обрабатывет событие, когда объект уходит с платформы. Автоматически вызывается Unity  
 /// </summary>  
 /// <param name="other"></param>  
 private void OnCollisionExit(Collision other) {  
 if (other.transform.parent == transform)  
 other.transform.parent = null;  
 }  
  
 /// <summary>  
 /// Переключает плафторму в режим движения  
 /// </summary>  
 /// <param name="direction">Направление движения</param>  
 public void SetMoveState(int direction) {  
 if (direction != this.direction) {  
 gUtil.Swap(ref lastPosition, ref nextPosition);  
 }  
  
 if (direction == this.direction || state == MOVE\_STATE)  
 transform.position = lastPosition;  
 this.direction = direction;  
 this.state = MOVE\_STATE;  
 }  
  
 /// <summary>  
 /// Время, которое платформе осталось стоять на месте  
 /// </summary>  
 private float currentStayingTime = -100;  
   
 /// <summary>  
 /// Обновляет состояние платформы и перемещает её  
 /// </summary>  
 public void Update() {  
 if (id == 0) return;  
 if (state == MOVE\_STATE) {  
 transform.position = Vector3.MoveTowards(transform.position, nextPosition, speed\*Time.deltaTime);  
 if (transform.position == nextPosition) {  
 state = STAY\_STATE;  
 currentStayingTime = stayTime;  
 }  
 } else if (state == STAY\_STATE) {  
 currentStayingTime -= Time.deltaTime;  
 if (currentStayingTime < 0) {  
 if (ObjectID.IsOwned(id)) {  
 CommandsHandler.gameModeRoom.RunSimpleCommand(new SetPlatformStateCommand(id, 1 - direction),  
 MessageFlags.NONE);  
 currentStayingTime = 0.1f;  
 }  
  
 state = WAITING\_FOR\_COMMAND;  
 }  
 }  
   
 }  
  
 /// <summary>  
 /// Обрабатывает событие, когда у платформы появляется новый владелец  
 /// </summary>  
 /// <param name="owner">Новый владелец</param>  
 public void HandleOwnTaken(int owner) {  
 if (ObjectID.IsOwned(gameObject)) {  
 if (state == WAITING\_FOR\_COMMAND)  
 state = STAY\_STATE;  
 }  
 }  
 }  
}

## ParabolaFlyingObject.cs

﻿using System;  
using Interpolation;  
using UnityEngine;  
  
namespace GameMechanics {  
 /// <summary>  
 /// Компонента для объекта, летящего по тректории параболы  
 /// </summary>  
 public class ParabolaFlyingObject : MonoBehaviour {  
 /// <summary>  
 /// Стартовая позиция  
 /// </summary>  
 public Vector3 start;  
 /// <summary>  
 /// Средняя позиция  
 /// </summary>  
 public Vector3 medium;  
 /// <summary>  
 /// Конечная позиция  
 /// </summary>  
 public Vector3 stop;  
 /// <summary>  
 /// Время полёта  
 /// </summary>  
 public float totalTime;  
  
 /// <summary>  
 /// Время, когда начался полёт  
 /// </summary>  
 public float startTime = -1f;  
  
 /// <summary>  
 /// Инициализирует переменные  
 /// </summary>  
 public void Start() {  
 startTime = Time.time;  
 }  
  
 /// <summary>  
 /// Автоматически прекращет движение при столкновении объектом  
 /// </summary>  
 /// <param name="other">Информация о столкновении</param>  
 private void OnCollisionEnter(Collision other) {  
 Destroy(this);  
 }  
  
 /// <summary>  
 /// Перемещает объект. Автоматически вызывается Unity каждый кадр  
 /// </summary>  
 public void Update() {  
 float t = (Time.time - startTime) / totalTime;  
 if (t > 1)  
 t = 1;  
 transform.position = InterpolationFunctions.BezierCurve(start, medium, stop, t);  
 }  
 }  
}

## GameModeFunctions.cs

﻿using CommandsSystem.Commands;  
using Networking;  
using UnityEngine;  
using UnityEngine.Assertions;  
  
namespace GameMode {  
 /// <summary>  
 /// Класс с функциями для игровых режимов  
 /// </summary>  
 public static class GameModeFunctions {  
 /// <summary>  
 /// Создает персонажа в случайном месте  
 /// </summary>  
 /// <param name="playerId">ID игрока</param>  
 public static void SpawnPlayer(int playerId) {  
 var pos = FindPlaceForSpawn(1, 0.5f);  
 var rot = new Quaternion();  
 var id = ObjectID.RandomID;  
 var owner = sClient.ID;  
   
 CommandsHandler.gameModeRoom.RunSimpleCommand(new SpawnPlayerCommand(new SpawnPrefabCommand("Robot", pos, rot, id, owner, id),   
 playerId), MessageFlags.IMPORTANT);  
 }  
  
 /// <summary>  
 /// Создаёт персонажа для каждого из игроков  
 /// </summary>  
 public static void SpawnPlayers() {  
 foreach (var player in PlayersManager.players) {  
 if (player.owner == sClient.ID) {  
 SpawnPlayer(player.id);  
 }  
 }  
 }  
  
 /// <summary>  
 /// RaycastHit для внутреннего использования (нужен, чтобы уменьшить нагрузку на сборщик мусора)  
 /// </summary>  
 private static RaycastHit \_raycastHitInfo;  
   
 /// <summary>  
 /// Ищет место для создания объекта заданного размера  
 /// </summary>  
 /// <param name="height">Высота, на которой нужно создать объект</param>  
 /// <param name="radius">Радиус объекта</param>  
 /// <returns></returns>  
 public static Vector3 FindPlaceForSpawn(float height, float radius) {  
 int layerMask = 1 << 9;  
 layerMask = ~layerMask;  
 // (possible) TODO: reserve space on network before spawn   
 Vector3 pos1, pos2;  
 for (int iterCount = 0; ; iterCount++) {  
 pos1 = pos2 = Client.client.spawnPolygon.RandomPoint();  
 pos1.y = -3;  
 pos2.y = height;  
 if (iterCount++ >= 100) {  
 Debug.LogError($"Unable to find free place for object with height: {height:F2}, radius: {radius:F2}");  
 return new Vector3(0, height + Random.value \* 10, 0);  
 }  
 // Assert.IsTrue(iterCount++ < 100, $"Unable to find free place for object with height: {height:F2}, radius: {radius:F2}");  
 var intersections = Physics.OverlapCapsule(pos1, pos2, radius, layerMask, QueryTriggerInteraction.Ignore);  
 if (intersections.Length != 1) continue;  
 var b = intersections[0];  
 Ray ray = new Ray();  
 ray.direction = Vector3.down;  
  
 bool flag = true;  
 for (int x = -1; x <= 1; x += 2) {  
 for (int z = -1; z <= 1; z += 2) {  
 ray.origin = new Vector3(pos2.x + x \* radius, pos2.y + 10f, pos2.z + z \* radius);  
 if (!b.Raycast(ray, out \_raycastHitInfo, 100f)) {  
 flag = false;  
 break;  
 }  
 }  
 }  
 if (flag) break;  
 }  
 // capsules.Add(new CapsuleGizmos(pos1, pos2, radius));  
  
 return pos2;  
 }  
 }  
}

## IGameMode.cs

﻿namespace GameMode {  
 /// <summary>  
 /// Интерфейс для игрового режима  
 /// </summary>  
 public interface IGameMode {  
 /// <summary>  
 /// Обновляет состояние игрового режима  
 /// </summary>  
 /// <returns>false, если режим закончился. Иначе true</returns>  
 bool Update();  
 /// <summary>  
 /// Завершает выполнение игрового режима  
 /// </summary>  
 /// <returns>false, если режим закончился. true, если надо ещё подождать</returns>  
 bool Stop();  
 /// <summary>  
 /// Время в секундах, которое длится игровой режим   
 /// </summary>  
 float TimeLength { get; }  
 }  
}

## PickCoinsGameMode.cs

﻿using System;  
using CommandsSystem.Commands;  
using Events;  
using Networking;  
using UnityEngine;  
  
namespace GameMode {  
 /// <summary>  
 /// Класс для игрового режима со сбором монет  
 /// </summary>  
 public class PickCoinsGameMode : IGameMode {  
 /// <summary>  
 /// Перечисление состояний режима  
 /// </summary>  
 private enum STATE {  
 INIT,  
 UPDATE  
 }  
  
 /// <summary>  
 /// Текущее состояние режима  
 /// </summary>  
 private STATE state = STATE.INIT;  
  
 /// <summary>  
 /// Количество созданных монет  
 /// </summary>  
 public int coinsCount = 0;  
  
 /// <summary>  
 /// Создает монету в случайном месте  
 /// </summary>  
 /// <param name="num">Порядковый номер монеты (нужен чтобы не создать одну и ту же два раза)</param>  
 public void SpawnRandomCoin(int num) {  
 var position = GameModeFunctions.FindPlaceForSpawn(10, 1);  
 CommandsHandler.gameModeRoom.RunUniqCommand(new SpawnPrefabCommand("coin",  
 position, Quaternion.identity, ObjectID.RandomID, sClient.ID, 0),  
 UniqCodes.SPAWN\_COIN, num,  
 MessageFlags.IMPORTANT);  
 }  
  
  
 /// <summary>  
 /// Обновляет состояние игрового режима  
 /// </summary>  
 /// <returns>false, если режим закончился. Иначе true</returns>  
 public bool Update() {  
 switch (state) {  
 case STATE.INIT:  
 MainUIController.mainui.SetTask(" pick coin = <color=green>+1</color>");  
 MainUIController.mainui.gunsPanel.SetActive(false);  
 GameModeFunctions.SpawnPlayers();  
  
 EventsManager.handler.OnPlayerPickedUpCoin += (playerObj, coin) => {  
 SpawnRandomCoin(coinsCount++);  
 var player = playerObj.GetComponent<PlayerStorage>().Player;  
 if (player.owner == sClient.ID)  
 PlayersManager.AddScoreToPlayer(player, 1);  
 };  
  
 for (int i = 0; i < 20; i++) {  
 SpawnRandomCoin(coinsCount++);  
 }  
  
  
 state = STATE.UPDATE;  
 break;  
 case STATE.UPDATE:  
  
 break;  
 default:  
 throw new Exception($"Unknown state: {state}");  
 }  
  
 return true;  
 }  
  
 /// <summary>  
 /// Завершает выполнение игрового режима  
 /// </summary>  
 /// <returns>false, если режим закончился. true, если надо ещё подождать</returns>  
 public bool Stop() {  
 return false;  
 }  
   
 /// <summary>  
 /// Время в секундах, которое длится игровой режим   
 /// </summary>  
 public float TimeLength => 30;  
 }  
}

## ShooterGameMode.cs

﻿using System;  
using Character;  
using Character.Actions;  
using Character.Guns;  
using Character.HP;  
using CommandsSystem.Commands;  
using Events;  
using Networking;  
using UnityEngine;  
using UnityEngine.Animations;  
using Object = UnityEngine.Object;  
using Random = UnityEngine.Random;  
  
namespace GameMode {  
 /// <summary>  
 /// Класс для игрового режима в формате шутера  
 /// </summary>  
 public class ShooterGameMode : IGameMode {  
 /// <summary>  
 /// Перечисление состояния игрового режима  
 /// </summary>  
 private enum STATE {  
 INIT,  
 UPDATE  
 }  
  
 /// <summary>  
 /// Текущее состояние игрового режима  
 /// </summary>  
 private STATE state = STATE.INIT;  
  
 /// <summary>  
 /// Количество созданного на игровом поле оружия  
 /// </summary>  
 private int spawnedGunsCount = 0;  
 /// <summary>  
 /// Время, через которое нужно создать следующее оружие  
 /// </summary>  
 private float timeToSpawnNextGun = 0f;  
  
 /// <summary>  
 /// Создаёт случайное оружие на игровом поле  
 /// </summary>  
 /// <param name="num">Порядковый номер создаваемого оружия (нужен чтобы не создать одно и то же два раза)</param>  
 private void SpawnRandomGun(int num) {  
 var position = GameModeFunctions.FindPlaceForSpawn(0.1f, 1);  
   
 int gunType = Random.Range(0, 3);  
 string gunName;  
 switch (gunType) {  
 case 0:  
 gunName = "pistol";  
 break;  
 case 1:  
 gunName = "semiauto";  
 break;  
 case 2:  
 gunName = "shotgun";  
 break;  
 default:  
 throw new Exception("Unknown gun type");  
 }  
 CommandsHandler.gameModeRoom.RunUniqCommand(new SpawnPrefabCommand(gunName,  
 position, Quaternion.identity, ObjectID.RandomID, sClient.ID, 0),  
 UniqCodes.SPAWN\_GUN, num,  
 MessageFlags.IMPORTANT);  
 }  
  
 /// <summary>  
 /// Обновляет состояние игрового режима  
 /// </summary>  
 /// <returns>false, если режим закончился. Иначе true</returns>  
 public bool Update() {  
 switch (state) {  
 case STATE.INIT:  
 MainUIController.mainui.SetTask( " Kill enemy = <color=green>+100</color>\n" +  
 " Deal <color=red>1</color> damage = <color=green>+1</color>");  
 MainUIController.mainui.gunsPanel.SetActive(true);  
 GameModeFunctions.SpawnPlayers();  
 EventsManager.handler.OnSpawnedPlayer += (gameObject, player) => {  
 if (player.owner == sClient.ID) {  
 gameObject.GetComponent<ActionController>().SetAction<ShootPistolAction>(action =>   
 action.gun = new Pistol());  
 }  
 };  
  
 EventsManager.handler.OnObjectDead += (go, source) => {  
 if (!go.TryGetComponent<PlayerStorage>(out \_)) return; // check that it is player  
  
 var killedPlayer = go.GetComponent<PlayerStorage>().Player;  
   
 if (killedPlayer.owner == sClient.ID)  
 GameModeFunctions.SpawnPlayer(killedPlayer.id);  
 Client.client.RemoveObject(go);  
  
   
 UberDebug.LogChannel("GameMode", $"Shooter: player#{killedPlayer.id} '{killedPlayer.name}' dead. respawning");  
   
 var player = DamageSource.GetSourceGO(source)?.GetComponent<PlayerStorage>()?.Player;  
   
 if (player == null) return;  
 if (player.owner == sClient.ID)  
 PlayersManager.AddScoreToPlayer(player, 100);  
 };  
  
 EventsManager.handler.OnObjectChangedHP += (go, delta, source) => {  
 if (!go.TryGetComponent<PlayerStorage>(out \_)) return; // check that it is player  
   
   
 var player = DamageSource.GetSourceGO(source)?.GetComponent<PlayerStorage>()?.Player;  
 if (player == null) return;  
 if (player.owner == sClient.ID)  
 PlayersManager.AddScoreToPlayer(player, Mathf.RoundToInt(-delta));  
 };  
  
 for (int i = 0; i < 3; i++) {  
 SpawnRandomGun(spawnedGunsCount++);  
 }  
 timeToSpawnNextGun = 10f;  
   
 state = STATE.UPDATE;  
 break;  
 case STATE.UPDATE:  
 timeToSpawnNextGun -= Time.deltaTime;  
 if (timeToSpawnNextGun < 0) {  
 SpawnRandomGun(spawnedGunsCount++);  
 timeToSpawnNextGun = 10f;  
 }  
  
 break;  
 default:  
 throw new Exception($"Unknown state: {state}");  
 }  
   
 return true;  
 }  
  
 /// <summary>  
 /// Завершает выполнение игрового режима  
 /// </summary>  
 /// <returns>false, если режим закончился. true, если надо ещё подождать</returns>  
 public bool Stop() {  
 return false;  
 }  
  
 /// <summary>  
 /// Время в секундах, которое длится игровой режим   
 /// </summary>  
 public float TimeLength => 140;  
 }  
}

## InterpolationFunctions.cs

﻿using Character;  
using UnityEngine;  
  
namespace Interpolation {  
 /// <summary>  
 /// Класс с функциями для интерполяции  
 /// </summary>  
 public static class InterpolationFunctions {  
 /// <summary>  
 /// Вычисляет сплайн Эрмита  
 /// </summary>  
 /// <param name="start">Стартовая позиция</param>  
 /// <param name="stop">Конечная позиция</param>  
 /// <param name="m0">tan угла наклона в start</param>  
 /// <param name="m1">tan угла наклона в stop</param>  
 /// <param name="t">Коэф. интерполяции (от 0 до 1)</param>  
 /// <returns>Значение в сплайне Эрмита</returns>  
 public static float CubicHermiteSpline(float start, float stop, float m0, float m1, float t) {  
 return (2\*t\*t\*t - 3\*t\*t + 1) \* start + (t\*t\*t - 2\*t\*t + t)\*m0 + (-2\*t\*t\*t + 3\*t\*t) \* stop + (t\*t\*t - t\*t) \* m1;  
 }  
  
 /// <summary>  
 /// Вычисляет сплайн Эрмита для трёх точек  
 /// </summary>  
 /// <param name="p0">Предпредыдущая позиция</param>  
 /// <param name="p1">Предыдущая позиция</param>  
 /// <param name="p2">Следующая позиция</param>  
 /// <param name="dt01">Время между p0 и p1</param>  
 /// <param name="dt12">Время между p1 и p2</param>  
 /// <param name="t">Коэф. интерполяции (от 0 до 1)</param>  
 /// <returns>Значение в сплайне Эрмита</returns>  
 public static float CubicHermiteSpline3(float p0, float p1, float p2, float dt01, float dt12, float t) {  
 var m0 = (p1 - p0) / dt01;  
 var m1 = (p2 - p1) / dt12;  
 if (p0 + 0.01f >= p1 && p1 <= p2 + 0.01f) {  
 m0 = 0;  
 }  
  
 if (p0 <= p1 + 0.01f && p1 + 0.01f >= p2) {  
 m0 = 0;  
 }  
 /\*if (t > 1)  
 Debug.LogError("time is " + t);\*/  
 var res = CubicHermiteSpline(p1, p2, m0, m1, t);  
 /\* if (p1 <= p2 && res > p2)  
 return p2;  
 if (p1 >= p2 && res < p2)  
 return p2;\*/  
 return res;  
 }  
  
  
 /// <summary>  
 /// Интерполирует вектор  
 /// </summary>  
 /// <param name="p0">Предпредыдущий вектор</param>  
 /// <param name="p1">Предыдущий вектор</param>  
 /// <param name="p2">Следующий вектор</param>  
 /// <param name="dt01">Время между p0 и p1</param>  
 /// <param name="dt12">Время между p1 и p2</param>  
 /// <param name="t">Коэф. интерполяции (от 0 до 1)</param>  
 /// <returns>Интерполированный вектор</returns>  
 public static Vector3 Lerp3Points(Vector3 p0, Vector3 p1, Vector3 p2, float dt01, float dt12, float t) {  
 float x = CubicHermiteSpline3(p0.x, p1.x, p2.x, dt01, dt12, t);  
 float y = CubicHermiteSpline3(p0.y, p1.y, p2.y, dt01, dt12, t);  
 float z = CubicHermiteSpline3(p0.z, p1.z, p2.z, dt01, dt12, t);  
 return new Vector3(x, y, z);  
 }  
  
 /// <summary>  
 /// Интерполирет позицию между точкам  
 /// </summary>  
 /// <param name="lastlastPosition">Предпредыдущая позиция</param>  
 /// <param name="lastPosition">Предыдущая позиция</param>  
 /// <param name="nextPosition">Следующая позиция</param>  
 /// <param name="coef">Коэф. интерполяции (от 0 до 1)</param>  
 /// <returns>Интерполированную позицию</returns>  
 public static Vector3 InterpolatePosition(Vector3 lastlastPosition, Vector3 lastPosition, Vector3 nextPosition,   
 float coef) {  
 return Lerp3Points(lastlastPosition, lastPosition, nextPosition,  
 1, 1,  
 coef);/\*\*/ /\* \* Client.client.interpolationCoef  
 +\*/  
 /\* Vector3.LerpUnclamped(lastPosition, nextPosition, coef);/\*\*/// \* (1.0f - Client.client.interpolationCoef)/\*)\*/;  
 }  
  
 /// <summary>  
 /// Интерполирует поворот  
 /// </summary>  
 /// <param name="lastRotation">Предыдущий поворот</param>  
 /// <param name="nextRotation">Следующий поворот</param>  
 /// <param name="coef">Коэф. интерполяции (от 0 до 1)</param>  
 /// <returns>Интерполированный поворот</returns>  
 public static Quaternion InterpolateRotation(Quaternion lastRotation, Quaternion nextRotation, float coef) {  
 return Quaternion.Lerp(lastRotation, nextRotation, coef);  
 }  
  
 /// <summary>  
 /// Интерполирует анимацию персонажа  
 /// </summary>  
 /// <param name="last">Предыдущая анимация</param>  
 /// <param name="next">Следующая анимация</param>  
 /// <param name="coef">Коэф. интерполяции (от 0 до 1)</param>  
 /// <returns>Интерполированную анимацию</returns>  
 public static PlayerAnimationState InterpolatePlayerAnimationState(PlayerAnimationState last,  
 PlayerAnimationState next, float coef) {  
 bool idle = next.idle;//InterpolateBool(last.idle, next.idle, coef);  
 float speed = next.speed;//InterpolateFloat(last.speed, next.speed, coef);  
 float rotationSpeed = InterpolateFloat(last.speed, next.speed, coef);  
 return new PlayerAnimationState() {  
 idle = idle,  
 speed = speed,  
 rotationSpeed = rotationSpeed  
 };  
 }  
  
 /// <summary>  
 /// Интерполирует логическую переменную  
 /// </summary>  
 /// <param name="last">Предыдущее значение</param>  
 /// <param name="next">Следующее значение</param>  
 /// <param name="coef">Коэф. интерполяции (от 0 до 1)</param>  
 /// <returns>Интерполированную логическую переменную</returns>  
 public static bool InterpolateBool(bool last, bool next, float coef) {  
 if (last == next) return last;  
 if (coef < 0.5f) return last;  
 return next;  
 }  
  
 /// <summary>  
 /// Интерполирует число с плавающей запятой  
 /// </summary>  
 /// <param name="last"></param>  
 /// <param name="next"></param>  
 /// <param name="coef"></param>  
 /// <returns>Результат интерполяции</returns>  
 public static float InterpolateFloat(float last, float next, float coef) {  
 return (next - last) \* coef + last;  
 }  
  
 /// <summary>  
 /// Получает значение в кривой Безье  
 /// </summary>  
 /// <param name="p0">Первая точка</param>  
 /// <param name="p1">Вторая точка</param>  
 /// <param name="p2">Третья точка</param>  
 /// <param name="t">Коэффициент интерполяции (от 0 до 1)</param>  
 /// <returns>Значение в кривой Безье</returns>  
 public static float BezierCurve(float p0, float p1, float p2, float t) {  
 return (1 - t) \* (1 - t) \* p0 + 2 \* (1 - t) \* t \* p1 + t \* t \* p2;  
 }  
  
 /// <summary>  
 /// Получает координаты вектора в кривой Безье  
 /// </summary>  
 /// <param name="p0">Первая точка</param>  
 /// <param name="p1">Вторая точка</param>  
 /// <param name="p2">Третья точка</param>  
 /// <param name="t">Коэффициент интерполяции (от 0 до 1)</param>  
 /// <returns>Значение вектора в кривой Безье</returns>  
 public static Vector3 BezierCurve(Vector3 p0, Vector3 p1, Vector3 p2, float t) {  
 float x = BezierCurve(p0.x, p1.x, p2.x, t);  
 float y = BezierCurve(p0.y, p1.y, p2.y, t);  
 float z = BezierCurve(p0.z, p1.z, p2.z, t);  
 return new Vector3(x, y, z);  
 }  
 }  
}

## ManagedGameObject.cs

﻿using System;  
using CommandsSystem;  
using CommandsSystem.Commands;  
using Interpolation.Properties;  
using Networking;  
using UnityEngine;  
  
namespace Interpolation {  
 /// <summary>  
 /// Компонента для игрового объекта, управляемого из текущего клиента  
 /// </summary>  
 public class ManagedGameObject<T> : MonoBehaviour  
 where T: IGameObjectProperty, new() {  
 /// <summary>  
 /// Время, когда в последний раз был синхронищирован объект  
 /// </summary>  
 private float lastSendState = -1;  
   
 /// <summary>  
 /// Свойство объекта  
 /// </summary>  
 public T property;  
  
 /// <summary>  
 /// Период синхронизации состояния  
 /// </summary>  
 protected virtual float updateTime => 1f / sClient.NETWORK\_FPS;  
  
  
 /// <summary>  
 /// Инициализирует переменные  
 /// </summary>  
 public void Start() {  
 property = new T();  
 property.FromGameObject(gameObject);  
 }  
   
 /// <summary>  
 /// Отправляет состояние объекта, если нужно. Автоматически вызвается Unity каждый кадр  
 /// </summary>  
 void Update() {  
 float curTime = Time.time;  
   
 if (curTime - lastSendState > updateTime) {  
// Debug.Log("Sending coordianates " );  
 property.FromGameObject(gameObject);  
 ICommand command;  
 command = property.CreateChangedCommand(curTime - lastSendState);  
 // var command = property.GetCommand();  
 CommandsHandler.gameRoom.RunSimpleCommand(command, MessageFlags.NONE);  
 lastSendState = curTime;  
 }  
 }  
 }  
   
}

## UnmanagedGameObject.cs

﻿using System;  
using Interpolation.Properties;  
using UnityEngine;  
  
namespace Interpolation {  
 /// <summary>  
 /// Компонента для объекта, управляемого из другого клиента  
 /// </summary>  
 public class UnmanagedGameObject<T> : MonoBehaviour  
 where T : IGameObjectProperty, new() {  
  
 /// <summary>  
 /// Данные для синхронизации объекта  
 /// </summary>  
 private class Data {  
 /// <summary>  
 /// Состояние объекта  
 /// </summary>  
 public IGameObjectProperty s;  
 /// <summary>  
 /// Время, прошедшее между отправками ссостояний  
 /// </summary>  
 public float timeSinceLast;  
  
 public Data(IGameObjectProperty property, float timeSinceLast) {  
 this.s = property;  
 this.timeSinceLast = timeSinceLast;  
 }  
 }  
   
 /// <summary>  
 /// Период синхронизации состояния  
 /// </summary>  
 private float timePerFrame = 1f / sClient.NETWORK\_FPS;  
  
 /// <summary>  
 /// Состояния объекта  
 /// </summary>  
 private Data lastlastState, lastState, nextState;  
 /// <summary>  
 /// Текущее состояние объекта  
 /// </summary>  
 private IGameObjectProperty state;  
 /// <summary>  
 /// Состояние объекта после следующего  
 /// </summary>  
 private Data nextNextState;  
 /// <summary>  
 /// Время, когда было получено последнее обновление состояния  
 /// </summary>  
 private float lastMessageTime;  
  
 /// <summary>  
 /// Инициализирует переменные   
 /// </summary>  
 void Init() {  
 lastlastState = lastState = nextState = new Data(new T(), 1);  
 lastlastState.s.FromGameObject(gameObject);  
 state = new T();  
#if DEBUG\_INTERPOLATION   
 DebugGUI.SetGraphProperties("dx", "dx", -15f, 15f, 0, new Color(0, 1, 1), false);  
 DebugGUI.SetGraphProperties("x", "x", 5f, 25f, 1, new Color(0, 1, 1), true);  
#endif  
 }  
   
   
   
  
 /// <summary>  
 /// Коэффициент между предпредыдущим и предыдущим состоянием  
 /// </summary>  
 private float P0P1InterpolationCoef = 1;  
 /// <summary>  
 /// Коэффициент между предыдущим и следующим состоянием  
 /// </summary>  
 private float P1P2InterpolationCoef = 1;  
  
 /// <summary>  
 /// Переключается на следующее состояние  
 /// </summary>  
 private void SwitchToNextState() {  
 lastlastState = lastState;  
 lastState = nextState;  
 nextState = nextNextState;  
 nextNextState = null;  
   
   
 lastMessageTime = Time.realtimeSinceStartup;//-1f;  
 P0P1InterpolationCoef = P1P2InterpolationCoef;  
 }  
  
  
 /// <summary>  
 /// Устанавливает состояние объекта с плаынм переходом  
 /// </summary>  
 /// <param name="newState">Новое состояние</param>  
 /// <param name="deltaSinceLast">Время, прошедшее между отправками состояний</param>  
 public void SetStateAnimated(T newState, float deltaSinceLast) {  
 if (lastlastState is null) Init();  
 nextNextState = new Data(newState, deltaSinceLast);  
 }  
   
 /// <summary>  
 /// Текущее время интерполяции  
 /// </summary>  
 private float interpolationTime;  
  
 /// <summary>  
 /// Интерполирует и применяет состояние  
 /// </summary>  
 /// <param name="coef">Коэф. интерполяции</param>  
 private void Interpolate(float coef) {  
 var pos = transform.position.x;  
 state.Interpolate(lastlastState.s, lastState.s, nextState.s, coef);  
 state.ApplyToObject(gameObject);  
 // Debug.Log(transform.position.x - pos);  
#if DEBUG\_INTERPOLATION  
 DebugGUI.Graph("dx", (transform.position.x - pos) / Time.deltaTime);  
 DebugGUI.Graph("x", transform.position.x);  
#endif  
 }  
   
 /// <summary>  
 /// Анимирует состояние  
 /// </summary>  
 /// <param name="delta">Прошедшее количество времени</param>  
 private void Animate(float delta) {  
 /\* if (lastMessageTime < 0) lastMessageTime = Time.realtimeSinceStartup;/\* -   
 Math.Min(Time.deltaTime, timePerFrame)\*/; // endTime: lastMessageTime + timePerFrame  
 // (beginTime, endTime]  
  
 // var interpolationTime = Time.realtimeSinceStartup - lastMessageTime;  
 interpolationTime += delta;  
 float coef = interpolationTime / nextState.timeSinceLast;  
 if (interpolationTime > nextState.timeSinceLast) {  
   
 /\* state.Interpolate(lastlastState.s, lastState.s, nextState.s, 1f);  
 state.ApplyToObject(gameObject);\*/  
   
 // TODO adaptive correct  
 if (nextNextState == null) {  
 DebugExtension.DebugPoint(transform.position, Color.blue, 0.1f, 3);  
 var waitTime = Math.Round((interpolationTime - nextState.timeSinceLast) \* 1000);  
#if DEBUG\_INTERPOLATION   
 Debug.LogWarning($"Where is message? Waiting {waitTime} msec.");  
#endif  
  
 Interpolate(coef);  
   
 interpolationTime = nextState.timeSinceLast + 0.0000001f;  
   
 return;  
 }  
  
 interpolationTime -= nextState.timeSinceLast;  
 SwitchToNextState();  
 Animate(0);  
 return;  
 }  
  
  
   
 Interpolate(coef);  
 // P1P2InterpolationCoef = coef;  
 }  
  
 /// <summary>  
 /// Обновляет состояние. Автоматически вызывается Unity каждый кадр  
 /// </summary>  
 void Update() {  
 if (lastlastState is null) Init();  
 Animate(Time.deltaTime);  
 }  
 }  
}

## PlayerManagedGameObject.cs

﻿using CommandsSystem.Commands;  
  
namespace Interpolation.Managers {  
 /// <summary>  
 /// Компонента для персонажа, управляемого из текущего клиента  
 /// </summary>  
 public class PlayerManagedGameObject : ManagedGameObject<PlayerProperty> { }  
}

## PlayerUnmanagedGameObject.cs

﻿using CommandsSystem.Commands;  
  
namespace Interpolation.Managers {  
 /// <summary>  
 /// Компонента для персонажа, управляемого из другого клиента  
 /// </summary>  
 public class PlayerUnmanagedGameObject : UnmanagedGameObject<PlayerProperty> {  
 /// <summary>  
 /// Обрабатывает событие конца анимации.  
 /// Нужен, чтобы unity не кидал warning, что событие не было обработано  
 /// </summary>  
 public void pushEnd() {}  
 }  
}

## GameObjectProperty.cs

﻿using System;  
using CommandsSystem;  
using UnityEngine;  
  
namespace Interpolation.Properties {  
   
 /// <summary>  
 /// Базовый класс для состояния, которое можно синхронизировать по сети  
 /// </summary>  
 /// <typeparam name="T">Тип состояния</typeparam>  
 [Serializable]  
 public abstract class GameObjectProperty<T> : IGameObjectProperty  
 where T : GameObjectProperty<T>, new() {   
  
 /// <summary>  
 /// Копирует состояние из другого  
 /// </summary>  
 /// <param name="state">Состояние из которого нужно копировать</param>  
 public abstract void CopyFrom(T state);  
  
 /// <summary>  
 /// Копирует состояние из другого  
 /// </summary>  
 /// <param name="state">Состояние из которого нужно копировать</param>  
 public void CopyFrom(IGameObjectProperty state) {  
 CopyFrom(state as T);  
 }  
   
 /// <summary>  
 /// Интерполирует состояние между другими  
 /// </summary>  
 /// <param name="lastLastState">Предпредыдущее состояние</param>  
 /// <param name="lastState">Предыдущее состояние</param>  
 /// <param name="nextState">Следующее состояние</param>  
 /// <param name="coef">Коэффициент интерполяции между состояниями (от 0 до 1)</param>  
 public abstract void Interpolate(  
 T lastLastState,  
 T lastState,  
 T nextState,   
 float coef);  
   
 /// <summary>  
 /// Интерполирует состояние между другими  
 /// </summary>  
 /// <param name="lastLastState">Предпредыдущее состояние</param>  
 /// <param name="lastState">Предыдущее состояние</param>  
 /// <param name="nextState">Следующее состояние</param>  
 /// <param name="coef">Коэффициент интерполяции между состояниями (от 0 до 1)</param>  
 public void Interpolate(IGameObjectProperty lastLastState, IGameObjectProperty lastState, IGameObjectProperty nextState,  
 float coef) {  
 Interpolate(lastLastState as T, lastState as T, nextState as T, coef);  
 }  
  
 /// <summary>  
 /// Создаёт команду для отправки состояния другим клиентам  
 /// </summary>  
 /// <param name="deltaTime">Время, прошедшее с прошлой отправки</param>  
 /// <returns>Команду</returns>  
 public abstract ICommand CreateChangedCommand(float deltaTime);  
  
  
 /// <summary>  
 /// Получает состояние из объекта  
 /// </summary>  
 /// <param name="gameObject">Объект</param>  
 public abstract void FromGameObject(GameObject gameObject);  
   
 /// <summary>  
 /// Применяет состояние к объекту  
 /// </summary>  
 /// <param name="gameObject">Объект</param>  
 public abstract void ApplyToObject(GameObject gameObject);  
  
  
 }  
}

## IGameObjectProperty.cs

﻿using CommandsSystem;  
using UnityEngine;  
  
namespace Interpolation.Properties {  
 /// <summary>  
 /// Интерфейс для состояния, которое можно синхронизировать по сети  
 /// </summary>  
 public interface IGameObjectProperty {  
 /// <summary>  
 /// Копирует состояние из другого  
 /// </summary>  
 /// <param name="state">Состояние из которого нужно копировать</param>  
 void CopyFrom(IGameObjectProperty state);  
   
 /// <summary>  
 /// Получает состояние из объекта  
 /// </summary>  
 /// <param name="gameObject">Объект</param>  
 void FromGameObject(GameObject gameObject);  
   
 /// <summary>  
 /// Применяет состояние к объекту  
 /// </summary>  
 /// <param name="gameObject">Объект</param>  
 void ApplyToObject(GameObject gameObject);  
  
 /// <summary>  
 /// Интерполирует состояние между другими  
 /// </summary>  
 /// <param name="lastLastState">Предпредыдущее состояние</param>  
 /// <param name="lastState">Предыдущее состояние</param>  
 /// <param name="nextState">Следующее состояние</param>  
 /// <param name="coef">Коэффициент интерполяции между состояниями (от 0 до 1)</param>  
 void Interpolate(  
 IGameObjectProperty lastLastState,  
 IGameObjectProperty lastState,  
 IGameObjectProperty nextState,   
 float coef);  
  
 /// <summary>  
 /// Создаёт команду для отправки состояния другим клиентам  
 /// </summary>  
 /// <param name="deltaTime">Время, прошедшее с прошлой отправки</param>  
 /// <returns>Команду</returns>  
 ICommand CreateChangedCommand(float deltaTime);  
 }  
}

## PlayerProperty.cs

﻿using Character;  
using Interpolation;  
using Interpolation.Properties;  
using UnityEngine;  
  
namespace CommandsSystem.Commands {  
 /// <summary>  
 /// Состояние персонажа, которое можно синхронизировать по сети  
 /// </summary>  
 public class PlayerProperty : GameObjectProperty<PlayerProperty> {  
 /// <summary>  
 /// ID персонажа  
 /// </summary>  
 public int id;  
   
 /// <summary>  
 /// Позиция персонажа  
 /// </summary>  
 public Vector3 position;  
 /// <summary>  
 /// Поворот персонажа  
 /// </summary>  
 public Quaternion rotation;  
 /// <summary>  
 /// Состояние анимации персонажа  
 /// </summary>  
 public PlayerAnimationState animationState;  
  
 /// <summary>  
 /// Ссылка на CharacterAnimator персонажа  
 /// </summary>  
 private CharacterAnimator characterAnimator;  
   
 /// <summary>  
 /// Копирует состояние из другого  
 /// </summary>  
 /// <param name="state">Состояние из которого нужно копировать</param>  
 public override void CopyFrom(PlayerProperty state) {  
 id = state.id;  
 position = state.position;  
 rotation = state.rotation;  
 animationState.idle = state.animationState.idle;  
 animationState.speed = state.animationState.speed;  
 animationState.rotationSpeed = state.animationState.rotationSpeed;  
 }  
  
 /// <summary>  
 /// Получает состояние из объекта  
 /// </summary>  
 /// <param name="gameObject">Объект</param>  
 public override void FromGameObject(GameObject gameObject) {  
 id = ObjectID.GetID(gameObject);  
 position = gameObject.transform.position;  
 rotation = gameObject.transform.rotation;   
 if (characterAnimator is null) characterAnimator = gameObject.GetComponent<CharacterAnimator>();  
 animationState = characterAnimator.animationState;  
 }  
  
 /// <summary>  
 /// Применяет состояние к объекту  
 /// </summary>  
 /// <param name="gameObject">Объект</param>  
 public override void ApplyToObject(GameObject gameObject) {  
 gameObject.transform.position = position;  
 gameObject.transform.rotation = rotation;  
 if (characterAnimator is null) characterAnimator = gameObject.GetComponent<CharacterAnimator>();  
 characterAnimator.animationState = animationState;  
 }  
  
 /// <summary>  
 /// Создаёт команду для отправки состояния другим клиентам  
 /// </summary>  
 /// <param name="deltaTime">Время, прошедшее с прошлой отправки</param>  
 /// <returns>Команду</returns>  
 public override ICommand CreateChangedCommand(float deltaTime) {  
 return new ChangePlayerProperty(this, deltaTime);  
 }  
   
  
 /// <summary>  
 /// Интерполирует состояние между другими  
 /// </summary>  
 /// <param name="lastLastState">Предпредыдущее состояние</param>  
 /// <param name="lastState">Предыдущее состояние</param>  
 /// <param name="nextState">Следующее состояние</param>  
 /// <param name="coef">Коэффициент интерполяции между состояниями (от 0 до 1)</param>  
 public override void Interpolate(PlayerProperty lastLastState, PlayerProperty lastState, PlayerProperty nextState, float coef) {  
 position = InterpolationFunctions.InterpolatePosition(lastLastState.position, lastState.position, nextState.position, coef);  
 rotation = InterpolationFunctions.InterpolateRotation(lastState.rotation, nextState.rotation, coef);  
 animationState =  
 InterpolationFunctions.InterpolatePlayerAnimationState(lastState.animationState,  
 nextState.animationState, coef);  
 }  
   
   
 }  
   
   
   
  
   
}

## Request.cs

﻿using System;  
using LightJson;  
using Networking;  
using UnityEngine;  
  
namespace JsonRequest {  
 /// <summary>  
 /// Тип JSON запроса к серверу  
 /// </summary>  
 public enum RequestType {  
 GetMatchesList,  
 CreateMatch,  
 JoinMatch,  
 ChangeMatchState  
 }  
   
 /// <summary>  
 /// Класс для JSON запроса к серверу  
 /// </summary>  
 public class Request {  
 /// <summary>  
 /// Комната для запроса  
 /// </summary>  
 public int room;  
 /// <summary>  
 /// ID запроса  
 /// </summary>  
 public int id;  
 /// <summary>  
 /// Запрос в JSON формате  
 /// </summary>  
 public string json;  
  
 /// <summary>  
 /// Выполнен ли запрос  
 /// </summary>  
 public bool isCompleted = false;  
 /// <summary>  
 /// Действие, которое нужно сделать после получения ответа  
 /// </summary>  
 public Action<JsonValue> callback;  
   
 /// <summary>  
 /// Время в которое запрос будет признан невыполненным  
 /// </summary>  
 public float timeoutTime;  
 /// <summary>  
 /// Время, через которое запрос будет признан невыполненным  
 /// </summary>  
 public float timeout;  
 /// <summary>  
 /// Количество попыток повторить запрос  
 /// </summary>  
 public int retries;  
   
 /// <summary>  
 /// Конструктор JSON запроса  
 /// </summary>  
 /// <param name="room">ID комнаты для запроса</param>  
 /// <param name="type">Тип запроса</param>  
 /// <param name="json">Запрос в JSON формате</param>  
 /// <param name="callback">Действие, которое нужно сделать после получения ответа</param>  
 /// <param name="timeout">Время, через которое запрос будет признан невыполненным</param>  
 /// <param name="retries">Количество попыток повторить запрос</param>  
 public Request(int room, RequestType type, JsonValue json, Action<JsonValue> callback, float timeout=4, int retries=2) {  
 this.room = room;  
 json["\_id"] = this.id = ObjectID.RandomID;  
 json["\_type"] = type.ToString();  
 this.json = json.ToString();  
 this.callback = callback;  
 this.timeout = timeout;  
 this.retries = retries;  
 }  
  
 /// <summary>  
 /// Конструктор JSON запроса  
 /// </summary>  
 /// <param name="room">Комната для запроса</param>  
 /// <param name="type">Тип запроса</param>  
 /// <param name="json">Запрос в JSON формате</param>  
 /// <param name="callback">Действие, которое нужно сделать после получения ответа</param>  
 /// <param name="timeout">Время, через которое запрос будет признан невыполненным</param>  
 /// <param name="retries">Количество попыток повторить запрос</param>  
 public Request(ClientCommandsRoom room, RequestType type, JsonValue json, Action<JsonValue> callback, float timeout=4, int retries=2) :  
 this(room.roomID, type, json, callback, timeout, retries) {}  
   
 /// <summary>  
 /// Обновляет состояние запроса  
 /// </summary>  
 public void Update() {  
 if (Time.time > timeoutTime) {  
 if (retries <= 0) {  
 RequestsManager.openedRequests.Remove(id);  
 throw new Exception($"Timeout error for request#{id} to room {room} {json}");  
 }  
 retries--;  
 RequestsManager.Send(this);  
 }  
 }  
  
 /// <summary>  
 /// Обрабатывает получение ответа на запрос  
 /// </summary>  
 /// <param name="response">Ответ на запрос</param>  
 public void GotResponse(Response response) {  
 isCompleted = true;  
 RequestsManager.openedRequests.Remove(id);  
 callback(response.json);  
 }  
 }  
}

## RequestsManager.cs

﻿using System.Collections.Generic;  
using Networking;  
using UnityEngine;  
  
namespace JsonRequest {  
 /// <summary>  
 /// Класс для управления JSON запросами  
 /// </summary>  
 public static class RequestsManager {  
 /// <summary>  
 /// Словарь с запросами  
 /// </summary>  
 public static SortedDictionary<int, Request> openedRequests = new SortedDictionary<int, Request>();  
  
 /// <summary>  
 /// Обновляет состояния запросов  
 /// </summary>  
 public static void Update() {  
 foreach (var request in openedRequests.Values) {  
 request.Update();  
 }  
 }  
  
 /// <summary>  
 /// Отправляет запрос на сервер  
 /// </summary>  
 /// <param name="request">Запрос</param>  
 public static void Send(Request request) {  
 if (!openedRequests.ContainsKey(request.id))  
 openedRequests.Add(request.id, request);  
 request.timeoutTime = Time.time + request.timeout;  
 CommandsHandler.RoomById(request.room).RunJsonMessage(request.json);  
 }  
 }  
}

## Response.cs

﻿using System;  
using System.Text;  
using CommandsSystem;  
using Events;  
using Game;  
  
namespace JsonRequest {  
 /// <summary>  
 /// Класс, представляющий ответ сервера на JSON запрос  
 /// </summary>  
 public class Response : ICommand {  
 /// <summary>  
 /// ID запроса / ответа  
 /// </summary>  
 public int \_id;  
 /// <summary>  
 /// Ответ в JSON формате  
 /// </summary>  
 public LightJson.JsonValue json;  
  
 /// <summary>  
 /// Конструктор ответа на запрос  
 /// </summary>  
 /// <param name="\_id">ID ответа на запрос</param>  
 /// <param name="json">Тело ответа в формате JSON</param>  
 private Response(int \_id, LightJson.JsonValue json) {  
 this.\_id = \_id;  
 this.json = json;  
 }   
   
  
 /// <summary>  
 /// Десериализует ответ на запрос из бинарного сообщения  
 /// </summary>  
 /// <param name="arr">Массив с ответом на запрос</param>  
 /// <returns>Десериализованный ответ с свервера</returns>  
 public static Response Deserialize(byte[] arr) {  
 var jsonString = Encoding.UTF8.GetString(arr);  
 var json = LightJson.JsonValue.Parse(jsonString);  
 return new Response(json["\_id"].AsInteger, json);  
 }  
  
 /// <summary>  
 /// Обрабатывает ответ с сервера  
 /// </summary>  
 public void Run() {  
 switch (\_id) {  
 case -1: // changed match players  
 MatchesManager.HandleJsonMatchChanged(json);  
 break;  
 default:  
 RequestsManager.openedRequests[\_id].GotResponse(this);  
 break;  
 }  
 }  
  
 /// <summary>  
 /// Выводит ответ с сервера в строку  
 /// </summary>  
 /// <returns>Ответ с сервера в виде строки</returns>  
 public override string ToString() {  
 return "JSON: " + json.ToString();  
 }  
 }  
}

## ClientCommandsRoom.cs

﻿using System;  
using System.Collections.Generic;  
using System.Collections.Specialized;  
using CommandsSystem;  
using CommandsSystem.Commands;  
using Game;  
using UnityEngine;  
  
namespace Networking {  
 /// <summary>  
 /// Перчисление с кодами для уникальных команд  
 /// </summary>  
 public static class UniqCodes {  
 public static int PICK\_UP\_COIN = 0;  
 public static int PICK\_UP\_GUN = 1;  
 public static int SPAWN\_COIN = 2;  
 public static int SPAWN\_GUN = 3;  
  
 public static int ADD\_AI\_PLAYER = 4;  
 public static int CHOOSE\_GAMEMODE = 5;  
 public static int EXPLODE\_BOMB = 6;  
 public static int START\_GAME = 7;  
  
 public static int SPAWN\_PISTOL = 8;  
 public static int SPAWN\_SEMIAUTO = 9;  
 public static int SPAWN\_SHOTGUN = 10;  
 public static int SPAWN\_BOMBGUN = 11;  
 }  
   
 /// <summary>  
 /// Класс комнаты для общения с сервером  
 /// </summary>  
 public class ClientCommandsRoom {  
 /// <summary>  
 /// Номер последней обработанной команды  
 /// </summary>  
 private int lastMessage = -1;  
 /// <summary>  
 /// Слоаврь с потерянными сообщениями  
 /// </summary>  
 private OrderedDictionary losedMessages = new OrderedDictionary();  
 /// <summary>  
 /// CommandsSystem для кодирования сообщений  
 /// </summary>  
 private CommandsSystem.CommandsSystem commandsSystem = new CommandsSystem.CommandsSystem();  
 /// <summary>  
 /// Когда последний раз был отправлен запрос на переотправку команды  
 /// </summary>  
 private float lastTimeRequestSended = Time.time;  
  
 /// <summary>  
 /// ID комнаты  
 /// </summary>  
 public int roomID;  
 /// <summary>  
 /// Конструктор комнаты  
 /// </summary>  
 /// <param name="roomID">ID комнаты</param>  
 /// <param name="alreadyJoined">true, если клиент уже приосоединился к комнате</param>  
 public ClientCommandsRoom(int roomID, bool alreadyJoined=false) {  
 this.roomID = roomID;  
 if (!alreadyJoined)  
 RunJoinMessage();  
 RunAskMessage(0, -1, MessageFlags.NONE); // MessageFlags.SEND\_ONLY\_IMPORTANT);  
 }  
  
  
 /// <summary>  
 /// Деструктор комнаты. Отправляет команду выхода из комнаты  
 /// </summary>  
 ~ClientCommandsRoom() {  
 RunLeaveMessage();  
 }  
  
 /// <summary>  
 /// Отправляет простую команду на сервер  
 /// </summary>  
 /// <param name="command">Команда</param>  
 /// <param name="flags">Флаги</param>  
 public void RunSimpleCommand(ICommand command, MessageFlags flags) {  
 var data = commandsSystem.EncodeSimpleCommand(command, roomID, flags);  
 CommandsHandler.webSocketHandler.clientToServerMessages.Enqueue(data);  
 if (flags.HasFlag(MessageFlags.IMPORTANT))  
 UberDebug.LogChannel("SendCommand", "room#" +roomID+$" SimpleCommand {command} {flags}");  
 }  
  
 /// <summary>  
 /// Отправляет уникальную команду на сервер  
 /// </summary>  
 /// <param name="command">Команда</param>  
 /// <param name="i1">Первая часть уникального кода</param>  
 /// <param name="i2">Вторая часть уникального кода</param>  
 /// <param name="flags">Флаги</param>  
 public void RunUniqCommand(ICommand command, int i1, int i2, MessageFlags flags) {  
 var data = commandsSystem.EncodeUniqCommand(command, roomID, i1, i2, flags);  
 CommandsHandler.webSocketHandler.clientToServerMessages.Enqueue(data);  
 if (flags.HasFlag(MessageFlags.IMPORTANT))  
 UberDebug.LogChannel("SendCommand", "room#" +roomID+$" UniqCommand {command} {i1} {i2} {flags}");  
 }  
  
 /// <summary>  
 /// Отправляет JSON-запрос на сервер  
 /// </summary>  
 /// <param name="json">JSON строка</param>  
 /// <param name="flags">Флаги</param>  
 public void RunJsonMessage(string json, MessageFlags flags=MessageFlags.NONE) {  
 var data = commandsSystem.EncodeJsonMessage(json, roomID, MessageFlags.NONE);  
 CommandsHandler.webSocketHandler.clientToServerMessages.Enqueue(data);  
 UberDebug.LogChannel("SendCommand", "room#" +roomID+$" JsonMessage {json} {flags}");  
 }  
   
 /// <summary>  
 /// Отправляет команду с запросом сообщений на сервер  
 /// </summary>  
 /// <param name="firstIndex">Номер первого сообщения, которое нужно переотправить</param>  
 /// <param name="lastIndex">Номер последнего сообщения</param>  
 /// <param name="flags">Флаги</param>  
 private void RunAskMessage(int firstIndex, int lastIndex, MessageFlags flags) {  
 var data = commandsSystem.EncodeAskMessage(roomID, firstIndex, lastIndex, flags);  
 CommandsHandler.webSocketHandler.clientToServerMessages.Enqueue(data);  
 UberDebug.LogChannel("SendCommand", "room#" +roomID+$" AskMessage {firstIndex} {lastIndex} {flags}");  
 }  
  
 /// <summary>  
 /// Отправляет команду присоединения к комнате  
 /// </summary>  
 private void RunJoinMessage() {  
 var data = commandsSystem.EncodeJoinGameRoomMessage(roomID, MessageFlags.NONE);  
 CommandsHandler.webSocketHandler.clientToServerMessages.Enqueue(data);  
 UberDebug.LogChannel("Client", $"JoinRoom {roomID}");  
 }  
  
 /// <summary>  
 /// Отправляет команду покидания комнаты  
 /// </summary>  
 private void RunLeaveMessage() {  
 var data = commandsSystem.EncodeLeaveGameRoomMessage(roomID, MessageFlags.NONE);  
 CommandsHandler.webSocketHandler.clientToServerMessages.Enqueue(data);  
 UberDebug.LogChannel("Client", $"LeaveMessage {roomID}");  
 }  
  
 /// <summary>  
 /// Выполняет команду  
 /// </summary>  
 /// <param name="command">Команда</param>  
 private void ReceiveCommand(ICommand command) {  
 if (command == null) return;  
  
 if (command is ChangePlayerProperty || command is DrawPositionTracerCommand ||  
 command is DrawTargetedTracerCommand || command is SetPlatformStateCommand) { } else {  
 UberDebug.LogChannel("ReceiveCommand", command.ToString());  
 }  
 //try {  
 command.Run();  
 /\*} catch (Exception e) {  
 Debug.LogException(e);  
 }\*/  
 }  
  
 /// <summary>  
 /// Обрабатывает полученную команду  
 /// </summary>  
 /// <param name="commandId">Номер команды</param>  
 /// <param name="command">Команда</param>  
 public void HandleCommand(int commandId, ICommand command) {  
 if (commandId == -1) {  
 ReceiveCommand(command);  
 return;  
 }  
   
 if (commandId <= lastMessage || losedMessages.Contains(commandId)) {  
 UberDebug.LogWarningChannel("ReceiveCommand", "Got message twice. " + command.ToString());  
 return;  
 }  
  
  
  
 if (commandId == lastMessage + 1) {  
 lastMessage++;  
 ReceiveCommand(command);  
 } else {  
 losedMessages.Add(commandId, command);  
 }  
   
 while (losedMessages.Contains(lastMessage + 1)) {  
 lastMessage++;  
 var c = ((ICommand) losedMessages[(object) lastMessage]);  
 ReceiveCommand(c);  
 losedMessages.Remove(lastMessage);  
 }  
   
   
 if (losedMessages.Count != 0 && Time.time - lastTimeRequestSended > 2f) {  
 lastTimeRequestSended = Time.time;  
  
 var enumerator = losedMessages.GetEnumerator();  
 enumerator.MoveNext();  
 int currentId = (int) enumerator.Key;  
  
 RunAskMessage(lastMessage + 1, currentId - 1, MessageFlags.NONE);  
   
 Debug.LogWarning($"Server loosed messages from {lastMessage + 1} to {currentId - 1}");  
 }  
 }  
 }  
   
   
}

## CommandsHandler.cs

﻿using System;  
using CommandsSystem;  
  
namespace Networking {  
 /// <summary>  
 /// Класс для обработки команд с сервера  
 /// </summary>  
 public static class CommandsHandler {  
 /// <summary>  
 /// Обработчик WebSocket  
 /// </summary>  
 public static WebSocketHandler webSocketHandler = new WebSocketHandler();  
 /// <summary>  
 /// CommandsSystem для сериализации команд  
 /// </summary>  
 private static CommandsSystem.CommandsSystem commandsSystem = new CommandsSystem.CommandsSystem();  
  
  
 /// <summary>  
 /// Комната для поиска матча  
 /// </summary>  
 public static ClientCommandsRoom matchmakingRoom;  
  
 /// <summary>  
 /// Комната для матча (игры)  
 /// </summary>  
 public static ClientCommandsRoom gameRoom;  
 /// <summary>  
 /// Комната для игрового режима  
 /// </summary>  
 public static ClientCommandsRoom gameModeRoom;  
   
  
 /// <summary>  
 /// Сбрасывает значения переменных  
 /// </summary>  
 public static void Reset() {  
 matchmakingRoom = gameRoom = gameModeRoom = null;  
 }  
  
 /// <summary>  
 /// Полчает комнату по ID  
 /// </summary>  
 /// <param name="id">ID комнаты</param>  
 /// <returns>Комнату</returns>  
 public static ClientCommandsRoom RoomById(int id) {  
 if (matchmakingRoom?.roomID == id)  
 return matchmakingRoom;  
 if (gameRoom?.roomID == id)  
 return gameRoom;  
 if (gameModeRoom?.roomID == id)  
 return gameModeRoom;  
 return null;  
 }  
  
 /// <summary>  
 /// Получает и отправляет команды  
 /// </summary>  
 public static void Update() {  
 webSocketHandler.Update();  
 byte[] data;  
   
 while (CommandsHandler.webSocketHandler.serverToClientMessages.TryDequeue(out data)) {  
 int commandId, roomId;  
 ICommand command = commandsSystem.DecodeCommand(data, out commandId, out roomId);  
 var room = RoomById(roomId);  
 if (room != null) {  
 room.HandleCommand(commandId, command);  
 } else {  
 throw new Exception($"unhandled command to room {roomId}");  
 }  
 }  
 }  
  
 /// <summary>  
 /// Отключается от сервера  
 /// </summary>  
 public static void Stop() {  
 webSocketHandler?.Stop();  
 }  
 }  
}

## WebSocketHandler.cs

﻿using System.Collections.Concurrent;  
using System.Collections.Generic;  
using UnityEngine;  
using System.Threading.Tasks;  
using NativeWebSocket;  
  
/// <summary>  
/// Класс с дополнительными функциями для очериди  
/// </summary>  
public static class QueueExtension {  
 /// <summary>  
 /// Пытается достать элемент из очереди  
 /// </summary>  
 /// <param name="queue">Очередь</param>  
 /// <param name="res">Элемент</param>  
 /// <typeparam name="T">Тип элемента</typeparam>  
 /// <returns>true, если очердь была не пустая. Иначе false</returns>  
 public static bool TryDequeue<T>(this Queue<T> queue, out T res) {  
 if (queue.Count == 0) {  
 res = default(T);  
 return false;  
 }  
  
 res = queue.Dequeue();  
 return true;  
 }  
}  
  
/// <summary>  
/// Класс для работы с WebSocket  
/// </summary>  
public class WebSocketHandler {  
 /// <summary>  
 /// Очередь сообщений с клиента на сервер  
 /// </summary>  
 public Queue<byte[]> clientToServerMessages = new Queue<byte[]>();  
 /// <summary>  
 /// Очередь сообщений с сервера на клиент  
 /// </summary>  
 public Queue<byte[]> serverToClientMessages = new Queue<byte[]>();  
   
 /// <summary>  
 /// Ассинхронная задача подключиться к серверу  
 /// </summary>  
 private Task<WebSocket> connectTask;  
  
 /// <summary>  
 /// Асинхронная задача отправить сообщение на сервер  
 /// </summary>  
 private Task sendTask = Task.CompletedTask;  
   
 /// <summary>  
 /// Вебсокет для общения с сервером  
 /// </summary>  
 private WebSocket webSocket;  
  
 /// <summary>  
 /// Отключается от сервера  
 /// </summary>  
 public void Stop() {  
 Debug.Log("CLIENT: WebSocket closed");  
 webSocket?.Close();  
 }  
   
 /// <summary>  
 /// Обновляет состояние вебсокета  
 /// </summary>  
 public void Update() {  
 if (webSocket is null || webSocket.State == WebSocketState.Closed) {  
 if (connectTask is null) {  
 connectTask = CreateWebSocket();  
 }  
  
 if (connectTask.IsCompleted) {  
 webSocket = connectTask.Result;  
 connectTask = null;  
 }  
  
 return;  
 }  
   
 byte[] commands;  
 while (sendTask.IsCompleted && clientToServerMessages.TryDequeue(out commands)) {  
 /\* #if UNITY\_EDITOR  
 Thread.Sleep(60);  
 #endif \*/  
 var res = "";  
 for (int i = 0; i < commands.Length; i++) {  
 res += commands[i] +" ";  
 }  
   
 sendTask = webSocket.Send(commands);  
 }  
 }  
  
 /// <summary>  
 /// Создаёт вебсокет  
 /// </summary>  
 /// <returns>Асинхронную задачу создания вебсокета</returns>  
 private async Task<WebSocket> CreateWebSocket() {  
 Debug.Log("CLIENT: Connecting");  
 var webSocket = new WebSocket("ws://{host}/ws");  
 lock (webSocket)  
 {  
 webSocket.OnOpen += () => { Debug.LogWarning("CLIENT: connected"); };  
 webSocket.OnClose += e =>  
 {  
 Debug.LogWarning("CLIENT: disconnected. " + e);  
 };  
 webSocket.OnMessage += HandleWebSocketMessage;  
 webSocket.OnError += msg => { Debug.LogError("CLIENT: Websocket error. " + msg); };  
 }  
  
  
 await webSocket.Connect();  
 return webSocket;  
 }  
  
  
  
 /// <summary>  
 /// Обрабатывает сообщение, пришедшее в вебсокет  
 /// </summary>  
 /// <param name="data">Массив байт с данными</param>  
 private void HandleWebSocketMessage(byte[] data)  
 {/\*  
#if UNITY\_EDITOR  
 Thread.Sleep(60);  
#endif\*/  
 serverToClientMessages.Enqueue(data);  
// Debug.Log("" + data.Length); //data[0] + data[1] + data[2] + data[3] + data[4]);  
 }  
   
   
   
   
  
  
}

## Server.cs

﻿using UnityEngine;  
using System;  
/////  
///// Message formats:  
/////  
///// Client to Server  
///// 1: simple message  
///// - 1 byte message type + 4 byte room number + 1 flags + data  
///// - send to all clients  
///// 2: uniq message  
///// - 1 byte message type + 4 byte room number + 1 flags + 8 byte uniq code(4 byte code type, 4 byte code num) + data  
///// - if uniq code is new, message will be send to all clients. otherwise, message will be discarded  
///// 3: ask for new messages  
///// - 1 byte message type + 4 byte room number + 1 flags + 4 byte first message + 4 byte last message  
///// 4: join game room. if room does not exists, it must be created  
///// - 1 byte message type + 4 byte room number + 1 flags  
///// - optional: autoremove player if he not doing anything in room for some time (ping - pong)  
///// 5: leave game room. if last player leaved room, it must be deleted;  
///// - 1 byte message type + 4 byte room number + 1 flags  
/////   
///// Server to Client:  
///// - 4 byte message id + 4 byte room number + message data  
///// if message id == -1 means client need to do command instantly  
///// - optional ping: 4 byte message id + 4 byte data  
/////   
/////  
  
/// <summary>  
/// Перечиcления со видами сообщений, которые можно отправить на сервер  
/// </summary>  
public enum MessageType : byte {  
 SimpleMessage = 1,  
 UniqMessage = 2,  
 AskMessage = 3,  
 JoinGameRoom = 4,  
 LeaveGameRoom = 5,  
 JSON = 6,  
};  
  
/// <summary>  
/// Флаги для отправки сообщений на сервер  
/// </summary>  
[Flags]   
public enum MessageFlags : byte {  
 NONE = 0,  
 IMPORTANT = 1 << 0,  
 SEND\_ONLY\_IMPORTANT = 1 << 1  
}

## AutoRotateToCamera.cs

﻿using UnityEngine;  
  
/// <summary>  
/// Компонента для автоматического поворота интерфейса к камере  
/// </summary>  
public class AutoRotateToCamera : MonoBehaviour {  
  
 /// <summary>  
 /// Поворачивает объект к камере. Автоматически вызывается Unity каждый кадр  
 /// </summary>  
 private void LateUpdate() {  
 transform.LookAt(transform.position + Camera.main.transform.forward);  
 }  
}

## DebugUI.cs

﻿using System;  
using UnityEngine;  
using UnityEngine.UI;  
  
namespace UI {  
 /// <summary>  
 /// Компонента отладочного интерфейса  
 /// </summary>  
 public class DebugUI : MonoBehaviour {  
 /// <summary>  
 /// Элемент интерфейса для отладочного текста  
 /// </summary>  
 public Text textPanel;  
   
 /// <summary>  
 /// Флаг, сообщающий, что нужно перерисовать текст  
 /// </summary>  
 private static bool debugTextDirty = true;  
   
 /// <summary>  
 /// Массив с отладочным текстом  
 /// </summary>  
 private static string[] \_debugText = new string[10];  
  
 /// <summary>  
 /// Отладочный текст  
 /// </summary>  
 public static string[] debugText {  
 get {  
 debugTextDirty = true;  
 return \_debugText;  
 }  
 }  
  
 /// <summary>  
 /// Инициализирует переменные  
 /// </summary>  
 private void Start() {  
 debugText[0] = "DEBUG MODE";  
 }  
  
 /// <summary>  
 /// Перерисовывает отладочный текст, если требуется. Автоматически вызывается Unity каждый кадр  
 /// </summary>  
 private void Update() {  
 if (debugTextDirty) {  
 textPanel.text = String.Join("\n", debugText);  
 debugTextDirty = false;  
 }  
 }  
 }  
}

## MainUIController.cs

﻿using System;  
using System.Collections;  
using System.Collections.Generic;  
using System.Text;  
using Character.Guns;  
using CommandsSystem.Commands;  
using Events;  
using GameMode;  
using Networking;  
using TMPro;  
using UI;  
using UnityEngine;  
using UnityEngine.UI;  
using Object = UnityEngine.Object;  
  
/// <summary>  
/// Компонента, управляющая главным интерфейсом в игре  
/// </summary>  
public class MainUIController : MonoBehaviour {  
 /// <summary>  
 /// Переменная для хранения ссылки на MainUIController  
 /// </summary>  
 private static MainUIController \_mainui;  
 /// <summary>  
 /// Был ли создан главный интерфейс  
 /// </summary>  
 private static bool spawned = false;  
 /// <summary>  
 /// Ссылка на MainUIController. Автоматически создаёт интфрейс, если его нет на сцене  
 /// </summary>  
 public static MainUIController mainui {  
 get {  
 if (\_mainui != null) return \_mainui;  
   
 \_mainui = FindObjectOfType<MainUIController>();  
 if (\_mainui != null) return \_mainui;  
 if (spawned) return null; // means unity destroying scene  
   
 var go = Client.client.SpawnPrefab("MainUI");  
 \_mainui = go.GetComponent<MainUIController>();  
  
 return \_mainui;  
 }  
 }  
  
 /// <summary>  
 /// Элемент интерфейса с картинкой текущего оружия  
 /// </summary>  
 public Image gunImage;  
  
 /// <summary>  
 /// Картинки оружия  
 /// </summary>  
 public Sprite pistolSprite, shotgunSprite, semiautoSprite, grenadeLauncherSprite;  
  
 /// <summary>  
 /// Элемент интерфейса, показывающий оставшееся количество патронов  
 /// </summary>  
 public MultiImagePanel bulletsPanel;  
 /// <summary>  
 /// Элемент интерфейса, показывающий оставшееся количество магазинов  
 /// </summary>  
 public MultiImagePanel magazinesPanel;  
  
 /// <summary>  
 /// Панель интфрейса, показывающая информацию об оружии  
 /// </summary>  
 public GameObject gunsPanel;  
   
 /// <summary>  
 /// Текстовый элемент интерфейса, показывающий очки игроков в текущей игре  
 /// </summary>  
 public TextMeshProUGUI scoreText;  
 /// <summary>  
 /// Текстовый элемент интерфейса, показывающий задачу в текущей игре  
 /// </summary>  
 public TextMeshProUGUI taskText;  
 /// <summary>  
 /// Текстовый элемент интерфейса, показывающий время до конца текущей игры  
 /// </summary>  
 public TextMeshProUGUI timerText;  
  
 /// <summary>  
 /// Панель интерфейса, показывающая результаты игры  
 /// </summary>  
 public GameObject totalScorePanel;  
 /// <summary>  
 /// Текст с результатми игры  
 /// </summary>  
 public TextMeshProUGUI totalScoreText;  
 /// <summary>  
 /// Кнопка выхода из игры  
 /// </summary>  
 public Button exitButton;  
  
 /// <summary>  
 /// Панель с чатом  
 /// </summary>  
 public GameObject chatPanel;  
 /// <summary>  
 /// Поле для ввода сообщения в чат  
 /// </summary>  
 public TMP\_InputField chatInput;  
 /// <summary>  
 /// Текстовый элемент интфрейса, показывающий сообщения в чате  
 /// </summary>  
 public TextMeshProUGUI chatText;  
   
 /// <summary>  
 /// Список с сообщениями в чате  
 /// </summary>  
 public List<string> chatMessages = new List<string>();  
  
 /// <summary>  
 /// Возвращает цвет, в который нужно раскрашивать ник данного игрока  
 /// </summary>  
 /// <param name="player">Игрок</param>  
 /// <returns>Строка с названием цвета</returns>  
 public string ColorForPlayer(Player player) {  
 return PlayersManager.IsMainPlayer(player) ? "green" : "red";  
 }  
   
 /// <summary>  
 /// Перерисовывает текст с очками игроков  
 /// </summary>  
 private void RedrawScore() {  
 var text = new StringBuilder();  
 text.AppendLine("<size=130%><align=center>Score</align></size>");  
 foreach (var player in PlayersManager.players) {  
 text.AppendLine($"<color={ColorForPlayer(player)}> {player.name} <pos=65%>{player.score}</color>");  
 }  
  
 scoreText.text = text.ToString();  
 }  
  
   
 /// <summary>  
 /// Текст для результатов игры, в который нужно подставить оставшееся время  
 /// </summary>  
 private string totalScoreTextUnformatted = "{}";  
  
 /// <summary>  
 /// Добавляет StringBuilder таблицу с резльтатми игры  
 /// </summary>  
 /// <param name="text">StringBuilder</param>  
 private void AddScoreTable(StringBuilder text) {  
 text.AppendLine("<size=110%>Player <pos=23%>Score in last game <pos=65%>Total score</size>");  
 text.AppendLine();  
 foreach (var player in PlayersManager.playersSortedByScore) {  
 text.AppendLine($"<color={ColorForPlayer(player)}> {player.name}<pos=23%> {player.score} " +  
 $"<pos=65%>{player.totalScore}(+{PlayersManager.playersCount - player.placeInLastGame + 1})</color>");  
 }  
 }  
   
 /// <summary>  
 /// Показывает таблицу с результатами игры  
 /// </summary>  
 /// <param name="gamesRemaining">Количество игр, которое осталось сыграть</param>  
 /// <param name="timeRemaining">Время, через которое начнётся следующий игровой режим</param>  
 public void ShowTotalScore(int gamesRemaining, int timeRemaining) {  
 totalScorePanel.SetActive(true);  
 exitButton.gameObject.SetActive(false);  
   
 var text = new StringBuilder();  
 text.AppendLine($"<size=130%> Games Remaining: {gamesRemaining}");  
 text.AppendLine(" Time to next game: {0}</size>");  
 text.AppendLine();  
 text.AppendLine();  
 AddScoreTable(text);  
  
 totalScoreTextUnformatted = text.ToString();  
   
 SetTotalScoreTimeRemaining(timeRemaining);  
 }  
  
 /// <summary>  
 /// Изменяет оставшееся время в таблице с результатами игры  
 /// </summary>  
 /// <param name="time">Оставшееся количество времени</param>  
 public void SetTotalScoreTimeRemaining(int time) {  
 totalScoreText.text = String.Format(totalScoreTextUnformatted, time);  
 }  
  
 /// <summary>  
 /// Прекращает показ таблицы с результатми игры  
 /// </summary>  
 public void HideTotalScore() {  
 totalScorePanel.SetActive(false);  
 }  
  
 /// <summary>  
 /// Показывает финальную таблицу с результатми игры  
 /// </summary>  
 public void ShowFinalResults() {  
 totalScorePanel.SetActive(true);  
 exitButton.gameObject.SetActive(true);  
 var text = new StringBuilder();  
 text.AppendLine($"<size=130%> Game Finished");  
 var winner = PlayersManager.playersSortedByTotalScore[0];  
 text.AppendLine($" Winner: <color={ColorForPlayer(winner)}>{winner.name}</color></size>");  
 text.AppendLine();  
 text.AppendLine();  
 AddScoreTable(text);  
  
 totalScoreText.text = text.ToString();  
 }  
  
 /// <summary>  
 /// Входит из матча. Вызывается Unity при нажатии на кнопку выхода  
 /// </summary>  
 public void ExitButtonClicked() {  
 UberDebug.Log("Client", "exiting match");  
 sClient.Reset();  
 }  
  
  
 /// <summary>  
 /// Устанавливает обработчики событий  
 /// </summary>  
 public void SetupHandlers() {  
 EventsManager.handler.OnPlayerBulletsCountChanged += (player, count) => {  
 if (player != Client.client.mainPlayerObj) return;  
 bulletsPanel.SetActiveImagesCount(count);  
 };  
 EventsManager.handler.OnPlayerMagazinesCountChanged += (player, count) => {  
 if (player != Client.client.mainPlayerObj) return;  
 magazinesPanel.SetActiveImagesCount(count);  
 };  
 EventsManager.handler.OnPlayerPickedUpGun += (player, gun) => {  
 if (player != Client.client.mainPlayerObj) return;  
 gunImage.enabled = true;  
 switch (gun) {  
 case Pistol pistol:  
 gunImage.sprite = pistolSprite;  
 break;  
 case ShotGun shotGun:  
 gunImage.sprite = shotgunSprite;  
 break;  
 case SemiautoGun semiautoGun:  
 gunImage.sprite = semiautoSprite;  
 break;  
 case BombGun bombGun:  
 gunImage.sprite = grenadeLauncherSprite;  
 break;  
 default:  
 throw new Exception("Unknown gun:" + gun);  
 }  
  
 if (gun is ReloadingGun g) {  
 bulletsPanel.SetMaxImagesCount(g.GetBulletsInMagazine());  
 bulletsPanel.SetActiveImagesCount(g.bulletsCount);  
 magazinesPanel.SetMaxImagesCount(5);  
 magazinesPanel.SetActiveImagesCount(g.magazinesCount);  
 }  
 };  
  
 EventsManager.handler.OnPlayerDroppedGun += (player, gun) => {  
 if (player != Client.client.mainPlayerObj) return;  
 gunImage.enabled = false;  
 };  
  
 EventsManager.handler.OnPlayerScoreChanged += (\_player, score) => {  
 RedrawScore();  
 };  
 }  
  
 /// <summary>  
 /// Инициализирует переменные и перерисовывает интерфейс  
 /// </summary>  
 private void Awake() {  
 Object.DontDestroyOnLoad(gameObject);  
 spawned = true;  
 RedrawScore();  
 RedrawChat();  
 }  
  
  
 /// <summary>  
 /// Устанавливает текст задачи  
 /// </summary>  
 /// <param name="text">Текст задачи</param>  
 public void SetTask(string text) {  
 taskText.text = "<align=center><size=130%>Task</size></align>\n" + text;  
 }  
  
  
 /// <summary>  
 /// Устанавливает состояние на набор текста  
 /// </summary>  
 public void StartTyping() {  
 sClient.isTyping = true;  
 }  
  
 /// <summary>  
 /// Был ли в последнем кадре прекращён набор текста  
 /// </summary>  
 private bool recentlyStoppedTyping = false;  
 /// <summary>  
 /// Выключает состояние набора текста  
 /// </summary>  
 public void StopTyping() {  
 recentlyStoppedTyping = true;  
 sClient.isTyping = false;  
 }  
  
 /// <summary>  
 /// Отправляет сообщение в чат  
 /// </summary>  
 public void SendToChat() {  
 string message = chatInput.text;  
 if (message != "")  
 CommandsHandler.gameRoom.RunSimpleCommand(new CreateChatMessageCommand(PlayersManager.mainPlayer.id, message),   
 MessageFlags.NONE);  
   
 chatInput.text = "";  
 StopTyping();  
 chatInput.DeactivateInputField(true);   
 }  
  
 /// <summary>  
 /// Добавляет сообщение в чат  
 /// </summary>  
 /// <param name="player">Игрок, отправивший сообщение</param>  
 /// <param name="message">Сообщение</param>  
 public void AddChatMessage(Player player, string message) {  
 message = $"<color={ColorForPlayer(player)}>{player.name}: {message}</color>";  
 chatMessages.Add(message);  
 if (chatMessages.Count > 4)  
 chatMessages.RemoveAt(0);  
   
 RedrawChat();  
 StartCoroutine(DeleteMessageAfterTime(7, message));  
 }  
  
 /// <summary>  
 /// Корутина для удаления сообщения из чата через время  
 /// </summary>  
 /// <param name="time">Время, через которое нужно удалить сообщение</param>  
 /// <param name="message">Сообщение</param>  
 /// <returns>Корутину</returns>  
 private IEnumerator DeleteMessageAfterTime(float time, string message) {  
 yield return new WaitForSeconds(time);  
 chatMessages.Remove(message);  
 RedrawChat();  
 }  
  
 /// <summary>  
 /// Перерисовывает текст чата  
 /// </summary>  
 public void RedrawChat() {  
 var text = new StringBuilder();  
 foreach (var message in chatMessages) {  
 text.AppendLine(message);  
 }  
  
 chatText.text = text.ToString();  
 }  
  
 /// <summary>  
 /// Время, которое сейчас показывается на таймере с обратным отсчётом  
 /// </summary>  
 private int lasttime = -1;  
 /// <summary>  
 /// Устанавливает время на таймере с обратным отсчётом  
 /// </summary>  
 /// <param name="time">Время</param>  
 public void SetTimerTime(int time) {  
 if (lasttime == time) return;  
 lasttime = time;  
  
 int minutes = time / 60;  
 int seconds = time % 60;  
 timerText.text = $"{minutes:00}:{seconds:00}";  
 }  
  
 /// <summary>  
 /// Обновляет состояние интерфейса. Автоматически вызывается Unity каждый кадр  
 /// </summary>  
 private void Update() {  
 if (recentlyStoppedTyping) {  
 recentlyStoppedTyping = false;  
 } else {  
 if ((Input.GetKeyDown(KeyCode.Return) || Input.GetKeyDown(KeyCode.KeypadEnter)) && !sClient.isTyping) {  
 chatInput.ActivateInputField();  
 }  
 }  
 }  
}

## MultiImagePanel.cs

﻿using System.Collections.Generic;  
using UnityEngine;  
using UnityEngine.UI;  
  
namespace UI {  
 /// <summary>  
 /// Класс для панели интерфейса с одинаковыми изображенями  
 /// </summary>  
 [RequireComponent(typeof(RectTransform))]  
 public class MultiImagePanel : MonoBehaviour {  
 /// <summary>  
 /// Массив изображений  
 /// </summary>  
 private List<Image> images = new List<Image>();  
 /// <summary>  
 /// Изображение  
 /// </summary>  
 public Image image;  
  
 /// <summary>  
 /// Панель, на которой показывается данный элемента  
 /// </summary>  
 private RectTransform panel;  
   
 /// <summary>  
 /// Инициализирует переменные  
 /// </summary>  
 public void OnEnable() {  
 panel = GetComponent<RectTransform>();  
 image.enabled = false;  
 }  
  
 /// <summary>  
 /// Устанавливает максимальное число изображений, которое нужно показывать  
 /// </summary>  
 /// <param name="count">Максимальное число изображений</param>  
 public void SetMaxImagesCount(int count) {  
 foreach (var bullet in images) {  
 Destroy(bullet.gameObject);  
 }  
 images.Clear();  
   
   
 float bulletOffsetX = (panel.rect.width - image.rectTransform.rect.width) \* panel.lossyScale.x / (count - 1);  
 if (count == 1)  
 bulletOffsetX = 0;  
 for (int i = 0; i < count; i++) {  
 var go = Instantiate(image, panel.gameObject.transform, true);  
 go.enabled = true;  
 go.transform.Translate(i \* bulletOffsetX, 0, 0);  
 images.Add(go);  
 }  
 }  
  
 /// <summary>  
 /// Устанавливает число изображений, которые нужно показывать сейчас  
 /// </summary>  
 /// <param name="count">Число изображений</param>  
 public void SetActiveImagesCount(int count) {  
 for (int i = 0; i < images.Count; i++) {  
 images[i].enabled = i < count;  
 }  
 }  
 }  
}

## StartUIController.cs

﻿using System;  
using System.Text;  
using CommandsSystem.Commands;  
using Events;  
using Game;  
using GameMode;  
using Networking;  
using TMPro;  
using UnityEngine;  
using UnityEngine.UI;  
  
namespace UI {  
 /// <summary>  
 /// Компонента для управления стартовым интерфейсом в игре  
 /// </summary>  
 public class StartUIController : MonoBehaviour {  
 /// <summary>  
 /// Поле для ввода ника  
 /// </summary>  
 public TMP\_InputField nameInput;  
  
 /// <summary>  
 /// Текстовый элемент с информацией о текущем матче  
 /// </summary>  
 public TextMeshProUGUI matchInfoText;  
   
 /// <summary>  
 /// Панель с интерфейсом для ввода ника и кнопкой Play  
 /// </summary>  
 public GameObject JoinUI;  
 /// <summary>  
 /// Панель для интерфейса с информацией о чате  
 /// </summary>  
 public GameObject MatchUI;  
 /// <summary>  
 /// Ввёл ли пользователь ник  
 /// </summary>  
 public static bool specificName = false;  
   
 /// <summary>  
 /// Инициализирует переменные  
 /// </summary>  
 public void Awake() {  
 if (specificName)  
 nameInput.text = PlayersManager.mainPlayer.name;  
 JoinUI.SetActive(true);  
 MatchUI.SetActive(false);  
 }  
  
 /// <summary>  
 /// Устанавливает обработчики событий  
 /// </summary>  
 private void Start() {  
 MainUIController.mainui.gameObject.SetActive(false);  
  
 EventsManager.handler.OnCurrentMatchChanged += (last, currentMatch) => {  
  
 var text = new StringBuilder();  
 text.AppendLine(currentMatch.name);  
 text.AppendLine($"Waiting players {currentMatch.players.Count}/{currentMatch.maxPlayersCount}:");  
 foreach (var player in currentMatch.players) {  
 string color = player == PlayersManager.mainPlayer.name ? "green" : "red";  
 text.AppendLine($"<color={color}> -{player}</color>");  
 }  
 matchInfoText.SetText(text.ToString());  
   
   
 if (currentMatch.players.Count >= currentMatch.maxPlayersCount) {  
 MatchesManager.SendStartGame();  
 }  
 };  
 }  
  
 /// <summary>  
 /// Обрабатывает нажатие кнопки играть  
 /// </summary>  
 public void OnPlayClicked() {  
 if (nameInput.text != "") {  
 PlayersManager.mainPlayer.name = nameInput.text;  
 specificName = true;  
 }  
  
 sClient.StartFindingMatch();  
 JoinUI.SetActive(false);  
 MatchUI.SetActive(true);  
  
 matchInfoText.text = "Finding matches...";  
 }  
   
 /// <summary>  
 /// Переключает на главный интерфейс  
 /// </summary>  
 private void OnDestroy() {  
 MainUIController.mainui.gameObject.SetActive(true);  
 }  
   
 /// <summary>  
 /// При нажатии клавиши enter переходит к поиску матча  
 /// </summary>  
 private void Update() {  
 if (Input.GetKeyDown(KeyCode.Return) || Input.GetKeyDown(KeyCode.KeypadEnter)) {  
 OnPlayClicked();  
 }  
 }  
 }  
}

## AutoDisableRendererOnStart.cs

﻿using UnityEngine;  
  
namespace Util2 {  
 /// <summary>  
 /// Компонента, автоматически отключающая Renderer у объекта при старте  
 /// </summary>  
 public class AutoDisableRendererOnStart : MonoBehaviour {  
 /// <summary>  
 /// Отключает Renderer  
 /// </summary>  
 void Start() {  
 GetComponent<Renderer>().enabled = false;  
 }  
 }  
}

## AutoHideOnStart.cs

﻿namespace DefaultNamespace {  
 /// <summary>  
 /// Компонента, автоматически отключающая объект при старте  
 /// </summary>  
 public class AutoHideOnStart : UnityEngine.MonoBehaviour {  
 public void Start() {  
 gameObject.SetActive(false);  
 }  
 }  
}

## AutoID.cs

﻿using UnityEngine;  
  
namespace Util2 {  
 /// <summary>  
 /// Компонента, автоматически присваивающая объекту ID  
 /// </summary>  
 public class AutoID : MonoBehaviour {  
 /// <summary>  
 /// ID объекта  
 /// </summary>  
 public int ID;  
  
 /// <summary>  
 /// Генерирует случайный ID  
 /// </summary>  
 public void Reset() {  
 ID = ObjectID.RandomID;  
 }  
  
 /// <summary>  
 /// Сохраняет ID объекта  
 /// </summary>  
 public void Awake() {  
 ObjectID.StoreObject(gameObject, ID, 0, 0);  
 }  
  
 /// <summary>  
 /// Удаляет эту компоненту из объекта во время игры  
 /// </summary>  
 public void Update() {  
 Destroy(this);  
 }  
 }  
}

## AutoMatchJoiner.cs

﻿using System;  
using Events;  
using Game;  
using UnityEngine;  
  
namespace Util2 {  
 /// <summary>  
 /// Компонента для автоматического поиска и входа в матч  
 /// </summary>  
 public class AutoMatchJoiner : MonoBehaviour {  
 /// <summary>  
 /// Запущен ли автовыбор матча  
 /// </summary>  
 public static bool isRunning = false;  
 /// <summary>  
 /// Нужно ли ждать другого игрока для старта игры  
 /// </summary>  
 public static bool sneedWaitOtherPlayers;  
   
 /// <summary>  
 /// Нужно ли ждать другого игрока для старта игры  
 /// </summary>  
 public bool needWaitOtherPlayers = false;  
  
 /// <summary>  
 /// Инициализирует переменные  
 /// </summary>  
 public void Awake() {  
 isRunning = true;  
 sneedWaitOtherPlayers = needWaitOtherPlayers;  
 }  
  
 /// <summary>  
 /// Автоматически входит в матч  
 /// </summary>  
 public void Start() {  
 EventsManager.handler.OnCurrentMatchChanged += (last, currentMatch) => {  
  
 if (!needWaitOtherPlayers || currentMatch.players.Count >= currentMatch.maxPlayersCount) {  
 MatchesManager.SendStartGame();  
 }  
 };  
   
 sClient.StartFindingMatch();  
  
 }  
 }  
}

## gUtil.cs

﻿namespace Util2 {  
 /// <summary>  
 /// Класс с дополнительными методами  
 /// </summary>  
 public static class gUtil {  
 /// <summary>  
 /// Меняет переменные местами  
 /// </summary>  
 /// <param name="lhs">Первая переменная</param>  
 /// <param name="rhs">Вторая переменная</param>  
 /// <typeparam name="T">Тип переменных</typeparam>  
 public static void Swap<T>(ref T lhs, ref T rhs)  
 {  
 T temp;  
 temp = lhs;  
 lhs = rhs;  
 rhs = temp;  
 }  
 }  
}

## IOwnedEventHandler.cs

﻿/// <summary>  
/// Интерфейс для обработки событий, когда у объекта меняется владелец  
/// </summary>  
public interface IOwnedEventHandler {  
 /// <summary>  
 /// Обработчик события, когда у объекта меняется владелец  
 /// </summary>  
 /// <param name="owner">Новый владелец объекта</param>  
 void HandleOwnTaken(int owner);  
}

## RotatingItem.cs

﻿using System;  
using UnityEngine;  
using Random = UnityEngine.Random;  
  
namespace Util2 {  
 /// <summary>  
 /// Компонента для вращающегося игрвого объекта  
 /// </summary>  
 public class RotatingItem : UnityEngine.MonoBehaviour {  
 /// <summary>  
 /// Скорость вращения  
 /// </summary>  
 public float rotationSpeed = 70;  
 /// <summary>  
 /// Скорость передвижения вверх-вниз  
 /// </summary>  
 public float upDownSpeed = 3f;  
 /// <summary>  
 /// Амплитуда перемещения вверх-вниз  
 /// </summary>  
 public float upDownAmplitude = 0.25f;  
 /// <summary>  
 /// Сдвиг в фазе перемещения вверх вниз  
 /// </summary>  
 private float phase;  
  
 /// <summary>  
 /// Стартовая координата y  
 /// </summary>  
 private float startingY;  
  
 /// <summary>  
 /// Инициализирует переменные  
 /// </summary>  
 public void Start() {  
 startingY = transform.position.y;  
 phase = Random.value \* Mathf.PI \* 2;  
 transform.Rotate(0, Random.value \* 2 \* Mathf.PI, 0);  
 }  
   
 /// <summary>  
 /// Вращает объект и перемещает его вверх-вниз  
 /// </summary>  
 public void Update() {  
 transform.Rotate(0, rotationSpeed \* Time.deltaTime, 0);  
 var pos = transform.localPosition;  
 pos.y = upDownAmplitude \* Mathf.Sin(upDownSpeed \* Time.time + phase);  
 transform.localPosition = pos;  
 }  
 }  
}

# ТЕКСТ ПРОГРАММЫ СЕРВЕРА

## data.js

﻿ module.exports = {

rooms: new Map(),

matches: new Map()

};

## httpserver.js

﻿const serveStatic = require('serve-static');

const finalhandler = require('finalhandler');

const http = require("http");

// Serve up public/ftp folder

const serve = serveStatic('webgl', {

'index': "index.html",

'setHeaders': setHeaders

});

/\*

https://forum.unity.com/threads/changes-to-the-webgl-loader-and-templates-introduced-in-unity-2020-1.817698/

.gz files should be served with a Content-Encoding: gzip response header.

.br files should be served with a Content-Encoding: br response header.

.wasm, .wasm.gz or .wasm.br files should be served with a Content-Type: application/wasm response header.

.js, .js.gz or .js.br files should be served with a Content-Type: application/javascript response header.

\*/

function setHeaders (res, path) {

console.log(path);

if (path.endsWith(".gz")|| path.endsWith(".unityweb")) {

res.setHeader( "Content-Encoding", "gzip");

} else if (path.endsWith(".br") /\*|| path.endsWith(".unityweb")\*/) {

res.setHeader("Content-Encoding", "br");

}

if (path.endsWith(".wasm") || path.endsWith(".wasm.gz") || path.endsWith(".wasm.br")) {

res.setHeader("Content-Type", "application/wasm");

} else if (path.endsWith(".js") || path.endsWith(".js.gz") || path.endsWith(".js.br")) {

res.setHeader("Content-Type", "application/javascript");

}

}

// Create server

const httpServer = http.createServer(function onRequest (req, res) {

serve(req, res, finalhandler(req, res));

});

httpServer.listen(8080);

module.exports = httpServer;

## index.js

﻿

const data = require('./data');

const uniqid = require('uniqid');

const messages\_handler = require("./messages\_handler");

const logger = require("./logger");

const wss = require("./websocketserver");

wss.on('connection', function connection(ws, request) {

ws.id = uniqid();

ws.roomIds = new Set();

logger.info("Connected client#" + ws.id);

ws.on('message', function incoming(data) {

messages\_handler.handle\_message(data, ws);

});

ws.on("close", function() {

logger.info("Disconnected client#" + ws.id);

for (let roomid of ws.roomIds) {

if (data.rooms.has(roomid)) {

data.rooms.get(roomid).RemoveClient(ws);

}

}

});

});

const http = require('http');

http.createServer(function (req, res) {

res.writeHead(200, {'Content-Type': 'text/html; charset=UTF-8'});

res.write("<h3>clients:</h3><pre>");

for (let client of wss.clients) {

res.write(client.id + "\n");

}

res.write("</pre><h3>matches:</h3><pre>");

for (let match of data.matches.values()) {

res.write("match#" + match.id + " name: '" + match.name + "' maxPlayersCount: " + match.maxPlayersCount + "<br/>");

for (let client of match.clients) {

res.write(" client#" + client.id + "<br/>");

}

}

res.write("</pre><h3>rooms:</h3><pre>");

for (let room of data.rooms.values()) {

res.write("room#" + room.id + "<br/>");

for (let client of room.clients) {

res.write(" client#" + client.id + "<br/>");

}

}

res.end('</pre><script>setInterval(function() {document.location.reload()}, 2000) </script>');

}).listen(8886);

## JsonRequest.js

﻿ const logger = require("./logger");

let jsonCommandTypeBuf = Buffer.allocUnsafeSlow(1);

jsonCommandTypeBuf.writeUInt8(254);

class JsonRequest {

constructor(client, room, json) {

this.client = client;

this.room = room;

this.json = json;

}

respond(jsonResponse) {

jsonResponse["\_id"] = this.json["\_id"];

logger.debug("server#" + this.room.id + " -> client#" + this.client.id + " JSON " + JSON.stringify(jsonResponse));

let encoded = Buffer.from(JSON.stringify(jsonResponse), "utf8");

encoded = Buffer.concat([jsonCommandTypeBuf, encoded]);

let message = this.room.CreateMessageWithId(encoded, -1);

this.room.SendToClient(this.client, message, -1);

}

}

module.exports = JsonRequest;

## logger.js

﻿const log4js = require('log4js');

const logger = log4js.getLogger();

logger.level = 'debug';

logger.info("Starting server");

module.exports = logger;

## match\_maker.js

﻿ const data = require("./data");

const Room = require("./Room");

const rooms = data.rooms;

const matches = data.matches;

const logger = require("./logger");

const validate = require('json-schema');

const toJsonSchema = require('to-json-schema');

const JsonRequest = require("./JsonRequest");

const CreateMatchSchema = toJsonSchema({

roomid: 123,

name: "room#test",

maxPlayersCount: 4,

state: 0

}, {required: true}); // TODO: possible attack with invalid values

const GetMatchesList = toJsonSchema({

}, {required: true});

const JoinMatch = toJsonSchema({

name: "player#123",

matchid: 123

}, {required: true});

const ChangeMatchState = toJsonSchema({

matchid: 123,

state: 1

}, {required: true});

const validation\_schemas = {

"CreateMatch": CreateMatchSchema,

"GetMatchesList": GetMatchesList,

"JoinMatch": JoinMatch,

"ChangeMatchState": ChangeMatchState

};

class Match extends Room {

constructor(roomid, name, maxPlayersCount, state) {

super(roomid);

this.name = name;

this.maxPlayersCount = maxPlayersCount;

this.state = state;

matches.set(roomid, this);

}

playersCount = this.clients.size;

GetMatchInfo() {

let players = [];

for (let client of this.clients){

players.push(client.name);

}

return {

roomid: this.id,

name: this.name,

players: players,

maxPlayersCount: this.maxPlayersCount,

state: this.state

}

}

SendMatchInfo() {

let mi = this.GetMatchInfo();

for (let client of this.clients) {

(new JsonRequest(client, this, {"\_id": -1})).respond({"match": mi})

}

}

RemoveClient(client) {

super.RemoveClient(client);

this.SendMatchInfo()

}

}

class MatchMakerRoom extends Room {

CreateMatch(json) {

new Match(json.roomid, json.name, json.maxPlayersCount, json.state);

}

RemoveClient(client) {

this.clients.delete(client); // no auto room delete

}

HandleJsonRequest(request) {

const client = request.client;

const json = request.json;

if (json['\_type'] in validation\_schemas) {

const res = validate.validate(json, validation\_schemas[json['\_type']]);

if (!res.valid) {

logger.warn("client#"+client.id +" send invalid json " + json['\_type'] + " " + res.errors.map(val=> val["property"] + " " + val["message"]).join("; "));

request.respond({"result": "invalid"});

return;

}

}

switch (json['\_type']) {

case 'CreateMatch':

if (rooms.has(json.roomid)) {

request.respond({"result": "error", "message": "Roomid already exists"});

return;

}

this.CreateMatch(json);

request.respond({"result": "success"});

break;

case 'GetMatchesList':

let res = [];

for (let match of matches.values()) {

if (match.state !== 0) continue;

res.push(match.GetMatchInfo());

}

request.respond({"result": "success", "matches": res});

break;

case 'JoinMatch':

if (!matches.has(json["matchid"])) {

request.respond({"result": "error", "message": "This match does not exists"});

return;

}

const match = matches.get(json["matchid"]);

if (match.playersCount >= match.maxPlayersCount) {

request.respond({"result": "error", "message": "Too many players in match", "match": match.GetMatchInfo()});

return;

}

client.name = json["name"];

match.AddClient(client);

request.respond({"result": "success", "match": match.GetMatchInfo()});

for (let otherClient of match.clients) {

if (request.client === otherClient)

continue;

(new JsonRequest(otherClient, this, {"\_id": -1})).respond({"match": match.GetMatchInfo()})

}

break;

case 'ChangeMatchState': {

if (!matches.has(json["matchid"])) {

request.respond({"result": "error", "message": "This match does not exists"});

return;

}

const match = matches.get(json["matchid"]);

match.state = json["state"];

request.respond({"result": "success", "match": match.GetMatchInfo()});

match.SendMatchInfo();

break;

}

default:

logger.warn("got unknown json message type: '" + json['\_type'] + "' from client#" + request.client.id);

break;

}

}

}

const matchmaking\_room\_id = 42;

const matchmaking\_room = new MatchMakerRoom(matchmaking\_room\_id);

data.rooms.set(matchmaking\_room\_id, matchmaking\_room);

/\*

exports.handle\_create\_match = function (client, matchInfo) {

};

\*/

exports.Match = Match;

## MessageFlags.js

﻿ module.exports = {

NONE : 0,

IMPORTANT : 1 << 0,

SEND\_ONLY\_IMPORTANT : 1 << 1

};

## messages\_handler.js

﻿const MessageType = require('./MessageType');

const logger = require("./logger");

const Util = require("./Util");

const Room = require("./Room");

const JsonRequest = require("./JsonRequest");

const data = require("./data");

const rooms = data.rooms;

const match\_maker = require("./match\_maker");

function handle\_message(data, client) {

if (!(data instanceof Buffer)) {

logger.warn("got unknown data type " + data.prototype + " from client " + client);

return;

}

if (data.length < 6) {

logger.warn("got data without header: " + data.length + " from client#" + client.id);

return;

}

const messageTypeOrd = data.readInt8(0);

const roomId = data.readInt32LE(1);

const flags = data.readInt8(5);

data = data.slice(6);

if (data.length !== 46)

logger.debug("client#" + client.id + " -> server#" + roomId + " " +

Util.MessageTypeToString(messageTypeOrd) + " " + Util.MessageFlagsToString(flags) + " len: " + data.length);

if (messageTypeOrd === MessageType.JoinGameRoom) {

if (data.length !== 0) {

logger.warn("got JoinGameRoom with incorrect length " + data.length + "from client#"+client.id+". should be 0");

return;

}

if (!rooms.has(roomId))

new Room(roomId);

rooms.get(roomId).AddClient(client);

return;

}

if (!rooms.has(roomId)) {

logger.warn("client#" + client.id + " send message to room#" + roomId + " which is not exists");

return;

}

const room = rooms.get(roomId);

if (!room.clients.has(client)) {

logger.warn("client#" + client.id + " send message to room#" + roomId + " before joining it");

return;

}

switch (messageTypeOrd) {

case MessageType.SimpleMessage:

room.HandleSimpleMessage(client, data, flags);

break;

case MessageType.UniqMessage:

const uid = data.readBigInt64LE(0);

data = data.slice(8);

room.HandleUniqMessage(client, uid, data, flags);

break;

case MessageType.AskMessage:

if (data.length !== 8) {

logger.warn("got AskMessage with incorrect length " + data.length + "from client#"+client.id+". should be 8");

return;

}

const startIndex = data.readInt32LE(0);

const endIndex = data.readInt32LE(4);

room.HandleAskMessage(client, startIndex, endIndex, flags);

break;

case MessageType.LeaveGameRoom:

if (data.length !== 0) {

logger.warn("got LeaveGameRoom with incorrect length " + data.length + "from client#"+client.id+". should be 0");

return;

}

room.RemoveClient(client);

break;

case MessageType.JSON:

const json = Util.json\_safe\_parse\_buffer(data);

if (json == null) {

logger.warn("got bad json message from client#"+client.id);

}

if (!('\_type' in json)) {

logger.warn("got json message without '\_type': " + json + " from client#"+client.id);

return;

}

if (!('\_id' in json)) {

logger.warn("got json message without '\_id': " + json + " from client#"+client.id);

return;

}

logger.debug("client#" + client.id + " -> server#" + roomId + " JSON#" + json["\_id"] + " " + json["\_type"] + " " + JSON.stringify(json));

const request = new JsonRequest(client, room, json);

room.HandleJsonRequest(request);

break;

default:

logger.warn("got unknown message type: " + messageTypeOrd);

return;

}

}

module.exports = {

handle\_message: handle\_message

};

## MessageType.js

﻿ module.exports = {

SimpleMessage: 1,

UniqMessage: 2,

AskMessage: 3,

JoinGameRoom: 4,

LeaveGameRoom: 5,

JSON: 6

};

## Room.js

﻿ const MessageType = require('./MessageType');

const MessageFlags = require('./MessageFlags');

const logger = require("./logger");

const Util = require("./Util");

const data = require("./data");

const rooms = data.rooms;

const matches = data.matches;

const JsonRequest = require("./JsonRequest");

class Room {

constructor(id) {

this.id = id;

this.clients = new Set();

this.usedUniqIds = new Set();

this.importantMessages = new Map();

this.notImportantMessages = new Map();

this.lastMessageId = -1;

rooms.set(this.id, this);

}

AddClient(client) {

this.clients.add(client);

client.roomIds.add(this.id);

}

RemoveClient(client) {

this.clients.delete(client);

if (this.clients.size === 0) {

rooms.delete(this.id);

if (matches.has(this.id)) {

matches.delete(this.id);

}

}

}

CreateMessageWithId(message, messageId) {

const messageWithId = Buffer.allocUnsafe(8);

messageWithId.writeInt32LE(messageId, 0);

messageWithId.writeInt32LE(this.id, 4);

return Buffer.concat([messageWithId, message]);

}

BroadcastMessage(message, flags) {

const messageId = this.lastMessageId + 1;

this.lastMessageId++;

message = this.CreateMessageWithId(message, messageId);

if ((flags & MessageFlags.IMPORTANT) !== 0) {

this.importantMessages.set(messageId, message);

} else {

this.notImportantMessages.set(messageId, message);

if (this.notImportantMessages.size > 1100) {

let it = this.notImportantMessages.keys();

for (let i = 0; i < 100; i++) {

this.notImportantMessages.delete(it.next().value);

}

}

}

//for (int i = 0; i < 1; i++) { // random.Next(0, 2)

for (let client of this.clients) {

this.SendToClient(client, message, messageId);

/\* if (flags.HasFlag(MessageFlags.IMPORTANT) || true)

UberDebug.LogChannel("SERVER", $"server{id}->client{client} {message.Length} {flags}");

}\*/

//}

}

}

HandleSimpleMessage(client, message, flags) {

this.BroadcastMessage(message, flags);

}

HandleUniqMessage(client, uid, message, flags) {

if (this.usedUniqIds.has(uid)) return;

this.usedUniqIds.add(uid);

this.BroadcastMessage(message, flags);

}

HandleAskMessage(client, startIndex, endIndex, flags) {

if ((startIndex > endIndex || endIndex > this.lastMessageId) && endIndex !== -1) {

logger.warn("AskMessage from " + client.id + " is incorrect. " +

"startIndex:" + startIndex + " endIndex:" + endIndex + " lastMessageId: " + this.lastMessageId);

return;

}

if (startIndex < 0) {

startIndex = this.lastMessageId + startIndex;

if (startIndex < 0)

startIndex = 0;

}

if (endIndex === -1)

endIndex = this.lastMessageId;

const only\_important = (flags & MessageFlags.SEND\_ONLY\_IMPORTANT) !== 0;

for (let i = startIndex; i <= endIndex; i++) {

if (this.importantMessages.has(i)) {

this.SendToClient(client, this.importantMessages.get(i), i);

} else if (this.notImportantMessages.has(i) && !only\_important) {

this.SendToClient(client, this.notImportantMessages.get(i), i);

} else {

if (!only\_important)

logger.warn("trying to get too many not important messages. sending empty");

this.SendToClient(client, this.CreateMessageWithId(Util.EmptyMessage, i, this.id), i);

}

}

}

HandleJsonRequest(request) {

logger.warn("unhandled jsonrequest to room#" + this.id + " from client#" + request.client.id + " " + JSON.stringify(request.json));

}

SendToClient(client, data, messageId) {

if (data.length !== 54)

logger.debug("server#" + this.id + " -> client#" + client.id + " " + messageId + " len: " + data.length);

client.send(data);

}

}

module.exports = Room;

## Util.js

﻿ const MessageType = require("./MessageType");

const MessageFlags = require("./MessageFlags");

const \_MessageTypeToString = {};

for (let name in MessageType) {

\_MessageTypeToString[MessageType[name]] = name.toString();

}

/\*\*

\* @return {string}

\*/

exports.MessageFlagsToString = function (flags) {

let res = "";

for (let flag in MessageFlags){

if ((MessageFlags[flag] & flags) !== 0)

res += flag + ","

}

return res;

};

/\*\*

\* @return {string}

\*/

exports.MessageTypeToString = function (messageType) {

if (messageType in \_MessageTypeToString) {

return \_MessageTypeToString[messageType];

}

return "unknown" + messageType;

};

exports.json\_safe\_parse = function (json) {

let parsed;

try {

parsed = JSON.parse(json)

} catch (e) {

return null;

}

return parsed;

};

exports.json\_safe\_parse\_buffer = function (buffer) {

let parsed;

try {

parsed = JSON.parse(buffer.toString('utf-8'));

} catch (e) {

return null;

}

return parsed;

};

const EmptyMessage = Buffer.allocUnsafe(1);

EmptyMessage.writeUInt8(255);

exports.EmptyMessage = EmptyMessage;

## websocketserver.js

﻿ const WebSocket = require('ws');

const httpServer = require("./httpserver");

const wss = new WebSocket.Server({

server: httpServer,

path: "/ws",

perMessageDeflate: {

zlibDeflateOptions: {

// See zlib defaults.

chunkSize: 1024,

memLevel: 7,

level: 3

},

zlibInflateOptions: {

chunkSize: 10 \* 1024

},

// Other options settable:

clientNoContextTakeover: true, // Defaults to negotiated value.

serverNoContextTakeover: true, // Defaults to negotiated value.

serverMaxWindowBits: 10, // Defaults to negotiated value.

// Below options specified as default values.

concurrencyLimit: 10, // Limits zlib concurrency for perf.

threshold: 1024 // Size (in bytes) below which messages

// should not be compressed.

}

});

function noop() {}

function heartbeat() {

this.isAlive = true;

}

wss.on('connection', function connection(ws) {

ws.isAlive = true;

ws.on('pong', heartbeat);

});

const interval = setInterval(function ping() {

wss.clients.forEach(function each(ws) {

if (ws.isAlive === false) return ws.terminate();

ws.isAlive = false;

ws.ping(noop);

});

}, 30000);

wss.on('close', function close() {

clearInterval(interval);

});

module.exports = wss;

ПРИЛОЖЕНИЕ 2  
СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ

1. ГОСТ 19.101-77 Виды программ и программных документов. //Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.
2. ГОСТ 19.102-77 Стадии разработки. //Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.
3. ГОСТ 19.103-77 Обозначения программ и программных документов. //Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.
4. ГОСТ 19.104-78 Основные надписи. //Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.
5. ГОСТ 19.105-78 Общие требования к программным документам. //Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.
6. ГОСТ 19.106-78 Требования к программным документам, выполненным печатным способом. //Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.
7. ГОСТ 19.201-78 Техническое задание. Требования к содержанию и оформлению. //Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.
8. ГОСТ 19.603-78 Общие правила внесения изменений. //Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.
9. ГОСТ 19.604-78 Правила внесения изменений в программные документы, выполненные печатным способом. //Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.
10. ГОСТ 15150-69 Машины, приборы и другие технические изделия. Исполнения для различных климатических районов. Категории, условия эксплуатации, хранения и транспортирования в части воздействия климатических факторов внешней среды. – М.: Изд-во стандартов, 1997.
11. ГОСТ 19.602-78 Правила дублирования, учета и хранения программных документов, выполненных печатным способом. //Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.
12. ГОСТ 19.301-79 Программа и методика испытаний. Требования к содержанию и оформлению. //Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.
13. Облачное хранилище с текстом программы [Электронный ресурс] // URL: https://github.com/misha1sh/unity\_game\_final/ (Дата обращения: 23.05.2020, режим доступа: свободный).

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Лист регистрации изменений | | | | | | | | | |
| Номера листов (страниц) | | | | | Всего листов (страниц в докум.) | № документа | Входящий  № сопроводит ельного докум. и дата | Подп. | Дата |
| Изм. | Изменен ных | Заменен ных | Новых | Аннули рованных |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |

# ЛИСТ РЕГИСТРАЦИИ ИЗМЕНЕНИЙ