class StateHandlerRoom extends Room<State>

```
onCreate (options)
this.setState(new State());
// Подписка да сообщения от клиента:

this.onMessage("move", (client, data) => {
this.state.movePlayer(client.sessionId,
data);
});

this.onMessage("shoot", (client, data) => {
this.broadcast("Shoot", data, {except:
client});
});

this.onMessage("damage" (client, data) =>
{
    const clientID = data.id;
    const player =
this.state.players.get(clientID);
    let hp = player.currentHP - data.value;
    player.currentHP -= data.value;
    player.currentHP = hp;
});
```

```
onJoin (client: Client, data: any)
this.state.createPlayer(client.sessionId,
data, skin);
```

class State extends Schema

```
createPlayer(sessionId: string, data: any,
skin: number)
const player = new Player();
player.speed = speed;
player.currentHP = data.hp;
this.players.set(sessionId, player);
```

```
movePlayer (sessionId: string, data: any)
const player = this.players.get(sessionId);
player.pX = data.pX;
```

class Player extends Schema

```
@type("uint8")
skin = 0;

@type("uint8")
loss = 0;

@type("int8")
maxHP = 0;

@type("int8")
currentHP = 0;

@type("number")
speed = 0;

@type("number")
pX = 0

@type("boolean")
sit = false;
```

SERVER

class MultiplayerManager : ColyseusManager<MultiplayerManager>

```
private ColyseusRoom<State> _room;
private Dictionary<string, EnemyController>
_enemies = new Dictionary<string,
EnemyController>();
```

```
void Awake()
Instance.InitializeClient();
Connect();
```

```
async void Connect()
Dictionary<string, object> data = new
Dictionary<string, object>() {
    {"skins", skins.length },
    {"points", spawnPoints.length },
    {"hp", _player.maxHealth},
    {"speed", _player.speed},
    {"pX", spawnPosition.x}
};

_room = await
Instance.client.JoinOrCreate<State>("state_
handler", data);
// Подписка на событие:
_room.OnStateChange += OnChange;
// Подписка да сообщения от сервера:
_room.OnMessage<string>("Shoot",
ApplyShoot);
_room.OnMessage<string>("Gun", SetGun);
```

```
void OnChange(State state, bool
isFirstState)
state.players.ForEach((key, player) => {
    if (key == _room.SessionId)
CreatePlayer(player);
    else CreateEnemy(key, player);
});
// Подписка на события:
_room.State.players.OnAdd += CreateEnemy;
_room.State.players.OnRemove +=
RemoveEnemy;
```

```
void CreatePlayer(Player player)
var playerCharacter = Instantiate(_player,
position, rotation);
// Подписка на изменения полей в плеере на
сервере:
player.OnChange +=
playerCharacter.OnChange;
```

```
void CreateEnemy(string key, Player player)
var position = new Vector3(player.pX,
player.pY, player.pZ);
var enemy = Instantiate(_enemy, position,
Quaternion.identity);
enemy.Init(key, player);
_enemies.Add(key, enemy);
```

```
void SendMessage(string key,
Dictionary<string, object> data)
_room?.Send(key, data);
```

```
void SendMessage(string key, string data)
_room?.Send(key, data);
```

```
public string GetSessionID() =>
_room.SessionId;
```

```
void ApplyShoot(string jsonShootInfo)
ShootInfo shootInfo =
JsonUtility.FromJson<ShootInfo>(jsonShootIn
fo);
if (_enemies.ContainsKey(shootInfo.key) ==
false) {
    return;
}
_enemies[shootInfo.key].Shoot(shootInfo);
```

class Controller : MonoBehaviour

```
[SerializeField] private PlayerCharacter
_player;
private MultiplayerManager
_multiplayerManager;
```

```
void Start()
_multiplayerManager =
MultiplayerManager.Instance;
```

```
void Update()
```

```
void SendMove()
Dictionary<string, object> data = new
Dictionary<string, object>() {
    {"pX", position.x},
    {"pY", position.y}
};

_multiplayerManager.SendMessage("move",
data);
```

```
void SendShoot(ref ShootInfo shootInfo)
shootInfo.key =
_multiplayerManager.GetSessionID();
var json = JsonUtility.ToJson(shootInfo);
_multiplayerManager.SendMessage("shoot",
json);
```

CLIENT

class PlayerCharacter : Character

```
[SerializeField] private PlayerCharacter
_player;
private MultiplayerManager
_multiplayerManager;
```

```
void Start()
SetCamera();
_health.SetMax(maxHealth);
_health.SetCurrent(maxHealth);
```

```
void FixedUpdate()
Move();
RotateY();
```

```
void OnChange(List<DataChange> changes)
foreach (var dataChange in changes) {
    switch (dataChange.Field) {
        case "currentHP":
_health.SetCurrent((sbyte)dataChange.Value)
;
            break;
        default:
            Debug.LogWarning("Can't
processed changing of field " +
dataChange.Field);
            break;
    }
}
```

class Bullet : MonoBehaviour

```
[SerializeField] private PlayerCharacter
_player;
private MultiplayerManager
_multiplayerManager;
```

```
void OnCollisionEnter(Collision collision)
if (collision.collider.TryGetComponent(out
EnemyCharacter enemy)) {
    enemy.ApplyDamage(_damage);
}
```

class EnemyController : MonoBehaviour

```
[SerializeField] private EnemyCharacter
_enemyCharacter;
[SerializeField] private EnemyGun _gun;
[SerializeField] private EnemyGunUI _ui;
private Player _player;
```

```
void Init(string key, Player player)
_enemyCharacter.Init(key);
_player = player;
_enemyCharacter.SetMaxHP(player.maxHP);
// Подписка на изменения полей в плеере на
сервере:
player.OnChange += OnChange;
```

```
void Shoot(in ShootInfo info)
Vector3 position = new Vector3(info.pX,
info.pY, info.pZ);
Vector3 velocity = new Vector3(info.dX,
info.dY, info.dZ);
_gun.Shoot(position, velocity);
```

```
void OnChange(List<DataChange> changes)
foreach (var dataChange in changes) {
    switch (dataChange.Field) {
        case "currentHP":
            if ((sbyte)dataChange.Value >
(sbyte)dataChange.PreviousValue)
_enemyCharacter.RestoreHP((sbyte)dataChange
.Value);
            break;
        case "pX":
            position.x =
(float)dataChange.Value;
            break;
        default:
            Debug.LogWarning("Can't
processed changing of field " +
dataChange.Field);
            break;
    }
}
_enemyCharacter.SetMovement(position,
velocity, avarageInterval);
```

class EnemyCharacter : Character

```
void Start()
_targetPosition = transform.position;
```

```
void Update()
Move();
```

```
void RestoreHP(int newValue)
_health.SetCurrent(newValue);
```

```
void SetMovement(in Vector3 position, in
Vector3 velocity, in float averageInterval)
_targetPosition = position + (velocity *
averageInterval);
_velocityMagnitude = velocity.magnitude;
this.velocity = velocity;
```

```
void ApplyDamage(int damage)
_health.ApplyDamage(damage);
Dictionary<string, object> data = new
Dictionary<string, object>() {
    {"id", _sessionID },
    { "value", damage }
};
MultiplayerManager.Instance.SendMessage("da
mage", data);
```