Top-down Multiplayer Shooter Template

(Netcode for GameObjects, Lobby, Relay)
v1.0.4

HEAVYART
2024

About

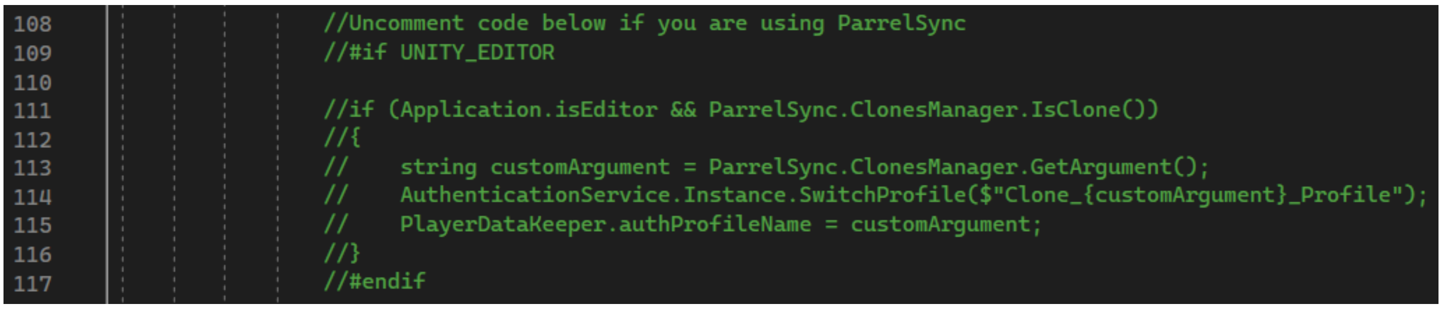
Multiplayer project template. Designed as a starter pack: recommended to import package into a new empty project.
Contains 5 scenes, 2 characters (player, enemy), 3 weapons (pistol, rifle, shotgun), 4 power-ups (movement speed, fire rate, weapon accuracy, medkit).
Uses Lobby, Relay multiplayer services and Netcode for GameObjects.

Project structure:
\Content - character models, weapon models, animations, icons, materials
\Prefabs - character prefabs (player, enemy), drop element prefabs, UI, weapons
\Scenes - all game scenes, including demo scene at \Scenes\Demo\Demo.unity
\ScriptableObjects - drop element configs, NetworkPrefabsList (required by Netcode)
\Scripts - all game code

Scenes:
BootScene - Scene to start from. Used as a place to initialize settings and network. Loads only once - on game start.
LoadingScene - Scene that we see, when switching between scenes.
MainMenu - menu, where we can start new game or join existing, set player name, select player color and switch server region.
GameScene - scene with gameplay itself
Demo - demo scene at \Scenes\Demo\Demo.unity

Where to start:
After the installation guide, we recommend to start from SettingsManager on Boot Scene. It's a permanent object (DontDestroyOnLoad) which contains all project settings, therefore it's the best entry point, which would lead to all the parts of the game.
Also we recommend to check PlayerDataKeeper. This script works with PlayerPrefs and contains data about local player.

Running on few instances:
To run several instances on the same computer, could be used ParrelSync for editor and command line arguments, for standalone builds: -authProfileName uniqueProfileName, where uniqueProfileName - unique name for every instance.
If you are using ParrelSync, uncomment code at LobbyDataControl, lines 109-117.



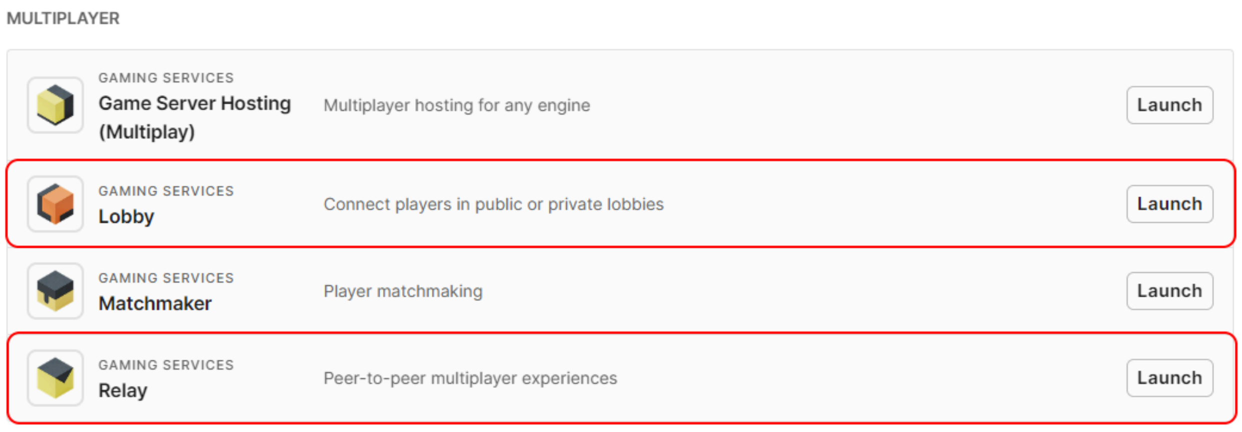
```
108                 //Uncomment code below if you are using ParrelSync
109                 //#if UNITY_EDITOR
110
111                 //if (Application.isEditor && ParrelSync.ClonesManager.IsClone())
112                 //{
113                 //    string customArgument = ParrelSync.ClonesManager.GetArgument();
114                 //    AuthenticationService.Instance.SwitchProfile($"Clone_{customArgument}_Profile");
115                 //    PlayerDataKeeper.authProfileName = customArgument;
116                 //}
117                 //#endif
```

Installation Guide

1. Create project in Unity Dashboard (if it's not created)
    1.1 Select Projects from the primary navigation menu.
    1.2 Click Create project in the upper-right of the Projects page.
    1.3 Enter a project name and COPPA designation.
    1.4 Click Create project.

For more information, see the documentation on managing Unity projects.

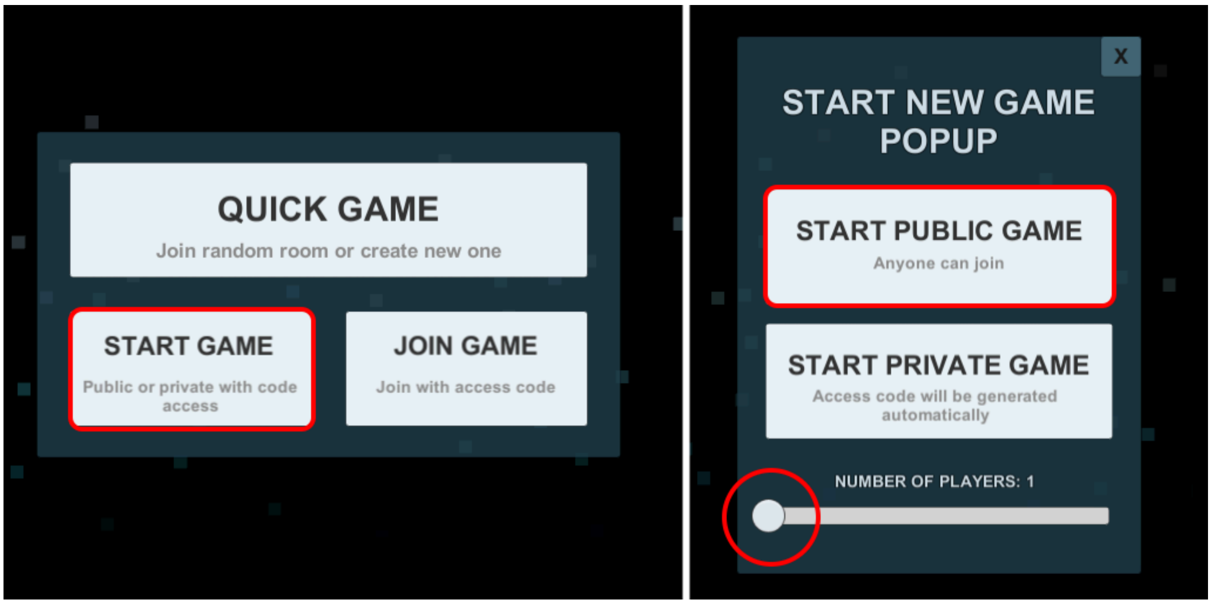
2. Setup Lobby and Relay in Unity Services Dashboard > Multiplayer


MULTIPLAYER

GAMING SERVICES Game Server Hosting (Multiplay)	Multiplayer hosting for any engine	Launch
GAMING SERVICES Lobby	Connect players in public or private lobbies	Launch
GAMING SERVICES Matchmaker	Player matchmaking	Launch
GAMING SERVICES Relay	Peer-to-peer multiplayer experiences	Launch

3. Launch full game version from **\Scenes\BootScene.scene**
or demo version from **\Scenes\Demo\Demo.scene**

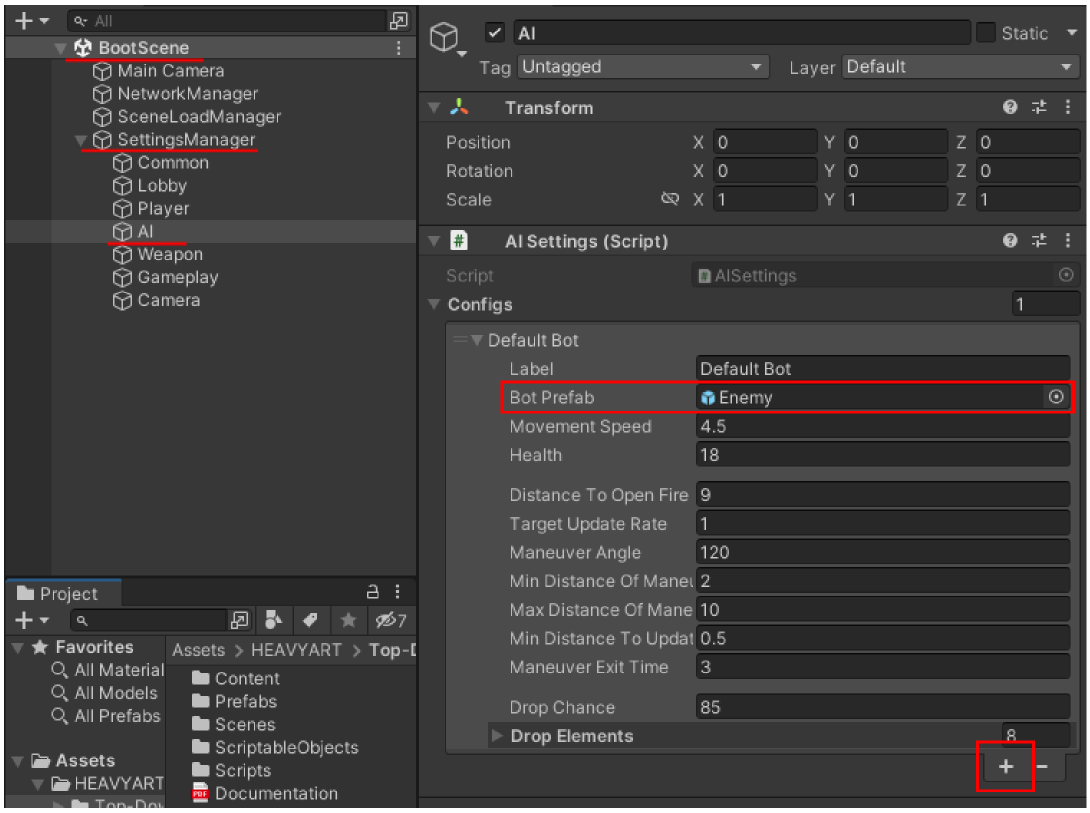
To launch full game for 1 player, press "START GAME",
set "NUMBER OF PLAYERS" to 1 (move slider left) and press "START PUBLIC GAME"



How To

How to add new bot
- Open boot scene.
- Find SettingsManager. Select AI child object.
- Press **+** below Configs collection. It will automatically copy last bot config and add it to the end of Configs collection. Now, it's possible to change its settings.

To add new AI prefab:
- Copy existing one and put its copy to Bot Prefab field.
- Add new prefab to NetworkPrefabsList at \ScriptableObjects\



AI | Static
Tag Untagged | Layer Default

Position	X 0	Y 0	Z 0
Rotation	X 0	Y 0	Z 0
Scale	X 1	Y 1	Z 1

Script: AISettings
Configs: 1
Default Bot

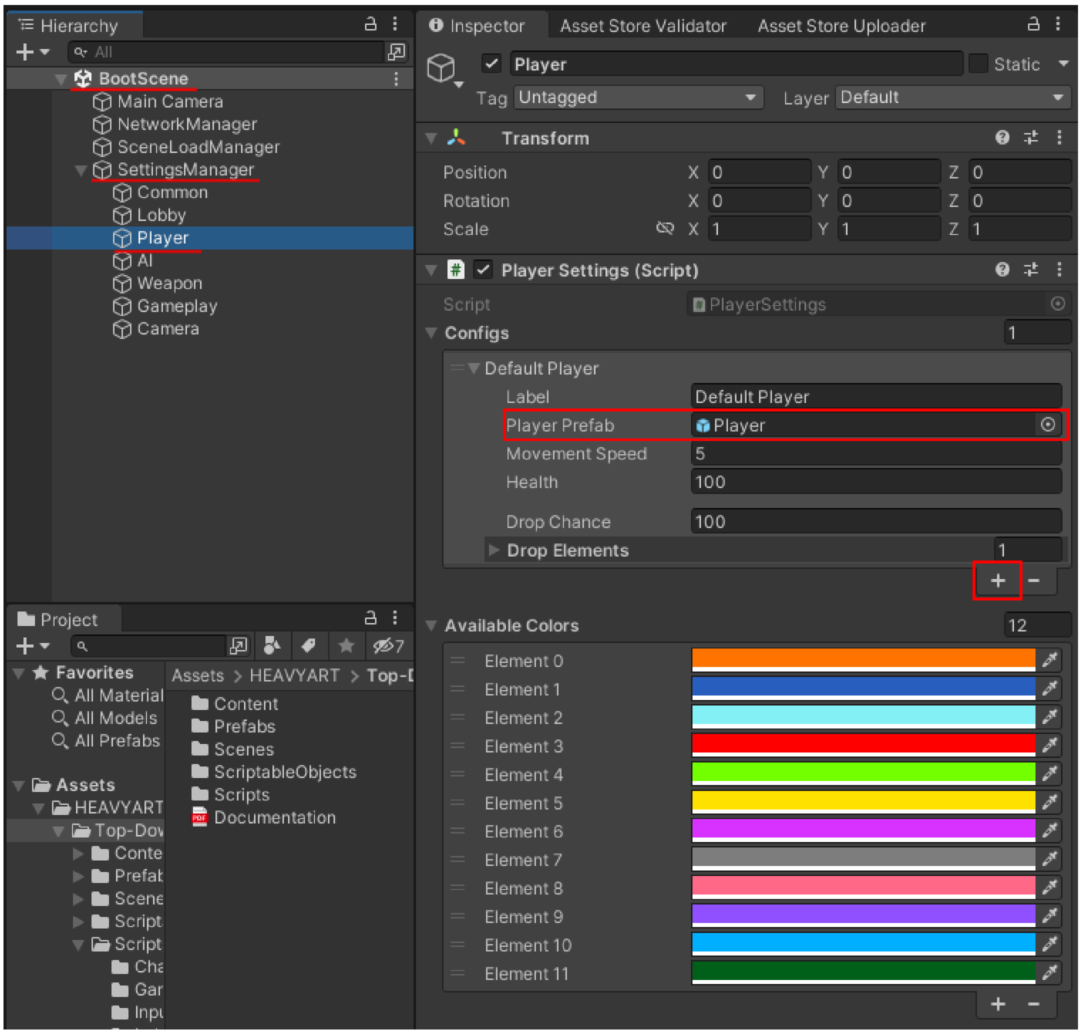
Label	Default Bot
Bot Prefab	Enemy
Movement Speed	4.5
Health	18
Distance To Open Fire	9
Target Update Rate	1
Maneuver Angle	120
Min Distance Of Maneu	2
Max Distance Of Mane	10
Min Distance To Updat	0.5
Maneuver Exit Time	3
Drop Chance	85
Drop Elements	8

When game spawns a bot, it selects one, randomly from the configs list.
For more details, check source code (NetworkObjectsSpawner.cs).

How to add new player
- Open boot scene.
- Find SettingsManager. Select Player child object.
- Press **+** below Configs collection. It will automatically copy last player config and add it to the end of Configs collection. Now, its possible change it's settings.

To add new player prefab:
- Copy existing one, and put its copy to Player Prefab field.
- Add new prefab to NetworkPrefabsList at \ScriptableObjects\

To select player, set PlayerDataKeeper.selectedPrefab value, from code, during runtime.
Currently there is only one player in project, and value is always 0.
However, it's ready to expand.
For more details, check source code: PlayerDataKeeper.selectedPrefab property and ServiceUserController.GetLocalPlayerSpawnParameters() method.



Player — Static
Tag Untagged — Layer Default

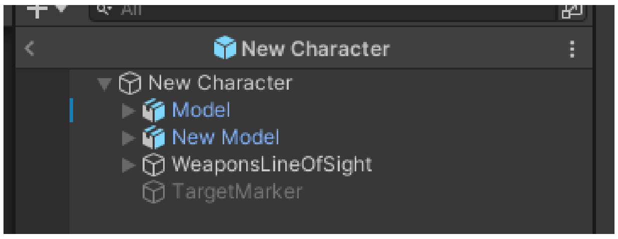
Position X 0 Y 0 Z 0
Rotation X 0 Y 0 Z 0
Scale X 1 Y 1 Z 1

Script PlayerSettings
Configs 1
Default Player
Label: Default Player
Player Prefab: Player
Movement Speed: 5
Health: 100
Drop Chance: 100
Drop Elements: 1

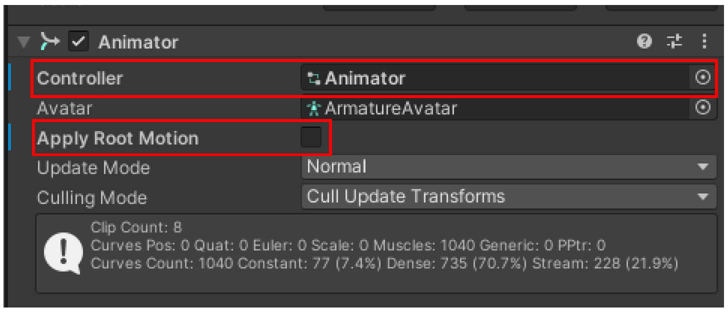
Available Colors 12
Element 0 … Element 11

How to change character model (Player, Enemy)

- Open character prefab.
- Put a new character model, next to the old one. Set its position and rotation to zero.



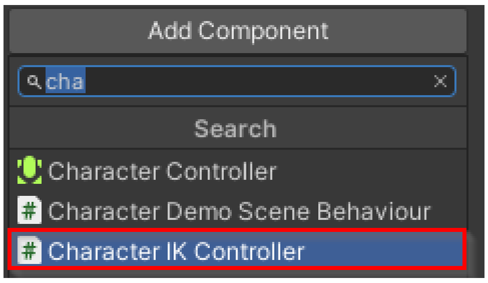
- Copy animator controller from old model to the new model and turn off root motion.



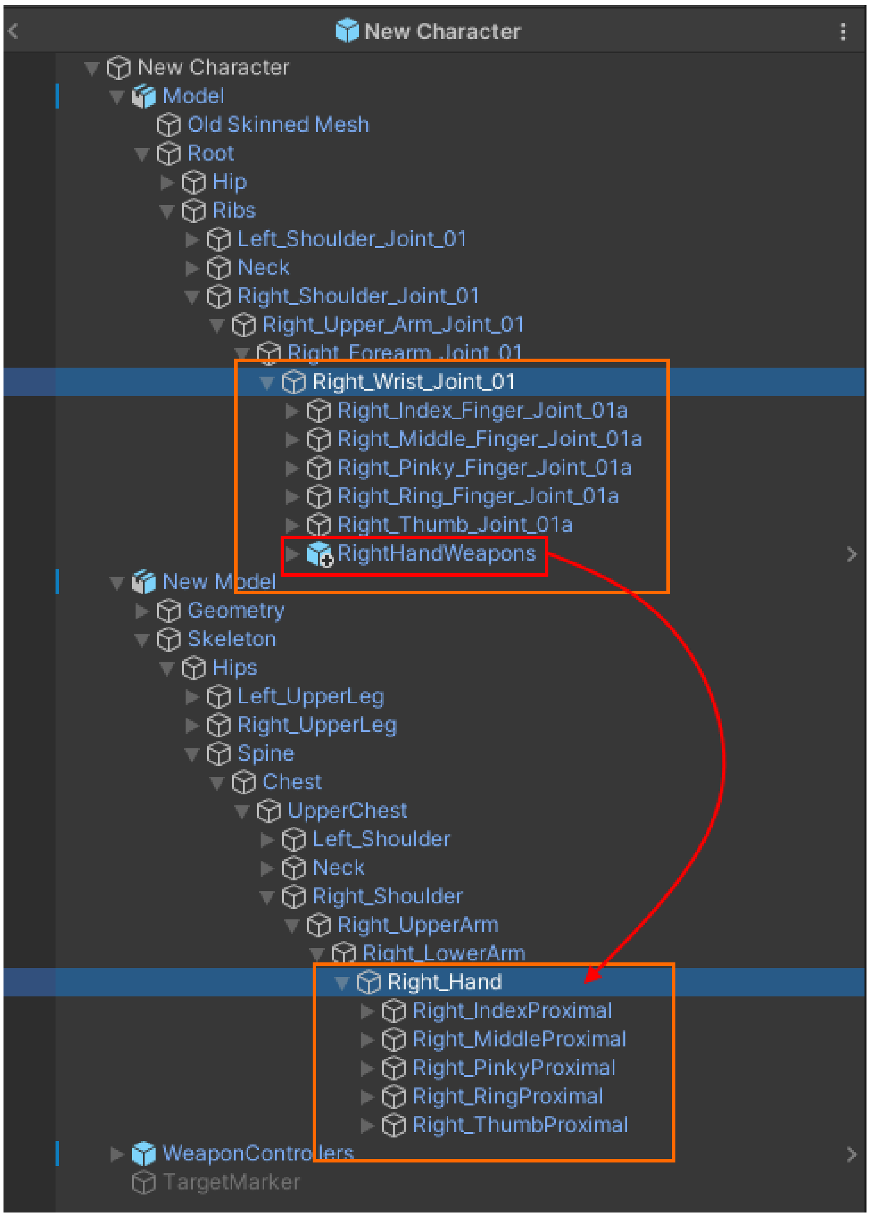
Controller: Animator
Avatar: ArmatureAvatar
Apply Root Motion: ☐
Update Mode: Normal
Culling Mode: Cull Update Transforms

```
Clip Count: 8
Curves Pos: 0 Quat: 0 Euler: 0 Scale: 0 Muscles: 1040 Generic: 0 PPtr: 0
Curves Count: 1040 Constant: 77 (7.4%) Dense: 735 (70.7%) Stream: 228 (21.9%)
```

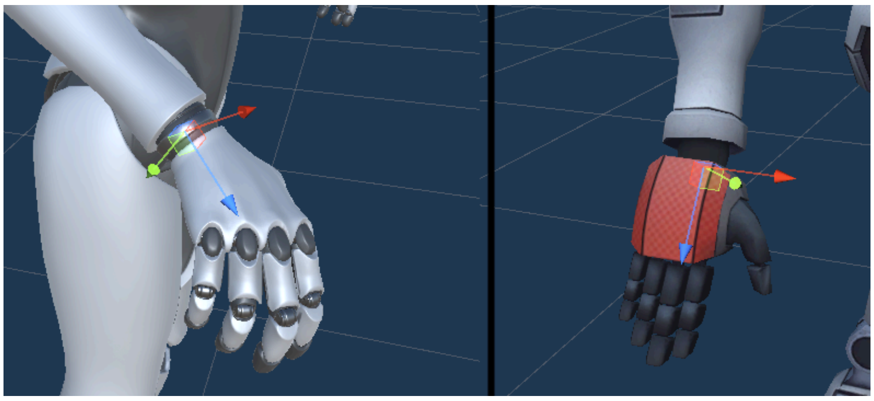
- Add a CharacterIKController component to a new character model. It will be used to attach model's left hand to the selected weapon's grip using Inverse Kinematics.



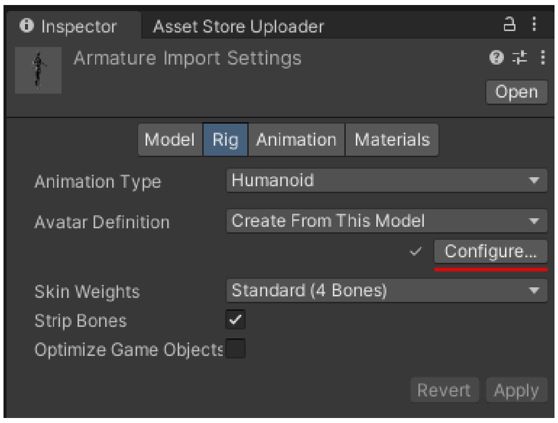
- Drag RightHandWeapons object, from old model's right hand (wrist), to new model's right hand (wrist):



- Set RightHandWeapons position and rotation to zero.
- Make sure RightHandWeapons gizmo pointed in the right direction. Rotate it, if it isn't.

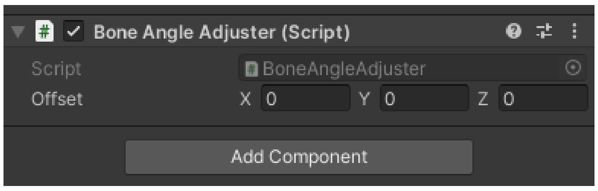


If something still looks wrong, we recommend configuring required bones in the Rig tab of the character's model or use **BoneAngleAdjuster** (details below).
Try to rotate required bone in avatar mapping tab and press “Done”. Changes will be applied in runtime.



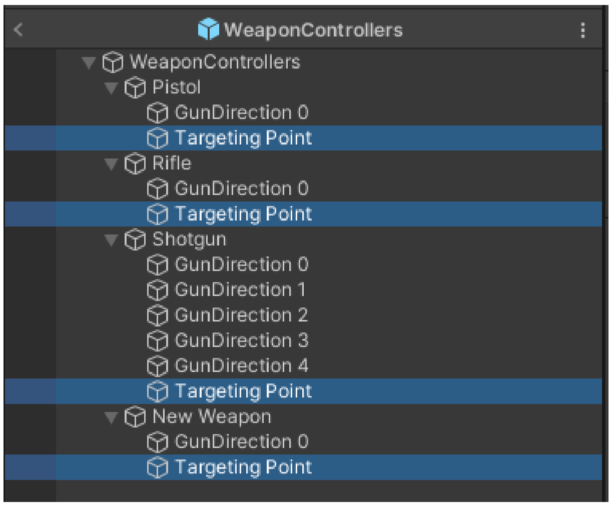
Inspector | Asset Store Uploader
Armature Import Settings
Open
Model | Rig | Animation | Materials
Animation Type: Humanoid
Avatar Definition: Create From This Model
✓ Configure...
Skin Weights: Standard (4 Bones)
Strip Bones: ✓
Optimize Game Objects: ☐
Revert | Apply

Caution: rig configuration would be saved, even after turning off play mode.
In addition to rig configuration, project includes **BoneAngleAdjuster** - simple tool for adjusting bone rotation in runtime. It’s designed to add offset to existing bone rotation.



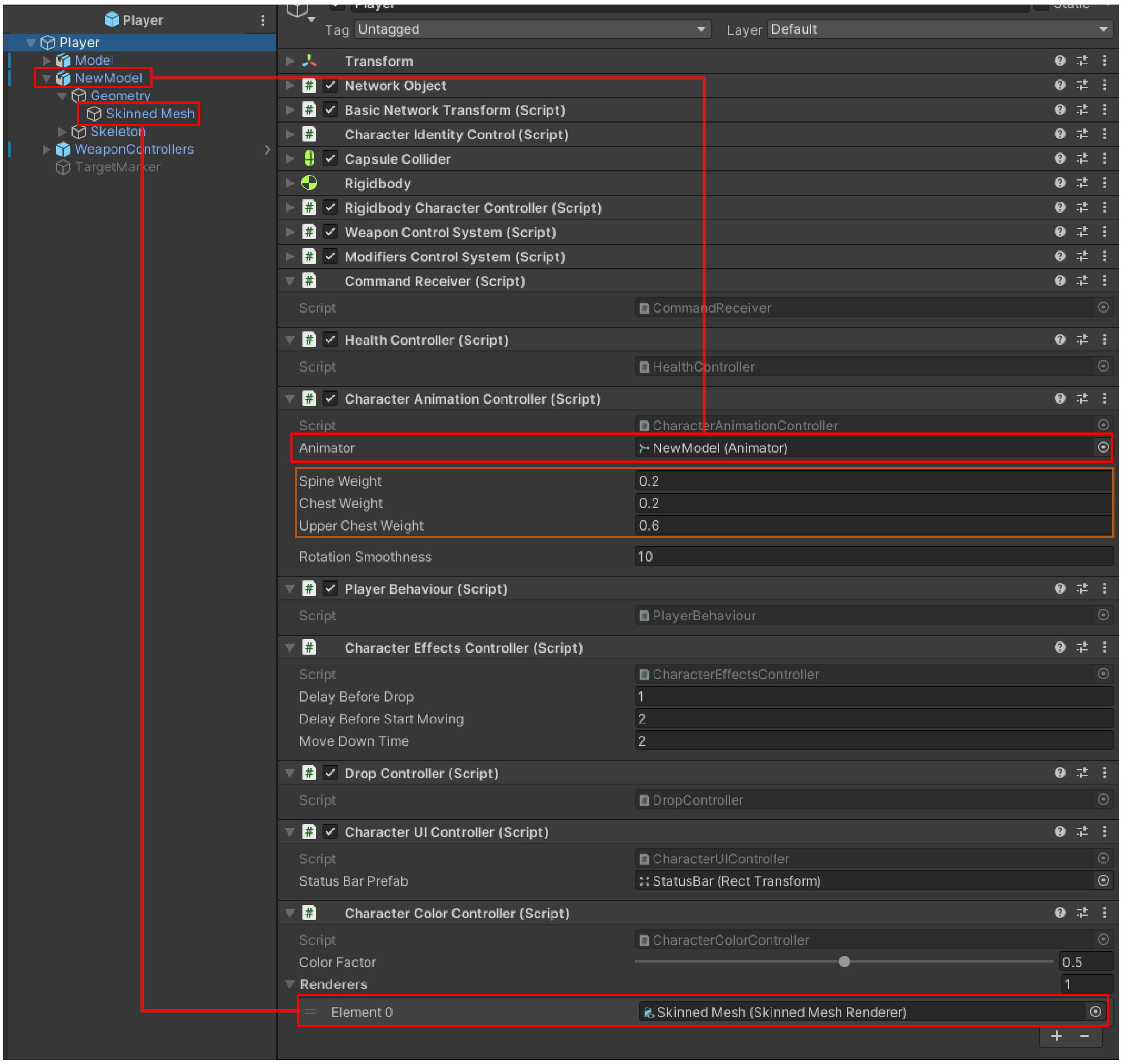
Bone Angle Adjuster (Script)
Script: BoneAngleAdjuster
Offset: X 0 Y 0 Z 0
Add Component

For correction of weapon aiming direction, every character contains Targeting Point transform, parrented to WeaponControllers. It makes it possible to adjust aiming offset for every weapon separately.



```
WeaponControllers
▼ WeaponControllers
  ▼ Pistol
      GunDirection 0
      Targeting Point
  ▼ Rifle
      GunDirection 0
      Targeting Point
  ▼ Shotgun
      GunDirection 0
      GunDirection 1
      GunDirection 2
      GunDirection 3
      GunDirection 4
      Targeting Point
  ▼ New Weapon
      GunDirection 0
      Targeting Point
```

- Update Animator field in Character Animation Controller with new model’s animator. (image below)
- If it’s a player, update Renderers collection in Character Color Controller with new model’s renderers. (AI doesn’t have Character Color Controller)
- Set up Spine Weight, Chest Weight, Upper Chest Weight parameters in Character Animation Controller.



```
Player
  Player
    Model
    NewModel
      Geometry
        Skinned Mesh
      Skeleton
    WeaponControllers
    TargetMarker
```

Tag Untagged   Layer Default

Transform
Network Object
Basic Network Transform (Script)
Character Identity Control (Script)
Capsule Collider
Rigidbody
Rigidbody Character Controller (Script)
Weapon Control System (Script)
Modifiers Control System (Script)

Command Receiver (Script)

Script	CommandReceiver

Health Controller (Script)

Script	HealthController

Character Animation Controller (Script)

Script	CharacterAnimationController
Animator	NewModel (Animator)
Spine Weight	0.2
Chest Weight	0.2
Upper Chest Weight	0.6
Rotation Smoothness	10

Player Behaviour (Script)

Script	PlayerBehaviour

Character Effects Controller (Script)

Script	CharacterEffectsController
Delay Before Drop	1
Delay Before Start Moving	2
Move Down Time	2

Drop Controller (Script)

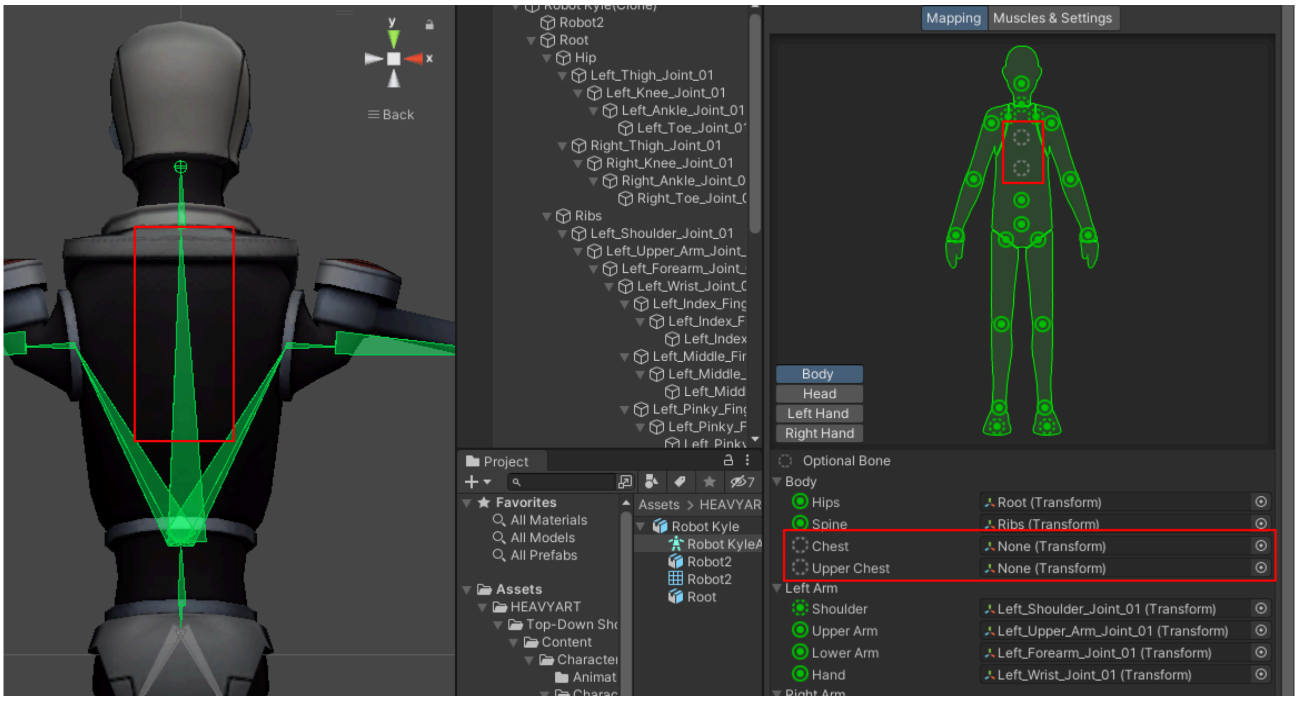
Script	DropController

Character UI Controller (Script)

Script	CharacterUIController
Status Bar Prefab	StatusBar (Rect Transform)

Character Color Controller (Script)

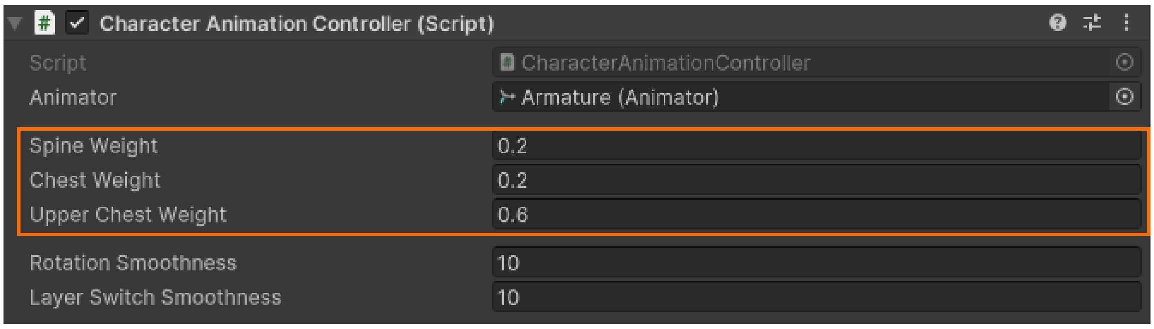
Script	CharacterColorController
Color Factor	0.5
Renderers	1
Element 0	Skinned Mesh (Skinned Mesh Renderer)



```
Robot2
Root
  Hip
    Left_Thigh_Joint_01
      Left_Knee_Joint_01
        Left_Ankle_Joint_01
    Right_Thigh_Joint_01
      Right_Knee_Joint_01
  Ribs
    Left_Shoulder_Joint_01
```

Optional Bone	
Body	
Hips	Root (Transform)
Spine	Ribs (Transform)
Chest	None (Transform)
Upper Chest	None (Transform)
Left Arm	
Shoulder	Left_Shoulder_Joint_01 (Transform)
Upper Arm	Left_Upper_Arm_Joint_01 (Transform)
Lower Arm	Left_Forearm_Joint_01 (Transform)
Hand	Left_Wrist_Joint_01 (Transform)

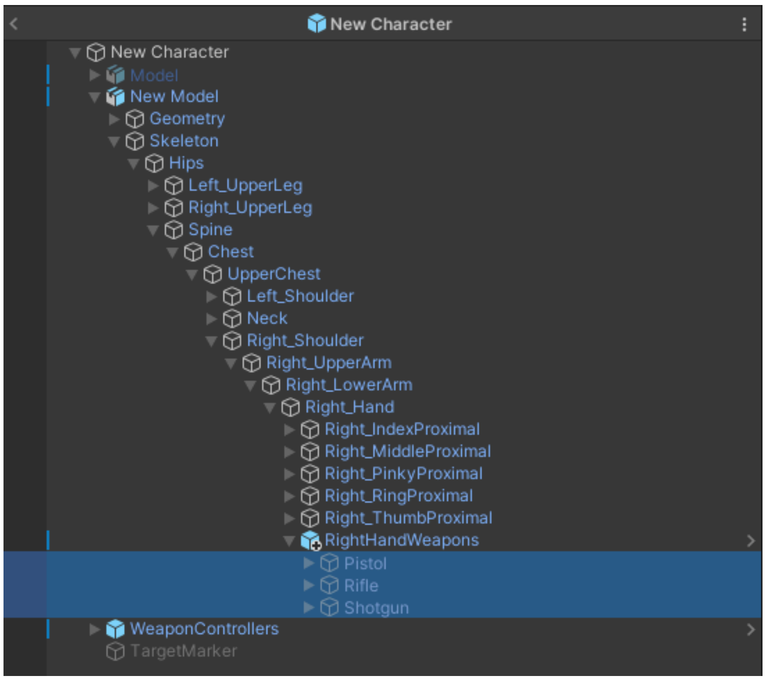
Some skeletons could miss spine bones, like a player's robot model. In this case we recommend to set weights to left existing bones.
Otherwise we recommend to spread weights through all spine bones.



Character Animation Controller (Script)

Script	CharacterAnimationController
Animator	Armature (Animator)
Spine Weight	0.2
Chest Weight	0.2
Upper Chest Weight	0.6
Rotation Smoothness	10
Layer Switch Smoothness	10

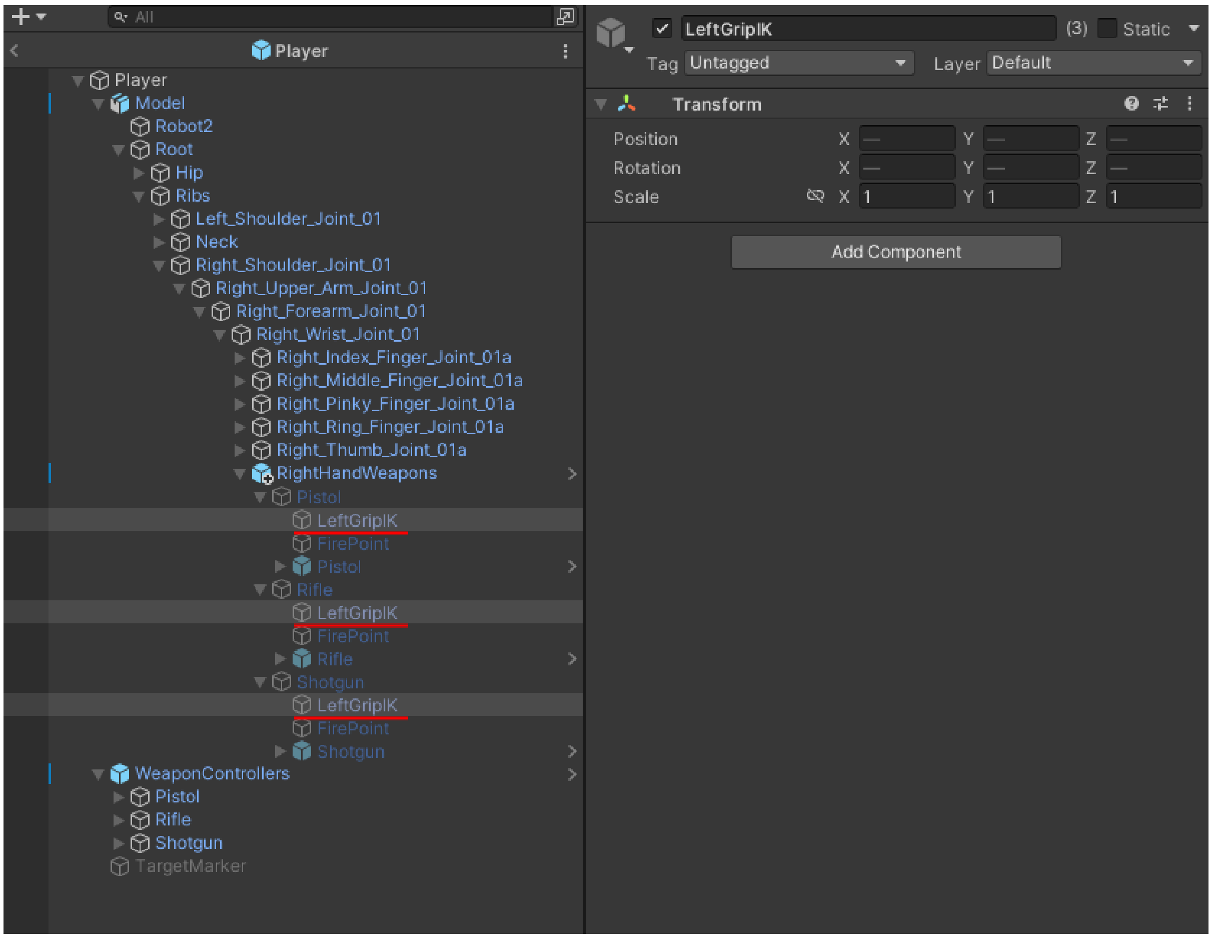
- Set up positions and rotations for weapon models (located in the right hand).
Note: Don't leave RightHandWeapons child objects turned on, otherwise they will be shown in game, from beginning.



```
New Character
New Character
  Model
  New Model
    Geometry
    Skeleton
      Hips
        Left_UpperLeg
        Right_UpperLeg
        Spine
          Chest
            UpperChest
              Left_Shoulder
              Neck
              Right_Shoulder
                Right_UpperArm
                  Right_LowerArm
                    Right_Hand
                      Right_IndexProximal
                      Right_MiddleProximal
                      Right_PinkyProximal
                      Right_RingProximal
                      Right_ThumbProximal
                      RightHandWeapons
                        Pistol
                        Rifle
                        Shotgun
  WeaponControllers
  TargetMarker
```

- Now, when everything is done, we can remove old model and save prefab.

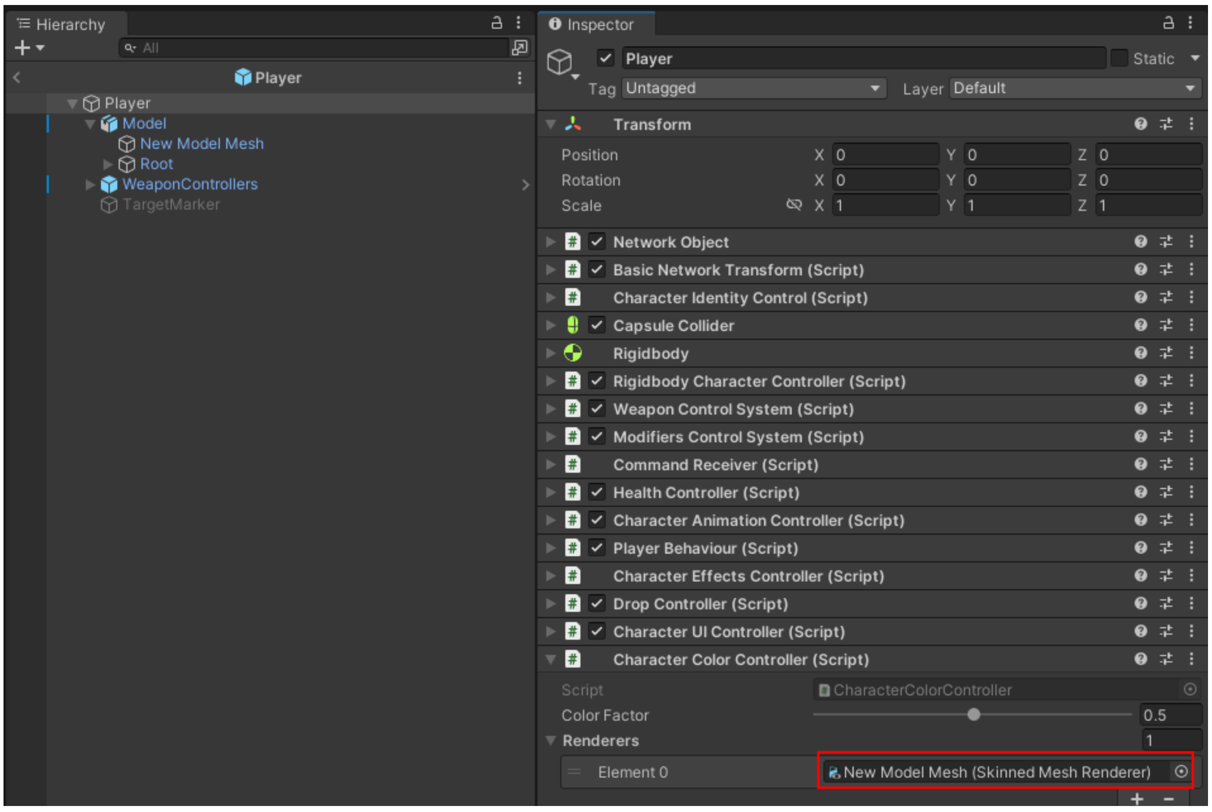
Note: In case if you want to further adjust left hand position of the model, you can tweak LeftGripIK position and rotation for each weapon.



```
Player
  Model
    Robot2
    Root
      Hip
      Ribs
        Left_Shoulder_Joint_01
        Neck
        Right_Shoulder_Joint_01
          Right_Upper_Arm_Joint_01
            Right_Forearm_Joint_01
              Right_Wrist_Joint_01
                Right_Index_Finger_Joint_01a
                Right_Middle_Finger_Joint_01a
                Right_Pinky_Finger_Joint_01a
                Right_Ring_Finger_Joint_01a
                Right_Thumb_Joint_01a
                RightHandWeapons
                  Pistol
                    LeftGripIK
                    FirePoint
                    Pistol
                  Rifle
                    LeftGripIK
                    FirePoint
                    Rifle
                  Shotgun
                    LeftGripIK
                    FirePoint
                    Shotgun
  WeaponControllers
    Pistol
    Rifle
    Shotgun
  TargetMarker
```

```
LeftGripIK (3)   Static
Tag Untagged   Layer Default
Transform
Position  X —  Y —  Z —
Rotation  X —  Y —  Z —
Scale     X 1  Y 1  Z 1
Add Component
```

- To make the new model change its color, link character’s mesh with **CharacterColorController** component.



Player
- Player
  - Model
    - New Model Mesh
    - Root
  - WeaponControllers
  - TargetMarker

Player — Static
Tag Untagged — Layer Default

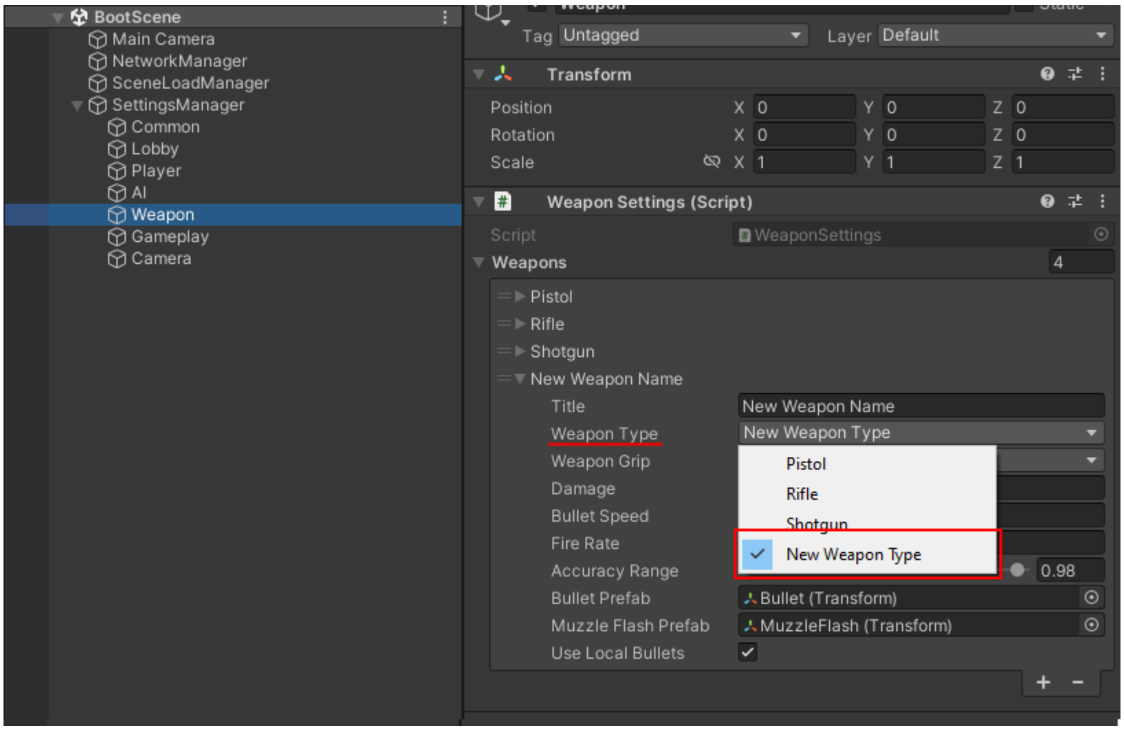
Transform
Position X 0 Y 0 Z 0
Rotation X 0 Y 0 Z 0
Scale X 1 Y 1 Z 1

Network Object
Basic Network Transform (Script)
Character Identity Control (Script)
Capsule Collider
Rigidbody
Rigidbody Character Controller (Script)
Weapon Control System (Script)
Modifiers Control System (Script)
Command Receiver (Script)
Health Controller (Script)
Character Animation Controller (Script)
Player Behaviour (Script)
Character Effects Controller (Script)
Drop Controller (Script)
Character UI Controller (Script)
Character Color Controller (Script)

Script CharacterColorController
Color Factor 0.5
Renderers 1
Element 0 New Model Mesh (Skinned Mesh Renderer)

Every **CharacterColorController** contains:
Renderers: reference to the mesh renderers that will be repainted.
All the materials related to the mesh are stored in **SkinnedMeshRenderer**(default mesh component).
Color factor: force of repaint (color replacement). 1 means 100%.

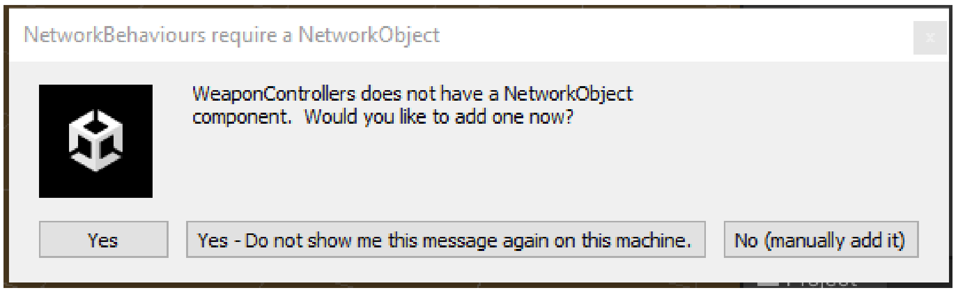
How to add new weapon
- Open WeaponType.cs enum and add new weapon type.
- Open boot scene.
- Find SettingsManager. Select Weapon child object.
- Press **+** below Weapons collection. It will automatically copy last weapon config and add it to the end of Weapons collection.
- Select new defined weapon type in Weapon Type field drop-down menu



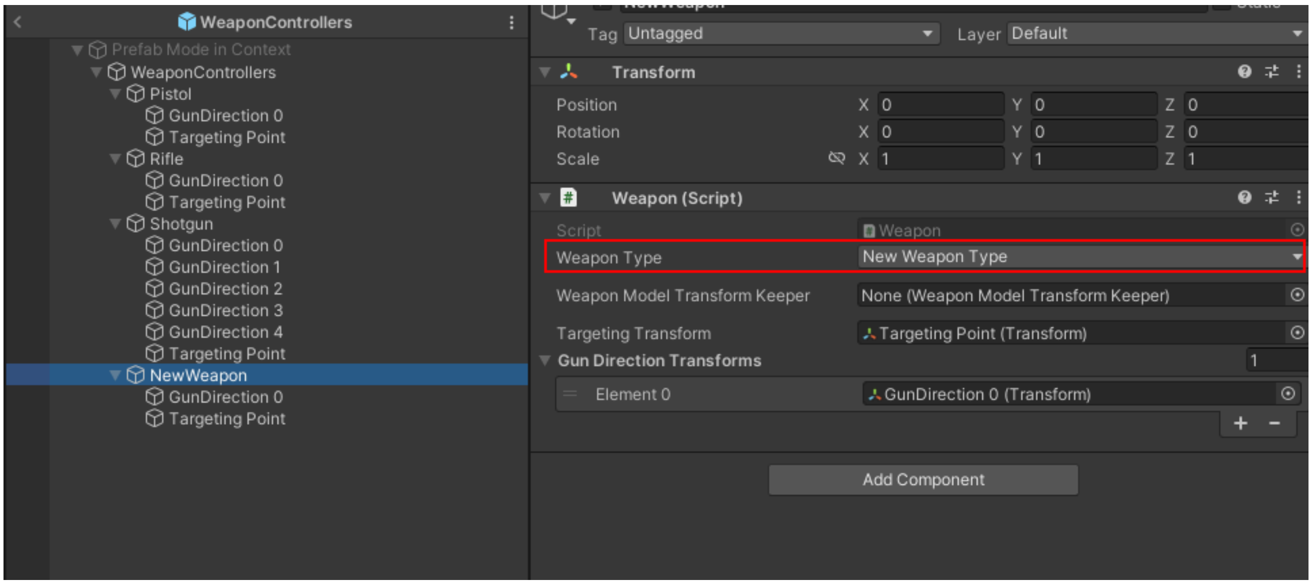
- Open WeaponControllers prefab at \Prefabs\CharacterParts
- Duplicate one of existing weapons and set its Weapon Type to our new type. Here you can add new guns(barrels) to weapons and set up their default directions.
Usual weapons have one barrel, shotgun has five.
WeaponModelTransformKeeper will be set up inside character prefabs, during next steps.

Note: To prevent editor from showing "NetworkBehaviours require a NetworkObject" popup, objects in WeaponControllers with components inherited from **NetworkBehaviour** have a **NetcodeComponentResolver** component.
It adds NetworkObject component on selecting object and removes it on its deselecting.

If something went wrong, please turn off play mode, select required object (click on it) and deselect it (click on a free space or another object).
Resolver will remove **NetworkObject** component on disable event.


NetworkBehaviours require a NetworkObject

WeaponControllers does not have a NetworkObject component. Would you like to add one now?



WeaponControllers

- Prefab Mode in Context
  - WeaponControllers
    - Pistol
      - GunDirection 0
      - Targeting Point
    - Rifle
      - GunDirection 0
      - Targeting Point
    - Shotgun
      - GunDirection 0
      - GunDirection 1
      - GunDirection 2
      - GunDirection 3
      - GunDirection 4
      - Targeting Point
    - NewWeapon
      - GunDirection 0
      - Targeting Point

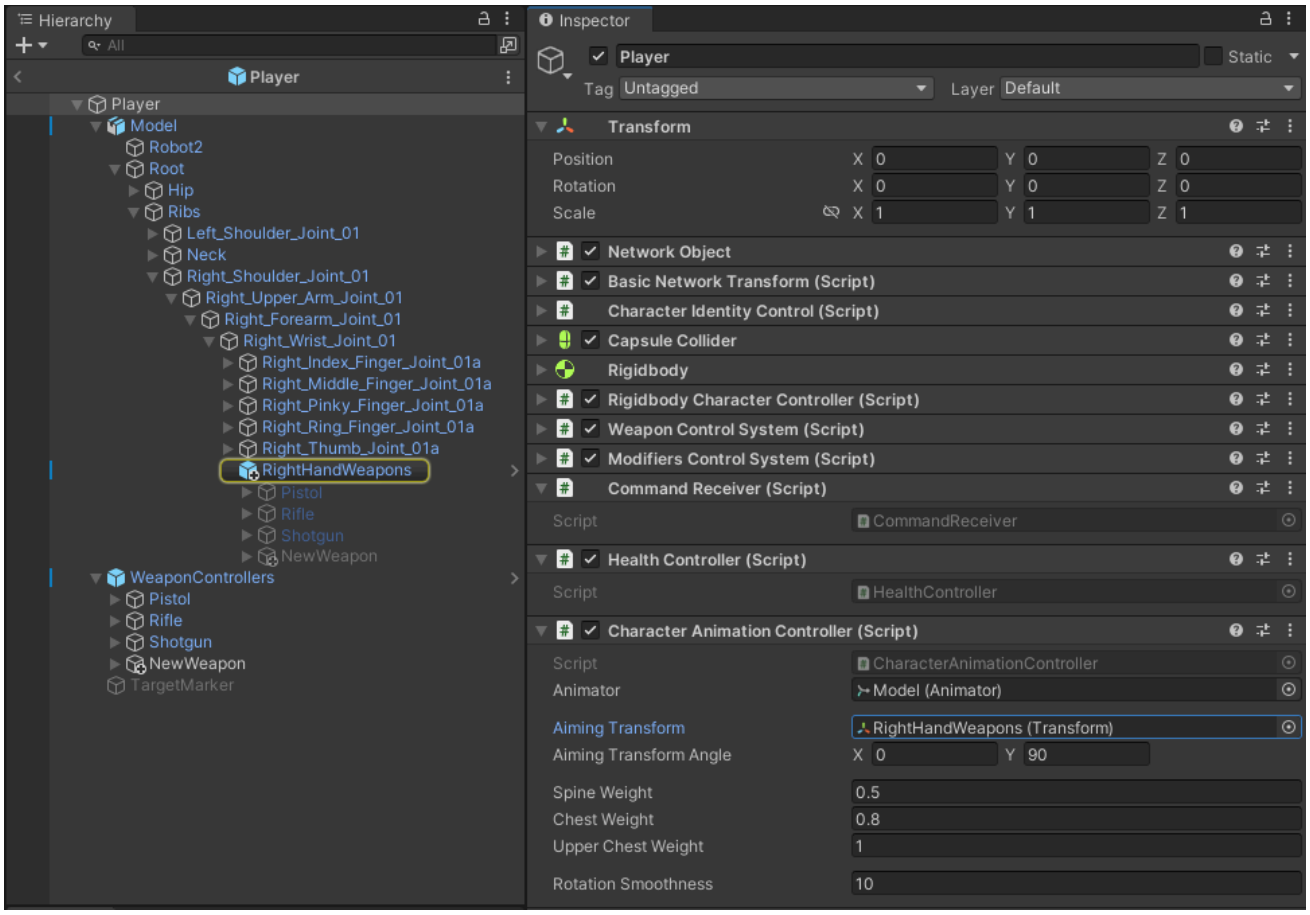
Tag Untagged Layer Default

Transform
Position X 0 Y 0 Z 0
Rotation X 0 Y 0 Z 0
Scale X 1 Y 1 Z 1

Weapon (Script)
Script: Weapon
Weapon Type: New Weapon Type
Weapon Model Transform Keeper: None (Weapon Model Transform Keeper)
Targeting Transform: Targeting Point (Transform)
Gun Direction Transforms: 1
Element 0: GunDirection 0 (Transform)

Add Component

- Open RightHandWeapons prefab at \Prefabs\CharacterParts
- Duplicate one of RightHandWeapons child objects (weapons).
- Open character prefab and find RightHandWeapons object in character's right hand.
It's easy to find by clicking on Aiming Transform Field in CharacterAnimationController.



Hierarchy
Player

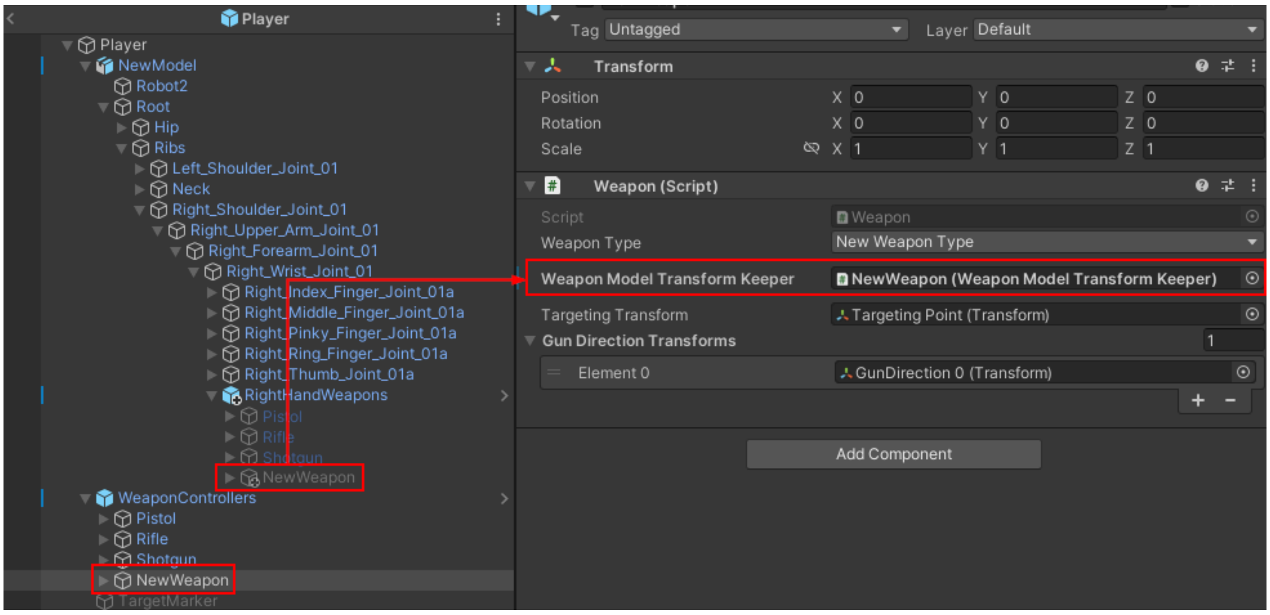
- Player
  - Model
    - Robot2
    - Root
      - Hip
      - Ribs
        - Left_Shoulder_Joint_01
        - Neck
        - Right_Shoulder_Joint_01
          - Right_Upper_Arm_Joint_01
            - Right_Forearm_Joint_01
              - Right_Wrist_Joint_01
                - Right_Index_Finger_Joint_01a
                - Right_Middle_Finger_Joint_01a
                - Right_Pinky_Finger_Joint_01a
                - Right_Ring_Finger_Joint_01a
                - Right_Thumb_Joint_01a
                - RightHandWeapons
                  - Pistol
                  - Rifle
                  - Shotgun
                  - NewWeapon
  - WeaponControllers
    - Pistol
    - Rifle
    - Shotgun
    - NewWeapon
  - TargetMarker

Inspector
Player Static
Tag Untagged Layer Default

Transform
Position X 0 Y 0 Z 0
Rotation X 0 Y 0 Z 0
Scale X 1 Y 1 Z 1

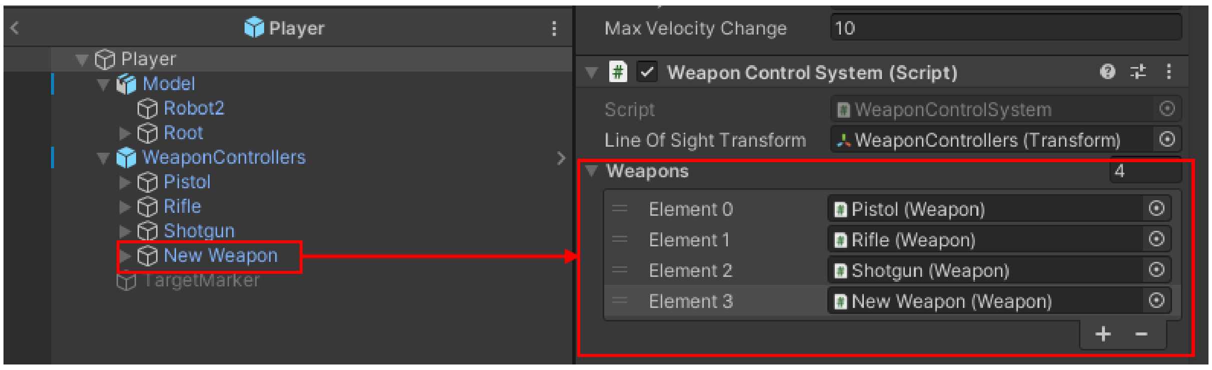
Network Object
Basic Network Transform (Script)
Character Identity Control (Script)
Capsule Collider
Rigidbody
Rigidbody Character Controller (Script)
Weapon Control System (Script)
Modifiers Control System (Script)
Command Receiver (Script)
Script: CommandReceiver
Health Controller (Script)
Script: HealthController
Character Animation Controller (Script)
Script: CharacterAnimationController
Animator: Model (Animator)
Aiming Transform: RightHandWeapons (Transform)
Aiming Transform Angle X 0 Y 90
Spine Weight 0.5
Chest Weight 0.8
Upper Chest Weight 1
Rotation Smoothness 10

- Link our new weapon from RightHandWeapons to our new weapon from Weapon Controllers, as shown on screenshot (drag and drop an object).
This is how we let weapons control know, where graphics, fire point and left IK grip are stored.
Every weapon in RightHandWeapons contains WeaponModelTransformKeeper component.

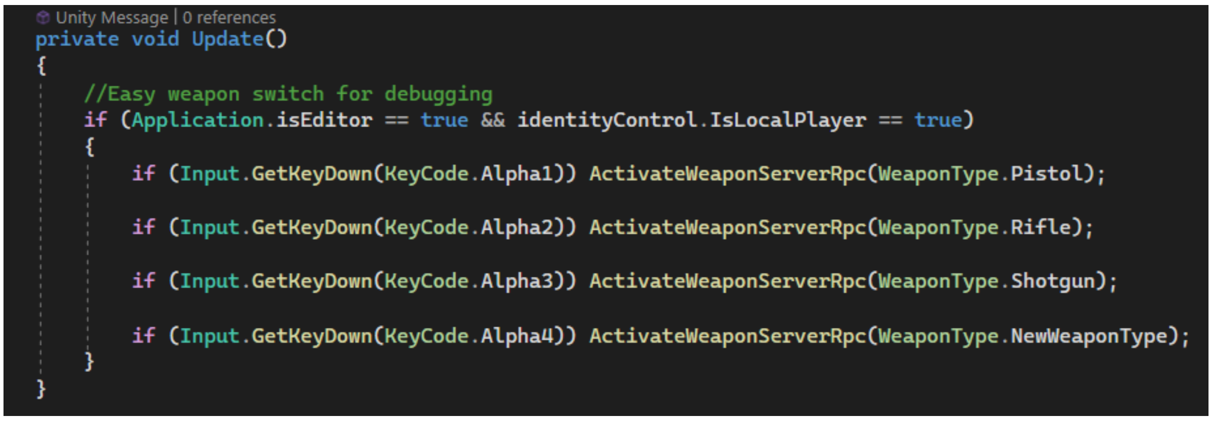


Note: Don’t leave RightHandWeapons child objects turned on, otherwise they will be shown in game, from beginning.

- Drag and drop our new weapon from Weapon Controllers to Weapons collection in WeaponControlSystem component on character root object.

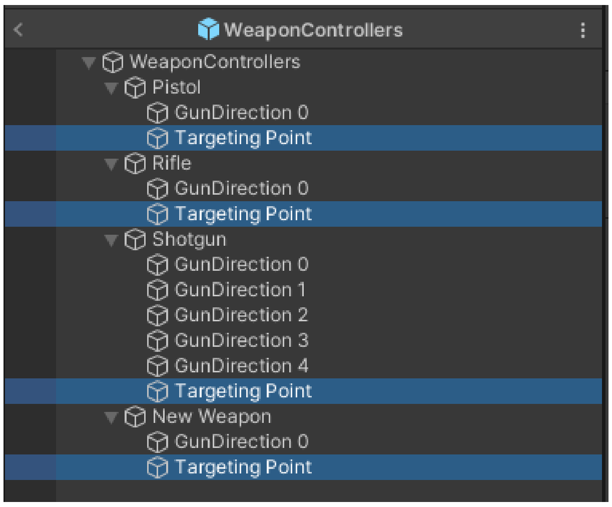

Player
Player
Model
Robot2
Root
WeaponControllers
Pistol
Rifle
Shotgun
New Weapon
TargetMarker
Max Velocity Change 10
Weapon Control System (Script)
Script WeaponControlSystem
Line Of Sight Transform WeaponControllers (Transform)
Weapons 4
Element 0 Pistol (Weapon)
Element 1 Rifle (Weapon)
Element 2 Shotgun (Weapon)
Element 3 New Weapon (Weapon)

- Check weapons with debug commands, stored in WeaponControlSystem.cs



```
Unity Message | 0 references
private void Update()
{
    //Easy weapon switch for debugging
    if (Application.isEditor == true && identityControl.IsLocalPlayer == true)
    {
        if (Input.GetKeyDown(KeyCode.Alpha1)) ActivateWeaponServerRpc(WeaponType.Pistol);

        if (Input.GetKeyDown(KeyCode.Alpha2)) ActivateWeaponServerRpc(WeaponType.Rifle);

        if (Input.GetKeyDown(KeyCode.Alpha3)) ActivateWeaponServerRpc(WeaponType.Shotgun);

        if (Input.GetKeyDown(KeyCode.Alpha4)) ActivateWeaponServerRpc(WeaponType.NewWeaponType);
    }
}
```

Note: For correction of weapon vertical aiming direction, every character contains Targeting Point transform, parrented to WeaponControllers. It makes it possible to adjust aiming offset for every weapon.



How to add another game scene
- Duplicate existing GameScene. Change anything in Environment object.
- Add scene to Build Settings
- Set PlayerDataKeeper.selectedScene value, from code, during runtime.
- Start new game session. All joined clients will automatically load scene after you.

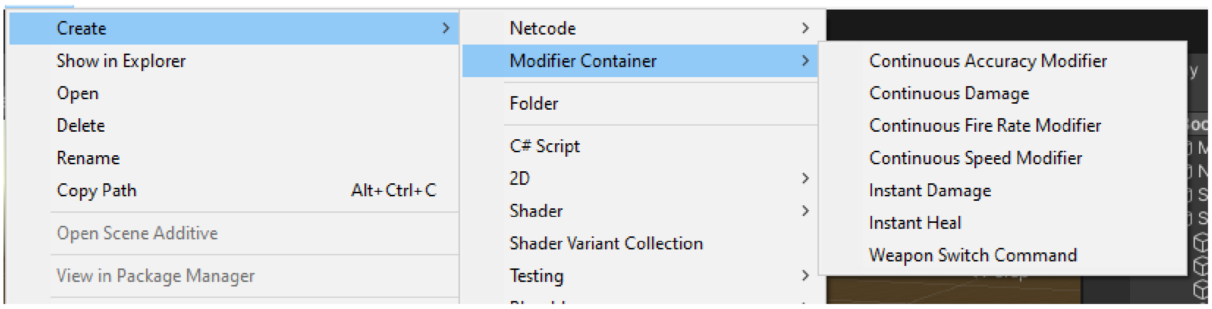
Currently there is only one scene in project, and value is always “GameScene”.
However, it's ready to expand.

How to add new pick-up object
Pick-ups are made from two parts: Drop Item (prefab) and Scriptable Object with command.
Project contains Modifiers Control System: solution similar to Command and Visitor patterns.
It used to broadcast and process commands from picking up objects (weapons, power-ups, medkit) and receiving damage.
We can use already prepared commands and customize them, or create new (instructions below).

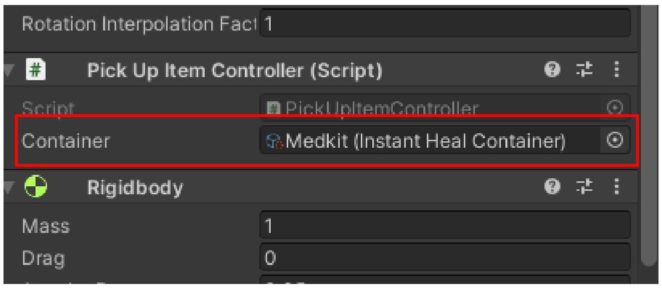
Existing commands	Description
Continuous Accuracy Modifier	Accuracy power-up with exit time
Continuous Damage	Damage (Fire, Poison, etc) with exit time
Continuous Fire Rate Modifier	Fire rate power-up with exit time
Continuous Speed Modifier	Movement speed power-up with exit time
Instant Damage	Damage. Used as bullet damage. Could be mines, etc
Instant Heal	Medkit
Weapon Switch Command	Weapon pick-up

Scriptable objects used with pick-ups stored at \ScriptableObjects\Drop Configs.
Pick-up prefabs stored at \Prefabs\Drop

- Create scriptable object from menu: Assets > Create > Modifier Container



- Copy one of existing prefabs at \Prefabs\Drop, and replace its scriptable object (in Container field) with the newly created scriptable object.



Rotation Interpolation Fact 1

Pick Up Item Controller (Script)

Script PickUpItemController

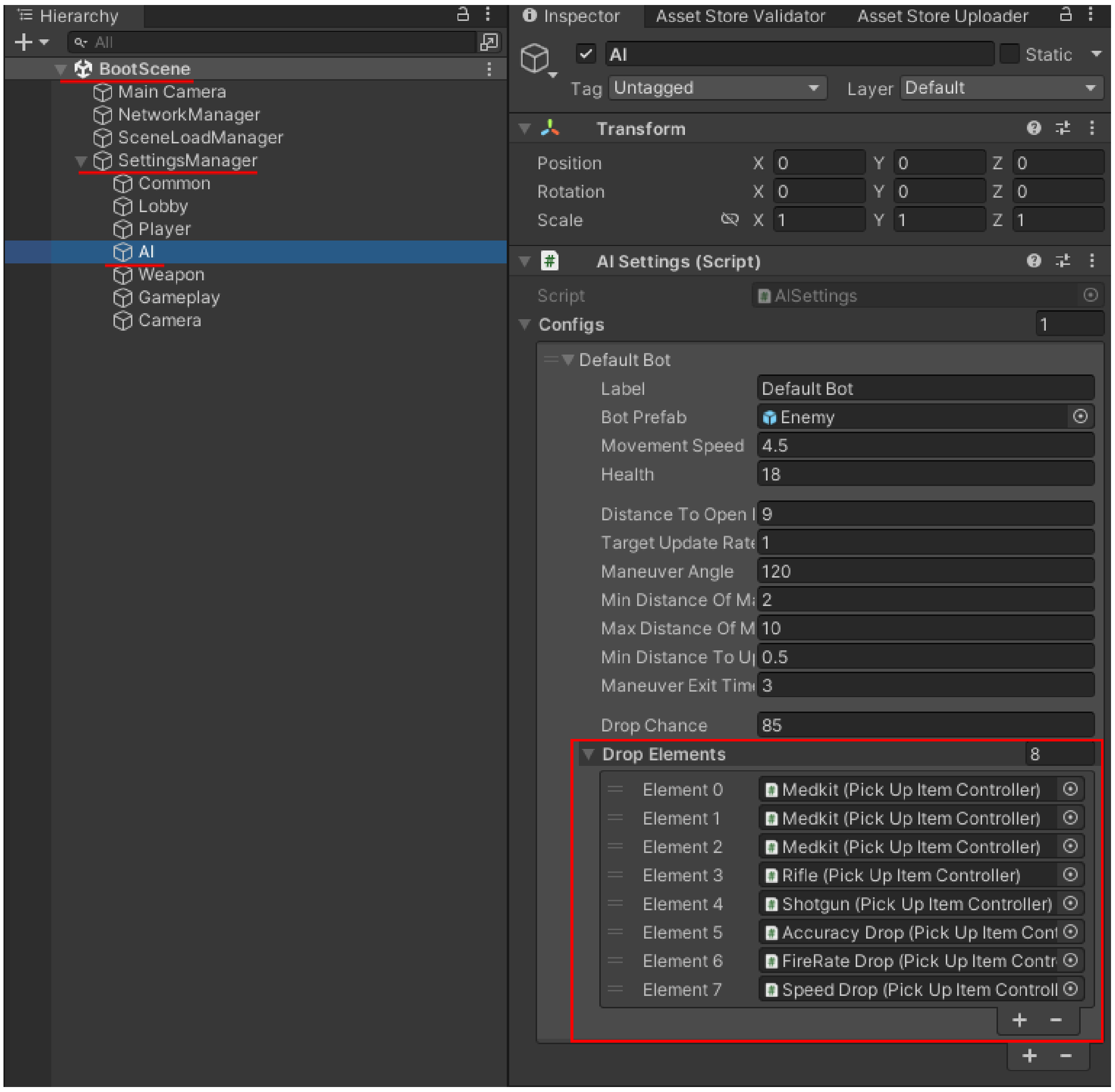
Container Medkit (Instant Heal Container)

Rigidbody

Mass 1

Drag 0

- Open boot scene. Find SettingsManager. Select AI child object.
- Add newly created prefab to AI Settings > Configs > Drop Elements collection.
- Add prefab to Network Prefabs List. \ScriptableObjects\NetworkPrefabsList.asset
If you want to increase chances for this drop to be instantiated, add it few times.



Hierarchy

- BootScene
  - Main Camera
  - NetworkManager
  - SceneLoadManager
  - SettingsManager
    - Common
    - Lobby
    - Player
    - AI
    - Weapon
    - Gameplay
    - Camera

Inspector | Asset Store Validator | Asset Store Uploader

AI — Static

Tag Untagged — Layer Default

Transform

	X	Y	Z
Position	0	0	0
Rotation	0	0	0
Scale	1	1	1

AI Settings (Script)

Script AISettings

Configs 1

Default Bot

Label	Default Bot
Bot Prefab	Enemy
Movement Speed	4.5
Health	18
Distance To Open I	9
Target Update Rate	1
Maneuver Angle	120
Min Distance Of Ma	2
Max Distance Of M	10
Min Distance To Up	0.5
Maneuver Exit Time	3
Drop Chance	85

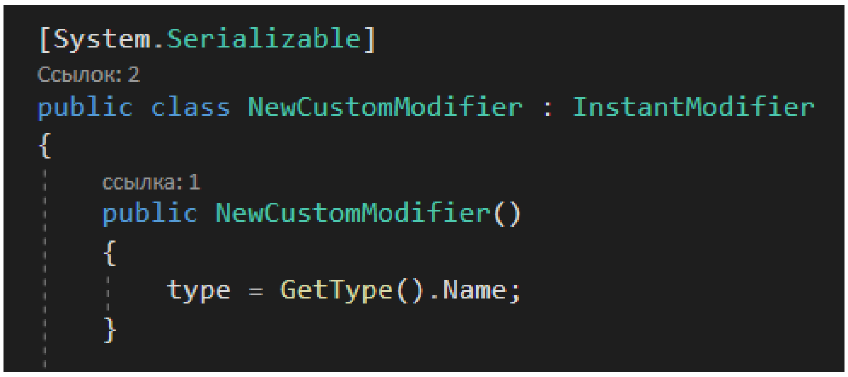
Drop Elements 8

Element 0	Medkit (Pick Up Item Controller)
Element 1	Medkit (Pick Up Item Controller)
Element 2	Medkit (Pick Up Item Controller)
Element 3	Rifle (Pick Up Item Controller)
Element 4	Shotgun (Pick Up Item Controller)
Element 5	Accuracy Drop (Pick Up Item Con
Element 6	FireRate Drop (Pick Up Item Contr
Element 7	Speed Drop (Pick Up Item Controll

How to create new modifiers/commands
Project contains Modifiers Control System: solution similar to Command and Visitor patterns. It used to broadcast and process commands from picking up objects (weapons, power-ups, medkit) and receiving damage.

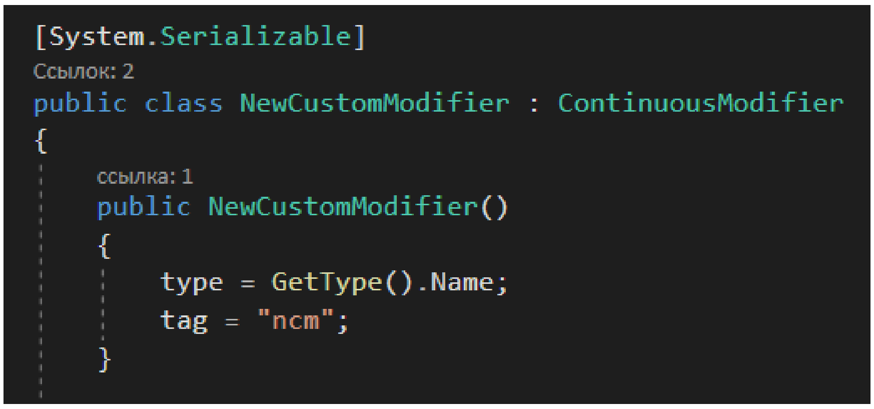
To create your own modifiers/commands follow next steps:
- Create new class and inherit it from **InstantModifier** or **ContinuousModifier**.
InstantModifier is a base class for modifiers with no duration (commands).
ContinuousModifier is a base class for modifiers with duration.
- Add **[Serializable]** attribute above class name.
- Define class constructor like on examples below

Instant modifier:



```
[System.Serializable]
Ссылок: 2
public class NewCustomModifier : InstantModifier
{
    ссылка: 1
    public NewCustomModifier()
    {
        type = GetType().Name;
    }
```

Continuous modifier (contains tag):



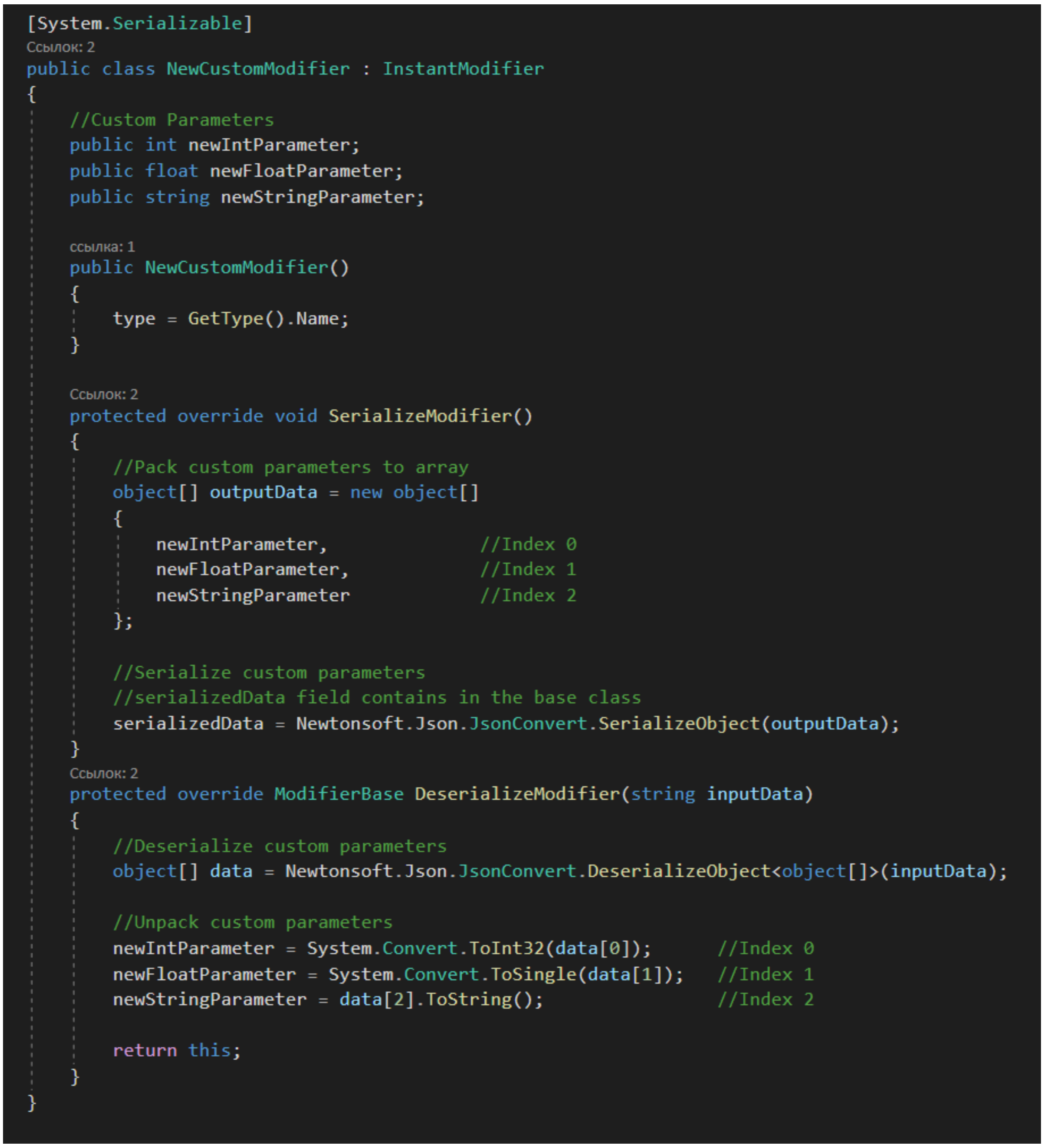
```
[System.Serializable]
Ссылок: 2
public class NewCustomModifier : ContinuousModifier
{
    ссылка: 1
    public NewCustomModifier()
    {
        type = GetType().Name;
        tag = "ncm";
    }
```

All continuous modifiers in project are tagged with first letters of every next word in a name. Tag for **N**ew**C**ustom**M**odifier could be "**ncm**".
Tags required to group continuous modifiers with the same effect into one single modifier with longer activity time.

- Override **SerializeModifier** and **DeserializeModifier** methods.

Important: Keep new modifiers in the same namespace as ModifierBase.

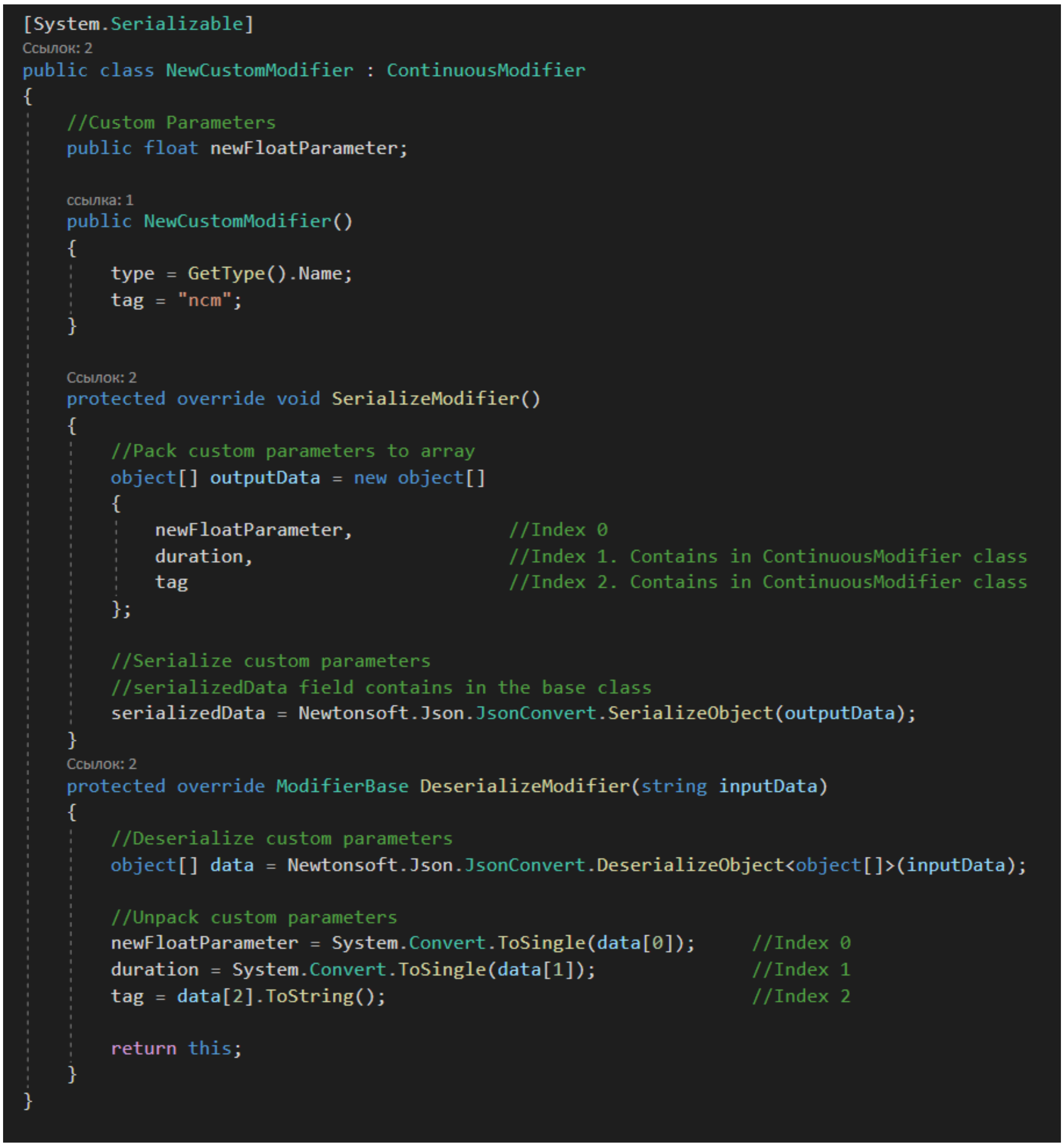
Instant Modifier:



```
[System.Serializable]
Ссылок: 2
public class NewCustomModifier : InstantModifier
{
    //Custom Parameters
    public int newIntParameter;
    public float newFloatParameter;
    public string newStringParameter;

    ссылка: 1
    public NewCustomModifier()
    {
        type = GetType().Name;
    }

    Ссылок: 2
    protected override void SerializeModifier()
    {
        //Pack custom parameters to array
        object[] outputData = new object[]
        {
            newIntParameter,            //Index 0
            newFloatParameter,          //Index 1
            newStringParameter          //Index 2
        };

        //Serialize custom parameters
        //serializedData field contains in the base class
        serializedData = Newtonsoft.Json.JsonConvert.SerializeObject(outputData);
    }
    Ссылок: 2
    protected override ModifierBase DeserializeModifier(string inputData)
    {
        //Deserialize custom parameters
        object[] data = Newtonsoft.Json.JsonConvert.DeserializeObject<object[]>(inputData);

        //Unpack custom parameters
        newIntParameter = System.Convert.ToInt32(data[0]);      //Index 0
        newFloatParameter = System.Convert.ToSingle(data[1]);   //Index 1
        newStringParameter = data[2].ToString();                //Index 2

        return this;
    }
}
```

SerializeModifier and **DeserializeModifier** methods designed to contain instructions to pack and unpack custom data, stored in fields of certain modifiers.
In this example, **SerializeModifier** takes three custom fields (newIntParameter, newFloatParameter, newStringParameter), packs them to object[] and serializes them into JSON string. There could be any other JSON-compatible variables.
Game will use this JSON to send modifier trough network.
DeserializeModifier required to receive modifier and restore it back from JSON.

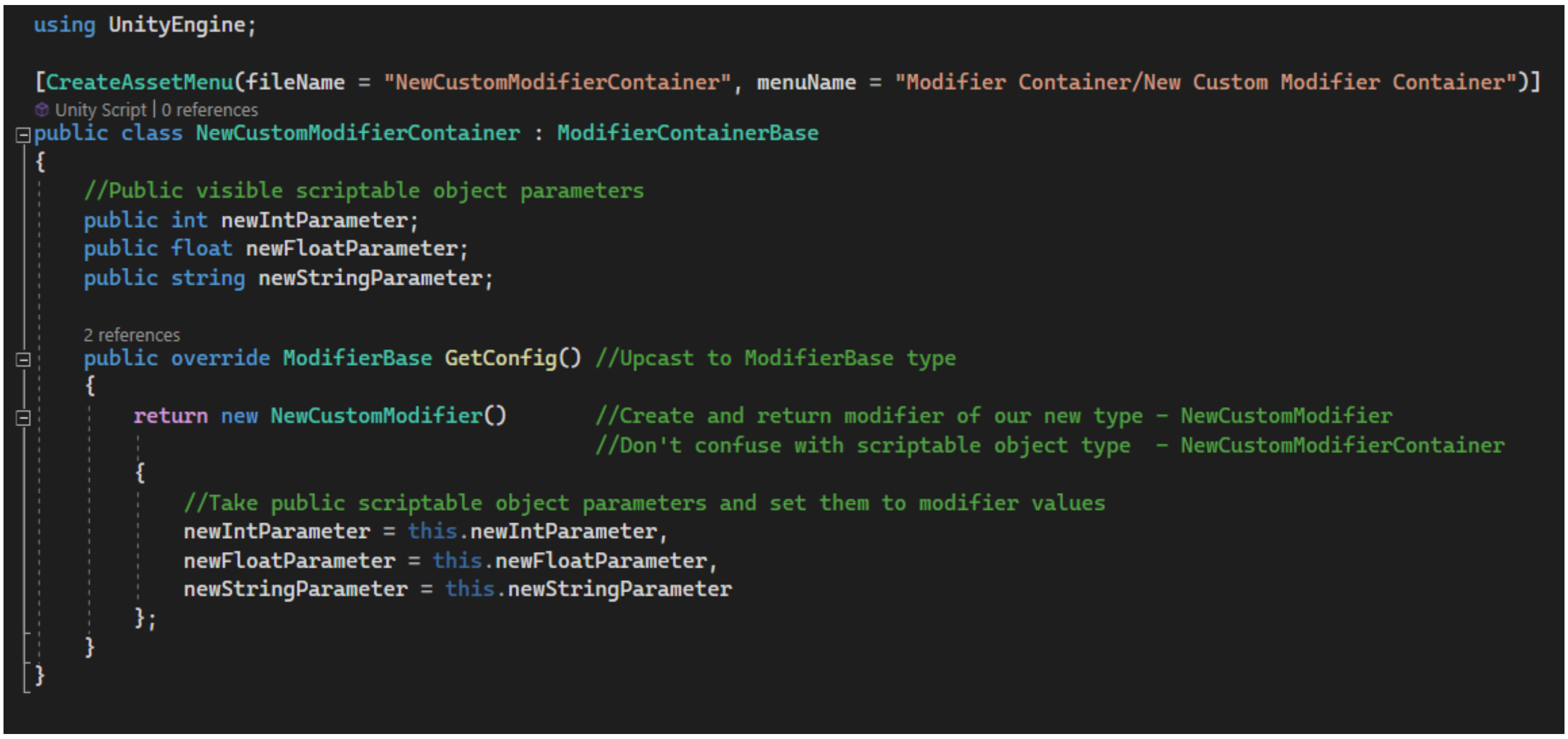
Continuous Modifier:



```
[System.Serializable]
Ссылок: 2
public class NewCustomModifier : ContinuousModifier
{
    //Custom Parameters
    public float newFloatParameter;

    ссылка: 1
    public NewCustomModifier()
    {
        type = GetType().Name;
        tag = "ncm";
    }

    Ссылок: 2
    protected override void SerializeModifier()
    {
        //Pack custom parameters to array
        object[] outputData = new object[]
        {
            newFloatParameter,              //Index 0
            duration,                       //Index 1. Contains in ContinuousModifier class
            tag                             //Index 2. Contains in ContinuousModifier class
        };

        //Serialize custom parameters
        //serializedData field contains in the base class
        serializedData = Newtonsoft.Json.JsonConvert.SerializeObject(outputData);
    }
    Ссылок: 2
    protected override ModifierBase DeserializeModifier(string inputData)
    {
        //Deserialize custom parameters
        object[] data = Newtonsoft.Json.JsonConvert.DeserializeObject<object[]>(inputData);

        //Unpack custom parameters
        newFloatParameter = System.Convert.ToSingle(data[0]);     //Index 0
        duration = System.Convert.ToSingle(data[1]);              //Index 1
        tag = data[2].ToString();                                 //Index 2

        return this;
    }
}
```

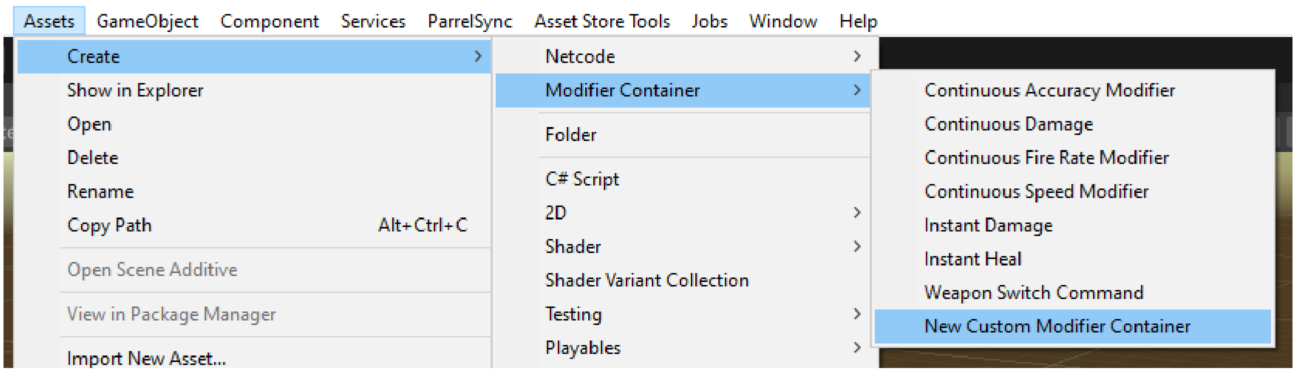
For more details, check other modifiers stored in **Scripts\Modifiers\Defined Modifiers** folder.

- Create new class **NewCustomModifierContainer** an inherit it from **ModifierContainerBase**
- Add **CreateAssetMenu**
- Override **GetConfig()** method

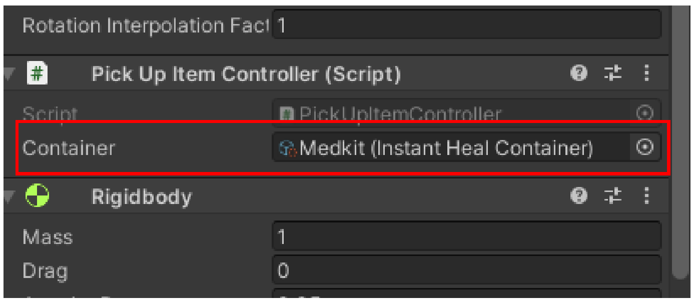


```
using UnityEngine;

[CreateAssetMenu(fileName = "NewCustomModifierContainer", menuName = "Modifier Container/New Custom Modifier Container")]
Unity Script | 0 references
public class NewCustomModifierContainer : ModifierContainerBase
{
    //Public visible scriptable object parameters
    public int newIntParameter;
    public float newFloatParameter;
    public string newStringParameter;

    2 references
    public override ModifierBase GetConfig() //Upcast to ModifierBase type
    {
        return new NewCustomModifier()      //Create and return modifier of our new type - NewCustomModifier
                                            //Don't confuse with scriptable object type  - NewCustomModifierContainer
        {
            //Take public scriptable object parameters and set them to modifier values
            newIntParameter = this.newIntParameter,
            newFloatParameter = this.newFloatParameter,
            newStringParameter = this.newStringParameter
        };
    }
}
```

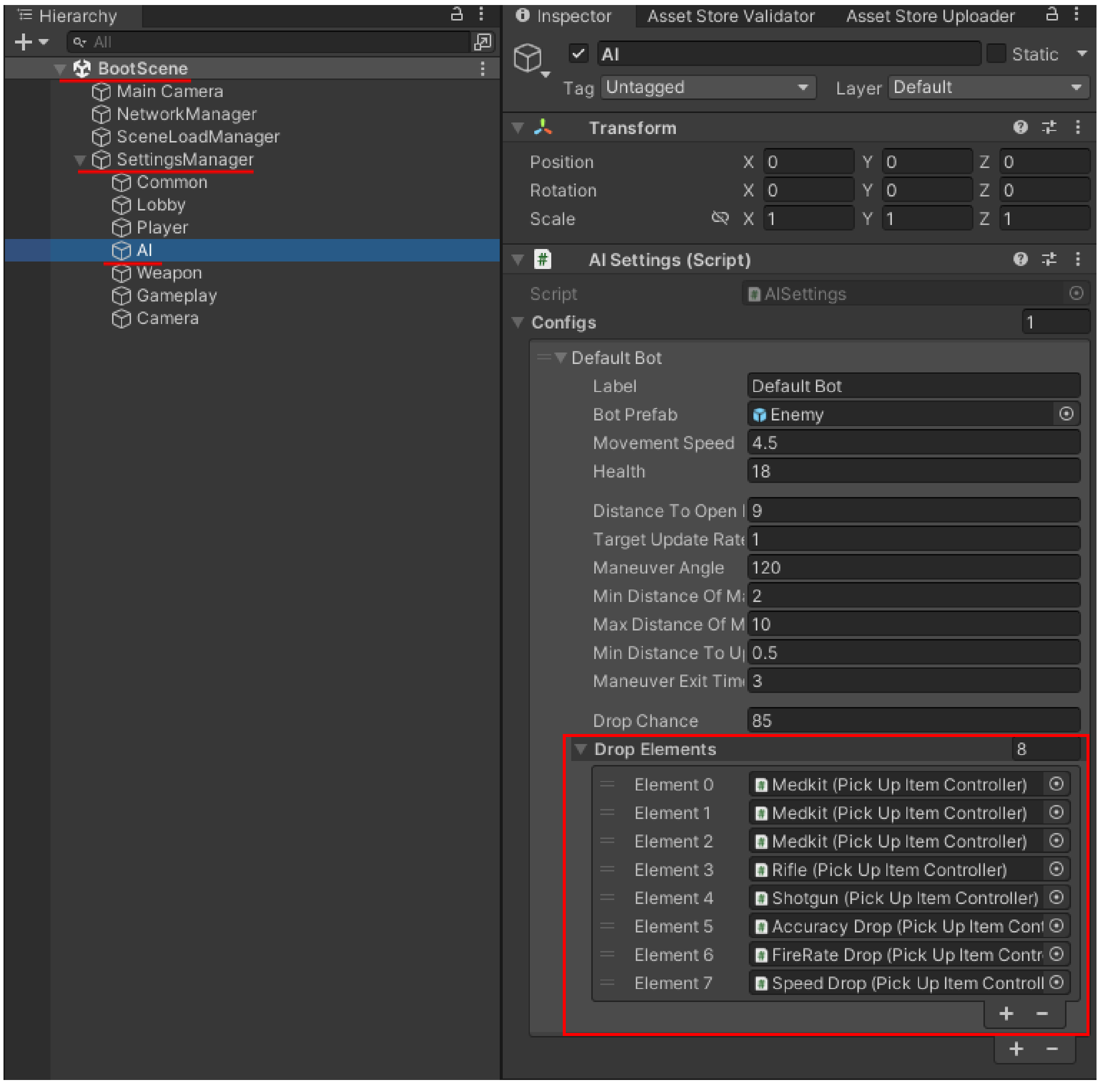
- Create container: Assets > Create > Modifier Container > **New Custom Modifier Container**



- Copy one of prefabs from **Prefabs\Drop** .
- Drag and drop just created **New Custom Modifier Container** file into **PickUpItemController** > **Container** field on a prefab.


Rotation Interpolation Fact 1
Pick Up Item Controller (Script)
Script PickUpItemController
Container Medkit (Instant Heal Container)
Rigidbody
Mass 1
Drag 0

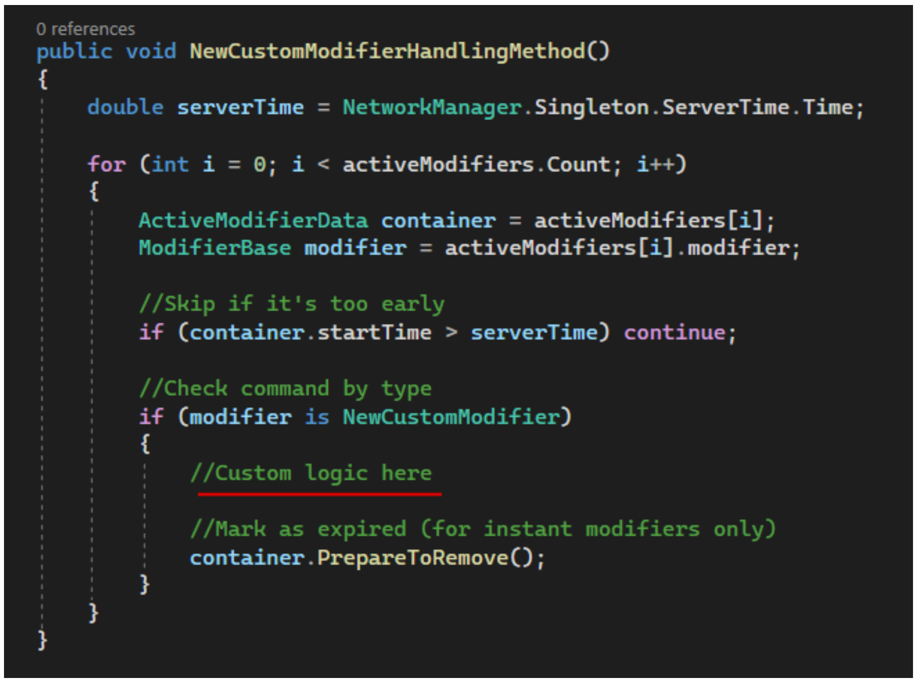
- Place prefab on scene or add it to AI settings drop elements list (recommended).



```
BootScene
  Main Camera
  NetworkManager
  SceneLoadManager
  SettingsManager
    Common
    Lobby
    Player
    AI
    Weapon
    Gameplay
    Camera
```

```
AI                                   Static
Tag  Untagged          Layer  Default

Transform
Position     X 0    Y 0    Z 0
Rotation     X 0    Y 0    Z 0
Scale        X 1    Y 1    Z 1

AI Settings (Script)
Script                    AISettings
Configs                   1
  Default Bot
    Label                 Default Bot
    Bot Prefab            Enemy
    Movement Speed        4.5
    Health                18

    Distance To Open I    9
    Target Update Rate    1
    Maneuver Angle        120
    Min Distance Of Ma    2
    Max Distance Of M     10
    Min Distance To U     0.5
    Maneuver Exit Tim     3

    Drop Chance           85
    Drop Elements         8
      Element 0   Medkit (Pick Up Item Controller)
      Element 1   Medkit (Pick Up Item Controller)
      Element 2   Medkit (Pick Up Item Controller)
      Element 3   Rifle (Pick Up Item Controller)
      Element 4   Shotgun (Pick Up Item Controller)
      Element 5   Accuracy Drop (Pick Up Item Cont
      Element 6   FireRate Drop (Pick Up Item Contr
      Element 7   Speed Drop (Pick Up Item Controll
```

- Add prefab to Network Prefabs List. **ScriptableObjects\NetworkPrefabsList.asset**
- Add **NewCustomModifier's** handling method to ModifiersControlSystem.



```
0 references
public void NewCustomModifierHandlingMethod()
{
    double serverTime = NetworkManager.Singleton.ServerTime.Time;

    for (int i = 0; i < activeModifiers.Count; i++)
    {
        ActiveModifierData container = activeModifiers[i];
        ModifierBase modifier = activeModifiers[i].modifier;

        //Skip if it's too early
        if (container.startTime > serverTime) continue;

        //Check command by type
        if (modifier is NewCustomModifier)
        {
            //Custom logic here

            //Mark as expired (for instant modifiers only)
            container.PrepareToRemove();
        }
    }
}
```

This method has to process modifier's logic. Also, it's supposed to be called <u>outside</u> ModifiersControlSystem, from component where it's required.

How it works: For every modifier type (or few) we have special handling methods stored in ModifiersControlSystem.
When we need to check something related to this modifier (status, value, etc), we call this method. Usually, it's called every frame.

Examples:
HandleHealthModifiers method calls from HealthController component, to process received heal and damage commands.
CalculateFireRateMultiplier method calls from Weapon component, to check its fire rate according to acquired power-ups.
HandleWeaponSwitchCommands method calls from WeaponControlSystem, to process changing of weapon.

Inside the method, we check if we have required modifier at all, if we do - we run custom logic and return the result. If we don't - return default value (or nothing).

Character can get modifiers from bullet hit or from picking up drop elements.
ModifiersControlSystem receives every modifier from CommandReceiver component.
Check ModifiersControlSystem and CommandReceiver for more details.

Modifiers could:
- take arguments and return updated data, like **HandleHealthModifiers** method.
- return multipliers, like **CalculateSpeedMultiplier, CalculateFireRateMultiplier** or **CalculateAccuracyMultiplier.**
- do nothing and work as flag: ContainsModifier<**NewCustomModifier**>()
- work with callback delegates, like **HandleWeaponSwitchCommands.**
- do any other custom logic
So, long story short: modifier is a command object. :)

Note: ModifiersControlSystem is a component. Every character has its own ModifiersControlSystem with its own active modifiers.
However it could be used anywhere else, outside character logic.

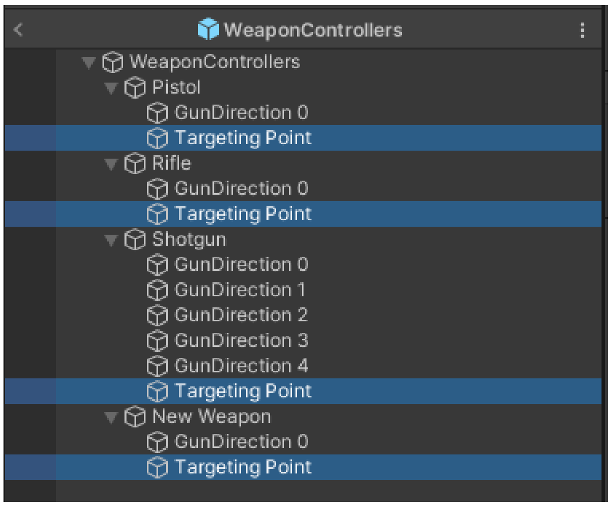
Lobby

Class name	Description
LobbyManager	Singleton. Contains methods to create and join lobby, including private rooms with password. Contains methods to host new game, when lobby is full and ready to start game session.
LobbyDataControl	Part of LobbyManager. Private object. Contains all the necessary properties, events and methods required by lobby to work with data (players, IDs, etc).
LobbyGameHostingControl	Part of LobbyManager. Private object. Contains methods to host or join hosted game using Relay service. Receives join/host commands from LobbyActivityControl. Also contains available regions and its updates.
LobbyActivityControl	Part of LobbyManager. Private object. Contains logic to update lobby timeouts. Required by lobby services, to know, lobby is still alive. Contains logic to check player statuses, before game start. Requires to make sure no one was disconnected (in unhandled way) while waiting, otherwise lobby updates its status to waiting for players.

Game

Class name	Description
GameManager	Singleton. Contains methods and events to control game status. Also contains list of user scores.
NetworkObjectsControl	Part of GameManager. Public object. Stores lists of all characters on scene: players, bots, service objects. Used to have easy access to any type of characters.
NetworkObjectsSpawner	Part of GameManager. Public object. Used to spawn/respawn characters and bots.

Character

Class name	Description
PlayerBehaviour	Playable character logic: movement, rotation, fire command.
AIBehaviour	Bot character logic: navigation, movement, rotation, looking for targets, battle maneuvers, fire command.
BasicNetworkTransform	Simplified logic of NetworkTransform. Easy to setup. No surprises. Contains interpolation and extrapolation.
CharacterIdentityControl	Contains identity markers: isPlayer, isBot. Overrides: IsLocalPlayer, IsOwner, OwnerClientId form NetworkBehaviour. Also contains custom character spawn parameters.
RigidbodyCharacterController	Using Unity rigidbody, instead of CharacterController.
CharacterAnimationController	Controls character animations, animation layers and skeleton to aim in the correct direction. For correction of aiming direction, every character contains Targeting Point transform, parrented to WeaponControllers:  WeaponControllers WeaponControllers Pistol GunDirection 0 Targeting Point Rifle GunDirection 0 Targeting Point Shotgun GunDirection 0 GunDirection 1 GunDirection 2 GunDirection 3 GunDirection 4 Targeting Point New Weapon GunDirection 0 Targeting Point It makes it possible to adjust aiming offset for every weapon.
CharacterColorController	Players start color setup.
CharacterEffectsController	Currently contains character death scenario (turn of colliders, move under ground etc.) Could be extended to use for appearance effect.
CharacterUIController	Controls HUD if it's a local player or status bar if it's bot or other player.
DropController	Controls what character could drop on death event.
HealthController	Controls current vitality status. Also runs death event.
TargetMarkerController	Controls marker that shows when someone puts a gun on character.
CommandReceiver	Receives modifiers/commands.

ModifiersControlSystem	Contains currently active power ups (speed, fire rate, accuracy) and commands, waiting for processing (damage, switch weapon). More detailed information is provided in the source code.
WeaponControlSystem	Main weapon control class. Contains all the weapons and controls to manage them. Every weapon has a Weapon component on it. Weapons spawn bullet game objects with Bullet components on them. More detailed information is provided in the source code.

Other

Class name	Description
ServiceUserController	Session user object. For service usage only. Spawns as separated game object.
PickUpItemController	Handles picking up and destroying weapons and power up boxes.
PlayerDataKeeper	Static class. Loads/Saves local data. Works with PlayerPrefs.
SceneLoadManager	Singleton. Works with loading target scenes through LoadingScene. Contains two ways of loading scenes: regular and network oriented. More detailed information is provided in the source code.
GameCameraController	Game camera, targeted on local player.
BoneAngleAdjuster	Small tool to adjust character bone rotation in runtime.