

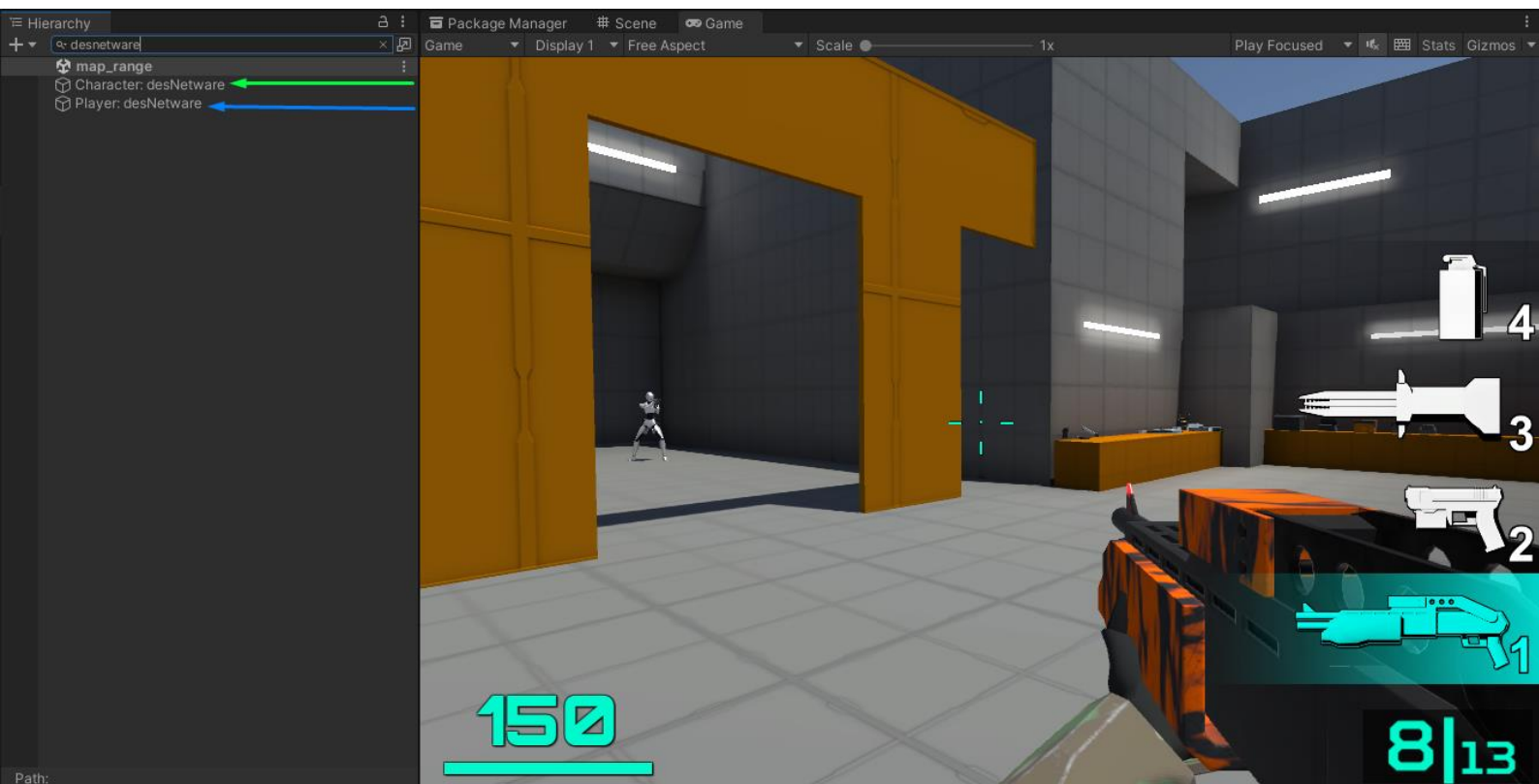


MultiFPS systems documentation

Contents

1.Players management.....	2
2. Player prefab	3
3. Item system	4
4. Gamemodes	5
5. Networking.....	6
6. How to add your custom map.....	7
Contact:	11

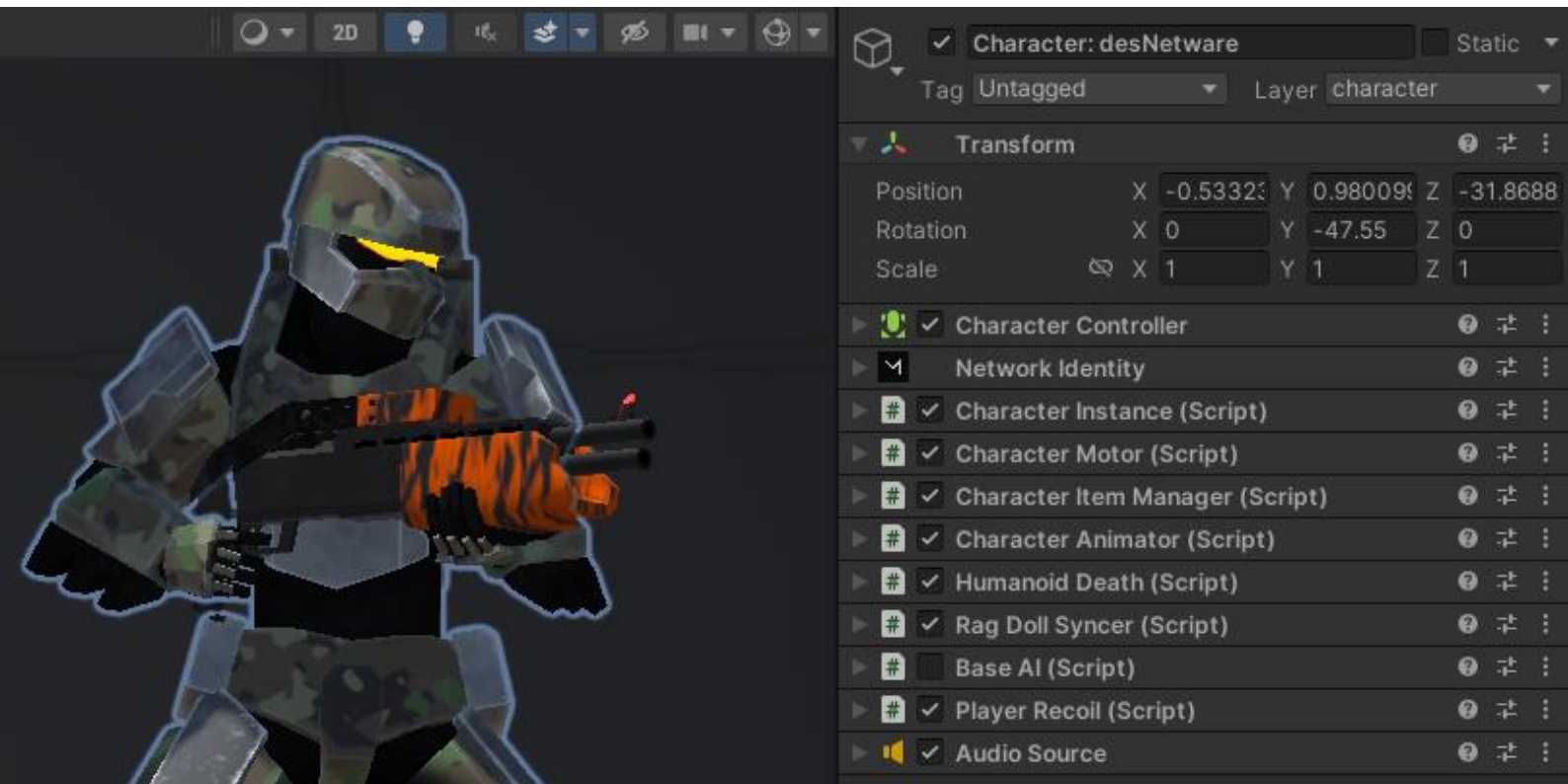
1.Players management



Each connected player (and bot) is represented by empty object with component `PlayerInstance`. This object is named "Player" with player nickname after that. From this objects players can communicate by chat and send spawn requests to the server, which can be accepted or not depending on game state and gamemode. This object exists as long as player is present in the game.

Then, when this object already exists, server will spawn **character** for given **player**, if certain conditions for given gamemode and gamestate are met. For example, if we play Team Deathmatch, spawn player after he chooses his team, or respawn player if he died and waited a couple seconds. Or don't let them spawn, and make them wait for next round to respawn, like it is in gamemodes like TeamEliminations or Defuse.

2. Player prefab



Components list with explanation:

CharacterInstance Responsible for basic character management. Like setting perspective from which character will be rendered, sending player input to server and clients, lerping position and more. Everything that character is controlled by players needs to have. This class also inherits **Health** class, what makes character able to be damaged and killed.

Health Responsible for receiving damage, and synchronizing it on server. Includes callbacks for being damaged and killed, specific for clients and server

CharacterMotor Responsible for character movement. Bots are also using this for their movement.

CharacterItemManager Responsible for managing player inventory, equipping items, putting them down, dropping them and picking new ones.

CharacterAnimator Based on player inputs stored in **CharacterInstance** component, animates character accordingly, thanks to that we don't need to use Mirror's CharacterAnimator, which saves us network bandwidth.

HumanoidAnimator Just spawns ragdoll on death

RagdollSyncer Responsible for sending to clients ragdoll information that is calculated on server so dead body lands for every client the same way.

`BaseAI` Enabled only for bots. This script is bot mind. It decides what to do and sends input to `CharacterInstance`. Go there, look there, shoot there, crouch in random moments. After sending input to `CharacterInstance` everything is handled as if it was normal player, no exceptions.

3. Item system



MultiFPS for both FPP and TPP perspective uses same item prefabs, so each item has only one prefab in game files. Difference lays in animations animation sets.

Item system consists of two scripts: `Item` (and classes that inherits `Item`) and `CharacterItemManager`

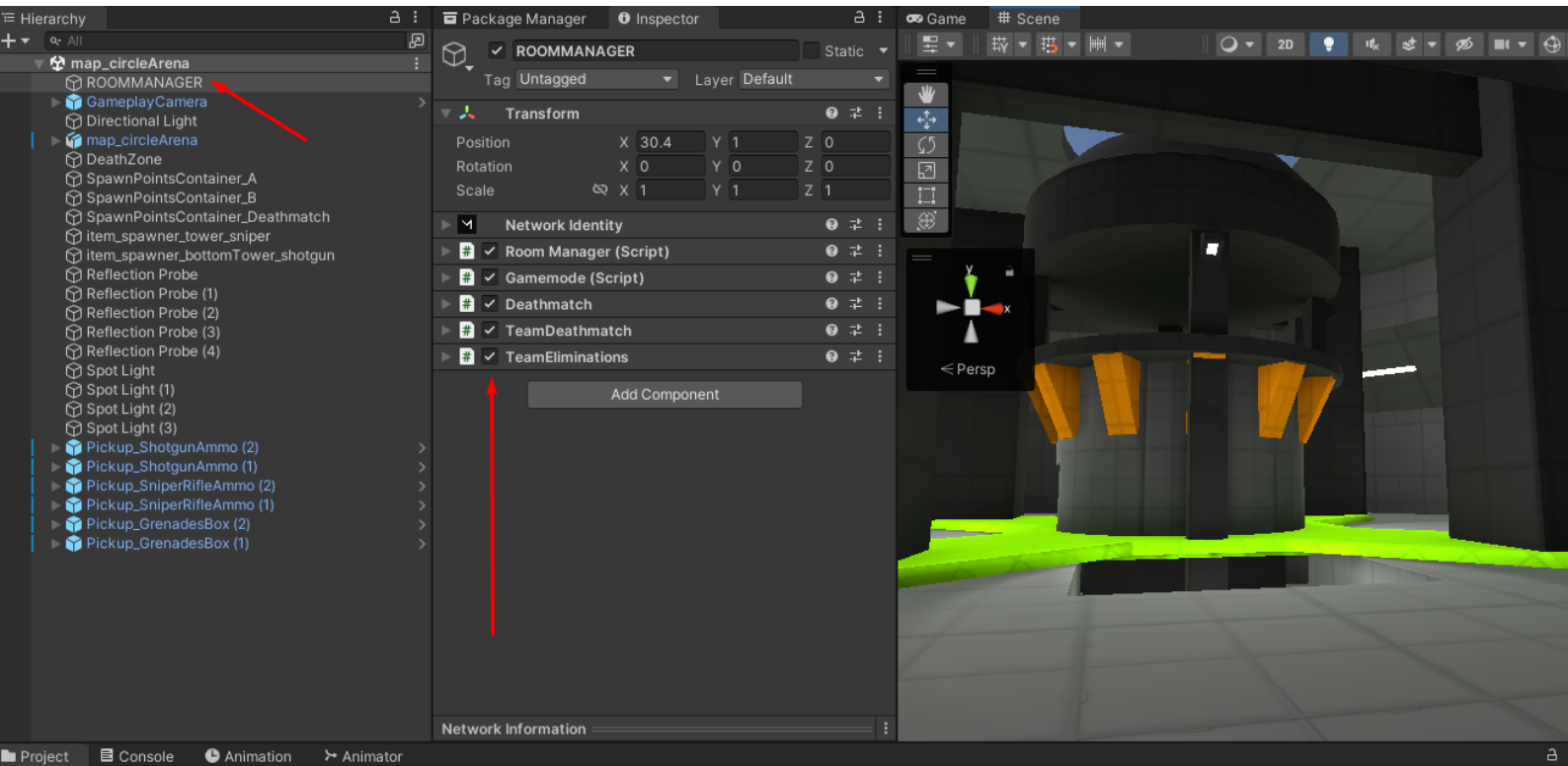
Every single item in the game that is usable for player inherits from `Item`. This class contains functions for picking, dropping, taking, and putting down item. It also synchronizes this functions for all players connected to the game.

`CharacterItemManager` manages player equipment. Thanks to this class we can take item, pickup it, drop, drop all of our items on death etc. It is also responsible for synchronization, so if we decide to change weapon, every client will see that.

4. Gamemodes

Gamemode system consists of those scripts: : `RoomManager` `Gamemode` and scripts that inherits from `Gamemode`: `Deathmatch` `TeamDeathmatch` `TeamEliminations` and `Defuse`

All of those scripts are present in map scene, on gameobject "ROOMMANAGER"

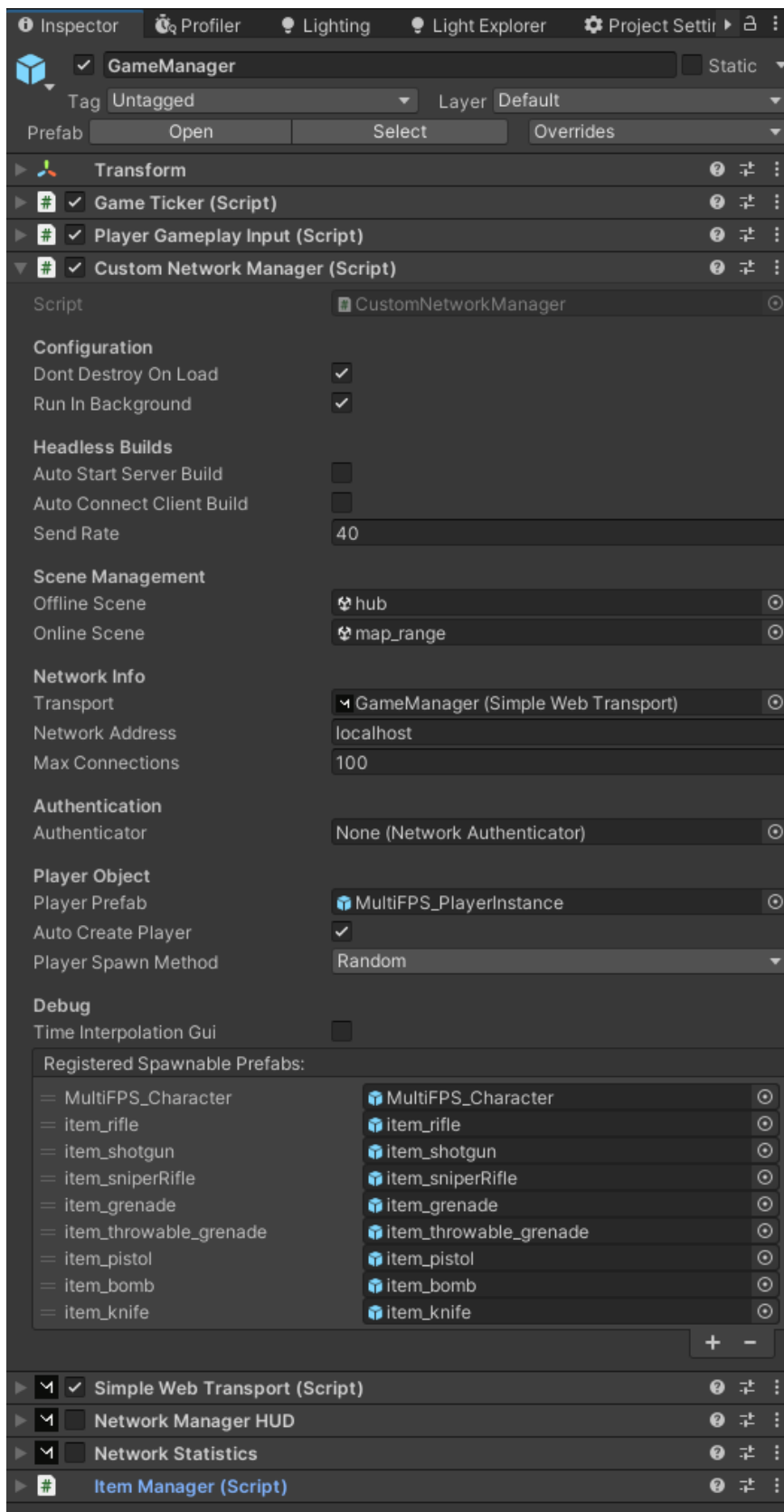


Each gamemode is placed in its own component. `RoomManager` component activates gamemode previously selected by user in hub scene (or loads it by properties if it is server headless instance). If map supports certain gamemodes, then their components have to be present and set up in this in object "ROOMMANAGER".

Base `Gamemode` class is also responsible for spawning bots if they are needed.

5. Networking

For networking MultiFPS uses Mirror, open-source network solution for server authoritative multiplayer games on Unity3D engine. For managing networking Mirror provides `NetworkManager` script, MultiFPS `CustomNetworkManager` inherits from that to add more functionality. This “GameManager” gameobject is located on “hub” scene and will be present also on gameplay maps



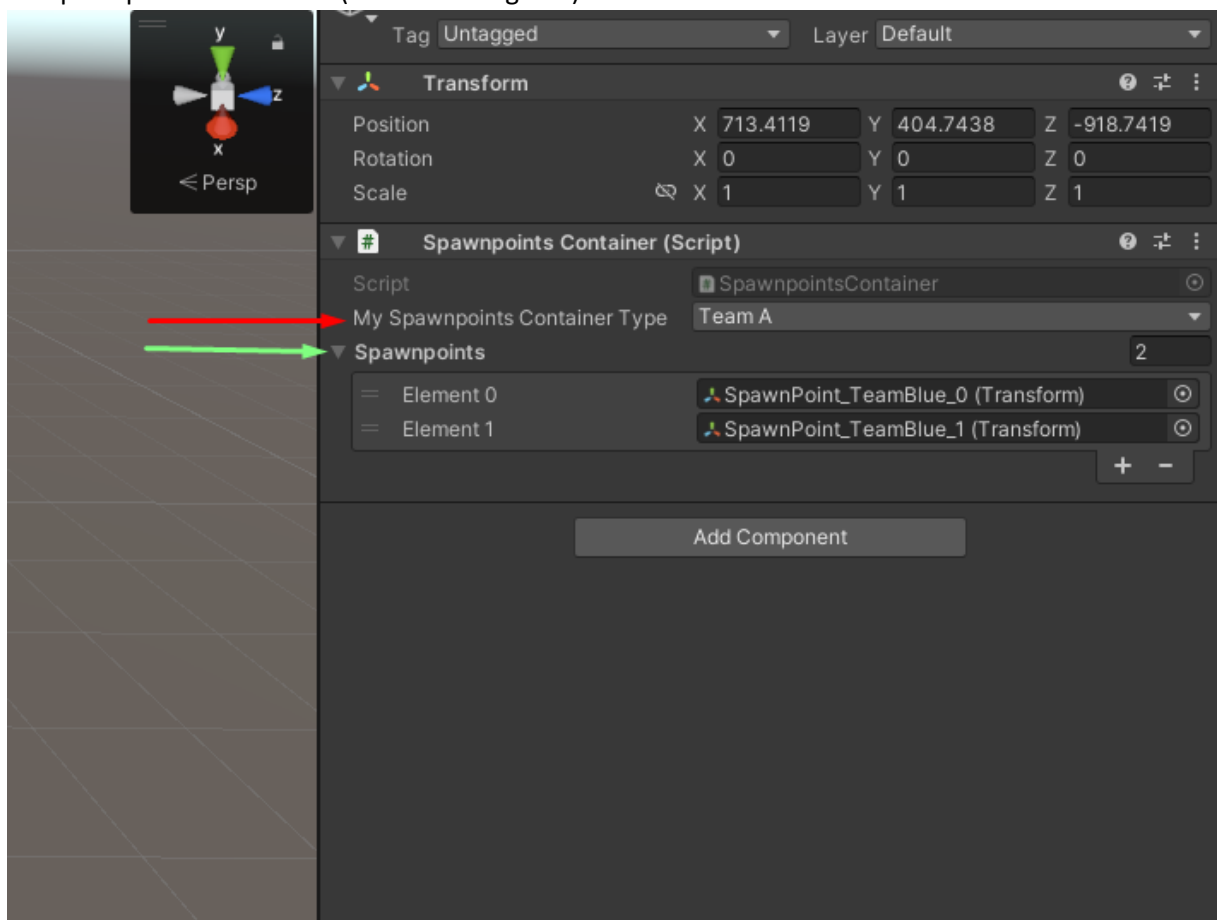
Offline scene is just scene that mirror will go to when client get disconnected, online scene is managed by

If You add any new object that have to be spawned in game for all participating players then you must add it to list “Registered Spawnable Prefabs”. First object in this list is always player prefab, so if you want to replace it, it must be also first prefab in this list.

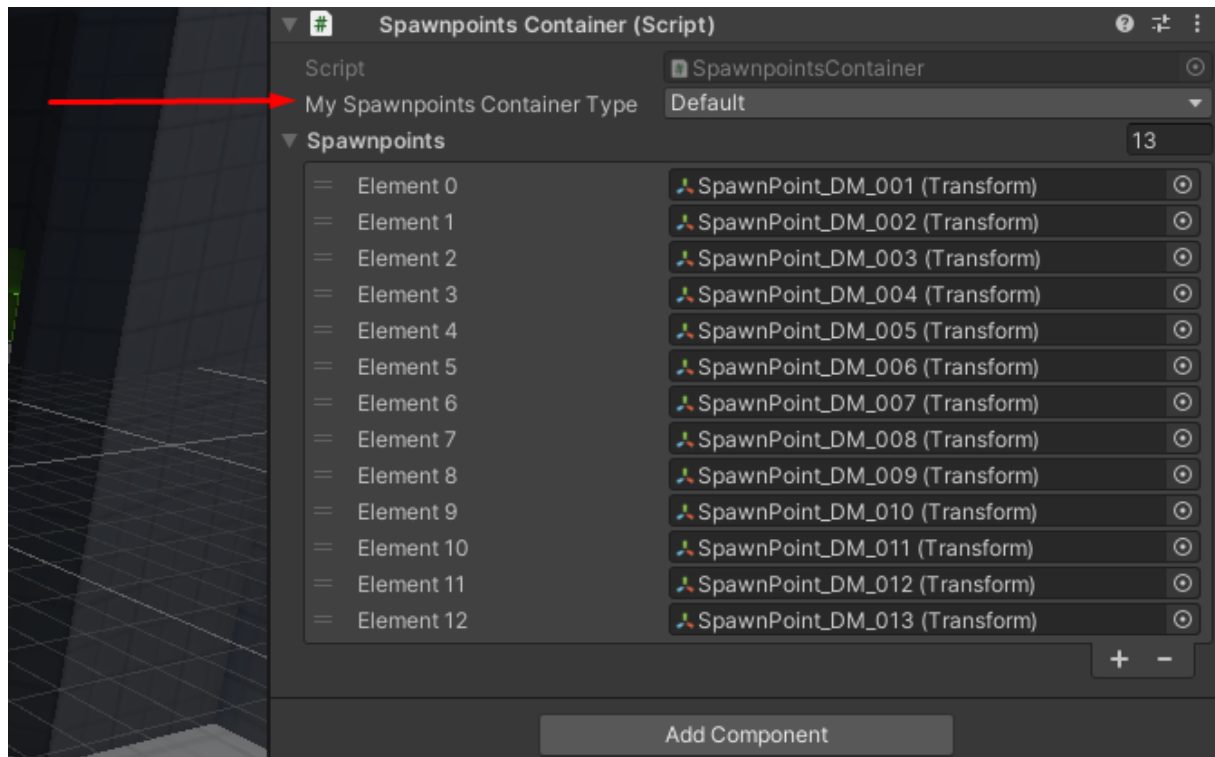
6. How to add your custom map

There is also youtube tutorial that shows this process [here](#)

1. Create unity scene
2. Place your map in this scene (don't forget to bake navigation map for it so bots will be able to navigate on your map, since they use Unity build-in navmesh system)
3. Create spawnpoints, as empty objects, their names does not matter.
4. MultiFPS categorizes spawnpoints for three **types**, and to group them apart we use another empty object with **SpawnpointsContainer** script attached to it. For gamemodes that use teams (for example Defuse, TeamDeathmatch, TeamEliminations) we will use group types TeamA and TeamB, to differentiate **spawnpoints** for each team. Example below shows setup for spawnpoints for team A (team blue in game)



For gamemodes that don't have teams (regular Deathmatch) we will use **type** "Default". This will tell deathmatch gamemode that these are the spawnpoints for every player present in match



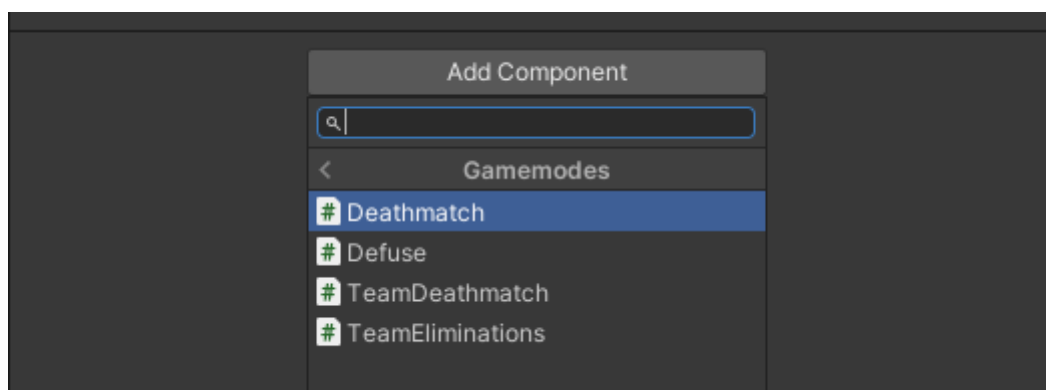
Add spawnpoints and their containers according to your needs, You can also have all three spawnpoints type present if Your map support multiple types of gamemodes at the same time.

Maps "BombBuilding" and "CircleArena" have all three types of spawnpoints, since they both support team based, and non-team based gamemodes.

5. Create empty gameobject. Name it "RoomManager", name is not crucial.

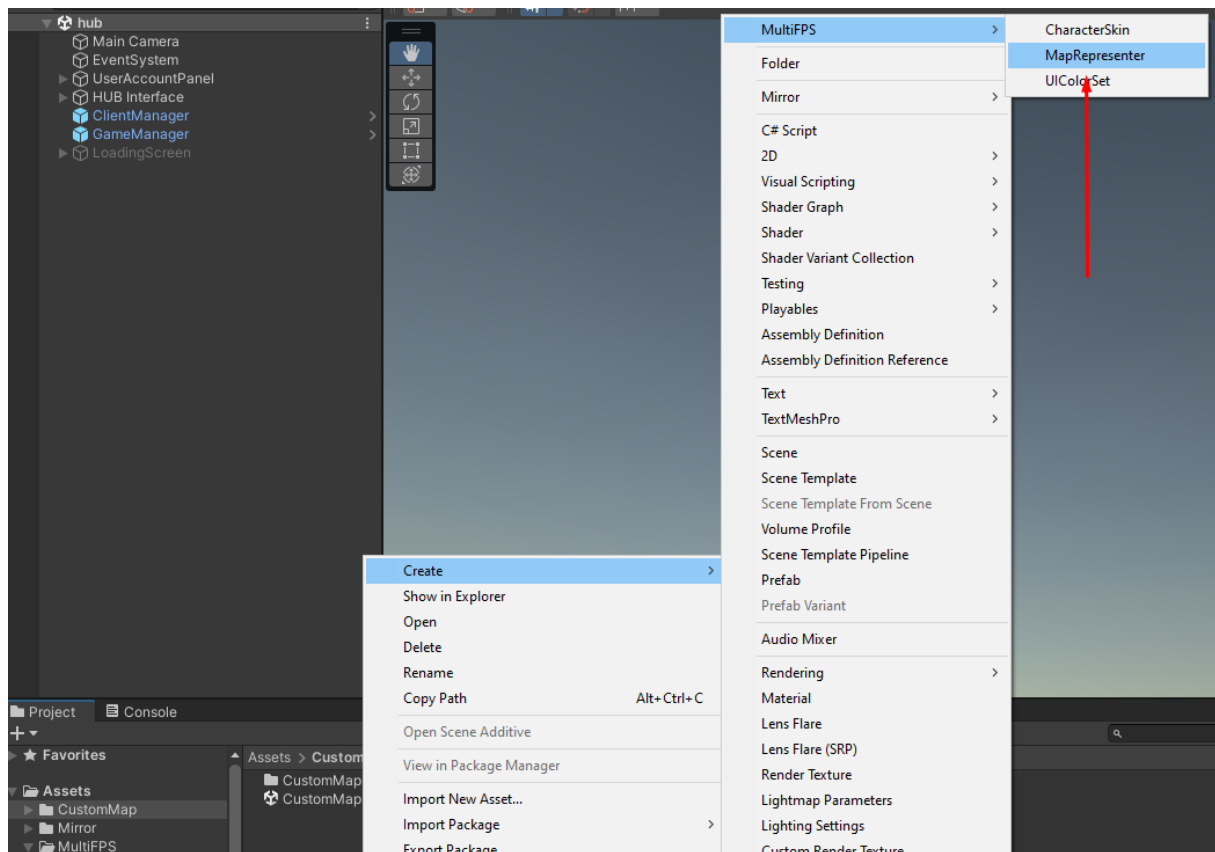
Add to this object `RoomManager` script and your desired gamemodes, they can be easily found by clicking:

Add Component -> MultiFPS -> Gamemodes



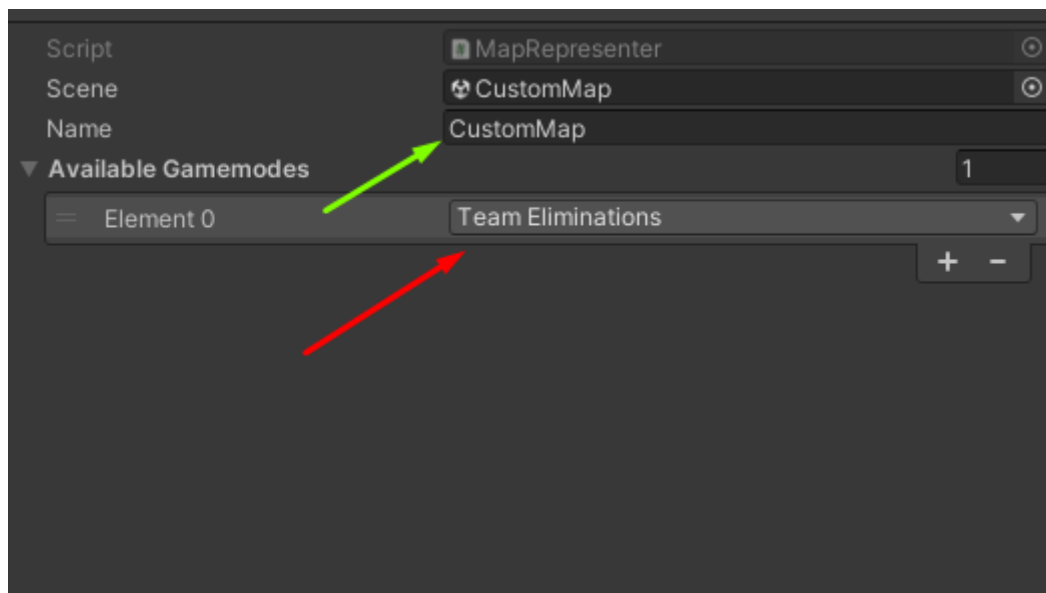
`RoomManager` will read properties set by player in main menu and apply them.

6. Map is ready, now we have to make it accessible for user to choose, to do that we need to create new file



Right click on project tab, and go select it Create->MultiFPS->MapRepresenter

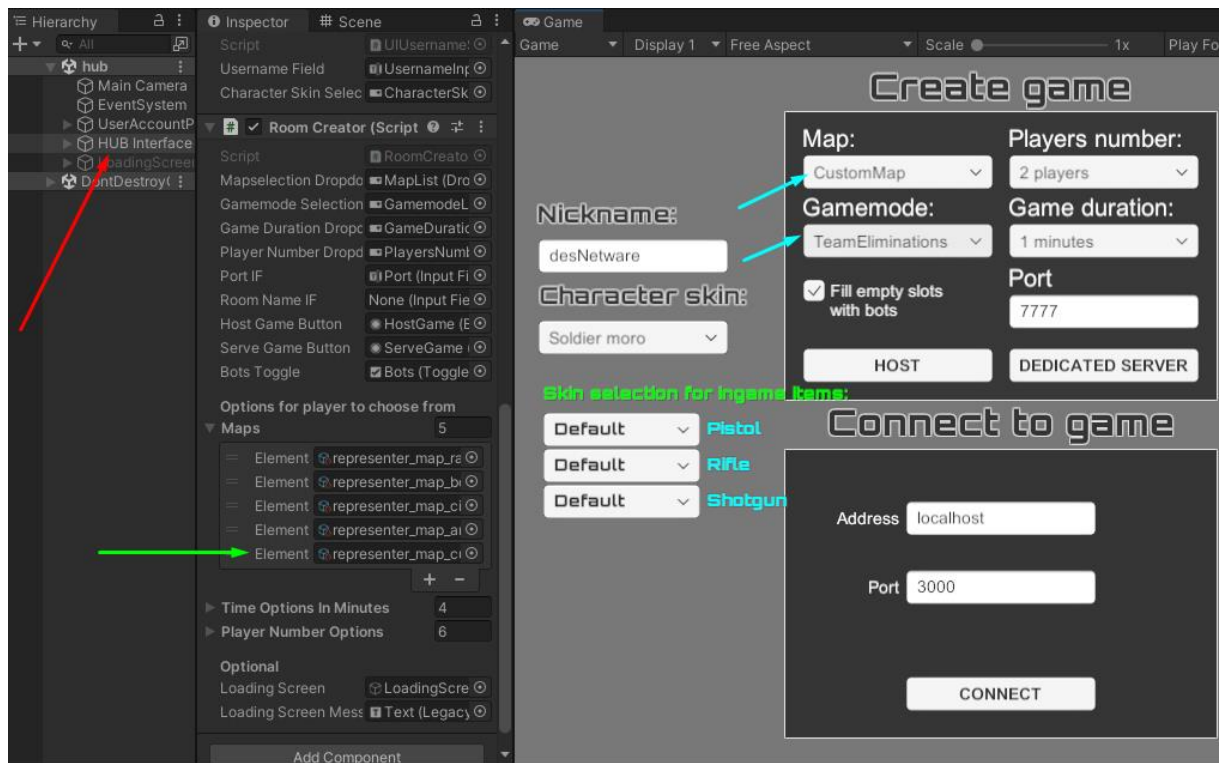
In the inspector of this object setup information about Your map



Name field is just for user, what he will see in map list, and available gamemodes list is what he will be able to choose for this map, so add here only gamemodes that we set in step 5.

7. Load hub scene (this is scene where player creates game)

Go to **HUB Interface** gameobject, and in script **RoomCreator** add map representer created in step 6 to **Maps** list. Now if we hit play we will be able to see our **new map** and **gamemodes** that we can choose for it. After selecting your map click **HOST** button to launch the game.



Thank You for using MultiFPS

Contact:

Email: desnetware@gmail.com

[Website](#)

[Youtube channel](#)

[AssetStore publisher page](#)