MultiFPS.Gameplay v1.0.1

Contents

Gameplay related components of MultiFPS

GameTicker

This component is instantiated on GameManager object. It is responsible for calling ticks in fixed intervals, based on sendrate(tickrate) set in CustomNetworkManager (Mirror's NetworkManager). Scripts that are listening to it can use it to send data on fixed intervals. Both server and client uses this component.

PlayerGameplayInput

This class is used for reading player gameplay input (moving, jumping, shooting, crouching, etc.) and applying it to one character that player is supposed to control. Also, if for example player dies, and takes control of bot in his team, then this class will change target, and instead of applying input to dead character, will now apply input to this bot.

PlayerInstance

This class is instantiated on object that will represent each player (and bot) in game session. It is used to send to server player inputs, like: chat messages, intention to join team, intention to change team. In this class we also count respawn cooldown, and manage spawning character for given player. It also calls PlayerGameplayInput to take control over spawned character if it belong to us.

Function	Description
`Server_SpawnCharacter()`	This method spawns character and registers it in game systems. If You don't want to gamemode to manage player spawns then use this method on Your own terms
`Server_ProcessSpawnRequest()`	It's launched when player send spawn request (by pressing space if dead) or when spawn cooldown went out. If current game state allows it then character will be spawned/respawned. Returns true if success

CustomNetworkBehaviour

This class extends Mirro'r NetworkBehaviour class. It contains one additional callback that is called on the server when new client joins during game. This callback can be overridden by class that inherits from it. It is very useful when for example new client joins the game, and we want him to see what equipment each player has, which weapon does they currently hold, how much health do they have, etc. So we can use this callback to note this fact that someone joined, and send him exclusively data about others that he need to know.

Function	Description
`OnNewPlayerConnected(NetworkConnection conn)`	Use argument conn to send TargetRPC call to new player

Health

Health stores entity health, max health, team, and name, so if we desire we can print it on killfeed or prompt when it's killed. For example, this cube have health component and hitbox component that relays to it:

Every entity in the game that is meant to be able to be damaged needs to have this component attached to itself or have other component attached that inherits from health. In case of player CharacterInstance component inherits health.

Function	Description
OnClientDamaged(int currentHealth, uint attackerID)	Override this method to program reaction on being damaged. It can be used for playing certain animations indicates something was hit, spawn ragdoll on death etc. Don't place any game logic related code here since this callback runs on clients only
Server_OnDamaged(int damage, byte hittedPartID, AttackType attackType, uint attackerID)	Equivalent of method above, but for server, place here code related to game logic, for example: if(CurrentHealth <= 0) DisableAI();
OnClientHealthAdded(int currentHealth, int addedHealth, uint healerID)	Launched on client from server when he was healed. Used for giving player feedback in form of little text about how much he was healed
ServerHeal (int healthToAdd, uint healerID)	This method car be run only on server, use it to heal entity by given health amount
Vector3 GetPositionToAttack()	Since bots can target and attack every entity in the game that has health, this Vector is needed to give them position that they should shoot toward. This position can be adjusted in the inspector. For example: if CenterPosition variable is set to [0,1,0] in the inspector, this means bots will shoot 1 meter above object origin that this health is attached to.

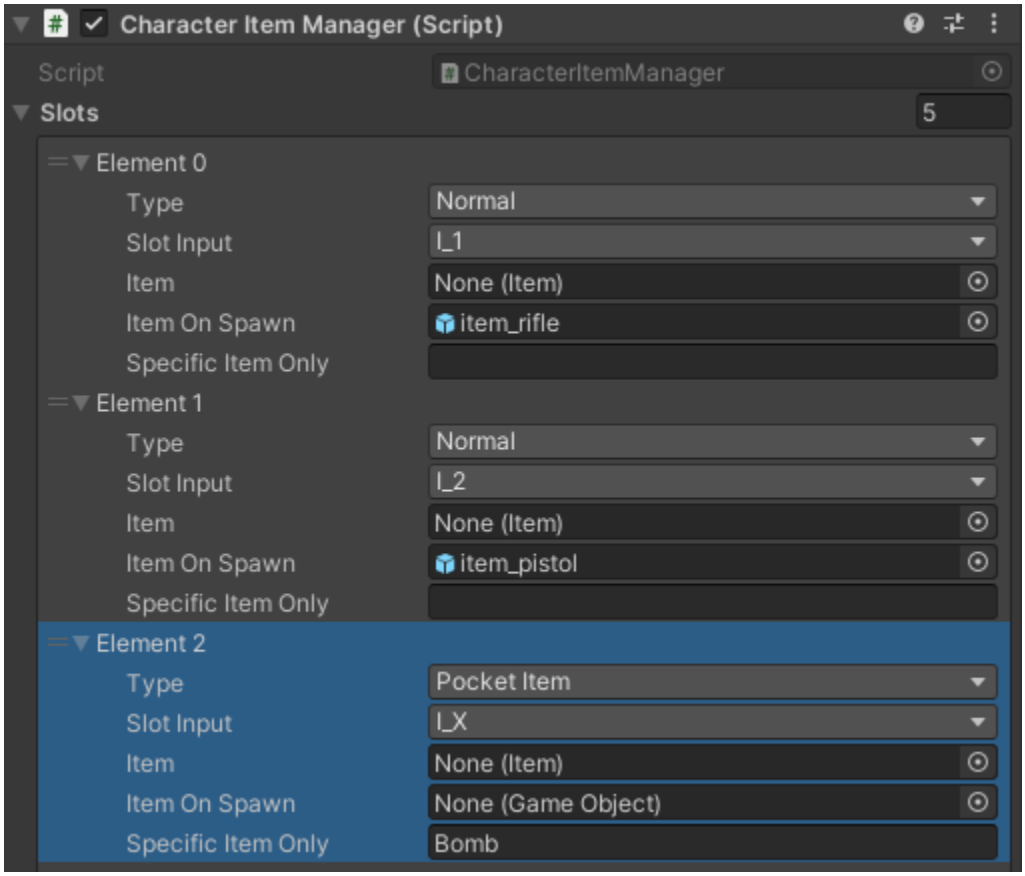
CharacterInstance

Character instance component is used for basic character management and contains universal information about character. Every game character, that is meant to be controlled by player need to have this, since PlayerGameplayInput class stores character to control by this component. CharacterInstance inherits from Health class to manage player events related to being damaged or killed internally.

It also synchronizes player/bot inputs, position and rotation of character.

CharacterItemManager

This component manages character equipment. In the inspector. You can specify number of slots for items and if they are meant to store every item, or only specific one.



Delegate	Description
`CharacterEvent_EquipmentChanged(int currentlyUsedSlot)`	Use this delegate void to perform some actions when player changes items. For example now it used to play taking item animation for player character model.

Item

Every item in the game that can be used by player/bot (which means every item that can be managed by CharacterItemManager class) must contain script that inherits this class.

Function	Description
`Take()`	Override this method to program things that have to happen to this item when it is selected by player, for example: start counting cooldown for shooting. Always include base of this method.
`PutDown()`	Override this method to program things that have to happen to this item when it is put down or dropped by player, for example: cancel reloading animation. Always include base of this method.
`Use()`	Override this method to program what item will do on left click of mouse, for example shooting. It already account for cooldown.
`SecondaryUse()`	Equivalent of above but for right mouse click
`AssignToCharacter()`	Override this method to program things that have to happen when item is picked up by player and assigned to his inventory. For example it is now used to disable physics of item when that happens
`Drop()`	Override this method to program things that have to happen when item is dropped by player. For example it is now used to enable physics and interaction triggers for item when that happens, so it can be picked up again by someone.

RoomManager

Every game map that is meant to be played on contain instance of this object, it is responsible for initializing proper gamemode on map. Gamemodes components must be attached to the same object with this component, as shown in multiple example maps that MultFPS come with.

Gamemode

Gamemode is responsible for managing the game state. Every gamemode in MultiFPS (Deathmatch, TeamDeathmatch, TeamEliminations, Defuse) inherits from it, since they share similar functionalities

Variables:

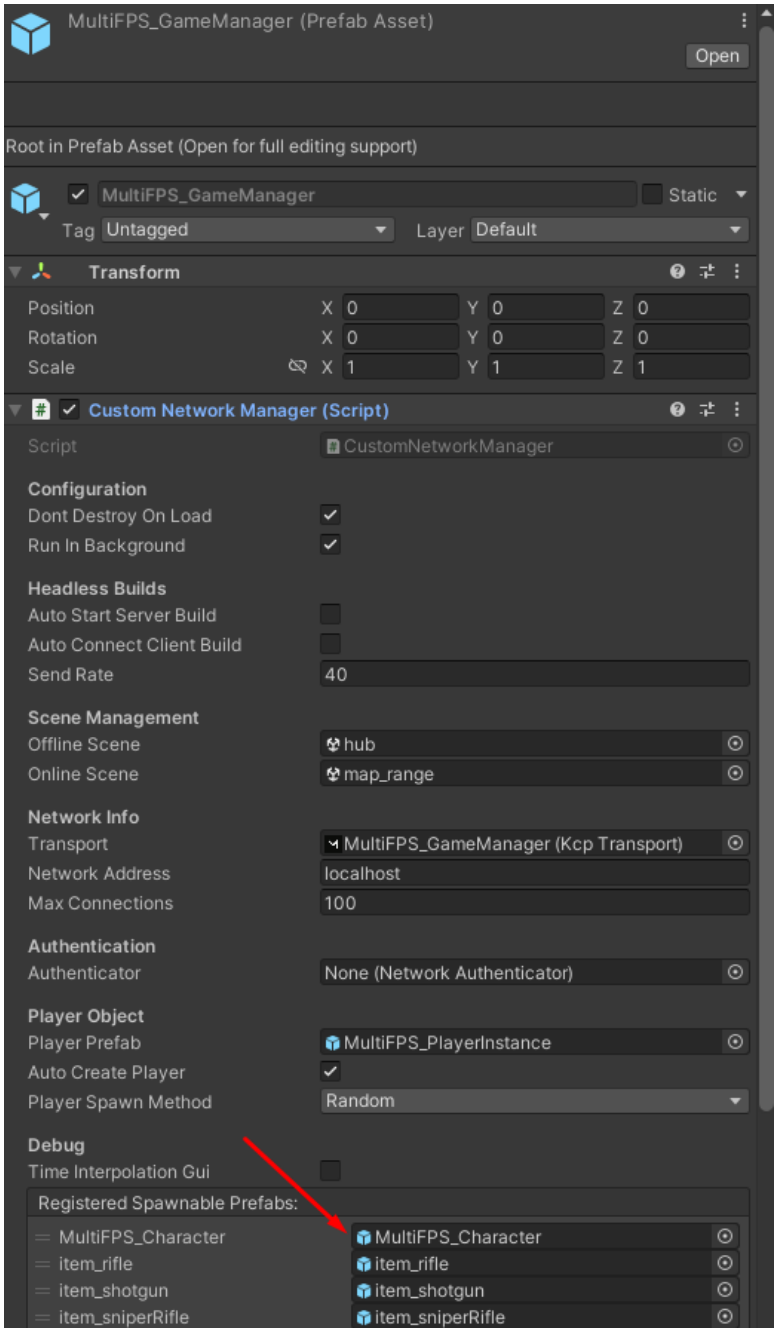
Variables	Description
`public bool FFA`	Free far all, thanks to this Boolean bots know if they should attack their teammates. It is only true for deathmatch, because in this gamemode technically everyone are in the same team, but everybody needs to fight each other.
`public bool FriendyFire`	True if players are meant to be able to damage their teammates
`public bool LetPlayersSpawnOnTheirOwn`	True only for Deathmatch and TeamDeathmatch, for Defuse and TeamElimination gamemodes we want to manage player spawns from gamemode component, they will be spawned when new round starts

Function	Description
`Server_OnPlayerInstanceAdded(PlayerInstance player)`	Override this method to determine what should happen when given player joins the game. For example DeathMatch gamemode immediately in this case adds player to default team and spawns him.
`Server_OnPlayerInstanceRemoved(PlayerInstance player)`	Override this method to determine what should happen if player disconnects from game. For now it is used in team based gamemodes to remove certain player from his team, if he was in one.
`PlayerSpawnCharacterRequest(PlayerInstance playerInstance)`	This method is called when player asks to be spawned. Use it to determine if he can, for example, in Defuse gamemode, we always deny that because gamemode will respawn everyone for new round, there is no way to be spawned earlier.
`AssignPlayerToTeam(PlayerInstance player, int teamToAssingTo)`	Call this method to assign player to team, this will call `OnPlayerAddedToTeam`
`OnPlayerAddedToTeam(PlayerInstance player, int team)`	Override this method to determine what should happen when player joins team. It can be used to check if both teams has enough players to start the game
`OnPlayerRemovedFromTeam(PlayerInstance player, int team)`	Override this method to determine what should happen when player left team (by changing teams or disconnecting), for example end the game when one team has no players left present in game session.
`SpawnBot(int team = 0)`	Use this method to spawn bot for given team. If both teams are full, then bot will exist as lonely empty player instance with no character gameobject

FAQ:

1: How Can I replace player prefab with my own?

- In MultiFPS_GameManager prefab that You can find by searching project files replace first registered spawnable prefab in CustomNetworkManager component with Your player prefab (red arrow on screenshot). Remember that if You want it to work with MultiFPS systems it must contain CharacterInstance component.



Thank You for using MultiFPS

Contact:

Email: desnetware@gmail.com

Discord

Website

Youtube channel

AssetStore publisher page