DOCUMENTATION

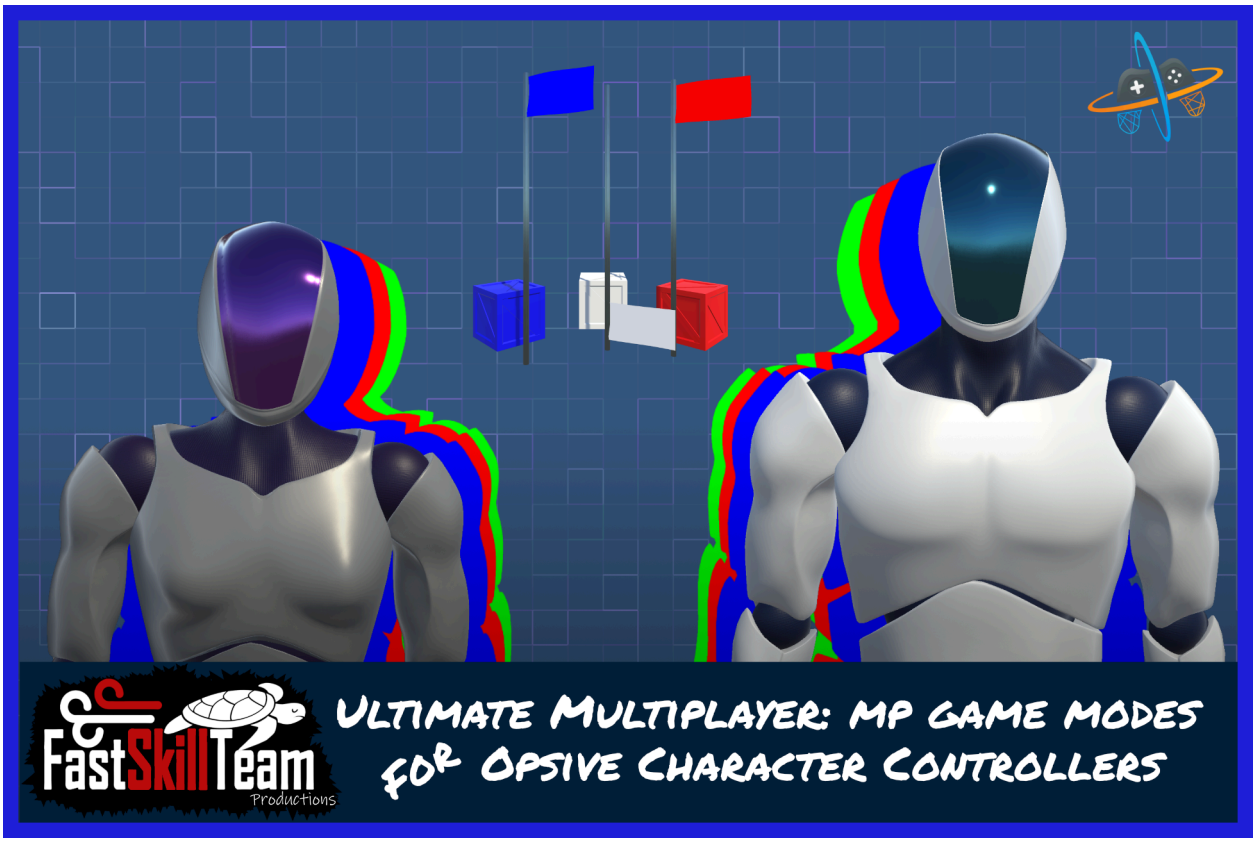

Table of Contents

First Notes

First of all I will say. It is expected that anyone building a multiplayer game, has some basic (at the least) knowledge of multiplayer logic. I cannot predict what your building or the style, and am not building a game for you, although I will provide examples of how to do anything you will require for the core of any AAA TPS/FPS battle. Should you find something more that is required for flexibility, let me know and I will certainly take consideration.

With the above already considered. Multiplayer is a vast base. I have provided in this asset a major head start in getting a multiplayer game realised for any TPS/FPS battle type scenario covering a majority of what you can find in any AAA game today without going too far with feature creep (or this asset would never see light of day!)..

Initial Setup

Ensure you have PUNv2, UCC and PUN-Addon installed. Other MP solutions will be accounted for as they are released.

Optionally install USC and the Helicopter Addon for the fullest of battle setups.

Add to tags > “Flag”, “CaptureZone” (TODO: should not be required?)

Install Ultimate Multi Player.

Character Setup

The Pun character has some components that are unique to UMP. A character that has been setup via the Opsive Pun addon can easily be converted by changing two components.

Replace Pun Character Health with MP Character Health

Replace Pun Respawner Monitor with MP Character Respawner Monitor

Game Scene Setup

Setup as per Opsive Pun Addon, Remove UCC SinglePlayerSpawnManager, then just add the relative and optional UI and objectives as follows.

MP Kill Feed

Displays attacker, weapon/vehicle/vehicleweapon used and victim. Simply add a damage source name to the list with an icon to match, this is a great example, but could be somewhat better if made custom for your game style, for example, you could fade out the kills rather than scroll away, this is pure dev preference and therefore the example is there for you.

Mini Map Handler

This is the base of the entire MiniMap system. Attached to UI this handles the placement of everything map related.

Attribute Monitor

Attribute Monitor with an Attribute Name of ArmDisarm. For rush objectives and rush ability.

Score Monitor

A simple score monitor at the most basic, as an example for you to create your own neat score monitor with extra data if you wish. Anything you require to study for your own is in here.

Objectives

Any scene for any mode can have objectives, more-so, any scene can have all objectives for all gametypes in the very same scene. Gamemodes do not require their “own” scene at all. This is useful to save on build space, and more importantly for us devs TIME!

Study each objective type to see just how this works and create your own cool game type to add to any game mode!

In short the objectives disable themselves if the game type is not the same as the objective type.

Start Scene Setup

The multiplayer start scene should contain alot of the core logic, as it is preferred in any AAA game to be able to sync data between scenes, and even have a specific scene load if a player is disconnected.

MP Connection

The heart of the connection to the cloud. This script manages the initial connection, joining lobbies, joining rooms, room leaving and disconnects. Also includes some logic to prevent any player having the exact same name in a room.

MP Chat Connection

Voice Chat and or Text Chat (optional). Main connection and router for global chat and game chat, with some under the hood logic you can use for your own backend to setup friend invites and notifications, along with joining a match together. The code is there, for simple pun authentication as an example and it will still require you to setup some other than default auth values. If you know your stuff in this area, I have provided a nice headstart for you, so check this out.

Game Modes

The game modes should be constructed in a manner that all modes are first level children of a root object that has a DontDestroyOnLoad component attached. Take the following for example, direct from demo scene.

GameModes (Dont Destroy On Load)

> Coop

> Deathmatch

> TeamDeathmatch

> MultiTeamDeathmatch

> Conquest

> Rush

> CTF

Each of these children objects need to contain the relative components, setup is very modular to contruct a gamemode.

Each and every mode requires MPPlayerSpawner and a PhotonView. A type of MPMaster is required also. This can be MPMaster, MPDMMaster, or your own extension of MPMaster. With MPDMMaster being an extension of MPMaster you can study this to see how easy it is to create your own custom extension of MPMaster.

The same goes for MPDamageCallbacks. MPDMDamageCallbacks is an extension of MPDamageCallbacks.

With these components you can create two differenet gamemodes already. Lets see how you can combine these to create different modes...

Coop > MPPlayerSpawner, MPMaster, MPDamageCallbacks. The most basic usage.

Deathmatch > MPPlayerSpawner, MPDMMaster, MPDMDamageCallbacks. Handles scoring etc pure death match without any team logic.

By adding MPDMTeamManager component, we can then take deathmatch further, and create two more game modes, utilising team logic and scoring.

TeamDeathmatch > As above, a simple copy of Deathmatch with one extra component for team logic. MPDMTeamManager.

MultiTeamDeathmatch > As above, a simple copy of TeamDeathmatch with two extra teams.

More game modes can be created by first setting up your scene with types of MPDMObjective. Again, this is designed to be easily extended. Examples are provided for Conquest, Rush and CTF. Study them to see how you can create your own custom objectives. If you create your own custom objectives, you will need to add them to the GameType enum in MPMaster so you select them, and then add to all relevant code path's for GameType usage.

TeamDeathMatchConquest > As you may of guessed, this is a copy of TeamDeathMatch, with the Game Type option set to Conquest. You may have already also guessed that you can duplicate TeamDeathMatchConquest, and add some more teams making it MultiTeamDeathmatchConquest.

TeamDeathMatchRush > You guessed it! Just the same as above but with the Game Type set to Rush.

TeamDeathMatchCTF > You guessed it! Just the same as above but with the Game Type set to CTF.

For in game text chat you can also add MPGameChat to any gamemode. To use game chat, you should also have setup the MPChatConnection and the MPChatInput as per Game Scene Setup.

At the time of writing, scoring has only been done for deathmatch types, so it only makes sense to use the score board for modes that incorporate scoring logic which apply score to MPPlayerStats or MPDMPlayerStats.

Score Board

An example of how to create a simple "smart" deathmatch scoreboard that can dynamically adapt to team count and colors, player counts and a unique color for local player, along with provided stats. You can use this scoreboard as it is, or use it for an example for you own. Check out MPDMMaster to see how the score board is activated.

DebugGUI can be used, to save any UI setup work and is nice and neat. Great for prototyping and even on console or PC. For mobile this should be avoided for performance reasons.

MP Menu

Wires things together as an example. Shows how to trigger which scene (map) to load, along with the game mode and selected character, with some extra examples of player naming (ensuring no two players have exact same name) and some other simple settings. Study this script in order to develop your own game flow.

Teams

MP Team Manager works to ensure team colours, spawning and score logic is accounted for.

Name

The name of the team.

Grouping

The grouping index used for the spawnpoints.

Objective Grouping

Unused just yet. Ignore. Will possibly be marked obsolete due to core system changes.

Color

The team color.

Character Prefab

The base prefab that will be spawned for the team in question. You could have prefabs with different colored materilas (as per demo for example), or different clothes eg. RU vs US.

Model Index

The ModelIndex should be left to -1 (no override) if players can select model index. If the ModelIndex is 0 or greater the character model will be forced to that model. Aside from that the Character Prefabs defined is what will be spawned. Therefore, you can have a specific prefab per team, and still have the player select the model they will play with, or you can force the model too.

MP Player Spawner

MP Player Spawner is the soul of getting a players character into the game, spawned with any components in the Components to Add list, which is where you should add the MPPlayerStats or MPDMPlayerStats component, depending on game mode. This is where modularity comes intosome seriously useful effect. We can add components and prefabs at spawntime, and per remote or local player, you can note in the demo only the remote players character requires a nametag prefab, so we add it at runtime when the player is spawned. MPPlayerSpawner will sort out what needs to be assigned to who on what machine relevant to which list the component or prefab is placed in.

Default Character

This is the character prefab that should be spawned in the case of no MPDMTeamManger or any case wher a team prefab or other is not specified. Consider it a backup even.

Spawn On Join

This determines WHEN the character prefab should be spawned. If this is true, the character prefab will be spawned upon the scene loading. This may not be desired if using a spawn panel. Examples are provided for use of this option, true or false.

Mode

This determines PLACEMENT of where the character should be initially spawned. Generally it is by spawnpoints for any game desgin, but the options are available to spawn at a fixed location with an offset applied per player. If a player is “Inactive” eg a non user disconnect or leave, then the player can rejoin, and be placed at the location where they are “lagged” out.

Spawn Location

Only applies when not using spawnpoints. A reference location to begin spawning characters.

Spawn Offset

Only applies when not using spawnpoints. How far the character should be spawned from the previously spawned player ID.

Spawn Point Grouping

This is a default setting, that will only be reverted to when there is no other spawn group management happening ie. when a MPDMTeamManager exits spawn grouping is determined by the MPTeamManger only when the relative input parameter is -1. Any time this is forced, the force group is used. This is how Objective/MiniMap spawning works, hence the local logic in each of them components to update and refresh required components at runtime.

Inactive Timeout

The duration a player can be considered “Inactive” when a non user disconnect has occured, keeping the room with a slot open for the disconnected player. This allows the disconnected player to rejoin the same room and game they were playing if they are reconnected in time.

Add Prefabs

With modurality in mind, the Add Prefabs section helps to define which prefabs to add to a local or remote prefab that is spawned.

Local

Any prefabs that should be attached to the LOCAL character prefab should be defined here.

NOTE: If you wish to utilise minimap, you can add the relevant prefab here, it has specific intialization for characters.

Remote

Any prefabs that should be attached to the REMOTE character prefab should be defined here.

NOTE: If you wish to utilise minimap or name tag, you can add the relevant prefab/s here, They have specific intialization for characters.

Add Components

With modurality in mind, the Add Components section helps to define which components to add to a local or remote prefab that is spawned.

At a minimum, in general either MPPlayerStats, or MPDMPlayerStats should be added here to both the remote and local list, depending on GameMode/GameType. You can find examples in the demo scene.

Local

Any components that should be attached to the LOCAL character prefab should be defined here.

NOTE: MPLocalPlayer is added by default and is not required in this list.

Remote

Any components that should be attached to the REMOTE character prefab should be defined here.

NOTE: MPRemotePlayer is added by default and is not required in this list.

MP Player Stats

Class Name: MPPlayerStats

Description: This component serves as a hub for all the common gameplay stats of a player in multiplayer. It does not hold any data itself, but collects it from various external components and exposes it through public 'Get', 'Set', and 'Erase' methods. MPPlayerStats is heavily relied upon by MPMaster. The basic stats include CharacterIndex, Team, Health, Shots, Position, and Rotation.

Note: By default, this component is automatically added to every player upon spawn. If you inherit the component, you must update the class name to be auto-added. This can be altered in the Inspector by going to your MPPlayerSpawner component -> Add Components -> Local & Remote.

Note: New stats can be added by inheriting this class. For an example of this, refer to MPDMPlayerStats and MPDMDamageCallbacks.

MP Player Stats does not actually hold data, but instead it keeps required stats in sync via components designated, this system prevents hackers from modifying any of the provided stats and does not rely on PUN player props. Please study this and its extension MP DM Player Stats, you will easily see how to include your own custom stats, that can even be unique per game mode due to the modular design. These stats can be fed to scoreboard and score monitor for example. MPMaster will ensure these stats are kept up to date upon a player joining or leaving or in the case of a master handover.

Shots

The stat named “Shots” is provided as an example of how you can bridge stats with modules. You can create your own extra modules.

The ShotsFiredTracker module should be added to the FireEffects module group.

For a great combination, you could create another stat “Hits”. Design a module for it for the ImpactEffects group and add it. You would then have both of the stats required to calculate “Accuracy”. I will leave this for you to do, it will be a great learning experience and building block to create some very in depth stats.

Health

Player health is kept in sync via master only. This stat relays the value.

Frags

The amount of kills accumulated during a round.

Deaths

The amount of deaths accumulated during a round.

Score

The amount of score accumulated during a round.

MP DM PlayerStats

Class Name: MPDMPlayerStats

Description: MPDMPlayerStats is an example of how to extend the base (MPPlayerStats) class with additional player stats: Deaths, Frags, and Score. This class provides getter and setter actions for these stats and is designed to be used in a DeathMatch game mode. The stats are manipulated in the example class 'MPDamageCallbacksDeathMatch'.

Note: MPDMPlayerStats extends the base MPPlayerStats class and adds additional player stats specific to a DeathMatch game mode. The stats include Deaths, Frags (kills), and Score. This class provides getter and setter methods for accessing and modifying these stats. By default, a 'MPPlayerStats' component is automatically added to every player by the MPPlayerSpawner on startup. If you want to use this derived MPDMPlayerStats component instead, you need to update the name of the data component in the Inspector of the MPPlayerSpawner component.

MP Master

Class Name: MPMaster

Description: This component is the king of the multiplayer game! It manages the overall game logic of the master client, including multiplayer game time cycles, assembly and broadcasting of full or partial game states with game phase, clock, and player stats. It also handles team allocation, initial spawn points for joining players, and broadcasts the local version of the simulation in case of a local master client handover.

Note: MPMaster is responsible for managing the overall game logic of the master client in a multiplayer game. It includes variables for controlling match start delay, player counts, round counts, round duration, interval duration, and game type. Customize these variables to configure the desired behavior for your multiplayer game.

Variables:

Variable: m_GameStartDelay

Type: float

Description: Initial delay for starting the match. Set to -1 for no delay. This can be used in combination with m_MinStartPlayers to delay the start of the match until a minimum number of players have joined.

Variable: m_MinStartPlayers

Type: int

Description: Minimum number of players required to start the match. This can be used in combination with m_GameStartDelay to delay the start of the match until a minimum number of players have joined.

Variable: m_MaxPlayers

Type: int

Description: Maximum number of players allowed in the match.

Variable: m_RoundCount

Type: int

Description: How many rounds should be played in one game session. The game will be over after this number of rounds are completed. Set to -1 for endless rounds.

Variable: m_RoundDuration

Type: float

Description: How long each round plays out as a maximum, in seconds. Note that objective completion can end a round before this time is up.

Variable: m_IntervalDuration

Type: float

Description: How long the interval is between each round, in seconds. During this interval, the game is paused, and the score can be displayed.

Variable: m_GameType

Type: GameType

Description: Defines the type of game mode for objective-based game modes. Use GameType.Standard for game modes with no objectives.

MP DM Master

Class Name: MPDMMaster

Description: An example of how to extend the base (MPMaster) class with a call to show the deathmatch scoreboard when the game pauses on end-of-round, and to restore it when the game resumes.

Note: Study the base class (MPMaster) to learn how the game state works.

Note: MPDMMaster extends the base MPMaster class and adds additional functionality specific to deathmatch gameplay. It includes a call to show the deathmatch scoreboard when the game pauses at the end of a round and restores it when the game resumes. This extension provides an example of how to customize the base class to incorporate deathmatch-specific features. Study the base class (MPMaster) to understand the overall game state and customize MPDMMaster accordingly for your deathmatch gameplay.

MP Character Health

With a similar setup to Character Health this script is designed so the master can keep the values in sync. The base MP Health also contains events for the MP Damage Callbacks namely “TransmitDamage”, “TransmitKIll” and “TransmitHeal”.

Characters are required to use this component to sync health online with Ultimate Multiplayer. Character Health should be replaced with this component.

MP Health

With a similar setup to Health this script is designed so the master can keep the values in sync. It also contains events for the MP Damage Callbacks namely “TransmitDamage”, “TransmitKIll” and “TransmitHeal”. While this is optional for objects (they can use opsive health) if you want full master control only over the damage to these objects, you must use MP Health.

MP Local Player

Available once the player has spawned a character on thier own machine, this script provides an easy way to access much of what is required and more.

This class represents the UCC local player in multiplayer. It extends MPPlayer with all functionality specific to the one-and-only local player on this machine. This includes listening to all sorts of events triggered by the EventHandler (such as round resets and chat functions). It also prevents input during certain multiplayer game phases.

MP Remote Player

Available once the player has spawned a character on all other machines, this script provides an easy way to access much of what is required and more.

This class represents a remote player in multiplayer. It extends MPPlayer with all the functionality specific to remote controlled player gameobjects in the scene. There can be an arbitrary number of remote player objects in a scene.

MP Character Respawner Monitor

MPCharacterRespawnerMonitor is designed to handle respawning via rpc calls sent to and picked up by MPPlayer.

MP Kill Feed

Class Name: MPKillFeed

Description: Displays kill data typically seen in multiplayer games. It is used in conjunction with MPKillInfoContainer, which is attached to a prefab. The MPKillFeed should be attached to a UI canvas in the GAME scene.

Variables:

Variable: m_MpKillFeedScrollRect

Type: ScrollRect

Description: Specifies the UI ScrollRect that will be controlled by the MPKillFeed. The ScrollRect is responsible for scrolling the kill feed content.

Note: This variable should be attached to the appropriate UI ScrollRect component in the scene.

Variable: m_MpKillInfoContent

Type: GameObject

Description: Specifies the UI GameObject that will serve as the parent for the MPKillInfoPrefabs that are spawned. The spawned kill info prefabs will be placed as children of this GameObject.

Note: This variable should be attached to the appropriate UI GameObject in the scene.

Variable: m_MpKillInfoPrefab

Type: GameObject

Description: Specifies the prefab GameObject that contains the MPKillInfoContainer component attached to it. This prefab is used to instantiate kill info elements in the kill feed.

Note: This variable should be assigned the appropriate prefab in the Unity Editor.

Variable: m_KillInfoLifeTime

Type: float

Description: Specifies the duration in seconds for which the kill feed will display the kill data. After this duration, the kill info elements will be removed from the feed.

Default Value: 2f

Variable: m_WeaponIconData

Type: List<WeaponIconData>

Description: Contains a list of WeaponIconData objects representing the data used for the kill feed. Each WeaponIconData object represents a weapon with its associated icon, scale, and rotation.

Note: This list should be populated with the necessary weapon icon data in the Unity Editor.

Nested Class: WeaponIconData

Description: Represents the data for a weapon icon in the kill feed.

Variables within WeaponIconData:

Variable: m_WeaponName

Type: string

Description: Specifies the name of the weapon associated with the icon.

Variable: m_Icon

Type: Sprite

Description: Specifies the icon to be displayed for the weapon.

Variable: m_Scale

Type: Vector3

Description: Specifies the scale of the icon. Allows customization of the size of the weapon icon.

Default Value: new Vector3(1f, 1f, 1f) (uniform scale of 1 in all dimensions)

Variable: m_Rotation

Type: Vector3

Description: Specifies the rotation of the icon. Allows customization of the rotation of the weapon icon.

Default Value: new Vector3(0f, 0f, 0f) (no rotation)

Properties within WeaponIconData:

Property: WeaponName

Type: string

Description: Retrieves the name of the weapon associated with the icon.

Property: Icon

Type: Sprite

Description: Retrieves the icon sprite associated with the weapon.

Property: Scale

Type: Vector3

Description: Retrieves the scale of the icon.

Property: Rotation

Type: Vector3

Description: Retrieves the rotation of the icon.

MP DM Objective Base

Class Name: MPDMObjectiveBase

Description: The base class for all objectives in the game modes. It provides functionality for applying bonus scores to players within the bonus zone trigger. The position and rotation of the root object can be synchronized.

Tips: For examples on how to extend this class, refer to MPDMObjectiveRush.cs, MPDMObjectiveConquest.cs, and MPDMObjectiveCTF.cs.

Variables:

Variable: m_SpawnGrouping

Type: int

Description: Specifies the grouping value for child spawnpoints when owned by the local team. UI buttons for spawning utilize this value. It determines the grouping behavior of spawnpoints.

Default Value: -1

Variable: m_DefendingTeamNumber

Type: int

Description: Specifies the team number for the defending team. A value of -1 indicates no team is assigned for defense.

Default Value: -1

Variable: m_TeamScoreAmount

Type: int

Description: Specifies the bonus score awarded to the team when the objective is completed successfully.

Default Value: 100

Variable: m_DefendScoreAmount

Type: int

Description: Specifies the bonus score awarded for defending the objective while inside the objective's bonus trigger.

Default Value: 10

Variable: m_AttackScoreAmount

Type: int

Description: Specifies the bonus score awarded for attacking the objective while inside the objective's bonus trigger.

Default Value: 10

Variable: m_CapturingScoreAmount

Type: int

Description: Specifies the score awarded for capturing progress towards the objective.

Default Value: 10

Variable: m_CaptureScoreAmount

Type: int

Description: Specifies the bonus score awarded for successfully capturing the objective.

Default Value: 50

Variable: m_SyncPosition

Type: bool

Description: Specifies whether the root object of the objective should synchronize its position. Set it to false if the object is not meant to move or if it's a child of another moving object, as it helps save data.

Default Value: false

Variable: m_SyncRotation

Type: bool

Description: Specifies whether the root object of the objective should synchronize its rotation. Set it to false if the object is not meant to rotate or if it's a child of another rotating object, as it helps save data.

Default Value: false

Variable: m_LayerMask

Type: LayerMask

Description: Specifies the layer mask used to check for bonus scoring within the objective. Only objects on the specified layers will contribute to bonus scoring.

Default Value: 1 << 0 (Default layer)

Variable: m_ObjectiveGameObject

Type: GameObject

Description: Specifies the GameObject representing the objective. If null, it defaults to the current object itself.

Default Value: null

Variable: m_ObjectiveRenderers

Type: Renderer[]

Description: An array of Renderers representing the objective's visual components. If null, it defaults to the first Renderer found among the children of the objective GameObject.

Default Value: null

Variable: m_MiniMapSceneObjects

Type: MiniMap.MiniMapSceneObject[]

Description: An array of MiniMapSceneObject instances representing the objective's data for the minimap. It provides visual information about the objective's position and status on the minimap.

Default Value: Empty array (length = 0)

MP DM Objective Conquest

Class Name: MPDMObjectiveConquest

Description: An example of how to extend MPDMObjectiveBase.cs for Conquest/ Zone Control style gameplay.

Note: MPDMObjectiveConquest extends MPDMObjectiveBase.cs and adds specific functionality for Conquest/Zone Control style gameplay. It includes variables related to flag positions, scoring intervals, capturing speed, and renderer color. Modify these variables as needed to suit the requirements of your gameplay.

Variables:

Variable: m_TargetRaised

Type: Transform

Description: The position the flag will be at when it is captured.

Variable: m_TargetFallen

Type: Transform

Description: The position the flag will be at when it is neutral.

Variable: m_ScoreUpdateInterval

Type: float

Description: The interval, in seconds, for bonus scoring while a team owns the flag. Determines how often the score is updated.

Variable: m_SpeedPerPlayer

Type: float

Description: The speed at which the flag will rise or fall, multiplied by the number of players capturing it. Adjusting this value allows customization of the capturing speed based on the number of players involved.

Variable: m_UseDefendingTeamColor

Type: bool

Description: If true, the objective renderer color will be set to the defending team's color as per MPTeamManager. If false, the renderer color will be blue for defenders and red for potential attackers. Leaving this at the default (false) provides clarity.

MP DM Objective Rush

Class Name: MPDMObjectiveRush

Description: An example of how to extend MPDMObjectiveBase.cs for Rush/Arm/Diffuse style gameplay.

Note: MPDMObjectiveRush extends MPDMObjectiveBase.cs and adds specific functionality for Rush/Arm/Diffuse style gameplay. It includes variables related to objective behavior, progression, interaction timings, scoring, audiovisual alerts, and event triggers. Customize these variables as needed to suit your specific Rush gameplay requirements.

Variables:

Variable: m_IsPartOfFirstSet

Type: bool

Description: Determines whether this objective should be active when a round starts. When the round resets, the "first set" of objectives will be activated, while the rest are deactivated.

Variable: m_Invincible

Type: bool

Description: Specifies whether this objective can be destroyed by damage. If set to true, the objective is invincible.

Variable: m_MaxHealth

Type: float

Description: Specifies the maximum health of this objective. It also represents the starting health of the objective.

Variable: m_AllowFriendlyDamage

Type: bool

Description: Determines whether the objective can be damaged by the defending team. If set to true, friendly fire can damage the objective.

Variable: m_DisableOnDeath

Type: bool

Description: Specifies whether to disable this gameObject when the objective is destroyed. If set to true, the objective gameObject will be disabled upon destruction.

Variable: m_UseDefendingTeamColor

Type: bool

Description: If true, the objective renderer color will be set to the defending team's color as per MPTeamManager. If false, the renderer color will be blue for defenders and red for potential attackers. Leaving this at the default (false) provides clarity.

Variable: m_NextObjectiveSet

Type: MPDMObjectiveRush[]

Description: Objectives that will be activated when this objective is destroyed. These objectives represent the next set of objectives in the progression.

Variable: m_DependentObjectives

Type: MPDMObjectiveRush[]

Description: Other objectives that need to be destroyed to progress. If null, progression will occur when this objective is destroyed.

Variable: m_SpawnOnDeath

Type: GameObject[]

Description: Any objects that should spawn when this objective is destroyed. These objects are instantiated upon the destruction of the objective.

Variable: m_DestroyOnDeath

Type: GameObject[]

Description: Any objects that should be destroyed when this objective is destroyed. These objects are destroyed upon the destruction of the objective.

Variable: m_SetActiveOnDeath

Type: GameObject[]

Description: Any objects that should be set active when this objective is destroyed. These objects are activated upon the destruction of the objective.

Variable: m_DeactivateOnDeath

Type: GameObject[]

Description: Any objects that should be set deactivated when this objective is destroyed. These objects are deactivated upon the destruction of the objective.

Variable: m_ArmMessage

Type: string

Description: The UI message that should be displayed when the objective can be armed.

Variable: m_DisarmMessage

Type: string

Description: The UI message that should be displayed when the objective can be disarmed.

Variable: m_ArmDuration

Type: float

Description: How long players must interact with the objective until it is armed. The objective needs to be interacted with continuously for this duration.

Variable: m_DisarmDuration

Type: float

Description: How long players must interact with the objective until it is disarmed. The objective needs to be interacted with continuously for this duration.

Variable: m_ArmedDuration

Type: float

Description: How long the objective can stay armed until it is destroyed. Once armed, the objective remains active for this duration.

Variable: m_ArmScoreAmount

Type: int

Description: When the objective is armed, the player who armed it will be awarded this much score.

Variable: m_DisarmScoreAmount

Type: int

Description: When the objective is disarmed, the player who disarmed it will be awarded this much score.

Variable: m_ArmedAlertAudioClip

Type: AudioClip

Description: Optional audio clip for alerting players when the objective is armed.

Variable: m_PulsingLight

Type: PulsingLight

Description: Optional pulsing light for alerting players when the objective is armed.

Variable: m_ExecuteOnDeathEvent

Type: bool

Description: Specifies whether to execute the on-death event when the objective is destroyed.

Variable: m_OnDamageEvent

Type: UnityFloatVector3Vector3GameObjectEvent

Description: Unity event invoked when taking damage.

Variable: m_OnHealEvent

Type: UnityFloatEvent

Description: Unity event invoked when healing.

Variable: m_OnDeathEvent

Type: UnityVector3Vector3GameObjectEvent

Description: Unity event invoked when the object dies.

MP DM Objective CTF

Class Name: MPDMObjectiveCTF

Description: An example of how to extend MPDMObjectiveBase.cs for Capture The Flag/Capture the Objective style gameplay.

Note: MPDMObjectiveCTF extends MPDMObjectiveBase.cs and adds specific functionality for Capture The Flag/Capture the Objective style gameplay. It includes variables related to captures required to win, flag behavior, renderer color, pickup collider, and bonus collider. Adjust these variables as needed to match the requirements of your Capture The Flag gameplay.

Variables:

Variable: m_CapturesToWin

Type: int

Description: Specifies the number of times this flag needs to be captured for a round to be won. Once this number is reached, the round is considered won.

Variable: m_RespawnOnCarrierDeath

Type: bool

Description: Determines whether the flag should return to its start position if the carrier is killed. If set to true, the flag will be respawned. If set to false, the flag will be dropped at the position where the carrier was killed.

Variable: m_UseDefendingTeamColor

Type: bool

Description: If true, the objective renderer color will be set to the defending team's color as per MPTeamManager. If false, the renderer color will be blue for defenders and red for potential attackers. Leaving this at the default (true) provides clarity for Capture The Flag with multiple flags.

Variable: m_PickupCollider

Type: Collider

Description: The collider that will trigger the pickup of the flag or objective. Players can interact with this collider to pick up the flag.

Variable: m_BonusCollider

Type: Collider

Description: The collider representing the bonus zone, if any. This collider defines an area where bonus scoring or special effects can be triggered when a player carrying the flag enters it.

MP Rush Ability

Class Name: MPRushAbility

Description: This ability allows the character to arm and disarm rush objectives and can display UI to show progress of arming or disarming by utilizing the Attribute Manager.

Note: MPRushAbility provides the functionality for the character to interact with rush objectives, specifically arming and disarming them. It also involves UI display to indicate the progress of arming or disarming. The m_CancelDistance variable sets the maximum distance at which the action is canceled if the character moves too far from the objective. Adjust this value as needed based on the desired gameplay mechanics.

Variables:

Variable: m_CancelDistance

Type: float

Description: If arming or disarming and the character moves further than this value away from the objective, the action will be cancelled. This distance determines the threshold at which the arming or disarming action is considered canceled if the character moves too far from the objective.

MP CTF Ability

Class Name: MPCTFAbility

Description: This ability allows the character to pickup and capture CTF flags/ objectives.

Note: MPCTFAbility provides the functionality for the character to interact with CTF flags or objectives, including picking them up and capturing them. The m_CarryPosition variable determines where the picked-up objective is visually attached to the character. The m_CaptureZoneTag and m_PickupTag variables define the tags of the trigger colliders used for capturing and picking up the objectives, respectively. Modify these variables as needed to match your CTF gameplay mechanics and object tagging conventions.

Variables:

Variable: m_CarryPosition

Type: Transform

Description: The carry position at which the objective will be placed when it is picked up. This determines the position where the CTF flag or objective will be visually attached to the character.

Variable: m_CaptureZoneTag

Type: string

Description: The tag of the game object with a trigger collider that represents the capture zone for the pickup. The objective is completed when the pickup (CTF flag or objective) is brought to this capture zone.

Variable: m_PickupTag

Type: string

Description: The tag of the game object with a trigger collider that represents the pickup (CTF flag or objective). This tag is used to identify the pickup object for interaction by the character.

Score Monitor

Class Name: ScoreMonitor

Description: The ScoreMonitor is responsible for updating the UI for any external object messages that send score updates. It can be used as a reference for displaying score-related information. An example of its usage can be found in MPDMPlayerStats.

Note: The m_ObjectVisibilityDuration and m_ObjectFadeSpeed variables control the visibility and fading behavior of the message after scoring points. Adjust these values as needed for desired timing and visual effects.

Variables:

Variable: m_Icon

Type: Image

Description: A reference to the object that will show the message icon.

Variable: m_Text

Type: Text

Description: A reference to the object that will show the message text.

Variable: m_ObjectVisibilityDuration

Type: float

Description: The length of time, in seconds, that the message should be visible for after scoring points.

Variable: m_ObjectFadeSpeed

Type: float

Description: The amount by which the message should fade after it should no longer be displayed. This value determines the speed of the fade effect.

Default Value: 0.05f

MP Score Board

Class Name: MPScoreBoard

Description: A classic multiplayer scoreboard that can be used for displaying player stats or for debugging purposes. The scoreboard supports both Unity GUI (UGUI) and Immediate Mode GUI (imGUI) implementations.

Note: MPScoreBoard provides a customizable scoreboard implementation for multiplayer games. It supports both UGUI and imGUI implementations. Use the UGUI implementation if you are using Unity's UI system, and use the imGUI implementation for immediate mode GUI rendering. Customize the provided prefabs, colors, fonts, and textures to match the visual style of your game.

UGUI Implementation:

Variables:

Variable: m_RefreshRate:

Type: float

Description: The delay between refreshing data while the scoreboard is open. Specifies how often the scoreboard updates its content.

Variable: m_GridLayoutGroup:

Type: GridLayoutGroup

Description: The required grid layout group. If null, it will attempt to get the component from the same GameObject.

Variable: m_TeamsParent:

Type: Transform

Description: The parent for the team containers. If null, it will use the transform of this component.

Variable: m_TeamContainerPrefab:

Type: GameObject

Description: The prefab that will be spawned to create a container for each team.

Variable: m_TitleRowPrefab:

Type: GameObject

Description: The prefab to act as a title row in the scoreboard.

Variable: m_TitleSpacerPrefab:

Type: GameObject

Description: The prefab to act as a spacer between the header and the title. Optional.

Variable: m_HeaderRowPrefab:

Type: GameObject

Description: The prefab to act as a header row in the scoreboard.

Variable: m_RowPrefab:

Type: GameObject

Description: The prefab to act as a player info row in the scoreboard.

Variable: m_CellPrefab:

Type: GameObject

Description: The prefab to act as a cell within each row in the scoreboard.

Variable: m_LabelPrefab:

Type: GameObject

Description: The prefab containing the text component for labels in the scoreboard.

Variable: m_RowColourA:

Type: Color

Description: The color scheme for alternate rows in the scoreboard, color A.

Variable: m_RowColourB:

Type: Color

Description: The color scheme for alternate rows in the scoreboard, color B.

imGUI Implementation:

Variables:

Variable: Font:

Type: Font

Description: The font used for the scoreboard. Note that this cannot be altered at runtime.

Variable: TextFontSize:

Type: int

Description: The font size for the text in the scoreboard.

Variable: CaptionFontSize:

Type: int

Description: The font size for captions in the scoreboard.

Variable: TeamNameFontSize:

Type: int

Description: The font size for team names in the scoreboard.

Variable: TeamScoreFontSize:

Type: int

Description: The font size for team scores in the scoreboard.

Variable: Background:

Type: Texture

Description: The background texture used in the scoreboard.

MP Damage Callbacks

Class Name: MPDamageCallbacks

Description: This is a base class manager that responds to damage, kills, and respawns on the current master client. It syncs the results to all other clients, ensuring that non-master clients can experience death and related gameplay events. MPDamageCallbacks serves as a foundation for handling damage-related mechanics and can be overridden to create more complex responses.

Note: MPDamageCallbacks is responsible for managing damage, kills, and respawns in a multiplayer game. It ensures that the effects of damage are synchronized across all clients. Override this class to create custom responses for handling damage-related events in your game.

Variables:

Variable: m_SyncPropHealth

Type: bool

Description: Set this to true to always keep non-player Health/IDamageTargets in perfect sync on all machines. This is not necessarily needed unless true pro play sync is required, as the master client will force-kill things that die in its scene anyway.

MP DM DamageCallbacks

Class Name: MPDMDamageCallbacks

Description: This is an example of how to extend the base class (MPDamageCallbacks) with additional callback logic for 'Damage' events. In this example, we refresh the 'Deaths', 'Frags', and 'Score' stats declared in MPDMPlayerStats every time a player dies. We also broadcast a new game state (containing these updated stats only) to reflect the changes on all machines.

Note: MPDMDamageCallbacks is an example of extending the base MPDamageCallbacks class to implement additional logic specific to the deathmatch game mode. It demonstrates how to handle 'Damage' events and update player stats accordingly. Study the base class to understand the 'TransmitKill' callback functionality. Customize MPDMDamageCallbacks based on your game's scoring and stat tracking requirements for the deathmatch game mode.

Variables:

Variable: m_ScorePerFrag

Type: int

Description: The score awarded per frag (kill) in the deathmatch game mode.

Variable: m_ScorePerDeath

Type: int

Description: The score deducted per death in the deathmatch game mode.

Variable: m_ScorePerTeamKill

Type: int

Description: The score deducted per team kill in the deathmatch game mode.

MP Connection

Class Name: MPConnection

Description: MPConnection is responsible for initiating and managing the connection to Photon Cloud. It regulates room creation, enforces a maximum player count per room, and handles the logon timeout. It also keeps the 'IsMultiplayer' and 'IsMaster' flags up-to-date, which are frequently relied upon by core classes.

Note: MPConnection is responsible for managing the connection to Photon Cloud in multiplayer games. Customize the variables to configure the connection behavior, debug logging, scene loading, and other connection-related settings according to your game's requirements.

Variables:

Variable: m_LogOnTimeOut

Type: float

Description: If a stage in the initial connection process stalls for more than this many seconds, the connection will be restarted.

Variable: m_MaxConnectionAttempts

Type: int

Description: After this many connection attempts, the script will abort and return to the main menu.

Variable: m_SceneToLoadOnDisconnect

Type: string

Description: The scene that will be loaded when the 'Disconnect' method is executed.

Variable: m_Debug

Type: bool

Description: Enable debug logging for the MPConnection component.

Variable: m_DebugToGameChat

Type: bool

Description: Enable debug logs to be displayed in the in-game chat.

Variable: m_DontDestroyOnLoad

Type: bool

Description: Determines whether the MPConnection component should be preserved across scene changes.

Variable: m_PingReportInterval

Type: float

Description: The interval at which ping reports are sent.

Variable: m_StartConnected

Type: bool

Description: Determines whether the connection should be started automatically on scene load or object creation.

Variable: m_LoadSceneOnJoinRoom

Type: bool

Description: Determines whether the scene should be loaded automatically when joining a room.

MP Chat Connection

Class Name: MPChatConnection

Description: MPChatConnection is the base class used for main chat functionality within the lobby and handles "under hood" operations throughout the game. It serves as a foundation for managing chat-related operations and facilitates friend interactions in multiplayer gameplay.

Note: MPChatConnection provides essential functionality for implementing chat features in multiplayer games. It supports main chat functionality within the lobby and handles various operations behind the scenes throughout the game. Customize and extend this base class to incorporate specific chat-related functionality and integrate friend interactions within your game.

Variables:

Variable: m_LogOnTimeOut

Type: float

Description: After this amount of time, if a connection attempt fails, the script will attempt to reconnect. If a stage in the initial connection process stalls for more than this many seconds, the connection will be restarted.

Variable: m_MaxConnectionAttempts

Type: int

Description: After this many connection attempts, the script will abort and return to the main menu. Set to 0 for unlimited connection attempts.

MP Game Chat

In game chat only. Registers to and relays events for chat functionality.

Custom Damage Processor

Class Name: CustomDamageProcessor

Description: CustomDamageProcessor is a custom damage processor for character weapons. It is used throughout all FastSkillteam assets for the Ultimate Character Controller.

Note: CustomDamageProcessor provides a customizable damage processing system for weapons. It allows you to block damage and/or force applied to specific objects by specifying their names in the Blocked Damage Target Names array. Adjust the variables according to your game's requirements for handling damage and blocking specific targets.

Assets > Create > FastSkillTeam > UltimateCharacterController > Damage Processors > Custom Damage Processor

Variables:

Variable: m_DebugBlocked

Type: bool

Description: Enable debug logging for blocked damage targets.

Variable: m_AllowDamageOnBlockedTargets

Type: bool

Description: Determines whether damage is allowed on blocked targets.

Variable: m_AllowForceOnBlockedTargets

Type: bool

Description: Determines whether force is allowed on blocked targets.

Variable: m_BlockedDamageTargetNames

Type: string[]

Description: An array of names for objects that should be blocked from receiving damage. Damage from weapons will be blocked if the object's name contains any of the specified Blocked Damage Target Names. Useful for blocking damage to specific objects such as tanks, helicopters, or rush objectives.

Mini Map Handler

Class Name: MiniMapHandler

Description: MiniMapHandler is a system that manages the registration and unregistration of minimap components. It functions online without the need to add any PhotonView components. This component serves as the heart of the system and handles the initialization and management of minimap UI objects.

Note: MiniMapHandler is a versatile component that allows you to manage and customize the behavior and appearance of a minimap in your game. Adjust the variables according to your specific requirements for minimap functionality, such as using baked maps, rendering live maps, and setting up UI and world anchor references.

Variables:

Variable: m_MiniMapGameObject

Type: GameObject

Description: The GameObject that is the parent of the minimap UI. If set to null, the MiniMapHandler component will default to using itself as the parent.

Variable: m_AutoInitialize

Type: bool

Description: Determines whether this component should be automatically initialized along with any registered UI objects.

Variable: m_IconPrefab

Type: GameObject

Description: The prefab to spawn for registered minimap objects as icons.

Variable: m_ShapeColliders

Type: Collider2D[]

Description: The colliders used for clamping icons on the minimap. Multiple colliders can be provided, allowing for custom-shaped maps.

Variable: m_ShapeColliderIndex

Type: int

Description: The index of the shape collider to use for clamping icons on the minimap.

Variable: m_Border

Type: Sprite

Description: The border image of the minimap.

Variable: m_UseBorderAsMapTexture

Type: bool

Description: Set this to true if you want to use the border image as the baked minimap texture.

Variable: m_MapMask

Type: Sprite

Description: The mask for the minimap display.

Variable: m_BorderOpacity

Type: float

Description: The opacity of the minimap border.

Variable: m_MapOpacity

Type: float

Description: The opacity of the minimap.

Variable: m_MapScale

Type: Vector3

Description: Scale factor for the graphics of the minimap.

Variable: m_OnlyScaleBorder

Type: bool

Description: If true, only the border of the minimap will be scaled.

Variable: m_HideBoardedIcons

Type: bool (optional, requires Ultimate Seating Controller)

Description: If true, any boarded character icons will be hidden.

Variable: m_UseBakedMap

Type: bool

Description: Set this to true if you will not use a render texture or render camera and want to use a baked map.

Variable: m_MoveMap

Type: bool

Description: Determines whether the map should move with the local player's character.

Variable: m_BakedMap

Type: Sprite

Description: The baked map image to use if you're not using a render texture or render camera.

Variable: m_UIAnchorReferenceA

Type: RectTransform

Description: The UI anchor reference A position on the UI map that is relative to the world anchor reference A.

Variable: m_UIAnchorReferenceB

Type: RectTransform

Description: The UI anchor reference B position on the UI map that is relative to the world anchor reference B.

Variable: m_WorldAnchorReferenceA

Type: Transform

Description: The position in the world where the UI anchor reference A will match in the minimap.

Variable: m_WorldAnchorReferenceB

Type: Transform

Description: The position in the world where the UI anchor reference B will match in the minimap.

Variable: m_MiniMapRenderMaterial

Type: Material

Description: The material used for rendering the live minimap.

Variable: m_RenderLayers

Type: LayerMask

Description: Determines which layers to show in the minimap.

Variable: m_CameraOffset

Type: Vector3

Description: The offset of the camera from the target.

Variable: m_CameraRenderSize

Type: float

Description: The orthographic size of the camera.

Variable: m_CameraFarClipPlane

Type: float

Description: The far clip plane of the camera.

Variable: m_CameraRotation

Type: Vector3

Description: The rotation of the camera.

Variable: m_RotateMapWithLocalPlayer

Type: bool

Description: If true, the camera rotates according to the local player's rotation.

Mini Map Scene Object

Class Name: MiniMapSceneObject

Description: MiniMapSceneObject is a component that is part of the MiniMap system. It represents an object in the scene that will be registered and displayed on the minimap. Add this component to the scene object to be tracked and displayed.

Note: MiniMapSceneObject is used to represent objects in the scene that will be displayed on the minimap. Customize the variables according to the specific requirements of your game to control the appearance and behavior of the map icons for different GameObjects, such as adjusting the icon size, rotation, and visibility.

Variables:

Variable: m_OwnerIsGameObject

Type: bool

Description: Sets the owner of the MiniMapSceneObject to be the GameObject itself and prevents checking for a player or board source as the owner.

Variable: m_IsSpawnZone

Type: bool

Description: Determines if a spawn button should be added to the map icon object.

Variable: m_SpawnGrouping

Type: int

Description: The spawn grouping for the map icon object button.

Variable: m_SyncRespawnerGrouping

Type: bool

Description: Determines if the respawning character should have the respawner grouping synced. If true, the player can automatically respawn at the last selected spawn grouping, provided it is still valid. If it is not valid, the respawner will reset to the team grouping.

Variable: m_RegisterOnEnable

Type: bool

Description: Determines if the component should register itself.

Variable: m_SyncEnabled

Type: bool

Description: Determines if the map icon object should be hidden when this GameObject is disabled.

Variable: m_RegisterDeath

Type: bool

Description: Determines if the Death event should be registered. If true, the map icon object will be disabled on death.

Variable: m_RegisterRespawn

Type: bool

Description: Determines if the Respawn event should be registered. If true, the map icon object will be re-enabled on respawn.

Variable: m_OverrideType

Type: OverrideType

Description: Determines the override type for the icon rotation. This specifies whether the icon should rotate with the GameObject or not.

Variable: m_FriendlyIcon

Type: Sprite

Description: The icon to be used for the GameObject when it is on the friendly team.

Variable: m_EnemyIcon

Type: Sprite

Description: The icon to be used for the GameObject when it is on the enemy team.

Variable: m_CustomIcon

Type: Sprite

Description: A custom icon for the GameObject. This overrides the friendly and enemy icons and can be useful for scene objects.

Variable: m_TeamColorCustomIcon

Type: bool

Description: If using a custom icon, determines if the team color should be set when boarding. If false, it will use the boarding character's icon color.

Variable: m_IconSize

Type: Vector2

Description: The size of the icon.

Variable: m_RotateWithObject

Type: bool

Description: Determines if the icon should rotate with the GameObject.

Variable: m_Up

Type: Vector3Int

Description: Adjusts the up axis of the icon according to the GameObject. Each axis value can be -1, 0, or 1.

Variable: m_IconOffsetRotation

Type: float

Description: Adjusts the initial rotation of the icon.

Variable: m_ClampIcon

Type: bool

Description: If true, the icons will be clamped within the border of the map display.

Variable: m_HideDistance

Type: float

Description: Sets the distance from the target after which the icon will not be shown. A value of 0 will always show the icon.

Mini Map UI Object

Class Name: MiniMapUIObject

This component should not be added to any gameobjects, it is not required to do so. It will add itself upon registration of a MiniMapSceneObject.

Description: MiniMapUIObject is a component that is part of the MiniMap system. It represents the UI object that will be created for the system and handles the position, rotation, and clamping of the object to the UI map border using a specified 2D shape collider.

Note: MiniMapUIObject is responsible for managing the UI objects that correspond to the scene objects on the minimap. It handles positioning, rotation, and clamping of the UI objects to the UI map border using a specified 2D shape collider. The UI objects are created based on the MiniMapSceneObject components attached to the scene objects.