Imitation Studios www.Imitationstudios.com
www.Imitationstudios.com/mscclanding Support@imitationstudios.com

Multi-State Character Controller (MSCC)
10/1/2022

For video tutorials and other written guides: www.Imitationstudios.com/mscclanding

For questions and other support:
Support@imitationstudios.com

This package requires you to set up layers and input axes to function. For a video tutorial and written guide visit www.imitationstudios.com/msccgettingstarted, See MSCC Documentation Section 1.0 for more information. The following, section 0.1, contains an input and layer quick reference guide.

Thank you for your purchase! For free assets, click the link below to check out our buyer rewards program! We sincerely hope you enjoy using this package!
www.imitationstudios.com/vouchers

Section 0:
References and Preliminary Information

Section 0.0:
Layer and Input quick-reference guide:

Add the following Input Axes:
"Sidestep"; Positive Key: e; Negative Key: q
"UpDown"; Positive Key: z; Negative Key: x

Layers:
Step 1. New Layers:
1. "Ground"
2. "ProjectedPlayer"
3. "CharacterRep"
4. "DisableLand"

Step 2. Disabled Collisions:
1. "CharacterRep" : "CharacterRep"
2. "ProjectedPlayer" : "CharacterRep"
3. "ProjectedPlayer" : "ProjectedPlayer"
4. VERY IMPORTANT: the projected player is essentially a collider that is NOT visually representative of the character mesh. If you have other game implementations, like projectiles for example, that should only contact the character mesh, you will need to disable certain collisions with the "ProjectedPlayer".

Step 3. Set the following layers:
1. "Player" Gameobject – "ProjectedPlayer"
2. "Gyro" Gameobject – "CharacterRep"
3. Any Walkable Surface – "Ground"

Step 4. Set the following layer masks:

Camera Collision Script located on "MainCamera" gameobject
1. Camera Clip Layer

a. Disable "Projected Player"
b. Disable "Character Rep"

MSCC Script located on "Player" gameobject
1. Ground Layer
    a. Enable only "Ground"
2. Overhead Layer (Optional)
    a. Enable "Ground"

3. Water Layer
    a. Enable only "Water"

4. IKLayer
a. Enable "Ground"
b. (Optional) Enable custom IK Layers

5. DisableLandLayer
    a. Enable only "DisableLand"

Section 0.A:
Profile Information:

1. Beginning with update 1.1, the primary settings of the multi-state character controller, listed in this documentation, have been moved to a scriptable object class. The local transform, animation, and object references required are still listed under the MultiStateCharacterController Script.
2. To create a new profile:
    a. Right click in your project tab.
    b. Click on Create/MSCC/MSCC Profile
    c. Assign the new profile to your player.
3. **Important:** Profiles do not update after the start of the scene. To make runtime parameter changes, modify the MultiStateCharacterController.settings class. **DO NOT modify the profiles at runtime.**
4. The profile in use can be changed to a seperate profile at runtime through the following method:
   MultiStateCharacterController.AttemptProfileChange(MSCCProfile profile)
   Note: A very simple demo script has been included to serve as an example.

Section 0.1:
Level-Design Tips:

1. Do not assign barriers and other obstacles to the ground layer.
2. Use ramps as the ground layer colliders for stairs and assign a separate IK layer for the steps.
3. Keep transitions between ground objects as smooth as possible.
4. Rail-system positional events can enable and disable objects such as invisible walls.
5. Water surfaces are expected to be surfaces not volumes.
6. Use the "DisableLandLayer" (See Section 3.0.1.a.v) mask on the MSCC to prevent the player from landing over certain areas. Note: this simply does not allow the player to exit the flying state. It does not force the player into the flying state.
7. Use the "Overhead" Layer (See Section 3.0.1.a.ii) to block the player from exiting the crouching substate while under low objects.
8. Invisible walls not in the ground layer can be used to contain player flight, and the "DisableLandLayer" can prevent a player from exiting the flying state. (See Section 3.0.1.a.v)

Section 0.2:
Changing Animations:

Directional animations can be substituted in this package if they meet the following requirements:

1. Animations must represent motion in at least the four cardinal directions.
    a. Backwards motion can be the reverse of the forward motion.
    b. The same foot must lead in both the left and right directional animations.
    c. Animations representing movement at the 45, 135, 225, and 315-degree angles are usually recommended but not always necessary.
2. Swimming and Flight animations are setup in the exact same manner as normal grounded animations, but their up and down motion is simulated by essentially rotating the animator's planar x axis. The projected player never rotates along its x axis. Be aware of this if you experience the character mesh clipping when in the swimming and flying states as properly setting another collider on the character representation can alleviate these problems.
3. Root motion is **NOT** supported and therefore must be baked into the pose.
4. Animation speed is multiplied by the velocity's magnitude when greater than the "Animation Base Transition" parameter of the MSCC script, located on the projected player, as speeds below this value are assumed to be in a "blending" state. (See 3.0.1.n.vi and **3.0.1.n.vi.1.c**)
    a. The "Animation Base Transition" has a default value of 0.9 which is the velocity's magnitude when the "Grounded Transitional Speed" is set to 1 and the "Running Speed Multiplier" is also set to 1.
    b. The "Animation Speed Multiplier" parameter of the MSCC script can be used to find the correct speed of an animation relative to the current animator base speed for easy adjustments at runtime, which can then be used to calculate the correct animation base speed. (See 3.0.1.n.iv)
    c. **The correct base animation speed for new directional animations should be calculated when the animator's magnitude parameter is equal to the animation base transition speed.** (See 3.0.1.n.vi and **3.0.1.n.vi.1.c**)

Section 0.4:
Inverse Kinematics:

This package is not intended to be, and never will be, a complete IK solution. All the IK elements in this package can be easily disabled by a single bool, "enableIk" on the MSCC script located on the projected player, so that you can implement your own IK solutions. Knowing this, if you wish to use our supplied IK system be aware of the following:

1. Animation Weight is set by a curve on each animation clip using the following parameters:
    a. IKLeftFootWeight
    b. IKRightFootWeight
    c. IKLeftHandWeight

d. IKRightHandWeight
e. IKBodyForward
   i. This moves the entire body forward by the specified amount.

2. Setting an IK Weight larger than 0 for any normal movement-related animation while using our system is highly discouraged.
3. The feet will always face forward when feet IK is enabled.

Section 1: Getting Started

Section 1.0:
Hierarchy Overview:
The player should be setup as follows:
(*Refer to visual reference, Figure 1.)

1) Player Unit
   a) Empty gameobject functioning as encapsulating parent.
      i) **CameraRig**
         (1) Child of Player unit
         (2) Focal position of the camera
         (3) Located at the player
         (4) Contains MultiStatePlayerInput Script
         (a) **XAxis**
            (i) Child of the CameraRig
            (ii) located at the local position (0, 0, 0)
               1. **Actual Camera**
                  a. Child of XAxis
                  b. Located at the furthest zoomed out location
                  c. Main Camera – Located at local position (0, 0, 0)
                  d. Contains Camera Collision Script (Optional)
                  e. Contains standard Camera Script
               2. **CameraPlaceHolder**
                  a. Empty Gameobject located at Actual Camera's location
                  b. Child of XAxis
      ii) **Player (Also called ProjectedPlayer)**
         (1) Child of Player Unit
         (2) Located at local position (0, 0, 0)
         (3) Contains Rigidbody
         (4) Contains CapsuleCollider (Material should be set to multistatePhyMat)
         (5) Contains MultistateCharacterController Script iii) **Character (Also called Character Representation)**

(1) Child of Player Unit
(2) Located on ground
(3) Contains Simple Health Script (Optional)
    (a) **Gyro**
        (i) Child of Character
        (ii) Located at local position (0, value equal to half of character height, 0)
            1. **Character Prefab**
                a. Child of Gyro
                b. Located at local position (0, negative value equal to half of character, 0) height)
                c. Prefab Containing Character Mesh
                d. Contains Animator
                e. Contains Animation Event Handler Script
                f. Contains Capsule Collider (Optional)
                g. Contains Rigidbody set to Kinematic (Optional)
                h. **All other player related scripts not associated with movement should be attached to this gameobject!** **Section 1.1:**

Setting up Input axes:

Input settings are located under "Edit/ProjectSettings/Input"

Unity Input Documentation:
https://docs.unity3d.com/Manual/class-InputManager.html

The following input axes need to be setup:

2. **Sidestep**
    1. Name: "Sidestep"
    2. Positive Key: e
    3. Negative Key: q
3. **UpDown**
    1. Name: "UpDown"
    2. Positive Key: z
    3. Negative Key: x

Section 1.2:
Setting up layers and layer masks:

Layer settings are located under "Edit/ProjectSettings/TagsAndLayers"
Layer collision settings are located under "Edit/ProjectSettings/Physics"

Unity Layer Documentation:
https://docs.unity3d.com/Manual/Layers.html
Unity Layer Collision Documentation:
https://docs.unity3d.com/Manual/LayerBasedCollision.html

The following layers need to be added:

1. "Ground"
2. "ProjectedPlayer"
3. "CharacterRep"
4. "DisableLand"

Disable the following collisions in the physics window:

1. "CharacterRep : CharacterRep"
2. "ProjectedPlayer : CharacterRep"
3. "ProjectedPlayer : ProjectedPlayer"
4. **VERY IMPORTANT: the projected player is essentially a collider that is NOT visually representative of the character mesh. If you have other game implementations, like projectiles for example, that should only contact the character mesh, you will need to disable certain collisions with the "ProjectedPlayer"**

Set the layer for the gameobjects as follows: (refer to hierarchy (Figure 1.) if necessary)

1. Player – Layer: "ProjectedPlayer"
2. Gyro (including children) – Layer: "CharacterRep"
3. Any walkable surface – Layer: "Ground"

Set the layer masks as follows:

1. **Camera Collision Script located on Main Camera**
    1. Camera Clip Layer
        1. Disable "ProjectedPlayer"
        2. Disable "CharacterRep"
2. **MultiStateCharacterController Script located on Player**
    1. Ground Layer
        1. Enable only "Ground"
        2. Overhead Layer (Optional)
            1. Enable "Ground"
    3. Water Layer
        1. Enable only "Water"
    4. Ik Layer
        1. Enable "Ground"
        2. Enable separate IK Layers (Optional)
    5. Disable Land Layer
        1. Enable only "DisableLand"

Section 1.3:
Changing the Character Mesh:

This process involves the Unity Ragdoll Wizard. See https://docs.unity3d.com/Manual/wizardRagdollWizard.html for more information.

1. Create or load a scene with a gameobject that can serve as flat ground.
2. Place the PlayerUnit_Empty prefab, located in "MultiStateCharacterController/Prefabs/Player", in the scene at the position (0, 0, 0)
3. Place your custom character mesh into the scene.
4. Remove any colliders attached to your character mesh.
5. Scale and position your character mesh so that it fits in the PlayerUnit_Empty's capsule collider.
6. Set the character mesh's parent to be the Gyro gameobject.
7. Attach the AnimationEventHandler script to your character mesh.
8. Attach an animator component to your character mesh.
9. Assign the animator's controller value to be "MultiState_Animator"
10. Run the ragdoll wizard on your character mesh and adjust the results as required.
11. Assign the character mesh's animator to the "animator" parameter of the "Player" MSCC script.
12. Assign every rigidbody in the ragdoll to the "Player" "Ragdoll" array parameter of the MSCC script.
13. Assign Layers Normally (See Section 1.2)
14. Set and the adjust the MSCC script IK parameters, as necessary. (See 3.0.1.q.iv – 3.0.1.q.xi)
15. Adjust "CharacterRepHeightOffset" and "BodyOffset" parameters of the MSCC script as needed. (See 3.0.1.b.iii and 3.0.1.q.xi)

Section 2:
Camera and Input

Section 2.0:
MultistatePlayerInput Script Parameters and Guidelines

This script is located on the "CameraRig" gameobject.

Purpose: The purpose of this script is to control camera rotation based on the set control scheme, move the camera towards the required location, generate an ACV every frame based on the control scheme, send the ACV to the MSCC script every frame, and request MSCC script grounded substate changes.

1. **MultistatePlayerInput Script Parameters**

a. Control Schemes

i. Control Scheme Enum

1. Classic - When using this control scheme, the player will respond similarly to that of classic MMORPGs in which player movement is normally independent of the camera, but movement can be associated with the camera depending on mouse input.
2. SemiModern (see AllowClassicSemiModernSwitching (2.0.1.a.ii)) – This is an extension of the classic control scheme that defines movement based on mouse input. Setting the Control Scheme to this value will default it to the Classic control scheme.
3. ModernUnlocked
   a. Type A (See IsModernUnlockedTypeA (2.0.1.a.iii)) - When using this control scheme, the player will rotate to face the movement direction. This control scheme is typically encountered in classic third-person platformers.
   b. Type B (See IsModernUnlockedTypeA (2.0.1.a.iii)) - When using this control scheme, the player will rotate to face the same direction as the camera as the player moves. This control scheme will typically be found in modern third-person RPGs.
4. ModernLocked - When using this control scheme, the player will always face the same direction as the camera, independent of player movement. This control scheme will typically be found in third-person shooters.

ii. Allow Classic Semi Modern Switching Bool

1. True – When the control scheme is set to classic, left-clicking will rotate the camera around the player without affecting player rotation, rightclicking will align the player with the camera, and holding the left and right mouse button will move the player towards the camera direction.
2. False – When the control scheme is set to classic, mouse input has no effect. Note: the camera's rotation will be fixed

iii) IsModernUnlockedTypeA Bool

1. True – See Control Scheme Enum (2.0.1.a.i.3.a)
2. False - See Control Scheme Enum (2.0.1.a.i.3.b) iv.

Character Controller

1. This script should be attached to the player (projectedPlayer) gameobject.

b. Required Transforms

i. Character Transform

1. The transform of the character gameobject (Refer to hierarchy overview)

ii. Camera X-Axis Gimbal

        1. The transform of the CameraXAxisGimbal gameobject (Refer to hierarchy overview)
    iii. Camera Holder Transform
        1. The transform of the CameraHolder gameobject (Refer to hierarchy overview)
    iv. Actual Camera Transform
        1. The transform of the ActualCamera gameobject (Refer to hierarchy overview)

c. Basic Settings
    i. Camera Movement Speed
        1. This value represents the speed at which the camera rig's position adjusts towards the character representation's position. A higher value will cause the camera to move slower, and a lower value will cause the camera to move faster. ii. Character Rep Adjustment Speed
        1. This value represents the effect the change in the MSCC's character representation speed has on the camera. As the character representation moves faster towards the projected player in certain states, the camera slows down to reduce jarring camera motion.
    iii. Key Sensitivity
        1. This value represents the effect keystrokes have on rotation when relevant.
    iv. Mouse Sensitivity
        1. This value represents the effect mouse movement has on rotation when relevant.
    v. X Rotation Max
        1. This value represents the maximum local rotation value the XAxis gameobject can reach.
    vi. X Rotation Min
        1. This value represents the minimum local rotation value the XAxis gameobject can reach. A value of 270 would allow the camera to look up at an angle of 90 degrees.
    vii. Camera Player Target Offset
        1. The positional offset of the lookAt function of the camera. In most circumstances, a value of (0, 0, 0) would cause the camera to focus on the player's feet.
    viii. Reverse Camera Offset Multiplier
        1. This value represents the amount that the camera rig moves behind the player when the player is moving backwards.
        2. This is used to compensate for animations which would normally place part of the character off-screen.

d. Camera Collision
    i. Camera Collision

1. The camera collision script located on the main camera gameobject (Refer to hierarchy overview). This is optional.

e. Zoom
   i. Zoom
      1. The current zoom value clamped between 0 and 1. A value of 0 is fully zoomed out. A value of 1 is fully zoomed in. Maximum zoom is defined by the world location of the CameraPlaceHolder and ActualCamera gameobjects (Refer to hierarchy overview).
   ii. Scroll Multiplier
      1. The effect that scrolling has on the current zoom. A lower value will cause the camera to zoom slower. A higher value will cause the camera to zoom faster.
   iii. Camera Minimum Distance
      1. This defines the minimum world position distance the camera can zoom in to.

f. Rail System Settings
   i. Camera Dolly Rotation Speed
      1. This is the camera's rotational speed when attached to a rail system dolly (See Rail-Systems for more information). It is recommended that this value is kept low for smooth transitions in and out of rail-systems. ii. Camera Dolly Movement Speed
      1. This is the camera's movement speed when attached to a rail system dolly (See Rail-Systems for more information). It is recommended that this value is kept low for smooth transitions in and out of rail-systems.
   iii. Actual Camera Transition Speed
      1. This is the actual camera gameobject's movement speed when attached to a rail system dolly (See Rail-Systems for more information). It is recommended that this value is kept low for smooth transitions in and out of rail-systems.
   iv. Actual Camera Adjustment Speed
      1. This is the speed at which the current actual camera transition speed adjusts towards the actual camera transition speed. (See Rail-Systems for more information). It is recommended that this value is kept low for smooth transitions in and out of rail-systems.

g. Grounded Substates
   i. Require Substate Key Hold Bool
      1. True – A substate is only active if the toggle key is being held else the MSCC will default to the running substate while in the grounded state.
      2. False – Substates will toggle based on the toggle key, and the key can be released without defaulting to the running substate.
   ii. Crouch Toggle

1. This is the key that will cause the player to enter the crouching grounded substate. To disable, set this value to none.

iii. Sprint Toggle

1. This is the key that will cause the player to enter the sprinting grounded substate. To disable, set this value to none.

iv. Walk Toggle

1. This is the key that will cause the player to enter the walking grounded substate. To disable, set this value to none.

h. Amendment: Remote Sampling

i. Enable Remote Sampling

1. Should the input script check for remote access opportunities?
2. Remote Access must also be enabled on the rail systems.

ii. Remote Sampling Layer Mask

1. Defines the layers of the triggers and obstructions affecting remote sampling.
2. A separate remote sampling layer is not required but has been added for simplicity.

iii. Remote Sampling Request Delay

1. Time required after attempting remote access before a second attempt is made.

i. Amendment: Lunge Keybinds

i. Note: Lunge Actions can be initiated through multiple means. See https://www.imitationstudios.com/mscclunge **for more information.**

ii. Lunge KeyBinds

1. Name
   a. The name identifier of the lunge action.
2. Key (optional)
   a. The key used to initate the lunge action.
3. Enabled
   a. Is the lunge action enabled?
4. Lunge Advanced Direction
   a. Flat Vector
      1. The lunge direction is calculated relative to player input in the x and z directions.
   b. Free Vector
      1. The lunge direction is calculated relative to player input in the x, y, and z directions.
   c. Relative Static
      1. The lunge direction is calculated relative to the player’s direction based on the lunge action’s simple direction.
5. Cooldown
   a. The reset time of the lunge action.

6. Lunge Action
    a. Simple Direction
        1. The relative direction used when the advanced direction is set to relative static.
    b. Intenstiy
        1. The force intensity multiplier of the lunge action.
    c. Duration
        1. The amount of time the lunge action will last for.
    d. Recovery Time
        1. The amount of time spent in the recovery state after the lunge action.
    e. Animation Trigger
        1. The animation trigger of the lunge action.
    f. Can Transition From
        1. Defines the movement state the lunge action can be initated from.

Section 2.1:
CameraCollision Script Parameters and Guidelines

This script is located on the "Main Camera" gameobject.

Document Definitions:

1. Frame: This script checks for clipping and obstructions by spherecasting between five pairs of points. One part of the pairs is around the character rep, another part is around the camera. The camera's points should be in line with its viewport, and the distance forward of the camera can be adjusted based on the results. The character's points are normally located at a single point value in the center of the character unless different results are required.

Purpose: The purpose of this component is to prevent the camera from clipping into defined objects or from having it's view of the player obstructed by moving the camera forward based on the result of five spherecasts, four of which at each predefined corner of the frame, and one at the camera's center. This script can be removed completely without affecting any other aspect of the MSCC functionality if you wish to use your own camera collision solution.

2. **CameraCollision Script Parameters**
    a. Character Rep
        i. This value should be set to the "Character" gameobject.
    b. Player Offset
        i. This value determines where the focus of the camera's look at function is in relation to the character rep's location. Normally only the Y value is adjusted so that the camera is not looking towards the players feet.

c. Movement Speed

i. This value is the speed that the camera moves forward and backwards at when preforming its primary function. A speed value that is too high may seem jarring, while a speed value that is too low may allow the user to see the camera clipping through an object for a fraction of a second or possibly longer. d. Camera Clip Layer

i. This layer defines what the camera tries to avoid clipping with. This script will not function properly if it is attempting to avoid clipping with the Projected Player or Ragdoll (See Section 1.2 for setting up layers). Use the minimum distance parameter for problems with the camera clipping with the character mesh (1.i).

e. Border Width

i. This value is the width of the frame around the camera and the base width of the frame around the character.

f. Border Height

i. This value is the height of the frame around the camera and the base height of the frame around the character.

g. Border Distance

i. This value is the forward distance of the frame in front of the camera and the base forward distance of the frame in front of the character.

h. Border Radius

i. This value represents the radius of the spherecasts. Increasing this value "thickens" the borders for different camera results as needed.

i. Minimum Distance

i. This value represents the closest world position distance the camera can be at in relation to the character. Setting this value too low will produce undesired results.

j. Player Multiplier

i. The width and height of the player's frame is the border width and border height parameters multiplied by this value. This value is normally set to 0 unless different results are required.

k. Distance Multiplier

i. The forward distance of the player's frame is the border distance parameter multiplied by this value. This value is normally set to 0 unless different results are required.

l. Camera Offset

i. This value determines how far in front of the spherecast collision point the camera will be. This value is normally kept to a low value.

m. Draw Gizmos

i. When enabled and Gizmos are enabled in the render window, the camera's and player's frames will be visualized. This is used for initial camera collision setup and debugging purposes.

Section 3:

Player Movement

Section 3.0:
MultistateCharacterController Script Parameters and Guidelines

This script is located on the “Player” gameobject.

Document Definitions:

1. The terms “Player” and “ProjectedPlayer” are used interchangeably in this document.
2. This script is referred to as the “MSCC script”

Purpose: The purpose of this component is, ultimately, to move the character representation gameobject, with regards to the assigned movement state and movement substate, when applicable, and to update the animator’s parameters.

3. **MultistateCharacterController Script Parameters**

Amendment: Movement State Settings

Active Movement States

Defines the movement states that the player can and cannot enter.

Requested State

The request movement state that the player should enter into. Note: setting this value does not gaurnatee that the player will stay in the requested state for any meaningful length of time.

Eject player from disabled states

Should the player’s movement state be recalculated when the player is in a disabled state?

a. **Layer Masks**
   i. Ground Layer
      1. This layer represents smoothed, walkable surfaces.
      2. Rough terrain details should be handled in the IK layer as much as possible, such as stair steps.
      3. Walls and other obstructions should not be present in this layer.
      4. When sliding is enabled, there is no guarantee the player will be able to reach every position on the ground layer due to steep inclines.
      5. You are not required to have a mesh attached to objects in this layer in order to function, only a collider.
   ii. Overhead Layer

1. This layer represents objects that block the player from exiting the crouching substate.
2. You are not required to have a mesh attached to objects in this layer in order to function, only a collider.

iii. Water Layer

1. This layer represents overhead water surfaces.
2. The swimming state is checked by raycasting. This means the water object should be a very thin “trigger” marked collider at the top of the water volume. Checks preformed within a collider intended to be used as a volume and not a surface will not trigger the water state under the default physics settings.
3. You are not required to have a mesh attached to objects in this layer in order to function, only a collider.

iv. IK Layer

1. Refer to Section 0.4
2. This layer represents IK eligible surfaces.
3. The ground layer is included in this mask under normal circumstances.
4. The intent of having a separate Ik Layer is to allow you to put the rough details of the terrain in this layer, so the physics of the projected player can operate more smoothly.
5. You are not required to have a mesh attached to objects in this layer in order to function, only a collider.

v. Disable Land Layer

1. This layer prevents the player from exiting the flying state manually over gameobjects in this layer.
2. This does not protect against the player moving into higher priority override states, such as swimming.
3. This does not force the player **into** the flying state.
4. You are not required to have a mesh attached to objects in this layer in order to function, only a collider.

vi. Amendment: Blink Collision Layer

1. This layer represents objects that will cause the player to fail the blink on a calculated collision.

b. Visual Representation

i. Character Representation

1. This transform represents the “Character” gameobject’s transform (see Section 1.0)

ii. Character Rep Adjustment Speed

1. This is the speed that the current character rep speed adjusts to the state specific character rep speed.
2. This package functions by having the object containing the character mesh smooth damp towards the object that does the physics

calculations. This greatly smooths visual character movement but can result in strange visual occurrences under normal circumstances, such as the character making a jump you would visually not expect it to make. To compensate for this, the character mesh speeds up and slows down while chasing the projected player based on movement state. (See Section 3.0.1.c.i – iv for state specific information.)

3. This value should be lowered if the character seems to be snapping on state changes.
4. This value should be raised if visual oddities, such as the jump scenario, occur.
5. The Camera's movement speed Is inversely proportional to this value in order to reduce camera snapping.

iii. Character Rep Height Offset

1. This adjusts the character height in relation to the projected player.
2. There is another parameter in the Ik settings that has a similar function. Be aware, that the Ik setting is based of foot Ik weight, and this parameter is not. (See 3.0.1.q.xi)

c. Character Representation Movement Speeds

i. Grounded Character Rep Speed

1. This value represents the character rep speed in the grounded state.
2. This value should be around 0.2 for smooth results.
3. See Section 3.0.1.b.ii for Generic Character Representation Movement Speed Information.

ii. Falling Character Rep Speed

1. This value represents the character rep speed in the falling state.
2. This value should be very low for ideal results.
3. See Section 3.0.1.b.ii for Generic Character Representation Movement Speed Information.

iii. Flying Character Rep Speed

1. This value represents the character rep speed in the flying state.
2. See Section 3.0.1.b.ii for Generic Character Representation Movement Speed Information. iv. Swimming Character Rep Speed
1. This value represents the character rep speed in the swimming state.
2. See Section 3.0.1.b.ii for Generic Character Representation Movement Speed Information.

d. Ground Check Raycast Distances By State

i. Ground Raycast Distance

1. This Value represents the length of the raycast used to check for the grounded movement state while in the grounded movement state.
2. This value is used in conjunction with the constant y grounded force to prevent the player from unnecessarily entering the falling state. (See 3.0.1.h.ii)

ii. Falling Raycast Distance
   1. This Value represents the length of the raycast used to check for the grounded movement state while in the falling movement state.
   2. This value should be kept low so the landing transition does not occur prematurely.

iii. Recovery Raycast Distance
   1. This Value represents the length of the raycast used to check for the grounded movement state while in the recovery movement state.

iv. Sliding Raycast Distance
   1. This Value represents the length of the raycast used to check for the grounded movement state while in the sliding movement state.
   2. This value should be increased if the player is prematurely exiting the sliding state.

v. Jumping Raycast Distance
   1. This Value represents the length of the raycast used to check for the grounded movement state while in the jumping movement state.
   2. This value should be set to 0 under normal circumstances.

vi. Flying Raycast Distance
   1. This Value represents the length of the raycast used to check for the grounded movement state while in the flying movement state.
   2. This value should be relatively low, so landing does not occur prematurely.

vii. Swimming Raycast Distance
   1. This Value represents the length of the raycast used to check for the grounded movement state while in the swimming movement state.
   2. This value should be used in conjunction with the swimming raycast offset multiplier if the player is swimming in very shallow water.

(3.0.1.i.iii)

e. Basic Controls

i. Player Width
   1. This is the predefined player width normally set to the capsule collider's radius under normal circumstances.

ii. Player Height
   1. This is the predefined player height normally set to the capsule collider's height under normal circumstances.

f. Directional Vector Modifiers

i. Grounded Transitional Speed
   1. This value represents the primary base speed value upon which all other states are based. Setting this value to 1 while leaving all other settings at their default values will result in the character representation's normal movement directional magnitude having a value of 0.9.

ii. Average Y Value Count
   1. Y Value motion is averaged by the number of frames defined by this parameter.
   2. This value should be kept as high as possible as long as the user does not perceive excessive y value motion lag.

iii. Maximum Rotational Speed
   1. This value represents the maximum possible player rotation speed.
   2. This value is normally used to limit mouse input rotation but will affect key rotation as well if it is low enough.

iv. Rotational Speed Multiplier
   1. This value effects key input rotation completely independent of camera rotation.
   2. By default, this only occurs in the classic control scheme while the player is holding only the left mouse button.

Amendment A.3.1: Physics Gyro:

Enable Physics Gyro

Is the physics gyro enabled? The physics gyro rotates the collider to better represent horizontal swimming and flying animation.

Physics Gyro Radius Multiplier

Collider radius size multipler for the swimming and flying movement states.

(Flying and Swimming - Respective) Physics Gyro Maximum Animation Angle

The maximum amount of gyro rotation used to replicate the visual appearance of animations.

(Flying and Swimming - Respective) Physics Gyro Input Factor

Fader for gyro control between player input and character representation velocity.

(Flying and Swimming - Respective) Physics Gyro Magnitute Factor

A constant value against the gyro input factor resulting in the maximum animation angle.

(Flying and Swimming - Respective) Physics Gyro Transition Speed

The rotation speed of the physics gyro.

g. Grounded Movement Substate Modifiers

i. Default Grounded Movement Substate (Amendment: “Replaced Grounded Movement Substate”)
   1. This value represents the default grounded movement substate.
   2. This value will not affect key input in any way.

ii. Running Speed Multiplier

1. The “groundedTransitionalSpeed” is multiplied by this value to achieve the final running speed. (See. 3.0.1.f.i)
2. **This value should NOT be treated as a substitute for the “groundedTransitionalSpeed”**

iii. Walking Speed Multiplier
1. The “groundedTransitionalSpeed” is multiplied by this value to achieve the final walking speed. (See. 3.0.1.f.i)

iv. Crouching Speed Multiplier
1. The “groundedTransitionalSpeed” is multiplied by this value to achieve the final crouching speed. (See. 3.0.1.f.i)

v. Sprinting Speed Multiplier
1. The “groundedTransitionalSpeed” is multiplied by this value to achieve the final sprinting speed. (See. 3.0.1.f.i)

vi. Crouching Collider Difference
1. This value represents the projected player’s collider height difference between the crouching state and all other states.
2. The “overheadLayer” holds the crouching state under low objects. (See 3.0.1.a.ii)

h. Rigidbody Force Modifiers

i. Force Multiplier
1. The final calculated directional vector, based on input, movement state, and external factors, is multiplied by this value to produce the final force value applied to the rigidbody.

ii. Constant Y Grounded Force
1. This value represents constant Y value force applied to the projected player to keep it on the ground.
2. Setting this value to a high value will cause the player to slow down while climbing and speed up while moving downhill.
3. This value is used in conjunction with the ground raycast distance to prevent the player from unnecessarily entering the falling state. (See 3.0.1.d.i)

iii. Variable Distance Y Value Force
1. This value represents a variable Y value force multiplier applied based on the attempted climb angle to reduce bouncing.

iv. Normal Drag
1. This value is the rigidbody’s normal drag value when the movement state does not require a different drag value.

v. Gravity
1. This value represents the downward directional vector applied in the falling state.

vi. Fall Velocity Preservation Factor

1. This value represents the degree to which the player decelerates planar forces while in the falling state.
2. This value is typically lower than the Fall Velocity Y Preservation Factor to produce a better motion curve. (See 3.0.1.h.vii)

vii. Fall Velocity Y Preservation Factor

1. This value represents the degree to which the player decelerates y value forces while in the falling state.
2. This value is typically higher than the Fall Velocity Preservation Factor to produce a better motion curve. (See 3.0.1.h.vi)

i. Slide Controls

i. Enable Sliding

1. Set this value to false to disable advanced sliding.
2. Note: advanced sliding is the only prevention against allowing the player to climb steep angles.

ii. Slide Constant Y

1. This value represents constant Y value force applied to the projected player while in the sliding state.
2. This value should be set relatively high to prevent the player from unnecessarily entering the falling state and to prevent the player from prematurely exiting the sliding state.

iii. Slide Physics Checks

1. This value represents the possible directions the player will be able to slide towards.
2. Degrees between possible slide directions is equal to 360/SlidePhysicsChecks
3. **Caution: Raising this value has a large effect on performance and does not always lead to better results.**

iv. Minimum Fall Angle

1. This value defines the minimum angle that represents a fall instead of a slide.
2. This value prevents the player from throwing itself off leading edges.

v. Maximum Climb Angle

1. This value represents the maximum angle the player will try to climb.
2. Note: Ultimately, this value is rather inconsequential as there are many ways to circumvent it, but it is nonetheless an extra layer of climb protection.

vi. Slide Entry Angle

1. This value defines the minimum angle required to enter the sliding state.
2. This value should be much higher than the Slide Exit Angle to prevent slide state stuttering. (See 3.0.1.i.vii)

vii. Slide Exit Angle

1. This value defines the maximum angle required to exit the sliding state.
2. This value should be much lower than the Slide Entry Angle to prevent slide state stuttering. (See 3.0.1.i.vi)

viii. Slide Height Offset
1. This value represents the Y value offset for the linecast check for obstructions between the projected player's feet and the tested slide location.

ix. Min Slide Range
1. This value defines the minimum slide range used when the player is not currently in the sliding state.
2. This value should be much lower than the max slide range. (See 3.0.1.i.x)
3. The player will be pulled towards steep grades if this value is too high.

x. Max Slide Range
1. This value defines the maximum slide range used when the player is currently in the sliding state.
2. This value should be much higher than the max slide range. (See 3.0.1.i.ix)
3. The player will not maintain the sliding state if this value is too low.
4. This value effects the overall slide speed.

xi. Slide Speed Multiplier
1. This value represents a speed adjustment for the sliding state.
2. This value effects the overall slide physics. The slide ranges may need to be adjusted to compensate for changes to this value. (See 3.0.1.i.ix and 3.0.1.i.x)

xii. Slide Recovery Time
1. This value represents time spent in the recovery state after exiting the sliding state.
2. This functions as a stutter control.

j. Jumping Controls

i. Require Ground State
1. True – The player is required to be grounded to be eligible to enter the jumping state.
2. False – The player is not required to be grounded to be eligible to enter the jumping state.
    a. The player will be able to "double jump" as long as the fall recovery time is set to 0. (See 3.0.1.j.ii)
    b. The number of jumps the player is able to preform while falling is not limited.

ii. Fall Recovery Time
1. This value represents time spent in the recovery state after exiting the falling state.

2. This value is used to compensate for landing animations. iii. Jump Force
1. This value represents the force applied in the jump direction.
2. This value is linearly reduced each frame for the jump duration. (See 3.0.1.j.iv)

iv. Jump Duration

1. This value defines the duration of which the linearly time-reduced jump force is applied. (See 3.0.1.j.iii)

v. Jump Delay

1. This value defines the time initially spent in the jumping state before movement actually occurs.
2. This value is used to compensate for the jumping animation "build-up". vi. Zero Pre Jump Velocity
   1. This value represents whether the player's rigidbody velocity is set to 0 before the jump.
   2. Note: Rigidbody velocity is normally not maintained in any substantial manner under normal operations regardless of this setting.

vii. Jump Horizontal Force Multiplier

1. This value represents the effect planar motion will have on the jump cycle.

k. Flying Controls

i. Automatic Land

1. This value controls whether the player will exit the flying state based on the flying raycast distance or if it will stay in the flying state regardless of the presence of ground directly below the player. (See 3.0.1.d.vi)

ii. Flying Drag

1. This value defines the projected player's rigidbody's drag while in the flying state.

iii. Flying Base Speed Multiplier

1. The base flying state speed is equal to the "groundedTransitionalSpeed" multiplied by this value. (See. 3.0.1.f.i)

iv. Flying Dive Multiplier (Additive)

1. The final flying speed when moving downwards is equal to the base flying state speed multiplied by this value linearly interpolated between 0 and 90 degrees downward.
2. This value should be positive if the player should speed up when moving downwards.

v. Flying Ascension Multiplier (Additive)

1. The final flying speed when moving upwards is equal to the base flying state speed multiplied by this value linearly interpolated between 0 and 90 degrees upward.
2. This value should be negative if the player should slow down when moving upwards.

vi. Flying Constant Forward
    1. When this value is enabled the player constantly moves forward in the flying state.
    2. Note: the “freeVectorInput” “gyroCalculationMethod” is not recommended for applications that have this value enabled. (See 3.0.1.n.iii)

vii. Flying Max Rotation Speed Multiplier
    1. The flying state maximum rotation speed is equal to the “maximumRotationSpeed” multiplied by this value. (See 3.0.1.f.iii)

viii. Flying Key Rotation Speed Multiplier
    1. Flying state rotational movement in the unlocked modern type A and B control schemes are modified by this value. (See 2.0.1.a.i.3)

ix. Flying Gravity
    1. This value represents constant downward force applied while in the flying state.
    2. Note: the “freeVectorInput” “gyroCalculationMethod” is not recommended for applications with a flying gravity greater than a value of 0. (See 3.0.1.n.iii)

x. Enable Flying from Falling
    1. True – The player can enter the flying state from the falling state by pressing the jump key as long as the player will not immediately land. This is dependent on the automatic land parameter. (See 3.0.1.k.i)
        a. The player will be able to enter the flying state after jumping if it meets the requirements listed above.
    2. False – The player will not enter the flying state unless a custom function requests that it does.
        a. **This effectively disables the flying state under normal operations.**

l. Swimming Controls
  i. Swimming Drag
    1. This value defines the projected player’s rigidbody’s drag while in the swimming state.
  ii. Swimming Speed Multiplier
    1. The swimming state speed is equal to the “groundedTransitionalSpeed” multiplied by this value. (See. 3.0.1.f.i)
  iii. Swimming Raycast Offset
    1. This represents the value of which the swimming raycast is offset by its Y value.
    2. This should be used in conjunction with the swimming raycast distance to adjust for the characters position in the water required before entering the swimming state. (See 3.0.1.d.vii)
  iv. Surface Break Adjustment

1. This value represents the y location offset of the player against the top of the water plane.
2. Note: this value is limited in its functionality.

v. Allow Dive

1. When this value is enabled the player can freely dive below the water plane.
2. Note: when this value is disabled, the player will still dive below the water surface upon entry, depending on the preliminary forces, but will immediately adjust back toward the water surface level afterwards. m. Ragdoll Controls

i. Ragdoll

1. This value represents every rigidbody component created by the Unity ragdoll wizard. (See 1.3)

ii. Initial Ragdoll Y Velocity

1. This value represents the initial ragdoll y velocity every rigidbody in the ragdoll array is set to upon entry into the ragdoll state.
2. This prevents the ragdoll from glitching through the ground upon transitioning into the ragdoll state.

n. Animation Settings

i. Animator

1. This is the animator component located on the gameobject containing the character mesh. (Refer to Player Unit Hierarchy)

ii. Character Gyro Control

1. This is the “Gyro” gameobject’s transform. (Refer to Player Unit Hierarchy)

iii. **Gyro Calculation Method**

1. This value defines how the gyro transform rotates to visualize up and down motion and how the x and z animator variables are calculated while using free vectors.
2. Note: The animator is setup for freeVectorInput by default. The blend values of the animator may need to be raised or lowered for other Gyro Calculation Methods.
3. *Method 1*: characterRepVelocity
    a. This method is based on the Character Rep’s smoothdamp’s reference velocity variable.
    b. **Warning**: This method may be unstable during certain maneuvers or in certain directions.
        1. This usually manifests as the gyro snapping.
4. *Method 2*: freeVectorInput
    a. This method is based on the user input.
    b. This is the default Gyro Calculation Method.
    c. Cons

1. This method fails to represent external forces including constant forward and flying state gravity. (See 3.0.1.k.vi and 3.0.1.k.ix)

d. Pros

1. This method is very responsive to the user.
2. This method is very stable.

5. *Method 3*: externalTargetDirectional
    a. This method is based on the Projected Player's calculated directional movement.
    b. **Warning**: This method may be unstable during certain maneuvers or in certain directions.
        1. This usually manifests as the gyro snapping.
6. *Method 4*: characterRepPlayerOffset
    a. This method is based on the difference in location between the character representation and the projected player.
    b. Cons
        1. This method is very conservative in its motion and may appear rather unresponsive to the user.
        2. It is highly recommended that the animator's flight direction animation's blend values are reduced to exaggerate the flying state's animations when using this method.

    c. Pros

    1. This method will adjust for external forces.
    2. This method is very stable.

iv. Animation Speed Multiplier

1. This value directly effects the calculated animation speed sent to the animator.
2. **This value should be changed for calculating the correct animator speed for new animations only!**

v. Grounded Maximum Animation Speed

1. This value represents the maximum speed a grounded animation can be multiplied by. vi. Animation Base Transition

1. This value defines the blend value of the animator for directional animation before their speed is scaled.
    a. For Example: the idle animation has an animator blend value of 0. The forward run animation has an animator blend value of 0.9. The Animation Base Transition should be set to 0.9.
    b. Example Continued: the animator blend value represents velocity. When the velocity the is below the animation base transition value the animation's speed will not be scaled. Only

after the velocity exceeds the animation base transition will the speed be scaled.

c. **The correct base animation speed for new directional animations should be calculated when the animator's magnitude parameter is equal to the animation base transition speed.**

vii. Grounded Animation Substate Transition Speed

1. This value represents the speed that the animator blends at when transitioning grounded movement substates.

o. Swimming Animation Settings

i. Swimming Maximum Animation Speed

1. This value represents the maximum speed the animator will multiply directional animations by when in the swimming state.

ii. Swimming Animation (Amendment - Gyro speed has been isolated) Transition Speed

1. This value represents the speed at which directional animations are blended while in the swimming state. iii. Swimming Minimum Gyro Depth

1. This value defines the minimum distance below the water surface that is required for the gyro to activate while in the swimming state.

Amendment A.1: Gyro Fade Distance:

The distance below the minimum gyro depth in which the gyro will linearly lose intensity as the player nears the surface.

Amendment A.2 Gyro Fade Multiplier:

The rate at which gyro intensity lost as the player traverses the fade zone.

iv. Swimming Free Vector Upper Vert Mag Limit

1. This value represents the degree of upward motion visualized while in the swimming state based on the gyro calculation method. (See 3.0.1.n.iii)

v. Swimming Free Vector Lower Vert Mag Limit

1. This value represents the degree of downward motion visualized while in the swimming state based on the gyro calculation method. (See 3.0.1.n.iii)

p. Flying Animation Settings

i. Flying Maximum Animation (Amendment - Gyro speed has been isolated) Speed

1. This value represents the maximum speed the animator will multiply directional animations by when in the flying state.

ii. Flying Animation Transition Speed

1. This value represents the speed at which directional animations are blended while in the flying state.

iii. Flying Free Vector Upper Vert Mag Limit

1. This value represents the degree of upward motion visualized while in the flying state based on the gyro calculation method. (See 3.0.1.n.iii)

iv. Flying Free Vector Lower Vert Mag Limit

1. This value represents the degree of downward motion visualized while in the flying state based on the gyro calculation method. (See 3.0.1.n.iii) q. IK Controls

i. Enable Ik

1. This value completely enables or disables all IK elements. (See 0.4) ii. Enable Feet Ik Rotation

1. This value completely enables or disables rotational feet IK.
2. Note: the feet will always face forward when their weight is set to 1 and feet IK is enabled.

iii. Enable Hand Ik Rotation

1. This value completely enables or disables rotational hand IK.

iv. Left Foot

1. This value represents the left foot transform of the character prefab's armature.

v. Right Foot

1. This value represents the right foot transform of the character prefab's armature.

vi. Left Hand

1. This value represents the left hand transform of the character prefab's armature.

vii. Right Hand

1. This value represents the right hand transform of the character prefab's armature.

viii. Foot Height

1. This value defines the location of the calculated foot Ik target in relation to the IK raycast collision point's y value when the feet are projected down.

ix. Foot Length

1. This value defines the location of the calculated foot Ik target in relation to the IK raycast collision point's z value when the feet are projected forward.

x. Hand Thickness

1. This value defines the location of the calculated hand Ik target in relation to the IK raycast collision point's z value when the hands are projected forward.

xi. Body Offset

1. This value adjusts the character representation along its Y value when foot IK is enabled and weighted.
2. **Do not use this value as a substitute for the "characterRepHeightOffset" parameter. (See 3.0.1.b.iii)**

xii. Body Drop Factor

1. This value defines the maximum y value distance the feet can pull down the entire body to compensate for feet IK.

xiii. Body Offset Multiplier

1. This value multiplies the amount of body drop that is calculated. (See 3.0.1.q.xii)

Amendment: Body Drop Limit

2. This value caps the maximum distance the feet will attempt to move down.

xiv. Forward Projection Distance

1. This value represents the maximum distance hands and feet will be projected forward.

r. Events

i. On Player Land

1. This event is called when the player enters the grounded/recovery state from the falling state.
2. This event passes the total fall time as an argument.

ii. On Player Movement State Change

1. This event is called when the player initially changes into a different movement state.
2. The new movement state is passed as an argument.

s. Debug

i. Debug Mode Enabled

1. When this value is enabled, several helpful debug calls will be activated to help in both debugging and adjusting some parameters.
2. Note: the circularly drawn rays at the base of the player, when this value is enabled, represent the current slide range. (See 3.0.1.i.ix and 3.0.1.i.x)

Section 4: Rail-Systems and Triggers

Section 4.0:

Document Definitions:

Rail-System: A rail-system linearly moves the player along a set path of predefined length and position while rotation is controlled by one of several options.

Rail-System Dolly: A rail-system dolly is an object that linearly moves along a set path of predefined length and position in response to the player or another dolly moving along its own path.

Camera Dolly: A special Rail-System Dolly differentiating only in purpose as the Actual Camera gameobject can be assigned by the Rail-System to follow it as needed.

Primary Rail-System: A primary rail-system is a rail-system that the player directly interacts with.

Secondary Rail-System: A secondary rail-system is a system that the player does not directly interact with.

Rail-System Positional Event: A rail-system positional event is an event called based on the evaluated position of a rail system with respect to both primary and secondary systems. **Note: primary rail-system positional events reset upon entering and exiting the rail-systems while secondary rail-system positional events only reset upon loading the scene.**

Rail System Core Scripts Parameters and Guidelines

1. MultiStateRailSystem Script

a. Universally Applied Settings

i. **Note**: Universal Settings are applicable to both primary and secondary rail systems.

ii. Base Settings

1. Duration
    a. This value defines the length of the rail-system.
    b. This value is relative to speed or time.
    c. Secondary systems with dollies operating under default settings will use the primary rail-system's speed. Therefore, if the primary rail-system duration is 10 and the secondary rail-system duration is 15, the dolly will not advance past a duration of 10 without changing the camera dolly's settings.
2. Calculate Waypoint by Uniform Distance
    a. True – The position of the player or dolly along its respective rail system is dependent on the distance between the waypoints.
    b. False – Distance between a rail-system waypoint does not affect the position of the player or dolly.
3. Starting Waypoint
    a. This value represents the starting waypoint's transform.
    b. The rest of the waypoints should be a child of this transform.
    c. Waypoint order is determined by the order of the hierarchy.
4. Round Position
    a. This value represents whether the position in a rail system should be rounded based on the round to parameter. (See

4.0.1.a.ii.5)

5. Round To
    a. This value defines to what position the rail-system should round to. (See 4.0.1.a.ii.4)
    b. For Example: Setting this value to 10 would round the position to the tenth decimal place.

iii. Dolly Settings
1. On Rail Dollies
    a. This list defines all of the dollies moved by this rail-system including the camera dolly.
    b. The dollies in this list cannot be attached to this rail-system. They must be attached to a separate secondary rail-system.
        i. Failing to do this will result in a circular dependency error being thrown.
            1. Circumventing the error will allow the unity editor to crash upon rail-system evaluation.

iv. Events
1. Rail System Positional Events (**See Definitions**)
    a. Position
        i. This value defines the position in the rail-system that will trigger the event based on the call setting. (See 4.0.1.a.iv.1.b)
    b. Call Setting
        i. This value defines when the positional event will be called in relation to the positional event position value. (See 4.0.1.a.iv.1.a) ii.

        Trigger Before Position
            1. The event will be triggered when the railsystem's evaluated position is less than the positional event's position parameter. (See 4.0.1.a.iv.1.a)

        iii. Trigger After Position
            1. The event will be triggered when the railsystem's evaluated position is greater than the positional event's position parameter. (See 4.0.1.a.iv.1.a)

        iv. Trigger At Position
            1. The event will be triggered when the railsystem's evaluated position is within the range of the positional event's position parameter plus and minus the trigger at position tolerance.
            (See 4.0.1.a.iv.1.a and 4.0.1.a.iv.1.c)

c. Trigger at Position Tolerance
   1. This value defines the tolerance range in the positive and negative direction for the positional event's position value when using the trigger at position call setting. (See 4.0.1.a.iv.1.a and 4.0.1.a.iv.1.b)
   ii. If this value is set to 0 and the call setting is set to trigger at position, it is unlikely the event will be called. (See 4.0.1.a.iv.1.a and 4.0.1.a.iv.1.b)

d. Event
   i. This event is called when the positional event's requirements are met.

b. Primary Settings
   i. **Note**: Primary Settings are applicable to primary rail-systems only.
   ii. Position Settings
      1. Speed
         a. This value defines the speed at which the player evaluates the rail-system relative to the rail-system's duration. (4.0.1.a.ii.a)
      2. Absolute Position Lock
         a. True – The projected player's position will be set to the evaluated rail-system position.
         b. False – The project player will move towards the evaluated railsystem position.
   iii. Camera Settings
      1. Attach Camera to Dolly
         a. True – The camera's position will move towards a predefined dolly for the duration of the rail system. (See 4.0.1.b.iii.2)
         b. False – The camera will function normally.
      2. Camera Dolly Index
         a. This value defines which dolly in the On Rail Dollies list the camera will be attached to if the Attach Camera To Dolly bool is set to true. (See 4.0.1.b.iii.1 and 4.0.1.a.iii)
   iv. Player Interaction Settings
      1. Character Rep Speed
         a. This value represents the character rep speed when attached to the rail-system. (See 3.0.1.b.ii)
      2. Y Value Offset
         a. This value represents a Y value offset for the position of the player against the evaluated rail-system position.
         b. Depending on character setup the evaluated position will normally be around the character's head position by default.
      3. On Rails Drag

   a. This value represents the projected player's set rigidbody drag while interacting with the rail-system.

4. Recovery
   a. This value defines the time spent in the recovery state after the player detaches from the rail-system.
5. Key Controlled
   a. True – key input will advance the rail-system position according to the key axis. (See 4.0.1.b.iv.6)
   b. False – Key input has no effect on advancing the rail-system position.
6. Key Axis
   a. This value defines the input axis used to advance the rail-system position. (See 1.1)
   b. If the rail-system should be advanced only in one direction, either the positive or negative key of the input axis should not be assigned. (See 1.1)
7. Time Controlled
   a. This value represents whether time will advance the rail-system in the positive direction.
8. Rotation Control
   a. This value defines how player rotation is handled by the railsystem.
   b. Rotate Towards Next Waypoint
      i. The player will rotate towards the next upcoming waypoint relative to rail-system movement direction.
   c. Rotate Towards Target
      i. The player will rotate towards a predefined transform. (See 4.0.1.b.iv.9)
   d. Lock Rotation
      i. The player rotation upon attaching to the rail-system will be locked for the duration of the rail-system evaluation.
   e. Free Rotation
      i. The player will rotate normally.
9. Rotation Target
   a. This value represents the target of the player's rotation when the rotation control parameter is set to Rotate Towards Target. (See 4.0.1.b.iv.8)
   b. Tip: this value can be a rail-system dolly's gameobject's transform.
10. Rotate on Y Axis Only

        a. True – The player will only rotate on the Y axis based on the rotation control parameter. (See 4.0.1.b.iv.8)
        b. False – The player will rotate on all axes based on the rotation control parameter. (See 4.0.1.b.iv.8)
    11. Player Rotation Speed
        a. This value defines the rotation speed of the player while attached to the rail-system.
    12. Animation Trigger
        a. This value defines the animator trigger that will be called when the player attaches to the rail system.
    13. Rail Animation Position Multiplier
        a. This value represents a multiplier for the animator's "railPosition" parameter which is based on the rail-system's position.
    14. Rail Animation Position Offset
        a. This value represents an offset for the animator's "railPosition" parameter which is based on the rail-system's position.

v. Events
    1. On Rail System Latch
        a. This event is called when the player latches to the rail-system.
    2. On Rail System Unlatch
        a. This event is called when the player unlatches from the railsystem.

vi. IK Settings
    1. Feet Location Method
        a. This value defines the feet Ik target setting used when the player is attached to the rail-system. (See 0.4)
        b. None
            i. Feet Ik is disabled. (See 0.4)
        c. Project Down
            i. The feet's Ik target will be projected downwards towards the IK Layer based on the weight set by the current animation. (See 0.4 and 3.0.1.a.iv)
        d. Project Forward and Down
            i. The feet's Ik target will be projected forwards towards the IK Layer and then projected downwards from that position based on the weight set by the current animation. (See 0.4 and 3.0.1.a.iv)
        e. Lock to Target
            i. The feet's Ik target will be locked to the closest target of a predefined target set. (See 0.4 and 4.0.1.b.vi.3)
            ii. This method is not limited by distance.
    2. Hand Location Method

a. This value defines the hand Ik target setting used when the player is attached to the rail-system.

b. None

   i. Hand Ik is disabled. (See 0.4)

c. Project Forward

   i. The hand's Ik target will be projected forwards towards the IK Layer based on the weight set by the current animation. (See 0.4 and 3.0.1.a.iv)

d. Lock to Target

   i. The hand's Ik target will be locked to the closest target of a predefined target set. (See 0.4 and 4.0.1.b.vi.3) ii. This method is not limited by distance. 3. Ik Target Parent

c. Secondary Settings

   i. **Note**: Secondary Settings are applicable to secondary rail-systems only.

   ii. Camera Settings

      1. Disable Camera Collision

         a. This value represents whether camera collision will be deactivated for the evaluated duration of the rail-system.

      2. Set Camera Zoom

         a. This value defines the camera zoom value set by interacting with the rail-system.

         b. Note: This value is used to match the camera's apparent zoom position upon detachment with the current zoom value of the camera to smooth transitions.

         c. This value does nothing while the player is actually interacting with the rail-system. 2. MultiStateMovementRailCap Script

a. Rail System

   i. This value represents the rail-system that the player will be attached to, therefore making the rail-system the primary system.

b. Interaction Key

   i. This value represents the key that needs to be pressed in order for the player to attach to or detach from the rail-system while in the bounds of the trigger. c. Allow Latch

   i. This value defines whether the player will latch to the rail-system when interacting with the trigger.

d. Allow Unlatch

   i. This value defines whether the player will unlatch from the rail-system when interacting with the trigger.

e. Latch Position

   i. This value defines the rail-system position the player will latch to when interacting with this trigger.

ii. Example 1: this value should be set to 0 for a latch position at the starting waypoint.
iii. Example 2: this value should be set to the rail-system's duration for a latch position at the final waypoint. (See 4.0.1.a.ii.1)

Amendment:

1. Latch Type
    a. Set Position
        i. The player's latch position is equal to the position value.
    b. Calculated Position
        i. The player's latch position is calculated as a percentage between the position and end position parameters, relative to the distance between the markers and the player.
2. Position
    a. Either the starting position of a calculated latch or the fixed position of a set latch.
3. End Position
    a. The end window position of a calculated latch.
4. Marker One
    a. The first reference position of a calculated latch. This can be a waypoint.
5. Marker Two
    a. The second reference position of a calculated latch. This can be a waypoint.

f. Direction
i. This value defines the direction of the player on the rail-system when interacting with the trigger.
ii. 1 defines movement in the positive direction. iii. -1 defines movement in the negative direction iv. This value does NOT represent a multiplier of any kind.
v. This value affects time-controlled rail-systems. (See 4.0.1.b.iv.7)

Amendment: Remote Access

Allow Remote Access

Can the player blink to the latch position?

Remote access must also be enabled on the multiStatePlayerInput script.

Remote Access Minimum Distance

The minimum distance between the player and the rail system required to access it remotely.

Remote Access Maximum Distance

The maximum distance between the player and the rail-system required to access it remotely.

Animation Trigger

The blink animation trigger.

Speed Factor

The speed of the blink in conjunction with distance.

3. OnRailDolly Script
   a. Base Function Settings
      i. Rail System
         1. This value defines the secondary rail-system the dolly is attached to.
         2. This is NOT the rail-system that drives the dolly as defined by the On Rail Dollies parameter of the rail-system. (See 4.0.1.a.iii.1)
      ii. Offset
         1. This value represents a linear offset in the dolly's rail-system position from the driving rail-system's position.
      iii. Multiplier
         1. This value represents a multiplier in the dolly's rail-system position from the driving rail-system's position.
      iv. Speed
         1. This value represents the speed that the dolly moves towards the evaluated position.
         2. **This does NOT control the speed of the rail-system evaluation.**

   b. Rotation Settings
      i. Rotate Towards Next Waypoint
         1. True – the dolly's gameobject will rotate toward the next waypoint of the path relative to direction and the allow reverse rotation parameters. (See 4.0.3.b.iii)
         2. False – the dolly's gameobject rotation is not affected by the OnRailDolly script in any way.
            a. Dolly rotation can be controlled by a custom script if desired.
      ii. Rotation Speed Multiplier
         1. This value represents the speed at which the dolly rotates. iii.

Allow Reverse Rotation
         1. When Rotate Towards Next Target is enabled, this value defines whether the waypoint target will be calculated in both directions (true) or only in the positive direction (false).
      iv. Limit Rotation to Y
         1. When this value is enabled the dolly will only rotate on its y axis compared to it rotating on all its axes.

Section 4.1:

Simple Trigger Scripts Parameters and Guidelines

1. AnimationTrigger Script

a. This value defines the animator trigger that will be set when interacting with the trigger.

2. SimpleJumpPlate Script

a. Interrupt Rail-System

i. This value defines whether the trigger will detach the player from a rail-system.

ii. Note: this is not a clean detachment by necessity and therefore only its temporal use is recommended.

b. Jump Direction

i. This value defines the direction and extra magnitude of the player's jump. c.
Disable on Trigger

i. This value defines whether the trigger will be disabled once it has been called.

3. TriggerFader Script

a. Fader

i. This value represents the fader script located on a black UI image that covers the entire screen.

b. Disable on Trigger

i. This value defines whether the trigger will be disabled once it has been called. c.
Seconds In

i. This value defines the time in seconds it takes for the opacity of the black UI image to reach its maximum value from its minimum value.

d. Seconds Stay

i. This value defines the time in seconds the opacity of the black UI image will remain at its maximum value.

e. Seconds Out

i. This value defines the time in seconds it takes for the opacity of the black UI image to reach its minimum value from its maximum value.

f. Note: the entire length of the fade cycle can be determined by adding the Seconds In, Seconds Stay, and Seconds Out values together.

4. TriggerGroundedSubstate

a. Grounded Movement Substate

i. This value defines the grounded movement substate the player will enter when interacting with the trigger. (3.0.1.g)

b. Disable on Trigger

i. This value defines whether the trigger will be disabled once it has been called.

5. TriggerMoveTo

a. Target

i. This value represents the target transform the player will move towards. b.
Exit Distance

i. This value represents the distance between the player and the target required for the player to stop the Move To cycle. (See 4.1.5.a)

c. Disable on Trigger
   i. This value defines whether the trigger will be disabled once it has been called.

Section 4.2:
Rail System Implementation Guidelines

1. Overview
   a. This section contains the minimum amount of information required for a rail-system to function. For detailed guides and video tutorials visit: www.imitationstudios.com
   b. A primary rail-system requires the following:
      i. A gameobject with the MultiStateRailSystem Script attached.
      ii. At least two gameobjects to serve as waypoints.
         1. The starting waypoint should be the parent of all other waypoints in the system.
         2. The starting and ending waypoints should:
            a. Have the MultiStateMovementRailCap script attached to their respective gameobjects
            b. Have a collider marked as trigger to serve as the bounds for the interaction with respect to latching and or unlatching.
         3. Note: a debug line will be drawn to visualize the path between waypoints.
2. Sample Rail-System Hierarchy
   a. (Gameobject) Rail System
      i. (Gameobject) Primary System
         1. (Contains Script) MultiStateRailSystem
         2. (Gameobject) StartingWaypoint
            a. (Contains Script) MultiStateMovementRailCap
               b. (Contains Component) Trigger Collider
                  c. (Gameobject) Waypoint 1
                  d. (Gameobject) Waypoint 2
                  e. (Gameobject) End Waypoint
                  i. (Contains Script) MultiStateMovementRailCap ii. (Contains Component) Trigger Collider

Section 5:
Player Health Demo

Section 5.0:
Simple Health Script Parameters and Guidelines

This script is located on the "character" gameobject by default, but its location in the hierarchy is not important.

Purpose: The purpose of the simple health script is to provide an example of how to setup fall damage using the MSCC and of how to enable the MSCC script's ragdoll state.

1. Simple Health Script
    a. Character Controller
        i. This represents the MSCC script found of the projected player gameobject
        (Refer to the Player Unit hierarchy)
    b. Health
        i. This value represents the current health value between 0 and 100.
        ii. Health does not regenerate in this example.
        iii. When the player "dies" the MSCC script enables the ragdoll. (See 3.0.1.m) c. Fall Damage Multiplier
        i. This value defines the damage value of one second of falling. No falling damage will be applied if the total falling duration is below o.6 seconds.

Section 6: Barricades

Section 6.0:
MS Barricade Script Parameters and Guidelines:

The barricade script triggers the blink state in the z direction relative to the object containing the barricade script. A complete barricade system has three parts: the rail-systems (crouch and slide functionality), the barricade trigger (vault functionality), and the mesh with it's collider. Each of these parts are mostly independent of the other. However, when disabling rail-systems, their references on the blink trigger must be removed, and when disabling barricade triggers, it's related event must be removed from the rail-systems.

MS Barricade Script Parameters:

1. Interaction Key
    a. The key used to initiate the vault movement. The rail-systems should have the same key marked as an unlatch key.
2. Barricade Width
    a. The travel distance of the vault.
    b. When Edit Mode is enabled, this parameter will move the rail-systems out and in accordingly.

3. Animation Trigger
    a. The animation trigger of the vault action.
4. Animation Height Offset
    a. This value is sent to the animator to simulate a height change by blending three vault animations.
5. Transition Speed
    a. The overall movement speed of the vault. Warning: setting this parameter to extreme values will cause snapping.
6. Movement Delay
    a. A movement delay that occurs at the start of the vault. This is used for animation buildups.
7. Respect Ground
    a. Should a ground level check be preformed on the other side of the barricade?
8. Max Height Change
    a. The maximum height distance the player can move without failing the blink.
9. Achievement Distance
    a. The distance between the character rep and projected player in which the player will exit the blink state.
    b. Note: the character rep will decelerate has it nears the player. If this behavior is unwanted, set the achievement distance and the barricade width to higher values with edit mode disabled.
10. Recovery Time
    a. The amount of time in seconds spent in the recovery state after the vault.
11. Reset Time
    a. The time in seconds it takes for the barricade to reset after the vault. This is used to control rapid vaulting.
12. Key Black List
    a. If any of these keys are pressed along with the interaction key, the vault will not occur.
13. Rail Systems
    a. Reference to the rail-systems of the barricade. Disabled rail-systems must be removed from this list.
14. Rail System Caps
    a. Reference to the rail-system caps of the barricade. Disabled rail-systems must be removed from this list.
15. Character Controller (optional)
    a. Reference to the character controller. This will allow the player to vault the barricade immediately after unlatching from the rail-system, without the need for two individual keystrokes.
16. Edit Mode
    a. When this is enabled, the barricade width parameter will move the rail-systems out or in accordingly.