Unity Versatile Third Person-Advanced-Locomotion-Controller System Documentation

Table of Contents

        - Enter Climb State
        - Climb State
        - Exit Climb State
        - Parkour State
        - Roll State
        - Aim Lock State
        - Interact State
        - Death State
    - Player Controller Manager
        - Player Controller
        - Player UI Controller
    - Input Action Key Manager
        - Input Action Manager
        - Movement Input Handler
        - Menu Input Handler
    - Player Interfaces
        - I-Player State
        - I-Input Command
    - Camera System Manager
        - Free Roam Camera
        - Aim Lock Camera
        - Crawl Camera
    - IK Base System
        - Player Hand IK System
        - Player Foot IK System
    - Interactable Base System
        - Door
        - Elevator
        - Cover
        - Climb
        - Parkour
    - Health Base System
        - Player Health (Take Damage – Heal)
    - Stamina Base System
        - Player Stamina (Consume Stamina – Recharge Stamina)
    - UI Base System
        - Health UI
        - Stamina UI
        - Pause Menu UI
        - Controls Wizard
    - Interfaces
        - I-Health-State
        - I-Stamina-State
        - I-Interactable-State
- Enhanced Gameplay Mechanics
    - Camera System.
    - Basic Movement + Roll Mechanic.
    - Cover System (2 different action).
    - Climb System (3 Different actions up to anything).
    - Interact System (Doors & Elevators).
    - Parkour System (3 different actions up to anything).

1. Introduction

Welcome to the Unity Versatile Third Person Advanced Locomotion Controller System documentation! This asset offers a comprehensive suite of tools designed to elevate your Unity projects. Whether you're developing a game or simulation, this controller goes beyond essential functionalities, providing advanced features like the Parkour System, Climb System, Dynamic Collider on Slopes, and a customizable Interaction System to streamline your development process and enhance player experience.

2. Installation

To install the Unity Versatile Third Person Advanced Locomotion Controller System, follow these steps:

- Open your Unity project.
- Go to the Unity Asset Store.
- Search for "Unity Versatile Third Person Advanced Locomotion Controller System" and click on the asset.

- Click the "Download" button.
- Import the package into your Unity project.

Compatibility

Ensure your Unity version is compatible with this package and that Cinemachine, Input System, and Animation Rigging are updated to the latest versions.

Unity Starter Assets and Pro Builder: are used solely in the demo scene for showcasing features. They are not required in production environments and can be omitted.

3. Getting Started

After installation, follow these steps to get started with the Unity Versatile Third Person Advanced Locomotion Controller System:

- Add the Player Controller Component to the Player Game Object.
    - After adding the Player Controller Component, components like Input Action Manager, Movement Input Handler, and others will be automatically added to the Player Game Object, simplifying the setup process.
- Add the Input Action Manager Component to the Player Game Object.
- Add the Movement Input Handler Component to the Player Game Object.
- Add the Menu Input Handler Component to the Player Game Object.
- Add Capsule Collider to the Player Game Object.
- Add Rigid Body to the Player Game Object.
- Add Animator to the player game object.
- Add Rig Builder to the player game object.
- Add Player Health to the game object.
- Add Player Stamina to the player game object.
- Add Player Controller UI to the player game object.
- Add Player Hand IK System to the player game object.
- Add Player Foot IK System to the player game object.

4. Dependencies

The package relies on the following Unity packages:

- **Cinemachine**.
- **Input System**.
- **Animation Rigging**.
- **Unity Starter Assets Third Person:** for the demo scene only.
- **Pro Builder:** for the demo scene only.

5. Scene Setup

To set up the Versatile Third Person Advanced Locomotion Controller in a new scene, you need to accomplish some simple steps

1. Layers: add the following Layers:
    a. Ground
    b. Interactable
    c. Player
2. Tags: add the following tags:
    a. MainCamera
    b. Player
    c. CinemachineTarget

6. Data Management with Scriptable Object

Scriptable Objects in Unity are a powerful tool to manage character-specific data and configurations in an organized and reusable manner. This section covers the various configuration scripts provided in the package and guides you on creating their corresponding assets in the Unity Editor.

Creating Scriptable Object Assets

Each configuration script can be created as a Scriptable Object asset by right-clicking in the Project Window and navigating to the specified menu paths, such as Create > Character > [Config Type].

Character Configuration Scripts

1. Character Config Script

    Consolidates all character-related settings into a single asset.
    a) Use: Centralized character behavior management.
    b) Create: Create > Character > Character Config.

2. Rigid Body Config Script

    Holds physics-based settings for the character's Rigid-Body component.
    a) Use: Define mass, drag, and angular drag.
    b) Create: Create > Character > Rigid-Body Config.

3. Movement Config Script

    Defines movement parameters like speed and acceleration.

    a) Use: Customize character movement styles and dynamics.
    b) Create: Create > Character > Movement Config.

4. Ground Config Script

    Manages settings related to the character’s interaction with the ground.

a) Use: Ground detection and handling.
b) Create: Create > Character > Ground Config.

5. Jump Fall Config Script

Configures jump and fall behavior, including gravity and terminal velocity.

a) Use: Fine-tune jumping and falling dynamics.
b) Create: Create > Character > Jump Fall Config.

6. Aim Lock Config Script

Contains aiming-related settings for both free aim and strafe mechanics.

Settings Include:

a) Aim look weight speed.
b) Strafe walk thresholds.
c) Create: Create > Character > Aim Lock Config.

Advanced Movement and State Configurations

1. Climb Config Script

Settings for climb behavior, including speed thresholds and timeouts.

a) Create: Create > Character > Climb Config.

2. Cover Config Script

Handles settings for cover mechanics, including looking and movement while in cover.

a) Create: Create > Character > Cover Config.

3. Roll Config Script

Controls roll animation speed and completion time.

a) Create: Create > Character > Roll Config.

4. Parkour Object Config

Configures dynamic parkour actions such as vaults, small jumps, and high jumps.

a) Use: Includes animation curves and thresholds for precision control.
b) Create: Create > Character > Parkour > Traversal Object.

5. Collider Config Script

Manages collider dimensions for different states (e.g., crouching, parkour, crawling, rolling).

a) Use: Ensure proper collider adjustments during state transitions.
b) Create: Create > Character > Collider Config.

Utility Configurations

1. Ray Config Script

Defines ray casting parameters for interaction (e.g., doors, covers).

a) Create: Create > Character > Ray Config.

2. Health Config Script

Manages health settings, including maximum health and recovery rates.

a) Create: Create > Character > Health Config.

3. Stamina Config Script

Holds stamina parameters like max stamina, drain rate, and recovery speed.

a) Create: Create > Character > Stamina Config.

4. Layer Mask Config Script

Layer Mask Config Script

a) Assigns and manages layer masks for interactions and physics.
b) Create: Create > Character > Layer Mask Config.

5. Constants Config Script

Stores constant values for critical references like camera indices and ray origins.

a) Create: Create > Character > Constants Config.

6. Physic Config Script

Specifies Physic Materials for handling diverse terrain interactions.

a) Create: Create > Character > Physic Config.

7. Interaction Config Script

The Interaction Config class manages settings for interacting with objects, including interaction timeout and time for object interaction.

a) Create: Create > Character > Interaction Config.

Tips for Efficient Data Management

- **Centralization:** The Character Config Script is your central point for managing all character settings.
- **Real-Time Tweaks:** Test and adjust configurations using the Inspector window to fine-tune character behavior.
- **Reusability:** Duplicate configuration assets to use across multiple characters.

7. Third-Person Controller System Components

The following scripts and components play a crucial role in managing the player's movement, states, and actions within the Unity Versatile Third Person Advanced Locomotion Controller System. They help streamline and optimize player behavior for smooth interactions and transitions across various game mechanics.

Player Movement Context Script

The Player Movement Context class acts as the central hub for managing the player's movement settings and contexts. It encapsulates all the necessary components and configurations required to control the player's movement in the game world. This script is responsible for maintaining context across various movement states, ensuring all relevant data is properly handled and accessible.

- **Purpose:** Establishes a unified context for movement behavior across all states.
- **Role:** Coordinates various components involved in player movement, ensuring consistency and efficiency.

Player Movement Instances Script

The Player Movement Instances class handles all possible states that the player can occupy during gameplay, including idle, walking, jumping, and more. This class ensures that each movement state is instantiated once and reused, which optimizes performance and prevents unnecessary object creation during gameplay.

- **Purpose:** Manages movement state instances to improve performance by avoiding redundant object instantiation.
- **Role:** Ensures that only a single instance of each movement state exists, thereby enhancing performance and memory usage.

Player Enums Script

The Player Enums class organizes various states, commands, and interaction types into Enums, helping streamline player behavior management. By using Enums, the script allows for cleaner, more efficient transitions between states and actions.

Summary of Enums:

- **Current-Movement-State:** Defines different movement states (e.g., Idle, Walking, Jumping).
- **Current-Movement-Stage:** Indicates the stages of a movement (Start, Active, End).
- **Movement-Command-Type:** Lists the types of commands available (e.g., Move, Look, Jump).
- **Menu-Command-Type:** Defines commands related to menus (e.g., Pause).
- **Cover-Type:** Specifies types of cover (e.g., High-Cover, Low-Cover).
- **Climb-Object-Type:** Denotes climbable object types (e.g., Ladder, Rope, Wall).
- **Interaction-Type:** Categorizes interactable objects (e.g., Elevator, Door).
- **Parkour-Object-Action:** Lists parkour actions (e.g., Vault, Jump-Over-Small-Object).
- **Parkour-Object-Animation:** Defines animations related to parkour actions (e.g., Vault, Jump-Over-High-Object).
- **IK-Limbs:** Specifies limbs used during traversal (e.g., Right Hand, Left Foot).
- **Collider-Direction:** Indicates directions for collider adjustments (e.g., X, Y, Z).

Player State Manager Script

The Player State Manager class is the backbone of state management, overseeing transitions between various movement states. It maintains the current player state and ensures smooth transitions based on the player's actions and context. The state manager is initialized with predefined movement instances and is responsible for processing state transitions efficiently.

Key Features:

- **Initialization:** The manager initializes with the player movement instances and sets the initial state to idle, ensuring the player starts in a default, stable state.
- **State Transition:** It processes and facilitates state changes based on real-time conditions, such as moving from idle to walking, or from walking to jumping.
- **Error Handling:** The state manager logs errors if the player instances or the current state are null, ensuring smooth debugging and preventing unexpected issues during gameplay.

Player Movement Base State Script

The **`Player Movement Base State`** class serves as the base state for all player movement states. It handles common logic for managing player input, movement, and state transitions. By implementing the `I-Player-State` interface, it ensures consistent state management.

Key Features:

- **State Management**: Tracks and manages the current movement state, stage, and next state.
- **Events**: Defines delegates and events for state changes, allowing for responsive state management.
- **Transition Handling**: Manages transitions between different movement states to ensure smooth gameplay.
- **Camera Management**: Handles the camera system by managing mouse position and following the player.
- **Ground, Jump, and Fall Logic**: Includes methods for checking if the player is grounded, falling, or jumping, adjusting related parameters and physics accordingly.
- **Movement Handling**: Orchestrates player's movement, setting target speeds, handling input direction, adjusting speed, managing rotation, and setting Rigid-body movement.
- **State Checks**: Provides methods to check various movement states such as walking, jogging, sprinting, crouching, crawling, strafing, and rolling.
- **Collider System**: adapts the player's collider properties in real-time, ensuring optimal interaction with the environment. This system enhances realism and responsiveness, particularly in dynamically changing environments such as uneven terrains and varying slopes.
- **Animator Control:** Manages animator parameters and root motion settings.
- **Rigid-body Control**: Methods to enable or disable gravity in the Rigid-body component.
- **Input Control**: Abstract method to disable input keys, customizable in derived classes.
- **IK Control**: Methods to enable or disable the hand and foot IK systems.
- **Cover Logic**: Checks if the player is in cover by ray-casting from the player's position and detecting interactable cover objects.

- **Climbing Logic**: Manages climbing detection and direction, checking for climbable objects around the player and updating the climbing state accordingly.
- **Interaction Logic**: Checks if the player is interacting with an object using sphere casting, detecting interactable objects like doors and elevators.
- **Parkour Logic**: Checks if the player is vaulting over an object by sphere casting, detecting parkour objects.
- **Stamina Logic**: Manages the player's stamina, including methods for draining and recharging stamina, and handling stamina depletion.
- **Health Logic**: Manages health recovery over time and checks the player's health status, transitioning to the death state if health is zero.
- **Static Helper Methods**: Includes helper methods for precise rotation adjustment and movement towards objects during interactions.
- **Visual Debug Methods:** Includes methods to visualize different mechanics in the game.

Player Idle State Script

The **`Player Idle State`** class is part of a state machine that controls the player's idle state in the game. It Uses information for the player's collider (Height, Center, and Direction). It also handles the transitions to all the other states. It also Recharge the stamina of the player.

Player Walk State Script

The **`Player Walk State`** class is part of a state machine that controls the player walking state in the game. It Uses information for the player's collider (Height & Center). It also handles the transitions to all the other states. It also recharges the stamina of the player after a certain time and if the player is walking for a long time it will start to consume stamina.

Player Jog State Script

The **`Player Jog State`** class is part of a state machine that controls the player jogging state in the game. It Uses information for the player's collider (Height & Center). It also handles the transitions to all the other states. It also consumes the stamina of the player after a certain time of start jogging.

Player Sprint State Script

The **`Player Sprint State`** class is part of a state machine that controls the player's sprinting state in the game. It Uses information for the player's collider (Height & Center). It also handles the transitions to all the other states. It also consumes the stamina of the player after a certain time of start sprinting.

Player Crouch State Script

The **`Player Crouch State`** class is part of a state machine that controls the player's crouching state in the game. It Uses information for the player's collider (Height & Center). It also handles the transitions to other states. It also recharges stamina after a certain time from entering the crouch state.

Player Crawl State Script

The **`Player Crawl State`** class is part of a state machine that controls the player's crawling state in the game. It Uses information for the player's collider (Height, Center, and Direction). It also handles the transitions to other states. It also recharges stamina after a certain time from entering the crawl state.

Player Interact State Script

The **`Player Interact State`** class is part of a state machine that controls the player's interaction state in the game. This class handles situations when the player is in front of an interactable object, such as a door or elevator. It determines which hand should be used based on proximity to the handle or button of the object being interacted with, utilizing the hand IK system to ensure accurate and natural hand placement during interactions.

Player Cover State Script

The **`Player Cover State`** class is part of the state machine that controls the player's cover movement in the game. It Detects how the player will enter cover (High – Low). It uses information to determine the player's collider's height and center according to the height of the cover.

Player Roll State Script

The **`Player Roll State`** class is part of the state machine that controls the player roll state. It determines how the player will roll based on the input. It uses information to determine the player's collider's height and center while performing the roll action.

Player Aim Lock State Script

The **`Player Aim Lock State`** class is part of a state machine that controls the player's movement in the game. This specific state represents the player aiming and locking onto a target. It handles the animation, position adjustment, and state transition for when the player is aiming and locked onto a target.

A. **Add a new float variable for the walking threshold:** This variable will determine the speed at which the player will walk.
B. Initialize the walking threshold in the constructor: Use the movement-Context to initialize the walking threshold. This will allow you to set the walking speed in the game's configuration.
C. **Add a new entry in the input-Speed-Map dictionary for the walking input:** The input-Speed-Map dictionary maps input actions to animation speeds. You'll need to add a new entry for the walking input action and its corresponding animation speed.
D. **Update the Handle-Movement method to handle the walking input action:** This could involve checking if the walking input action is triggered and then setting the target-Speed to the walking threshold.
E. Update the Set-Animation-Speed method to set the animation speed for the walking input action: This will ensure that the correct animation is played when the player is walking.

Player Climb State Script

The **`Player Climb State`** class is part of the state machine that controls the player's climbing movements in the game. It determines how the player will climb, using information based on ray casts to detect whether the player has reached the top or bottom of an object. The class initializes two rays: one from the eye, cast forward to detect upcoming steps, and another from the feet, cast downward to measure the distance between the player's feet and the ground. This helps in dynamically adjusting the climbing animation and movement.

Player Enter Climb State Script

The **`Player Enter Climb State`** class is part of the state machine that controls the player's actions when beginning to climb an object. It casts two rays to determine the object's relative position to the player, ensuring accurate alignment and initiation of the climbing process.

Player Exit Climb State Script

The **`Player Exit Climb State`** class is part of the state machine that controls the player's actions when exiting a climb from the top of an object. It calculates the distance and force needed for the player to transition out of the climb smoothly, simulating the force in conjunction with the animation to provide a realistic climbing experience.

Player Parkour State Script

The **`Player Parkour State`** class is part of the state machine that controls the player's parkour state. It determines how the player performs parkour actions over objects, utilizing parkour action animations, parkour action speed, and parkour action force to simulate various parkour behaviors. It also incorporates distance, height, and animation curves to fine-tune each parkour action for a more realistic and dynamic player experience. The class integrates with the hand and foot IK systems to ensure accurate and natural movements during parkour actions.

Player Death State Script

The **`Player Death State`** class is part of the state machine that controls the player when their health reaches zero. It handles the transition to the death state and disables all player functionalities, ensuring that the player can no longer perform any actions. It uses information for the player's collider's (Center & Direction).

Player Controller Script

The **`Player Controller`** class is the central component of the Unity Versatile Third-Person-Advanced-Locomotion Controller System. It manages the player's movement and transitions between different movement states, ensuring smooth and responsive control. When you add the Player Controller script to a game object, it automatically includes all the required components:

Key Features:

1. **Animator**: Controls the animations of the player character based on input and state.

2. **Capsule Collider**: Used for collision detection, creating an invisible capsule shape around the player that interacts with other colliders.
3. **Rigid-body**: Allows the player to be affected by physics, including forces and collisions.
4. **Player Input**: A built-in Unity component that manages player input, translating it into actions that the Player Controller can understand and respond to.
5. **Rig Builder**: Part of Unity's Animation Rigging package, responsible for managing and building the rig configuration for a character, enabling complex animations such as IK (Inverse Kinematics).

Additional Features:

- **Initialization**: Sets up references and initializes the player movement context and state manager.
- **State Updates**: Continuously updates the player's state based on input for dynamic gameplay.
- **Component Management**: Manages various components needed for player control, including input handlers, camera system, health and stamina systems, IK systems, and animation rigging.
- **Gizmos for Debugging**: Draws Gizmos in the editor for visual debugging of ground, climb, and parkour states.
- **Animation and IK**: Integrates animation rigging and IK systems for realistic player movements.
- **Character Configuration**: Utilizes a character configuration to manage settings for various states.

Player UI Controller Script

The **`Player UI Controller`** class manages the UI elements related to the player's health and stamina. It updates the health and stamina bars and subscribes to relevant events to ensure the UI reflects the player's current state accurately.

Key Features:

- **Health Management**: Updates the health UI when the player's health changes, using the Player-Health component.
- **Stamina Management**: Updates the stamina UI when the player's stamina changes, using the Player-Stamina component.
- **Event Subscription**: Subscribes to the On-Health-Changed and On-Stamina-Changed events to receive updates on health and stamina changes.
- **Component Initialization**: Initializes references to the player's health and stamina components and their corresponding UI elements.
- **UI Updates**: Methods to update the health and stamina UI elements based on current and maximum values.

Input Action Manager Script

The **`Input Action Manager`** class is responsible for handling input from the player and storing the values for the player's movements and actions. Utilizing Unity's new Input System, this class maps

movement and menu commands to their respective handlers, ensuring efficient input management. It initializes two main dictionaries: one for movement commands and one for menu commands, associating each command type with its corresponding input handler.

Movement Input Handler Script

The **`Movement Input Handler`** class handles the player's movement input. It implements the `I-Input-Command` interface and provides methods for executing various movement commands. This class maps different types of player movement actions (such as walking, jumping, jogging, and more) to their corresponding input commands, ensuring that player inputs are accurately captured and processed in real-time.

Menu Input Handler Script

The **`Menu Input Handler`** class handles the player's menu input. It implements the `I-Input-Command` interface and provides methods for executing menu commands and managing the pause input. This class captures and processes player inputs related to menu interactions, ensuring smooth and responsive menu navigation and controls.

I Player State Script

The **`I-Player-State`** interface defines the methods required for managing player states. It mandates the implementation of methods for entering, updating, and exiting a state, ensuring consistent state management and smooth transitions between different player behaviors.

I Input Command Script

The **`I-Input-Command`** interface defines a method for executing commands based on player input. It is designed to handle various player commands by associating input values with specific command types, enabling responsive and intuitive player control.

Camera System Manager Script

The **`Camera System Manager`** class handles the camera movement and virtual camera activation for the player. It manages the position of the mouse cursor in the game and can be used to add more camera systems to the game, ensuring a dynamic and responsive camera setup.

Key Features:

- **Layer Masks**: Configurable layer masks for aim point collision detection.
- **Camera Activation**: Methods to activate specific virtual cameras and deactivate others.
- **Mouse Position Handling**: Tracks and updates the aim point based on mouse cursor position.
- **Camera Rotation Management**: Updates camera rotation based on player input, with clamping for vertical and horizontal angles.
- **Flexible Integration**: Easily extendable to add more camera systems for varied gameplay scenarios.

Player IK Base System Script

The **`Player IK Base System`** class manages the behavior of the player's IK (Inverse Kinematics) system, enabling realistic interactions with objects in the game world. It allows the player to place their hands and feet on objects dynamically, supporting various actions such as interacting with doors and elevators, and setting IK targets for parkour actions using ray-cast data.

Key Features:

- **Hand IK Targeting**: Sets the IK targets for the player's hands, enabling precise hand placement during interactions.
- **Foot IK Targeting**: Sets the IK targets for the player's feet, ensuring accurate foot placement on different terrains.
- **Parkour Limb Adjustment**: Adjusts limb placement on parkour objects by calculating positions and rotations based on object size and orientation.

Player Hand IK System Script

The **`Player Hand IK System`** class manages the behavior of the player's hand IK (Inverse Kinematics) system. It allows the player to interact with objects in the game world by placing their hands accurately on these objects. Utilizing the IK system from the Animator, this class supports realistic hand placements for various interactions, such as with doors and elevators, and adjusts limb placement for parkour actions.

Key Features:

- **Hand IK Targets**: Sets IK targets for the player’s right and left hands, ensuring accurate hand placement.
- **Parkour Limb Adjustment**: Adjusts hand positions based on parkour objects’ size and orientation.
- **IK Activation**: Methods to enable or disable the IK system.
- **Reach Detection**: Checks if the hand has reached the target position for precise interactions.

Player Foot IK System Script

The **`Player Foot IK System`** class manages the behavior of the player's foot IK (Inverse Kinematics) system. It ensures that the player’s feet are placed accurately on different terrains and parkour objects, providing realistic and dynamic foot movements. This class adjusts the foot positions and rotations based on the environment, enhancing the immersion and responsiveness of the player’s actions.

Key Features:

- **Foot IK Targets**: Sets IK targets for the player’s right and left feet, ensuring accurate foot placement.
- **Terrain Adjustment**: Adjusts foot positions based on terrain height and angle using ray-cast data.

- **Parkour Limb Adjustment**: Adjusts foot positions based on the size and orientation of parkour objects.
- **IK Activation**: Methods to enable or disable the IK system for both feet.
- **Body Positioning:** Adjusts the player's body position based on ground detection to maintain realistic posture.

Interactable Base System Script

The **`Interactable Base System`** class provides a foundational framework for all interactable systems in the game. It implements the **`I-Interactable`** interface, requiring an Interact method. This abstract class also includes methods for obtaining the interaction point and the size of the object, which must be implemented in any subclass.

Key Features:

- **Interact Method**: Abstract method to manage the object's behavior when interacted with.
- **Interaction Point**: Abstract method to get the interaction point, used for setting hand IK targets during interactions.
- **Object Size**: Abstract method to get the size of the object, providing information on its dimensions.

Door Script

The **`Door`** class manages the behavior of a door, allowing it to be opened and closed. It inherits from **`Interactable-Base-System`** and implements the **`I-Interactable`** interface. This class defines the interaction logic for doors, managing their rotations, open and close states, and interaction points.

Key Features:

- **Interaction Type**: Specifies the type of interaction for the door.
- **Interaction Logic**: Handles the opening and closing of doors when interacted with.
- **Interaction Point**: Retrieves the interaction point of the door, typically the handle.
- **Object Size**: Provides the size of the door object, although in this implementation it returns zero.
- **Open and Close Mechanics:** Manages the door's rotation to open and close it smoothly, including a delay before closing.

Elevator Script

The **`Elevator`** class manages the movement of an elevator, allowing it to alternately move up or down a specified distance when interacted with. The direction of the elevator's movement changes after each interaction. This class inherits from **`Interactable-Base-System`** and implements the **`I-Interactable`** interface.

Key Features:

- **Interaction Type**: Specifies the type of interaction for the elevator.

- **Interaction Logic**: Starts the elevator's movement when interacted with if it is not already moving.
- **Interaction Point**: Retrieves the interaction point of the elevator, typically the position of the elevator button.
- **Object Size**: Provides the size of the elevator object, although in this implementation it returns zero.
- **Movement Management**: Uses a coroutine to move the elevator up or down a specified distance, alternating direction after each interaction.
- **Parenting Rigid-bodies:** Sets the elevator as the parent of any Rigid-body that enters its trigger collider, ensuring the Rigid-body moves with the elevator and removes the parent when the Rigid-body exits.

Climb Object Script

The **`Climb Object`** class represents a climbable object in the game. It inherits from **`Interactable-Base-System`** and implements the **`I-Interactable`** interface. This class defines the interaction logic for climbable objects, such as ladders or walls, and manages the interaction points and object size for accurate player interactions.

Key Features:

- **Climb Object Type**: Specifies the type of climbable object (e.g., Ladder, Rope, Wall).
- **Interaction Logic**: Manages interaction with climbable objects.
- **Interaction Point**: Retrieves the interaction point of the object, used for setting hand IK targets.
- **Object Size**: Retrieves the size of the object, aiding in accurate interaction and animation.

Cover Object Script

The **`Cover Object`** class represents a cover object in the game. It inherits from Interactable Base System and implements the **`I-Interactable`** interface. This class defines the interaction logic for cover objects and manages the interaction points and object size for accurate player interactions.

Key Features:

- **Cover Type**: Specifies the type of cover the object provides (e.g., High Cover, Low Cover).
- **Interaction Logic**: Manages interaction with cover objects, though they typically do not have interaction logic like doors or elevators.
- **Interaction Point**: Retrieves the interaction point of the object, used for setting hand IK targets.
- **Object Size**: Retrieves the size of the object, aiding in accurate interaction and animation.

Parkour Object Script

The **`Parkour Object`** class represents a parkour object in the game. It inherits from Interactable Base System and implements the `I-Interactable` interface. This class defines the interaction logic for parkour objects and manages the interaction points and object size for accurate player interactions.

Key Features:

- **Parkour Action Type**: Specifies the type of parkour action (e.g., Vault, Jump-Over-Small-Object, Jump-Over-High-Object).
- **Interaction Logic**: Parkour objects typically do not have interaction logic like doors or elevators, but this class provides the structure for potential interactions.
- **Interaction Point**: Retrieves the interaction point of the parkour object, used for setting hand IK targets.
- **Object Size**: Retrieves the size of the parkour object, aiding in accurate interaction and animation.
- **Closest Point Calculation**: Returns the closest point on the object to the player, helping in precise positioning of the player's hands and feet.

Health Base System Script

The **`Health Base System`** class provides a foundation for all health systems in the game. It manages the health and destruction states of objects, implementing core functionalities such as taking damage, healing, and triggering events when health changes or objects are destroyed.

Key Features:

- **Health Management**: Abstract properties for getting and setting maximum and current health.
- **Initialization**: Abstract method to initialize the health system with a specified maximum health value.
- **Damage and Healing**: Abstract methods for applying damage and healing to the object.
- **Events**: Delegates and events for health changes and object destruction, allowing other components to respond to these changes.
- **Event Invocation**: Protected methods to invoke health change and destruction events, ensuring consistent event handling.

Player Health Script

The **`Player Health`** class manages the player's health, allowing the player to take damage and heal. It inherits from **`Health-Base-System`** and implements the **`I-Health-State`** interface. This class defines the core functionalities required to manage the player’s health, including initialization, taking damage, healing, and triggering health-related events.

Key Features:

- **Health Configuration**: Uses a Health-Config object to set health-related parameters.
- **Health Management**: Abstract properties for getting and setting maximum and current health.
- **Initialization**: Method to initialize the player's health at the start of the game.
- **Damage and Healing**: Methods for applying damage and healing to the player, with corresponding event triggers.
- **Destroyed State**: Manages a flag indicating whether the player is destroyed and triggers events when the player's health reaches zero.

Stamina Base System Script

The **`Stamina Base System`** class serves as a foundation for all stamina systems in the game. It manages the stamina levels of characters, allowing for the consumption and recovery of stamina. This abstract class also triggers events when stamina changes or is depleted, ensuring responsive and interactive stamina management.

Key Features:

- **Stamina Management**: Abstract properties for getting and setting maximum and current stamina.
- **Initialization**: Abstract method to initialize the stamina system with a specified maximum stamina value.
- **Stamina Usage and Recovery**: Abstract methods for using and recovering stamina.
- **Events**: Delegates and events for stamina changes and depletion, allowing other components to respond to these changes.
- **Event Invocation**: Protected methods to invoke stamina change and depletion events, ensuring consistent event handling.

Player Stamina Script

The **`Player Stamina`** class manages the player's stamina, allowing it to be used and recovered. It inherits from **`Stamina-Base-System`** and implements the **`I-Stamina-State`** interface. This class defines the core functionalities required to manage the player's stamina, including initialization, consumption, recovery, and triggering stamina-related events.

Key Features:

- **Stamina Configuration**: Uses a Stamina-Config object to set stamina-related parameters.
- **Stamina Management**: Abstract properties for getting and setting maximum and current stamina.
- **Initialization**: Method to initialize the player's stamina at the start of the game.
- **Stamina Usage and Recovery**: Methods for using and recovering stamina, with corresponding event triggers.
- **Drained State**: Manages a flag indicating whether the player's stamina is drained and triggers events when the stamina is depleted.

Player UI Base System Script

The **`Player UI Base System`** class provides a foundational framework for all player UI systems. This abstract class defines methods for showing and hiding UI elements, as well as updating UI components based on player data. Subclasses must implement these methods to ensure consistent and interactive UI behavior.

Key Features:

- **Visibility Management**: Abstract methods for showing and hiding UI elements.

- **UI Updates**: Abstract method for calculating and updating the UI element, ensuring real-time feedback based on player data.
- **Visibility Flag**: Protected flag indicating whether the UI is currently visible.

Player Health UI Script

The **`Player Health UI`** class manages the player's health UI, specifically updating the health bar with the player's current and maximum health values. It inherits from **`Player-UI-Base-System`** and defines methods for showing and hiding the health UI, as well as updating the health bar based on the player's health data.

Key Features:

- **Health Management**: Updates the health bar to reflect the player's current and maximum health.
- **Visibility Control**: Methods for showing and hiding the health UI.
- **UI Updates**: Calculates and updates the health bar's fill amount to represent the player's health status accurately.
- **Health Data**: Utilizes serialized fields to store the player's current and maximum health values.

Player Stamina UI Script

The **`Player Stamina UI`** class manages the player's stamina UI, specifically updating the stamina bar with the player's current and maximum stamina values. It inherits from **`Player-UI-Base-System`** and defines methods for showing and hiding the stamina UI, as well as updating the stamina bar based on the player's stamina data.

Key Features:

- **Stamina Management**: Updates the stamina bar to reflect the player's current and maximum stamina.
- **Visibility Control**: Methods for showing and hiding the stamina UI.
- **UI Updates**: Calculates and updates the stamina bar's fill amount to represent the player's stamina status accurately.
- **Stamina Data**: Utilizes serialized fields to store the player's current and maximum stamina values.

Pause Menu System Script

The **Pause Menu System** class manages the pause menu system, including showing and hiding the pause menu and controls wizard. It handles pausing and resuming the game, ensuring smooth transitions between gameplay and pause states. This class inherits from Player UI Base System and implements the core functionalities for managing the pause menu UI.

Key Features:

- **Pause Management**: Methods for showing and hiding the pause menu, pausing and resuming the game.
- **Controls Wizard**: Methods for showing and hiding the controls wizard UI element.
- **UI Updates**: Overridden method for updating the UI element, though not utilized in this implementation.
- **Toggle Functionality**: Toggles the pause menu based on the pause input, ensuring responsive and intuitive menu management.
- **Game Exit:** Static method to exit the game, with specific handling for the Unity editor.

I Health State Interface

The **`I-Health-State`** interface defines the properties and methods required for a health system in the game. It outlines the essential health-related functionalities that any class implementing this interface must provide, ensuring a consistent approach to managing health across different objects.

Key Features:

- **Max Health**: Property for getting and setting the maximum health of the object.
- **Current Health**: Property for getting and setting the current health of the object.
- **Destroyed State**: Property for checking if the object is destroyed.
- **Take Damage**: Method for applying damage to the object, reducing its health.
- **Heal**: Method for healing the object, increasing its health.

I Stamina State Interface

The **`I-Stamina-State`** interface defines the properties and methods required for a stamina system in the game. It outlines the essential functionalities that any class implementing this interface must provide, ensuring a consistent approach to managing stamina across different objects.

Key Features:

- **Max Stamina**: Property for getting and setting the maximum stamina of the object.
- **Current Stamina**: Property for getting and setting the current stamina of the object.
- **Drained State**: Property for checking if the object is drained of stamina.
- **Use Stamina**: Method for consuming a specified amount of stamina.
- **Recover Stamina**: Method for recovering a specified amount of stamina.

I Interactable Interface

The **`I-Interactable`** interface is used to define interactions with objects in the game. It requires implementing methods for interacting with objects, getting the interaction point, and obtaining the object's size. This interface ensures a consistent approach to handling interactive elements within the game.

Key Features:

- **Interact**: Method for defining the interaction logic when the object is engaged.
- **Get Interaction Point**: Method for retrieving the interaction point of the object, typically used for setting IK targets.
- **Get Object Size**: Method for obtaining the size of the object, aiding in accurate interaction and animation placement.

8. Enhanced Gameplay Mechanics

1. Dynamic Camera System
    a. Provides a flexible and adaptable camera setup that enhances gameplay by following the player and adjusting views based on the action.
2. Basic Movement + Roll Mechanic + Aim Lock Mechanic
    a. Includes foundational player movements such as walking, crouching, crawling, jogging, sprinting, and jumping, along with aim lock mechanic and a roll mechanic for evasion and quick maneuvering.
3. Cover System
    a. Allows players to take cover behind objects, enhancing strategy and realism in gameplay by providing protective and tactical advantages.
4. Climb System
    a. Enables players to interact with and climb various objects, creating more dynamic and vertical gameplay opportunities.
5. Parkour System
    a. Supports advanced movement techniques like vaulting and jumping over obstacles, making navigation more fluid and responsive.
6. Interaction System (Doors & Elevators)
    a. Provides mechanisms for players to interact with environmental objects like doors and elevators, adding depth to the gameplay and enabling complex environment navigation.
7. Dedicated Scripts for Interactable Objects
    a. Each type of interactable object, such as cover, climb, parkour, doors, and elevators, has its own dedicated script. This ensures specific and optimized behavior for each type of interaction, as defined in the Interactable Base System documentation.
    b. The Interaction System can be extended with new actions to suit specific gameplay needs in every system of the created ones
8. Unified Interaction Script
    a. While each interactable object has a dedicated script for specific behaviors, the Interactable Base System provides a foundational framework for all interactable systems. All objects can interact with the player using this unified approach, eliminating the need for tags and simplifying the setup process.
9. Health & Stamina Systems
    a. Manages player health and stamina, ensuring that players need to balance their actions and recovery times. Stamina is consumed during various actions and recharges over time, while health can be recovered gradually or through specific events.
10. Dynamic Collider System

The Dynamic Collider System is an advanced mechanism designed to adapt the player's collider properties in real-time, ensuring optimal interaction with the environment.

Key Features:

a. **Automatic Adjustment**: Modifies the collider's height, center, and direction based on the player's current state and environmental context.
b. **Slope Adaptation**: Utilizes the Adjust-Collider-For-Slope method to dynamically alter collider properties according to the slope angle.
c. **Environmental Adaptation**: Through the Adjust-Collider-For-Environment method, the system adjusts collider properties based on calculated slope Normals.
d. **State-Based Configuration**: Employs the Set-Collider-State-Properties method to apply specific collider settings corresponding to different player states.
e. **Physics Integration**: Integrates with Unity's physics engine to dynamically adjust physic materials.

Benefits:

f. **Enhanced Realism**: Continuous adaptation of collider properties ensures player movements are fluid and realistic across various terrains.
g. **Improved Performance**: Automatic adjustments minimize the need for manual configuration, saving development time and enhancing game performance.
h. **Versatility**: The system's ability to adapt to different states and environments makes it a versatile tool for developers.

11. IK Foot & Hand Systems

Hand IK System:

a. **Dynamic Adjustment**: Adjusts hand positions in real-time to ensure accurate interaction with objects and environments.
b. **Smooth Transitions**: Enhances the fluidity of hand movements, making actions like grabbing and climbing appear natural.

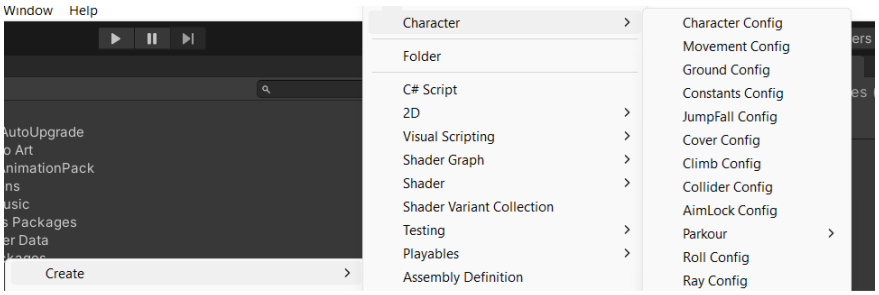
Foot IK System:

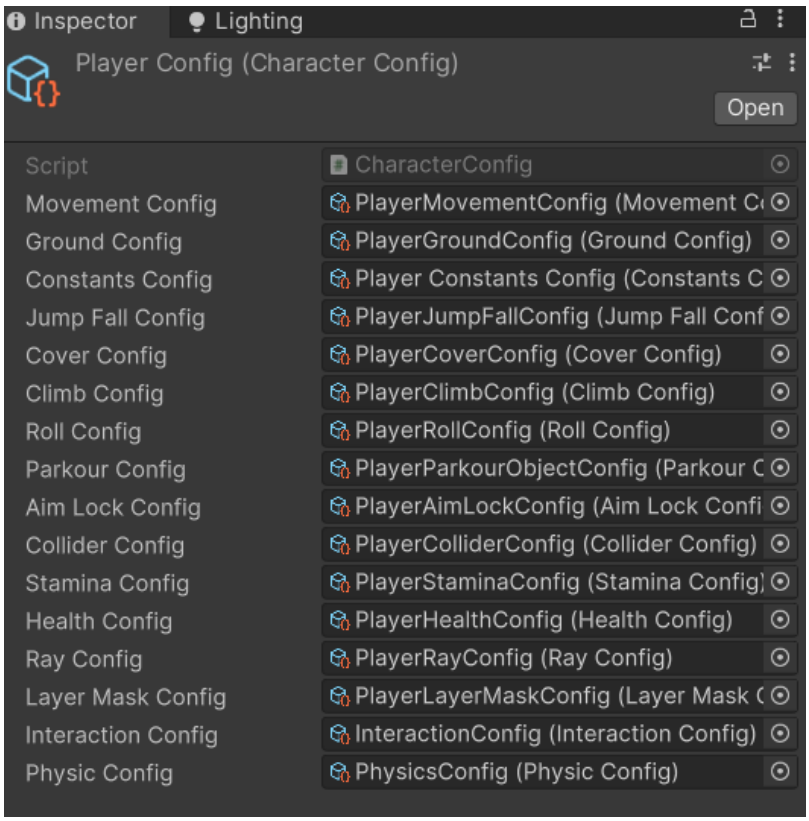
c. **Environmental Interaction**: Dynamically adjusts foot positions to match uneven terrain and slopes, providing realistic contact points.
d. **Improved Animation Quality**: Ensures smooth and believable walking, and running animations by accurately placing feet on the ground.

9. Usage Examples

Character Configuration Setup

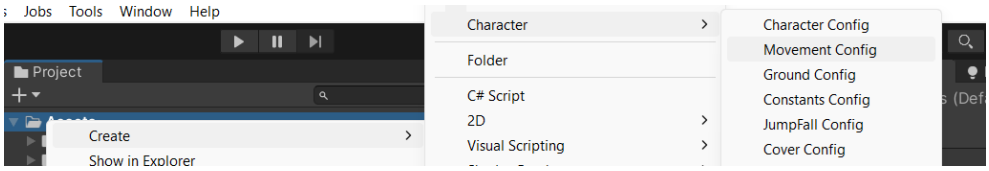
To setup the character configuration, right click in the project window and selecting Create > Character > Character Config

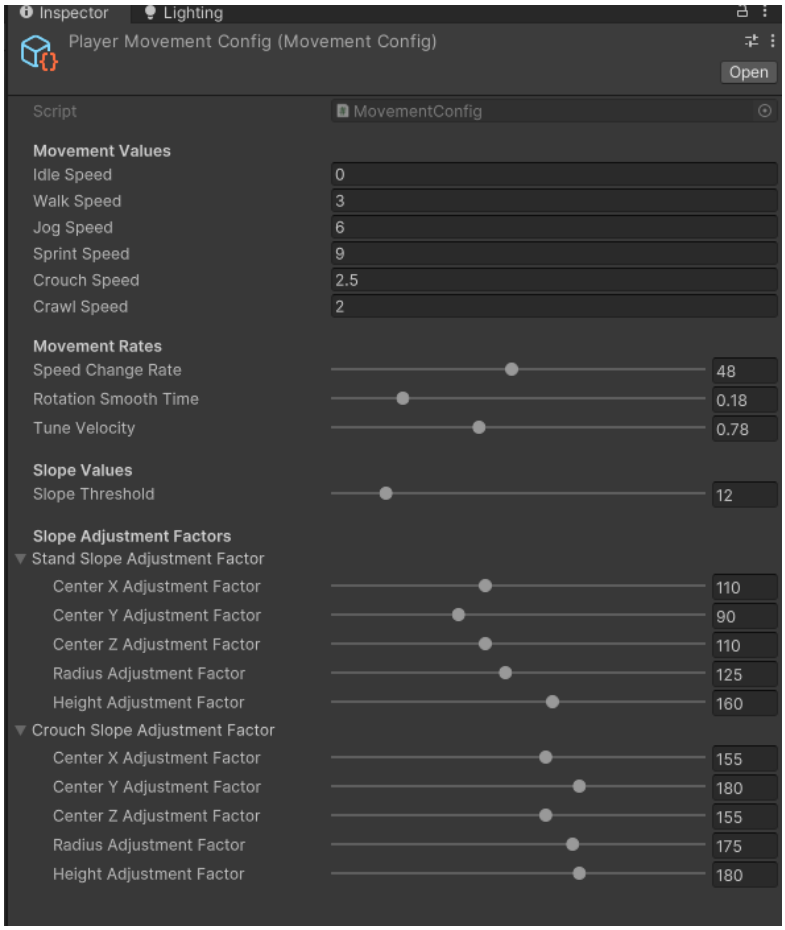




Movement Configuration Setup

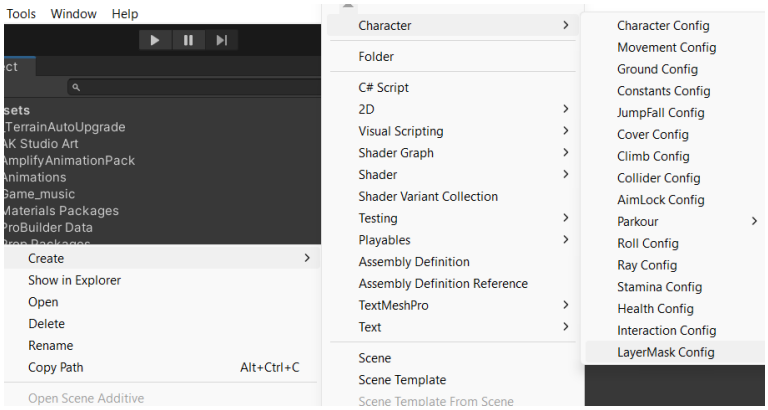
To setup the movement configuration, right click in the project window and selecting Create > Character > Movement Config

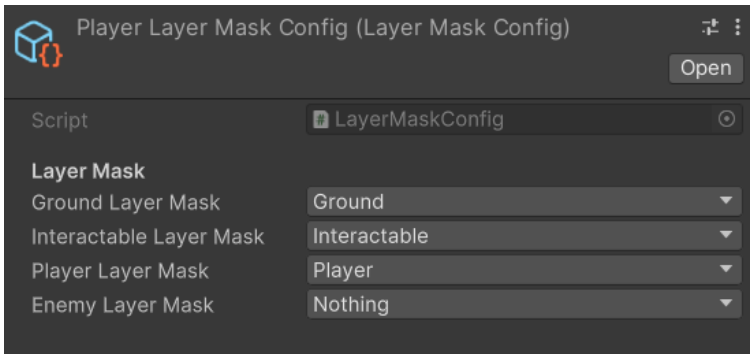




Layer Mask Configuration Setup

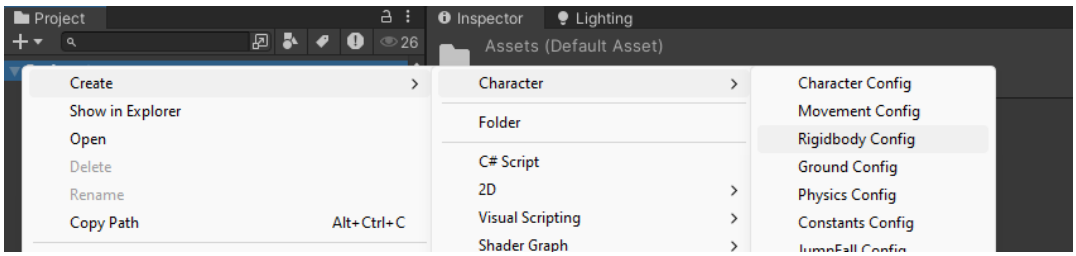
To setup the layer mask configuration, right click in the project window and selecting Create > Character > Layer Mask Config

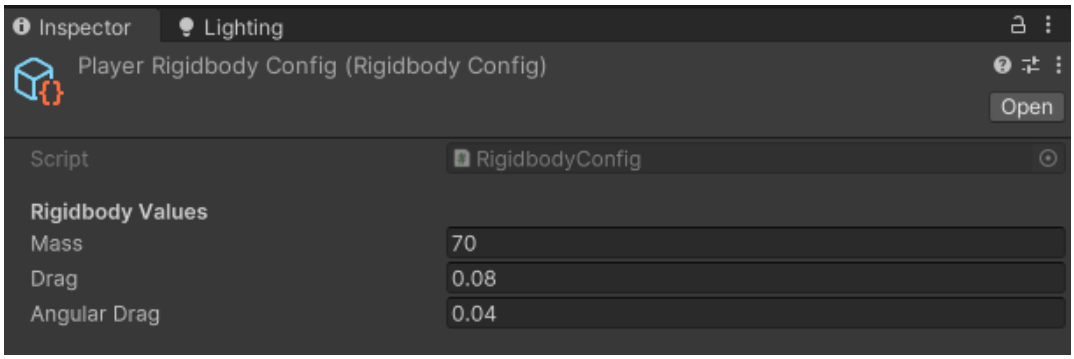




Player Rigid Body Configuration Setup

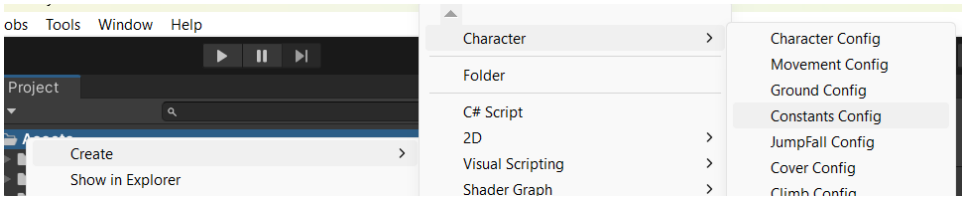
To setup the layer mask configuration, right click in the project window and selecting Create > Character > Rigid-Body Config.

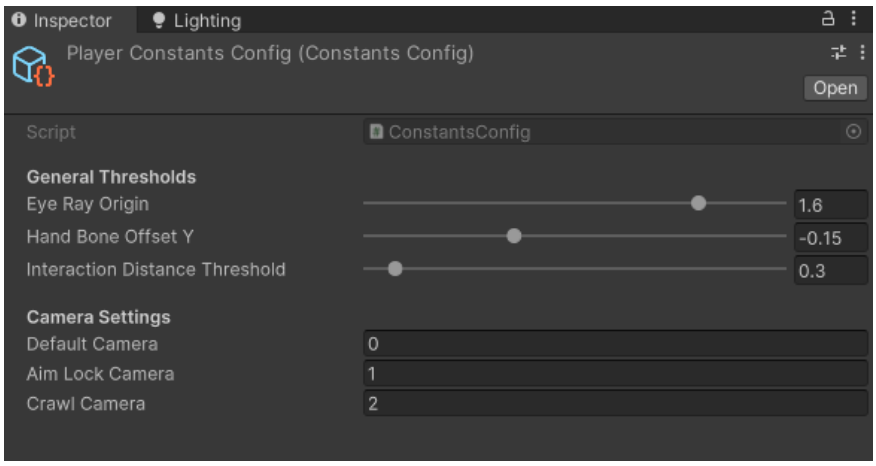




Constants Configuration Setup

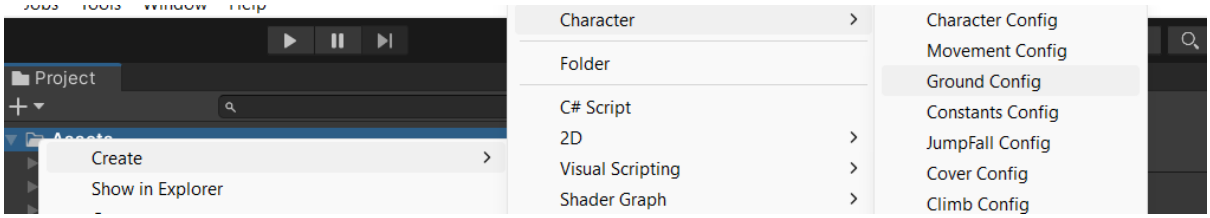
To setup the constants configuration, right click in the project window and selecting Create > Character > Constants Config

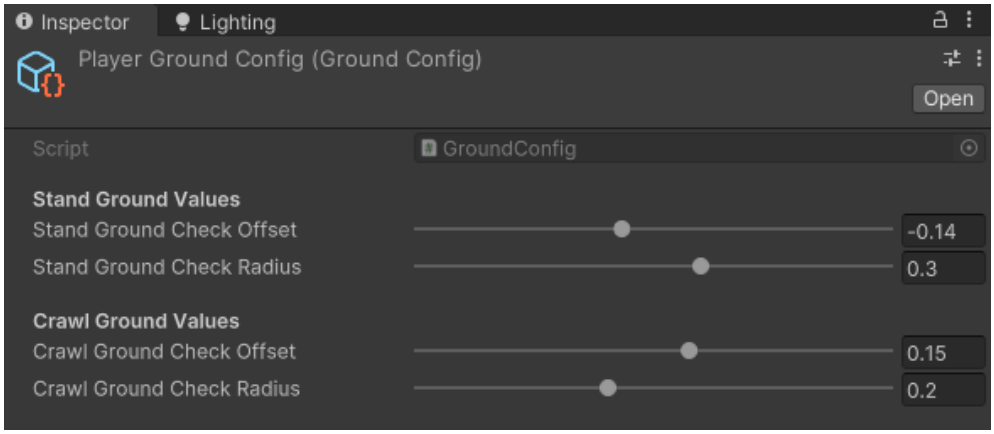




Ground Configuration Setup

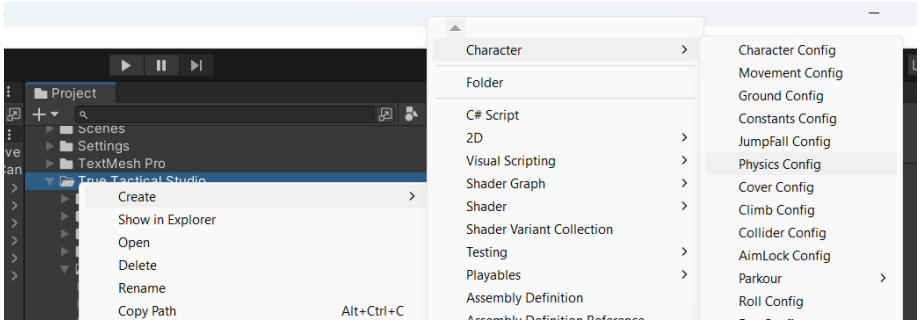
To setup the ground configuration, right click in the project window and selecting Create > Character > Ground Config

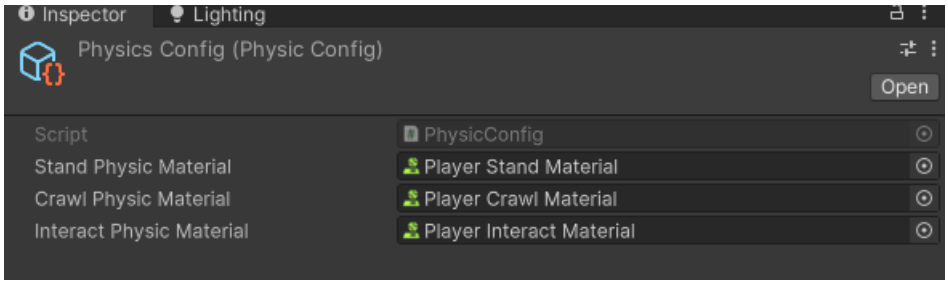




Physic Configuration Setup

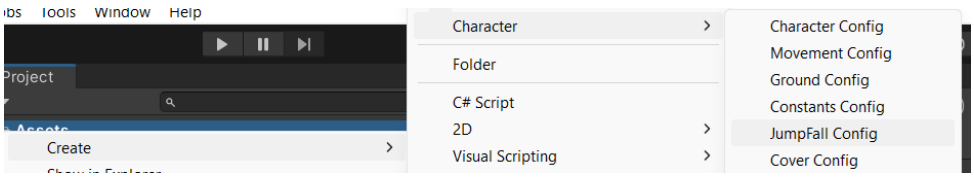
To setup the physic configuration, right click in the project window and selecting Create > Character > Physic Config

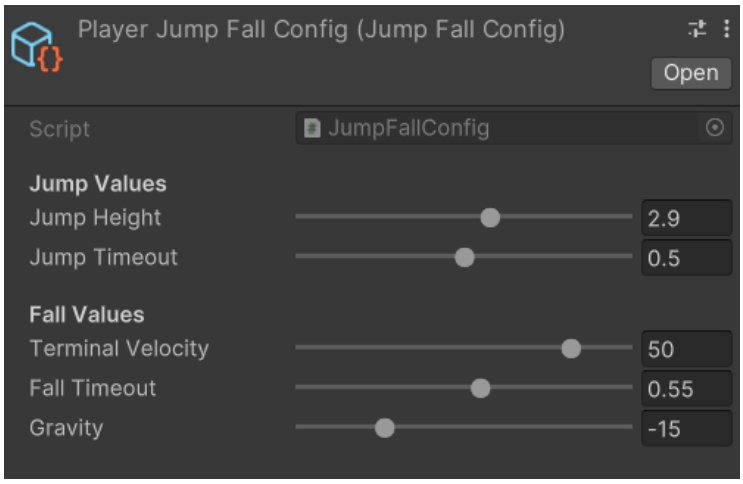




Jump - Fall Configuration Setup

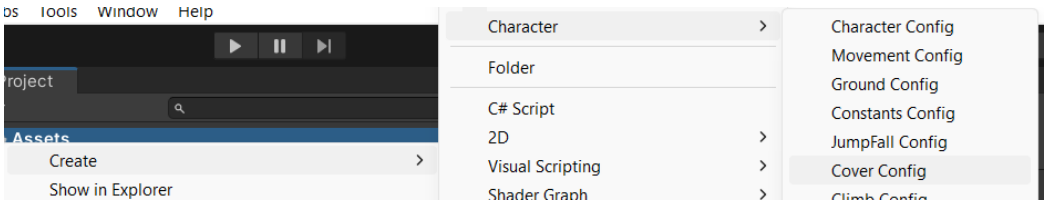
To setup the jump-fall configuration, right click in the project window and selecting Create > Character > Jump-Fall Config





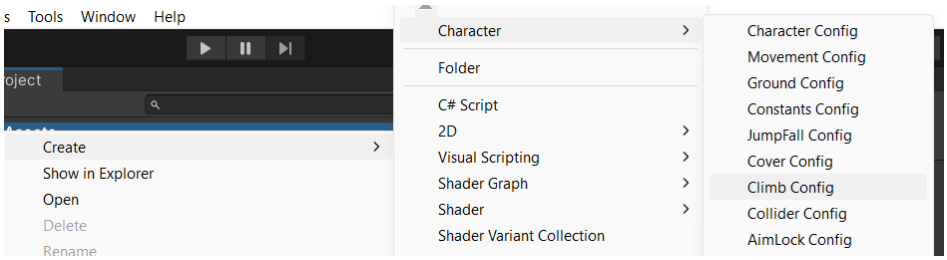
Cover Configuration Setup

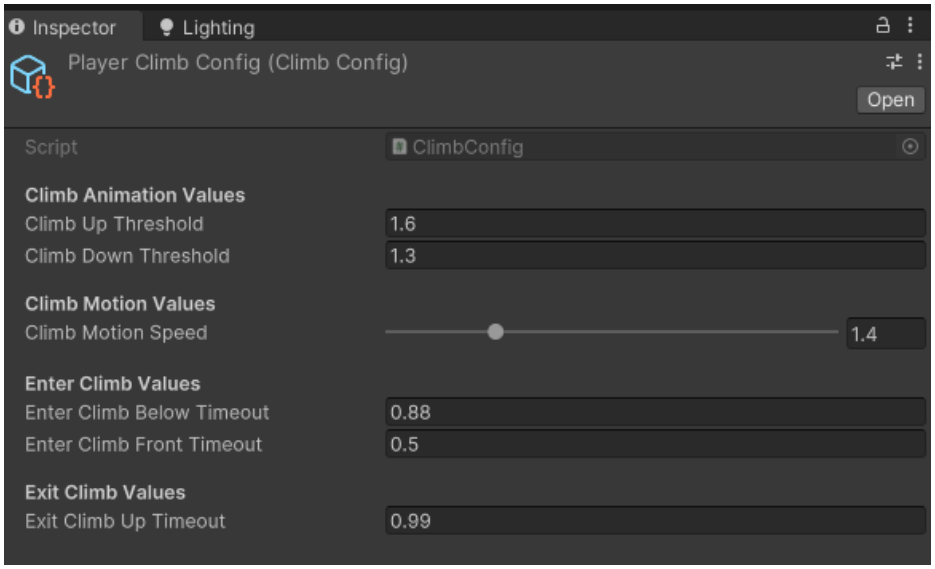
To setup the cover configuration, right click in the project window and selecting Create > Character > Cover Config



Climb Configuration Setup

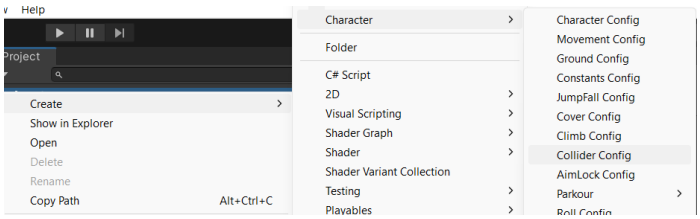
To setup the climb configuration, right click in the project window and selecting Create > Character > Climb Config

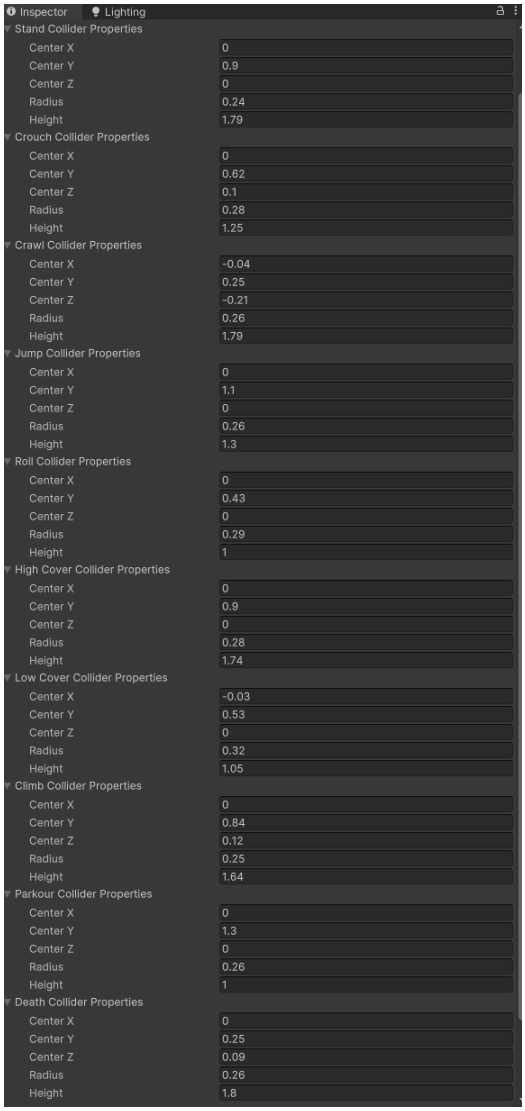




Collider Configuration Setup

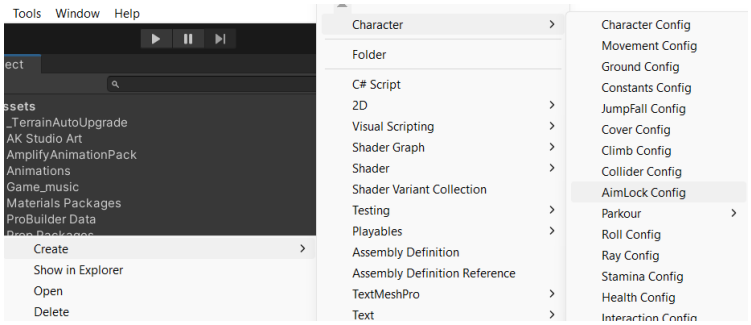
To setup the collider configuration, right click in the project window and selecting Create > Character > Collider Config

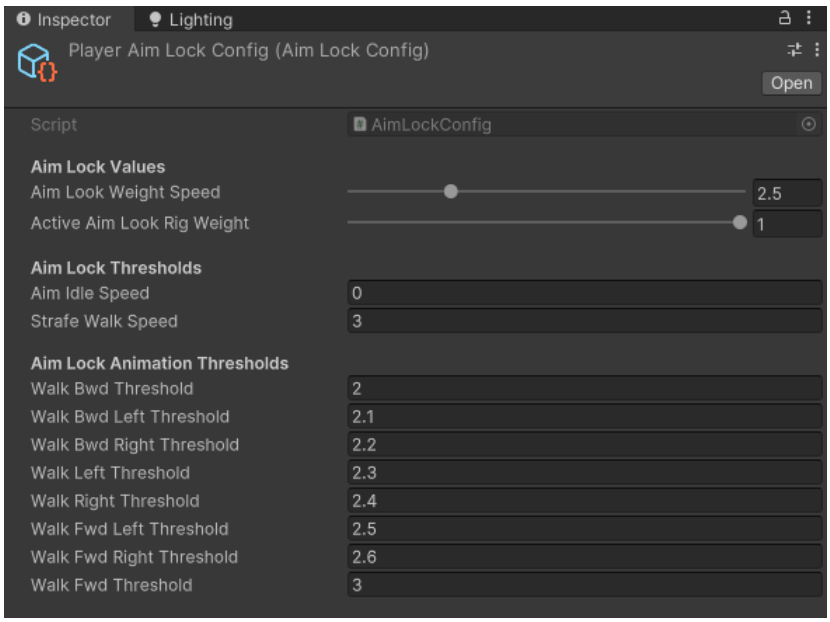




Aim Lock Configuration Setup

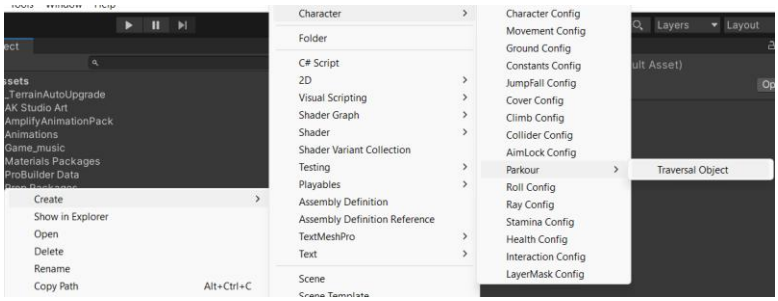
To setup the aim lock configuration, right click in the project window and selecting Create > Character > Aim Lock Config

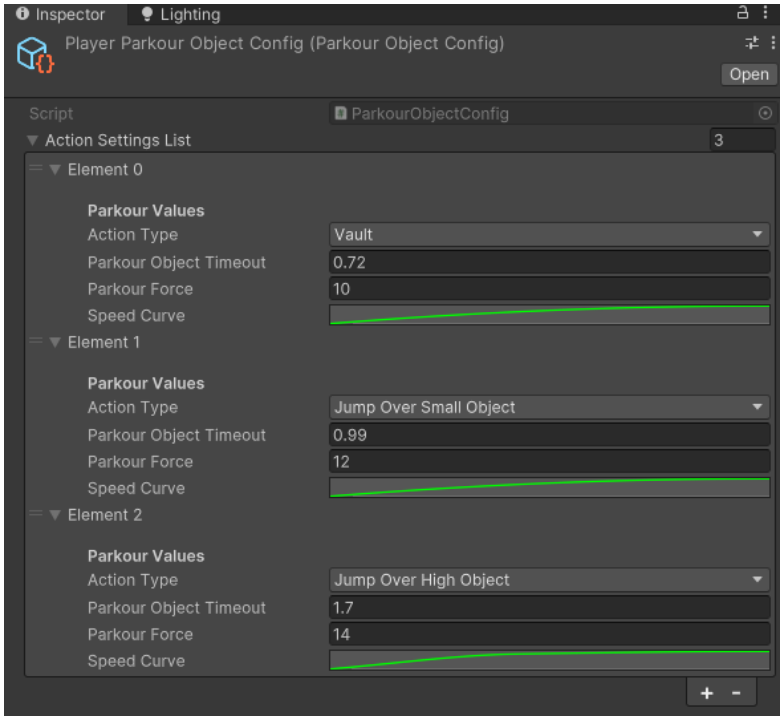




Parkour Configuration Setup

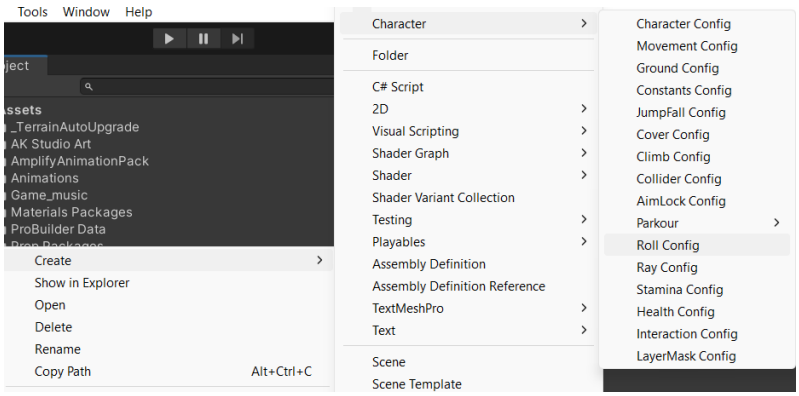
To setup the Parkour configuration, right click in the project window and selecting Create > Character > Parkour > Traversal Object

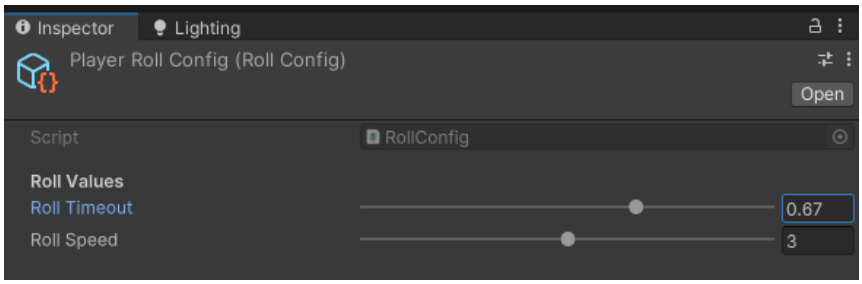




Roll Configuration Setup

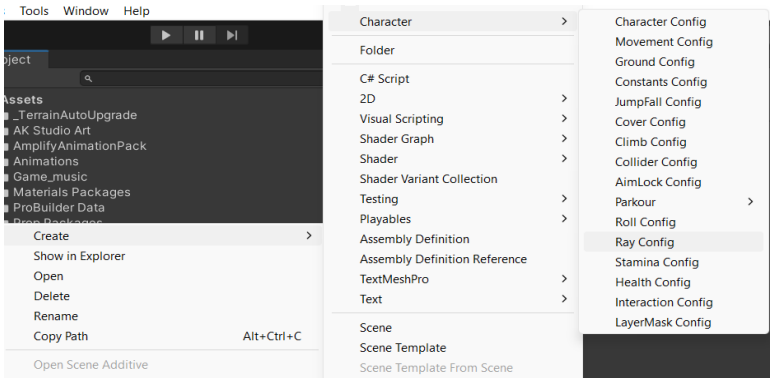
To setup the roll configuration, right click in the project window and selecting Create > Character > Roll Config

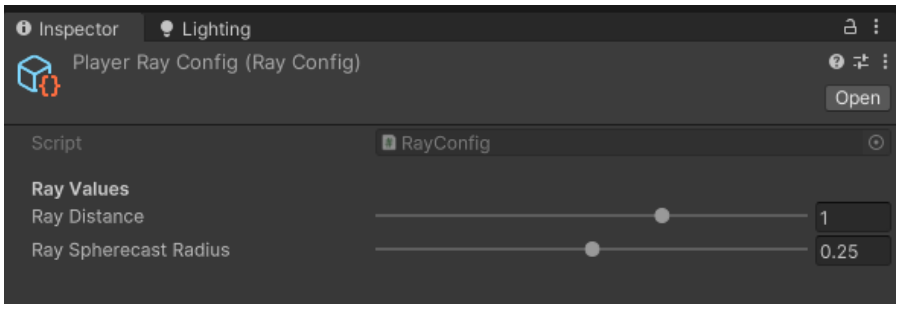




Ray Configuration Setup

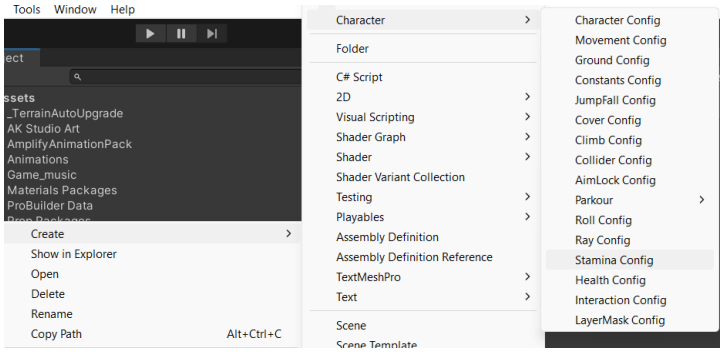
To setup the ray configuration, right click in the project window and selecting Create > Character > Ray Config

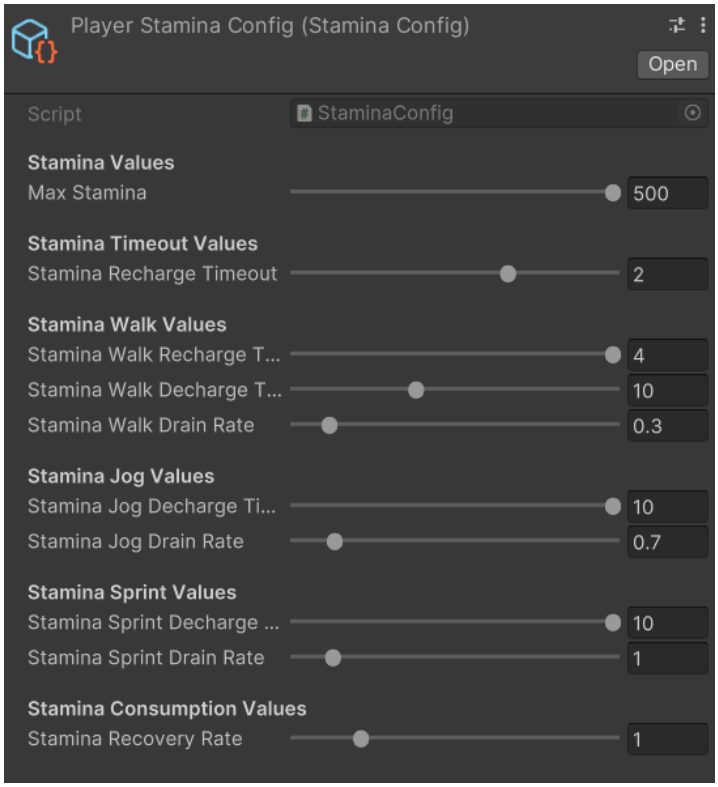




Stamina Configuration Setup

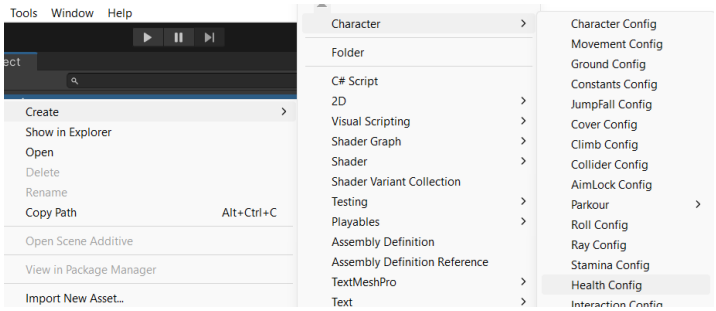
To setup the stamina configuration, right click in the project window and selecting Create > Character > Stamina Config

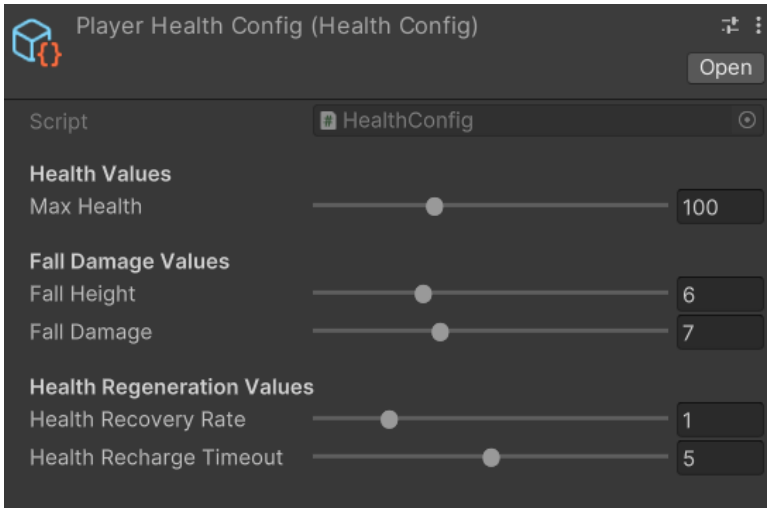




Health Configuration Setup

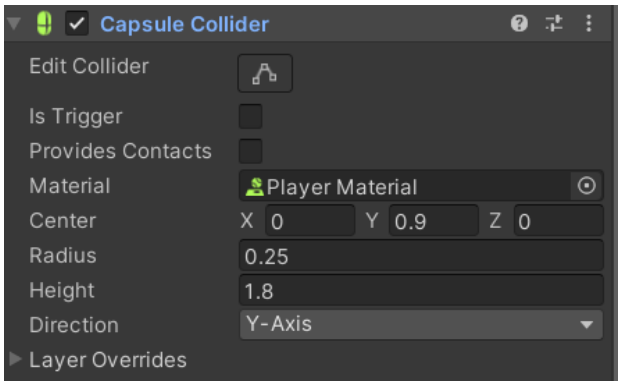
To setup the health configuration, right click in the project window and selecting Create > Character > Health Config





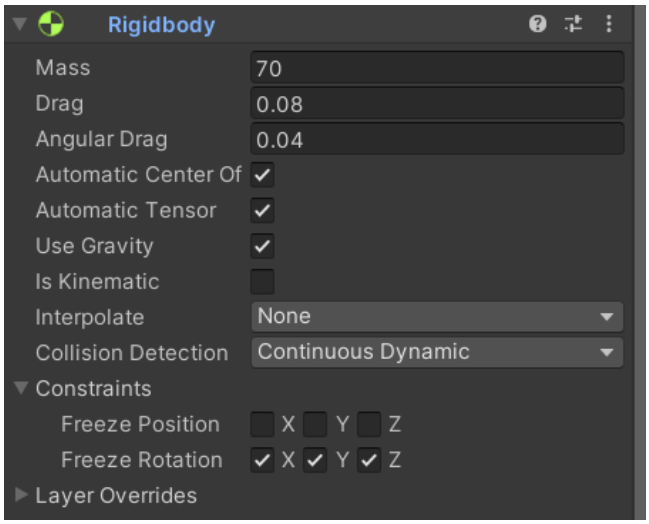
Player Collider Component Setup

To set up the player Collider component, add capsule collider component to the player game object, by default it will be added after adding the Player Controller script.



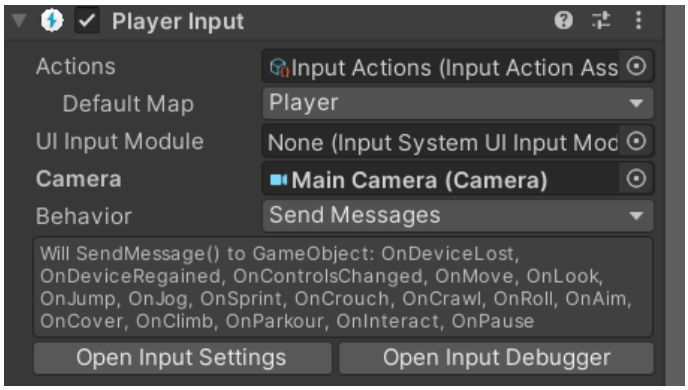
Player Rigid-body Component Setup

To set up the player rigid-body component, add rigid-body component to the player game object, by default it will be added after adding the Player Controller script.



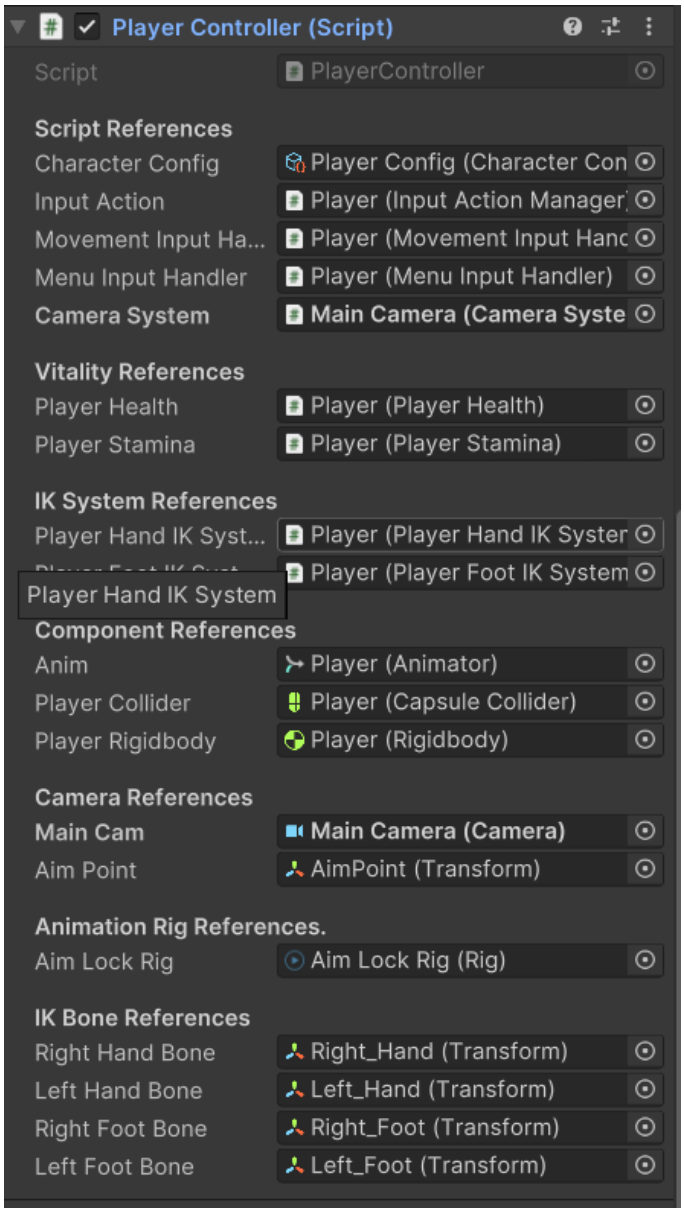
Player Input Component Setup

To setup the Player Input Component, add the Player Input to the player game object, make sure you've installed the Input System Package, by default it will be added after adding the Player Controller script.



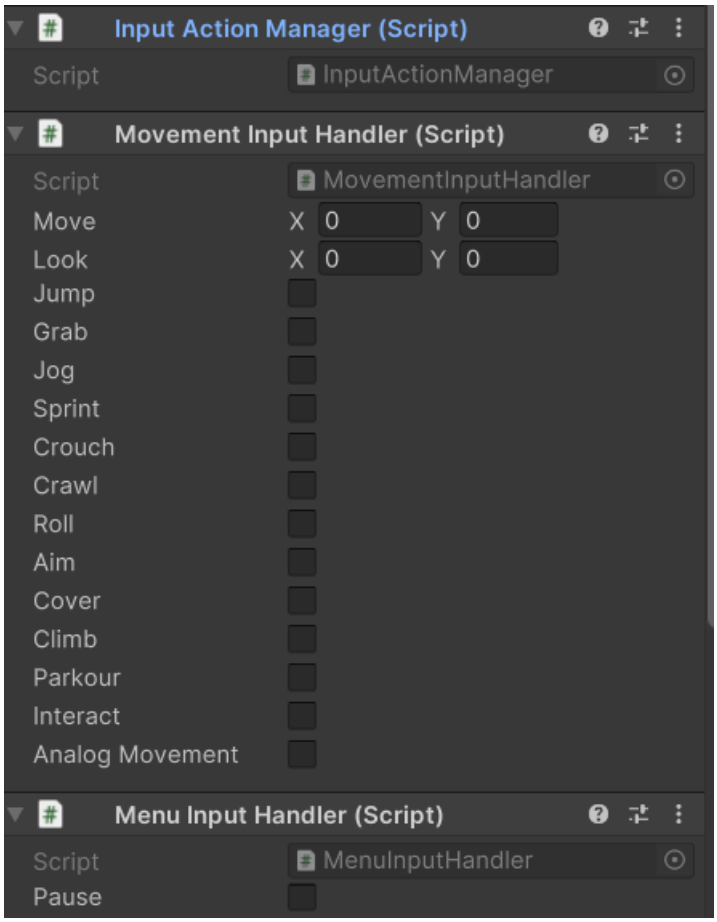
Player Controller Script Setup

To set up the player controller script, add player controller script to the player game object into your scene and it will add all the necessary components and scripts.



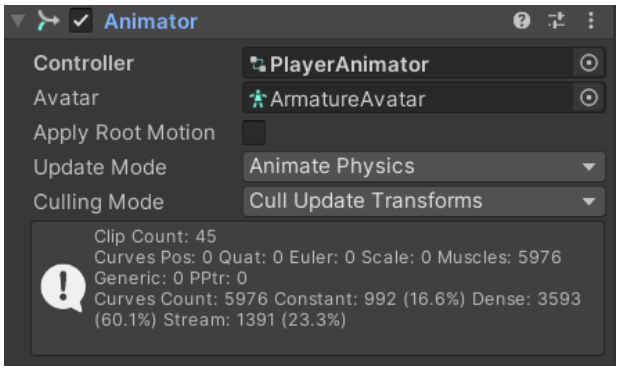
Player Input Manager Scripts Setup

To setup the Player Input Action Manager Scripts, add the **`Input-Action-Manager`**, **`Movement-Input-Handler`**, and **`Menu-Input-Handler`** to the player game object, make sure you've installed the Input System Package, by default it will be added after adding the Player Controller script.



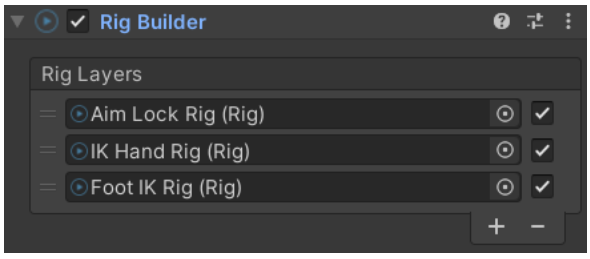
Animator Component Setup

To setup the animator component, add the animator to the player game object, by default it will be added after adding the Player Controller script.



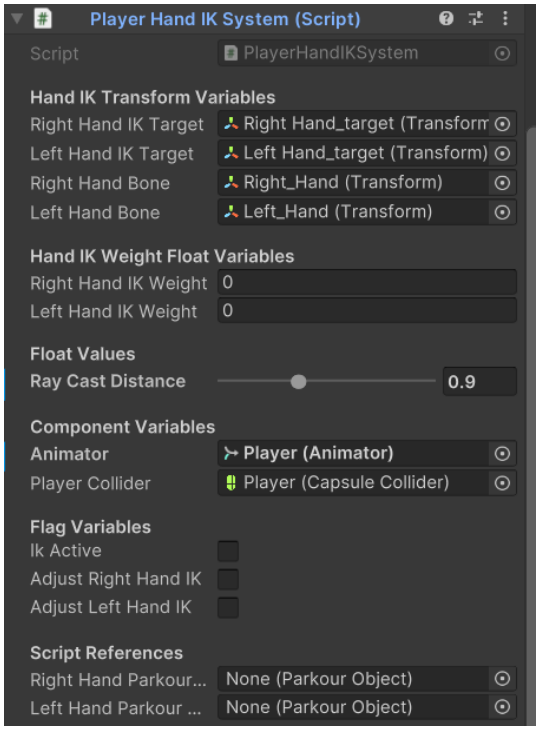
Rig Builder Component Setup

To setup the rig builder component, make sure you've installed the Animation Rigging package from the package manager, then add the Rig Builder component to the player game object, by default it will be added after adding the player controller script but make sure you've installed the Animation Rigging package.



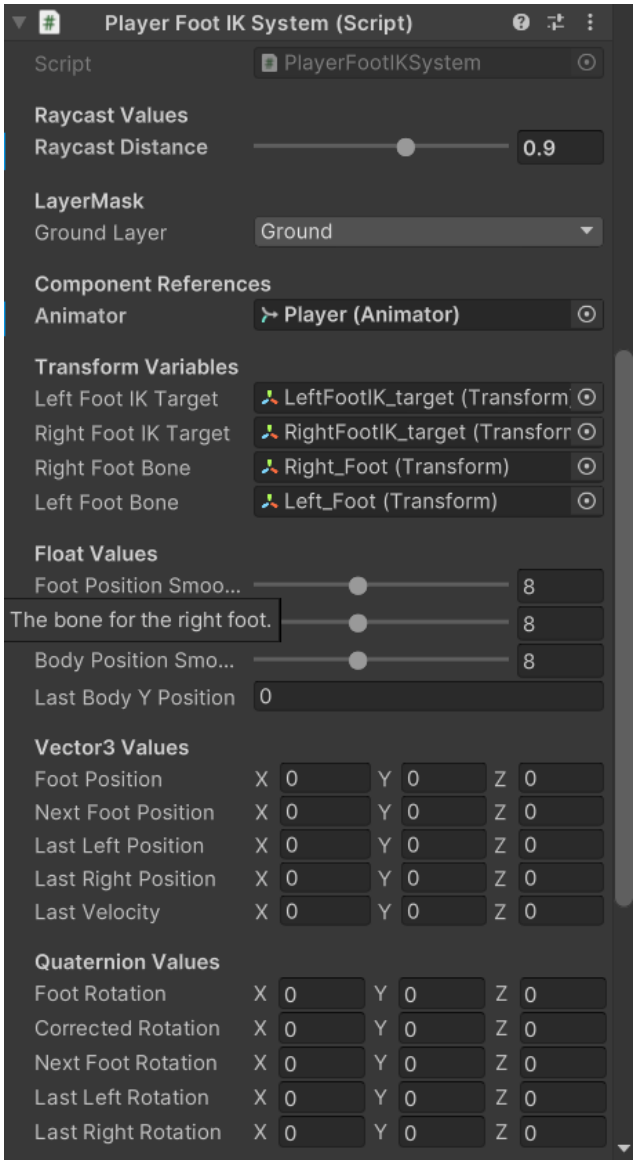
Player Hand IK Script Setup

To setup the player hand IK system script, drag the player hand IK system to the player game object, by default it will be added automatically when added the player controller script.



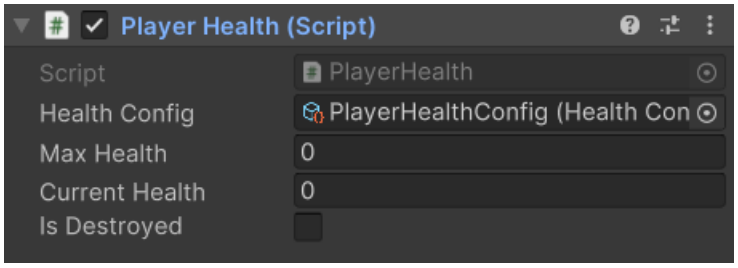
Player Foot IK Script Setup

To setup the player foot IK system script, drag the player foot IK system to the player game object, by default it will be added automatically when added the player controller script.



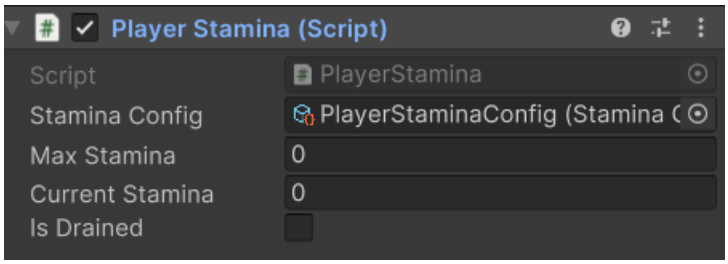
Player Health Script Setup

To set up the player health script, drag the health configuration you've created and configured with the scriptable object then assigns it.



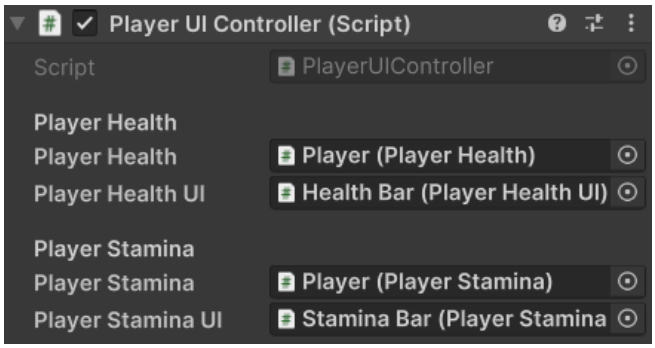
Player Stamina Setup

To set up the player stamina script, drag the stamina configuration you've created and assign it.



Player UI Controller Setup

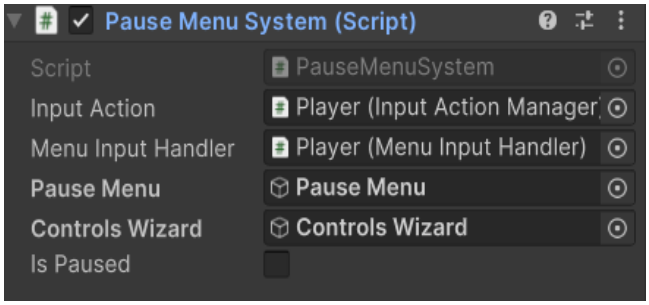
To set up the player UI controller script, assign the script components to each field, and drag the UI of each one and assign to the appropriate field.



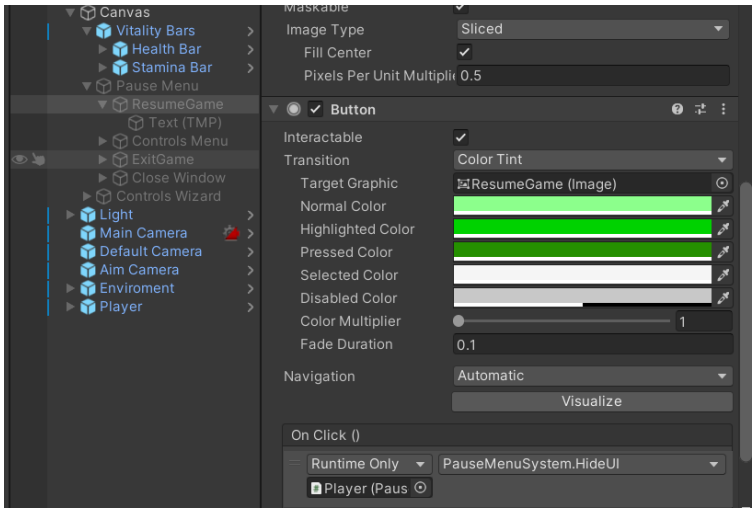
Pause Menu System Setup

To set up the pause menu system script component, follow the below instructions for each game object

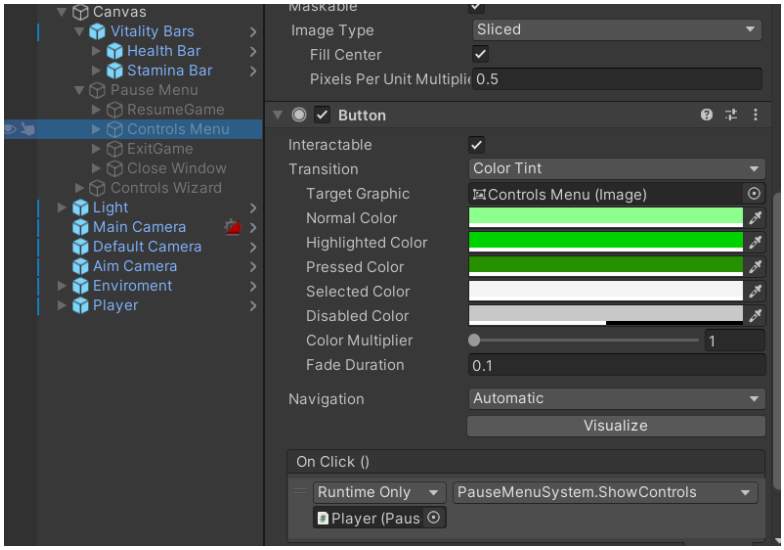
Add the Pause Menu System script to the player game object and assign the fields as shown below



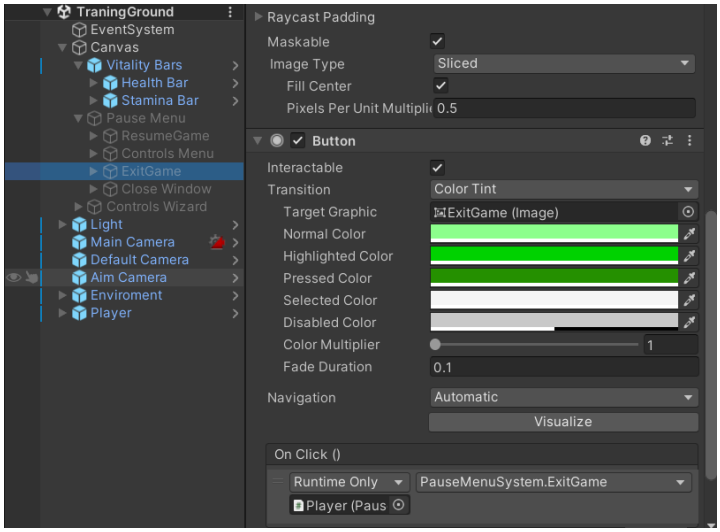
- **Resume Game:** This Game Object is associated with the button that resumes the game when clicked. When the game is paused, clicking this button will hide the pause menu and resume the game.



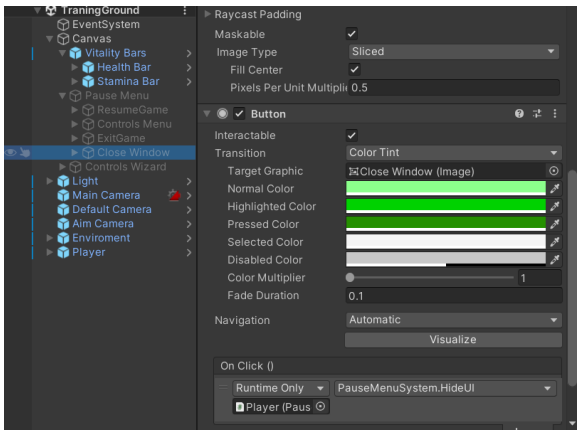
- **Controls Menu:** This Game Object is linked to the button that opens the controls wizard. When clicked, it hides the pause menu and displays the controls wizard, allowing the player to view and modify the game controls.



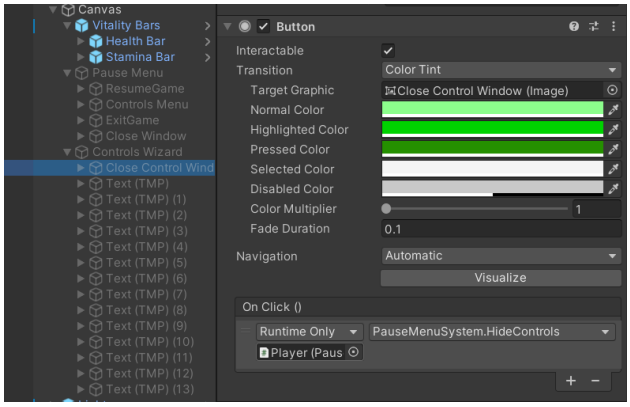
- **Exit Game:** This Game Object is tied to the button that exits the game. Clicking this button will close the game. If the game is running in the Unity editor, it will stop the play mode. If the game is running in a build, it will quit the application.



- **Close Window:** This Game Object is connected to the button that closes the currently open window. It can be used to close the pause menu or the controls wizard, returning the player to the game.

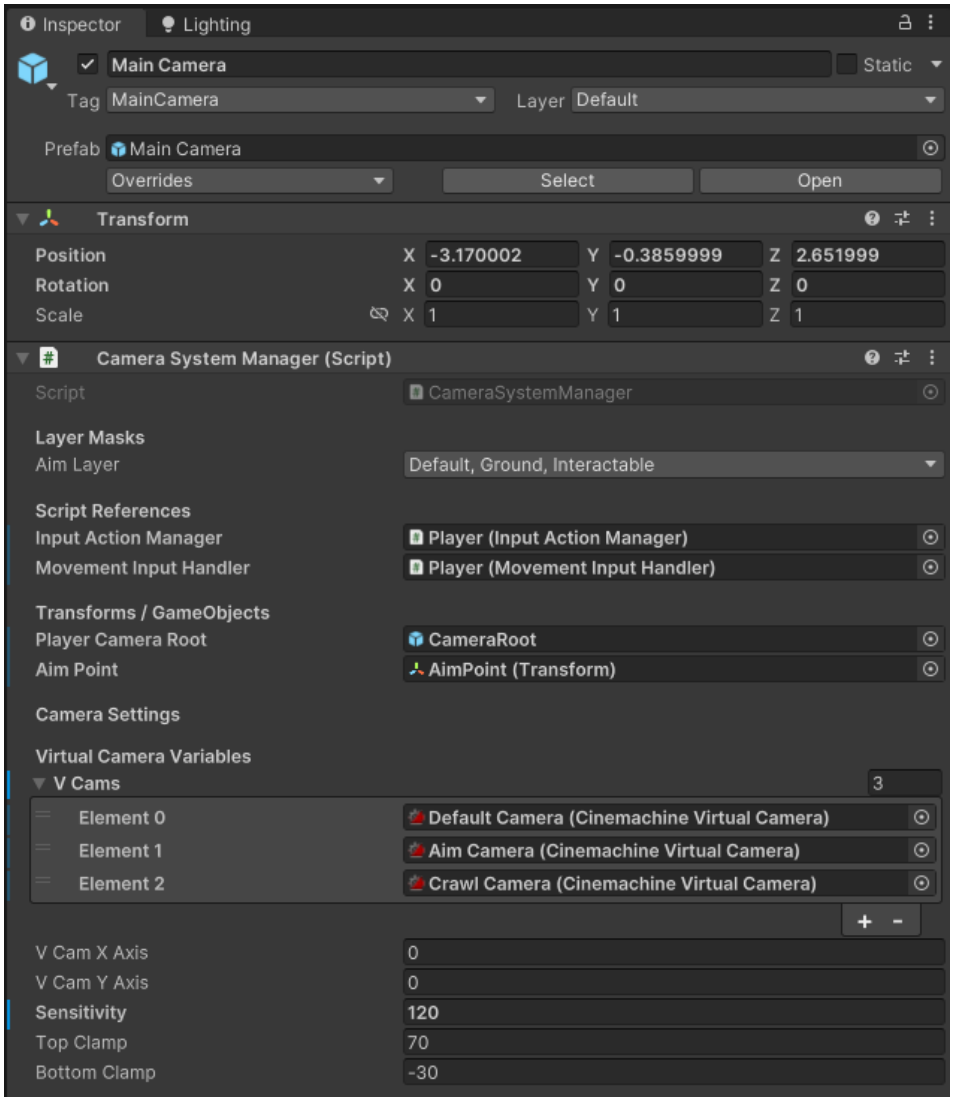


- **Wizard**: This Game Object represents the controls wizard itself. It is a UI element that displays the game controls to the player. It can be shown or hidden using the ‘Show Controls’ and ‘Hide Controls’ methods, inside this game object there is a close window button.



Camera System Manager Setup

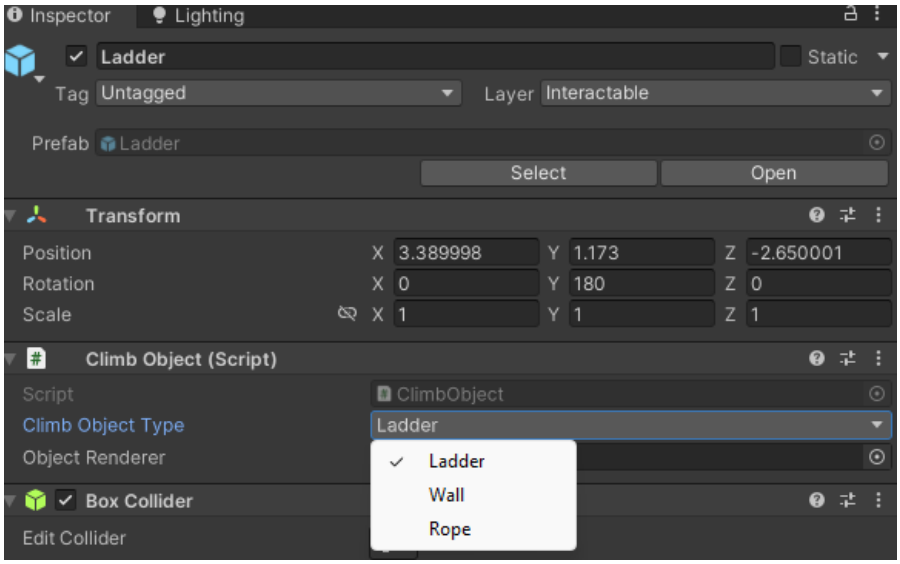
To set up the camera system manager script, assign the required fields as shown below



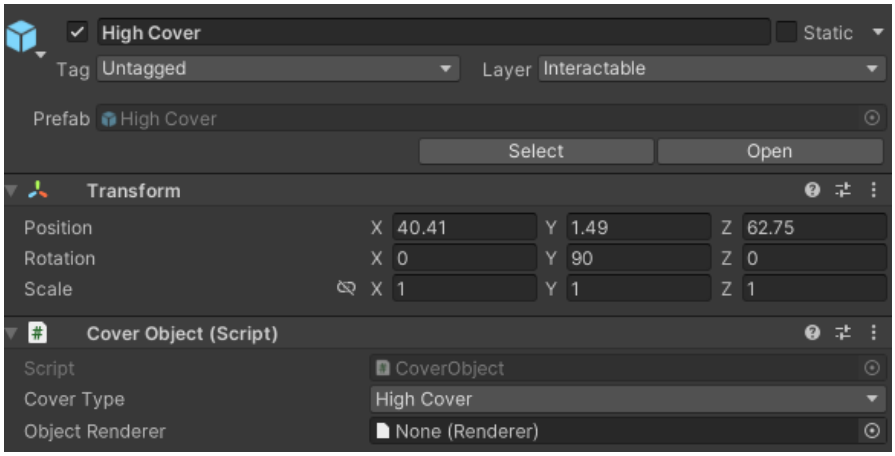
Interaction System Setup

To set up the different interaction objects in the environment assign each script to the corresponding objects

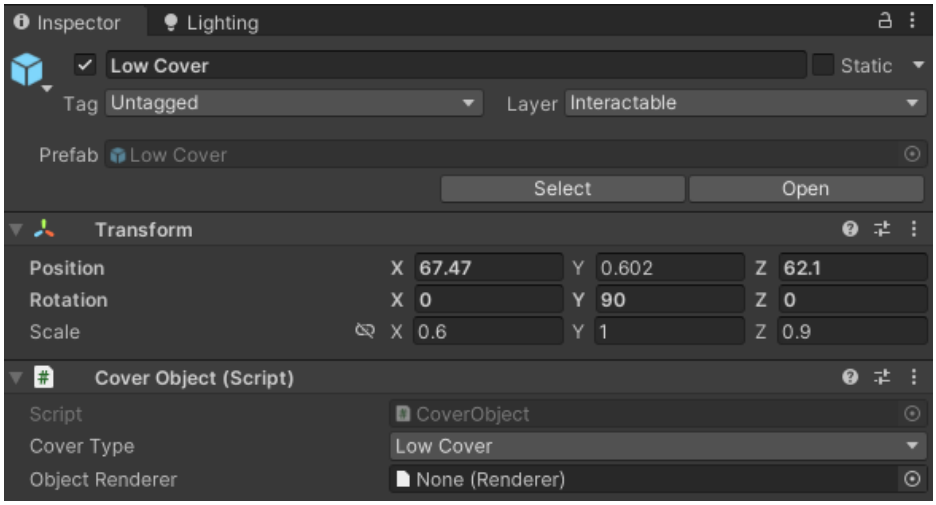
Climb Object:



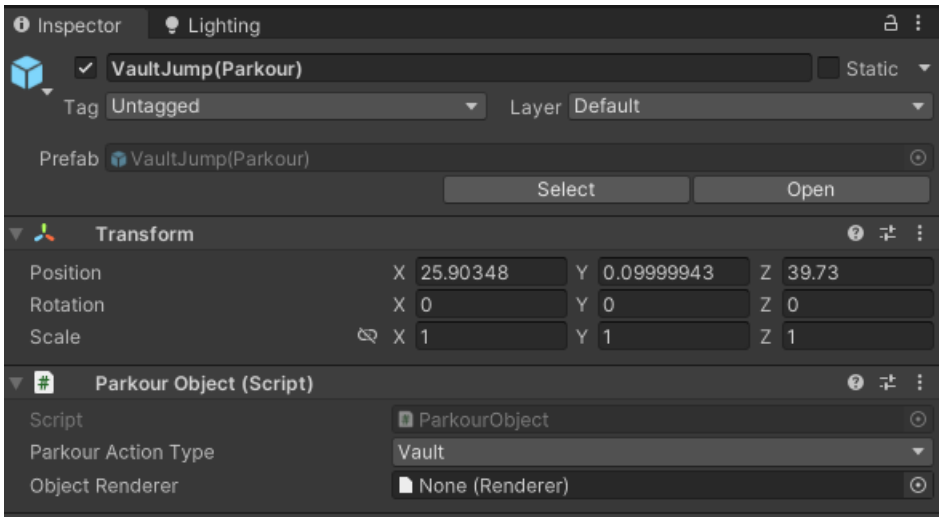
High Cover Object:



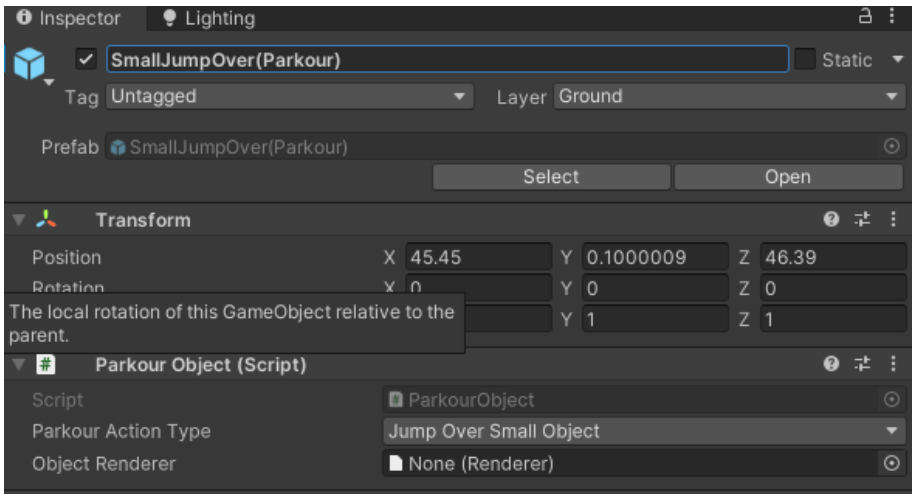
Low Cover Object:



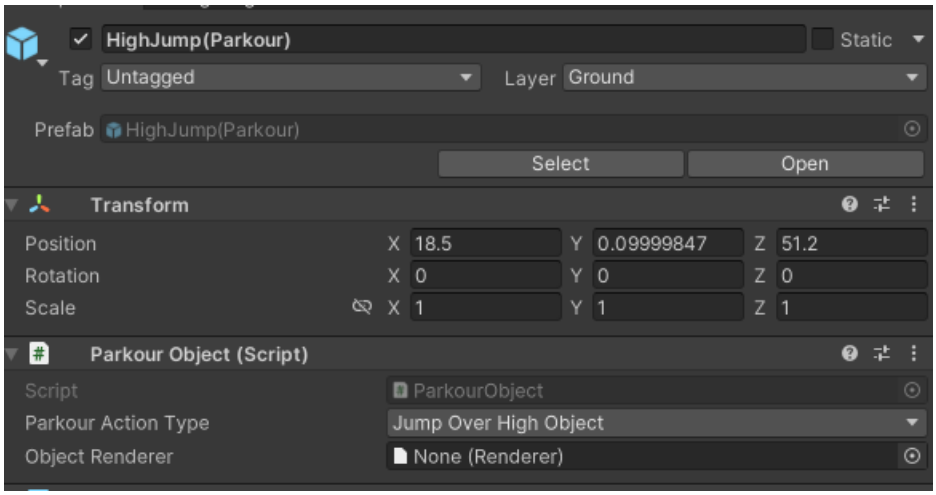
Vault Parkour Object:



Small Parkour Object:



High Parkour Object:



10. Troubleshooting

If you encounter issues, refer to the Usage Examples section in the documentation or visit our Support page for assistance.

11. FAQs

1. **Q: Can I customize player movement states or add new states?**

A: Yes, the system is designed to be extensible. You can customize existing states or add new states by modifying the provided scripts and implementing your desired functionality.

2. **Q: What should I do if my player's character animations are not playing correctly?**

A: Ensure you've unchecked the "Apply Root Motion" in the animator component, check for any warnings or errors in the console and verify the animations are properly triggered based on the player input and state transitions.

3. **Q: Can I add custom UI elements for the player's health and stamina?**

A: Yes, you can customize the player UI element by modifying the provided prefabs that handle the UI under the canvas or create import your own.

4. **Q: How Can I implement additional camera systems or controls?**

A: The camera system manager script allows you to manage camera systems for the player, you can extend this functionality to add more camera systems based on your game requirements.

5. **Q: Can I use this asset with my existing AI solution?**

A: Yes, the core Health, Stamina and Interaction Base Systems are designed to be modular and can be easily integrated with your AI, providing a unified framework for character behavior.

6. Q: Can a single object be used to perform multiple player mechanics, such as cover and vault actions?

Yes, you can configure a single object to perform multiple player mechanics. To do this, change the input keys through the input action settings, and the changes will be automatically reflected in the code base. This allows for seamless interaction with objects, enabling various mechanics such as cover, vault, climb, and more.

7. **Q: What should I do if I encounter issues during setup or usage?**

A: If you encounter any issues, first refer to the "**Troubleshooting**" section in the documentation. If the problem persists, visit our Support page or contact us at support@truetacticalstudio.com for assistance.

12. Support

For technical support or inquiries, please feel free to reach out to us:

- **Email:** support@truetacticalstudio.com
- **Visit our website:** www.truetacticalstudio.com