-Documentation-

Based on Version 4.0

Arktentrion | Artjom Ri.

Introduction

First and foremost, I want to extend a heartfelt thank you for choosing the 2D Action Platformer Kit. Your support and interest in this toolkit are greatly appreciated. This kit is designed to be a comprehensive solution for creating engaging and dynamic 2D action platformer games, offering a range of features to streamline your development process.

For detailed tutorials on how to make the most of this kit, as well as insights into other projects, I invite you to visit and subscribe to my YouTube channel. Here, you will find a wealth of resources and guides to help you at every step of your game development journey.

Additionally, I encourage you to join our official Discord server. This platform serves as a vibrant community for users of the 2D Action Platformer Kit and my other projects. It's a great place to connect with fellow developers, share your progress, and get your questions answered directly by me and other experienced users.

YouTube Channel
Join Our Discord Community

1. Overview of the Toolkit

The 2D Action Platformer Kit is crafted to empower developers, from novice to expert, in bringing their creative visions to life. The toolkit is anchored by a robust Scene Management system, encompassing the Title Screen, Stage Select, End Screen, and various engaging stages, each tailored to captivate players and guide them through a memorable gaming journey.

At the heart of character development and interaction is the comprehensive Actor Prefab System. This system integrates a range of components, from the foundational Actor Game Object to intricate elements like Weapon and Projectile Spawners. These features ensure dynamic and responsive character behavior and interactions within the game world.

A key highlight of this toolkit is its advanced Animation Utilization. By employing an Animator Override Controller, based on a base animator controller found in the Animations/Base Animations folder, the kit allows for seamless and customizable animations for all playable

characters and NPCs. This flexibility ensures that each actor in your game can be uniquely animated, enhancing the visual appeal and player engagement.

Management systems form the backbone of this kit, with dedicated managers for characters, gameplay, items, weapons, and scenes. These managers provide meticulous control over various aspects of gameplay, streamlining the development process and enhancing the overall game quality.

The Player UI Prefab is another integral feature, designed to engage players with intuitive and informative displays that keep them immersed in the gameplay.

The kit is also rich in character variety, featuring player characters with distinct abilities and a wide array of enemy types, including formidable Boss Enemies. This diversity ensures a challenging and varied gameplay experience.

With the inclusion of Scriptable Objects for items, actors, and weapons, the toolkit offers unparalleled flexibility and ease in customizing game elements. This feature allows you to tailor various aspects of your game, from item properties to actor stats and weapon functionalities, with ease and precision.

Adding depth to gameplay are the versatile pickups and projectiles, ranging from health and weapon energy boosts to unique weapon options. These elements not only enrich the gameplay but also provide strategic depth to player interactions and combat scenarios.

The toolkit is also equipped with advanced systems such as a robust Checkpoint System and a seamless Screen Transition Manager, ensuring a smooth and uninterrupted gaming experience.

2. Scenes

The 2D Action Platformer Kit is structured into several categories of scenes, each serving a distinct purpose within the game. These categories include "System Scenes," "Tutorial Scenes," "Regular Stages," and "Final Stages." This structure helps in organizing the game flow and managing the player's progression through different levels and interfaces.

Scene Categories

System Scenes:
These scenes include essential game interfaces like the Title Screen, Stage Select, and the End Screen.

- Title Screen:
  The initial interface where players can start a new game, load, or quit.
- Stage Select: Allows players to choose from available stages to play, access the shop and save progress.
- End Screen: Displays upon game completion.
  The game starts from the Title Screen when built and packaged for release.

Tutorial Scenes:
Designed to introduce players to the game mechanics and controls.
Acts as a bridge between the Title Screen and the Regular Stages.

Regular Stages:
The main levels of the game, available to players from the beginning.
Completion of these stages is typically required to unlock the Final Stages.

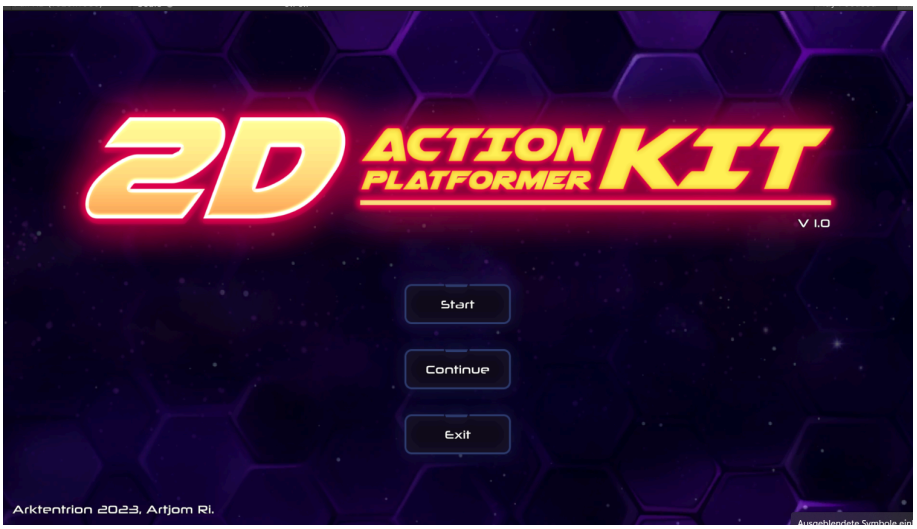
Final Stages:
Advanced levels that are unlocked after completing all Regular Stages.

Development and Testing in Unity Editor

When working within the Unity Editor, developers and testers can start the game from any of these scenes.

2.1 Title Screen

The Title Screen is the first point of interaction for players in the 2D Action Platformer Kit. It serves as the gateway to the game, providing key options such as starting a new game, loading a previous game, or exiting the game.



Features and Structure

New Game:
Initiates a new game session.
Character Selection: Players are presented with a choice between two characters. Selecting a character initiates several actions:

- Session Assignment: The game session is assigned to temporary save slot 0 in the Save System Manager.
- Character Index Setting: The chosen character is set in the Character Assigner.
- Initial Lives: The character is given 3 lives to start the game.
- Tutorial Stage Loading: The game proceeds to load the Tutorial Stage.

Load Game:
Opens the load interface for accessing saved games.

- Save Slots Display: Shows 8 save slots for player's saved games.
- Progress Preview: Highlighting a save slot displays the progress of the player within that slot.
- Loading Process: Selecting a save slot loads the Stage Select screen, reflecting the player's progress corresponding to that slot.

Exit:

Closes the game.

Implementation Details

UI Elements:

The Title Screen consists of buttons or interactive elements for each of the options (New Game, Load Game, Exit).

Character Selection Logic:

Incorporates scripts to handle character selection and initialization of the game session.

Save System Integration:

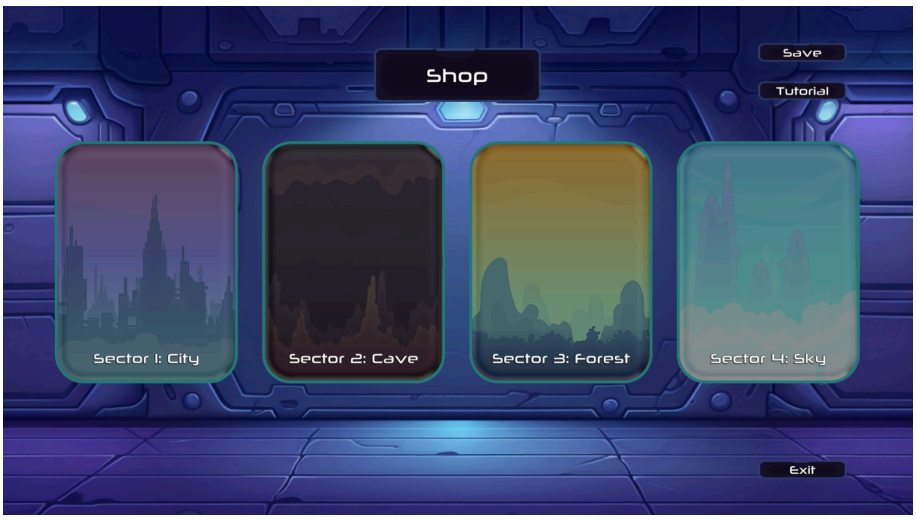
Interacts with a Save System Manager to handle game data for new and existing sessions.

Scene Management:

Utilizes Unity’s scene management system to transition between the Title Screen, Tutorial Stage, Stage Select Screen, and Load Interface.

2.2 Stage Select

The Stage Select Screen in the 2D Action Platformer Kit serves as a pivotal navigation hub for players. It offers a range of choices, including selecting stages to play, accessing an in-game shop, saving progress, revisiting the tutorial, and exiting to the Title Screen. Crucially, it also unlocks and displays the "Final Stages" once all "Regular Stages" are completed.



Features and Structure

Stage Selection:

- Players can select and play "Regular Stages" in any order.
- Stages that have been completed are marked as such.
- Final Stages Access: Upon completion of all Regular Stages, the Final Stages are unlocked and displayed on the screen, offering additional challenges.

In-Game Shop:

Players can purchase items such as Health Tanks, Weapon Energy Tanks, and Mega Tanks using in-game currency collected from stages.

Saving Progress:

The screen includes an option to save the game's progress.

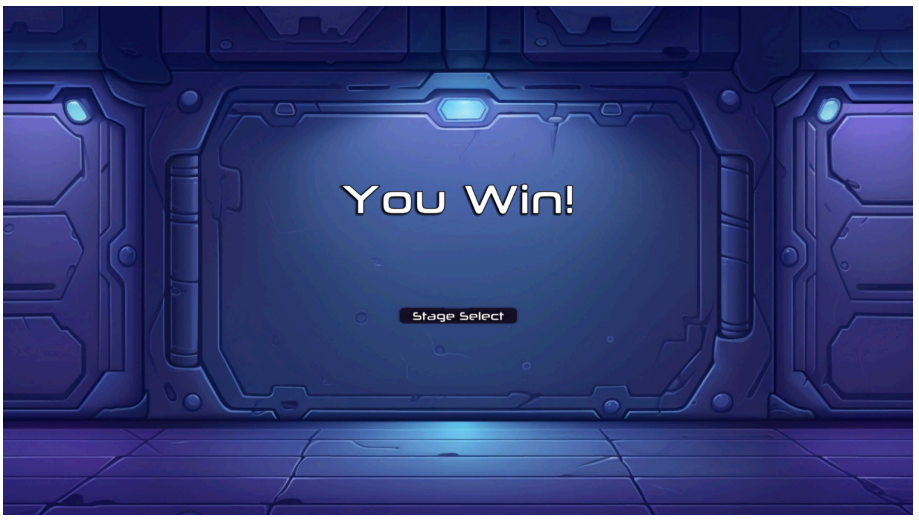
Players can choose from 8 save slots, with each save capturing and displaying the current progress.

Tutorial Access:
The tutorial can be accessed at any point from this screen, allowing players to revisit game mechanics as needed.

Exit to Title Screen:
An option to exit from the Stage Select Screen to the Title Screen is provided for players wishing to leave the game or change settings.

2.3 End Screen

The End Screen in the 2D Action Platformer Kit is a significant component that marks the completion of the game's final stage. It is designed to provide closure to the gaming experience and can be customized as per the game developer's vision.



Key Features

Activation:
The End Screen is triggered upon the completion of the last "Final Stage" in the game.
It signifies the successful conclusion of the main game content.
Default Display:
By default, the End Screen displays the message "You win," acknowledging the player's achievement. This message serves as immediate feedback to the player for completing the game.

Customization Capability:
The End Screen is designed to be customizable. Developers have the flexibility to modify it to display different text, images, or even trigger other actions like playing a cutscene or showing credits.

Navigation Option:
A "Stage Select" button is included, allowing players to return to the Stage Select Screen. This feature provides an option for players to replay stages or access other parts of the game after completing the final stage.

2.4 Stages

Stage scenes in the 2D Action Platformer Kit are the core playable levels where players interact with various elements of the game. These scenes are carefully designed to provide a diverse and engaging gaming experience.

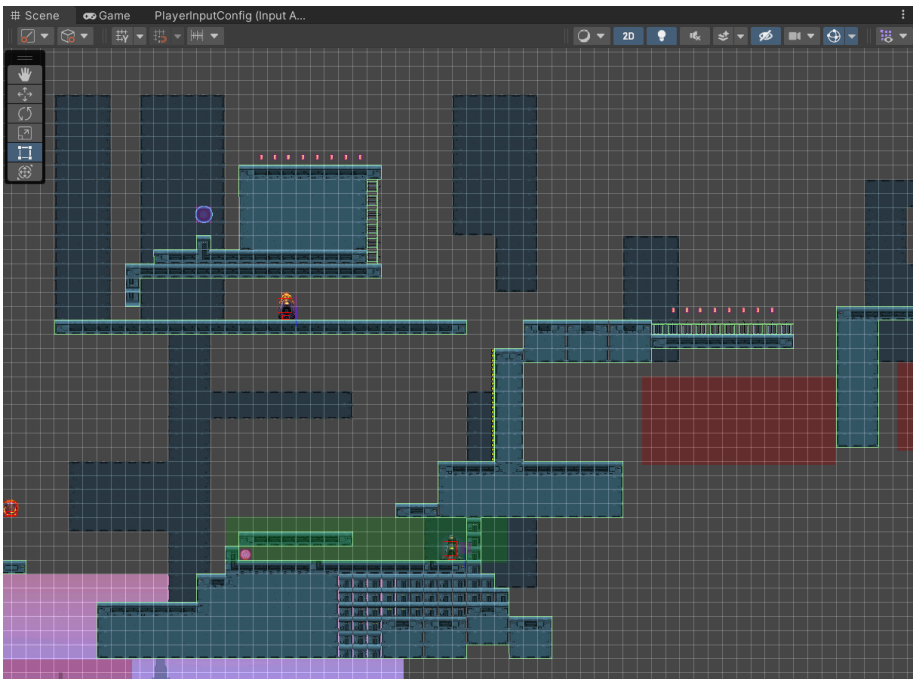
Components of Stage Scenes

Background Music:
Each stage features a Background Music Game Object to select and loop an audio file, enhancing the atmosphere of the stage.

Background (Parallax):
Implements a parallax background with multiple layers.
Each layer can have a different scrolling speed, adding depth and dynamism to the stage's visual appeal.

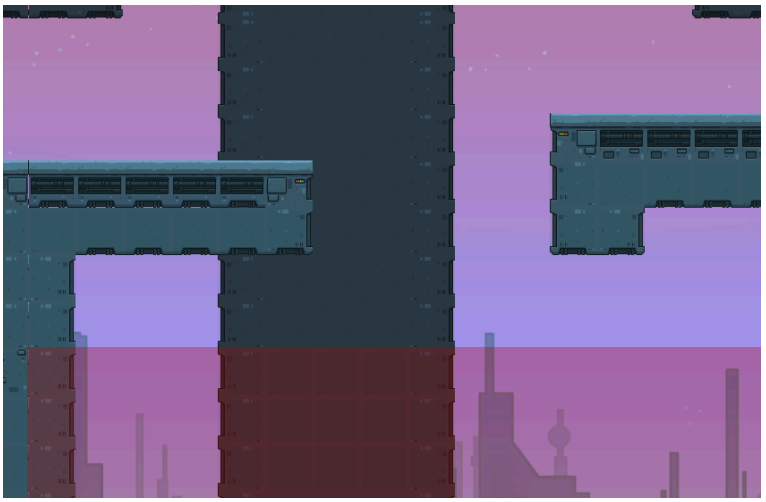
Tilemap Architecture:
The architecture of the stage is drawn using a Tilemap, consisting of several layers:

- Ground Layer: Used to draw the ground and walls.
- Background Layer: For non-interactive scenic elements.
- Ladder Layer: To create climbable structures.
- One Way Platforms Layer: Platforms that can be jumped through from underneath.
- Spikes Layer: For creating hazardous areas.



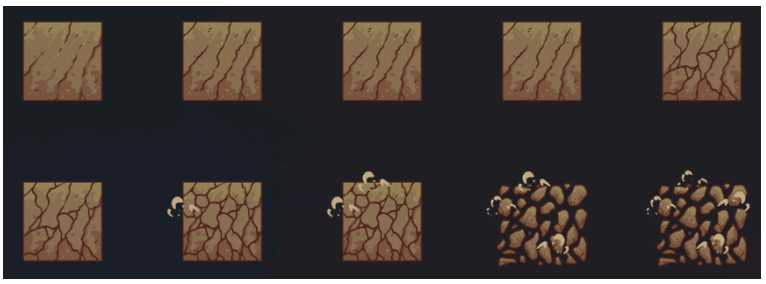
Pit Colliders:

Resizable colliders placed throughout the stage to destroy game objects falling into pits, adding challenge and hazards.



Breakable Blocks:

Blocks that can be placed in various parts of the stage, which are destructible by the player or other game elements.



Enemy Placement:

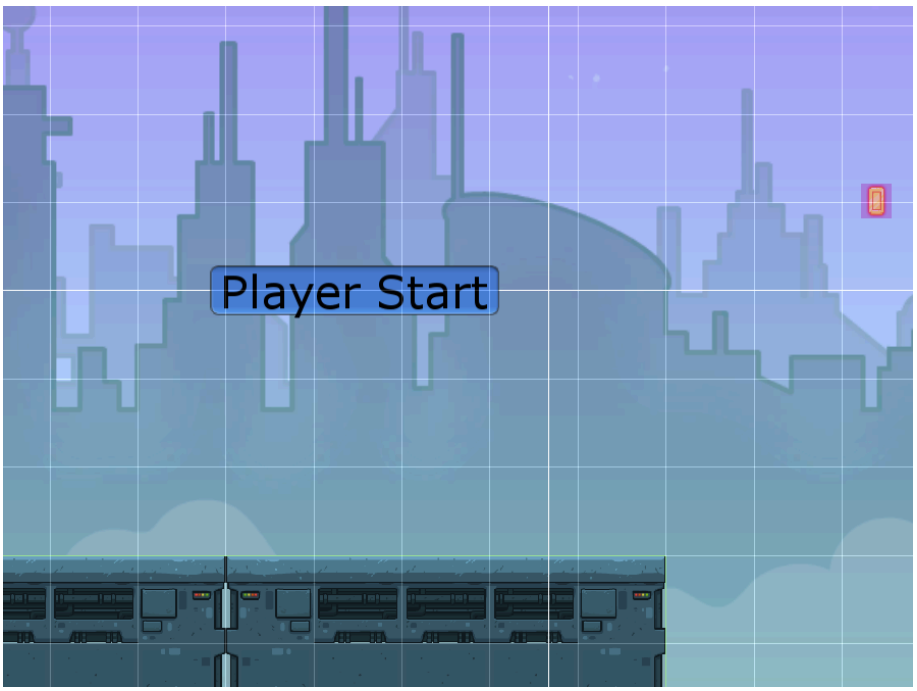
Enemies can be strategically placed throughout the stage, providing challenges and combat experiences.

Boss Fights:

Each stage concludes with a boss fight, which must be defeated to complete the stage.

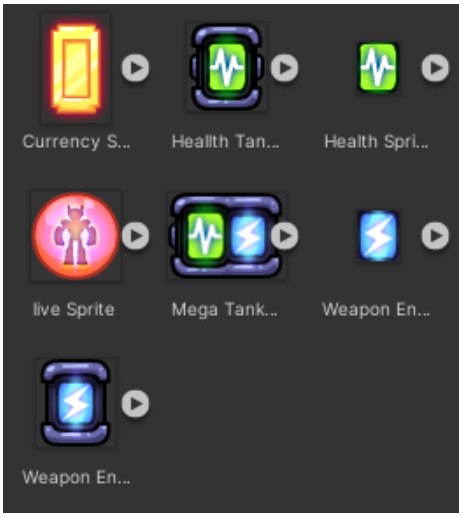
Checkpoints:

Checkpoints can be set within each stage, allowing players to save their progress at specific points. These checkpoints are crucial for maintaining player progress and providing safe respawn points after character defeat.



Collectible Objects:

Stages are populated with various objects like currency, weapons, items, lives, health pickups, and weapon energy pickups.

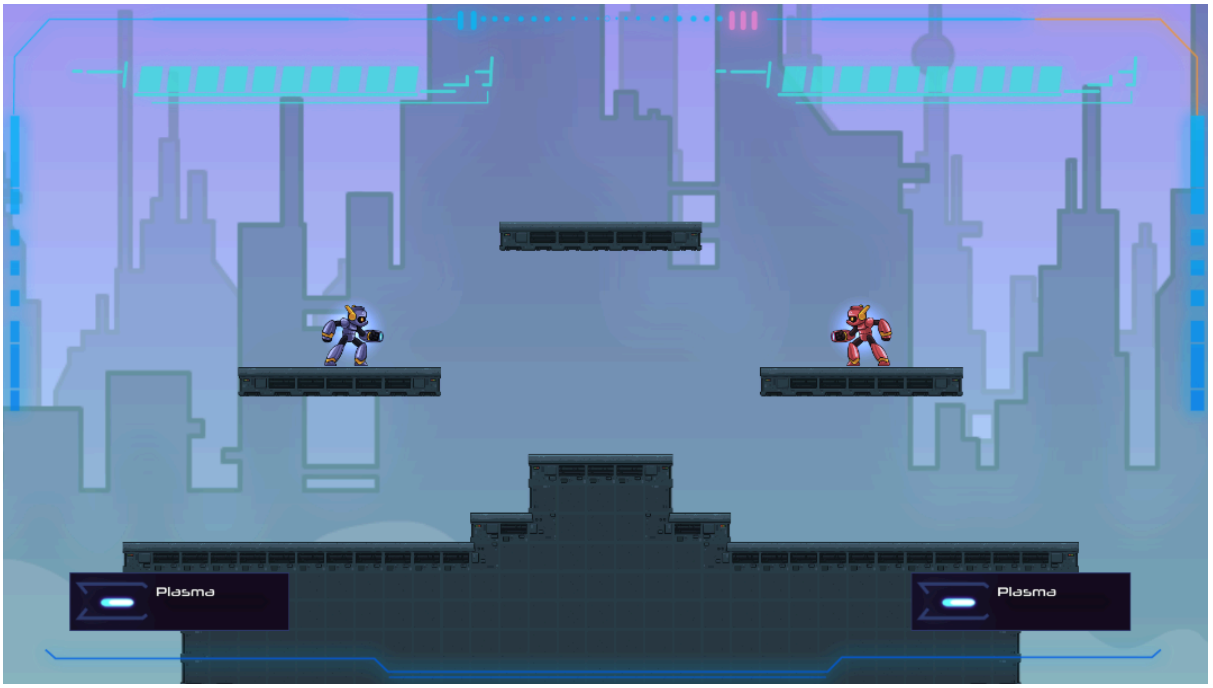


Template Stage:

The Stages folder includes a template stage, which can be duplicated to create new stages. The template comes equipped with all necessary game objects for immediate functionality.

2.5 Arena

The **Arena** is a unique scene in the 2D Action Platformer Kit designed specifically for **Player vs. Player** gameplay. This setup allows two players to face off against each other, showcasing the multiplayer capabilities from a competitive perspective rather than cooperative gameplay.



Features and Structure

Player vs. Player Setup:

- The **Character Manager** loads two player actors, each configured with their own unique controls and input devices.
- Equipment and abilities are set to damage actors tagged as "Player" and "Player 2," ensuring proper interaction and combat mechanics between the two players.
- Spawn points for each player can be customized by repositioning the "Player 1 Start" and "Player 2 Start" objects within the scene, allowing for flexible starting positions in the arena.

Short Multiplayer Experience:

- The Arena is structured as a quick and dynamic demonstration of the multiplayer system, emphasizing head-to-head combat.
- This scene provides a focused environment for testing and enjoying the Player vs. Player features of the kit.

Game Mechanics:

- Players can utilize the same movement, attack, and special abilities available in the main game stages.
- Damage calculations, health management, and victory conditions follow the standard mechanics of the kit, ensuring consistency with the core gameplay experience.

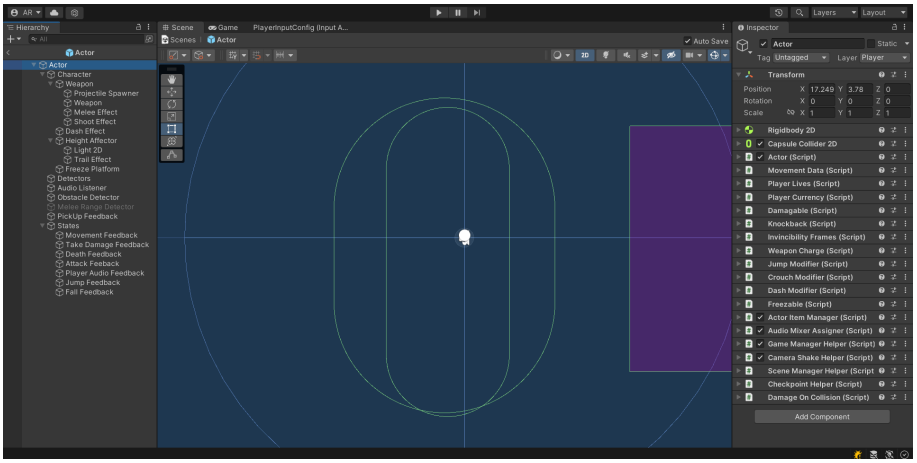
Customization Options:

- Developers can expand or modify the Arena scene to introduce additional features, such as power-ups, hazards, or different layouts, to create varied Player vs. Player experiences.
- The equipment, abilities, and character stats can be adjusted independently for Player 1 and Player 2 to balance or enhance gameplay.

3. Actor Prefab

3.1 Actor Game Object

The Actor Prefab in the 2D Action Platformer Kit is a versatile base used for both playable and non-playable characters. It incorporates a comprehensive set of components, including a state machine, to ensure proper functionality.



Components of the Actor Game Object

Rigidbody 2D:
Provides physics capabilities for the actor, including gravity and collision responses.

Capsule Collider 2D (Set as Trigger):
Enables the actor to collect objects and interact with trigger zones in the game.

Actor Script:
Central script managing the actor's states and transitions between them. Coordinates animations, movements, and interactions.

Movement Data Script:
Stores and manages data related to the actor's movement, such as speed and direction.

Player Lives Script:
Manages the number of lives the player character has left.

Player Currency Script:
Tracks and manages the in-game currency accumulated by the player.

Damagable Script:
Handles the actor's health and responses to taking damage.

Knockback Script:
Applies knockback effects to the actor when hit by enemies or obstacles.

Invincibility Frames Script:
Provides temporary invulnerability periods after the actor takes damage.

Weapon Charge Script:
Manages the charging mechanics for weapons used by the actor.

Jump Modifier Script:
Customizes the actor's jumping behavior, like number of jumps and jump height.

Crouch Modifier Script:
Enables and manages the actor's ability to crouch.

Dash Modifier Script:
Adds and controls dashing abilities, including air dashing and invincible dashing.

Freezable Script:
Allows the actor to be temporarily frozen or immobilized.

Actor Item Manager Script:
Manages the items held by the actor.

Audio Mixer Assigner Script:
Assigns audio sources to specific audio mixers for sound management.

Game Manager Helper Script:
Provides utility functions related to the game's overall management.

Camera Shake Helper Script:
Triggers camera shake effects under certain conditions.

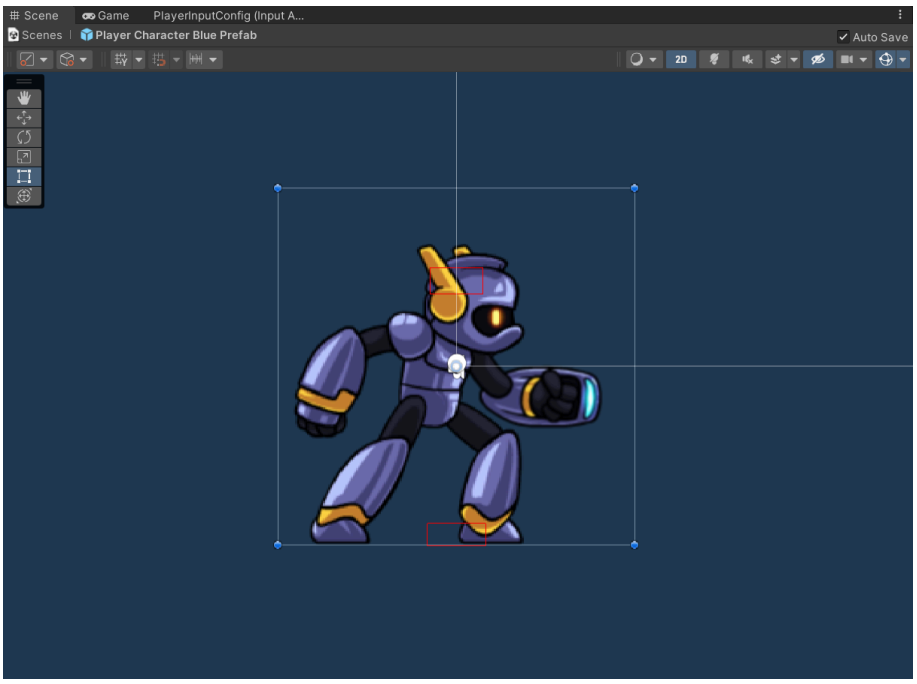
Scene Manager Helper Script:
Assists with scene transitions and management.

Checkpoint Helper Script:
Manages checkpoint registration and player respawning.

Damage on Collision Script:
Applies damage to the actor or other objects upon collision.

3.2 Character Game Object

the "Character" Game Object within the Actor Prefab is crucial for visual representation and animation of the actor.



Components of the Character Game Object

Sprite Renderer:
This component is responsible for rendering the sprite (2D image) of the character on the screen. It allows for customization of the sprite's appearance, such as changing the image, color, and material.

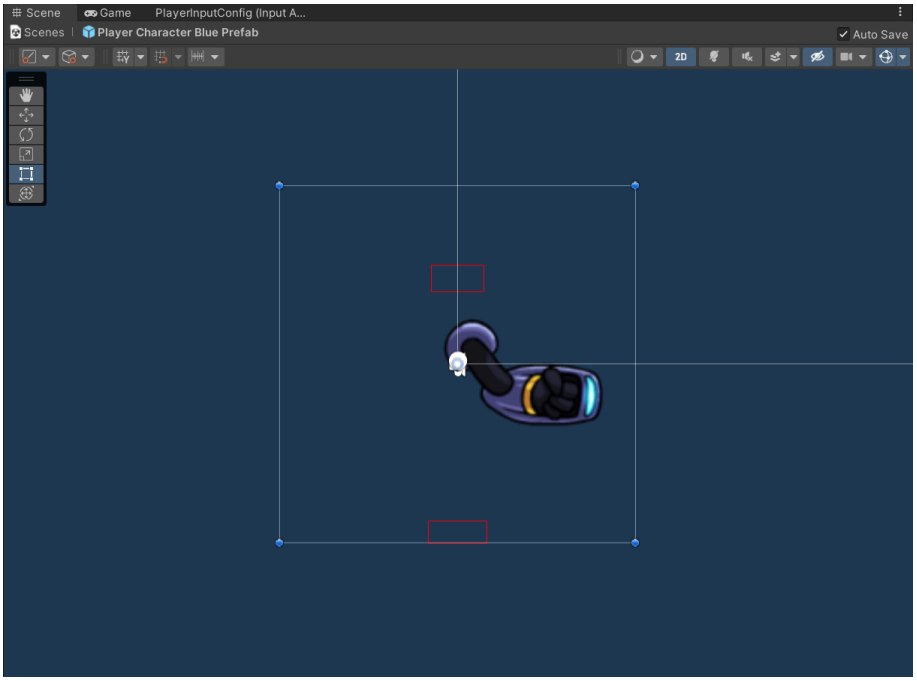
Animator:
The Animator component is linked to an Animation Controller, which defines the various animation states and transitions for the character. It plays a crucial role in animating the character based on various actions or states (like walking, jumping, attacking).

Actor Renderer:
The Actor Renderer is responsible for updating the visual aspects of the actor, such as flipping the sprite depending on the direction of movement. It ensures that the character's sprite always faces the appropriate direction during gameplay.

Actor Animations:
This component manages the specific animations for the actor.
It plays, stops, and controls various animations based on the gameplay, such as triggering a jump animation when the state machine calls the animation.

3.3 Weapon Game Object

The Weapon Game Object within the Actor Prefab is a crucial component for playable characters in the 2D Action Platformer Kit, especially for those characters that utilize ranged weapons. This Game Object is equipped with several components essential for the visual representation and animation of the weapon.



Components of the Weapon Game Object

Sprite Renderer:
This component is responsible for rendering the sprite (2D image) of the weapon.
It allows for customization in terms of the weapon's appearance on the screen.

Animator:
Linked to an Animation Controller, this component manages the weapon's animations.
It handles various animation states and transitions, such as shooting or reloading.

Range Weapon Animations:
This script manages the specific animations related to ranged weapons.It includes functions to play animations based on the weapon's action, like firing or charging. Additionally, it contains UnityEvents for animation actions and animation ends, allowing for the triggering of specific events at key animation points.

3.4 Projectile Spawner Game Object

The Projectile Spawner Game Object, nested within the Weapon Game Object of the Actor Prefab, is a dedicated component used for spawning projectiles. This Game Object is essential for characters equipped with ranged weapons.

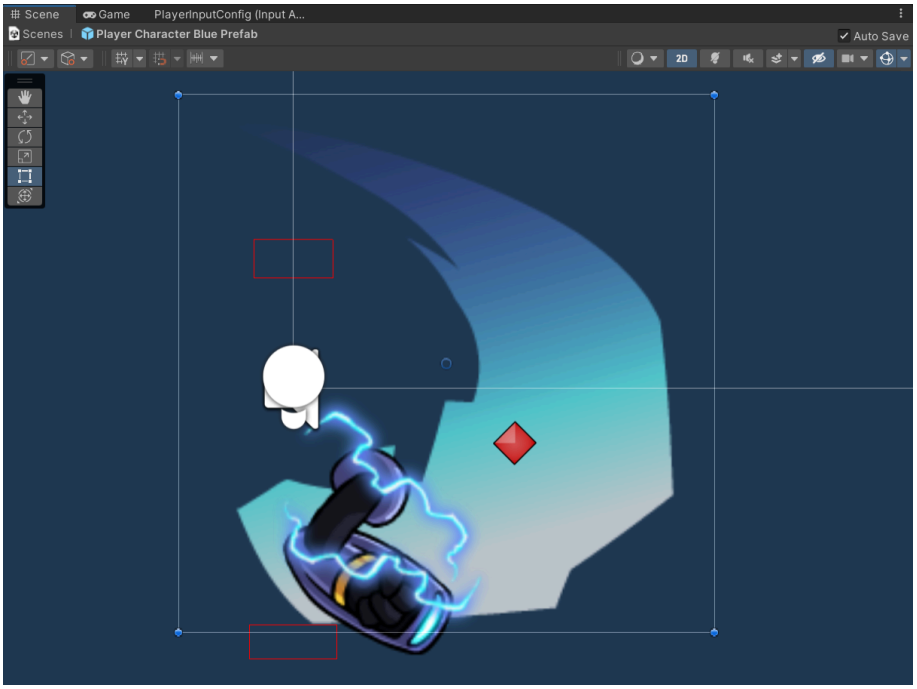
Component of the Projectile Spawner Game Object

Projectile Spawner Script:

- Responsible for the instantiation and management of projectiles fired by the actor.
- Maintains the initial rotation of the projectile spawner, ensuring that projectiles are spawned with the correct orientation relative to the character and weapon.
- Includes functionality to adjust the spawner's rotation, facilitating correct projectile direction based on the actor's facing direction, particularly useful for actors who can face or attack in multiple directions.

3.5 Melee Effect Game Object

The Melee Effect Game Object, nested within the Weapon Game Object of the Actor Prefab, plays a crucial role in enhancing the visual impact of melee attacks in the game. This Game Object is specifically designed to add dynamic visual effects to melee combat, making each attack more vivid and engaging.



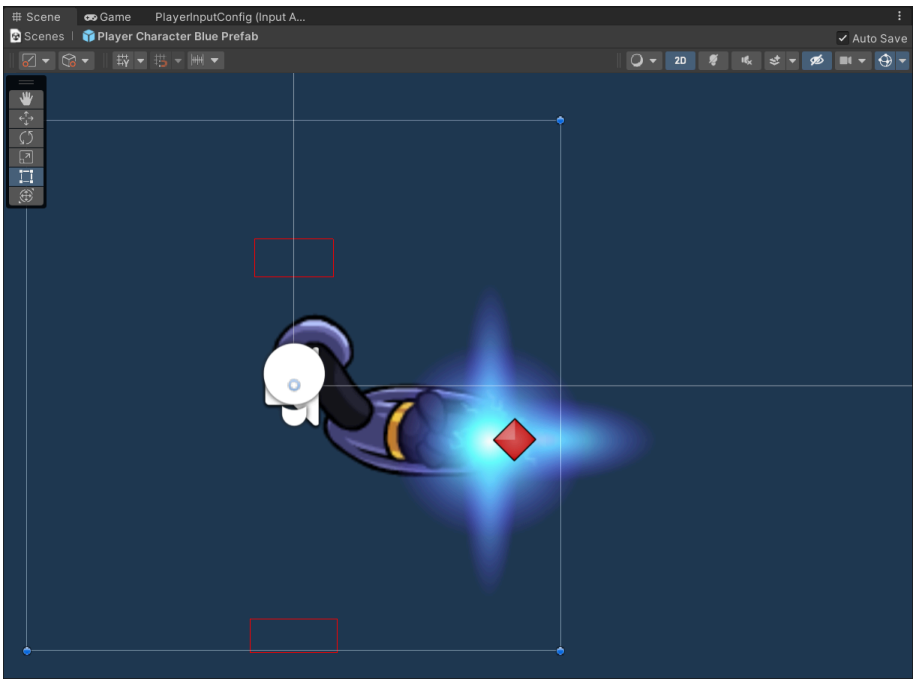
Component of the Melee Effect Game Object

Sprite Renderer:
The primary component of the Melee Effect Game Object.

- Responsible for rendering the sprite that represents the visual effect of a melee attack.
- This sprite is typically a stylized representation of the impact or force of the melee attack, such as a swipe, slash, or hit effect.
- The sprite renderer can be customized with different sprites to suit various types of melee attacks or weapons.

3.6 Shoot Effect Game Object

The Shoot Effect Game Object, nested within the Weapon Game Object of the Actor Prefab, is a dedicated component designed to enhance the visual representation of shooting attacks in the game. Similar to the Melee Effect, this Game Object focuses on adding dynamic visual effects to ranged attacks, contributing to a more immersive and engaging combat experience.



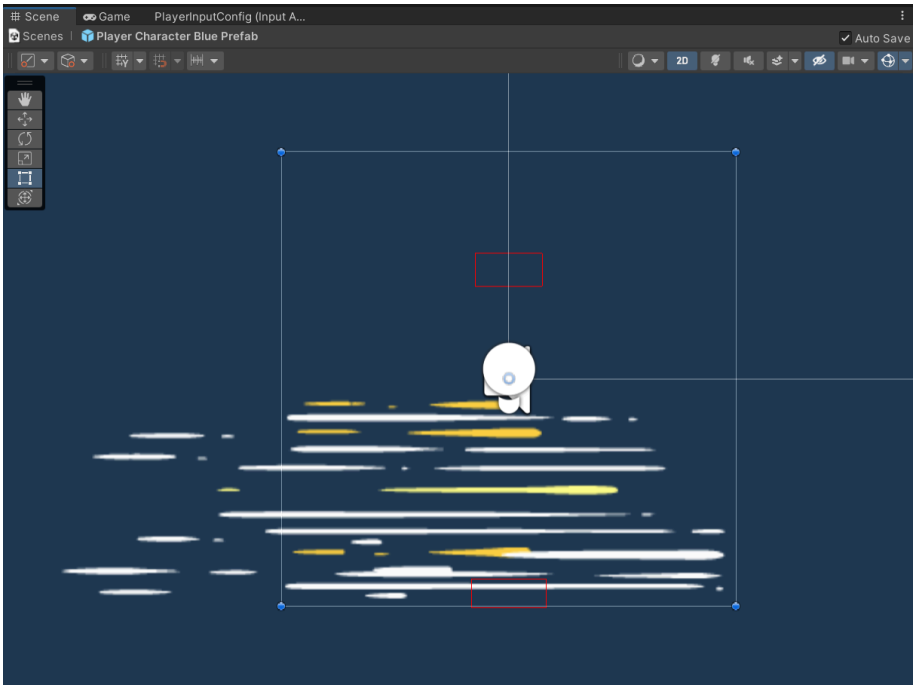
Component of the Shoot Effect Game Object

Sprite Renderer:
This is the sole component of the Shoot Effect Game Object.

- It is responsible for rendering the sprite that represents the visual effect of a shooting attack.
- The sprite typically illustrates the effects associated with firing a weapon, such as a muzzle flash, bullet trail, or any other kind of shooting effect.
- The Sprite Renderer allows for customization with different sprites to accommodate various shooting styles or weapon types.

3.7 Dash Effect Game Object

The Dash Effect Game Object, located within the Actor Prefab, plays a crucial role in visually representing the dash action in the game. This Game Object is designed to add a dynamic visual effect to the character's dash movements, enhancing the overall perception of speed and agility.



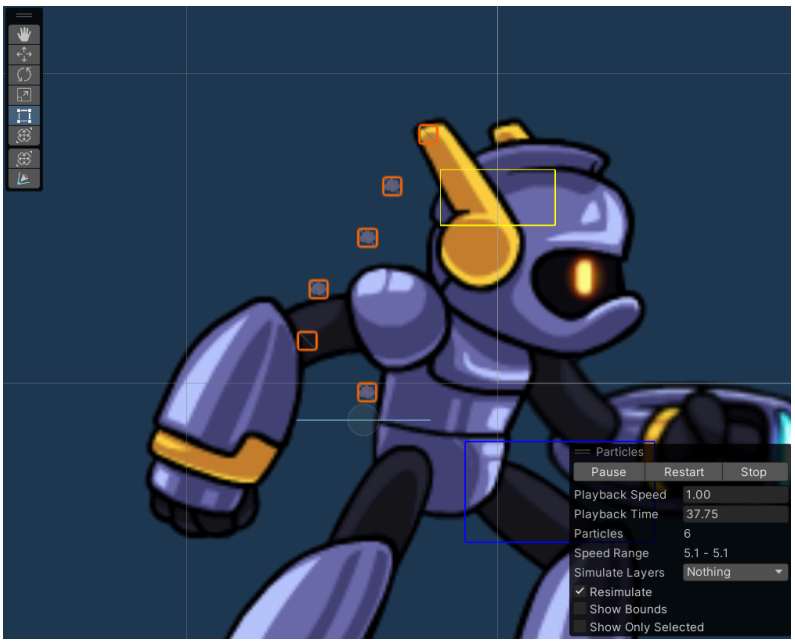
Component of the Dash Effect Game Object

Sprite Renderer:
The primary component of the Dash Effect Game Object.

- It renders the sprite that visually represents the dash effect.
- The sprite typically depicts a motion blur, streaks, or other effects that convey the rapid movement associated with dashing.
- This component allows for customization with different sprites to suit various dash styles or character aesthetics.

3.8 Wall Slide Effect Game Object

The Wall Slide Effect Game Object, nested within the Actor Prefab, is designed to visually enhance the gameplay experience during wall sliding. This Game Object ensures that players can see dynamic effects when their character interacts with walls, making the mechanics more immersive and visually appealing.



Component of the Wall Slide Effect Game Object

Particle System:

- The primary component of the Wall Slide Effect Game Object.
- Responsible for displaying a stream of dust particles that visually represent the friction and interaction between the character and the wall during a slide.
- The Particle System is fully customizable, allowing developers to adjust parameters such as the size, color, and emission rate of the particles to suit their game's aesthetic.
- The effect is activated when the character enters the wall sliding state and deactivates when the character leaves this state.

3.9 Height Affector Game Object

The Height Affector Game Object, nested within the Actor Prefab, serves as a dynamic component for managing the visual representation of the actor's height. This Game Object is particularly crucial for ensuring visual consistency during actions that alter the character's height, such as crouching and dashing.

Components and Functionality

Height Affector Script:
The primary component of the Height Affector Game Object.

- Its role is to indicate and adjust for the current height of the actor.
- It dynamically positions certain visual elements to match the actor’s changing height.

Child Game Objects

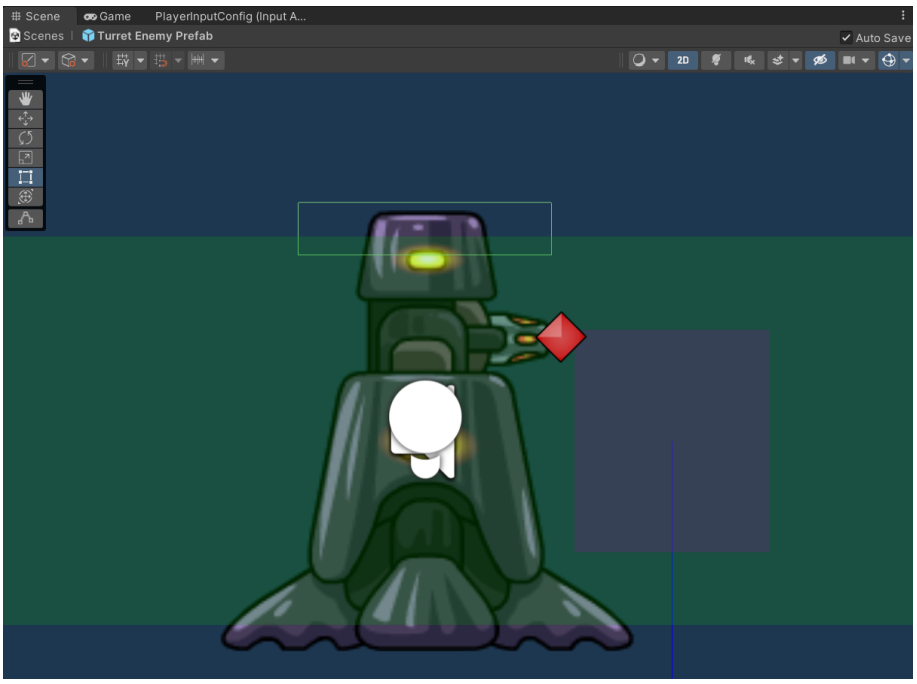
Light 2D:
A child Game Object that may be used for lighting effects centered around the actor. The Height Affector ensures this light effect remains aligned with the actor’s current height.

Trail Effect:
Another child Game Object typically used for visual trailing effects, like motion blur or afterimages during rapid movement. The Height Affector script adjusts the trail’s position in response to height changes.

Height Adjustment Functionality:
During actions like crouching or dashing, the actor’s visual height changes. The Height Affector script responds to these changes by repositioning child Game Objects like Light 2D and Trail Effect. This ensures that these elements are always appropriately centered, maintaining visual coherence and accuracy.

3.10 Freeze Platform Game Object

The Freeze Platform Game Object, a part of the Actor Prefab, is designed to enhance gameplay mechanics by introducing an interactive element when an actor is frozen. This Game Object becomes functional when the actor is affected by a freeze weapon, adding a strategic layer to the game's combat and environment interaction.



Components and Functionality

Box Collider 2D:
The primary component of the Freeze Platform Game Object.

- This collider is enabled when the actor is frozen, creating a solid platform that other characters or objects can stand on.
- When the freeze effect is not active, the Box Collider 2D is disabled, making the platform non-interactive.

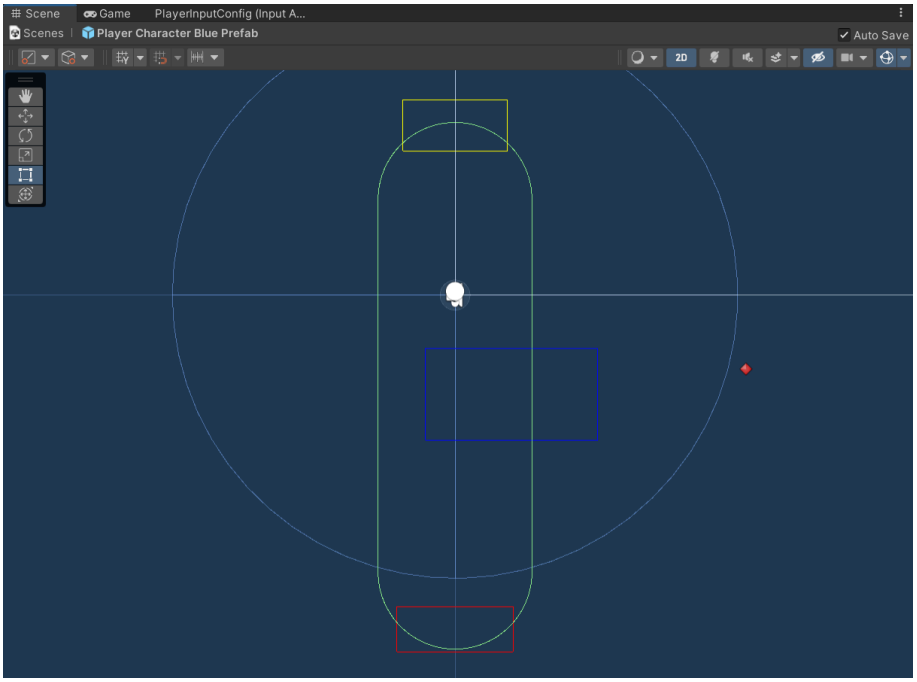
Activation Mechanism:
The Box Collider 2D's activation is tied to the actor being affected by a freeze weapon.
An event-driven mechanism detects when the actor is frozen and enables the collider, transforming the actor into a temporary platform.

Deactivation Mechanism:
Once the freeze effect wears off, the mechanism disables the Box Collider 2D.
This returns the actor to its normal state, no longer serving as a platform.

3.11 Detectors Game Object

The Detectors Game Object, part of the Actor Prefab, houses various components essential for detecting environmental interactions such as collisions, climbable objects, walkable

ground, walls, and ceilings. This Game Object plays a crucial role in character movement and interaction within the game environment.



Components of the Detectors Game Object

Capsule Collider 2D:
Used to detect collisions with other objects in the game. Essential for determining interactions between the actor and the game environment.

Actor Ladder Detector:
Detects climbable objects, enabling the actor to climb ladders or similar structures.
This functionality enhances vertical movement and exploration possibilities within the game.

Actor Ground Detector:
Detects walkable ground, ensuring that the actor can stand, walk, or run on solid surfaces.
Accurate ground detection is vital for realistic movement and preventing undesired interactions like falling through platforms.

Actor Wall Detector:
Detects walls, enabling mechanics like wall sliding or wall jumping. This component adds an extra layer of mobility and agility to the actor's movement capabilities.

Actor Ceiling Detector:
Detects ceilings, especially useful during actions like crouching and dashing.
It ensures that the actor's movement is constrained appropriately when interacting with overhead obstacles or environments.

3.12 Audio Listener Game Object

The Audio Listener Game Object, incorporated within the Actor Prefab, is a critical component for managing audio perception in the game. It plays a key role in creating an immersive audio experience, ensuring that sound effects and music are heard correctly from the perspective of the actor.

Functionality of the Audio Listener

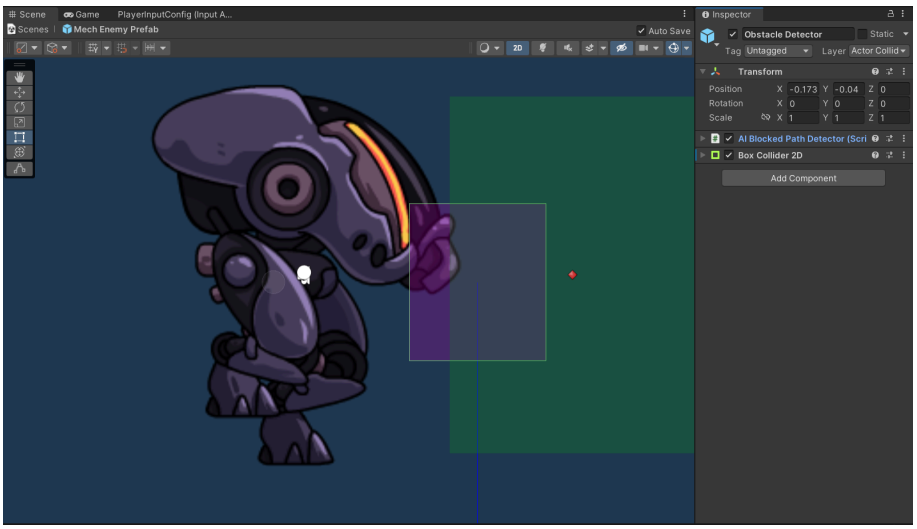
Audio Reception:
The Audio Listener acts as the 'ear' of the game world. It receives and processes audio signals from various sources in the game, such as sound effects, background music, and environmental noises.

Perspective-Based Audio:
The Audio Listener is typically attached to the main character. This positioning ensures that audio playback is perspective-appropriate, changing as the actor moves through the game environment.

Single Instance Requirement:
Unity requires that there be only one active Audio Listener in the scene at any given time. If the Actor Prefab is used for multiple characters, care must be taken to ensure that only the main character or player-controlled character has an active Audio Listener.

3.13 Obstacle Detector Game Object

The Obstacle Detector Game Object, situated within the Actor Prefab, is a critical component for AI behavior in the game. It enables AI characters to detect obstacles in their path, enhancing their navigation and interaction with the game environment.



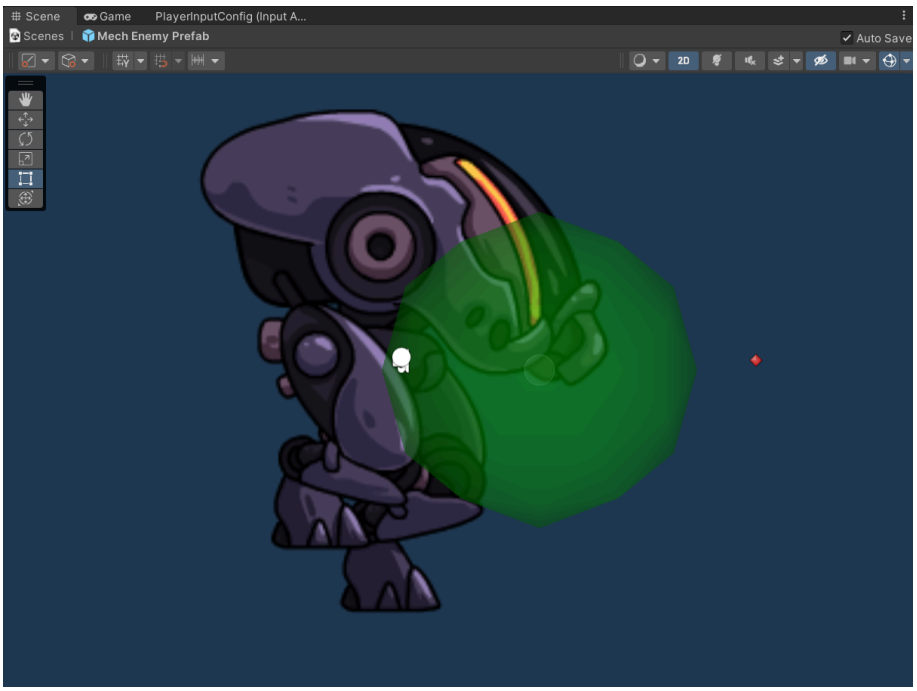
Components of the Obstacle Detector Game Object

AIBlockedPathDetector:
This script is essential for AI-controlled characters to detect if their path is blocked.
It uses a Box Collider 2D to determine if there are obstacles directly in front of the AI character. The script includes parameters such as groundMask and groundRaycastLength to configure the detection range and the types of surfaces or objects it can detect.
When a blocked path is detected, it triggers the OnPathBlocked event, which can be used to make AI characters change direction or perform other actions.
Gizmo parameters are provided for visualizing detection areas in the Unity Editor, aiding in debugging and setup.

Box Collider 2D:
Used in conjunction with the AIBlockedPathDetector script for collision detection.
Configurable to match the size and shape of the AI character, ensuring accurate obstacle detection.

3.14 Melee Range Detector Game Object

The Melee Range Detector Game Object, positioned within the Actor Prefab, is a crucial element for AI behavior in melee combat scenarios. It is designed to detect when a player or other target is within the range of a melee attack, enabling AI characters to initiate their attack sequences effectively.



Functionality of the Melee Range Detector

Detection Mechanism:
The Melee Range Detector uses a collider component, typically a Box Collider 2D or Circle Collider 2D, to establish a detection zone around the AI character.

AI Behavior Trigger:
When a player or target enters this detection zone, the AI character is alerted to their presence, and if conditions are met, initiates a melee attack.

Adjustable Range:
The size of the collider can be adjusted to represent different ranges for melee attacks, accommodating various types of AI characters and their specific attack styles.

3.15 Pick Up Feedback Game Object

The Pick Up Feedback Game Object, part of the Actor Prefab, is designed to enhance the player's experience by providing audio feedback when collecting items in the game. This Game Object plays a crucial role in creating a more immersive and responsive gameplay environment.

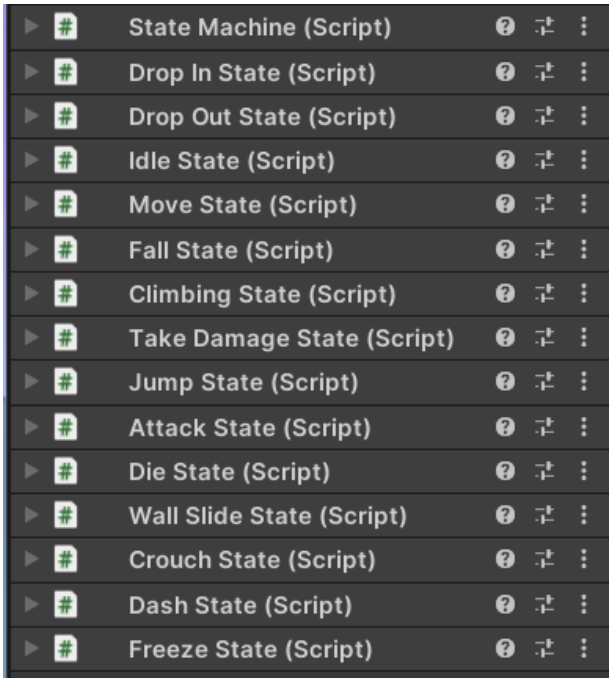
Components of the Pick Up Feedback Game Object

Audio Source:
This component is used to play sound effects.
It can be configured with various audio clips to represent different types of item pickups.
Volume, pitch, and other audio properties can be adjusted to suit the desired feedback effect.

Pick Up Feedback Script:
This script controls when and how the audio feedback is triggered.
It listens for events or interactions related to item pickups and triggers the appropriate sound effect through the Audio Source. The script can be customized to handle different types of pickup events, such as collecting health, currency, or power-ups.

3.16 States Game Object

The States Game Object, integral to the Actor Prefab, serves as the state machine handling all different states of the actor. This Game Object is crucial for managing the various behaviors and actions of the actor based on the gameplay context.



Components of the States Game Object

StateMachine Script:
Central component that orchestrates the transition between different states of the actor.
Manages state logic and ensures appropriate responses to in-game events or conditions.

Drop In State Script:
Handles the level start animation, setting the initial state of the actor when a level begins.

Drop Out State Script:
Manages the level over animation, defining how the actor behaves when a level ends.

Idle State Script:
Governs the actor's behavior when in an idle state, typically when no input is received.

Move State Script:
Controls the actor's movements, responding to player inputs for walking or running.

Fall State Script:
Manages the behavior of the actor when falling, such as during jumps or when walking off platforms.

Climbing State Script:
Governs actions and animations related to climbing, such as ladders or climbable surfaces.

Take Damage State Script:
Handles the actor's response to taking damage from enemies or environmental hazards.

Jump State Script:
Manages the jumping mechanics, including animations and movement while in the air.

Attack State Script:
Controls the actor's attack actions and animations, responding to attack inputs.

Die State Script:
Manages the actor's behavior upon dying, including death animations and game over logic.

Wall Slide State Script:
Governs the actor's interactions with walls, such as wall sliding or sticking to surfaces.

Crouch State Script:
Handles the crouching actions and animations, typically in response to player input.

Dash State Script:
Manages the dashing action, providing quick bursts of speed or movement.

Freeze State Script:
Handles the actor's response to being frozen, typically by enemy attacks or environmental effects.

3.17 Feedback Game Objects

Within the States Game Object of the Actor Prefab, a collection of Feedback Game Objects exists. These Game Objects are designed to provide specific feedback for different actions or states, such as movement, taking damage, death, attacks, and more. This feedback system is integral to enhancing the player's experience by offering immediate and context-specific responses to in-game actions.

Components of Feedback Game Objects

Movement Feedback:
Provides visual or audio feedback for movement actions like walking or running.
Enhances the perception of motion and adds realism to the actor's movements.

Take Damage Feedback:
Triggers feedback when the actor takes damage.
This could include visual effects like flashing or audio cues to indicate harm.

Death Feedback:
Activated upon the actor's death.
May include specific animations, sound effects, or other visual cues to signify defeat.

Attack Feedback:
Provides immediate response to attack actions, such as sound effects or visual impacts.
Reinforces the sense of power and impact of the actor's attacks.

Player Audio Feedback:
Contains an Audio Source component.
Plays various sound effects corresponding to different actions or states of the actor.

Jump Feedback:
Activates when the actor jumps.
Could include sound effects or visual elements to emphasize the action.

Fall Feedback:
Provides feedback for falling actions, enhancing the sense of gravity and descent.

3.18 Behaviors

The Behaviour Object, attached to enemy-type actors, adds AI functionality to these characters. It consists of various scripts that define distinct behaviors, including melee attacks, patrol patterns, player detection, and more. This object is essential for creating dynamic and intelligent enemy behaviors in the game.

Components of the Behaviour Object

AIBehaviourPatrol:
Handles basic patrol behavior of AI characters.
Includes a path detector and movement vector for navigating the environment.

AIBehaviourPatrolPath:
Manages patrol behavior along a defined path.
Utilizes patrol points and wait times to create more structured patrol routines.

AIBehaviourMeleeAttack:
Governs melee attack actions of AI characters.
Uses a melee attack detector to initiate attacks when the player is within range.

AIShootPlayer:
Handles shooting behavior directed at the player.
Typically includes logic for aiming and firing at the player's position.

AIMeleeAttackDetector:
A specialized detector for identifying when a player is within melee attack range.
Triggers melee attacks based on player proximity.

AIPlayerDetector:

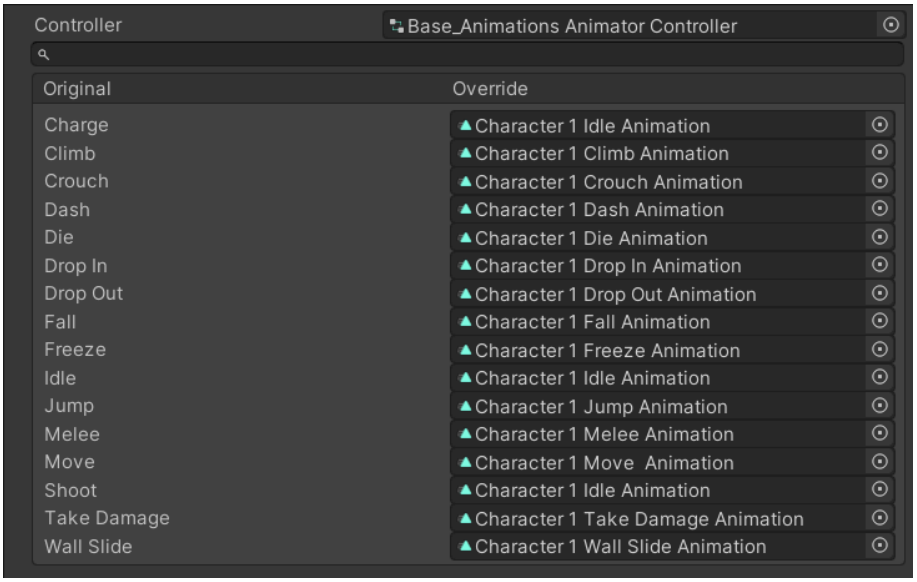
Detects the presence of the player within a certain range or field of view.
Essential for initiating chase or attack behaviors.

AIBehaviourPatrolBounce:
Manages patrol behavior with a bouncing movement pattern, adding variety to AI movement.

AIBehaviourPatrolPath:
Enables AI characters to follow a specific path, enhancing patrol behaviors and making them more predictable for strategic gameplay.

4. Actor Animation Utilization

In the 2D Action Platformer Kit, both playable actors and non-playable actors utilize animations through an Animator Override Controller. This system is based on a base Animator Controller located in the "Animations/Base Animations" folder, allowing for streamlined and customizable animation implementation across different actors.



Animator Override Controller System

Base Animator Controller:
Located in the "Animations/Base Animations" folder, this controller includes the animations for all standard states an actor might experience (e.g., idle, walking, jumping, attacking).
Serves as the foundation for animation behaviors across different characters.

Animator Override Controller:
Each actor uses an Animator Override Controller that is based on the base Animator Controller. This system allows for the customization of animations for each actor while maintaining a consistent animation structure. Animations for specific states can be easily overridden to suit the unique characteristics or aesthetics of different characters.

Assignment in Actor Prefab:
The Animator is assigned within the Actor Prefab, specifically in the 'Character' Game Object.
This setup ensures that each actor has its animation controller, allowing for independent animation control and customization.

Weapon Animations:
For weapon-specific animations, there is an additional Animator component located within the 'Weapon' Game Object of the Actor Prefab. This Animator handles animations related to weapon usage, such as swinging or firing, allowing for more detailed and weapon-specific animation behaviors.

Customizing Animations

The Override Controller system enables developers to easily substitute different animations for various actor states without altering the underlying animation logic.
Custom animations can be created and assigned to specific actions or states to give each character a unique look and feel.

5. Managers

5.1 Game Manager

The Game Manager Prefab in the 2D Action Platformer Kit is a central component responsible for managing various aspects of the game's overall state and flow. It contains the Game Manager script, which is crucial for coordinating different game systems and maintaining the game's state across sessions.

Functionality of the Game Manager Script

Save Game Data:

The Game Manager script includes a method to save game data, such as player progress, character selections, and in-game settings. This functionality is essential for ensuring that player progress is not lost between gaming sessions.

Load Saved Data:
It also manages the loading of saved game data, restoring the game state to where the player last left off. This includes re-establishing character status, level progress, and other relevant game settings.

Checkpoint Management:
The Game Manager handles the creation and deletion of checkpoints within the game.
It ensures that players can resume their progress from specific points in the game, providing a seamless gaming experience.

Weapon Data Management:
The script includes functions to remember and clear current weapon data lists, maintaining the state of the player's inventory and available weapons.

Destroy Game Manager:
A method is provided to safely destroy the Game Manager instance, typically used when transitioning between scenes or closing the game.

5.2 Character Manager

The Character Manager Prefab in the 2D Action Platformer Kit serves as a central hub for managing playable characters. It is designed to hold prefabs of all available characters and to spawn the specific characters chosen by the players into the game.

Functionality and Structure

Character Holding:

- The Character Manager Prefab contains references to all the prefabs of the playable characters in the game. It acts as a repository, allowing easy access to any character prefab based on player selection.

Character Selection and Spawning:

- When a player selects a character, the Character Manager identifies the corresponding prefab and spawns it into the game. This mechanism ensures that players can choose their preferred character before each playthrough.
- For multiplayer scenarios, the Player 2 character is selected and spawned using the Player 2 Characters list.

Index Tracking:

- The Character Manager maintains an 'index' to track which characters are currently active or selected. This index is saved within the game's save system, allowing the game to remember the players' character choices across sessions.
- Both Player 1 and Player 2 characters are selected using the same index. For example, if the index is set to `0`, the character from position `0` in the primary character list is used for Player 1, and the character from position `0` in the Player 2 Characters list is used for Player 2.

Save System Integration:

- The chosen characters' indices are stored in the Save System. This integration ensures continuity in the players' experience, maintaining their character choices even after closing and reopening the game.

Scalability for Multiple Characters:

- The Character Manager is designed to handle any number of playable characters, offering scalability and flexibility for game expansion.

Player 2 Characters List:

- In addition to the primary character list, the Character Manager includes a Player 2 Characters list.
- This list is populated with the child objects of the Character Manager.
- Each child object in this list must include a Player Input Manager component.
- The Player Input Manager component’s Player Prefab property should be set to a Player 2 character prefab, ensuring proper functionality for Player 2.

By linking Player 1 and Player 2 characters to the same index, the Character Manager provides a streamlined way to ensure both players are matched with corresponding characters for gameplay.

5.3 Item Manager

The Item Manager Prefab is a crucial component in the 2D Action Platformer Kit, designed to manage and organize the various items available within the game. It contains the Item Manager script, which plays a key role in item data handling and accessibility.

Functionality of the Item Manager Script

Item List Management:
The Item Manager script holds a List<ItemData> that contains all the item types available in the game. This list serves as the central repository for item information, including their properties and states.

Item Dictionary Creation:
Upon initialization (Awake method), the script populates a dictionary (itemDictionary) using items from the itemList. Each item in the itemList is added to the itemDictionary using its name as the key. This allows for quick and efficient retrieval of item data based on item names.

AddToDictionary Method:
A dedicated method for populating the itemDictionary.
It iterates through each item in the itemList and adds it to the dictionary, ensuring all items are accessible through a key-value pair mechanism.

5.4 Weapon Manager

The Weapon Manager Prefab is an integral component within the 2D Action Platformer Kit, designed to handle the management and organization of various weapons within the game. It features the Weapon Manager script, which plays a pivotal role in managing weapon data and accessibility.

Functionality of the Weapon Manager Script

Weapon List Management:

The Weapon Manager script maintains a List<WeaponData> containing all weapon types available in the game. This list acts as a central inventory of weapons, including their properties and attributes.

Weapon Dictionary Creation:
Upon initialization (in the Awake method), the script populates a dictionary (weaponDictionary) with items from the weaponList.
Each weapon in the weaponList is added to the weaponDictionary using its name as the key, allowing for efficient retrieval of weapon data based on weapon names.

AddToDictionary Method:
A dedicated method for populating the weaponDictionary.
It iterates through each weapon in the weaponList and adds it to the dictionary, ensuring all weapons are accessible through a key-value pair mechanism.

5.5 Scene Manager

The Scene Manager Prefab in the 2D Action Platformer Kit is crucial for managing the stages in the game and their progression. It contains the Scene Manager script, which oversees the completion state of stages, the triggering of Unity events upon stage completion, and the visual display of completed stages in the Stage Select Screen.

Functionality of the Scene Manager Script

Stage Progression Management:
The Scene Manager script keeps track of which stages have been completed and which have not. It saves this information to ensure continuity in the player's game progression.

Unity Events on Stage Completion:
The script can trigger Unity events when a stage is completed. These events can be used for various purposes, such as unlocking new stages or providing rewards.

Visual Display of Completed Stages:
In the Stage Select Screen, Unity events are utilized to handle the visual indication of which stages have been completed, enhancing the player's understanding of their progress.

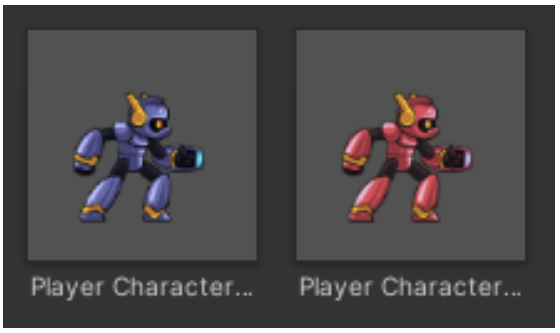
Event Handlers:

- On Level Over:
    - This event is triggered when the current stage is completed.
- On All Normal Stages Completed:
    - Handles the scenario when all regular stages have been finished, potentially unlocking final stages or other game features.
- On All Final Stages Completed:
    - Triggers when all final stages are completed, which could lead to the end-game sequence or bonus content.

6. Player Characters

6.1 Player 1

Player Character Prefabs in the 2D Action Platformer Kit are designed to provide a controllable and customizable player experience. These prefabs combine the Actor Prefab with the Player UI Prefab, creating a comprehensive character entity for players to control. The kit includes two different player characters, each with unique skills and stats, and supports the addition of any number of playable characters.



Components of Player Character Prefabs

Player Script:
Central script managing player-specific actions and behaviors. Coordinates the player's movements, interactions, and responses to game events.

Menu Controller Script:
Manages the in-game menu system, allowing players to access and interact with various game options and inventories.

Player Starting Weapon Script:
Determines and assigns the starting weapon for the player character. Essential for customizing the initial loadout of different player characters.

Player Input Helper Script:
Assists in processing and managing player inputs. Ensures smooth and responsive control over the player character's actions.

Player Input System Script:
Integrates with the new Unity Input System to capture and respond to player inputs.
Allows for flexible and customizable control mappings.

Player Input Component:

- A component that supports Unity's new Input System by linking the player input configuration with the player character.
- Input action maps trigger events that directly call respective movement functions from the Player Input System Script, ensuring efficient and responsive input handling.

Unique Characteristics

- The player characters are not equipped with AI behavior, ensuring that they are fully controllable by the player.
- Each player character can have distinct abilities, stats, and appearances, offering varied gameplay experiences.
- The characters are equipped with components that facilitate player-specific functionalities, such as inventory management and input handling.

Implementation Details

Character Customization:
Each player character can be customized to have different abilities and appearances, allowing for a diverse range of gameplay styles.

Input Responsiveness:
Ensure that the input system is finely tuned for responsive and intuitive player control.

UI Integration:

Seamlessly integrate the Player UI Prefab to display relevant information like health, inventory, and weapons.

6.2 Player 2

Player 2 characters function through the same Character Player Actor setup as Player 1 characters. Unity's Device Management System ensures seamless handling of multiple input devices, automatically binding the controls of a second player to a second input device. This system spawns the appropriate Player 2 character into the scene, which uses the same input actions as Player 1 but operates independently.

Functionality and Structure

Device Management:

- Unity detects and binds a second input device (e.g., a gamepad or keyboard) to Player 2.
- This ensures that both players can control their characters simultaneously without interference.

Player 2 Character Spawning:

- The Character Manager spawns the Player 2 character based on the index corresponding to the Player 1 character.
- For example, if Player 1 selects a character at index `0`, the corresponding Player 2 character at index `0` in the Player 2 Characters list will also be spawned.

Independent Input Actions:

- Player 2 uses the same input action mappings as Player 1 but remains independent. Each character responds only to its respective input device.

Player 2 Helper Script:

- This additional script is attached to the Actor Game Object of the Player 2 character.
- Key functionalities include:
    - **Game Manager Registration:** Automatically registers the instantiated Player 2 character with the Game Manager, ensuring proper tracking and management within the game.
    - **Camera Target Group Assignment:** Adds the Player 2 character to the camera's target group, allowing the camera to follow both players dynamically during gameplay.

6.3 Arena Player

Arena characters are a special type of actor designed specifically for arena gameplay. These characters utilize the Player Arena Helper script to initialize their positions within the arena and ensure proper camera integration through the camera's target group. Their setup is tailored for competitive arena-style gameplay with unique features that differentiate them from standard player characters.

Key Features

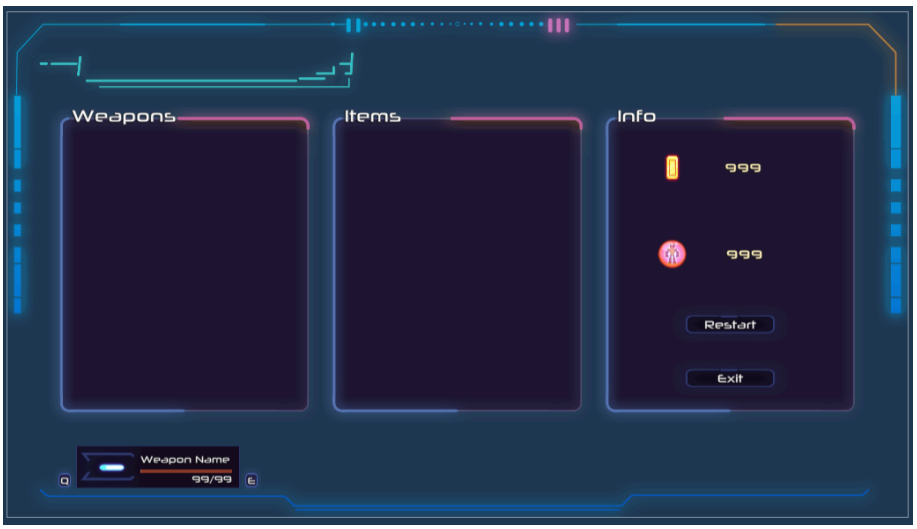
1. **Position Initialization**:
    - Arena characters automatically position themselves at predefined spawn points in the arena, such as `"Player 2 Start"`, using the Player Arena Helper script.
    - This ensures consistent and accurate placement of characters in each match.
2. **Camera Integration**:
    - The Player Arena Helper script dynamically assigns arena characters to the camera's target group.
    - This allows the camera to follow both characters seamlessly, providing a clear view of the arena and its participants.
3. **Weapon Setup**:
    - Weapons for arena characters are configured to inflict damage on both Player 1 and Player 2, enabling competitive gameplay.
    - This setup ensures fairness by allowing all characters to engage with and affect each other equally.
4. **Score Tracking**:
    - Each character is associated with a score-tracking system displayed in the UI.
    - When a character dies, the win counter of the opposing character is incremented. This provides a clear and immediate feedback loop for players to track their progress in the match.

Implementation Details

- **Player Arena Helper Script**:
    - Central to arena gameplay, this script manages position initialization and camera integration.
    - It is attached to each arena character and ensures seamless setup within the arena environment.
- **UI Integration**:
    - A dedicated score display uses a TextMeshPro component to show the win counter for each character.
    - The UI dynamically updates during gameplay, reflecting real-time events such as player eliminations.
- **Customization**:
    - Arena characters can be customized with unique abilities, stats, and weapons, offering varied gameplay experiences tailored to arena combat.

7. Player UI Prefab

The Player UI Prefab in the 2D Action Platformer Kit is an essential user interface element attached to a player-type actor. It provides players with crucial in-game information and interactive elements, enhancing their gameplay experience and interaction with the game.



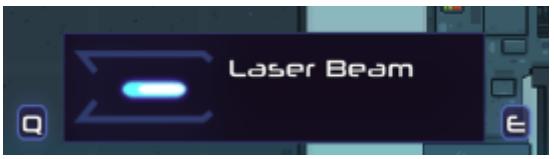
Components of the Player UI Prefab

Health Bar:

- Visually represents the player's current health.
- Decreases as the player takes damage and replenishes with health pickups or other recovery methods.

Weapon UI:

- Displays the currently equipped weapon.
- Can include weapon icons or other indicators to show active weapon status.



Menu:

Presents various options and information to the player, including:

- Weapon Inventory:
    - Shows all weapons collected by the player.
- Item Inventory:
    - Lists items the player has acquired.
- Collected Currency:
    - Indicates the amount of in-game currency the player has collected.
- Lives:
    - Displays the number of lives remaining.

Includes interactive elements:

- Restart Button:
    - Allows players to restart the current stage.
- Exit Button:
    - Enables players to exit the current stage and return to the Stage Select Screen.

Game Over UI:

Activated when the player runs out of lives or meets certain defeat conditions.

- Includes:
- Restart Button:

    - To restart the stage or from the last checkpoint.
- Exit Button:
    - To exit to the Stage Select Screen or main menu.

8. Enemies

8.1 Regular Enemies

The 2D Action Platformer Kit features various enemy types, each with unique behaviors and characteristics. These enemies enhance the game's challenge and diversity, requiring players to adapt their strategies accordingly.

Existing Enemy Types

Bouncing Enemy:

- Description: This enemy type exhibits bouncing behavior, adding unpredictability and dynamism to its movement.
- AI Behavior: Utilizes the AIBehaviourPatrolBounce behavior, allowing it to patrol an area with a bouncing motion.

Flying Enemy:

- Description: A flying enemy that patrols along a customizable path, posing an aerial threat to the player.
- AI Behavior: Employs AI Behaviour Shoot Player to engage in shooting at the player upon detection.
- Path Customization: Integrated with the Flying Path prefab and AIBehaviourPatrolPath for patrolling along a predefined route, offering flexible path design.

Mech Enemy:

- Description: A ground-based enemy that patrols horizontally, changing direction when encountering obstacles.
- AI Behavior: Uses AIBehaviourShootPlayer to shoot at the player when in range, combining patrolling with offensive capabilities.
- Movement Pattern: Its left-to-right patrol pattern is influenced by obstacles in its path, necessitating player anticipation and timing.

Road Wheel Enemy:

- Description: Similar to the Mech Enemy, this ground enemy patrols back and forth, reacting to obstacles.
- AI Behavior: Primarily focused on patrolling, it changes direction upon encountering obstacles, providing a consistent ground-level threat.

Turret Enemy:

- Description: A stationary enemy type that targets and shoots at the player when they come into range.
- AI Behavior: Capable of turning to face the player and engaging in shooting when the player is detected, making it a fixed point of danger in levels.

8.2 Boss Enemy

The Boss Enemy Prefab in the 2D Action Platformer Kit is designed to provide a challenging and memorable encounter for players. This prefab is a more complex and feature-rich version of a standard enemy, equipped with AI behavior and additional components to elevate the boss fight experience.

Components of the Boss Enemy Prefab

Actor with AI Behavior:
The core of the boss enemy is an Actor Prefab integrated with AI behavior scripts.
These scripts govern the boss's actions, movements, and attack patterns, providing a unique challenge compared to regular enemies.

Boss Health UI Canvas:
A dedicated canvas is included to display the boss's health bar.
This UI element provides players with visual feedback on the boss's remaining health, adding to the intensity and strategy of the battle.

Boss Trigger:
The Boss Trigger is a crucial component that initiates the boss fight.
Consists of a Box Collider 2D set as a trigger, which can be strategically placed in the level.

On Trigger 2D Enter Utility Script:
This script detects when the player enters the trigger area.
Upon player entry, it fires a Unity event to initiate the boss fight sequence, which may include playing boss music and displaying the boss health UI.

9. Scriptable Objects

9.1 Pickup Drop Table Scriptable Object

The Loot Table Scriptable Object in the 2D Action Platformer Kit is a customizable tool used to define the probability and types of items that enemies drop upon defeat. This Scriptable Object allows game designers to create varied and interesting loot drops, enhancing the player's experience with rewards for defeating enemies.

Structure of the Loot Table Scriptable Object

List of Droppable Items:
The Loot Table Scriptable Object contains a List<RewardItem> that defines all possible items that can be dropped. Each RewardItem in the list includes the item itself and its drop probability.

Drop Probability:
The drop probability for each item determines the likelihood of that item being dropped by an enemy. This system allows for the creation of rare and common drops, adding excitement and variability to loot collection.

GetRandomItem Method:
This method is used to randomly select an item from the loot table based on the defined probabilities. It uses a random number generator and the drop probabilities to select an item, ensuring that drops are both random and weighted according to their likelihood.

Implementation in Enemy Actors

Integration with Enemy Death Feedback:
The Loot Table Scriptable Object is typically integrated into the enemy actor's death feedback mechanism. Within the DropManager component of an enemy, the DropItem function is called, usually via a Unity event, when the enemy dies. This function then uses the Loot Table to determine and instantiate the dropped item.

9.2 Actor Data Scriptable Object

The Actor Data Scriptable Object in the 2D Action Platformer Kit is a powerful tool used to define and manage the stats and capabilities of both playable actors and enemy actors. This object allows for the customization of various attributes, such as health, movement, jumping, and climbing capabilities, which are crucial for character behavior and performance.

Components of the Actor Data Scriptable Object

Health:
Determines the health points of the actor. This value sets the maximum and initial health, impacting the actor's survivability in the game.

Movement Data:
Defines the movement characteristics of the actor, including speed and agility. Customizable parameters can include running speed, acceleration, deceleration, and other
ovement-related stats.

Jump Data:
Specifies the jumping capabilities of the actor, such as jump height and duration.
These settings influence how the actor interacts with the game environment, especially vertical navigation and obstacle traversal.

Climb Data:

Determines the actor's ability to climb objects like ladders or walls. Includes attributes like climb speed and stamina, defining how effectively the actor can navigate climbable surfaces.

Implementation and Usage

The Actor Data Scriptable Object is placed within the Actor component of the Actor Prefab.
For playable characters, it allows for the customization of the player's abilities and stats, tailoring the gameplay experience to different character designs or player preferences.
For enemy actors, it provides a way to set distinct attributes for various enemy types, creating diverse challenges and encounters in the game.

9.3 Item Scriptable Object

Item Scriptable Objects in the 2D Action Platformer Kit are designed to define and manage the various items that players can collect and use within the game. These objects allow for the customization of item characteristics, including appearance, quantity, and functionality.

Components of the Item Scriptable Object

Name:
Specifies the name of the item, which is displayed in the game's UI and inventory system.
The name is used for identification and reference throughout the game.

Sprite:
Defines the visual representation of the item as it appears in the game.
The sprite is crucial for player recognition and adds to the visual appeal of the item.

Max Amount:
Sets the maximum quantity of the item that a player can hold in their inventory.
This limit is important for inventory management and game balance.

Item Action Script (Interface):
A crucial component that determines the item's effect when used. The Item Scriptable Object is linked to an IItemActionScript interface, which contains the PerformAction() function.

Customizable Item Action:

Developers can customize the item's action by creating a custom script that implements the IItemActionScript interface. This customization allows for a wide range of item effects and interactions, from healing to granting special abilities.

PerformAction() Function:
The behavior of the item when used in the game is defined within the PerformAction() function. This function can be tailored to trigger specific effects, animations, or gameplay changes based on the item's intended purpose.

9.4 Melee Weapon Scriptable Object

The MeleeWeaponData Scriptable Object in the 2D Action Platformer Kit is designed to define and customize the abilities and characteristics of melee weapons. This object allows for detailed specification of various weapon attributes, enabling diverse melee combat experiences.

Components of the MeleeWeaponData Scriptable Object

Weapon Name:
Specifies the name of the melee weapon, used for identification in the game's UI and inventory.

Sprite:
The visual representation of the weapon in the game.
Important for player recognition and for matching the game's aesthetic.

Visibility Toggle (Bool Visible):
Determines whether the weapon is visible or invisible on the actor.
Provides flexibility in how the weapon is displayed during gameplay.

Hittable Layer:
Defines which layers in the game can be affected or hit by the weapon.
Essential for ensuring that the weapon interacts correctly with targets and environments.

Attack Range:
Specifies the range within which the weapon can hit targets. Critical for balancing the weapon's effectiveness in combat.

Normal Weapon Damage:
Sets the amount of damage the weapon inflicts under normal, uncharged conditions.

Charged Weapon Damage:
Defines the damage amount when the weapon is used in a charged state. Offers a strategic element to combat, allowing for more powerful attacks.

Weapon Swing Sound:
The sound effect played when the weapon is swung. Adds to the sensory feedback and immersion of using the weapon.

Weapon Energy:
Indicates the amount of energy the weapon possesses. Important for weapons that consume energy with each use.

Infinite Weapon Energy (Bool):
Determines whether the weapon should use energy for each attack or can be used infinitely. Allows for differentiation between energy-based and unlimited-use weapons.

Can Be Charged (Bool):
Specifies if the weapon has the capability to be charged for more powerful attacks.
Adds a layer of depth and strategy to the weapon's use in combat.

9.5 Range Weapon Scriptable Object

The RangeWeaponData Scriptable Object in the 2D Action Platformer Kit is designed for defining the capabilities and attributes of ranged weapons. It provides a comprehensive framework for specifying various properties of these weapons, from visual appearance to combat functionality.

Components of the RangeWeaponData Scriptable Object

Weapon Name:
Identifies the ranged weapon, used in the game's UI and inventory systems.

Sprite:

The visual representation of the weapon, essential for player recognition and visual consistency with the game's style.

Visibility Toggle (Bool Visible):
Allows for the weapon to be either visible or invisible on the actor, providing flexibility in its display during gameplay.

Hittable Layer:
Defines which layers in the game the weapon can interact with, ensuring correct targeting and impact.

Attack Range:
Specifies the effective range of the weapon, crucial for balancing its reach and effectiveness in combat.

Normal Weapon Damage:
Sets the damage inflicted by the weapon in its uncharged state.

Charged Weapon Damage:
Determines the damage level when the weapon is used in a charged state, offering a more powerful attack option.

Weapon Swing Sound:
The sound effect that plays when the weapon is used, enhancing the auditory experience of the weapon.

Weapon Energy:
Indicates how much energy the weapon has, relevant for weapons that consume energy with each use.

Infinite Weapon Energy (Bool):
Specifies whether the weapon uses energy per attack or can be used without energy constraints.

Can Be Charged (Bool):
Indicates if the weapon has the functionality to be charged for enhanced attacks.

Normal Range Weapon Prefab:
Defines the projectile instantiated when the weapon is used in its normal, uncharged state.

Charged Range Weapon Prefab:
Specifies the projectile type for charged attacks, allowing for different effects or behaviors.

Weapon Throw Speed:
Sets the speed at which projectiles are thrown, impacting how quickly they reach their target.

10. Pickups

10.1 Currency Pickup

The Currency Pickup in the 2D Action Platformer Kit is a game object designed to be collected by players throughout various stages. It is a key component for enhancing the gameplay experience by providing players with in-game currency that can be used for various purposes.

Components of the Currency Pickup

Box Collider 2D:
Enables the Currency Pickup to detect collisions with the player. Configured to trigger an event when the player interacts with it, allowing the player to collect the currency.

CurrencyPickup Script:
This script is the core of the Currency Pickup functionality.

- Pick Up Method: When a player interacts with the Currency Pickup, the script's PickUp method is invoked. This method then accesses the PlayerCurrency component of the player object and adds a specified amount of currency to the player's total.

- Currency Value: The amount of currency that each pickup adds to the player's total is customizable and is set using a serialized field in the script. The default value is 1 currency unit per pickup.
- Unity Event Integration: The script includes a UnityEvent, OnPickUp, which can be used to trigger additional actions or effects when the currency is collected, such as playing a sound effect or visual animation.

10.2 Health Pickup

The Health Pickup in the 2D Action Platformer Kit is an interactive game object that can be placed in stages for players to collect and regain health. It comes in two predefined versions: Large (L) for healing 3 HP and Small (S) for healing 1 HP.

Components of the Health Pickup

Rigidbody 2D:
Provides physics properties to the Health Pickup, allowing it to interact with other physical objects and forces within the game environment.

Box Collider 2D:
Defines the physical boundaries of the Health Pickup for collision detection.
Ensures that interactions with the player, such as collecting the pickup, are accurately detected.

Health Pickup Script:
The core script that manages the pickup's behavior.

- Pick Up Method: When a player interacts with the Health Pickup, this method is called, and the player's Damagable component is accessed to add health.
- Health Value: The script contains a variable representing the amount of health restored. This value is set to 3 for the Large version and 1 for the Small version of the Health Pickup.

On Trigger Enter 2D Utility Script:
This script triggers the pickup event when the player enters the collider, enabling the pickup action.

Stop Rigidbody 2D Movement Utility:
Used when the pickup is dropped by an enemy to stop its movement upon touching the ground. This utility prevents the Health Pickup from continuously moving or falling, making it easier for players to collect.

10.3 Weapon Energy Pickup

The Weapon Energy Pickup in the 2D Action Platformer Kit is a collectible item that players can use to replenish the energy of their weapons. It is strategically placed in stages and comes in two predefined variations: large (L) and small (S), restoring different amounts of weapon energy.

Components of the Weapon Energy Pickup

Rigidbody 2D:
Provides physics properties to the pickup, enabling interactions with other physical objects and forces within the game environment.

Box Collider 2D:
Defines the physical boundaries of the pickup for collision detection, allowing the player to collect it upon contact.

Weapon Energy Pickup Script:
Manages the functionality of the pickup, adding weapon energy to the player's current weapon.

- Pick Up Method: When a player collects the pickup, this method is called. It retrieves the player's current weapon and adds the specified amount of weapon energy to it. If the added energy exceeds the weapon's maximum capacity, it is set to the maximum allowed amount.
- Energy Amounts: The script is configurable for customizable energy amounts: the large (L) variant typically restores 10 WP, while the small (S) variant restores 5 WP.
- Unity Event Integration: The script includes a UnityEvent, OnPickUp, which can be used to trigger additional actions or effects when the lives are collected, such as playing a sound effect or visual animation.

On Trigger Enter 2D Utility Script:
Detects when the player enters the pickup's collider and triggers the collection process.

Stop Rigidbody 2D Movement Utility:
Stops the pickup from moving when it hits the ground, particularly useful when the pickup is dropped by an enemy. This ensures the pickup stays in place for the player to collect.

10.4 Lives Pickup

The Lives Pickup in the 2D Action Platformer Kit is a critical game object designed to be collected by players. It helps enhance gameplay by providing players with extra lives, thereby increasing their chances of success in the game.

Components of the Lives Pickup

Box Collider 2D:
Facilitates the detection of collisions with the player. It is configured to trigger an event when the player interacts with it, allowing the player to collect the extra lives.

Lives Pickup Script:
Central to the Lives Pickup's functionality.

- Pick Up Method: When the player interacts with the Lives Pickup, the script's PickUp method is invoked. This method then accesses the PlayerLives component of the player object and adds a specified number of lives to the player's total.
- Lives Value: The amount of lives added is customizable and set using a serialized field in the script. The default value is 1 life per pickup.
- Unity Event Integration: The script includes a UnityEvent, OnPickUp, which can be used to trigger additional actions or effects when the lives are collected, such as playing a sound effect or visual animation.

10.5 Tank Pickups

Tank Pickups in the 2D Action Platformer Kit are valuable items that can be purchased from the in-game shop or found within stages. These tanks are crucial for replenishing the player's health and weapon energy by consuming them through the menu. There are three types of tanks: Health Tank, Weapon Energy Tank, and Mega Tank, each offering different restoration benefits.

Components of Tank Pickups

Rigidbody 2D:
Provides physics properties to the tank pickups, allowing them to interact with the game's physical environment. Essential for implementing realistic behaviors, such as falling or colliding with other objects.

Box Collider 2D:
Defines the physical boundaries of the tank for collision detection.
Ensures that interactions with the player, such as collecting the tank, are accurately detected.

Item Pickup Script with Scriptable Object:
The core script that manages the pickup logic of the tanks. It references a Scriptable Object that defines the specific properties and behaviors of each tank type.
This setup allows for easy customization and management of different tank types within the game.

On Trigger Enter 2D Utility Script:
Detects when the player enters the collider of the tank, triggering the pickup process.

Ensures that the player can collect the tank upon contact.

Stop RB 2D Movement Utility:
Stops the tank's movement when it touches the ground, especially useful when the tank is dropped by an enemy. Prevents the tank from continuously falling or moving away from the player's reach.

Types of Tanks and Their Functions

Health Tank:
Completely replenishes the player's health when used. Vital for survival in challenging stages or after intense combat.

Weapon Energy Tank:
Fully restores all weapon energy, enabling continued use of energy-consuming weapons. Important for maintaining the player's combat effectiveness.

Mega Tank:
Provides complete restoration of both health and weapon energy.
The most valuable tank, offering significant recovery and strategic advantages.

10.6 Weapon Pickups

Weapon Pickups in the 2D Action Platformer Kit are interactive items that players can collect to acquire new weapons. These pickups are a crucial part of the game's combat mechanics, offering players a variety of weapons to enhance their offensive capabilities.

Components of Weapon Pickups

Box Collider 2D:
This component enables the weapon pickup to detect collisions, specifically when a player comes into contact with it. The collider's dimensions and shape can be adjusted to match the visual representation of the weapon.

Weapon Pickup Script with Scriptable Object:
The key script that handles the logic of picking up a weapon. It references a Scriptable Object that contains the weapon's data, such as its type, stats, and any unique properties. This structure allows for easy management and customization of different weapons within the game.

Updated Functionality

The **Weapon Pickup** script now supports an advanced system for handling different weapon pickups for Player 1 and Player 2:

1. **Single Weapon Data for Both Players**:
    - If only the primary `WeaponData` is set and the `AlternateWeaponData` is not provided, both Player 1 and Player 2 will receive the same weapon.
2. **Different Weapon Data for Each Player**:
    - If both `WeaponData` and `AlternateWeaponData` are set:
        - Player 1 (tagged `"Player"`) will receive the weapon defined in `WeaponData`.
        - Player 2 (tagged `"Player 2"`) will receive the weapon defined in `AlternateWeaponData`.

Sprite Renderer:
Responsible for rendering the visual appearance of the weapon pickup in the game.
This component can be customized with different sprites to visually represent various weapon types available for pickup.

Predefined Weapons Included in the Kit

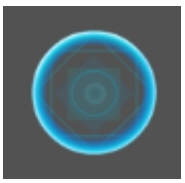
- Freeze Weapon: Allows players to freeze enemies or objects, potentially opening up new pathways or combat strategies.

- Jelly Weapon: A unique weapon with specific properties that can vary from standard offensive capabilities.

- Light Saber Weapon: A classic, high-energy weapon for close combat.

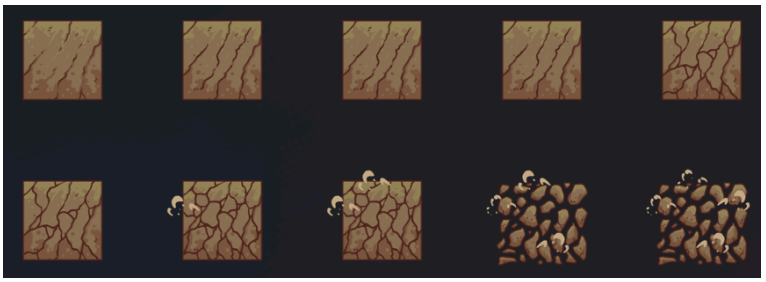
- Red Light Saber Weapon: A variant of the Light Saber with potentially different attributes or effects.

- Shield Weapon: Offers defensive capabilities, allowing players to block or deflect incoming attacks.



11. Breakable Block

The Destructible Block is a versatile game object in the 2D Action Platformer Kit, designed to add interactive elements to game levels. It can be destroyed by the player or other game mechanics, creating dynamic environments and adding a layer of strategy to gameplay.



Components of the Destructible Block

Sprite Renderer:
Renders the visual appearance of the block in the game.
Can be customized with different textures or sprites to match various level designs.

Animator:
Manages the animations of the block, such as breaking or crumbling effects when it is destroyed. Allows for visually appealing destruction sequences that enhance the player's experience.

Rigidbody 2D:
Provides the block with physics properties, enabling it to interact with other physical objects and forces in the game. Essential for implementing realistic destruction and collision behaviors.

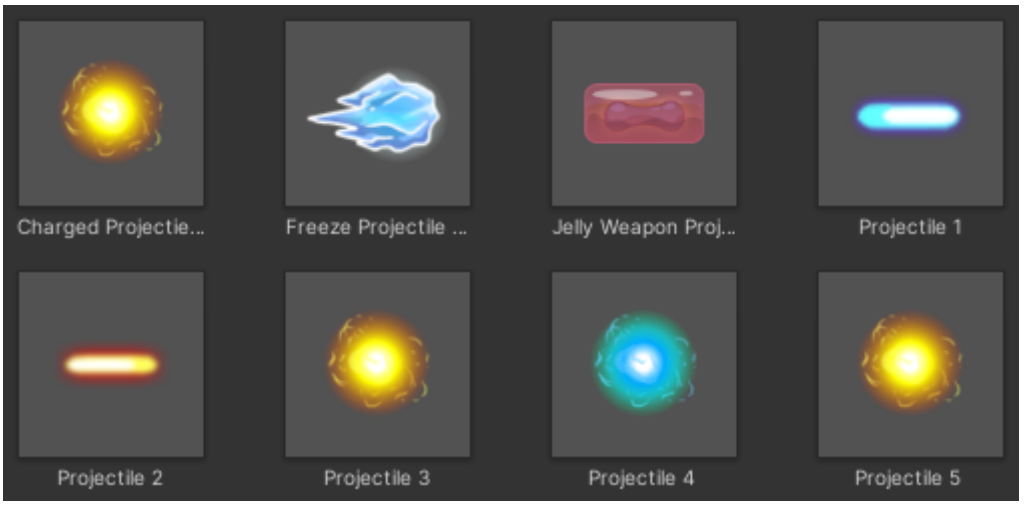
Box Collider 2D:
Defines the physical boundaries of the block for collision detection.
Ensures that interactions with the block, such as hits or collisions, are detected accurately.

Audio Source:
Plays sound effects associated with the block, like the noise of breaking or crumbling. Enhances the sensory feedback of the block's destruction, adding to the immersion.

Destroyable Block Script:
The core script that handles the block's destructibility logic. Manages how and when the block is destroyed, including triggering animations, sound effects, and any gameplay effects resulting from the block's destruction.

12. Projectile Prefabs

Projectile Prefabs in the 2D Action Platformer Kit are essential for ranged combat mechanics, allowing characters to fire projectiles that interact with actors and objects in the game environment. These projectiles come in various types, each with unique functionalities and effects.



Components of Projectile Prefabs

Rigidbody 2D:
Provides the projectiles with physics properties, enabling them to move and interact within the game's physical world. Essential for realistic projectile motion, including flying through the air and colliding with targets.

Projectile Script:

The core script managing the behavior of the projectile. Controls aspects like speed, direction, damage, and what happens upon hitting a target or object. Determines the lifespan of the projectile and how it interacts with different game elements.

Customization with Additional Scripts:
Projectiles can be customized with additional scripts to provide various functionalities:

- Freeze Projectile: Equipped with a 'Freeze Actor' script, this projectile can freeze enemies or objects upon impact, adding a tactical element to combat.
- Jelly Projectile: Comes with a 'Jelly Weapon' script, which imparts a bouncing effect to objects hit, creating dynamic interactions in the game environment.

Types of Projectiles

- Regular Projectiles: These are standard projectiles used for basic ranged attacks. They fly towards a target and can cause damage or other effects upon impact.
- Specialized Projectiles: These include projectiles like Freeze and Jelly, each designed to offer unique effects that can influence combat strategy and level navigation.

13. Checkpoint System

The Checkpoint System in the 2D Action Platformer Kit is a crucial feature that enhances gameplay by providing players with specific restart points within stages. Checkpoints are strategically placed to ensure a fair and enjoyable game progression.

Structure and Functionality of Checkpoints

Placement in Stages:
Checkpoints are placed at key locations within stages where it is deemed appropriate for players to restart after dying. The placement should consider the level's difficulty, ensuring that players do not have to replay excessively challenging sections repeatedly.

Initial Checkpoint - Player Start:
The first checkpoint in any stage is always the player's starting point. This checkpoint serves as the initial respawn location when the stage begins.

Box Collider 2D as Trigger:

Each checkpoint includes a Box Collider 2D component set to act as a trigger.
When the player character enters this collider, the checkpoint is activated, marking it as the new respawn point.

Checkpoint Activation:
Upon activation, the checkpoint saves the player's progress up to that point.
If the player character dies after activating a checkpoint, they will respawn at that location instead of starting the stage over.

14. Screen Transition Manager

The Screen Transition Animation feature in the 2D Action Platformer Kit is an integral part of the SceneManager Prefab. It enhances the player's experience by providing smooth visual transitions between different scenes or stages within the game. This system employs animations for fading the screen in and out, creating a seamless and polished change of scenes.

Structure and Functionality

Integration with Scene Manager Prefab:
The Screen Transition Manager is a component of the SceneManager Prefab.
It is responsible for handling the visual transition animations whenever a new scene is loaded.

Transition Animations:
The system includes animations for both fading in and fading out of the screen.
These animations are played at the beginning and end of scene transitions, respectively.

Animator Controller:
Located in the "Animations/Screen Transition Animations" folder, there is a base Animator Controller. This controller acts as the foundation for the transition animations.

Animator Override Controller:
The base Animator Controller is used in conjunction with an Animator Override Controller. This allows for the introduction of customized fade-in and fade-out animations, making it possible to tailor the transition effects to match the game's style or thematic needs.

Customization of Transitions

The use of an Animator Override Controller enables developers to easily replace the default animations with custom ones, providing flexibility in the visual style of scene transitions. Custom animations can be created and assigned to the fade-in and fade-out actions, offering a unique transition experience for different parts of the game.

15. Input System

The 2D Action Platformer Kit leverages Unity's new Input System to offer a modern and flexible approach to handling player inputs across various platforms, including keyboard and gamepad. The configuration for these inputs is centralized in the Input/_Input 1/"PlayerInputConfig" file, facilitating easy adjustments and expansions to input mappings.

Action Maps and Actions

Action Map: Player Movement

This action map is dedicated to controlling the player character, encompassing all necessary actions for movement and interaction within the game environment.

- **Move Actor:** Manages the movement of the player character using both keyboard and gamepad.
- **Jump:** Allows the player character to jump, supported by keyboard and gamepad inputs.
- **Enter Menu:** Opens the in-game menu, accessible via both keyboard and gamepad.
- **Attack:** Initiates the player character's attack action with keyboard and gamepad.

- **Dash:** Enables the player character to dash, using keyboard and gamepad controls.
- **Previous Weapon:** Switches to the previous weapon in the inventory with keyboard and gamepad.
- **Next Weapon:** Switches to the next weapon in the inventory with keyboard and gamepad.

Action Map: Pause Menu

Handles controls relevant when the in-game menu is open, effectively disabling character movement.

- **Exit Menu:** Allows exiting the in-game menu, utilizing keyboard and gamepad inputs.

Action Map: Button Press Events

Focuses on controls used to navigate menus outside of the active gameplay, such as the main menu or settings.

- **Confirm:** Executes confirm actions within menus, supported by keyboard and gamepad.
- **Cancel:** Executes cancel actions or backs out of menus, supported by keyboard and gamepad.

16. Mobile Support: Enabling On-Screen Buttons for Touch Controls

The 2D Action Platformer Kit is designed with cross-platform development in mind, offering full support for mobile devices through touch controls. To facilitate an intuitive and responsive gameplay experience on mobile platforms, the kit includes a specially designed set of on-screen buttons.

Configuring Touch Controls

Player UI Prefab: On-Screen Buttons

Embedded within the player UI prefab is a GameObject called "On Screen Buttons." This GameObject houses all the necessary buttons that replicate the actions available to players, such as moving, jumping, attacking, dashing, and navigating through weapons or menus.

Activation for Mobile Devices

To adapt the game for mobile devices equipped with touch input, the "On Screen Buttons" GameObject can be easily activated. This process involves:

- Navigating to the player UI prefab within your project's asset structure.
- Locating the "On Screen Buttons" GameObject.
- Setting the GameObject to active within the Unity Editor.

Upon activation, the on-screen buttons will become visible and interactive in the game, providing a seamless way for players to control the game on touch-screen devices. These buttons are designed to mirror the actions configured in the Unity Input System, ensuring that the gameplay experience is consistent across different platforms.

Customization and Scalability

The on-screen buttons are fully customizable to match the needs of your game and the preferences of your target audience. You can adjust the size, position, and appearance of these buttons within the Unity Editor to ensure they blend seamlessly with your game's aesthetic and user interface design.

Furthermore, should your game require additional actions beyond the default set, you can easily extend the "On Screen Buttons" GameObject by adding new buttons and linking them to the corresponding actions in the Input System.

17. Technical

17.1 State Pattern in the 2D Action Platformer Kit

The State Pattern is a behavioral design pattern that is particularly useful in game development for managing complex state changes in game entities, such as characters or enemies. It allows an object to change its behavior when its internal state changes, thus

appearing to change its class. This pattern is encapsulated within the 2D Action Platformer Kit to manage various states of game characters seamlessly.

Components of the State Pattern

Context (Actor):

- The primary entity that exhibits different behaviors based on its internal state, such as a player character or an enemy.
- Maintains a reference to a state object that represents its current state.
- Delegates state-specific behaviors to the current state object.

State (Abstract Class or Interface):

- Defines a common interface for all concrete states.
- Each specific state implements this interface to provide state-specific behaviors.

Concrete States:

- Classes implementing the State interface to provide behaviors for different states.
- Examples include IdleState, JumpState, AttackState, each managing behavior associated with a state of the context.

Transitions:

- Triggered by game events or conditions, such as user input or in-game triggers.
- The context changes its state based on these transitions.

17.2 Interface Scripts in the 2D Action Platformer Kit

In the 2D Action Platformer Kit, interfaces are used to define contracts for various behaviors and functionalities. These interfaces ensure that different game components, such as characters, items, and AI, conform to a standard structure, making the code more organized and easier to manage. Below is an explanation of how these interface scripts are utilized within the toolkit.

Key Interfaces and Their Usage

ISavingLoading:

- Purpose: Defines methods for saving and loading data.
- Methods:
    - SaveData(): Saves relevant data, such as player progress or game settings.

    - LoadData(): Loads saved data.
- Implementation: Used by classes that handle data persistence, ensuring a consistent approach to saving and loading game data.

IItemActionScript:

- Purpose: Provides a framework for actions that can be performed with items.
- Method:
    - PerformAction(Actor actor, int itemIndex): Executes the action associated with an item.
- Implementation: Applied to scripts attached to items, enabling diverse item-specific actions, such as using health potions or equipping weapons.

IDamageable:

- Purpose: Defines a standard for objects that can take damage.
- Method:
    - TakeDamage(Game Object opponent, int damageAmount): Applies damage to the object.
- Implementation: Used by player characters, enemies, and destructible objects, allowing for a unified way to handle damage across various game elements.

IAIDetector:

- Purpose: Sets a guideline for AI detection mechanisms.
- Method:
    - GetDetectorValue(): Retrieves the current detection status.
- Implementation: Integral to AI scripts for detecting players or other entities, facilitating AI responses based on detection.

IActorInput:

- Purpose: Standardizes the input handling for actors.
- Properties and Events:
    - MovementVector: Represents the directional input for movement.
    - Events like OnAttackPressed, OnJumpPressed, etc., triggered by specific player inputs.
- Implementation: Ensures a consistent input response system for player-controlled characters, streamlining input processing across different character types.

IWeapon:

- Purpose: The IWeapon interface is created to define the essential behaviors and properties that all weapon types in the game must possess. This includes methods for using the weapon, handling its activation and deactivation, and other weapon-specific actions.
- Key Methods and Properties:
    - Attack(): This method is responsible for executing the weapon's primary action, which usually involves inflicting damage on targets. It might also include animations, sound effects, and other visual feedback.
    - Equip(): Method to handle the weapon being equipped by a character. This can involve setting up initial states, animations, or other properties.
    - Unequip(): Method to manage the weapon being unequipped. This might involve deactivating certain states or visual effects.
    - GetDamage(): A property or method that returns the damage value associated with the weapon. This is crucial for calculating the impact of the weapon on targets.
- Implementation:Various classes representing different types of weapons (such as swords, guns, etc.) implement the IWeapon interface. Each class provides its own specific implementation of the interface's methods, ensuring that each weapon behaves according to its unique characteristics while adhering to the overall weapon framework.

Advantages of Using Interfaces

- Consistency: Interfaces enforce a uniform method structure, ensuring that all implementing classes adhere to a defined contract.
- Flexibility: Allows different objects to implement the same interface in unique ways, providing versatility in gameplay mechanics.
- Interchangeability: Facilitates the use of different objects interchangeably in the code, as long as they implement the same interface.
- Maintainability: Makes the codebase more organized and manageable, simplifying future modifications or extensions.

17.3 Documentation for SaveSystem Script in 2D Action Platformer Kit

The SaveSystem script is a critical component of the 2D Action Platformer Kit, designed to manage the saving and loading of game data. This system is responsible for preserving

player progress, including player stats, inventory, and game states, which can be restored in subsequent game sessions.

Key Features of SaveSystem

- Data Persistence: Enables game data to be stored between play sessions, ensuring continuity in the gaming experience.
- Flexibility: Supports saving various types of data, such as player health, currency, inventory items, and game states.
- Ease of Use: Offers a straightforward API for saving and loading data, abstracting complex file handling operations.

How the SaveSystem Works

Saving Data:
The script contains methods to save different types of data. For instance, SaveSystem.SaveCurrency(int currency) is used to save the player's currency. These methods typically serialize data into a format suitable for storage (like JSON) and write it to a file or PlayerPrefs.

Loading Data:
Corresponding methods are provided to load the saved data. For example, SaveSystem.LoadCurrency() will retrieve the saved currency value. The script reads the stored data, deserializes it, and restores it to the relevant game components.

Managing Save Slots:
The script can handle multiple save slots, allowing players to maintain multiple independent game progressions.

Integration with Game Components

- Game components that need to save or load data will interact with the SaveSystem script, typically in response to game events like level completion, player death, or manual save triggers.
- Event handlers or specific method calls can be set up within these components to trigger save or load operations.

Extending the SaveSystem

The SaveSystem can be extended to accommodate additional data types or to integrate more complex saving mechanisms, like cloud saving or database integration.

17.4 Save System Structure in the 2D Action Platformer Kit

In the 2D Action Platformer Kit, the save system is structured around a central concept where all game progress is temporarily stored in a default save slot (Slot 0) during gameplay. When a player decides to save their progress, the data from Slot 0 is then transferred to the chosen save slot. Similarly, when loading from a different save slot, the data is initially loaded into Slot 0 and then used during the gameplay. This approach standardizes the saving and loading process, making it more streamlined and consistent.

How It Works

Default Save Slot (Slot 0):
All gameplay progress is automatically saved to Slot 0 during a game session.
This includes player stats, inventory, level progress, and other relevant game data.

Saving Progress:
When a player chooses to save their game, the data from Slot 0 is copied to the selected save slot (e.g., Slot 1, Slot 2, etc.). This action effectively creates a permanent record of the game state up to that point in the chosen save slot.

Loading Progress:
When a game is loaded from any save slot other than Slot 0, the data is first loaded into Slot 0. The game then utilizes the data from Slot 0 for ongoing gameplay, ensuring a seamless transition.

Specifying a Save Slot:
Players have the option to select a specific save slot for storing their game progress.
Once a slot is selected, all subsequent saves will transfer data from Slot 0 to this chosen slot.

17.5 Technical Explanation of the Actor Script in the 2D Action Platformer Kit

The Actor script is a fundamental component in the 2D Action Platformer Kit, representing a character or entity within the game (such as the player character or an NPC). This script typically manages various aspects of the character's behavior, including movement, interaction with the game environment, and state transitions.

Key Responsibilities

Movement Control:
Handles the character's movements, such as walking, running, jumping, and crouching.
Interprets input from the player (or AI for NPCs) to move the character within the game world.

State Management:
Works in conjunction with a state machine or similar logic to manage different states of the actor (e.g., idle, moving, jumping).
Transitions between states based on gameplay scenarios, such as moving to a jump state when the jump input is received.

Interaction with Game Elements:
Manages interactions with other game elements, like collectibles, enemies, or obstacles.
Includes collision detection and response, which dictate how the actor reacts when coming into contact with different objects.

Implementation Details

Component-Based Structure:
The Actor script interacts with various Unity components such as Rigidbody2D for physics-based movement, Collider2D for collision detection, and Animator for managing animations. It also utilizes custom components for specific behaviors (e.g., DashModifier for dashing ability, WeaponManager for handling weapons).

Input Handling:
For player-controlled characters, the script interprets input from the player through keyboard, mouse, or gamepad.

For NPCs, the script uses AI logic to determine actions.

Modular Design:
The script is designed in a modular way to allow easy customization and extension. For example, adding a new behavior involves adding a new module or component rather than modifying the core Actor script.

17.6 ActorAnimations Script Documentation

The ActorAnimations script in the 2D Action Platformer Kit is a key component designed to manage and control the animations of the game's characters, commonly referred to as "actors." This script interfaces with Unity's animation system, enabling seamless transitions and interactions between different animation states based on the character's actions or environment.

Key Features

Animation State Management:
ActorAnimations script manages various animation states like walking, jumping, attacking, etc. It triggers these animations based on the character's current state or actions.

Seamless Transitions:
Ensures smooth transitions between different animations, enhancing the visual fluidity and responsiveness of character movements.

Event-Driven Animation Triggers:
Animations can be triggered by specific events, such as landing on the ground, picking up an item, or taking damage.

Implementation

Integration with Animator:
The script interacts with Unity's Animator component, which should be attached to the same Game Object. The Animator component controls the actual animation assets (like sprites or models) based on the parameters set by the ActorAnimations script.

Animation Parameters:

Uses parameters (like speed, isJumping, isAttacking) to trigger transitions in the Animator’s state machine. These parameters are set based on the character's current activity.

17.7 Player Input Script

The Player Input Script in the 2D Action Platformer Kit is a crucial component that manages the input from players and translates it into actions and movements of the player character. This script is essential for creating a responsive and interactive gameplay experience.

Technical Details of the Player Input Script

Input Detection:
The script continuously monitors player inputs, such as keyboard presses or gamepad inputs, to determine the player's intended actions.

Movement Data Integration:
The script retrieves movement data from the MovementData component associated with the actor. This data includes parameters like speed and direction that influence the character's movement.

Example: In the case of the FallState class, the script checks the actor's horizontal movement input to determine the direction the character should face during a fall.

Input to Action Mapping:
Based on the input detected, the script maps these inputs to specific actions like moving left or right, jumping, attacking, etc. The mapping is typically done using conditional checks that match certain inputs with corresponding actions.

18. License

All art assets provided within the kit, including sprites and music, are available for use in both commercial and non-commercial projects. This allows for a wide range of creative freedom and commercial opportunities for game developers who utilize these assets in their games. Whether you're working on a personal project or a commercial venture, these assets can be incorporated seamlessly into your game design.

Furthermore, games developed and created using the 2D Action Platformer Kit are fully cleared for commercial use. This means that you can monetize and distribute games that you develop using the kit without any restrictions on the revenue generated from these projects.

However, it is important to note that the kit itself, along with the scripts included within it, are not licensed for commercial distribution or resale. This means that while you can use the kit to create commercial games, the kit and its scripts cannot be sold, redistributed, or repackaged as part of another commercial product. This restriction is in place to protect the proprietary elements of the kit and to ensure fair use by all developers.