FARMING ISLAND – HYPER CASUAL FARMING SIMULATOR

Table of Contents

1. Introduction

Thank you for purchasing Farming Island! We appreciate your support and are excited to provide you with a complete hyper-casual farming simulator game template. This documentation will guide you through the setup, usage, and customization of the template to help you get the most out of this package.

1.1 Template Overview

"Farming Island - Hyper Casual Farming Simulator" is a simple yet engaging farming simulation template designed for mobile platforms and low-end devices. This template allows developers to quickly create a farming game where players can plant crops, harvest resources from animals, collect fruits from trees, fish and purchase new island.

The game features a variety of environments, including a lush grassland island, a harsh desert island, and a snowy island, each with its own unique attributes and challenges. It features easy-to-understand mechanics, customizable assets, and optimized performance for hyper-casual gameplay.

1.2 Key Features

This template offers a range of engaging and versatile features designed to provide a seamless farming simulation experience:

- **Hyper-Casual Farming Mechanics:** Enjoy intuitive and easy-to-learn farming gameplay that focuses on casual, enjoyable mechanics. Perfect for quick play sessions and accessible to players of all skill levels.
- **Customizable Farming Environment:** Tailor your farming experience with a variety of environments, including lush grasslands, arid deserts, and snowy landscapes. Each environment offers unique challenges and opportunities, allowing for diverse gameplay.
- **Simple Controls and Easy-to-Understand User Interface:** Navigate the game effortlessly with straightforward controls and a user-friendly interface. Designed to be intuitive for both new and experienced players, ensuring a smooth and enjoyable experience.
- **Optimized for Mobile and Low-End Devices:** Experience high performance on mobile platforms and low-end hardware. The template is optimized to ensure smooth gameplay with minimal load times and efficient resource management.

These features combine to create a flexible and engaging farming simulation experience, ideal for developing hyper-casual games with a variety of customizable elements.

2. Getting Started

2.1 System Requirements

To ensure optimal performance and compatibility with "Farming Island - Hyper Casual Farming Simulator," please use Unity version **2021.3** or higher.

2.2 Quick Setup

The package contains three distinct levels in the “Scenes” folder, each offering a unique farming environment and gameplay experience based on the environment type and available resources:

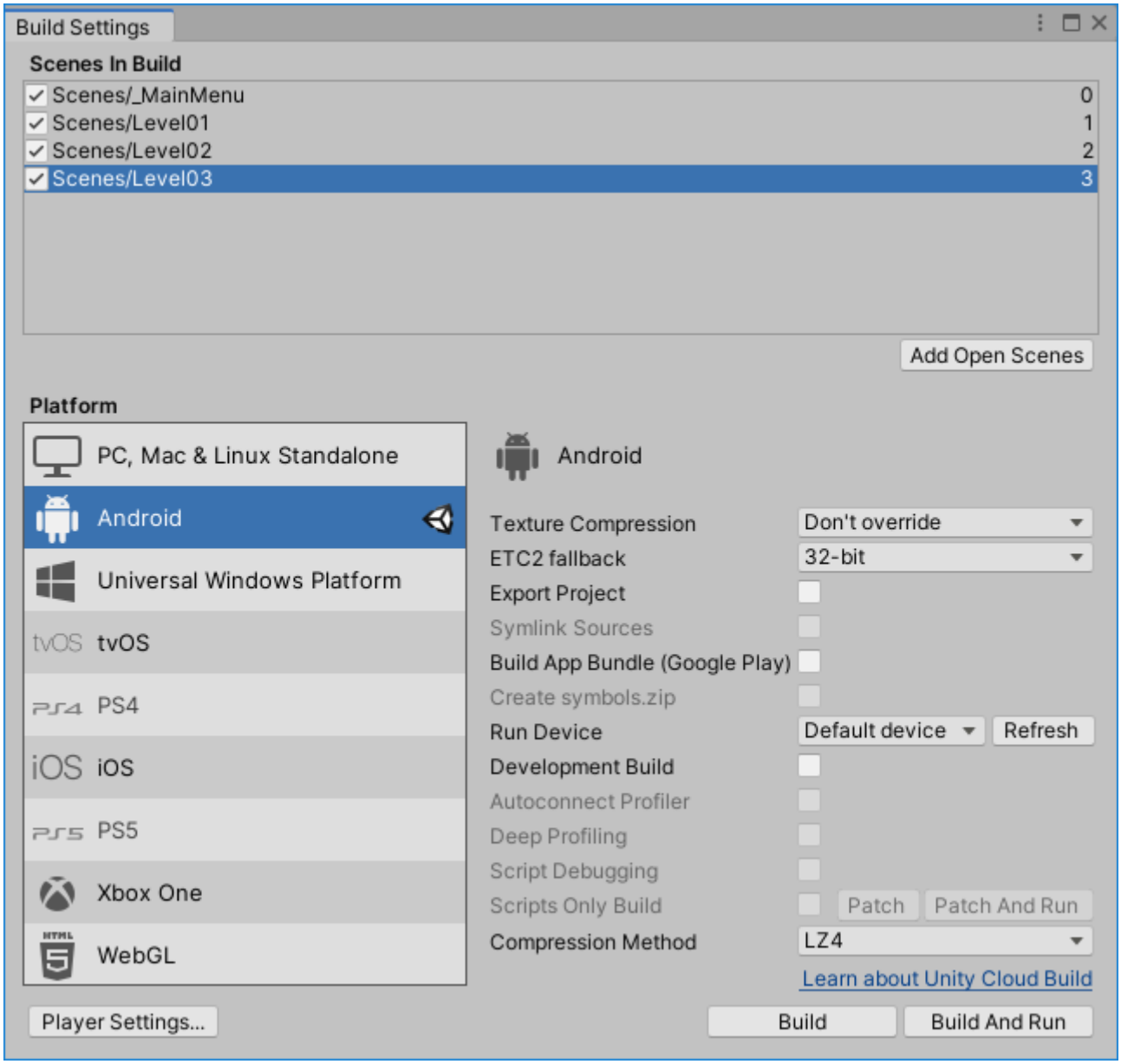
- **Level01:** Grassland Island
- **Level02:** Desert Island
- **Level03:** Snowy Island

You can open these scenes in Unity to explore how the game template functions. Inspect the layouts, game objects, and scripts to understand the structure and gameplay mechanics. Each level provides a distinct experience and can serve as a foundation for creating your own customized environments.

2.3 Creating New Level

2.3.1 Initial Setup

Locate and duplicate the **"_LevelTemplate"** scene in the Scenes folder. This template includes all the core components needed to create a new level from scratch. While not mandatory, I suggest rename the duplicated scene using a numbering system for easy organization (e.g., Level01, Level02, Level03, etc.). Don't forget to add the new level to the Build Settings, and make sure the **"_MainMenu"** is always set to index 0.



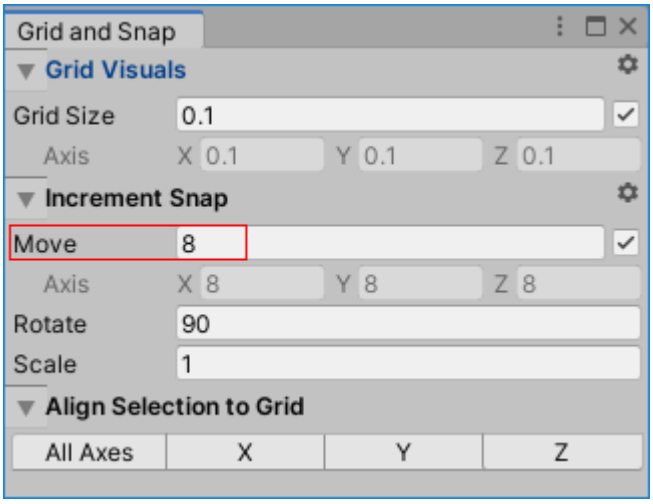
Lastly, open the new scene and let's briefly review each of the core components one by one:

- **Main Camera:** This is the primary camera used to render the entire game. It features a "Cinemachine Brain" component, leveraging Unity's Cinemachine for smooth camera control and transitions.
- **Directional Light:** The main and sole light source used in the game.

- **Player:** The character you control throughout the game.
- **Player Virtual Camera:** A Cinemachine Virtual Camera that consistently follows the Player's movements.
- **Island Manager:** This component manages the logic for the island elements in the scene, including buying new islands and configuring them accordingly.
- **Pool Manager:** Utilizes a pooling system to minimize garbage collection in C#. It handles the instantiation of frequently used items in large quantities at the start of the game, avoiding the performance hit of frequent instantiation and destruction.
- **Audio Manager:** Manages the playback of sound effects and background music in the game.
- **UI Manager:** Manages and displays all user interface elements within the game.

2.3.2 Laying Out Islands

Before placing islands in the scene, it's important to adjust the grid snapping to match the island dimensions. Go to **Edit > Grid and Snap Settings > Increment Snap**, and set the "Move" value to 8 (the width and length of the islands in this template). This will make arranging the islands in the scene much easier.



2.3.2.a Island Component

The **Island Manager** already has one island placed in the scene as its child. You can replace this island to a different theme by selecting one from the **Prefabs > Islands** folder. You'll find several island prefabs there (Island01, Island02, Island03), each differing only by their material. Island01 uses a material that matches the Level 1 theme, which is grassland; Island02 is for the desert theme, and Island03 is for snowy environments.

Once you've chosen the island that fits your current level's theme, simply duplicate it and arrange them as needed, holding **Control** to snap them to the 8x8 grid.

At least one island ***must*** have the "**Is Unlocked**" flag set to **true**, preferably the island directly below the player's starting position. This ensures the island is unlocked at the start of the level without requiring the player to purchase it.

IMPORTANT:

- *Make sure the islands remain children of the Island Manager*
- *The sibling order of the islands determines their unlock price*

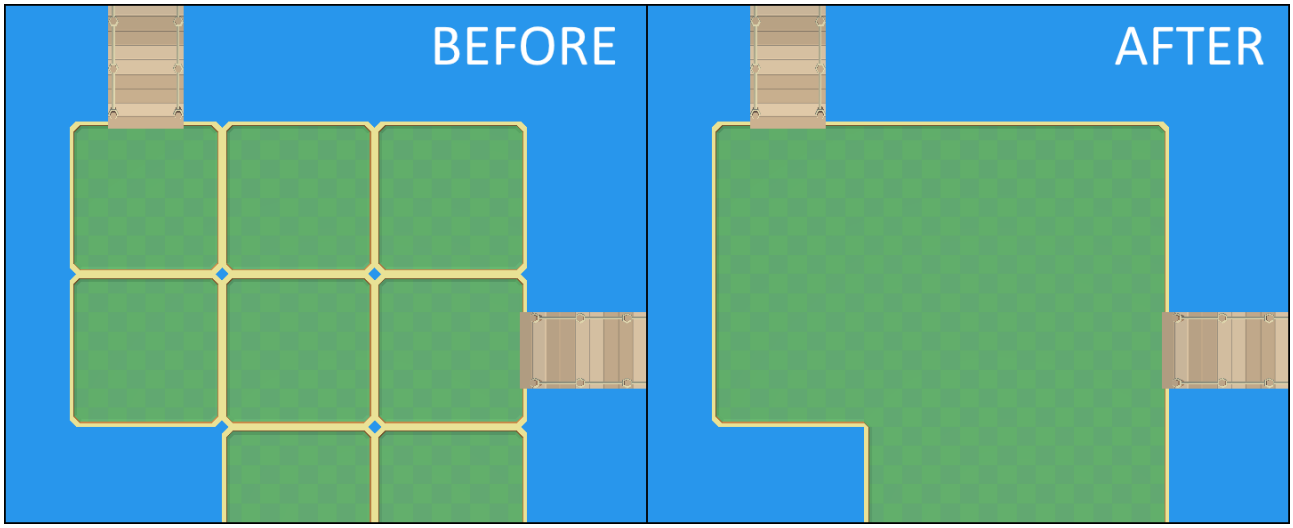
2.3.2.b Island Variant

There is a variant of island called **bridge**, which naturally have a bridge mesh and the "**Is Bridge**" flag set to **true**. Bridges also have a **Fishing Spot** as a child object. While island prefabs differ in materials based on the level theme, bridge prefabs feature different types of fish that can be caught at their Fishing Spots.

Bridges may be rotated 90 degrees on the Y-axis, depending on the positions of the islands they connect.

2.3.2.c Island Tiles Bitmasking

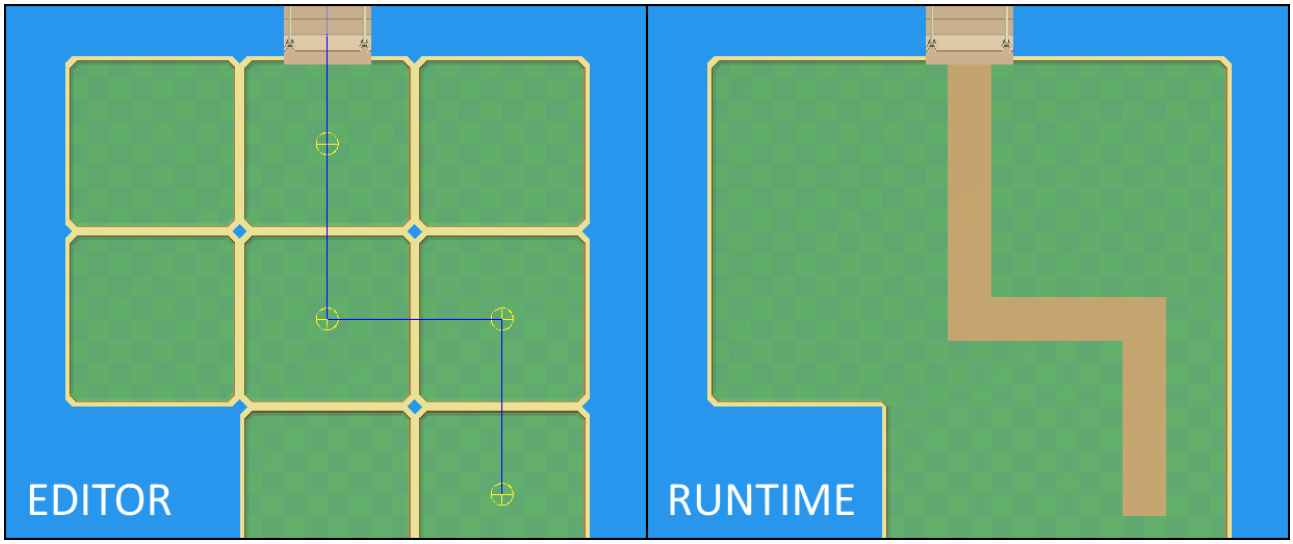
You may notice that the islands placed in the scene appear disconnected, resembling individual cube-shaped islands arranged side by side. The **Island Manager** takes care of this at runtime by calculating neighboring islands and applying the correct meshes, making them look seamlessly connected as one continuous landmass. You can even test the level now to see this in action.



This technique, called **Tile Bitmasking**, is what enables the islands to blend together smoothly. For a detailed explanation of how it works, feel free to check out this article.

2.3.2.d Island Settings

There are a few important things to keep in mind when placing islands in the scene. First, the "**Has Road**" flag determines whether a road is drawn on the island. The road network is visualized with blue lines in the editor, and at runtime, a mesh is generated to represent the road visually. This uses a technique similar to the island's tile bitmasking, but simpler.



Second, there's a reference field called "**Constraint**", where you can assign another island in the scene. If an island is assigned to this field, the current island cannot be purchased until the island in the constraint field has been bought.

2.3.3 Island Props

All props, such as farms, trees, animals, buildings, and decorations, implement the **IProp** interface. This interface ensures consistent behavior and interaction across all prop types.

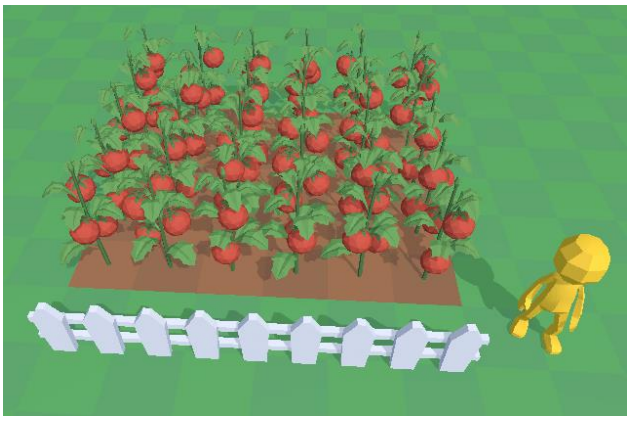
Each prop includes a public method called **Animate**, which is used to animate the prop when the island is unlocked. Additionally, every prop has an Enum called **Location**, which defines the type of location the prop represents (if applicable). This location data is later used by the **Minimap** to display the corresponding icon on the map.

IMPORTANT: *Every prop must be a child of an island for proper functionality.*

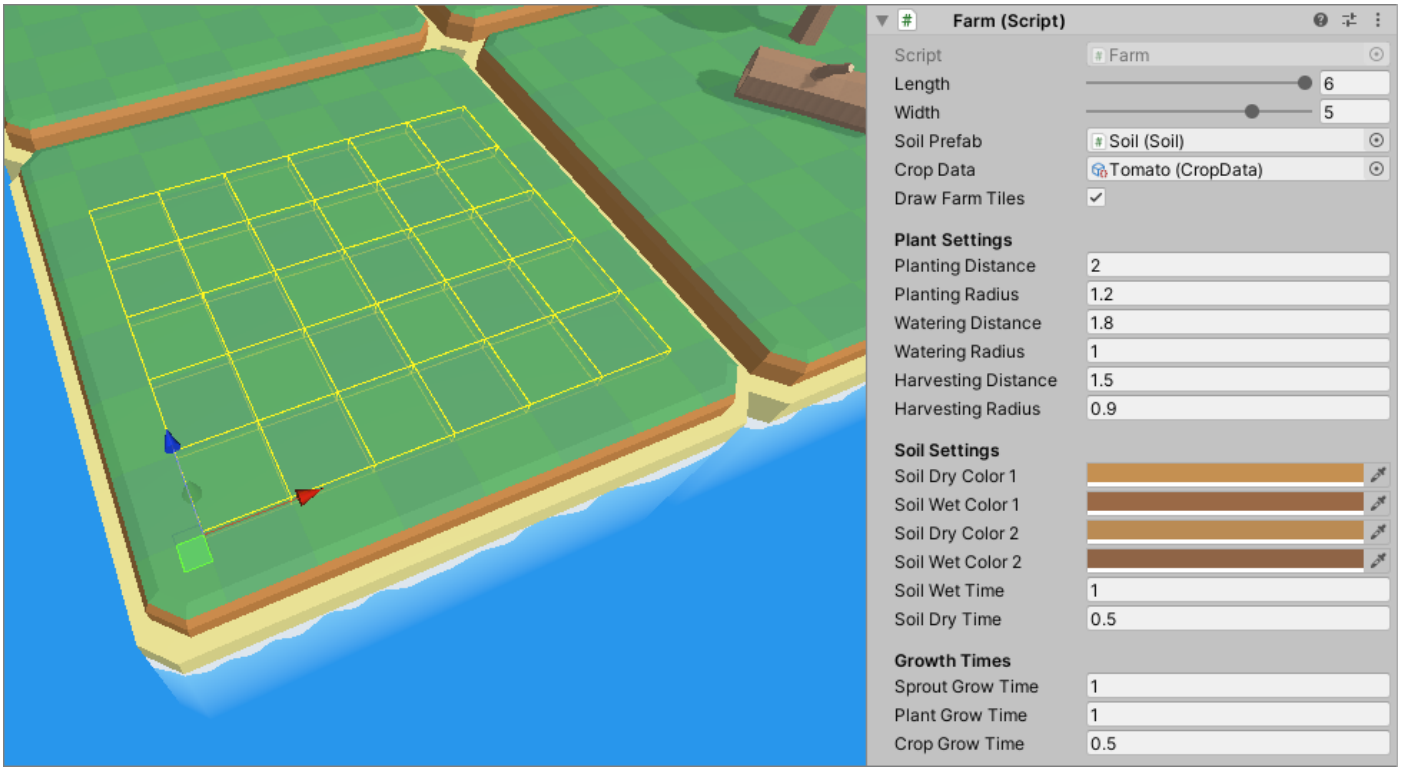
Let's take a closer look at each type of prop and how they function within the game!

2.3.3.a Farm

The **Farm** class in this game template manages farming functionality by setting up a grid of soil tiles where players can plant, water, and harvest crops.



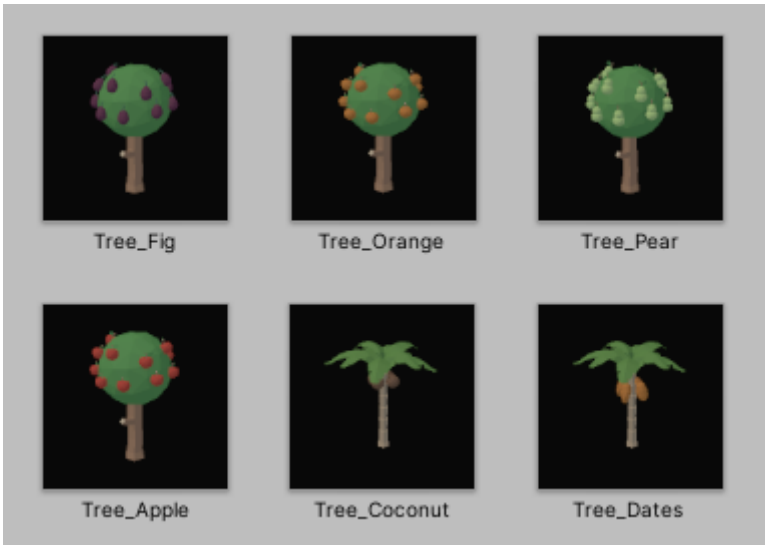
You can find the farm prefab in the **Prefabs > Farming** folder. Simply drag it into the scene and set it as a child of an island.



The **Farm** class has a key field called **Crop Data**, which you can assign as needed depending on the crops you want for the farm. Other settings are already configured just fine, but feel free to tweak them if necessary. You can hover over each field in the Inspector, as they all come with helpful tooltips to guide you.

2.3.3.b Fruit Tree

The **FruitTree** class represents a tree that produces fruit in the game. Players can interact with the tree by shaking it to drop fruits, which they can then collect. The tree's interaction involves a shaking mechanic, where players must reach a certain shake intensity (by shaking the virtual joystick) to drop the fruits. Once the fruits are dropped, they can regrow after a set time.



For now, select one of the available tree prefabs from the **Prefabs > FruitTrees** folder, drag and drop it into the scene, and again, make it as a child of an island. Detailed instructions for customizing the tree with different fruits will be provided later.

2.3.3.c Stall

The **Stall** is a prop where players can sell items from their inventory. Each stall has a list of accepted items that can be sold, and the player earns coins based on the value of the items. When the player approaches the stall, items are sold automatically until the inventory is emptied, the maximum selling time is reached, or the player leaves.

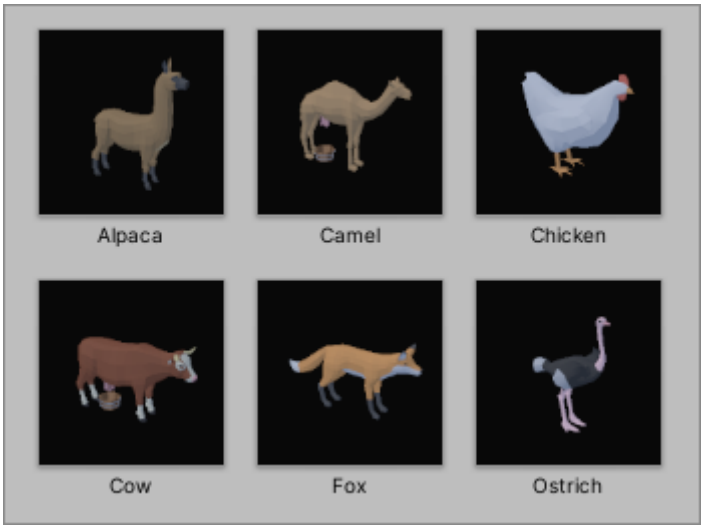
As usual, drag and drop a stall prefab from the **Prefabs > Building** folder and set it as a child of an island. To customize the appearance, simply replace the mesh in the **Mesh Filter** component of the prefab. Several stall models (meshes) are available in the **Models > Buildings** folder.

2.3.3.d Animal Controller

The **Animal Controller** is a class that manages the behavior of animals in the game. Although it might seem unusual, it also implements the **IProp** interface, allowing it to function like other props within the game.

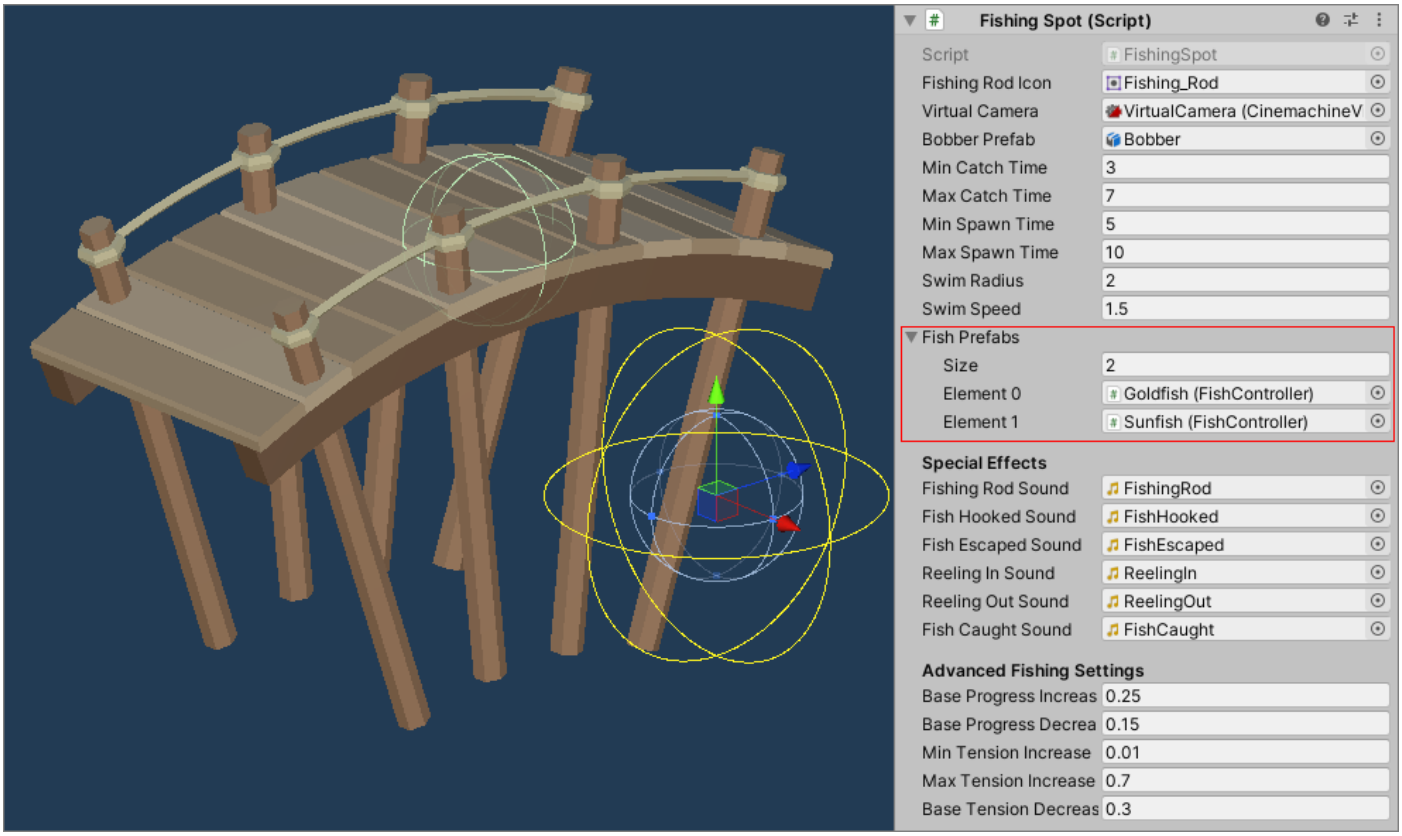
It works alongside the **Animal** class, which represents individual animals responsible for producing products such as eggs, milk, or wool. In addition to controlling the animal's wandering behavior, the Animal class handles product generation, player interaction, and product collection. Each animal has a specified production time based on its type (e.g., Poultry, Dairy, Fiber) and drops items for the player to collect when ready.

There are several animal prefabs available in the **Prefabs > Animals** folder, allowing you to choose based on the theme of the current level. Simply pick the ones you like, drag and drop them into the scene on the island where you want the animal to appear.

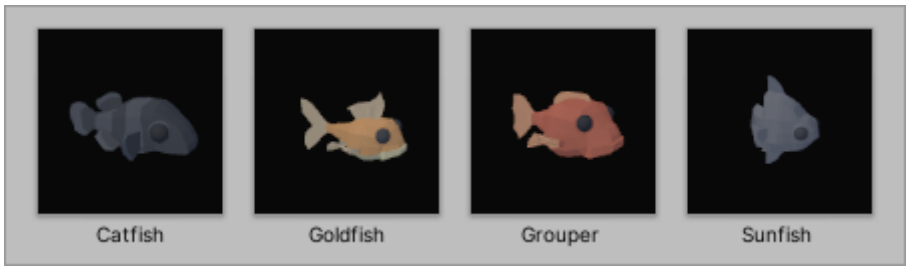


2.3.3.e Fishing Spot

The **Fishing Spot** is the only prop you don't need to manually drag and drop when creating a level, as it's already a child of the **Bridge** prefab.



However, you may want to adjust which fish can be caught there. Simply find the fish prefabs in the **Prefabs > Fish** folder, choose the ones you like, and assign them to the **Fish Prefabs** field.

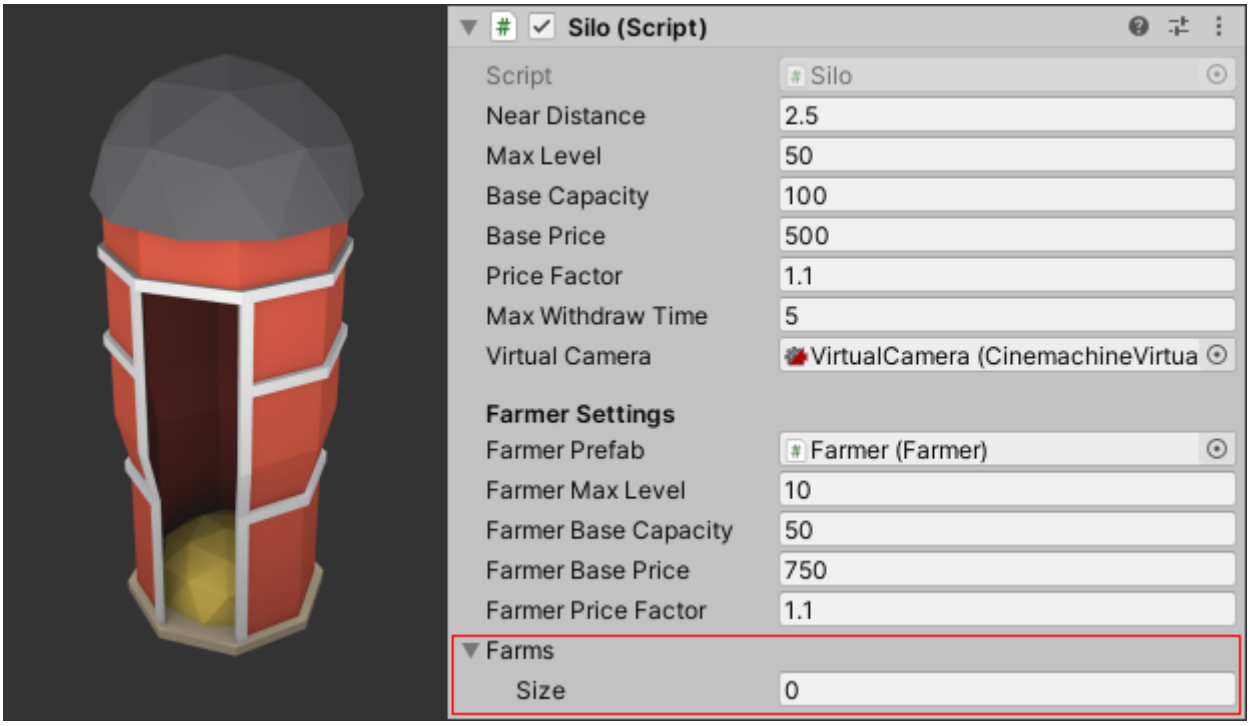


The default settings for the fishing spot are already well-suited for general gameplay. However, you are free to tweak the settings as needed to fit your specific requirements. Each field comes with helpful tooltips that provide guidance on what the values mean, allowing you to adjust them with confidence.

2.3.3.f Silo

The **Silo** serves as a key structure for hiring **Farmers** and storing the crops they harvest. You'll need to assign **farms** in the scene to the silo, enabling farmers to work with those specific farms. These farmers will help you with planting, watering, harvesting, and storing crops. Keep in mind that the silo has a maximum capacity, so it's essential to collect crops regularly; if it's full, farmers will stop their work.

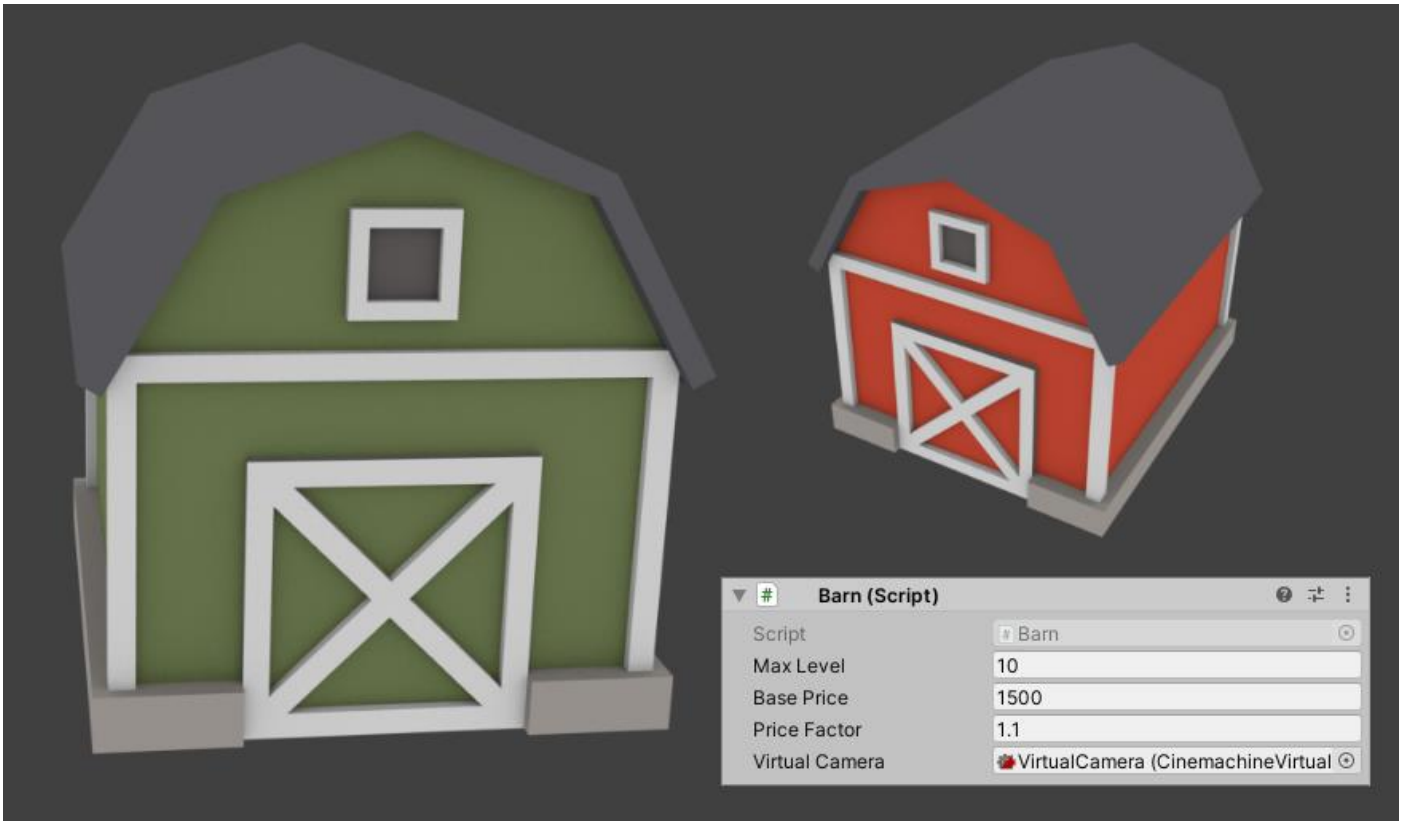
IMPORTANT: *Be mindful of the silo's placement on the island, as it needs a clear, unobstructed path to the farms. Farmers rely on pathfinding to access the farms, so placing the silo in an optimal location is crucial for maintaining efficient crop storage and workflow.*



Additionally, you can enhance the silo to boost its storage capacity. Upgrading farmers will also allow them to carry more crops back to the silo each time, increasing overall efficiency.

2.3.3.g Barn

The **Barn** is where animals can be upgraded in the game. There are three available upgrade categories: Poultry, Dairy, and Fiber, which correspond to animals that produce eggs, milk, and wool. These upgrades reduce the time it takes for animals to produce goods after each collection.



2.3.3.h Decoration

The **Decoration** prop is purely cosmetic, serving no functional purpose other than to enhance the visual appeal of the game world (except for **Helipad**). You can place various decorations around the islands to personalize and beautify the environment, making each island feel unique.

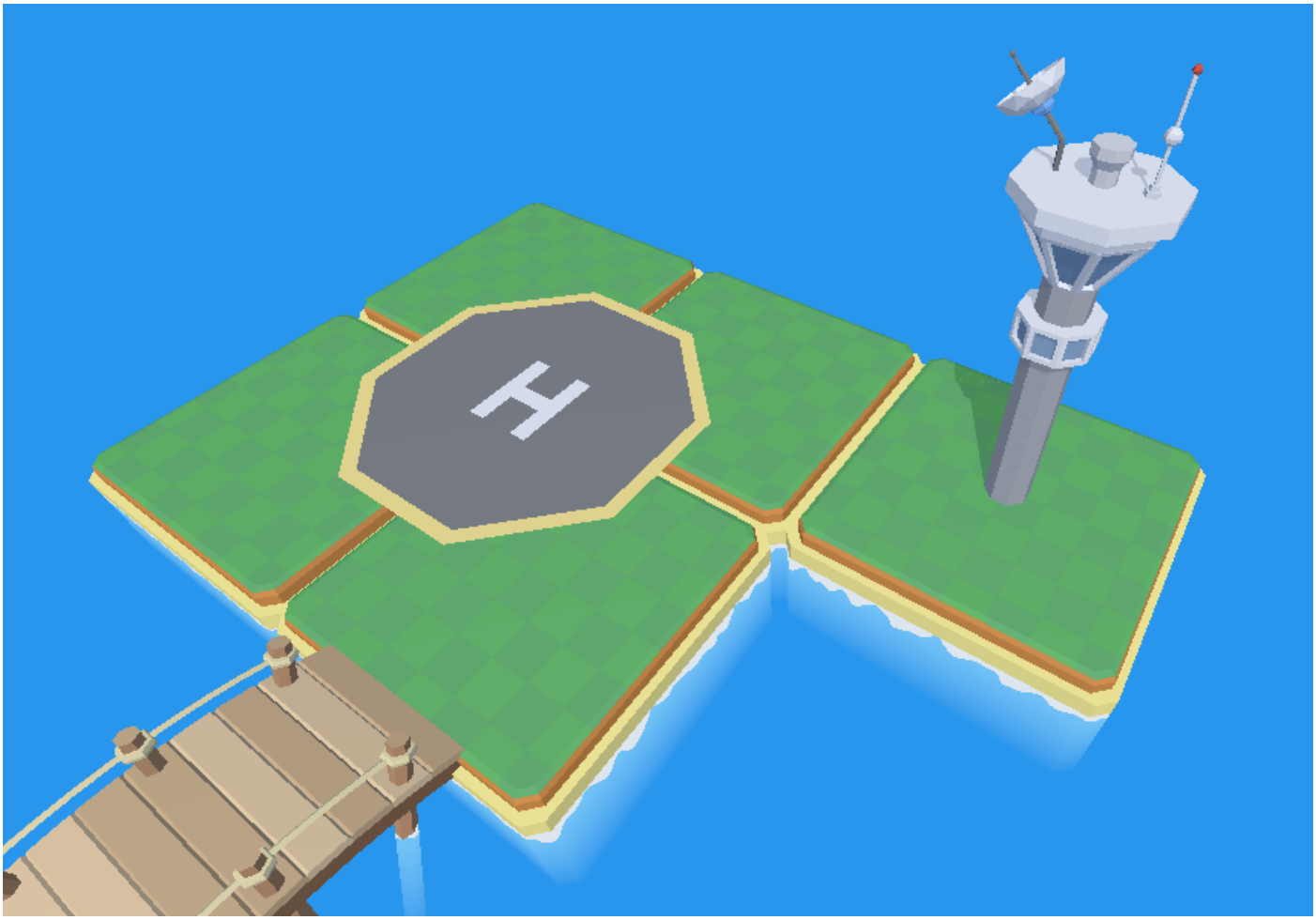
2.3.3.i Dock

The **Dock** is a key prop that allows player to travel between islands across the archipelago. By interacting with the dock, player can move from one dock to another. This makes long-distance travel faster and more convenient, eliminating the need for walking.

2.3.3.j Air Traffic Control Tower (ATC Tower)

The last prop is **ATC Tower**, it is a crucial prop that should only be placed on the final island to unlock the next level. When the player interacts with the ATC Tower, it triggers a sequence involving a helicopter that transports the player to the next scene / level. This prop marks the transition from one level to the next, serving as the final objective player need to complete before advancing in the game.

IMPORTANT: *Please ensure that you only place the ATC Tower if there is a next level available beyond the current level. The ATC Tower is designed to facilitate movement to the next scene. If the current level is the last level in the game, do not place the ATC Tower, as it will not serve a functional purpose in this context.*



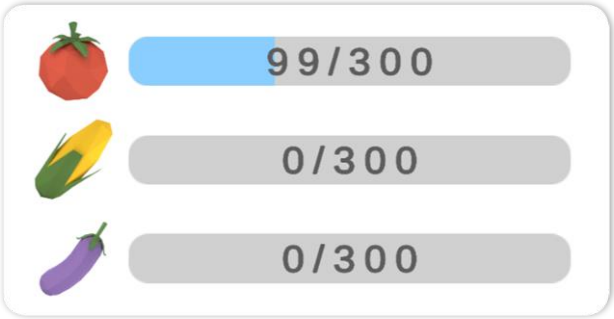
In addition to the ATC Tower, the final island should also have a **Helipad** (a **Decoration**) that needs to be assigned to the tower. The placement of the helipad doesn't need to be exactly on top of the final island; you can prepare a wide area on other islands for placement. This helipad serves as the landing and takeoff point for the helicopter.

3. User Interface (UI)

All the UI elements for the gameplay are contained in a prefab called **UI Manager**, which can be found in the **Prefabs > UI** folder. This manager includes an **Event System** for handling user input and interactions, as well as a **Canvas** to display the UI elements. Let's take a closer look at them one by one.

3.1 Silo Info

The first UI class in the list is **Silo Info**, which is responsible for displaying the details of crops stored in the **silo**. It dynamically creates a UI element for each crop and updates their status in real-time. The class uses an offset to position the UI above the silo by utilizing the camera's screen-space conversion, it ensures the UI stays properly aligned in the world.



3.2 Joystick (Virtual)

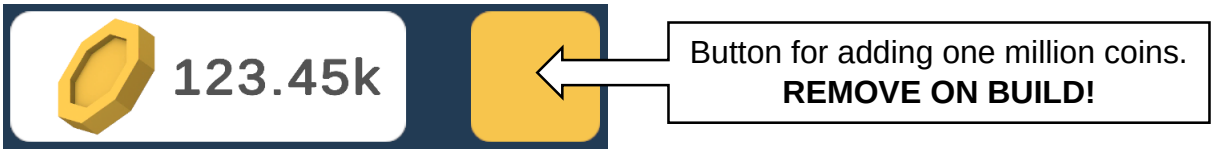
Next is the **Joystick** UI, implemented using [Yasirkula's SimpleInput]. It simplifies touch input handling, making it easy to integrate responsive and intuitive controls for player movement, especially on mobile devices. SimpleInput's setup is quick and user-friendly, allowing for smooth navigation with minimal configuration.

3.3 FPS Counter

The next UI element is the **FPS Counter**, primarily used for debugging purposes. It tracks the game's frame rate, helping you monitor performance during development. While useful for optimizing gameplay, it can be safely removed from the final build to avoid cluttering the player's interface.

3.4 Coin Display & Debug Coin Button

The next UI component is the **Coin Display**, which shows the player's current coin amount. It uses **TextMeshProUGUI** to provide a clear and formatted representation of the coins. The display updates automatically whenever the coin count changes, thanks to its subscription to the coin change event in the **IslandManager**. This ensures players always have an accurate view of their resources during gameplay.

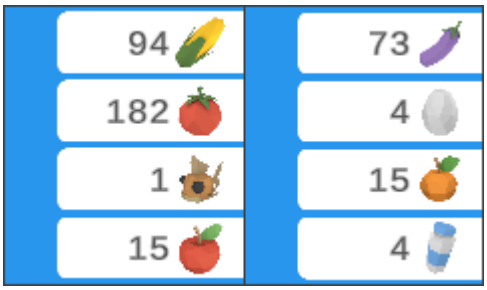


3.5 Minimap Button

Next is the **Minimap Button**, which simply opens the **Minimap** screen that will be covered later.

3.6 Item Info Container

Next is the **Item Info Container**, a plain **Rect Transform** with a **Vertical Layout Group** and a **Content Size Fitter**. It holds **Item Info** elements, which display the items and quantities the player possesses in their inventory.



The Item Info elements are dynamically created, updated, or removed based on the state of the player's inventory. As items are added or removed, the container adjusts in real-time to reflect the current inventory.

3.7 Farming Button

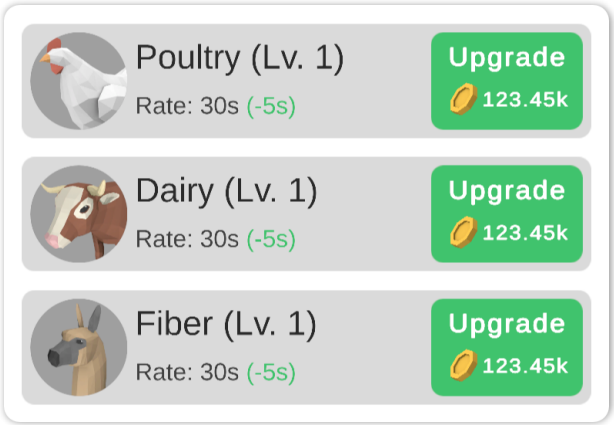
The **Farming Button** allows the player to perform farming tasks like planting, watering, and harvesting crops. When pressed near a farm, it triggers the appropriate action, such as planting seeds, watering growing plants, or harvesting fully grown crops. The icon and action adjust automatically based on the current farm's state.

3.8 Activity Button

The **Activity Button** handles non-farming tasks such as shaking trees to collect fruits, gathering products from animals, and fishing at designated fishing spots. Similar to the Farm Button, the icon and action will automatically adjust based on the resources the player is about to collect.

3.9 Animal Upgrade

The **Animal Upgrade** UI element allows players to upgrade the production efficiency of their animals when they are near a **barn**. It displays the current level, production rate, and upgrade cost for each animal type. Players can click the upgrade buttons if they have enough coins to enhance an animal's productivity, reducing its production time. The UI dynamically updates based on the animal levels and prices, and once an animal reaches its maximum level, the upgrade option is disabled.

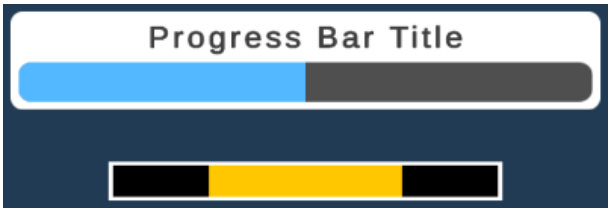


3.10 Silo Upgrade

The **Silo Upgrade** UI allows players to enhance their silo's storage capacity when they are near a silo. It shows the current silo level, its storage capacity, and the cost for the next upgrade. Players can upgrade the silo if they have enough coins, and the UI automatically updates to reflect the new capacity after each upgrade. In addition, players can upgrade the associated **farmers**, improving their efficiency. The UI dynamically adjusts to display upgrade options for all assigned farmers and their current statuses.

3.11 Progress Bar

The **Progress Bar** displays the completion status of various minigames, such as shaking trees, collecting animal products, and fishing. For the fishing minigame, there is also a smaller bar called **Line Tension** displayed below the progress bar, which indicates the tension of the fishing line. If the tension bar fills up completely, the line will break, adding a layer of challenge to the game.



3.12 Minimap

The **Minimap** screen shows a simplified view of the player's surroundings, displaying island tiles, props, and important locations like docks or **purchaser** spots. The player's icon is shown and updated in real time as they move around the island. The minimap also highlights specific locations using different colors and icons based on the type of prop or location. Players can also use the minimap to travel between docks if travel is enabled (which occurs when the minimap opens automatically as they approach a dock). The map adjusts dynamically based on the player's progress and unlocked areas.

3.13 Screen Fader

The last UI element is **Screen Fader**, it is a simple black image that covers the entire screen, used to smoothly transition between scenes or gameplay events. It fades in and out to provide a seamless and visually appealing transition effect for the player.

4. Customization & Extension

This section will explore how users can modify and enhance gameplay elements to suit their preferences, ensuring a unique and engaging farming adventure.

4.1 Island Customizations

You can customize islands in two ways. The easiest method is to duplicate an existing island prefab and update its materials and textures. Alternatively, you can replace the entire island model with one of your own design. Let's explore both approaches in detail.

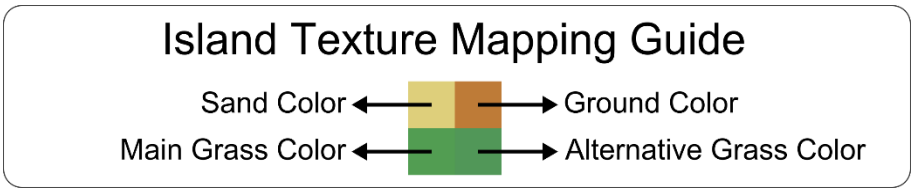
4.1.1 Duplicating and Modifying Existing Islands

To quickly customize an island, follow these steps:

1) **Duplicate the Island Prefab**
   In the **Prefabs > Islands** folder, locate one of the existing island prefabs. Right-click on the prefab and select Duplicate. This will create a copy of the island that you can modify without altering the original.

2) **Duplicate the Island Material**
   Navigate to the **Materials > Islands** folder. Find the material used by the original island, right-click on it, and select Duplicate. This copy will serve as the base for your custom texture.

3) **Apply Custom Texture**
   Replace the texture in the duplicated material with your own custom texture. You can do this by selecting the material and modifying the texture map in the Inspector window.



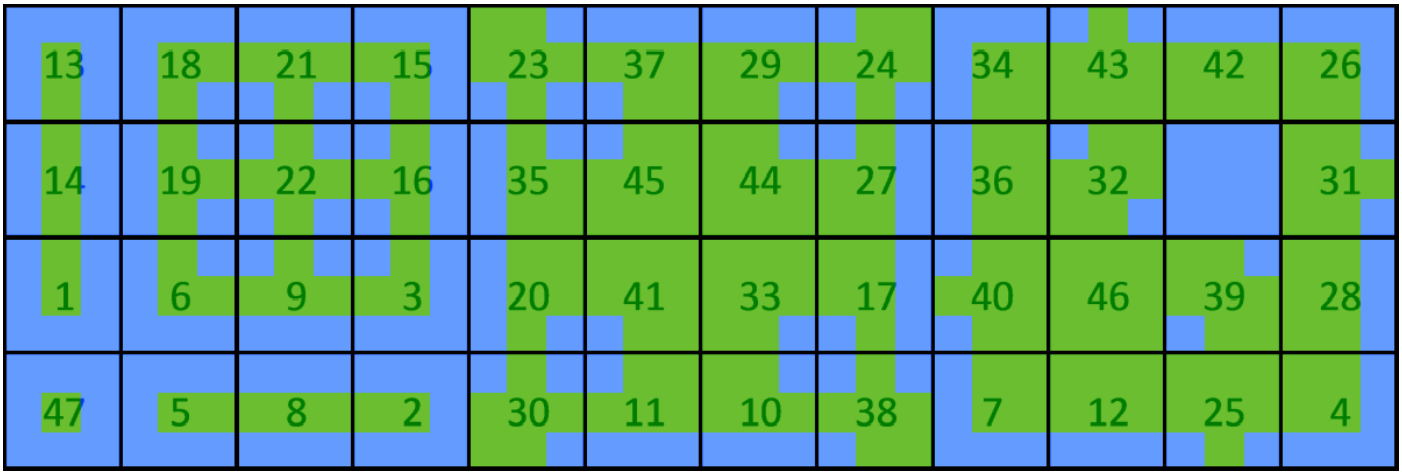
After following these steps, your new island will have a unique appearance, based on the material and texture you've applied, while preserving the original prefab as a reference.

4.1.2 Making Island with Custom Models

Creating an island from scratch using custom models is a more complex process. This template employs a technique called **Tilemap Bitmasking**, which requires a total of **47 unique island meshes / tiles**. This approach is similar to the concept of auto-tiles in 2D games, where the game dynamically determines the appropriate mesh to use based on the neighboring tiles.

To get a better understanding of how things are implemented, I recommend inspecting the **Island.fbx** (located in **Models > Islands** folder) using your preferred 3D modeling software. Take a closer look at the setup and structure of the island meshes. This will help you grasp how the game dynamically assembles island terrains based on neighboring tiles. It's a great way to familiarize yourself with the intricacies of island customization.

Here is chart of island tiles with all of the possible border configurations. The numbers on each tile represent the bitmasking value:



As I mentioned earlier, here is a more detailed explanation of this topic for those seeking an in-depth understanding.

4.2 Adding New Crop

Crops are a fundamental resource that players can plant, grow, and harvest. Each crop is represented by an instance of the **CropData** class, which inherits from the **ItemData** class (Scriptable Object). This structure allows for the organization of crop-related information, including:

1) **Item Identification:** Each crop has a unique **itemId**, which helps in identifying and referencing the crop within the game's ecosystem.

2) **Pricing:** Crops have a **price**, indicating their value when sold, providing an economic aspect to farming.

3) **Visual Representation:** Each crop has an **icon** that represents it visually in the game UI, making it easier for players to recognize different types of crops.

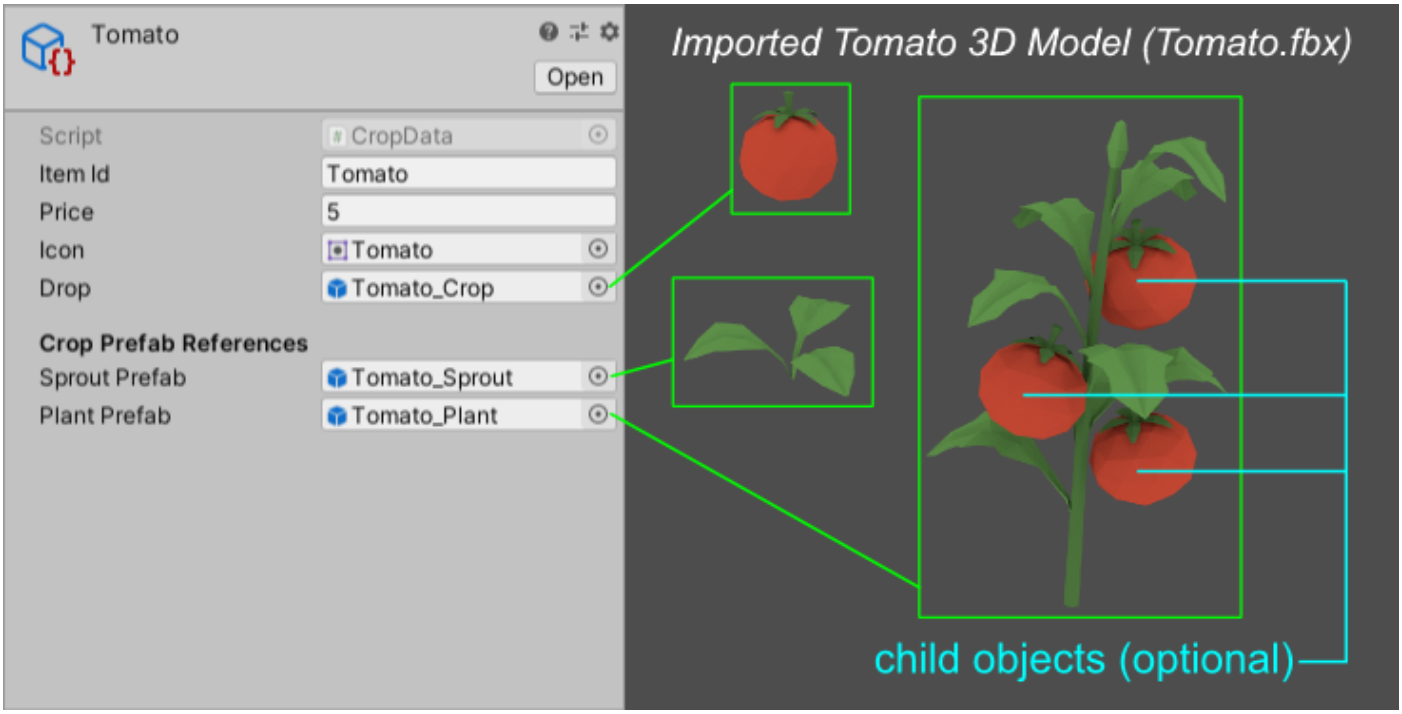
4) **Drop Prefab:** When harvested, crops can spawn a drop, which is a **GameObject** representing the harvested item.

5) **Growth Stages:** The CropData class also includes references to two prefabs: **sproutPrefab** and **plantPrefab**. The sproutPrefab represents the initial growth stage, while the plantPrefab signifies the fully grown crop. This allows players to see the progression of their crops from planting to harvesting.

To create a new crop (CropData asset), locate **Resources > Items** folder in project window. Right-click there and select **Create > Crop Data** from the context menu. Configure the new crop according to your preferences.

IMPORTANT: *It's crucial to place all your ItemData assets (including CropData) inside the* ***Resources > Items*** *folder because the* ***ItemDatabase*** *class utilizes Unity's* ***Resources.LoadAll*** *method to dynamically load all item data at runtime. This method only works for assets located within the Resources directory.*

To provide a clear example of how to configure a crop, let's take a look at the **Tomato** crop setup. The following images illustrate the various settings and parameters used in the CropData asset for tomatoes. By examining these references, you can gain a better understanding of how to create and customize your own crops effectively.



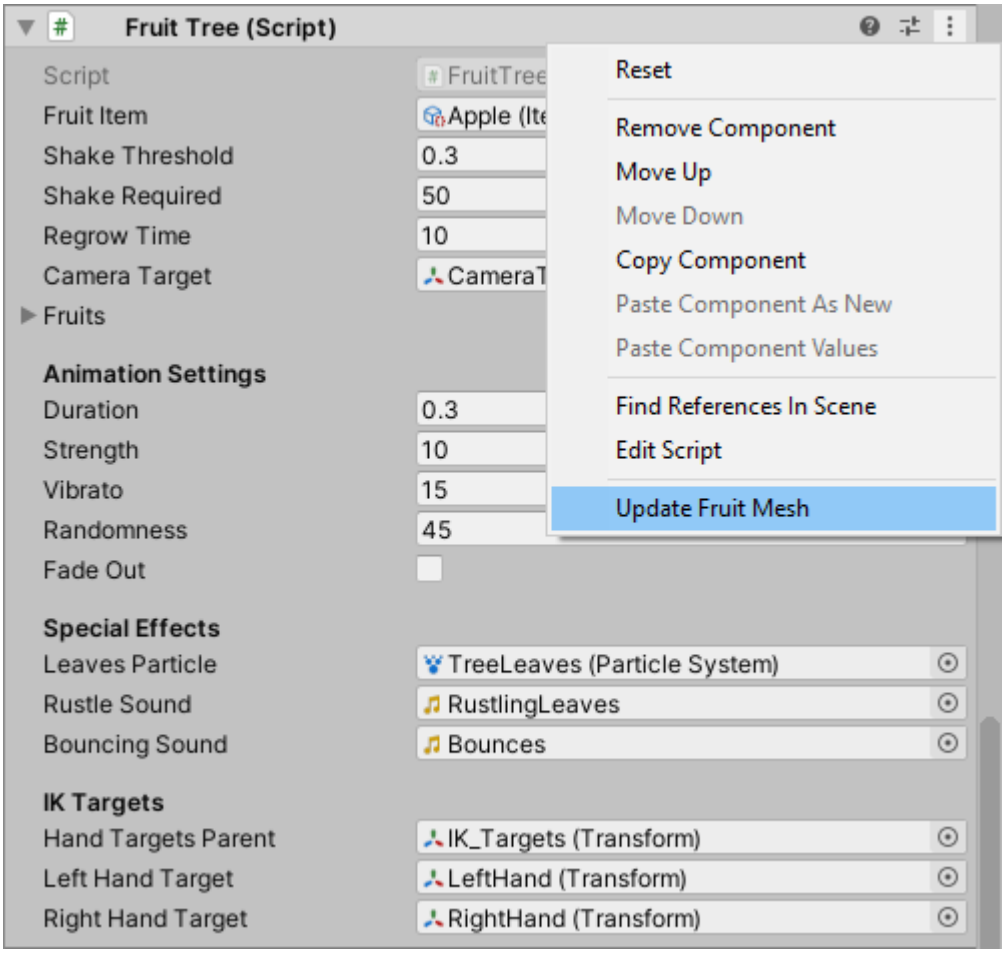
4.3 Customizing Fruit Tree

In this section, we'll go over two ways to customize fruit trees in the game. You can quickly modify existing fruit trees by duplicating and assigning different fruit item data, or take a more advanced approach by using your own custom tree models.

4.3.1 Quick Fruit Tree Customization

For basic customization, follow these steps to modify a fruit tree quickly:

1) **Duplicate a Fruit Tree Prefab**:
    - Go to the **Prefabs > FruitTrees** folder in your project.
    - Right-click on one of the existing fruit tree prefabs and select **Duplicate**.
    - Rename the tree to reflect the type of fruits it produces.
2) **Assign Different Fruit Item Data**:
    - With the new tree prefab selected, find the **Fruit Item** field in the inspector.
    - Assign the desired fruit item by dragging in a different **ItemData** asset from the **Resources > Items** folder.
3) **Update the Fruit Assignment**:
    - In the **FruitTree** script (attached to the tree prefab), use the context menu by right-clicking on the script's header in the inspector.
    - Select **Update Fruit Mesh** from the menu to apply the new fruit item.



This method allows you to easily modify the appearance and behavior of the fruit trees with minimal effort.

4.3.2 Advanced Fruit Tree Customization with Custom Models

Customizing fruit trees with your own models offers a more complex but rewarding way to add unique content to your game. Follow these steps to implement custom fruit tree models:

1) **Prepare Your Custom Tree Model**:
    - Create your custom tree model in your preferred 3D modeling software.
    - **Important**: Ensure the fruit models are parented to the tree model before exporting it to Unity. This will simplify the fruit positioning process.
2) **Make the Custom Tree into a Prefab**:
    - After importing the tree model, make it into a prefab in **Prefabs > FruitTrees**.
    - Add the **FruitTree** Component to this prefab. Upon doing so:
        - A **Sphere Collider** (trigger) will be added automatically to detect nearby players for the tree-shaking minigame.
        - A **Capsule Collider** will also be added to handle the tree's physical collision.
        - Adjust the size of these colliders to fit your custom tree model.
3) **Assign the Fruit Item Data**:
    - Create appropriate Fruit **ItemData** inside the **Resources > Items** folder that matches the fruits for your custom tree.
    - Assign this new ItemData to the **Fruit Item** field in the FruitTree component.

4) **Set the Camera Target**:
    - Create an empty child object under the tree prefab.
    - The game will generate a **Cinemachine Virtual Camera** at runtime, and this empty object will serve as the **camera target** during the minigame. Position this object slightly **lower, zoomed in, and tilted downward** to provide an ideal player view during tree-shaking minigame.
    - Assign this empty object to the **CameraTarget** field in the FruitTree component.
5) **Assign the Fruits**:
    - Drag and assign all the fruit child objects (from your tree model) into the **Fruits** field in the FruitTree script. These objects will represent the fruits that the player interacts with.
6) **Configure Tree Animations**:
    - In the **Animations** section of the FruitTree script, you'll find tweening settings for when the tree is shaken.
    - You can leave the default settings or adjust them as needed to match the style of your custom tree.
7) **Add Special Effects**:
    - Under the **Special Effects** section of the FruitTree component, assign:
        - A **leaves particle effect** (located in **Prefabs > Effects**) to simulate rustling leaves.
        - The **rustling leaves sound** for when the tree is shaken.
        - A **bouncing sound effect** for when the fruits drop.
8) **Set Up IK Targets for Player Interaction**:
    - In the **IK Targets** section, create an empty game object as a child of the tree and assign it to the **Hand Targets Parent** field.
    - Inside this parent object, create two more empty objects positioned where the player's left and right hands should interact with the tree.
    - Assign these new objects to the **Left Hand Target** and **Right Hand Target** fields in the FruitTree script to create a convincing hand-grabbing effect when the player shakes the tree.
    - **Tip**: If you prefer a quicker solution, you can duplicate the IK Targets from an existing tree prefab and assign them to the corresponding fields in your custom tree.

By following these steps, you'll have a fully customized fruit tree with your unique models, complete with dynamic animations, special effects, and seamless player interaction.

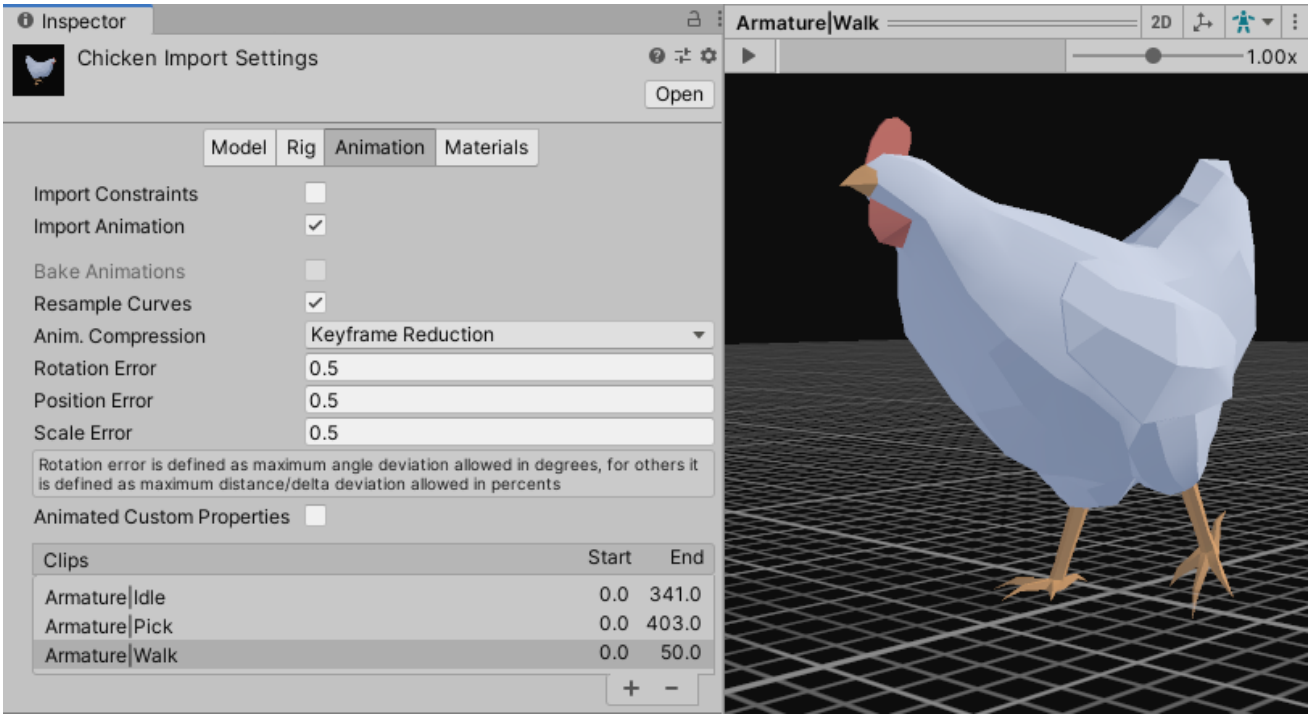
4.4 Adding New Animals

In this section, you'll learn how to add new animals to the game. There are three main categories of animals in the game: **Poultry**, **Dairy**, and **Fiber** animals. Each type of animal has unique characteristics and produces different types of goods.

4.4.1 Adding a New Poultry Animal

Poultry animals, such as chickens, provide eggs as their primary product. To add a new poultry animal, follow these steps:

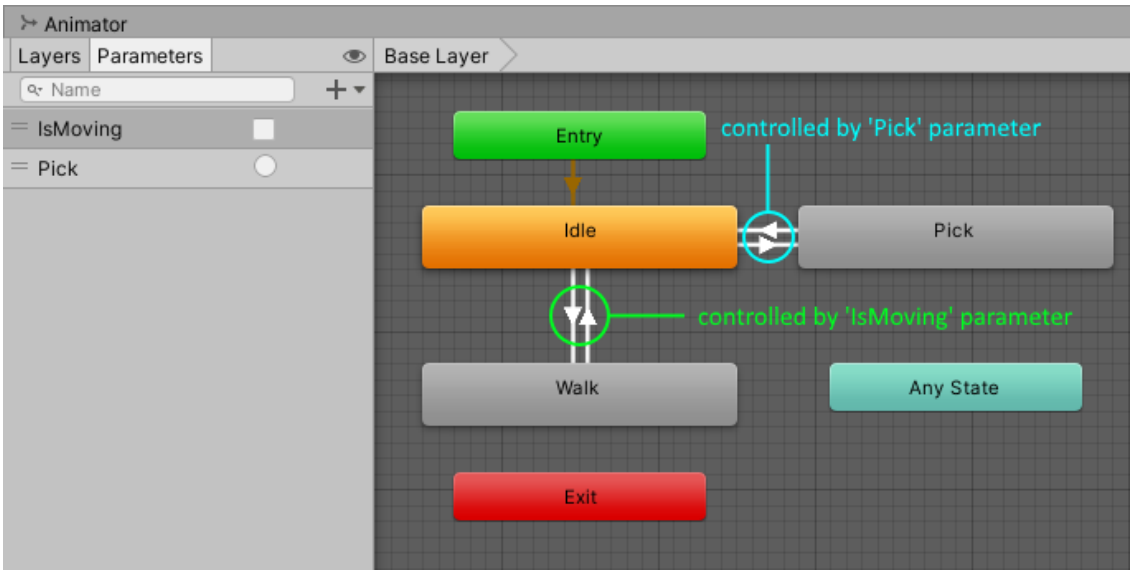
1) **Import your poultry animal model** (e.g., chicken, duck). The model must include at least two essential Generic Animations: "idle" and "walk." Additionally, it should have at least one action animation (such as eating, resting, or any other behavior you'd like the animal to perform). You can add more than one action animation if desired.



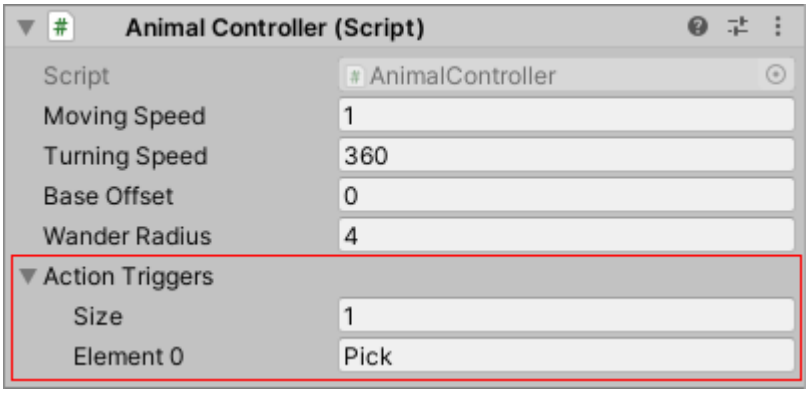
2) **Create a prefab** for this animal and save it in the appropriate folder, preferably **Prefabs > Animals**.

3) **Add the Animal script** to the prefab. This will automatically attach several required components:

   - **Sphere Collider (trigger)**: This collider triggers product collection when the player is near and the product is ready. For dairy and fiber animals, this will initiate a mini-game first. Adjust the size of the collider to determine how close the player needs to be for the collection to trigger.
   - **Audio Source**: Assign the appropriate animal sound here. This sound will play when the product is collected. Make sure to uncheck 'Play on Awake' so it only plays when triggered.
   - **Animator**: Assign the Animator Controller for the animal. The basic setup for poultry animals includes transitions between idle, walk, and action states. Here's an example of a basic animator controller setup:



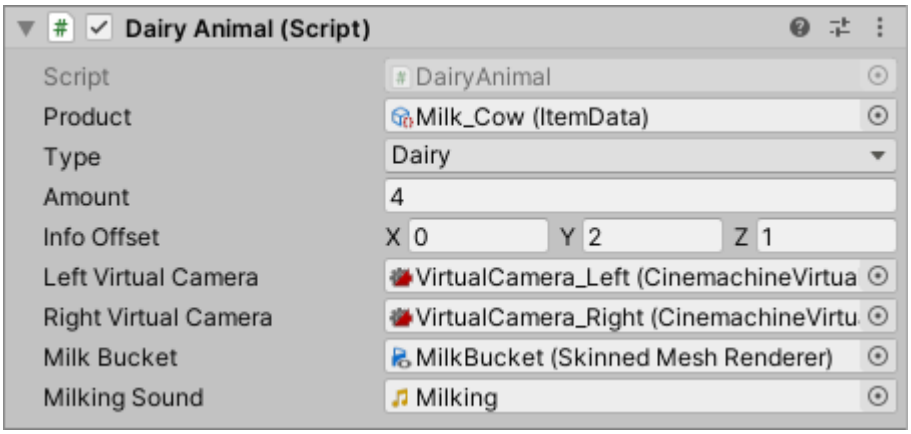
- **Animal Controller**: This handles the movement and behavior of the animal. Assign the appropriate action triggers to their respective fields in this component.



4) **Configure the Animal**: In the Animal script, assign the correct product (e.g., eggs for a chicken) and set the animal type to "Poultry." You can adjust the Amount field to determine how many products are collected each time.

4.4.2 Adding a New Dairy Animal

The process for adding **Dairy Animals** such as cow or goat, follows the same basic steps as adding **Poultry**, **with one key difference:** instead of adding the regular **Animal** script, you'll add the **DairyAnimal** script, which inherits from the Animal class. The DairyAnimal script includes additional configuration, such as the need for a product collection minigame when the player interacts with the animal.

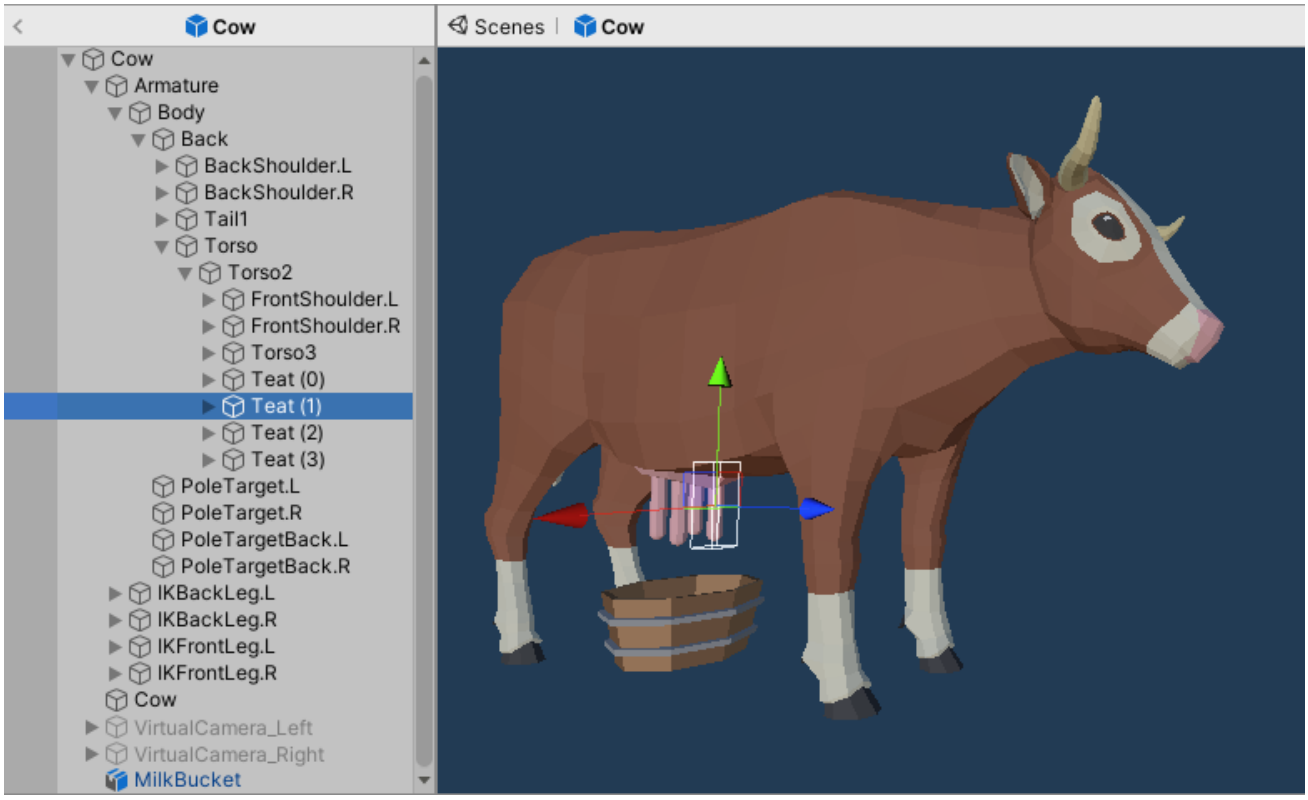


In addition to correctly configuring the product and animal type, there are a few extra steps required for Dairy animals:

1) **Attach Teat Prefabs:** Add the appropriate number of teat prefabs (depending on the animal) to the model. These should be parented to a bone near the udder or teats. You can find the teat prefabs in the **Prefabs > Animals > Parts** folder. Each teat prefab is a **Skinned Mesh Renderer** with blendshapes to simulate the milking process, and they include a milk-spraying particle effect.

   The teats' blendshapes function differently from the usual single-blendshapes adjustments. Instead, the animation is created by blending between multiple frames to simulate the action of milking. The **Teat** class manages this by adjusting blendshapes across a range of indices, allowing for smooth transitions in the milking sequence.

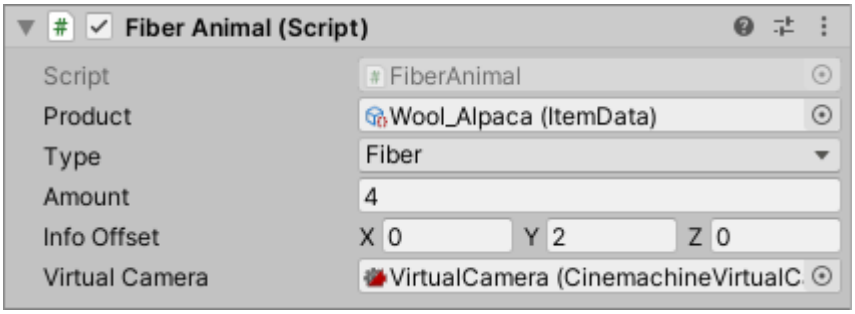
   For reference, you can inspect the Teat script, which dynamically adjusts blendshapes weights based on input values (swipes).



2) **Add Cinemachine Virtual Cameras:** Add two Cinemachine virtual cameras as children of the animal: one on the left and one on the right. Both cameras should be focused on the animal's udder/teat area. Assign them to the appropriate **Left Virtual Camera** and **Right Virtual Camera** fields in the DairyAnimal script. Make sure these camera are disabled by default.
3) **Add a Milk Bucket:** Drag the milk bucket model (located in the **Models > Animals > Parts** folder) as a child of the animal and position it directly under the udder/teats. This bucket is also a **Skinned Mesh Renderer** with blendshapes that animate the milk filling up. Assign the bucket to the **Milk Bucket** field in the DairyAnimal script.
4) **Assign Milking SFX:** Lastly, assign the milking sound effect (found in the **Sounds > SFX** folder) to the **Milking Sound** field in the script. This sound will play during the milking process.

4.4.3 Adding a New Fiber Animal

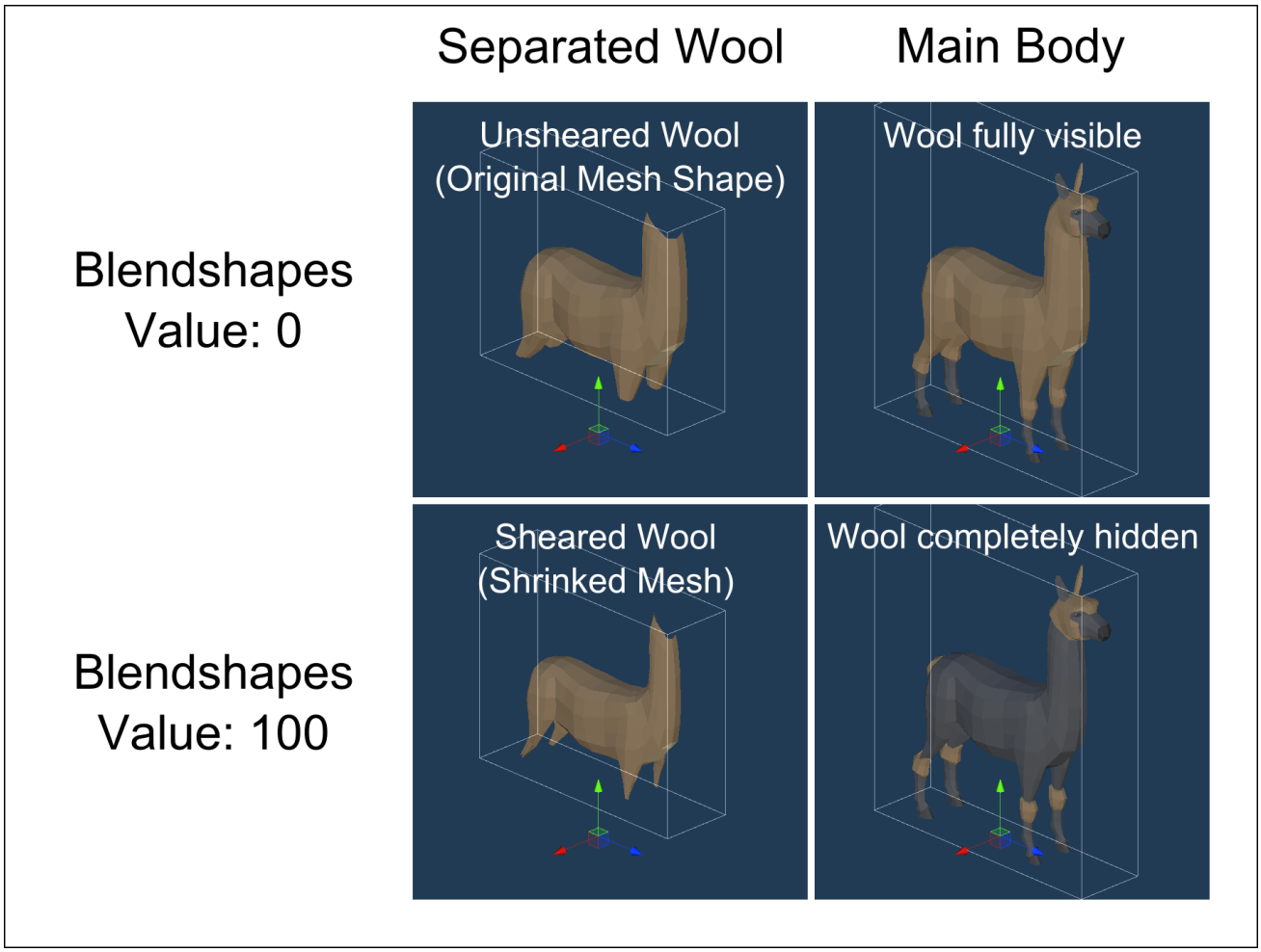
Adding **Fiber Animals** such as sheep or alpaca, follows a similar process to that of adding a Dairy animal, with the main difference being that you should attach the **FiberAnimal** script instead of the regular **Animal** script.



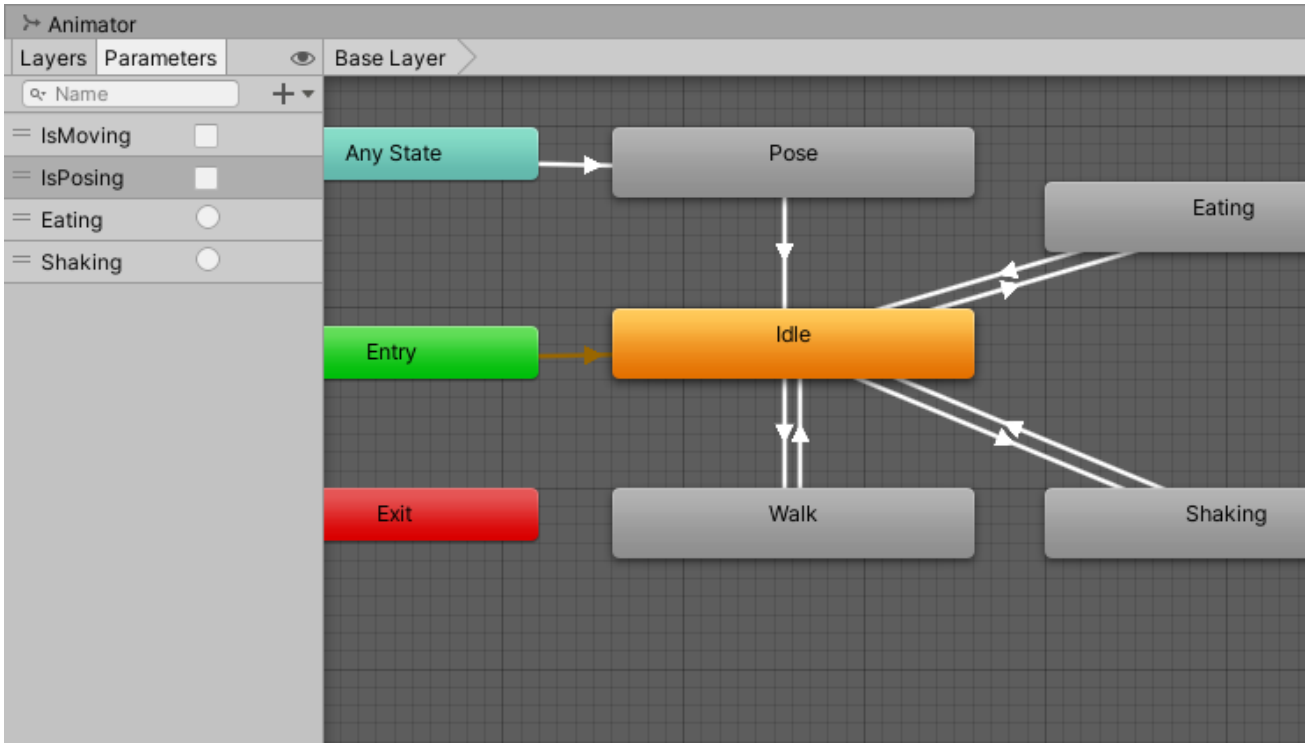
Here are the additional steps required for configuring Fiber animals:

- **Wool Blendshapes Configuration**: Fiber animals use a wool shearing mechanic that relies on blendshapes to simulate the shearing process. The animal model should include a separate child mesh for the wool, distinct from the main body, with blendshapes representing various stages of wool growth and shearing.

Here is the illustration for better understanding:



- **Add Rest Pose Animation**: The imported animal model should also include an additional animation that loops in a rest pose, similar to a T-pose in humanoid models. This rest pose will be used when the shearing minigame occurs. Ensure that the Animator Controller contains an additional boolean parameter called **IsPosing** to trigger the transition into this animation during the shearing process.



Reason: During the shearing process, the animal needs to remain still to ensure precision in the wool removal simulation. In the **Wool** script, vertices of the wool mesh are adjusted based on the player's actions, and any unnecessary animal movement would disrupt this process. The rest pose allows for smoother and more accurate interaction while vertices are being manipulated according to the shear bounds and blendshape adjustments.

- **Attach Wool Component**: Add the Wool script to the wool mesh. The default settings should work well, but feel free to adjust the values to suit your preferences. You can hover over the fields to view helpful tooltips that explain what each setting does. For the **Razor Sound** (shearing sound effect), you can find one in the **Sounds > SFX** folder.
- **Shearing Visual Effect**: Attach a **Wool Particle** to the wool for visual feedback during the shearing process. You can find a suitable effect in the **Prefabs > Animals > Parts** folder. Feel free to change the color to match the wool or fur color of your animal.
- **Create Camera Rig**: Create an empty child object that will serve as the camera rig. Then, create a Cinemachine virtual camera as a child of this rig. Finally, assign the virtual camera to the appropriate field in the Fiber Animal script. Disable this virtual camera by default.

These steps will allow players to interact with the Fiber animals through a wool shearing minigame, giving a dynamic gameplay experience.

5. Third Party Assets

This project utilizes several third-party assets to streamline development and enhance gameplay functionality. Below are the assets used, along with their respective roles within the game:

5.1 DOTween

DOTween is a fast and efficient animation engine used for tweening, which allows smooth transitions between values over time. It is used extensively in this game for various purposes, including:

- **UI Transitions**: Smooth animations for UI elements, such as fading and scaling.
- **Object Animations**: DOTween is used to smoothly animate various objects in the game, such as transitioning between different states or movements, rotating objects toward specific targets, and applying effects like scaling or fading.

For more information on how DOTween works, please refer to the official documentation at DOTween.

5.2 SimpleInput

SimpleInput is an asset designed to handle input from touch and joystick controls, which is crucial for mobile platforms. In this game, SimpleInput provides:

- **Virtual Joystick**: Used for character movement on mobile devices, allowing players to control characters via an on-screen joystick.
- **Touch Input Handling**: Provides flexible input management for different types of touch interactions, ensuring smooth control in both gameplay and UI navigation.

You can learn more about SimpleInput at SimpleInput GitHub.