# Farming Engine

Farming kit of this asset v1.04

# Brief

Farming engine is a code template allowing you to create third-person farming or simulation games.

To get started, take a look at the demo maps and start by integrating some of the prefabs into your own project. This document will explain the use of each individual script, as well as explain how to create new items, constructions, plants, characters or actions.

All the code in this asset has been structured in a way that makes it easy to customize (if you know how to code in C# of course). But the asset can also be used without coding knowledge if you're not planning on adding new features.

All items, constructions, plants and characters are using scriptable object files, so you can add new objects or edit existing ones directly from the Unity editor.

## Scripts

Here are the most important scripts in this package.

- PlayerCharacter.cs
- PlayerControls.cs and PlayerControlsMouse.cs
- TheGame.cs, TheRender.cs and TheCamera.cs
- Selectable.cs
- Destructible.cs
- Buildable.cs
- Animal.cs
- Item.cs, Construction.cs, Plant.cs, Character.cs
- CraftData.cs, ItemData.cs, ConstructionData.cs, PlantData.cs, CharacterData.cs
- AttributeData.cs
- UniqueID.cs
- PlayerData.cs

### PlayerCharacter.cs

Probably the most important script, controls the main character and allow it to interact with everything else. (Moving, Crafting, Attacking, Taking items, …).

- **move_speed**: Movement Max Speed
- **move_accel**: Movement Acceleration
- **rotate_speed**: Movement Rotation Speed
- **fall_speed**: When the character is not touching the floor, how fast does it fall.
- **slope_max_angle:** Maximum slope angle the player can climb.
- **ground_detect_dist**: Distance for detecting the ground, starting at the the surface of the capsule collider. It is usually a small value like 0.1.
- **ground_layer**: Layers that are considered ground to know if the player should fall.
- **use_navmesh**: If checked, will use the Unity Navmesh when moving? (For detecting and avoiding obstacles).
- **hand_damage**: Default damage when attacking without any weapons.
- **base_armor**: Default armor value when not wearing any equipment.

- **attack_speed**: Speed of the interval in between each attack. 100 = 1 attack per second.
- **attack_windup:** Time in seconds before in between the start of the animation and the moment where the attack will hit.
- **attack_range**: Minimum distance between the character and the target for the attack to be possible.
- **attributes:** List of character attributes and their properties (health, hunger, etc...)
- **inventory_size**: How many inventory slots you have. Make sure the UI has enough slots too.
- **craft_groups:** Crafting categories that can be crafted by default, without requiring a crafting station.

### PlayerCharacterHoe

- **hoe_item**: which item group you need to have equipped to use the hoe feature.
- **hoe_soil**: the object that is spawned on the ground (soil) when using the hoe.
- **hoe_range**: how far from the character can you hoe the ground.
- **hoe_build_radius**: must not have any other soils aready in that radius to be able to hoe.
- **hoe_energy**: how much energy it cost to hoe.

### PlayerControls.cs and PlayerControlsMouse.cs

Manager script that handle all player controls. Part of the Managers prefabs that should be included in every scene. PlayerControls is for keyboard/controller controls, while PlayerControlsMouse if for mouse and touch (mobile). Each script has events and functions that allow you to know what the player is pressing or clicking. For example, the onClickObject event will be fired every time the player clicks on an object, passing the clicked object as a parameter.

### TheGame.cs, TheRender.cs, TheAudio.cs

Managers scripts that are part of the Managers prefab (include it in every scene). TheGame handles the generic events of the game, like loading the UI, detecting if the main player is dead, or spawning objects after a save file is loaded.
TheRender is an optimization script, and will disable any object (Selectable) that are too far from the player and camera. It is not necessary but strongly recommended for large maps.
TheAudio manages all sound effects and music, its two main functions are PlaySFX() and PlayMusic().

### Selectable.cs

Most objects that are placed on the map are Selectables. These are everything the player can click on and interact with. You can attach actions or other scripts to it like Destructible, Item, Construction, Plant, Animal. (These all require the Selectable script).

- **type**: Determine how the player can interact with this. If set to Surface, the player can interact with this anywhere along the surface, instead of just at the center/pivot point. CantInteract will ignore this selectable for interaction, but will still allow being attacked.
- **use_range:** The distance at which the player needs to be to interact with this selectable.
- **actions**: List of actions that can be performed on this selectable (explained later).
- **groups**: List of groups that define this selectable. Groups don't do anything by themselves, but other script can require interaction only with selectable of specific groups.
- **active_range:** If within this range of the player, the selectables and all its scripts will be active, otherwise they will be deactivated, this is very important to optimize the scene when there are thousands of objects.

- **always_run_scripts:** Some objects need to have their script running at all time, even when this selectable is deactivated by distance. Leave it off for most objects, and only turn it on for the most important objects that need to have scripts always running.
- **outline:** Reference to the outline mesh that will be toggled on and off when the mouse is hovering this object. (optional, if null there will be no effect when hovering)
- **generate_outline:** Instead of having to setup manually a child object as outline, you can also let the engine generate the outline automatically if this is on. The first mesh found inside this object will be used as a reference for the outline.
- **outline_material:** the material used for the outline when generating the outline automatically.

### Destructible.cs

Destructibles are anything that can be destroyed by the player. They have HP and can be attacked. Resources that can be gathered are destructibles, as well as animals that can be attacked. All destructibles also need to have the Selectable script.

- **hp:** Starting health points, dies at 0.
- **armor**: Reduce all damages by this amount, damages can't be reduced below 1.

- **attack_group:** Set who is able to attack it. Ally = will be attacked by wild animals. Enemy = will be attacked by pets and wild animals (of different team). Neutral = can be attacked by anyone, but only if clicked on it. CantAttack= None can attack it.
- **team_group**: Wild animals won't attack other wild animals in the same team.
- **attack_melee_only:** If true, cannot be attacked by a ranged weapon, mostly useful for resources that you don't want the player to shoot at.
- **required_item:** Group of items that you need equipped to attack this destructible, for example in the demo, a tree can only be attacked if you equip an axe. (The value you put here is a Group, groups don't do anything by itself but it will reference items that have the same group).
- **hit_range**: Range from which target can be attacked, bigger objects should have a bigger hit size. The attacked destructible hit range is added to the attacker's attack range when checking for attack range.

- **loots:** Items that will be dropped when this destructible die.
- **loots_random**: Items that MAY be dropped, with a probability value (0 to 1)

- **shake_on_hit**: If set to true, the destructible will shake a bit and stop moving for a few miliseconds when hit by an attacked.
- **destroy_delay**: delay after death before the object is destroyed. Useful when the object has a death animation.
- **death_fx:** FX spawned when the object dies.
- **death_sound:** Sound that is played when it dies.
- **hit_sound:** Sound that is played when hit.

## Buildable.cs

Buildable is a new script introduced in 1.06, that allow player to create objects and place them on the map manually. This was previously done by Construction.cs, but since plants can now also be placed in the same way, a separate script is now doing this. Both plants and constructions require this script.

- **type**: Floor: can only be built on the floor. Anywhere: Can be built on top of other constructions. Grid: will snap to a grid.
- **build_distance**: how far from the player this can be built.
- **grid_size**: Size of the grid in grid mode.
- **floor_layer**: layers on which this building can be built.
- **obstacle_layer**: layers that are considered 'obstacles' when trying to place the object. You won't be able to place the construction on top of an obstacle.
- **build_obstacle_radius:** Minimum range needed in between this buildable and another object, this restriction is in addition to the obstacle layer check.
- **build_ground_dist**: the ground must be at least this distance on all 4 sides of the buildable. This will prevent you to build on a terrain that is too steep. The smaller the number is, the flatter the terrain must be.
- **build_audio**: Sound played
- **build_fx:** FX spawned when this is built

## Item.cs, Construction.cs, Plant.cs and Character.cs

These are directly linked to their data section. Except that these ones are MonoBehavior and added to the prefab that is put on the map. While the ones ending with 'Data' are scriptable objects and just contains the data (but not attached to a prefab). Items are objects that you can add to your inventory or equip. Constructions are objects that you can build and place on the map. Plants are objects that can spawn and grow on the map, some of them also grow fruits/vegetables.

- **Data:** The data scriptableObject used by this object. (See next section).

## Plant.cs

- **growth_stage**: Current stage of this prefab, some plants have multiple growth stages, each of them being a different prefab and mesh. Starts at 0.
- **time_type**: Whether the grow_time params are in game-hours or in game-days. If in days, the growth will happen at the beginning of the day.
- **soil_range**: Soil will be considered in range of that plant if within this range. Soil that are in range provide water to the plant if the soil is watered.
- **grow_time**: time it takes for the plant to grow (in game hours or game days, not in real hours).
- **Grow_require_water**: If true, the plant will stop growing if it is not watered.
- **regrow_on_death**: When this plant is destroyed, it will return to first growth stage instead of being destroyed.
- **fruit**: item that can grow and be harvested from this plant (it's called fruit but also works for veggies!). Set to null if no fruits are growing.
- **fruit_grow_time**: How long before a new fruit appear after one has been harvested (in game hours or game days, not in real hours).
- **fruit_require_water**: If true, the fruits will stop growing if the plant is not watered.
- **fruit_model**: the mesh to hide/show when a fruit is available.

- **death_on_harvest**: Means that the plant will be destroyed after the fruit is harvested (Makes sense for something like lettuce or a carrot).

### Character.cs

Generic character script that can be used for NPC, pets, etc. Give it orders by calling the functions like MoveTo(). Compatible with the navmesh.

- **move_enabled:** If on, this script will manage movement.
- **move_speed, rotate_speed:** speed of movement and rotation
- **avoid_obstacles:** If turned on, will detect obstacles in front of them and turn left or right instead of trying to go through it. More performant but less accurate than unity navmesh.
- **use_navmesh:** If turned on, will use Unity's navmesh to go around obstacles when moving.
- **fall speed:** How fast it falls when not on the ground, set to 0 for flying characters.
- **slope_max_angle:** Maximum slope angle the character can climb.

- **ground_detect_dist:** Offset distance for detecting the ground. It is usually a small value like 0.1. Smaller values have more precisions but also have more chance of missing to detect the ground if falling too fast.
- **ground_layer:** Layers that are considered ground to know if the player should fall.

- **attack_enabled:** Can this character attack? If yes, functions must still be called from another script (like animal).
- **attack_range:** how far it can attack
- **attack_damage:** damage value of each attack
- **attack_cooldown:** time in second between each attack
- **attack_windup:** time in seconds between the start of the animation and the attack 'hit'
- **attack_duration:** time in seconds of the attack (movement is disabled during that time). Should be higher than the windup or the attack will stop before the hit.
- **attack_audio:** sound played during the attack hit.

### AnimalWild.cs

**wander_speed:** Move speed when in wander state.
**run_speed:** Move speed when escaping or chasing player
**wander_range:** How far from starting position can this animal wander.
**wander_interval:** How often the wander position target is changed.
**detect_range:** How far can this animal detect the player
**detect_angle:** What angle can this animal detect the player (like a cone in front of the animal).
**detect_360_range:** Within this range, the animal will detect player from all angles, even if the player is behind.
**reaction_time:** How long it takes for the animal to react to a threat after its detected.

### AnimalLivestock.cs

**wander_range:** How far from starting position can this animal wander.
**wander_interval:** How often the wander position target is changed.
**detect_range:** How far can this animal detect food
**time_type:** is the eat_interval, item_produce_time and grow_time in game-days or in game-hours?

**eat:** Does this animal eat food?

**eat_range:** How far from the food must the animal be to eat.
**eat_time_type:** Is the eat_interval_time in game-hours or game-days?
**eat_interval_time:** how often does the animal need to eat ? Time is either in game-hours or game-days.

**item_produce:** Which item does this animal produce. Produced items will be spawned on the floor near the animal.
**item_collect_type**: When an item is produced, is it dropped on the floor, or does the player manually need to collect it. When collected, it needs to be done through an action that calls the CollectProduct() function.
**item_eat_count:** How many time does the animal need to eat before it can produce 1 item?
**item_produce_time:** How long to wait before producing another item (in game-hours or game-days).
**item_max:** If there are more than X items already on the floor in range, animal will stop producing items.

**grow_to**: which animal can this animal grow into.
**grow_eat_count:** how many time does this animal need to eat before it can grow.
**grow_time:** how long before this animal can grow. (In game-hours or game-days).

Note: to make an item or destructible edible by animals, you must attach the script AnimalFood.cs to it.

### AnimalRide.cs

Add this script to an AnimalWild or AnimalLivestock to make it ridable.

**ride_speed:** how fast the animal moves while in riding state.
**ride_root:** to which bone object will the character attach to while riding.

### CraftData.cs, ConstructionData.cs, PlantData.cs, ItemData.cs, AttributeData.cs

Scriptable objects that contain all the stats and data of items, constructions and plants. CraftData is the parent object and represent things that can appear in the crafting panel. ItemData, PlantData, and ConstructionData inherit from CraftData. See the data section for a better understanding of the properties of these objects.

### UniqueID.cs

Unique ID of the object, used by the save system. Each instance of an object that have properties saved in the save file should have a Unique ID. Without a unique ID, nothing can be saved about this object. Add only to things that need to be saved.

### PlayerData.cs

Basically, the save file. All variables inside this script will be serialized to a save file when the game is saved. It also contains multiple function that allow you to easily access or update the data contained inside the save file. Make sure to call Save() to save the game. (There's no autosave by default).

# Data (Scriptable Objects)

Scriptable objects are the files that contain all the data about items, buildings, plants, and characters. Each object is represented by a file with all it's properties. Take a look inside the Resources folder to see all the data files. The files are located in that folder so that the game can load them automatically. To create a new data asset, right click in the Project Files window and select Create->Data. You can also duplicate an existing one with Ctrl-D.

## GameData

Contains generic data about the game. There should be only 1 of this. In the demo, that file is located inside. It is referenced by the Managers prefab.

- **game_time_multiplier**: factor that determines the in-game hours speed, compared to real hours. For example, a value of 24 means that 1 day in the game will be 1 hour in real life.
- **light_dir_intensity (day or night):** Intensity of the directional light during the day and during the night.
- **light_ambient_intensity (day or night):** Intensity of the ambient light during day and during the night.
- **FX:** References to various FX instantiated during the game.
- **Folders:** folders where data files are located (within the Resources folder)

## GroupData

Groups don't do anything by themselves, but they are useful for grouping objects and reference them by groups (instead of one by one). For example, you could have an action that can only be performed on a specific 'group' of objects. Groups can have a title and icon to represent them in the game.

## CraftData

Parent script of ItemData, ConstructionData and PlantData. Does not do anything by itself, but contain the common data that are in all other data types.

- **ID**: Used to reference that object data type, and for the save file, make sure all objects have different ids.
- **title/Icon/Desc**: Displayed values and image in the game (for example in the crafting UI).
- **groups**: Groups to which belong this object. (See GroupData section for more info). This can also be used to determine in which crafting tab the object will appear.
- **craftable**: true if it can be crafted by default. Set to false if you want this to be unlocked through the learn action
- **craft_quantity**: how many are crafted (items only)
- **craft_duration:** how long it will take the character to craft (in seconds)
- **craft_sort_order**: Ordering in the crafting menu. Small numbers appear first.
- **craft_near**: If set, the player need to be near the specified object (group) to craft that item (for example, some crafting receipe require a source of water or fire).
- **craft_items**: Items requires and consumed when crafting this object.
- **craft_requirements**: Objects that need to be in the scene (ex: constructions) to be able to craft this.

## ItemData

An item is an object you can hold in your inventory. Items can be crafted as they inherit from CraftData.

- **type**: Equipments can be equipped. Consumed can usually be eaten/drank/used. Basic have no special effects and are usually crafting materials.
- **inventory_max:** Maximum quantity of this item that can be held in a single inventory slot.
- **durability_type:** How is durability calculated. None=Never expire, UsageCount = each time you use the item (attack, or receive hit, etc.) UsageTime=Expire over time only when equipped. Spoilage=Expire over time no matter where it is.
- **urability:** In number of uses (UsageCount) or in game hours (Others).

**Equipments:**

- **equip_slot:** For equipement only, determine in which equipment slot this item can be equipped to (hand,head,foot,body).
- **armor:** armor gained when equipped
- **equip_side:** For hand equipment, tells if should be held in left hand or right hand.
- **equip_bonus:** Bonus effect applied to the character when equip (ex: speed boost)

**Weapons**:

- **weapons:** If true, will use its stats when attacking.
- **ranged:** Can shoot projectiles (if you have the projectile).
- **damage/Range:** Stats of the weapon.
- **strike_per_attack:** Number of hit, or bullet shot (if ranged) per attack animation.
- **strike_interval:** Time in seconds in between each strike, usually you want a really low number here like 0.1.

**Consumable:**

- **eat (HP,Hunger,Thirst,Happiness):** Attributes gained when consuming this item.
- **eat_bonus:** Bonus effect applied for a duration when eaten.
- **eat_bonus_duration:** Duration of the bonus effect in game hours.

**Other:**

- **actions:** List of actions that can be performed with this item while in your inventory.
- **container_data:** If set, when consumed, the item will be replaced by its container. So, if you have a water jug, the empty version of the jug would be set here.
- **plant_data:** Plant that will be created when you sow this item (for seeds)
- **construction_data:** Construction that will be created when you build this item.
- **projectile_group:** For ranged weapon, which group of items is used as a projectile.
- **item_prefab:** The prefab of the item when it is dropped on the floor. That prefab should have the Item.cs script.
- **equipped_prefab:** The prefab that appears attached to the character when this item is equipped. That prefab should have the EquipItem.cs script
- **projectile_prefab:** For projectile (not for the weapon), set the prefab that is spawned when this projectile is shot. That projectile should have the Projectile.cs script.

## ConstructionData

A construction is an object that can be built and placed on the map by the player.

- **construction_prefab:** The prefab that is spawned when this construction is built (crafted). That prefab should have the Construction.cs script.
- **take_item_data:** Only for construction that can be picked (lure, trap), this is its item equivalent.
- **durability_type:** How is durability calculated. None=Never expire, UsageCount = each time you use it (attack, or receive hit, etc.) UsageTime=Expire over time only when equipped. Spoilage=Expire over time no matter where it is.
- **durability:** In number of uses (UsageCount) or in game hours (Others).

## PlantData

A plant is an object that can be placed on the map (Usually with a seed). It will grow into different stages. The final stage can often produce items (fruits or veggies).

- **plant_prefab:** Default prefab of this object when spawned on the map. It should have the Plant.cs script.
- **growth_stages_prefabs:** Array of prefab of each stages of the plant growth. The size of this array determines the number of stages of this plant. When a plant grows (after a set duration) the plant will be replaced by the next stage.

## CharacterData

- **character_prefab:** Default prefab of this object when spawned on the map. It should have the Character.cs script.

## AttributeData

Attributes represent the life meters of the player. In order to survive, the player must keep its attribute in check. The health attribute will cause the player to die when it reaches 0.

- **type:** Reference to this attribute, acts as an ID to reference it elsewhere.
- **start_value:** The value the player starts with.
- **max_value:** Maximum value of this attribute.
- **value_per_hour:** Value gained or lost (if negative) of that attribute each hour (in game hour, not real hours).
- **deplete_hp_loss:** When this attribute reaches 0, the player will start losing HP at this rate (in game hours). Set a negative value to lose HP.
- **deplete_move_mult:** When this attribute reaches 0, the player will start moving slower by this factor (set a value between 0 and 1).

# Actions

Actions are a powerful tool that allow you to define your own player interactions with objects in the game. Some Selectable or Items will have a list of possible actions you can do when you click on them. You can code a script to create your own action by inheriting one of the three base action scripts: SAction, MAction or AAction.

Once you created a new script action, you must create a data file for this action by right clicking in the project window: Create->Data->Actions. OR you can also duplicate an existing action if you want a similar behavior with different properties. For example, you could duplicate the action 'Fill Water' and make a 'Fill Juice' action if you had a juice source in the game. In that example, they would both use the same action script but would have 2 different action files.

After you create your own action data file, reference it on a Selectable or ItemData. Actions put on a selectable will appear when the player click on that object in the scene. Actions put on an ItemData will appear when the player right-click on that item in their inventory.

You can take a look at all the existing actions included in this demo in the Resources/Actions folder. And also, the Scripts/Actions folder for their scripts.

### Creating your own actions scripts

If you are up to the challenge to create your own action scripts, there are 2 important functions you will need to override: DoAction and CanDoAction
DoAction is the code that is ran when the action is performed.
CanDoAction returns a bool that tells if the action can be performed or not, for example it may check if you have the required items to perform it. If the action has no conditions you may just return true.

Actions all need to inherit from one of the three base action scripts:

**SAction**: Are the basic 'Standard' actions. When the player clicks on an object with such actions, a UI list will popup and the player can select which action to perform. The action will be performed when the player selects one of them. If applied to an Item, the UI will popup when the player right clicks on that item in their inventory instead.

**AAction**: Are 'Automatic' actions. They are performed automatically when the player clicks on the Selectable in the scene and if the action conditions are respected. You can only have 1 action of this type on any given object.

**MAction**: Are 'Merge' or 'Mix' actions. They are performed by combining two items (example: axe and coconut to open it), or by combining one item with a selectable (example: jug and water source to fill it, or raw meat and fire to cook it).

## Improving the Engine

If you think that something important is missing or if you notice a bug. Feel free to contact me and I will take note of every request. The most popular ones will be added to the feature list for the next version.

If you have any questions or suggestions send me an email:
contact@indiemarc.com

Or join Discord:
https://discord.gg/JpVwUgG

Thank you!

## Credits

Programming and Unity integration
Indie Marc (Marc-Antoine Desbiens)
https://indiemarc.com

3D Art
Antonio Maričić (https://www.instagram.com/mocacinodesign/)
Algot Nordström (https://www.fiverr.com/algot_n)
Marwan Hany (https://www.fiverr.com/marwan_hany)
Pogwitch (https://www.fiverr.com/pogwitch)

Animation
Naufalrezki (https://www.fiverr.com/naufalrezki)
Hamzasketches (https://www.fiverr.com/hamzasketches)

2D UI
Helen Lien (http://helenlien.com)
Ayes Studios (https://www.fiverr.com/ayesstudios)

SFX
Andey Fellowes (https://soundcloud.com/andeyfellowescomposer)