Project Overview

Game Title: Marmota Meadow
Version: v1.0.0-release
Target Platforms: PC (Windows Only)
Engine & Tools: Unity (v2023.2.20f1, Universal Rendering Pipeline), Github, Jira
Repository: https://github.com/OverheatStudios/MarmotaMeadow

Help I’m not a programmer!

The following sections are useful to non-programmers:

- Development Process > Git Guide
- Development Process > Asset Guidelines
- Naming Conventions > General Rules
- Naming Conventions > Files
- Game Configuration

Development Process

- Create a branch for each feature, don’t directly push to main
- Profile features using Unity Profiler
- Once the feature is implemented & approved by relevant team members (producer/a designer/an artist) a pull request should be made so another programmer can review the code.

Unity Setup Guide

You must commit and push your work for it to be available on other PCs, don't forget to do this before you log out of university PC's.

One Time Setup

- Create an account on https://github.com/ (recommended to use your university email)
- Send the email you used to Jack and wait to be invited to the repository.
- Once you have been invited, you will need to accept the invitation (check your emails)
- Download https://github.com/apps/desktop and clone the repository. Github Desktop is available on all university PCs.
- Download https://unity.com/download and install Unity v2023.2.20f1. Unity is available on all university PCs (make sure to launch Unity v2023.2.20f1 from apps anywhere, **not Unity Hub**)
- In Unity, select "Projects", then "Add" then "Add project from disk". Navigate to the folder where you cloned the github repository and select the Prototype folder (DES315/Prototype)
- Open the project, this will take a few minutes on first launch.
- Open the relevant scene by going to Assets/Scenes in the Project panel and double clicking on the scene.

Per Feature Workflow

You should do this for every "feature" (e.g. main menu ui assets, or new groundhog variant)

- Ensure you are on the main branch, pull any changes.
- Create a branch
- Switch between 2D and 3D view by clicking the 2D/3D button in the top right.
- Implement the feature (adding the assets into the game, etc)
- Run the game and test it works by pressing the play button at the top center of the screen. This will run the current scene.
- Commit and push your changes to your branch.
- Ask a programmer to check your work for any technical or performance issues, they will handle merging your branch (adding it into the game) back into main.

Asset Guidelines

- Create greybox or programmer art for missing assets
- Models are preferably FBX format
- Check triangle count for models, ideally all models should be 3-4 digit triangle count (may vary based on asset importance)
- Model textures and materials should be stored in the same directory (or a Textures subdirectory) as the main model file
- Textures are preferably PNG format (JPG can be used if loss in quality isn't noticeable)
- Check texture files aren't unnecessarily large (e.g if a texture takes up 5% of the screen maybe it should be 5% of 1920x1080 and make sure it isn't 4000x4000 pixels)
- Check texture file sizes are powers of 2 (these generally perform better due to how gpus work, mipmapping, etc)
- Audio (both sound effects and music) should be OGG format
- Check sound effects are mono (unless stereo is necessary for sound quality - usually not)
- Check audio files have a reasonable sample rate (44-48khz in most cases)

Versioning

Version Format: major.minor.patch-identifier (e.g 0.1.0-alpha)
Major version should be incremented to 1.0 when the game is finished.
Minor version should be incremented each sprint.
Patch version should be incremented each time we make a new build for play testers.
Pre-release identifier should be pre-alpha for development, alpha for when the game is playable and beta when we get to polishing (final sprint)

Code Reviews - Things to look for

- Inconsistent naming conventions
- Bugs
- Hard to read/maintain code
- Performance issues
- Undocumented code
- Magic numbers

Core Systems

General Rules

- Use Update() and delta time over FixedUpdate unless FixedUpdate is required
- Delta time should still be used even in FixedUpdate
- Game shouldn't crash if it encounters an error (missing asset, corrupted save, etc)
- Use assert statements to help make sure the game is robust (e.g assert the parameters in public functions are valid)
- Use 0 based counting and indexing for everything, just add 1 if it is shown to player (e.g in code first night is 0, but if you need to tell player what night they are on, tell them (night+1)
- Don't prematurely optimise, code something that works and if there's no bugs and no performance issues then it's good.
- The debug menu can be annoying and get in the way when designing scene UI, if you need to disable it, you should disable the child panel, do not disable the entire prefab or the menu will break.
- Use seconds as the default unit of time.

Groundhog Type Creation Guide

- Create new value in GroundhogType enum
- Create new groundhog type prefab (copy PrefabBasicGroundhog), make sure to fill out the GroundhogTypeInfo MonoBehaviour script
- Add value to prefab list in ScrObjGroundhogTypes

Item Type Creation Guide

- Create a new ScriptableObject with the correct type
- Ensure it is named according to Asset Prefixes guidelines
- Fill out inspector values, see Game Configuration > Items for more information.

ScriptableObject Item Types

- BaseItem: An item with no functionality.
- Seeds: A plantable item
- Tool: A tool
- Gun: An item that can shoot groundhogs

Player Starting Inventory Modification Guide

- 1st go to the BaseSave/playerData.json
- 2nd to the 3rd las curly bracket add a coma and then open and close curly brackets then press enter
- Then like the other items they need to have: slotname, amount, multiplier, item name (If you want you can just copy and paste and just change values)
- Press save

Should look like this.

Input System

- The player has the ability to rebind most actions in the game to any button on their mouse or keyboard or controller.
- To create a new action, add a new keybind to the controls section of the UI (copy the Interact object then rename it)
  Use GameInput.GetKeybind(name) to get a GameControl which you can check if it is up, down, or pressed, where name is case sensitive and the name of the UI GameObject you just created.
- Player controls are stored in the PlayerPrefs.
- Player movement is hard coded to WASD or left joystick.
- Player can move the mouse cursor using right joystick on a controller and left click by using the A key, note that right clicking on a controller isn't possible at the moment.
- Switching selected item is hard coded to RT, LT on a controller and 1-9 on keyboard and scroll wheel on mouse.

Game Saves

- Serialized into json files
- Game version should be included in all json files
- Multiple files per save is acceptable
- Settings should be save-independent
- Saves are stored in the game data folder, settings are stored in appdata/locallow

Save Manager

The SaveManager scriptable object manages game saves (creating & switching saves).

- When creating a new save, it copies from the “BaseSave” folder, this functionality can be used to make the player start with certain items for example.
- All saves are stored in the GameSaves directory.
- BaseSave is not stored in the GameSaves directory.
- When running the game without selecting a save (meaning using the editor to run a scene, or using the debug menu to skip the menu) a save called “Test” will be used.
- BaseSave is not git ignored, but all game saves are.
- Do not attempt to access a save in Start() function **on scriptable objects** this has undefined behaviour.
- If you want to read or write to a file part of a game save, use the GetFilePath(string) function in SaveManager to convert a file name into an absolute path in the correct save directory.
- Saves are stored in Application.dataPath/GameSaves. When running in editor, dataPath is the assets folder, and when running a standalone build dataPath is gamename_Data
- **BaseSave is not copied into the gamename_Data folder on build and must be done manually at the moment.**
- **A GameObject with the GlobalDataSaveLoad script must exist in every game scene (not the main menu / game over though!)**

- Settings

Settings are saved to settings.json and are consistent across different player saves.
To add a new setting, add a new UI element in the correct settings category, add a field to the Settings class in SettingsScriptableObject (provide a default value, getter/setter, update ValidateSettings & Reset), and in MenuButtonScript update the Start and Update functions.

Item Manager

To access the inventory in the scene, instantiate an Inventory prefab and fill out the exposed fields (mainly save location, which should be playerData.json)
The ItemMager (typo that we haven't renamed yet) is the main script you need to interact with the inventory, it can be used to get, add, and remove items from the players inventory.
To register new item types, see the "Items "section in "Game Configuration"
The inventory UI is split into the MainInventory and the Toolbar, the main inventory can be opened by pressing E in-game. The toolbar inventory is separate from the MainInventory inventory, however the utility functions in the item manager will take both inventories into account.

Minigames

There are 3 different plant based minigames: Tiling, Watering and Harvesting

The watering and harvesting minigames work basically the same. On the watering minigame we start by calculating the points that make a hexagon while on the watering minigame we randomize the first point's position and then get the rest of the points to make a line that goes across the plant. After that we draw the line and add colliders to the points. In the end we just check if a raycast has collided with the collider and delete them if that's the case. When there are no more colliders we just say that the minigame has finished and call the function on the respective plant to end the minigame.

Tilling is based on the tilling mechanic from Palia: How to Till the Soil in Palia #tutorial
The user has to click on the plot using their hoe in a lot of different areas of the plot to fully till it. We render both the tilled and untilled models, but the untilled model is scaled up to completely encase the tilled model, then whenever the user uses their hoe, we raycast against a box collider (this is a different collider than the farm plot one, since that is too tall) for the point hoe, we spawn a "mask" object at that location which cuts out part of the untilled model allowing the tilled model to show. Once enough percentage of the plot has been tilled, the untilled model is hidden and a short animation is played on the tilled model. The mask model was generated from a sprite using the minecraft model generator tool. Masking is implemented based on this tutorial: How To Mask 3D cut-out with mouse using shader [Unity Tutorial]

Player Movement (GDD 2.2a)

This is implemented by adding a rigid body to the player and setting the velocity when the player presses movement keybinds.
Camera rotation is changed based on mouse delta every frame.
Jumping is implemented by setting the player's Y velocity to a positive value when the player presses space and hasn't collided with a collider below it within the last 10 frames and hasn't jumped within the last 0.1 seconds.
Crouching is implemented by lerping between two different camera Y positions and lerping between two different player collider heights.

Farming (GDD 2.2b)

Every time the player presses the lmb it shoots a raycast. If it hits a plot or plant it will call a function called ChangeState that also receives that item you are currently holding. If the item you are currently holding is the right one to change the plant's state it will return true, else it will return false. In the case it returns true it will try to change the plant's state.

In case the plant is on the state normal it will require the hoe to advance to the state tilled. The state tilled needs a seed to advance to be planted. Planted neads the watering can to go to watered and after a couple of seconds it will change to completed. And then you nead the scythe to go back to normal

Shopping (GDD 2.2c)

New guns and more seeds can be purchased by clicking a button in the shop. When the scene is loaded the new gun button is destroyed and replaced with “already bought” if the player’s inventory already contains the gun. This also happens on purchase of the gun.
Upgrading tools is handled by creating a new inventory slot in the shop where the player can put items, if the player puts an invalid item then an error image is shown, else if the player put a tool then an option to upgrade the tool and a price is shown. Upgrading the tool increments the level which is stored in the item.
Selling items is handled by having a sell price on each item type and creating a new inventory slot in the shop, if a player puts an item in there and the sell price is greater than 0 then a button to sell and the price is shown, else if an invalid item was put in the slot an error image is shown.

Defending (GDD 2.2d)

Several burrow objects are placed in the night scene, during the night a set amount of groundhogs will spawn in random burrows at set items. They are spawned below the burrow, the Y position is increased until the groundhog is fully above the burrow, then the groundhog idles before the Y position is decreased. Once the Y position reaches the original Y position the groundhog is destroyed and the player's health is decreased.
Crop growth speed in the following day is determined by the player's health at the end of the previous night, after each day the health resets.
Night ends a few seconds after the last groundhog has been spawned.
Ranged groundhogs will spawn a projectile every few seconds and give it velocity pointing slightly above the player.

Technical Goals & Risks

Goals

We are aiming to develop a game with cartoony graphics which aren’t too high res, this means performance isn’t our primary concern, though the game should run smoothly on low spec machines (see performance budget section). Polish is our primary concern and all mechanics should be thoroughly tested internally and externally for bugs and UX.

Performance Budget

Unity Minimum Specs

Unity Player Minimum Specs:
https://docs.unity3d.com/2023.2/Documentation/Manual/system-requirements.html#player
Unity Editor Minimum Specs:
https://docs.unity3d.com/2023.2/Documentation/Manual/system-requirements.html#editor

Recommended Specs

The game should run at 8.33ms per frame (120 FPS) on university PC's. This is a lenient goal for a game with our scope and graphical quality, so we can afford to implement some features in a non-performance orientated way.
Processor: 12th Gen Intel(R) Core(™) i7-12700 (20 CPUs), ~2.1GHz
https://www.intel.com/content/www/us/en/products/sku/134591/intel-core-i712700-processor-25m-cache-up-to-4-90-ghz/specifications.html
Graphics: NVIDIA GeForce RTX 3070
https://www.techpowerup.com/gpu-specs/geforce-rtx-3070.c3674
Memory: 32GB (4400 MHz)
OS: Windows 11
Graphics API: DirectX 12
Storage: 1GB Available

Minimum Specs

These are the lowest specs that we have been able to test the game runs smoothly (16.66ms per frame / 60FPS at 1080p) on, the actual minimum specs are probably lower.
Processor: AMD Ryze n5 4500U (6 cores, 2.38GHz)
https://www.techpowerup.com/cpu-specs/ryzen-5-4500u.c2285
Graphics: Integrated AMD Radeon Graphics Processor
https://www.notebookcheck.net/AMD-Ryzen-5-4500U-Laptop-Processor-Benchmarks-and-Specs.449978.0.html
Memory: 8GB
OS: Windows 10 version 21H1 (build 19043) or newer (Unity Player minimum specs, we've only tested Windows 11)
Graphics API: DirectX 10/11/12 or Vulkan (Unity Player minimum specs, we've only tested on DirectX 12)
Storage: 1GB Available

Risks

Developing Two Games

We are essentially developing two separate games, a night and day game. This will increase the technical complexity of the game, however due to the night and day both being relatively simple, this shouldn’t be a major concern.

External Libraries

Unity v2023.2.20f1 (Universal Rendering Pipeline)

- https://unity.com/

We are using the Unity game engine to create our game. This is because it is an engine our team is familiar with and it is the best engine for a 2D and 3D hybrid game such as ours.

Discord Rich Presence v1.3.0

- https://github.com/Lachee/discord-rpc-unity

This is a Unity package that adds Discord rich presence integration allowing Discord to recognise when someone is playing our game.

ProBuilder v5.2.3

- https://docs.unity3d.com/Packages/com.unity.probuilder@6.0/manual/installing.html

This is a Unity package that adds additional level design features to Unity.

Game Configuration

Game configuration (e.g gun damage, enemy spawning) should be modifiable using the inspector to allow game designers to test & balance easier. These should have tooltips providing a description of the setting. Below is a list of some of the available settings.

Global

- Num nights (the number of days is this + 1) (ScrObjNumNights)
- Money required to win (ScrObjNumNights)
- Not harvestable redness on harvest attempt (Player prefab, Actions on Item object)
- Plant wind intensity (Billboard script)
- Plant random rotation toggle (Billboard script)
- Plant tilling minigame + tilling animation values (Plant prefab, Soils object)
- Coin gain animation (CoinManager in Shop and Night scenes)
- Shop item tooltip (Parent object of each item in the shop, RectTransform should probably be the Image and not the parent)
- Action tooltip duration, fade out duration, rise speed, min and max scale, textures (Prefab in Assets/UI/Tooltips/Actions)

Player Movement & Camera

- Camera base sensitivity (multiplied by the users settings) (Player prefab)
- Player movement speed (Player prefab)
- Aim sway values (Main Camera object in the Player, maybe modify this per scene so that day doesn't have sway? Includes per-night options)
- Aim recoil values (Main Camera in Player prefab)
- Minecraft item model transform (Player prefab, Hand object)
- Maximum player interact distance (Player prefab, CameraScript)
- Crouch animation duration, speed penalty, collider and camera modifications (Player prefab, Movement script)
- Player maximum negative Y velocity (Player prefab)
- Movement speed penalty when in air (Player prefab)
- Required downward velocity to be considered "in air" (Player prefab)

Footsteps

Add the FloorTypeScript to any object that should produce footsteps sounds when you walk over them.
To add a new floor type, **a programmer** can add a value to the FloorType enum and add the AudioClip to WalkingSfx in the MovementScript on the Player prefab.

- Footsteps fade out duration when stop walking (MovementScript on Player prefab)

Particles

- Tilling particles (PrefabTillingParticle, and Soils object in Plant prefab)
- Harvesting particles (PrefabHarvestingParticle, and Soils object in Plant prefab)
- Blood particles (PrefabBloodParticle, and PrefabGroundhog)

Items

Items can be configured by modifying the scriptable object. See Assets/Player/items

- Item model
- Is minecraft model (was item model generated by minecraft model generator tool?)
- Item name
- Is stackable & max stack size
- Icon
- Seeds Grow Duration
- Seeds drop amount
- Crop sell price
- Gun damage
- Gun bullets per shot
- Gun shoot cooldown
- Gun reload cooldown
- Gun max ammo
- Gun bullet spread
- Gun bullet hole size (decal)
- Gun swap cooldown
- Gun recoil force
- Gun reload sfx
- Gun shoot sfx
- Gun camera shake

Item Purchase Price

You can change the sell/upgrade/purchase price of an item on its scriptable object. The Shop UI will automatically update when the game is running, so it doesn't need to be changed.

Night

If you can specify different values for different nights, if the current night is past the last one specified, the last specified value will be used. Nights are 0 indexed, so element 0 is the first night.

- Hurt overlay colour, animation duration, maximum opacity (HurtUI object)
- Groundhog damage to player (GroundhogEscapeScriptableObject)
- Groundhog bonus health (Burrow prefab)
- Groundhog health (Groundhog type prefab)

- Groundhog uptime (Groundhog type prefab)
- Groundhog (y axis) movement speed (Groundhog type prefab)
- Groundhog projectile attack interval (Groundhog type prefab)
- Groundhog projectile scale (Groundhog type prefab)
- Groundhog projectile speed (Groundhog type prefab)
- Groundhog projectile inaccuracy (Groundhog type prefab)
- Groundhog projectile damage (Groundhog type prefab)
- Groundhog projectile velocity distance scalar (Groundhog type prefab)
- Groundhog projectile additional Y velocity (Groundhog type prefab)
- Groundhog projectile spawn offset (Groundhog type prefab)
- Groundhog coins dropped on kill (Groundhog type prefab)
- Burrow count (BurrowContainer)

Groundhog Spawning

Spawns can be configured in the AllGroundhogSpawns scriptable object (Assets/Night/Enemies/Groundhog).
Night length is calculated automatically but the number of seconds between spawning ending and night ending can be configured in DayCountdownText
For each night, you can specify a list of “groundhog spawn sets”. Each spawn set is zero or more groundhogs of the same type.

- SecondsSinceNightBeginning: Time which the groundhogs in this set starts spawning
- Groundhog Type
- Number Groundhogs
- Spawn Interval (Seconds)

Coding Rules & Naming Conventions

Read the Development Process section too, this has important information about asset formats, git branches, versioning, and more!

General Rules

- Use descriptive names (e.g. GetHealth is better than GetFirstStat) though try not to make them too long, a 20 word name isn't much more readable than an undescriptive name.
- Be consistent (e.g. if you have a variable numStrawberries, name a raspberry variable numRaspberries rather than raspberryCount etc)
- Avoid acronyms (e.g. API) and abbreviations (e.g. dmg) unless commonly known
- Acronyms should be considered a single word in the naming case (e.g playerUi instead of playerUI)
- Public functions and inspector variables should have documentation comments
- Avoid spaces in names, this is due to issues in certain software.
- Always use classes, not structs.
- Use JsonUtility for data serialisation and deserialisation.

Git Branches

- Branches should be in lower-kebab-case.
- Branch names should be short though it should be obvious what is on the branch.
- Branches should be named based on the feature (e.g. tank-groundhog) not the person working on the feature (jacks-groundhog is a bad name!!)
- Avoid similar branch names, ui and more-ui are bad names! Menu-ui and shop-ui are better names.

Variables

- Use variables, not C# fields.
- Constants should be in BLOCK_CASE
- Local variables should be in lowerCamelCase
- Public fields and other global variables should be in UpperCamelCase
- Private fields should be prefixed with an m_ and in m_lowerCamelCase

Functions

- Functions should be in UpperCamelCase
- Function parameters should be in lowerCamelCase

- Functions that return a boolean should be prefixed with “is”, “are”, etc (e.g IsEmpty() is better than Empty()) and should be grammatically correct (e.g DoesRequireBullets instead of IsRequireBullets)

Classes (& more)

- Classes and structs should be in UpperCamelCase
- Interfaces should be prefixed with an I and in IUpperCamelCase
- Enums (and enum values) should be in UpperCamelCase

Comments

- Documentation comments should use triple slash (e.g /// This function plays the player death sound effect)

Game Objects

- Game objects should be UpperCamelCase

Files

- Files and folders should be UpperCamelCase
- Use numbers to indicate versions, e.g Groundhog2 should be a more recent version of Groundhog1
- Use letters to differentiate similar assets, e.g TreeA and TreeB should be separate assets.

Asset Prefixes

Add a prefix to all assets based on their type, as it is common to have, for example, a material and model with the same name.
The only exception to this rule is scenes which should be suffixed with Scene and scripts which should not have a prefix or suffix.
This naming convention was adopted midway through the project from the DES315 wiki so not all assets follow this naming convention currently.
Add another entry to the following list if an asset type is missing.

- “Anim” for animations
- “Mat” for materials
- “Tex” for 2d textures (include dimension for non-2d, e.g Tex3D)
- “ScrObj” for scriptable objects
- “Music” for music
- “Sfx” for sound effects
- “Font” for fonts
- “Pre” for prefabs
- “Item” for item scriptable objects

Tools

MinecraftModelGenerator

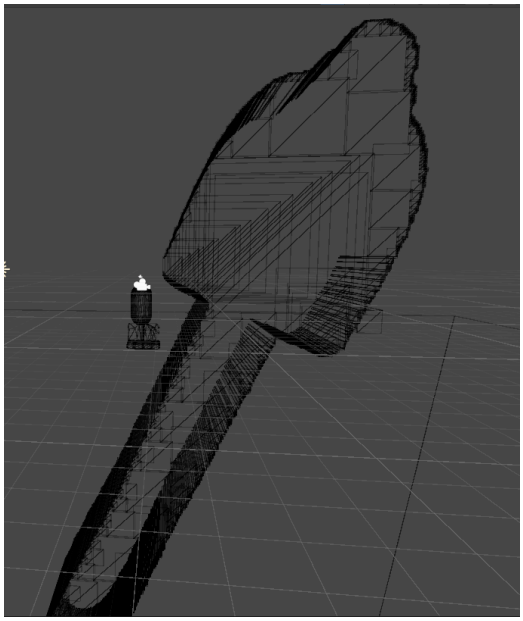
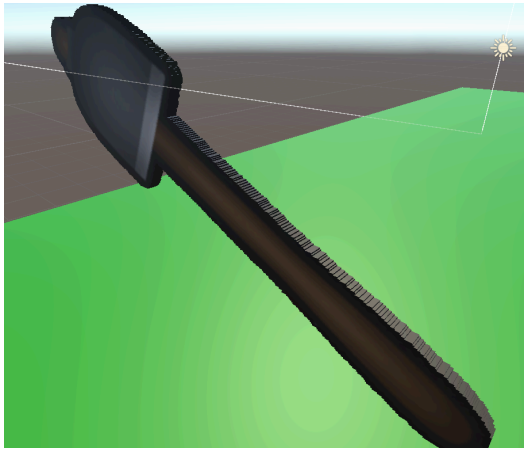
This tool will generate .obj models from images which can be used to avoid artists having to make a model for every item in the game. The models look similar to the models in minecraft when holding an item.


The emerald is an example model.

Due to the size of our icons (approx 500x500), several optimisations have had to be made to bring the triangle count down from ~300k to ~5k.

- Inner face culling (faces obscured by other voxels aren't rendered)
- Greedy meshing on front faces (merges multiple faces into a single quad, greatly reducing triangle count)
- Remove duplicated vertices (this is already done by unity but will decrease project/game size on disk)
- Don't generate back face (invisible when in players hand)

.obj models were chosen because they are much easier to generate than .fbx models.

How does it work?

When MinecraftModelGenerator.exe is executed, it will convert any images in the input directory into models and place it in the output directory.
You can then add this model into unity, and create a material and apply the icon as the base colour. **Note that you may have to set the icon as point sampling (see filter dropdown in import options)**
If you encounter issues with the generated model, ensure the input texture has 4 8-bit channels and is square. Contact Jack for any other issues.

Known Issues

- When building, you must copy the BaseSave folder from Assets into MarmotaMeadow_Data (the folder will exist but it won't have anything inside it)
- Game does not work on Switch Pro controller (unknown if this is an issue with a specific tester or with all Switch Pro controllers)
- Keyboard and mouse input is ignored when a controller is plugged in (workaround: use the controller or unplug it from your PC)
- Some UI renders slightly wrong on resolutions other than 1080p (there won't be any major issues on 16:9 resolutions)
- First tool upgrade may not have any effect in some cases