# Development Documentation

*Jeskiel Valley, developed by Raphael Gobi*

## Development Process

As a passionate developer that loves games like Stardew Valley, I used that inspiration to create my own valley, *Jeskiel Valley*.

With 48 hours as the time limit, it has been a challenge to deliver what I expected to this project. A lot of work had to be rethought to properly fit a functional and casual template game.

I started prototyping it by creating Managers to handle the player input data and added a few custom ScriptableObjects to save data for Player Stats, Player Inputs and other utilities that can be easily changed through the object.

After some hours, I had the basic movement and some animations working. I struggled for some time to think of the best way to implement a Paperdoll animation system, but after some tests and coding, I've implemented a supplementary animation system to properly animate all sprites according to the player's character's current animation.

For the animation system to work, I've added an Animator to the character's body; which should be the Main Animator for the character. It only has 3 animation states: Idle, Walk and Run. All of them are BlendTrees that use animation parameters to check the correct sprite for the animation. Every time the Main Animator loads a new sprite during an animation, it notifies all equipped items to get the next frame of the animation. For that, I've mapped all animations into indexes and made a ScriptableObject to sync them all automatically, saving a lot of time that I may have used for animating the equipment.

Things basically ran smooth after some hours of work into that system. I started implementing the UI system and prepared some UI art in Aseprite, which made me feel good and happy about the result.

After preparing the Inventory system, I started preparing the environment art from some free asset packages in *itch.io*. By using TilePalette, I could easily prepare a nice environment for the character to live in. After getting satisfied with the result, I finally started implementing the Crop system, enabling the player to harvest potatoes and tomatoes. They each have their own growing stages and are planted randomly when the player interacts with a CropPoint (crops can only be planted when the CropPoint is empty). After planting, the crops will keep track of their current progress for the next stage based on the Ticks from CropManager. The tick interval can also be easily changed in CropManager to make the game faster/slower.

Much was done at this point and some good hours of hard work were put into effort during this whole process. Then I finally moved to implementing the Shopkeeper. Because of the modularity of the system I prepared, the process during the Shopkeeper implementation was smooth; in a few hours, I had the system working.

After implementing all essential gameplay, I focused on optimizing and improving the code, until it was bug-free. When all gameplay was done, I could finally finish the remaining parts, which were mostly the Main Menu at this point.

# Game System

I will be listing below the main scripts for the game functionality, for further reference:

- **Managers (Singletons):**
  - **GameManager:** Keeps track of the player's current progression, updating the SaveData when a player modification happens, such as *Currency Changed*, *Item Equipped*, etc. Also provides easy access to other classes.
  - **InputManager:** Reads and notifies all relevant inputs from player to other subscribed classes.
  - **InventoryManager:** Keeps track of the current obtained items and provides easy access to items from any other class. Also responsible for checking Inventory input.
  - **CropManager:** Responsible for notifying all valid crops for each new tick.
  - **UIManager:** Manages all UI screens directly.
- **Configuration (ScriptableObjects):**
  - **DefaultAnimationKeyframes:** Keeps the data from the animation mapping for the equipment animation subsystem.
  - **PlayerAnimationConfig:** Track the data from the main animator to properly update the other animation subsystems.
  - **PlayerStatsConfig:** Simple data for player's stats properties, such as *moveSpeed, magnetRange,* and other relevant data for gameplay.
  - **UserInputConfig:** Binds all PlayerActions to a main and alternate key, making it easy to replace the input mapping.
- **Player:**
  - **PlayerController:** Controls all gameplay input for player
  - **PlayerAnimation:** Responsible for updating the Main Animator and its subsystems and handling all visual data from the player.
  - **PlayerCollision:** Handles all range-type based interactions, such as collisions and gameplay interactions.

# Conclusion

I feel extremely happy with what I've developed so far and it was really a game-jam feeling. I'm happy to have this opportunity, I learned a lot and I had fun while I was working on it. The challenging feeling made me worried about not finishing it until the deadline, but I'm glad I challenged myself for this project. I'm always up for learning new things, and I can certainly say that I learned a lot from this one.

I hope this document was clear and easy to read. Feel free to reach me anytime on my email (gobiraphael@gmail.com) if something isn't clear for you.

Thank you for this opportunity, I hope you enjoy *Jeskiel Valley*!

Raphael Gobi