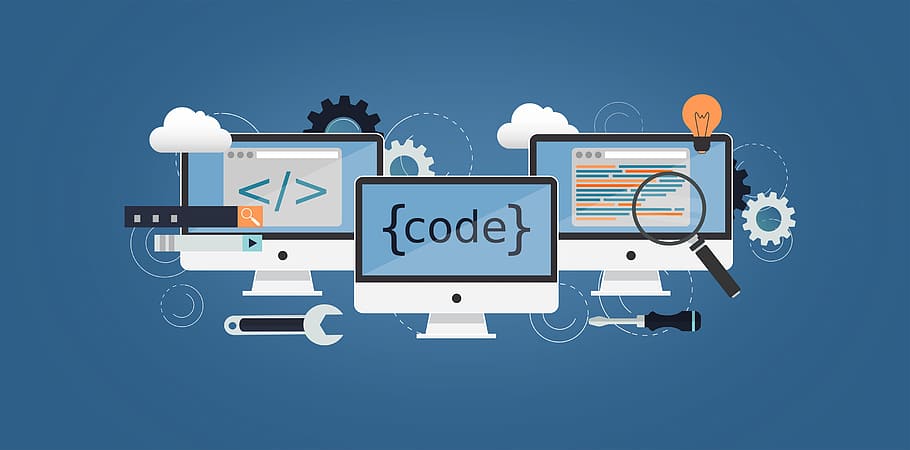
# 

# HSC Software Engineering

# Systems Report



#### Name: Hayden Mercier

#### Class: 12SWE1

Contents

[1. Identifying and defining 2](#_Toc135061669)

[1.1. Define and analyse problem requirements 2](#_Toc135061670)

[1.2. Tools to develop ideas and generate solutions 3](#_Toc135061671)

[2. Research and planning 4](#_Toc135061672)

[2.1. Project management 4](#_Toc135061673)

[2.2. Quality assurance 5](#_Toc135061674)

[2.3. Systems modelling 6](#_Toc135061675)

[3. Producing and implementing 12](#_Toc135061676)

[4. Testing and evaluating 13](#_Toc135061677)

[4.1. Evaluation of code 13](#_Toc135061678)

[4.2. Evaluation of solution 14](#_Toc135061679)

## 1. Identifying and defining

### 1.1. Define and analyse problem requirements

**Problem context**  
**Analyse** the problem by **describing** each of its individual components and **explaining** how each of these components contribute to the problem needing resolution.

As part of the Software Engineering Major Project, I need to develop a piece of software that solves the problem identified by my clients.

My client, a Y10 student, would like to be able to enjoy playing games during Y11 and Y12, which is a busy time for many students. However, the current games they play, such as Hypixel Skyblock and Gregtech, both based around gathering significant amounts of resources, sink large amounts of time and can lead to an extreme loss of productivity. They want to be able to play similar games, competing with their friends, but without the massive time loss that occurs due to the focus required. Even in the most basic skill in Hypixel Skyblock, farming, without Macros, which are not allowed on their servers, requires one hand on the mouse to move the character and farm crops in a custom farm if keys are rebound, which can distract students.

To solve the problem, I have determined that a semi-idle farming game that has a gameplay loop where a significant chunk of time can be spent away from the game, studying or exercising, is necessary, and that continued communication between myself, and the client will ensure that the optimal solution is developed.

As part of this major project, I will also need to complete the following to ensure my solution is feasible, functional, and meets the assessment requirements:

* The game is largely glitch-free due to significant testing.
* The game can write to a file to save the game. This save file should also be able to be loaded by the game.
  + This save file should be able to resist tampering in order to ensure that my clients can fairly compete against each other.
* The game has an intuitive and aesthetically pleasing GUI, and settings can be changed during the game to improve the user experience.
* The project uses an object-oriented programming language, as specified in the assessment requirements.
* All game assets and dependencies are sourced legally and ethically, using original content, creative commons assets, or licensed material
* Fulfil the requirements of my clients, which include:
  + The game is an idle game that enables users to have fun and compete against each other while not requiring huge amounts of gameplay time.
  + The game has a core gameplay loop that is enough time for one study session to occur, which can range from 30-60 minutes.
  + The game looks aesthetically pleasing, and the assets are detailed and have been approved by the clients.
  + The game comes with instructions or a tutorial on how to play the game, to ensure that all users understand the game sufficiently.
  + The game is largely scalable to ensure that future changes can be made to the game by me or by other developers, such as my clients or my friends, if they want to continue with the project.

**Needs and opportunities  
Describe** the needs of the new system to be built based on the problem context and using the table given below.

A successful project meets all needs and a maximum subset of wants.

|  |  |
| --- | --- |
| Need | Description |
| 1. Intuitive GUI that provides relevant information to the user. | The player interface should be both visually appealing and easy to understand at a glance. This includes clear indicators for time of day, tool selection and inventory contents. A messy or confusing GUI would lead to unnecessary frustration, so it must present information cleanly, use recognisable icons and avoid visual clutter to allow players to quickly understand their current status in the game without having to navigate through multiple menus, which can often be the case in MMORPGs such as Hypixel Skyblock. |
| 2. Ability to both save and load the game. | It is absolutely essential that players can exit the game and return to it later without losing progress, to ensure the game can be played over multiple short sessions, ideal for students. The save file system must not only work correctly but also be tamper-resistant. I used a a XOR obfuscation function on the save file to prevent dishonest modifications, especially since my client wants the game to have a fair, competitive aspect among friends. |
| 3. Idle or semi-idle gameplay loop | Since the game is meant to be played in short bursts between study sessions, the gameplay needs to function passively, like many idle games, such as TownShip. For instance, crop growth can continue even when the player is not farming or even looking at the game. This creates a sense of progression and satisfaction without requiring constant attention. This balance encourages healthy gaming habits and supports my client’s goal of reducing time sinks. |
| 4. Low system requirements | Many school laptops are not particularly powerful, so the game needs to run smoothly on low-end hardware. My benchmark was a 2019 school laptop with an i3-7020u CPU, 4GB RAM, and Intel HD 620 graphics. The game should not lag, crash, or cause performance issues on similar systems. It will be repeatedly tested to make sure it can run in the background and alongside light tasks like a browser or music, typical of a Y10 student who is studying. |
| |  | | --- | | 5. **Scalable design for future development.** | | The codebase should be modular and maintainable. This allows future additions or changes to be made by me, if requested, the client, or their friends, without needing to rewrite the whole system. I should structure my scripts using object-oriented principles and modular components to make future development simple and clear. For example, different objects like trees, crops, or animals are stored separate scenes and scripts, making it easy to replace or extend them. Components should be easily able to attach to new objects, and thus enable easy future development. |
| 6. Customisable settings | Players should be able to adjust things like music volume, sound effects, and any other quality-of-life settings, such as time speed or UI visibility. These settings must persist between sessions. This will allow the player to tailor the experience to their liking. |

**Boundaries**

**Analyse** any limitations or boundaries in which this new system will need to operate. Boundaries can include hardware, operating systems, security concerns etc.

Firstly, the game will be designed to run on Windows-based desktop systems. While the Godot engine supports cross-platform deployment, this project will be exported and tested primarily for Windows. As such, attempting to play the game on MacOS or Linux will result in compatibility issues unless the game is manually re-exported to those platforms, which falls outside my current scope due to the extensive process that I will need to undertake, as I do not have access to either operating system. If my client wishes to export to MacOS for one of their friends, they will be provided the GitHub repository and instructions on how to accomplish this within the Godot editor.

Secondly, the game is intended to be lightweight, able to run in the background on systems with minimum hardware specifications as many laptops used for schoolwork have, with a baseline of:

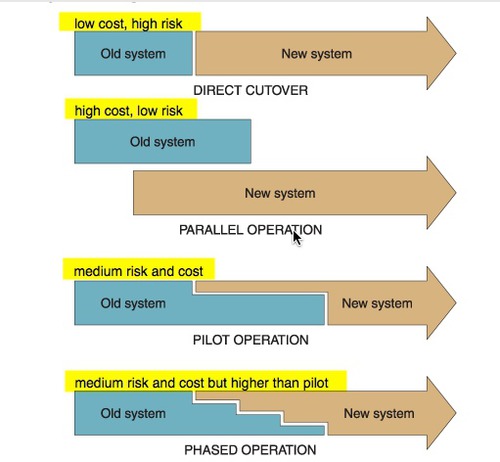
* A CPU with capacity equal to or greater than an i3-7020 processor.
* 4 GB of RAM.
* Integrated graphics equal to an Intel HD 620 graphics card.
* At least 200 MB of available storage.

This can easily be tested by running the game on my sister's laptop, which was bought in 2019 and is what many would consider to be a low-end machine, struggling to have even a handful of Chrome tabs open at once.

The game will be delivered as a ZIP file, a GitHub repository link, or an executable installer. There is no digital rights management or encryption implemented within the game, meaning the software is vulnerable to unauthorised distribution or modification. However, given the small, trusted user base (the client and their close friends), this risk is minimal and considered acceptable for the scope of the project. To keep the competitive nature of the game and prevent tampering with the save file, it will be obfuscated via an XOR obfuscation algorithm to prevent players from giving themselves resources.

The game also does not require internet connectivity and stores user data locally using the AppData directory, which helps avoid online security risks and cloud dependency. However, users must have write access to their local directories for save files to function properly.

### 1.2. Tools to develop ideas and generate solutions

**Implementation method**  
**Explain** the applicability of the implementation method for your project. These are normally: direct, phased, parallel and pilot.

A direct implementation method is most suitable for this task. My software is designed for a specific and limited group of users: the client and their close friends who will receive a completed version of the game once development and testing are concluded in a ZIP file, a GitHub repository link, or an executable file.

This means that the game will be deployed immediately, as there is no previous system and a small number of end users, there is no need for phased or parallel operation.

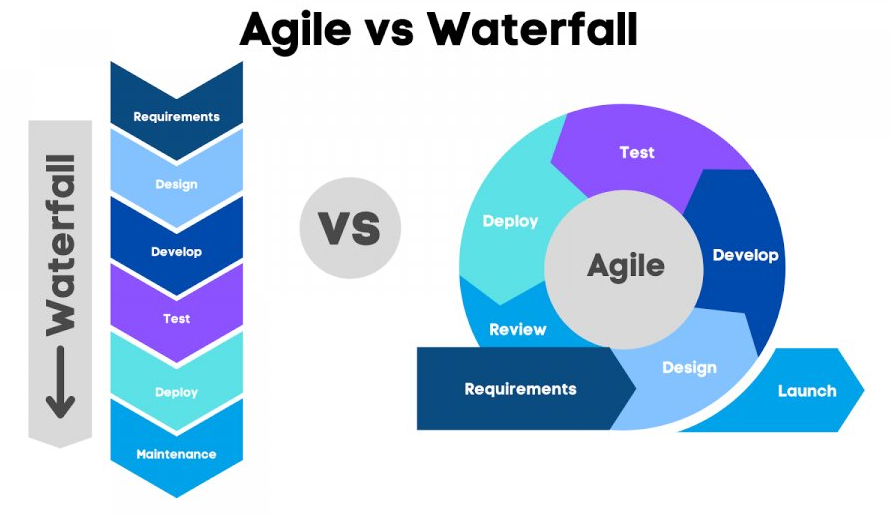
While it could be argued that initially providing the game to my client for personal use before sharing it more widely to their friends resembles a form of pilot operation, this still fits within the direct implementation model due to the immediate and complete deployment of the software. This method allows for rapid user access and is the simplest and most efficient deployment strategy.

The direct implementation method also enables rapid user access to the final product and is the simplest deployment process, useful in this project that will be shared to other players via a download or executable file link. Given that all final testing and debugging will have already been completed before release under white-box and black-box conditions, as part of my major work, the game can confidently be directly implemented into the user environment.

## 2. Research and planning

### 2.1. Project management

**Software development approach  
Explain** the software development approach most applicable for your project. These are normally: Waterfall, Agile and WAgile.

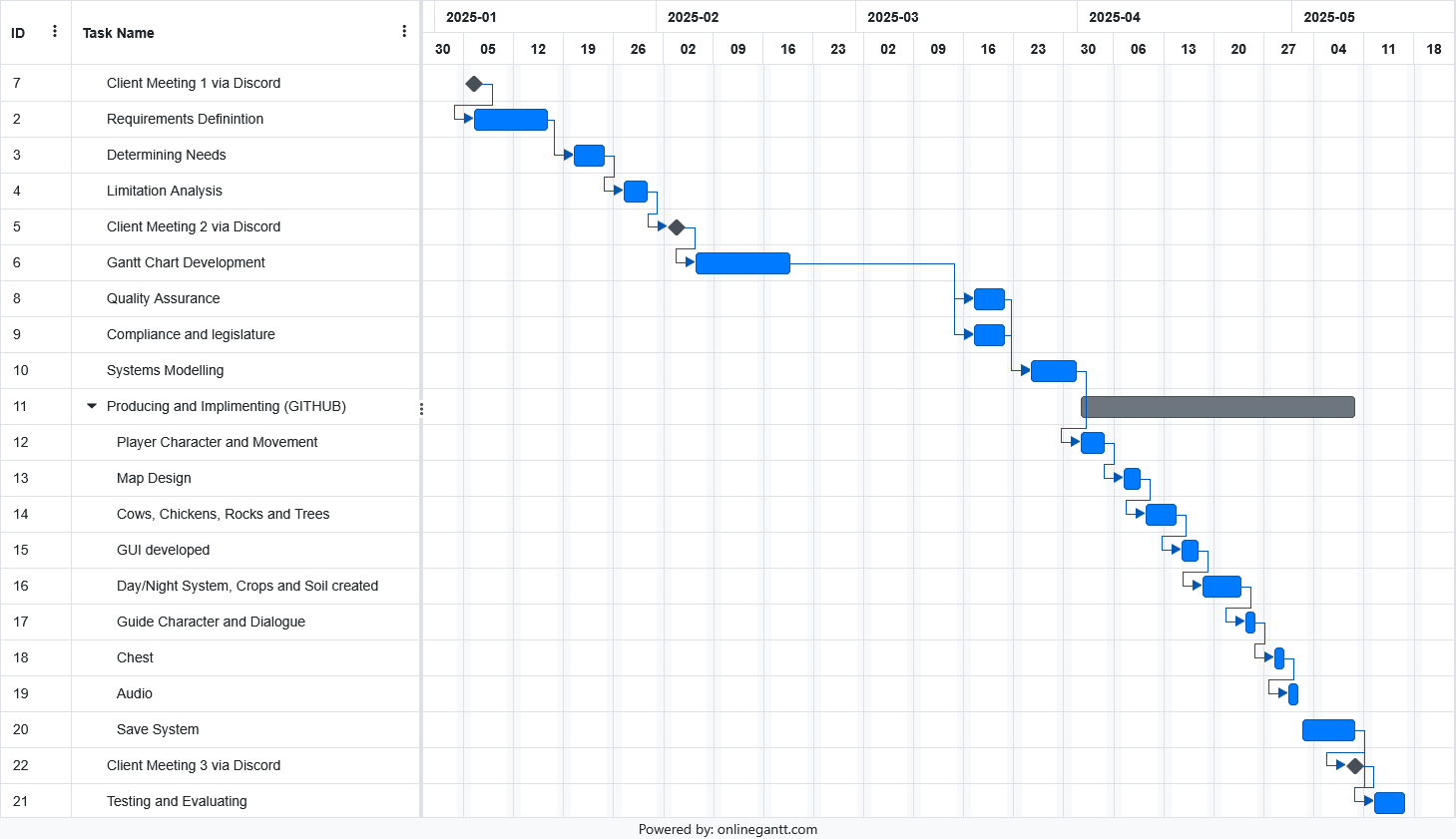
Waterfall is typically only used for large projects that need to be done in a specific order, such as large construction projects. Software solutions are usually never at this stage, and most benefit from the agile development approach. Every time feedback is received in the agile approach, it is received from the client and the public and can be quickly acted upon.

For my project, a farming game which can be enjoyed by a Year 10 student, I required feedback from many sources during the production of the game, to determine how it can be improved, meeting their functional requirements. This made the agile project methodology most suited towards this project, due to the relatively short timeframe, the ease of implementing new features, and the ease of communication I had with both my client and various play testers, which I intended to use to receive large amounts of constructive feedback.

Additionally, my client may change their functional requirements in the middle of the project, and I would need to be able to quickly adapt, making the waterfall methodology unsuitable, as well as, if I discovered significant bugs or discovered that planned features were not accomplishable with Godot or within my short timeframe, I would also need to review the project requirements, which is best suited to the agile project methodology.

**Scheduling and task allocation  
Develop** a Gantt Chart that details the tasks required to be completed and timeline that does not exceed the project due date. In addition, **identify** any collaborative tools used. For example Repl.it, GitHub and so on.

The use of [Gantt Project](https://www.ganttproject.biz/) is recommended.



### 2.2. Quality assurance

**Quality criteria  
Explain** quality criteria based upon the needs from Section 1.1. These quality criteria should contain qualities, characteristics or components that need to be included or visible – based on Section 1.1. – by the end of the current project.

|  |  |
| --- | --- |
| Quality criteria | Explanation |
| Successful operation on low end systems | Detailed quality planning is required to specify the minimum specifications and minimum performance benchmark to meet. An audit of current typical minimum specifications for machines in 2025 will be conducted and an old laptop will be used as a baseline. An analysis of minimum game performance benchmarks is required to accurately measure the quality of the solution. Extensive collaboration and communication with your client will determine approval of the game performing at the minimum performance benchmark on a minimum specification machine. |
| Ability to both save and load the game | Detailed quality planning was implemented during the design of the game’s data save system. This system ensures that key data such as inventory contents, tool selections, player position, and object states are retained between sessions. Only necessary values are stored to reduce save file size and improve load speed. An audit of player behaviour and object interactions was conducted to determine which values required saving and which could be instantiated through scene structure. Ongoing testing and debugging ensured that save files were written and loaded correctly across repeated sessions. XOR obfuscation was applied to prevent tampering and ensure fair competition. Client feedback was used to confirm that saving and loading behaved in line with their expectations. |
| Idle or semi-idle game loop | The design of the gameplay loop focused on ensuring that meaningful progression could occur without active player input. Detailed quality planning was required to identify which game systems could continue in the background, such as crop growth and time progression, using signal-based architecture and timer-based logic. Risk assessment determined that frequent updates in \_process() would negatively affect system performance, so optimisation should be applied to offload idle behaviour to efficient systems. Testing must be be utilised to ensure that players can step away from the game for extended periods and return to find their world progressing as intended, aligning with the client’s goal of enabling short and passive study breaks. |
| Scalable design for future content | Detailed planning will be required to ensure the game can be extended or modified after delivery. Systems will need to be modular, object-oriented, and loosely coupled to support future content such as new crops, animals, currency, or tools. Reusable components and scene inheritance will be implemented to avoid duplication of logic. A risk assessment will be used to identify dependencies that could prevent scalability. Collaboration with the client and version control practices will ensure changes can be made safely and that the client is familiar with the system, and testing will be conducted to validate that all added features integrate without introducing bugs and all systems that enable integration are bug free. |
| Settings should be customisable | Planning will be required to determine which settings will enhance user control and accessibility, such as audio volume, overlays, and game speed. A dedicated settings management system will be developed using exported variables and saveable state dictionaries. Risk assessment will identify any potential issues with settings not persisting between sessions, and mitigation strategies such as automatic saving and state reapplication on load will be implemented. Collaboration with the client will confirm which options should be customisable, and testing will be required to ensure all preferences are applied consistently during gameplay. |
|  | Detailed quality planning will be required to ensure that the GUI is clear and functional for all end users. The interface should display key gameplay information such as current tool selection, inventory contents, and time of day, using recognisable icons, minimal text, and consistent colour coding and themes. A risk assessment will be conducted to identify elements that may contribute to visual clutter or hinder user understanding, such as buttons without icons or unnecessary text. Interface elements must be positioned strategically on screen to minimise obstruction of gameplay while remaining accessible, such as interaction buttons with NPCs. Collaboration with the client will confirm that the layout is appropriate and understandable, and testing will be required to evaluate the ease of use of the GUI and ensure that the implemented solution aligns with the client’s preference for minimal distraction during gameplay. |

**Compliance and legislative requirements  
Explain** compliance and legislative requirements your project needs to meet and how they plan to mitigate them where possible. For example, projects that deal with sensitive personal data being publicly available may fall under the Australian [NSW Privacy and Personal Information Act (1998)](https://legislation.nsw.gov.au/view/whole/html/inforce/current/act-1998-133#statusinformation) and/or [Federal Privacy Act (1988)](https://www.legislation.gov.au/Series/C2004A03712). Alternatively, international standards on information security management such as [ISO/IEC 27001](https://www.iso.org/standard/27001) may also be applicable.

|  |  |
| --- | --- |
| Compliance or legislative issue | Methods for mitigation |
| Age Appropriateness | The game contains no violence, gambling, swearing, or suggestive content. All visual and audio assets are family-friendly and appropriate for school-age users. This ensures the game would receive a G rating under the Australian Classification Board standards. By avoiding gambling mechanics entirely, such as Loot boxes, the game is compliant with expectations for games playable by minors worldwide |
| Personal information | This single-player game collects no personal data from users as there is no login functionality. All gameplay data is stored locally using the user:// directory and does not include names, passwords (hashed or plain-text), email addresses, or other identifying information. As such, the game does not fall under the scope of the NSW Privacy and Personal Information Protection Act (1998) or the Federal Privacy Act (1988), and no data security concerns apply. |
| Copyright and Creative Commons | All assets used in the game, the sprites, sound effects, and music, are legally sourced from platforms like Itch.io, under appropriate Creative Commons licenses (e.g. CC0, which allows complete free-use, or CC-BY, which allows use as long as credit is provided). An asset log is maintained to ensure full compliance, with attribution provided where required. No unlicensed or proprietary assets were used, only one free-use sprite pack, and a set of sound assets and a music pack. |
| Security Standards | The game has no network connectivity, eliminating the risk of data breaches, online attacks, and any exposure to malware through in-game features or chat features, major problems in Hypixel Skyblock where phishing attacks are common. To maintain fair competition, the save file is protected using basic XOR obfuscation, preventing manual edits that would give players an unfair advantage. Although this is not a full encryption standard, it is appropriate for the limited, private user base. |
| Software dependencies | The project uses the open-source Godot Engine (version 4.3), licensed under MIT . No third-party proprietary software is bundled with the game, only an open-source Dialogue Manager that is included within Godot, which also ensures this game follows appropriate legal compliance. All dependencies are included within the exported build or available via GitHub for transparency and accessibility. |
| Consumer rights and guarantees | While the game is not sold commercially and is provided at no cost to the client, it still needs to meet ethical standards by functioning as described, containing a tutorial, and being fully playable. The game will undergo thorough testing to ensure it works as intended, and all known bugs will be resolved before its final release. It is provided free of charge with full access to source files, so my client and his friends are not misled about what they receive. Clients should not risk their health by playing this game, and thus there will be the ability to turn off the flashing lights that are generated as a result of the day-night cycle in cheetah mode. |

### 2.3. Systems modelling

**Develop** the given tables and diagrams. You should consult the [Software Engineering Course Specifications](https://library.curriculum.nsw.edu.au/341419dc-8ec2-0289-7225-6db7f2d751ef/94e1eb0a-0df7-4dbe-9b72-5d5e0d17143a/software-engineering-11-12-higher-school-certificate-course-specifications.PDF) guide should you require further detail, exemplars or information. Each subsection below should be completed with Section 1.1. in mind.

**Data dictionaries and data types**Take the needs identified in Section 1.1. of this Systems Report. For each need, **identify** the variables required, data types, format for display, and so on.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Variable | Data type | Format for display | Size in bytes | Size for display | Description | Example | Validation |
| player\_position | Vector2 | (x, y) | 8 | 7 | Stores player's global position on the map | (128, 64) | The vector lies within the bounds of the island |
| is\_chopped | Bool | True/False | 1 bit | 5 | Whether a tree has been chopped down | True | Boolean |
| is\_mined | Bool | True/False | 1 bit | 5 | Whether a rock has been mined | False | Boolean |
| inventory\_wood | Int | NNN | 2 | 3 | Tracks the number of wood in inventory | 120 | 0–999 |
| inventory\_stone | Int | NNN | 2 | 3 | Tracks the amount of stone in the inventory | 35 | 0–999 |
| current\_tool | String | Tool name | 12 | 12 | States the tool that is currently equipped | WateringCan | The value must match the tool ID, can be selected by indexing a list. |
| toolbar\_enabled | Boolean | True/False | 1 bit | 5 | Whether the toolbar is enabled | True | Boolean |
| settings\_music\_volume | Float | NN.NN | 4 | 5 | Music volume in decibels | -12.50 | -80.0 to 0.0 |
| settings\_sfx\_volume | Float | NN.NN | 4 | 5 | Sound effect volume | -10.00 | -80.0 to 0.0 |
| settings\_day\_night\_enabled | Boolean | True/False | 1 bit | 5 | Stores whether the day/night cycle and overlay is enabled | True | Boolean |
| crop\_type | String | Name | 10 | 10 | The type of crop planted | Corn | Must match the crop ID |
| crop\_growth\_state | Int | NN | 1 | 2 | Current growth stage of crop | 2 | 0–4 |
| crop\_watered | Boolean | True/False | 1 bit | 5 | Whether the crop is watered | True | Boolean |
| crop\_start\_day | Int | NNN | 2 | 3 | The day the crop started growing | 12 | >= 0 |
| crop\_current\_day | Int | NNN | 2 | 3 | The current in-game day | 15 | >= crop\_start\_day |
| tile\_layer\_used\_cells | Array[Vector2i] | [(x1,y1), …] | 60 | 60 | Used cells in tilemap layer, which can be saved. | [(1,1), (1,2)] | Array of coords |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| chest\_contents | Dictionary | Item: Count | 20 | 20 | Items taken by the chest after the player deposits them in their inventory | {Corn:10} | Valid inventory items less than 999 |
| npc\_guide\_dialogue\_index | Int | NN | 1 | 2 | Current dialogue index for guide NPC | 1 | 0–5 |
| npc\_settings\_dialogue\_index | Int | NN | 1 | 2 | Current dialogue index for settings NPC | 0 | 0–3 |
| day\_cycle\_time | Datetime | HH:MM DD | 16–20 | 16 | Time in game day in minutes | 12:00 3 | Is a valid time and the day is >= 1 |
| egg\_drop\_count | Int | NN | 1 | 2 | Number of eggs dropped by chickens | 2 | >= 0 |
| milk\_drop\_count | Int | NN | 1 | 2 | Number of milk items from cows | 3 | >= 0 |
| chest\_spawned\_rewards | Array | [item1, item2] | 20 | 20 | Items spawned around chest | [Milk, Egg] | Array of strings |
| crop\_tile\_position | Vector2i | (x,y) | 8 | 15 | Tile position of crop on map | (10,12) | Vector2i > 0 |
| chicken\_nav\_bounds | Rect2 | x,y,w,h | 12 | 12 | Navigation bounds for chicken | (1,1,5,5) | Positive bounds |
| animal\_pen\_region\_id | Int | NN | 1 | 2 | Navigation region index | 2 | >= 0 |

**Data flow diagrams**

**Develop** data flow diagrams (DFDs) at Level 0 and Level 1. These diagrams should explicitly include the variables from the data dictionaries previously identified as well as the needs identified in Section 1.1.

*Level 0*

A diagram of a software system

AI-generated content may be incorrect.

*Level 1*

A diagram of a software process

AI-generated content may be incorrect.

**Structure charts  
Develop** a structure chart demonstrating how the modules of the final solution are interconnected. A maximum 20 modules are recommended.

A notebook with writing on it

AI-generated content may be incorrect.

**Class diagrams  
Develop** class diagrams demonstrating how each class is related to the other.

A diagram on a white board

AI-generated content may be incorrect.A diagram on a piece of paper

AI-generated content may be incorrect.

**Storyboards  
Develop** storyboards, visually representing the software solution you will build.A notebook with a diagram

AI-generated content may be incorrect.

**Decision trees  
Develop** a decision tree to visually outline the logic flow of at least one process and chain of decisions or selections the final solution will need.

A notebook with lines and numbers

AI-generated content may be incorrect.

**Algorithm design  
Develop** algorithms using methods such as pseudocode or flowcharts to solve the problem and meet the needs from Section 1.1. These algorithms should explicitly include the variables from the data dictionaries created in the previous section.

|  |  |
| --- | --- |
| BEGIN FEED\_ANIMALS  FOR EACH item IN inventory\_dictionary  WHILE inventory\_dictionary[item] > 0  Create(dropped\_item(item))  SET dropped\_item.position = chest\_offset\_position  SET dropped\_item.pickup\_enabled = False  Spawn(dropped\_item, dropped\_item.position)  SET inventory\_dictionary[item] = inventory\_dictionary[item] – 1  PlayAnimation(dropped\_item, random\_time)  ENDWHILE  NEXT item  FOR i = 1 TO random(1, 2)  Create reward(RANDOM\_CHOICE([Milk, Egg]))  Spawn(reward, reward\_position)  NEXT i  END FEED\_ANIMALS | BEGIN INITIALISE\_TREES  FOR tree IN tree\_list  IF tree.name IN save\_data\_dictionary THEN  IF save\_data\_dictionary[tree.name] = is\_chopped THEN  HIDE tree  DISABLE tree.hitbox  SET tree.is\_chopped = true  ENDIF  ENDIF  NEXT tree  END INITIALISE\_TREES |

A screenshot of a computer

AI-generated content may be incorrect.

## 3. Producing and implementing

**Solution to software problem  
Include** screen shots of your final developed solution here. Each screenshot should include a caption that **explains** how it links to the:

* Needs identified in Section 1.1.
* Components of Section 2.3. such as the storyboards, data dictionaries and so on.

*A screenshot of a computer

AI-generated content may be incorrect.Player*

A screen shot of a computer program

AI-generated content may be incorrect.The player character was designed to be able to move around the map and perform essential actions such as chopping down trees and rocks, tilling the soil, and planting and watering crops. They were designed via a state machine, which enabled them to swap between designated “states”, such as walking, idle, chopping, and tilling. This ensured consistency in animations and made it extremely easy to program the player’s features. The idle state was swapped to after the player finished any action, or if they were not walking. The walk state could only be swapped to from the idle state, and the chopping and other action states could only be swapped to from the idle state, which prevented players from moving while they were harvesting, and allows for the future addition of states to be easily accomplished.

The animations for the sprite were 4-way, which influenced my decision to make the character only able to walk up, down, left and right.

In order to interact with the world around it, the player had both a collision hitbox, to stop it from walking through walls, trees, and on water, as well as a hitbox for the tools, which changed in size depending on the type of tool used. For example, the hitbox of the axe was much smaller than the hitbox of the watering can, in order to encourage planting crops close together, grouped up.

The player’s position could also be saved using the SaveDataComponent node that writes to the player data resource.

The player is able to interact with objects that have the interactable component by walking into their hitbox, which can be found on dropped items, NPCs, and chests.

The player movement and state machine system was developed to address the requirement for low-interaction gameplay. It ensures players can walk, act, and remain idle without unintended animations or motion. These behaviours align with the user expectations described during planning*A screenshot of a computer

AI-generated content may be incorrect.*.

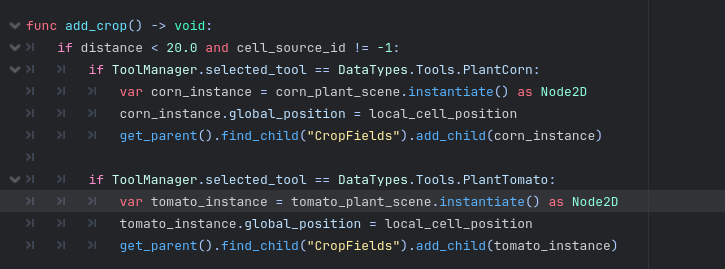
*Trees and Rocks*

Trees and rocks are very similar, with the only difference being the name of the variable that stores if they have been destroyed, dropping different items, referencing different save resources to better organise the save file, their sprites and the shape of their hitbox. They have the same gameplay function, posing as obstacles that the player must cut down with the axe in order to build large farms, but also pose a decorative function.

A screen shot of a computer code

AI-generated content may be incorrect.There are two types of trees, large and small trees. Large trees have more health but drop more logs. Rocks take longer to break than large trees and drop one stone each. When attacked, they shake slightly using a shader that vibrates the sprite from the bottom up.

Trees and logs both have area blockers that stop the player from planting on top of them, causing visual issues, which are removed when they are broken. This mimics the garden plot cleaning mechanic of Hypixel Skyblock, and adds some required gameplay in order to create large, uninterrupted farms.

*Crops*

The two types of crops in the game are corn and tomato, which are farmed by the player.

The farming process is rather simple. When placed on farmland and watered, crops will start to grow. They take ~20 minutes to grow fully, and have several growth stages as per their growth cycle script, and when they do, they are automatically harvested and drop as normal objects that can be picked up. These can then be fed to cows and chickens, with a 1:1 exchange.

*A screen shot of a computer program

AI-generated content may be incorrect.Soil*

Grass can be turned into the tilled dirt terrain via using the hoe on nearby grass to the player. Grass underneath rocks, trees, and houses cannot be tilled. Tilling the ground will remove all other nature features, such as flowers, from other tilemap layers, and place one tilled soil terrain block there.

Crops can only be placed on tilled dirt and will not be able to be placed on normal grass, as there is a check to see if the tile the user is attempting to instantiate a crop on exists on the tilled dirt tilemap layer.

The tree, rock, and crop systems were implemented as modular scenes using object-oriented practices, relying on reusable SaveDataComponent and DamageComponent nodes to allow scalability, satisfying the need for future development.

*A screen shot of a computer program

AI-generated content may be incorrect.Chickens and cows*

Chickens and cows exist to provide some much-needed life to the game. Narrative-wise, the items that go in chests are fed to the animals, and thus milk and eggs are created, however, they serve no functional purpose aside from aesthetics. This is a reference to the underdeveloped skill of Husbandry in Hypixel Skyblock. They are contained within a zone using Godot’s inbuilt navigation agents and navigation regions.

*A screenshot of a video game

AI-generated content may be incorrect.Chests*

A screen shot of a computer program

AI-generated content may be incorrect.The chests act as the medium between the player and the animals. When the chest is opened and items put in, all items from the player’s inventory are deposited onto the scene via decreasing the inventory dictionary values until they are zero, and for each item, instantiating the corresponding dropped item without the ability to be picked up. A short cutscene plays, which utilises a tween to move the items on top of the chest, where they are deleted, and either eggs or milk are spawned randomly around the chest. These can be collected by the player, which concludes 1 gameplay loop. The proposed purpose of the game by my client is to see how long it takes the user to reach 999 eggs and milk.

The above code block demonstrates the main core of the chest logic.

*Dropped items*

When items are instanced into the scene as dropped items, they have a sprite and an interactable component with a hitbox. When the player walks over the hitbox, the item is queue\_free()ed, and their inventory for that item is incremented by however many they walked over, usually just one.

*Inventory*

The inventory system was based off a simple dictionary that was able to keep track of items that the player had picked up, up to 999 items. When collectable components collided with the player, the name of the associated item was gotten and its slot in the *A screenshot of a video game

AI-generated content may be incorrect.*inventory was increased by 1. The inventory was part of the GUI, a sidebar that had a slot for wood, stone, corn, tomato, milk, and eggs.

This inventory GUI implementation referenced the interface requirements for the game. The GUI was developed with minimal text, large clear icons, and an accessible layout, ensuring readability on lower-resolution displays while avoiding icon distortion. These interface elements were defined previously in the storyboards and data dictionary entries.

A screen shot of a computer program

AI-generated content may be incorrect.

*A screen shot of a computer program

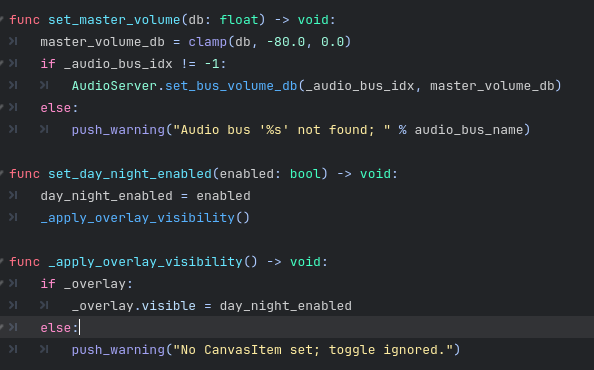
AI-generated content may be incorrect.Guide*

A screenshot of a video game

AI-generated content may be incorrect.The guide provides the player with seeds and tools (enables the toolbar), and provides a small amount of information about the game, giving the island more narrative and adding life into the game. They have dialogue with the player in a stylistic, consistent format.

A screenshot of a video game

AI-generated content may be incorrect.

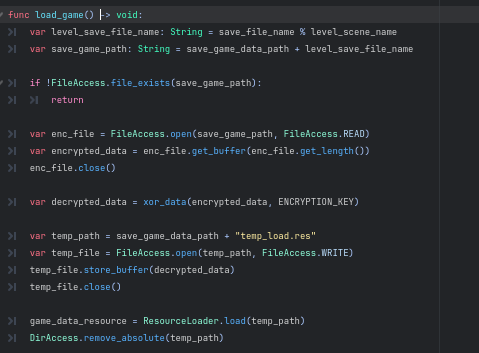
*Settings*

The settings are able to be changed as per the client’s request using a similar character, which is able to change the volume of the system and toggle the day-night cycle to satisfy the preferences of the individual user. The settings are saved between sessions via saving two values that correspond with the Boolean values of the settings, meaning that the player does not have to change them every time they log on.

*Save System*

The save system was the part of the game that took the longest to complete due to the intricacies of the data collection process. It is based around attaching a SaveDataComponent to each scene/node that is going to be saved, which has a corresponding resource, which can range from a tree data resource, specific for large and small trees to a tilemap data resource which gathers the positions of items on specific tilemap layers in order to save them. These are all connected to one save level data component attached to level 1, which is able to save the level to an obfuscated file that makes it harder for players to alter their character and give themselves items. This file is saved to a specific path within the users directory and is later referenced when the level is reloaded in order to load all saved components correctly, sequentially loading all node data resource holders (parent class of tree data resource, tilemap layer data resource, player data resource, etc..) as well as the global script values such as the settings and player position. This brings the game back to its state when it was saved seamlessly.

The save system’s design responds directly to the need for data persistence and tamper resistance. By using a NodeDataResource hierarchy with XOR obfuscation and modular components, the system both stores minimal and necessary data, and prevents manual save editing. The algorithm for this was documented in pseudocode and illustrated in the save flowchart and class diagram.

****

**A screen shot of a computer

AI-generated content may be incorrect.**

The load data function in the save level data component, which gets information from the file after it is de-obfuscated, and loads it into all necessary parts of the program.

A screen shot of a computer program

AI-generated content may be incorrect.

The save node data function in the same script. This saves all information attached to save data components and the global scripts.

A screen shot of a computer

AI-generated content may be incorrect.

The save game function. This saves all the data to one encrypted file, using a temporary file which is then obfuscated.

A computer screen shot of a program code

AI-generated content may be incorrect.

The lightweight and simple encryption algorithm used to encrypt the saved data.

**Version control  
Describe** what version control system or protocol was implemented in your solution. If none, make a proposal.

I used the version control software Git and GitHub in order to ensure that I was never at risk of losing progress due to the Godot Engine accidentally crashing or making reverse progress when attempting to fix major issues.

I used Git branching to protect the project when I made significantly impactful changes, such as when I rewrote large sections of the save system to incorporate the ability to save destroyable objects, such as rocks and trees, not just crops and tilemap layers like the tilled dirt layer. If a mistake was made here, and several were, especially in this save system and the development of settings, it would be extremely difficult to undo the changes within the Godot editor without access to version control systems. When mistakes were made, I simply reverted to the previous commit on the main branch and reimplemented my solution without the mistake made, then worked from there. I did have redundancy when I made some branches and commits, as my features were not implemented into a main scene until very late in the development process, meaning that errors and mistakes could be contained.

## 4. Testing and evaluating

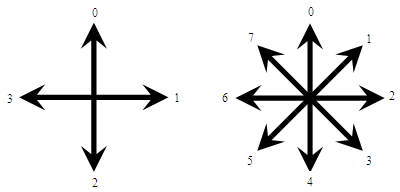
### 4.1. Evaluation of code

**Methodology to test and evaluate code  
Explain** the methodologies used to test and evaluate code in your solution. Methodologies include:

* unit, subsystem and system testing DONE
* Black, white and grey box testing
* Quality assurance.

Unit

Unit testing occurs by testing individual components or functions in isolation to verify they work correctly, in order to quickly catch bugs in specific modules. Unit testing occurred extensively throughout the development of the program. It occurred as I created each aspect of the game, testing how it functioned alone within a test scene.

The player’s operation was extensively tested before any other features were developed, as well as during the testing of other features that required the player to activate. One decision I needed to make was whether the movement of the player should be 8-way, to increase the mobility of the player and be similar to other 2D games, or 4-way, to make it match the sprites I had access to and make it feel smoother. I ended up going with 4-way movement, and thus I needed to ensure that the player could move around successfully. I had a few small issues with this due to the way that Godot handles input, flipping the Cartesian plane so that down was positive in the y-axis, but this was easily fixed. Additionally, I stopped the player from being able to go through walls by using a CollisionShape2D node and positioning it so that it did not cover the entirety of the character, allowing smoother gameplay when around walls and trees, but still preventing the sprite from clipping into walls and other barriers. This solved several visual issues I initially had with my player.

A screenshot of a video game

AI-generated content may be incorrect.I tested the chests by ensuring that the in-game indicator (the E), could appear when the player walked into the range of the chest. I also tested the location of the E on a test scene to visually position it correctly. Additionally, there were times early during testing when the dialogue did not activate properly, which occurred due to incorrectly setting the collision shape to the inner circle, not the outer, interactable circle, which cannot collide due to them being rigid collision shapes. This was done in a separate chest test scene before it was added to the main level.

Subsystem Testing

A screenshot of a computer

AI-generated content may be incorrect.A pixelated image of a planet

AI-generated content may be incorrect.A blue circle with a cross on it

AI-generated content may be incorrect.I similarly tested subsystems to unit testing by testing them as soon as they were implemented inside a test scene to ensure they functioned correctly.

This was particularly useful when testing the pickup and inventory mechanics. I initially had to make sure that the individual environmental scenes such as the trees and rock worked correctly, collided with the player correctly and had appropriate y-sorting (where sprites have correct ordering depending on their position relative to the player), ensure that they dropped at both the correct position and were interactable with the player, able to pick them up. This was sometimes an issue as the collision box for some pickup items was too large, so this was reduced significantly, so it was a consistent size no matter the sprite size. Finally, I had to test the inventory and check if the pickup was recognised, removing the item and adding it to their inventory. I had several issues with this, for instance, the inventory dictionary was case sensitive and I was incorrectly referencing it, so it would return an error. Additionally, the names of several unique items were the same, as I had duplicated their scenes , as they were extremely similar, only differing in their unique identifier and sprite, and forgotten to change the name, so it was registering the wrong items as having been picked up.

A screenshot of a computer

AI-generated content may be incorrect.A screenshot of a computer

AI-generated content may be incorrect.

I extensively tested the save function system and its interaction with the other parts of the game, such as the settings, the player, the inventory and toolbar, and the position of crops, farmland, trees and rocks, to seamlessly reload the game after it was closed and reopened.

A screen shot of a computer program

AI-generated content may be incorrect.A screen shot of a computer code

AI-generated content may be incorrect.I was able to get the save function working for both plants and the tilled soil very easily, however, it was more difficult to save for trees and rocks due to them being destroyed after use. I used a method known as queue\_free(), which destroys a node from the scene tree to save memory and resources during runtime. However, this also removed the information of the trees and rocks, preventing it from being properly saved by their corresponding data resources. This took me a while to fix, and I needed to change how the trees and rocks loaded in and were chopped down, so that queue\_free() would only run if their data was saved inside the save function and the game was reloaded, and that inside the game, they were hidden and their hitbox disabled. I then extensively tested the save function using the settings, UI information, and position of the character, which was significantly easier to debug as this was simply saving Boolean data for the settings and UI information, a Dictionary for the inventory, and a Vector2 for the player position, which could be easily obtained and worked out, as they were not as abstract as saving nodes and node information. There was an issue with not being a default save position, so the player would spawn out of bounds at (0,0), but this was easily remedied.

A black screen with yellow text

AI-generated content may be incorrect.

System

A video game screen with a green background

AI-generated content may be incorrect.System testing was performed extensively by me and a wide range of play testers. These included my family members and class members, allowing for a range of scenarios that I did not account for, allowing me to fix several bugs and features that were not fun or did not make sense, as per this feedback. I also engaged in my own system testing, running through the core gameplay loop and mimicking the actions of the clients in order to detect any issues within the main scene. From this, I found that the pathfinding of animals’ NavigationAgent2Ds within their pens was erratic and buggy, sometimes causing them to wander outside their NavigationRegion2D and get stuck attempting to walk through fences. To fix this, I reduced the region size, as seen in the screenshot, to prevent this possibility from occurring, particularly by moving the box away from the opening of the pen. I also found that flowers and other tiles from the overgrowth and undergrowth layers would grow on top of tilled dirt, which was an unintended feature. This caused the game to look unprofessional and, while only a visual bug, would impact the gameplay experience and immersion of my clients. I fixed this by clearing these Tilemap Layers when dirt was placed down.

Black Box Testing

Black box testing involves getting people who do not know what is inside the program to play the game, such as my family and younger Normanhurst students. They were able to uncover several significant issues with the game. This included the ability to channel Jesus and walk on water. To solve this, I added a basic invisible barrier around the island using collision boxes and then told other groups to try and make it onto the water. They also found some places around the edges where water did not cover the rest of the screen, and instead, a blank background was located. I fixed this by repositioning the island to further away from the edges of the water and adding more water. My client also engaged in remote-black box testing as they discovered a bug by simply asking “can you plant crops inside” when I was showing them the game, which, it turns out had not been sufficiently tested and players could. This was promptly fixed.

Grey Box Testing

Grey box testing involved me getting a class member who had a limited knowledge of the program to go through my game and try to find any flaws. Thanks to the assistance of the Godot documentation, tutorials, and the significant testing I had already engaged in, this search yielded very few issues, only uncovering the ability to plant crops on grass, which was quickly solved with an if statement to check if the ground below was in the TilledDirt Tilemap Layer.

White Box Testing

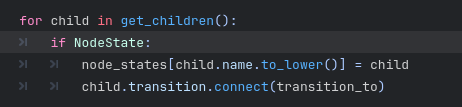
This involved my testing as I attempted to find things wrong with the game, with full knowledge of the internal system. This was performed throughout the development of the game as objects were implemented, before it was given to other people to test and evaluate, and included the aforementioned system, subsystem, and unit testing. After grey box testing was completed, I found that the state of crops (watered/unwatered, as well as their growth state) was not saved correctly, leading all fields to be reverted back to a base state. This was promptly fixed by saving the value for each object, similarly to my process for the trees and rocks.

**Code optimisation  
Explain** the methodologies used to optimise code in your solution so that it runs faster and more efficiently. Methodologies include:

* Dead code elimination
* Code movement
* Strength reduction
* Common sub-expression elimination
* Compile time evaluation – constant folding and constant propagation.

Dead code elimination (the removal of functions or programs that do not affect the program and are not used in the final solution) was completed by my removal of the test scenes that I used to test features during the development of the game, as well as their associated scripts. Additionally, I also removed all debug lines, such as print statements and if statements, to display the values of variables and other information during runtime in the testing phase. Furthermore, during the save functionality of the game, only the necessary values required to be saved, such as is\_chopped or is\_mined, for rocks and trees, are saved; no positions or other node information is saved, as these are instantiated in their position in the editor when the game begins and as such, would be redundant.

A screen shot of a computer

AI-generated content may be incorrect.Code movement (the movement of expensive operations/functions outside of loops that can run very often, such as \_physics\_process(), which is called at a fixed rate, 60 times per second, or \_process(), which is called every frame, in order to improve performance). This was done by minimising the amount of code that was to go within these functions, as well as any for or while loops, in order to decrease the time taken for actions such as feeding cows via the chests. This included removing the type verification for objects within the loop, and instead only running the loop on objects that shared that type by checking their type before hand. Additionally, all variable assignment, such as accidentally setting the position of the central spawn location every time it ran, that occurred in a loop was checked to see if it was absolutely necessary and some was taken outside of the loop/code was rewritten in order to save resources. The only code in \_process() and \_physics\_process() within my program are the day-night cycle logic, the growth state logic of flowers, the player movement, and the chicken/cow movement. This ensures that the game is not using intensive resources when it is not being viewed when my client or their friends are studying.

Strength reduction (the simplification of algorithms in order to improve performance and decrease runtime, such as changing xy to (x2)y/2), was not largely applicable to my project, as most of the math was contained within the day-night cycle script and the chest random-item placement uses constants such as TAU, 2π, and simple mathematical functions such as sin and cos. There are no powers/exponentials used in the program and as such, strength reduction was not completed.

Common sub-expression (the replacement of duplicate expressions that are calculated multiple times within runtime with one single variable that stores the result of the expression which is then reused, to improve the speed and memory efficiency) was completed across multiple scripts, such as in the saving system, where node references or function calls such as get\_node\_or\_null(path) were previously called multiple times within loops or conditionals, and were instead stored in a local variable once and reused. This avoided redundant node tree lookups. Additionally, the day/night cycle time recalculations originally recalculated the total time in minutes from scratch multiple times per frame. Instead, values such as total\_minutes and current\_day\_minutes were cached and reused throughout the recalculate\_time() function. This ensured that divisions and modulo operations, which are relatively expensive in processing time, were not performed more than once. Additionally, in the feeding mechanic for animals, the global\_position of the chest was stored before being used multiple times to position spawned harvest rewards. Without this, the engine would have recalculated the node’s position several times per loop iteration. These changes collectively contributed to a smoother experience during high-frequency operations like spawning or saving.

Constant folding and constant propagation (values and expressions that are constant, which can be inserted by the compiler at compile-time rather than recalculated at runtime) were not considered in my program as GDscript, like Python, is an interpreted language and is not compiled.

### 4.2. Evaluation of solution

**Analysis of feedback  
Analyse** feedback given to you on the new system you have just created. This feedback can be in the form of an interview, survey, focus group, observation or any other applicable method. You should also include overall positive, negative or neutral sentiments towards the new system in their response.

|  |  |
| --- | --- |
| Users and Feedback | Further Actions |
| **Client**  Good visuals, they approved of the sprites and audio used. However, they noted that the sound effects of the chickens and cows were quite jarring while playing the game compared to the sound of the background music.  They enjoyed the gameplay loop, said it was reminiscent of Hypixel Skyblock’s Garden region, but less active, which was its purpose.  The client asked if I could check where the player can till, which led to me adding blocking regions on houses and trees and rocks. | I changed the sound via the sound bus to reduce the sound of the animals in relation to the sound of the music, and I also reduced their frequency by changing two variables.  Blocking regions were added to houses and trees. |
| Y12 Play testers  Y12 play testers enjoyed the theme and the gameplay loop, they enjoyed the ability to speed up the game for active playing. Some of them did question the ability of the player to leave the island, and also expressed annoyance about not being able to cut down apple trees.  Informed me of a visual distraction with one of the buttons, text too close to the edge.  They disliked the flashing lights of the day-night cycle when the game was sped up. | The player was stopped from going onto the water by adding an invisible barrier between the water and the grass.  Fewer apple trees were included in the scene, they were placed further away from other trees to distinguish them.  Button config was changed to increase the margin around the text.  Added the option to turn off the day-night cycle. |
| Other Play testers  Several Y11s, Y10s, and other people such as my family provided feedback on my game.  I was informed that a quiet sound level was redundant due to the ability to change the sound on the user’s device.  I was told that it looked ugly when crop numbers surpassed 999 due to the width of the icons.  Y11 play testers noted that the character speed did not match the animation speed, which was changed during testing to improve convenience. | Removed the quiet mode, now the game only has the on and mute modes.  Added a limit to prevent the crops from surpassing 999.  Player speed was reduced back to a value that may have felt slower but matched the animation speed. |

**Testing methods  
Identify** the method or methods of testing used in this current project. For each use, you need to **explain** how and why it was used.

|  |  |  |
| --- | --- | --- |
| Method | Applicability | Reasoning |
| Functional testing | Applicable | Functional testing was used extensively to confirm that each system (e.g. saving/loading, inventory, crop growth, time cycle) performed as intended through the usage of a test scene. For example, tests were performed to ensure that feeding animals triggered crop drops correctly, that chopped trees didn’t respawn after a save/load cycle, and that UI elements responded to input events. This ensured all individual components met their functional requirements. |
| Acceptance testing | Highly Applicable | Acceptance testing was used to ensure that the product matched my client’s expectations and requirements. I did this at the end of the development process when I had received feedback from many other parties, in order to receive a final judgement and make last-minute fixes. This was done by sending them screen recordings of key features within the game, such as the movement of the character, the farming cycle and an interaction with a chest, as well as the save process after altering the game environment, and receiving feedback. |
| Live data | Somewhat applicable | The use of live data testing was somewhat applicable in the sense that all gameplay testing naturally involved live, in-game data. For example, the day-night cycle values, inventory contents, and object states (e.g. chopped trees, watered crops) were all generated through real interactions during gameplay. This helped validate that the game handled ongoing changes and saved dynamic states correctly during normal operation. |
| Simulated data |  | Simulated data was used to test edge cases that might not occur naturally during regular gameplay. This included manually adjusting the player’s inventory to include unusually high or empty values, injecting fake time states into the day-night cycle to test crop growth, and forcing interaction triggers without player movement. These simulations helped identify logic issues and fix bugs in the save/load and inventory systems. |
| Beta testing | Applicable | Beta testing was conducted with my peers and family members who had no prior experience with the project. They were given early builds of my game and were asked to complete basic tasks such as farming, feeding animals, and breaking objects. Their feedback led to several usability improvements, such as clearer interaction prompts and refining tool selection behaviour, as well as enabling the creation of targeted instructions to ensure the game was intuitive and enjoyable for new users. |
| Volume testing | Somewhat applicable | To ensure the save file and system could handle large amounts of data, testing was done with many trees, rocks, and crops instantiated, at one point almost the entire island was covered in fields and corn. This verified that the system could still run smoothly and save correctly without crashing or slowing down, especially when used in longer play sessions with many changes to the world state. The only change to the gameplay was a longer flash after the start button was pressed. |

**Test data tables  
Identify** variables which were used for either path and/or boundary testing. **Develop** these test data tables based on your algorithms versus their real code. Then **state** the reason for including said variables.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Variable | Maximum | Minimum | Default Value | Expected Output | Actual Output | Reason for Inclusion |
| player\_position | (885, 126) | (0, 0) | (256, 128) | Player is correctly placed on map and cannot leave bounds | Player constrained by collision. No spawn bugs. | To verify the system enforces map boundaries and the valid spawn point after load |
| inventory["corn"] | 999 | 0 | 0 | Corn is removed after placed in the feeding chest. Does not exceed cap on pickup | Corn drops stop at 999. Correctly decremented when fed to animals. | To ensure no overflow occurs in inventory logic or the UI |
| crop\_growth\_state | 4 | 0 | 0 | Growth state increases after each day if watered | Growth proceeds correctly. Harvests at 4. | To confirm semi-idle system correctly tracks state and triggers harvest |
| settings\_music\_volume | 0.0 | -80.0 | -12.0 | Music plays at correct volume. Saves and loads with correct values | Volume saved correctly. Adjusted in-game. | To confirm that float settings are stored and reloaded properly via the settings manager |
| toolbar\_enabled | TRUE | FALSE | FALSE | Toolbar appears/disappears based on dialogue interaction | Setting persists correctly. Reactivated when expected. | Ensures the UI state is preserved and conditionally enabled by dialogue |
| cell\_position | Valid tile | Invalid (-1,-1) | N/A | Tilling only occurs on valid grass tiles within range | Tilling fails on invalid tiles, as expected. | Validates terrain interaction logic and prevents invalid tile editing |

**Analysis of solution against quality success criteria**Copy each quality success criteria from Section 2.2 and paste below. For each quality criteria, **analyse** the components of the solution that met or did not meet each quality criteria. Give reasons why each success criteria were or were not met.

|  |  |  |
| --- | --- | --- |
| Quality criteria | Met? | Analysis |
| Game operates on low end systems | Yes | The game was designed from the outset to run smoothly on low-end hardware typical of laptops, with a baseline of an i3-7020U CPU, 4GB RAM, Intel HD 620 integrated graphics. To meet this requirement, only 2D assets were used, particle effects were minimised, and scene instancing was optimised through modular node structures. Intensive systems like pathfinding were restricted to confined fenced animal pens, and \_process() and \_physics\_process() logic was kept minimal, with no unnecessary per-frame calculations. Performance testing was conducted on a 2019 laptop to verify framerate stability under volume test conditions (e.g. entire island covered in corn). The system met performance expectations and did not crash or experience noticeable frame drops, even with high object counts. This was further confirmed through client collaboration. |
| Ability to both save and load the game | Yes | The game features a custom save/load system using modular NodeDataResource scripts and its children that store only essential values such as is\_mined, is\_chopped, inventory dictionaries, player position, and settings values. This reduced memory usage and the save file size. Save files are created as res files, lightweight binary files, obfuscated with an effective and efficient XOR cipher to prevent tampering. Load processes are able to safely restore the game objects using deferred instancing and resource re-application. Extensive functional and system testing was conducted to ensure reliability across different object types, including edge cases like queue-freed trees or rocks, and significant bug fixes were undertaken to solve this issue. Client feedback confirmed that the game saved and resumed correctly, even when closed and reopened, preserving progress fairly and consistently. |
| Idle or semi-idle game loop | Yes | The core gameplay loop supports semi-idle progression through signal-based systems (e.g. time\_tick\_day, on\_game\_time) that handle crop growth, animal rewards, and environmental transitions in the background. This design reduces reliance on constant player input, only requiring players to plant and water, and then pick up harvested crops, and avoids CPU-heavy functions inside \_process(). The player can leave the game running and return to find meaningful changes (e.g. mature or harvested crops waiting for them, whijch aligns with the client’s request for a game that encourages studying without distraction. System testing confirmed that time-dependent systems could progress correctly, and memory usage remained stable during idle sessions. Risk analysis validated that no long-running loops or memory leaks occurred after sufficient testing. |
| Scalable design for future content | Yes | All major game objects were implemented as modular scenes with shared components such as SaveDataComponent, InteractableComponent, and DamageComponent. Inheritance and composition were used to avoid duplication - for instance, all trees and rocks share the same save/load logic and can be extended via child scripts. Version control with Git allowed major refactors (like save system rewrites) to be isolated and tracked for future developers to understand how the program works. Testing confirmed new items could be added without altering core logic, and the client was given access to the GitHub repository with documentation, allowing them or peers to continue development independently. This supports the long-term sustainability of the project. |
| Settings should be customisable | Yes | A dedicated Settings manager (autoloaded script) manages audio, including sfx and music, as well as the day night overlay. These values are edited via interacting with an NPC inside the game and are saved using a persistent dictionary and reapplied automatically during load via set\_settings\_state(). A risk assessment identified potential issues with volume not applying on startup; this was resolved by storing both the volume level and bus index and applying them at \_ready() when the game boots up. Client feedback was used to adjust default values and improve user experience, and functional testing confirmed that these changes felt better for all users, and that changes to the settings persisted across sessions and applied consistently via the save system. |
| Intuitive GUI that provides relevant information to the user | Yes | The game interface was developed to provide clear and accessible information with minimal visual clutter and scrolling through menus. Elements such as tool icons, interaction prompts (e.g. "E" above chests), and inventory displays were laid out using anchored Control nodes for dynamic scaling no matter the screen size. Icons were chosen for recognisability and aesthetics, and text was minimised where unnecessary. The only text in the GUI occurs in the inventory, in the pause/start menu, and the numbers for the inventory. UI testing included checking contrast, responsiveness, and positioning relative to in-game objects, as well as that the theme was applied consistently across all buttons, containers, and GUI elements of the same type. The client and peer testers confirmed that information like current tool, time of day, and interaction feedback was consistently visible and easy to understand. Risk assessment ensured overlay elements did not obscure important content, and visual clarity was preserved even on lower-resolution displays by making sure that the concept of mixels, where the pixel size changes from sprite to sprite, did not occur, not only making the game more consistent aesthetically but preserving the readability of icons and text, no matter the screen size. The size of the NPC and chest prompt was carefully chosen to allow the indicator to be readable in the scene without being too large. |

## References

NSW Legislation (2023) [NSW Privacy and Personal Information Act (1998)](https://legislation.nsw.gov.au/view/whole/html/inforce/current/act-1998-133#statusinformation), website, **accessed 26 May 2023.**

Federal Register of Legislation (2023) [Federal Privacy Act (1988)](https://www.legislation.gov.au/Series/C2004A03712), Australian Government website, **accessed 26 May 2023.**

ISO Standards (2023) [ISO/IEC 27001](https://www.iso.org/standard/27001)**, website, accessed 26 May 2023.**

NESA (2023) [Software Engineering Course Specifications](https://library.curriculum.nsw.edu.au/341419dc-8ec2-0289-7225-6db7f2d751ef/94e1eb0a-0df7-4dbe-9b72-5d5e0d17143a/software-engineering-11-12-higher-school-certificate-course-specifications.PDF), **NESA website, accessed 26 May 2023.**

Atlassian, 2023. Types of software testing. [online] Available at: https://www.atlassian.com/continuous-delivery/software-testing/types-of-software-testing [Accessed 18 May 2025].

Atlassian, 2023. Project Management Guide. [online] Available at: https://www.atlassian.com/agile/project-management/project-management-intro [Accessed 18 May 2025].

Creative Commons Australia, 2023. What is Creative Commons? [online] Available at: https://smartcopying.edu.au/what-is-creative-commons/ [Accessed 18 May 2025].

Developer.mozilla.org, 2023. State machine - MDN Web Docs Glossary. [online] Available at: https://developer.mozilla.org/en-US/docs/Glossary/State\_machine [Accessed 18 May 2025].

Flowlab Community, 2020. Mini Rant - Stop Using Mixels. [online] Available at: https://community.flowlab.io/t/mini-rant-stop-using-mixels/38287 [Accessed 18 May 2025].

GeeksforGeeks, 2021. Common Subexpression Elimination - Code Optimization Technique in Compiler Design. [online] Available at: https://www.geeksforgeeks.org/common-subexpression-elimination-code-optimization-technique-in-compiler-design/ [Accessed 18 May 2025].

GeeksforGeeks, 2021. Code Optimization in Compiler Design. [online] Available at: https://www.geeksforgeeks.org/code-optimization-in-compiler-design/ [Accessed 18 May 2025].

GeeksforGeeks, 2021. Dead Code Elimination. [online] Available at: https://www.geeksforgeeks.org/dead-code-elimination/ [Accessed 18 May 2025].

GeeksforGeeks, 2021. Differences Between Black Box Testing vs White Box Testing. [online] Available at: https://www.geeksforgeeks.org/differences-between-black-box-testing-vs-white-box-testing/ [Accessed 18 May 2025].

GeeksforGeeks, 2021. Induction Variable and Strength Reduction. [online] Available at: https://www.geeksforgeeks.org/induction-variable-and-strength-reduction/ [Accessed 18 May 2025].

GeeksforGeeks, 2021. What is Code Motion? [online] Available at: https://www.geeksforgeeks.org/what-is-code-motion/ [Accessed 18 May 2025].

Godot Engine, 2023. About Godot - Introduction. [online] Available at: https://docs.godotengine.org/en/stable/about/introduction.html [Accessed 18 May 2025].

GregTech OvermindDL1, 2023. GregTech Wiki. [online] Available at: https://gregtech.overminddl1.com/ [Accessed 18 May 2025].

Hypixel Wiki, 2023. Hypixel Wiki Main Page. [online] Available at: https://wiki.hypixel.net/Main\_Page [Accessed 18 May 2025].

LinkedIn, 2023. What are the best strategies for designing scalable and modular software? [online] Available at: https://www.linkedin.com/advice/3/what-best-strategies-designing-scalable-modular [Accessed 18 May 2025].

LowSpecGamer Reddit, 2019. What is considered low spec? [online] Available at: https://www.reddit.com/r/lowspecgamer/comments/eo8nle/what\_is\_considered\_low\_spec/ [Accessed 18 May 2025].

Reddit r/godot, 2023. Help regarding queue\_free, free and remove\_child. [online] Available at: https://www.reddit.com/r/godot/comments/16yievc/help\_regarding\_queue\_free\_free\_and\_remove\_child/ [Accessed 18 May 2025].

Teach-ICT, 2023. Implementation methods. [online] Available at: https://www.teach-ict.com/as\_a2\_ict\_new/ocr/A2\_G063/335\_implementing\_systems/methods/miniweb/pg6.htm [Accessed 18 May 2025].

Township (video game), 2023. Wikipedia. [online] Available at: https://en.wikipedia.org/wiki/Township\_(video\_game) [Accessed 18 May 2025].

Wikipedia, 2023. XOR cipher. [online] Available at: https://en.wikipedia.org/wiki/XOR\_cipher [Accessed 18 May 2025].

AltexSoft, 2023. Non-Functional Requirements in Software Engineering. [online] Available at: https://www.altexsoft.com/blog/non-functional-requirements/ [Accessed 18 May 2025].

GitHub, 2023. What is GitHub? [online] Available at: https://www.techtarget.com/searchitoperations/definition/GitHub [Accessed 18 May 2025].