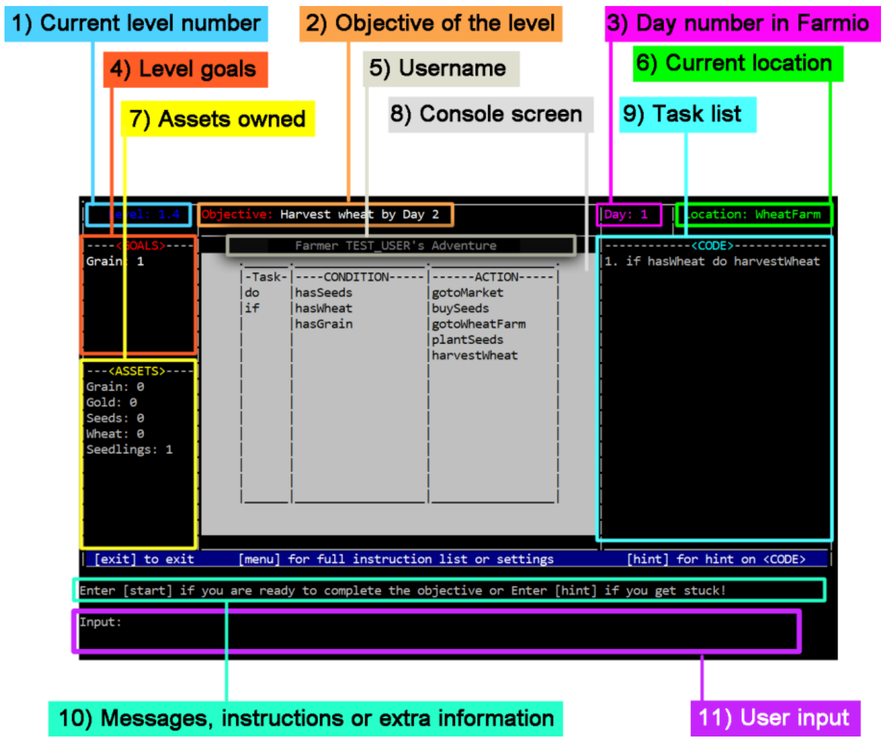
Tan Zheng Wen – Project Portfolio for Farmio

About the project

Farmio is a game that teaches computational thinking. It is mainly targeted for children aged 11-14. However, it is suitable for anyone who wants to start learning programming.

This is what our project looks like:



```
Level: 1.4   Objective: Harvest wheat by Day 2                Day: 1   Location: WheatFarm
----<GOALS>----      Farmer TEST_USER's Adventure        ------------<CODE>------------
Grain: 1                                                 1. if hasWheat do harvestWheat
                |-Task-|----CONDITION-----|------ACTION-----|
                |do    |hasSeeds          |gotoMarket       |
                |if    |hasWheat          |buySeeds         |
                |      |hasGrain          |gotoWheatFarm    |
                |      |                  |plantSeeds       |
                |      |                  |harvestWheat     |
---<ASSETS>----
Grain: 0
Gold: 0
Seeds: 0
Wheat: 0
Seedlings: 1

 [exit] to exit      [menu] for full instruction list or settings      [hint] for hint on <CODE>
Enter [start] if you are ready to complete the objective or Enter [hint] if you get stuck!
Input:
```

Figure 1. Farmio's Command Line Interface

My role was to design and write the codes for the user interface as well as the execution of actions and document the features I implemented in both the user guide and developer guide.

Note the following symbols and formatting used in this document

ⓘ	This symbol indicates important information
`start`	A grey highlight indicates that this is a command that can be typed into the command line and executed by the game
Enter	Grey text with no highlight indicate that this is a keypress
`Task`	A grey highlight with blue text indicates a component, class or object in the application

Summary of my contributions

The following section gives an overview of my contributions to this project, it shows a summary of my coding, documentation and other helpful contributions. This list is not exhaustive.

Key Features added:

1. I added the Task list Simulation feature. (PR #114, #176, #123)
   - What it does: It allows the user to easily visualise the execution of task they have added. This is achieved by visual representations of actions, highlighting of variable changes as well as highlighting the currently executing task.
   - Justification: One of the key problems faced in programming is the difficulty to visualise and understand the changes in variables during code execution. In Farmio, the task list acts as the code typed by users while the simulation of the task list allows users to visualise variable changes and helps them understand the code execution.
   - Highlights: The simulation feature of Farmio is not only implemented for the execution of the task list but also for various commands and the narratives. This feature beautifies the overall user interface and makes it more aesthetically pleasing compared to applications which are just plain text on most command line interfaces.

2. I added the Task Error Detection feature. (PR #114, #118, #141, #310)
   - What it does: It detects and identifies the task which has caused an error. It also shows the user why an error has occurred. This is achieved by highlighting the currently executing task and marking it as having an error. It also prints the corresponding explanations to why the error has been triggered.
   - Justification: With a framework that generates the explanation of errors based on what errors were made, the user can quickly understand exactly what went wrong and correct his code.
   - Highlights: The framework of the error detection mechanism caters for easy addition of action criteria as well as additional actions. This makes it easy to expand for future developments and modifications.

Other Features added:

1. Game console feature
    a. Layout design (PR #95, #96, #109, #118, #174)
    b. Interfacing username with the game console: (PR #183)
    c. Simulation framework (PR #73, #99, #122)
    d. Typewriter feature (PR #81)
2. Implemented action execution (PR #79, #99, #118, #98)
3. Implemented task execution (PR #80)
4. Implemented structure of the show list command (PR #145)
5. Implemented level progression (PR #140)
6. Story segment implementation and planning
    a. Narrative (PR #97, #130, #134)
    b. Hints (PR #174)

Testing added:

1. Test for actions (Pull request #298, #335)
2. Test for simulation (Pull request #283, #335)
3. Test for the game console (Pull request #298, #335)

Code contributions: Code contributed: Please click this link to see my code: [code contributions]

User guide contributions: Simulation segment.

Developer guide contributions: Task Visualisation Feature, Error Detection Feature and corresponding manual testing sections.

User Guide contributions

This section shows an excerpt of my contributions to the user guide. My full contributions include sections 2.2, 3.4.5, 3.4.12 and 3.5.

Simulation Segment

After you started the simulation, the game will now be in the Simulation Segment. During this segment, you should not enter any input until you are prompted to. If you do, your input will be carried forward to the next time the game requests for input.

This section explains what is happening during the simulation segment.

Do note that colours are not supported in the windows command prompt. As such, symbols are also used to help you understand the simulation.

Currently executing task

As seen in Figure 2., the task that is currently being executed is indicated by the '<<' symbol on the right of the simulation. Visual animation of the currently executing action is also shown in the centre of the game console.

On colour enabled terminals, the currently executing task is highlighted in green.

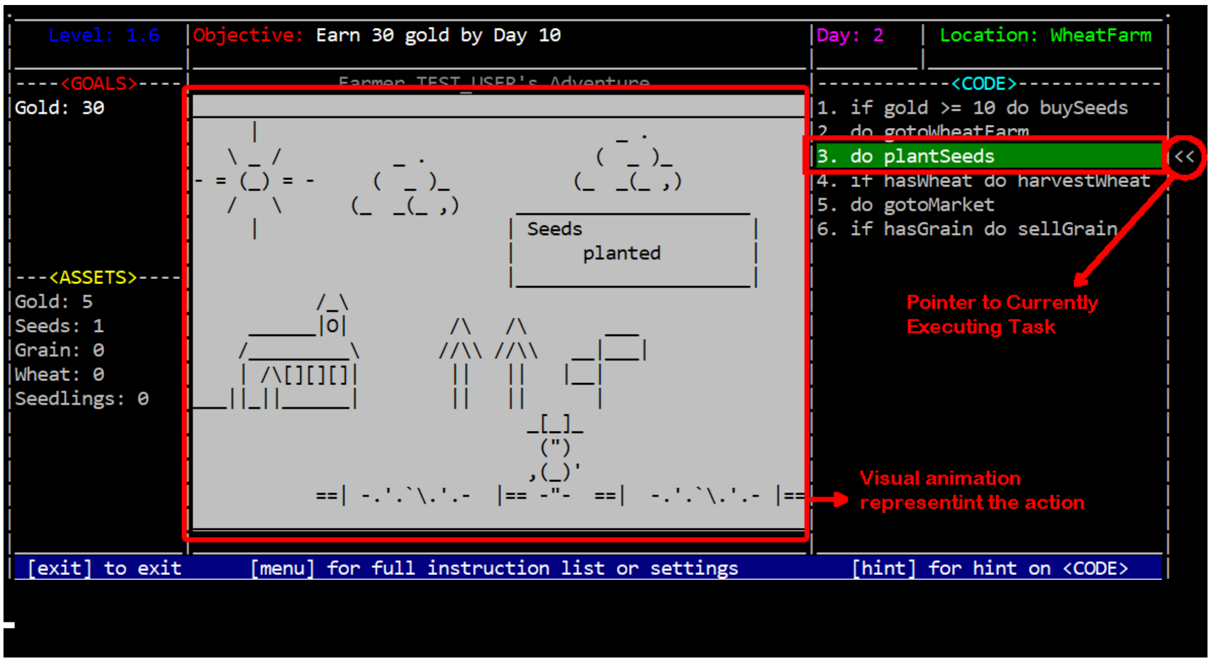

```
|   Level: 1.6   |Objective: Earn 30 gold by Day 10             |Day: 2    |  Location: WheatFarm |
|----<GOALS>-----|          Farmer TEST USER's Adventure         |------------<CODE>-------------|
|Gold: 30        |                                               |1. if gold >= 10 do buySeeds   |
|                |                                               |2. do gotoWheatFarm            |
|                |                                               |3. do plantSeeds               |<<
|                |                                               |4. if hasWheat do harvestWheat |
|                |                                               |5. do gotoMarket               |
|                |                              | Seeds        | |6. if hasGrain do sellGrain    |
|                |                              |      planted | |                               |
|---<ASSETS>-----|                                               |                               |
|Gold: 5         |                                               |                               |
|Seeds: 1        |                                               |                               |
|Grain: 0        |                                               |                               |
|Wheat: 0        |                                               |                               |
|Seedlings: 0    |                                               |                               |
| [exit] to exit       [menu] for full instruction list or settings       [hint] for hint on <CODE> |
```
Pointer to Currently Executing Task
Visual animation representint the action

Figure 2. Execution of 'plantSeeds' action

Error in executing a task

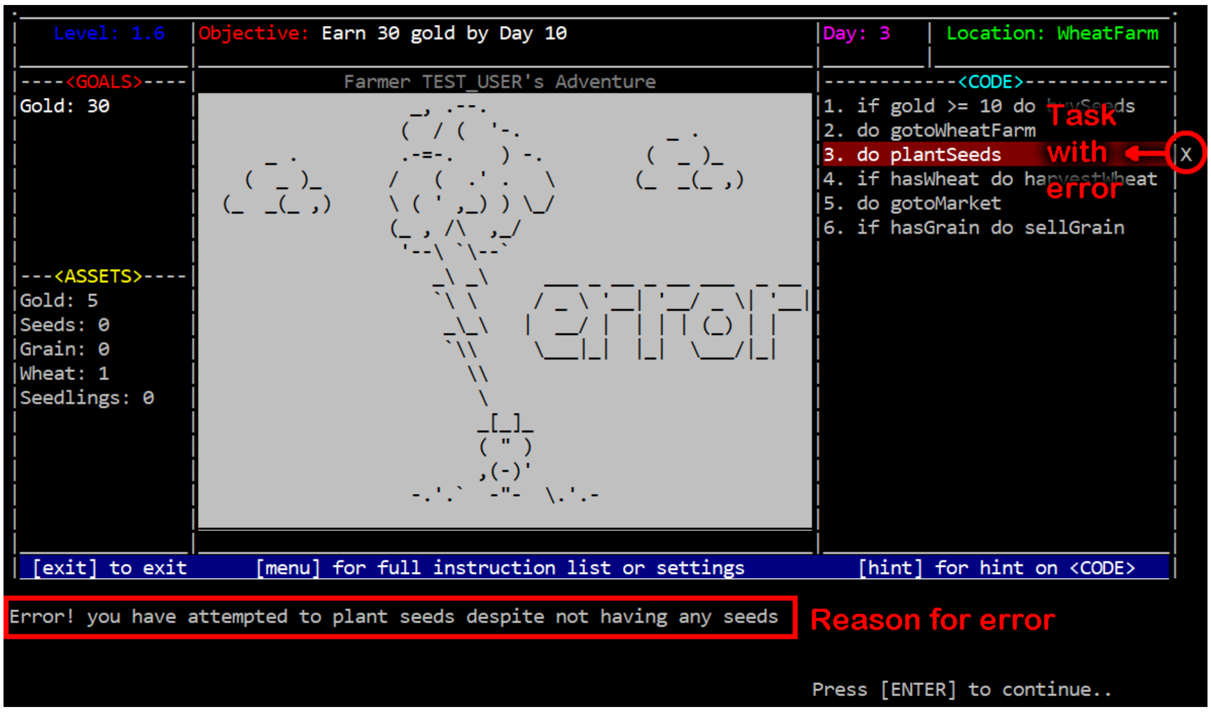
If an error is encountered during task execution, you can easily identify which task caused the error by finding the task marked with a symbol ‘X’ on the right.

On colour enabled terminals, the task with the error is highlighted in red.

You can also find out the reason for the error by looking at the message below the console. This can be seen in Figure 3. by the text marked as ‘Reason for error’.

Press `Enter` to proceed to reset the level. You will be brought back to the story segment of that level.

In the event that you have an error in execution, the game will reset the level for you. The existing list of tasks will be retained for easy modification.



```
|   Level: 1.6  |Objective: Earn 30 gold by Day 10                     |Day: 3   | Location: WheatFarm |
|----<GOALS>----|            Farmer TEST_USER's Adventure               |-----------<CODE>------------|
|Gold: 30       |                                                       |1. if gold >= 10 do buySeeds |
|               |                                                       |2. do gotoWheatFarm          |
|               |                                                       |3. do plantSeeds             |X
|               |                                                       |4. if hasWheat do harvestWheat
|               |                                                       |5. do gotoMarket             |
|               |                                                       |6. if hasGrain do sellGrain  |
|---<ASSETS>----|                                                       |                             |
|Gold: 5        |                                                       |                             |
|Seeds: 0       |                                                       |                             |
|Grain: 0       |                                                       |                             |
|Wheat: 1       |                                                       |                             |
|Seedlings: 0   |                                                       |                             |
| [exit] to exit     [menu] for full instruction list or settings          [hint] for hint on <CODE> |
Error! you have attempted to plant seeds despite not having any seeds

                                                        Press [ENTER] to continue..
```

Figure 3. Error in executing a task

Developer Guide contributions

This section shows an excerpt of my contributions to the developer guide. My full contributions include sections 3.2, 4.5, 4.6 and F.5, F.6, F.7.

Task Error Detection Feature

The Task error detection feature identifies and highlights tasks with errors and states which criteria of the action is not met.

Implementation

This error detection mechanism is facilitated by the `Action` abstract class and it's child classes. `Frontend` is used to format and show the error effectively. This feature is implemented through the following methods.

- `Action#execute()` - Executes the action.
- `Action#checkActionCriteria()` - Checks if the action criteria is met.
- `Farmer#setTaskFailed()` - Marks the current running task as failed.
- `Frontend#simulate()` - Highlights the task that triggered the error.
- `Frontend#show()` - Shows the reason for the error.

Below describes the high level behaviour of the error detection mechanism at each step when the task `do buySeeds` is executed but the action's criteria is not met. For example, the Farmer's location is not at the Market.

Step 1. In this case, the `DoTask#execute()` method will call the `BuySeedsAction#execute()` method to execute the action.

Step 2. However, as the Farmer being at the market is a criteria to buy seeds, `Action#checkActionCriteria()` would call `Farmer#setTaskFailed()` to mark that the current task has failed.

Step 3. `Frontend#simulate()` will then be called, triggering a cascade of method calls. Similar to section 4.5., `GameConsole#formatAndHighlightCode()` is eventually called.

Step 4. `GameConsole#formatAndHighlightCode()` is responsible for detecting that the currently running task is marked as failed and thus, highlighting it in red instead of green in the GameConsole.

Step 5. `Action#checkActionCriteria()` will then call `Frontend#show()` to show the reason why the task has failed (the farmer not being at the market in this case).

If more than one criteria for an action is not met, this feature will ensure that each explanation for every unmet criteria is generated by `Frontend#show()`.

Step 6. Lastly, a Farmio exception is thrown to signal the change in the stage of `Farmio` class to `LEVEL_FAILED` and the level is reset.

The sequence diagram below shows the behaviour of the task error detection mechanism.

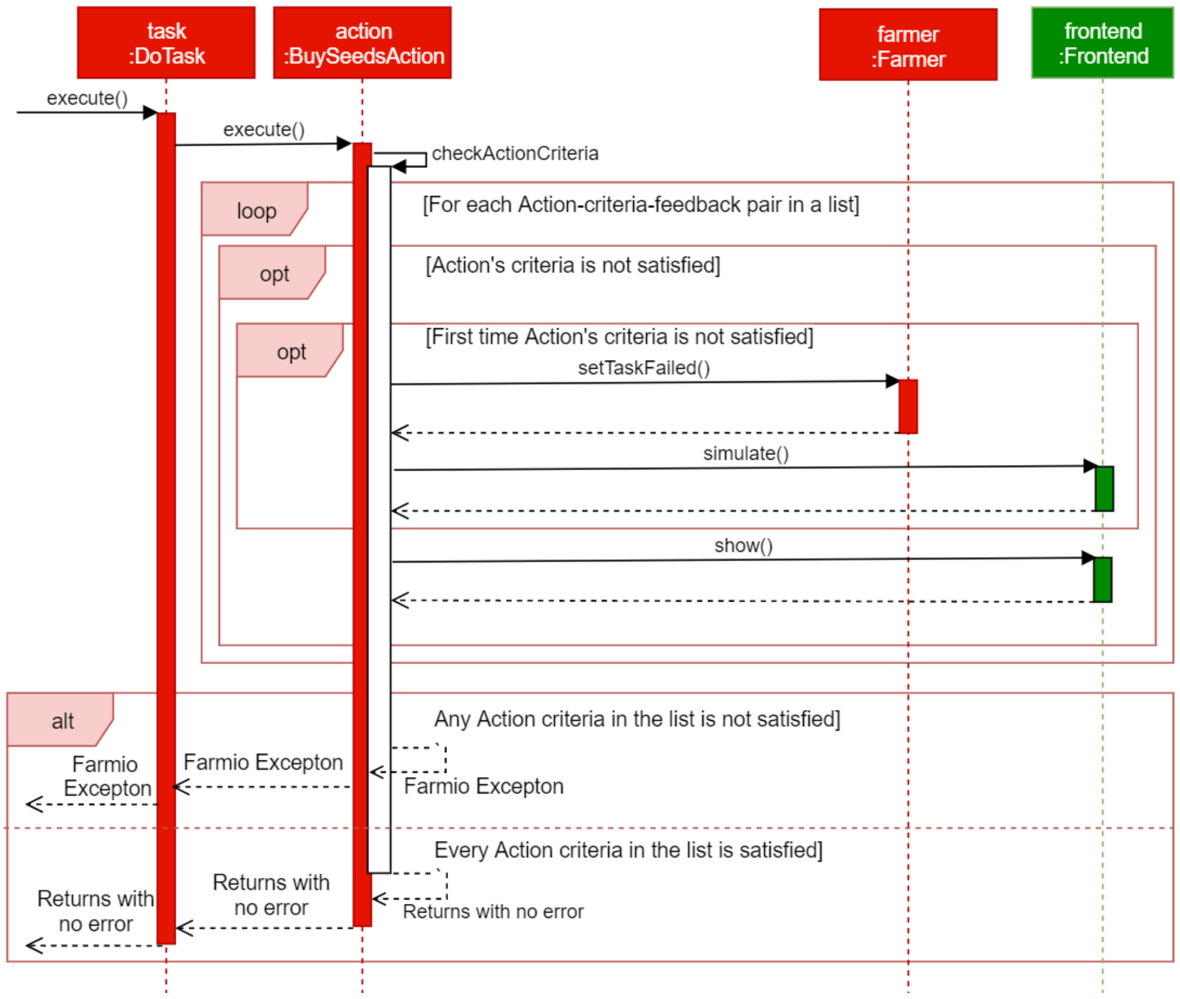


Figure 3. Sequence Diagram for Task Error Detection Mechanism

Design considerations

Aspect: Implementation of the error detection feature.

- Alternative 1 (current choice): Implemented the error detection method in `Action` abstract class.
    - Pros: Increased cohesion. Classes using the same methods would not need to define them separately. By localising the method, it is also easier to modify them.
    - Cons: Decreased coupling. Action child classes are more reliant on each other.
- Alternative 2: Implement individual checking methods in each execute method of the `Action` child class.
    - Pros: Increase coupling. Classes would depend less on one another. If the developer chooses to change the mechanics of one method, it will have less effect on the rest.
    - Cons: Decreased cohesion and increased complexity.