

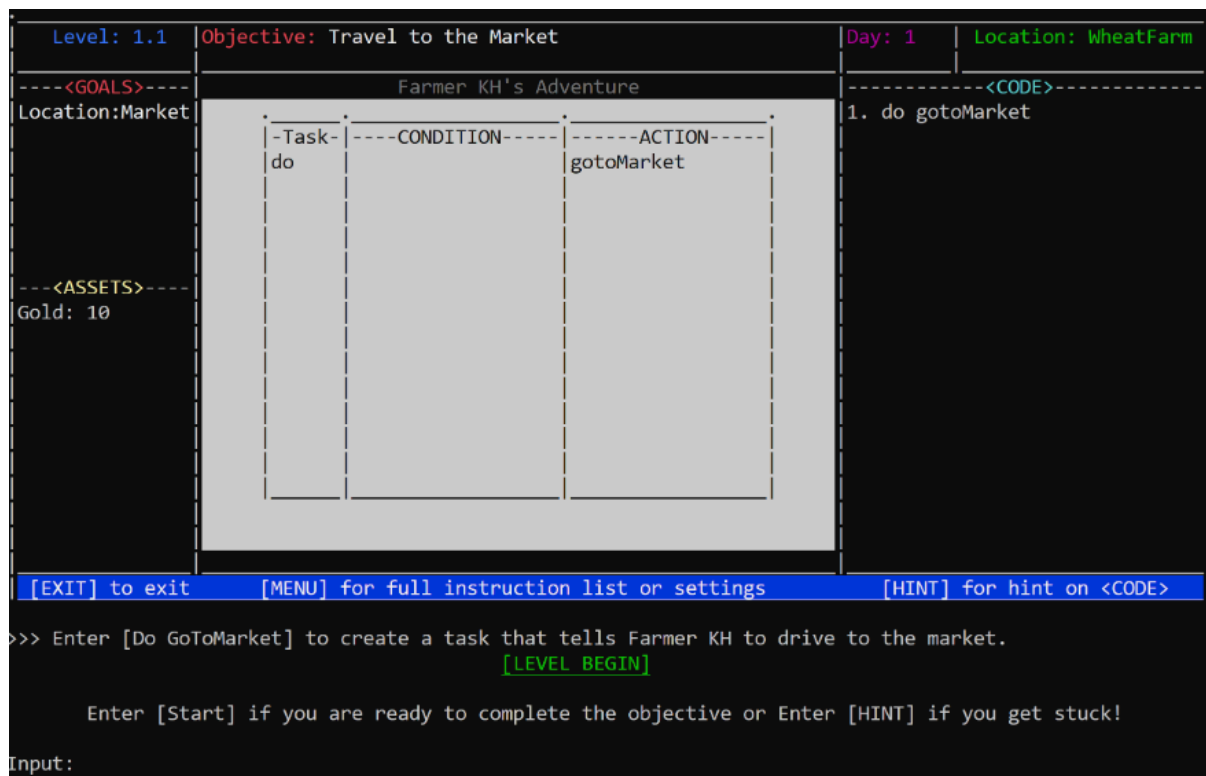
Umar Bin Moiz – Project Portfolio

Project: Farmio

About the Project

My team and I were tasked with enhancing a basic task list manager for our Software Engineering project. We decided to morph into a command line interface game called Farmio. The objective of our application is to teach basic computational thinking to students who are complete novices but are interested in learning programming.

This is what our project looks like:



```
Level: 1.1 | Objective: Travel to the Market | Day: 1 | Location: WheatFarm
-----<GOALS>-----
Location:Market
-----<ASSETS>-----
Gold: 10

Farmer KH's Adventure

-Task-  -CONDITION-  -ACTION-
do      [Empty]       gotoMarket

1. do gotoMarket

[EXIT] to exit | [MENU] for full instruction list or settings | [HINT] for hint on <CODE>

>>> Enter [Do GoToMarket] to create a task that tells Farmer KH to drive to the market.
      [LEVEL BEGIN]

      Enter [Start] if you are ready to complete the objective or Enter [HINT] if you get stuck!

Input:
```

My role was to design and implement a number of features, such as the level progression, user feedback and log feature. The following sections illustrate these enhancements in more detail, as well as the relevant documentation I have added to the user and developer guides in relation to these enhancements.

Note the following symbols and formatting used in this document:



This symbol indicates important information

New

A grey font indicates a button or clickable option.

start

A grey highlight indicates that this is a command that can be typed into the command line and executed by the game

Task

A grey highlight with blue text indicates a component, class or object in the application

Summary of Contributions

This section summarizes the contributions I made towards the project. It consists of my coding, documentation and other helpful contributions made towards the team project.

Enhancement added:

Level Progression/Reset feature

The level progression/reset feature controls whether the user advances to the next level or restarts the level.

- **What it does:** the level progression feature is used to check on whether the objectives of the level have been met. If the user passed the level, the level progression feature will move the user to the next level. If however, the user fails to complete the tasks, the level will be reset to its initial state.
- **Justification:** If the user completes the objectives of the level, they should be moved to the next level in order to continue playing the game. However, if they failed to achieve the objectives of the level, the level reset feature should allow them to revert back to the initial state of the level to allow them to reattempt the level.
- **Highlights:** This feature also works well with other commands such as the `reset` command which allows users to reset the level stage at the end of each day.

Log Feature

The log feature allows the user to see the tasks which were executed for the level which they had just played.

- **What it does:** The `log` command allows the user to see all the tasks that were executed for the previous task run
- **Justification:** This feature is important since it allows the user to see the runtime execution of tasks which they had programmed. In the event that they have failed to complete a level, the log feature will allow them look at their program and possibly find the error in their logic. If they had successfully completed the level, they are also able to look at the log list and recap what was required to complete the level.
- **Highlights:** This feature can span multiple pages

Other contributions:

Enhancement to existing features

- Provided the ascii art templates and animation for Farmio (Pull request #115)
- Added different types of feedback for different conditions (Pull request #199, #210, #296, #326, #338)
- Implemented test cases #345

Documentation:

- Made cosmetic improvements to the user guides to make it more reader friendly
- Improved the user guide

Community:

- Reviewed Pull Requests

Contributions to the User Guide

We had to create our own User guide from scratch since we decided to morph our project from a task manager to a game that teaches computational thinking. As such we needed to provide instructions for the user on the features and actions we had incorporated. The following is from Farmio's User guide, showing the additions that I had made for the level progression/reset, and log feature.

1. Log Feature

The log command allows you to see the task sequence that was run in the previous simulation.

Format: `log PAGENUMBER`



`PAGENUMBER` Must be a positive non-zero integer

Each log page displays a maximum of 15 tasks per page. You can navigate through the log pages using the log command. For example, to view page two of the log, enter `log 2`. If there was an error during code execution, the log does not display the particular task which caused the error

Contributions to the Developer guide

The following section shows my additions to the Farmio Developer Guide for the level progression and resetting features. My contributions focus on the level progression/reset and logging feature.

1. Level Progression and Reset feature

The level progression and reset feature is responsible for allowing the game to progress to the next level or whether to reset to the previous level.

1.1 Level Progression

The level progression feature allows the user to progress to the next level of the game if the user successfully completes the objectives for the current level.

1.1.1 Implementation

The level progression feature is facilitated by the `Command#CommandCheckObjectives()`, `Command#CommandLevelEnd()` methods and the `Level` class.

The `Command` class also utilises these methods from the `Level` class to check whether the objectives for the level have been met before attempting to progress onto the next level.

- `Level#checkAnswer()` - is used to check on the state of the state of the level, whether the user has completed the task successfully (`DONE`), have yet to complete the task but is within the allocated deadline (`NOT_DONE`), encountered an error during code execution (`INVALID`) or exceeded the stipulated deadline (`FAILED`) for the level.
- `Level#checkDeadlineExceeded()` - checks whether the user has exceeded the deadline.
- `Level#allDone()` - checks whether the user has completed all the tasks.

The steps below explain the behaviour of the level progression feature

Step 1: At the end of each day, the `Command#CommandCheckObjectives()` method will call the `Level#checkAnswer()` method in the `Level` class. The `Level#checkAnswer()` method will then invoke the `Level#checkDeadlineExceeded()` and `Level#allDone()` method to check whether the objectives for the level has been met within the preallocated deadline and will return either one of these `Level.ObjectiveResult` state - (`NOT_DONE/ DONE/ FAILED/ INVALID`)

Step 2 : If the objectives has been met for the level by the user. The `Command#CommandCheckObjectives()` method will then change the `STAGE` in the `Farmio` class to `LEVEL_END`. The `Parser` class will then call the `Command#CommandLevelEnd()` method.

Step 3 : The `Command#CommandLevelEnd()` method obtains the `Farmer` object. It updates the current level of the `Farmer` object to the next level before creating a new `Level` object instance. The `Level` class will then obtain values from a `JSON file` object to initialise and set the objectives for the next game level.

1.2 Level Reset

The Level Reset feature will reset the level of the game back to its initial state. There are 3 instances which would lead to the resetting of a level.

1. The first is if the user decides to initiate the reset manually when prompted with the option at the end of each day.
2. The user fails to complete the level within the allocated deadline.
3. There is an error encountered during code execution

1.2.1. Implementation

The Level Reset feature is facilitated by the following methods

- `Command#CommandCheckObjectives()` - checks whether the user has exceeded the deadline, completed the objectives or whether they encountered an error during task execution.
- `Command#CommandLevelReset()` - initiates the reset for the level.
- `Parser#parseDayEnd()` - checks whether the user wants to reset at the end of each day.

The steps below show the behaviour of the level reset feature:

Step 1: At the end of each day or level , the `Command#CommandCheckObjectives()` method will be invoked. It will check whether the objectives of the level has been met and whether the user has exceeded the deadline. It follows the same implementation for the checking objectives as level progression.

Step 2.1.1 : If the user has exceeded the deadline or if the user encountered an error during task execution, `Farmio.Stage` will be set to the `LEVEL_FAILED` state. `Parser#parse()` will then invoke the `Command#CommandLevelReset()` method.

Step 2.1.2 : If the user has not exceeded the deadline and have yet to complete the task , `Farmio.Stage` will be set to the `DAY_END` state. `Parser#parseDayEnd()` method will then check whether the user had decided to key in “reset”. If yes the `Command#CommandLevelReset()` method will be invoked

Step 2.3 : `Command#CommandLevelReset()` method will then reset the game level back to its initial state

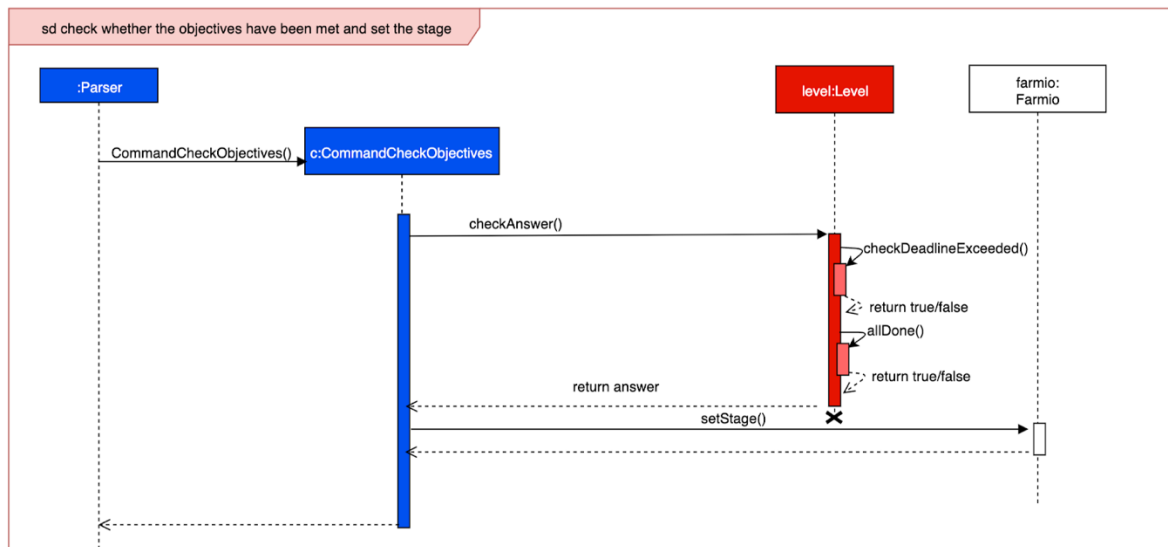


Figure 1.2.1.1. Sequence Diagram for Checking Objectives and Setting of Stage

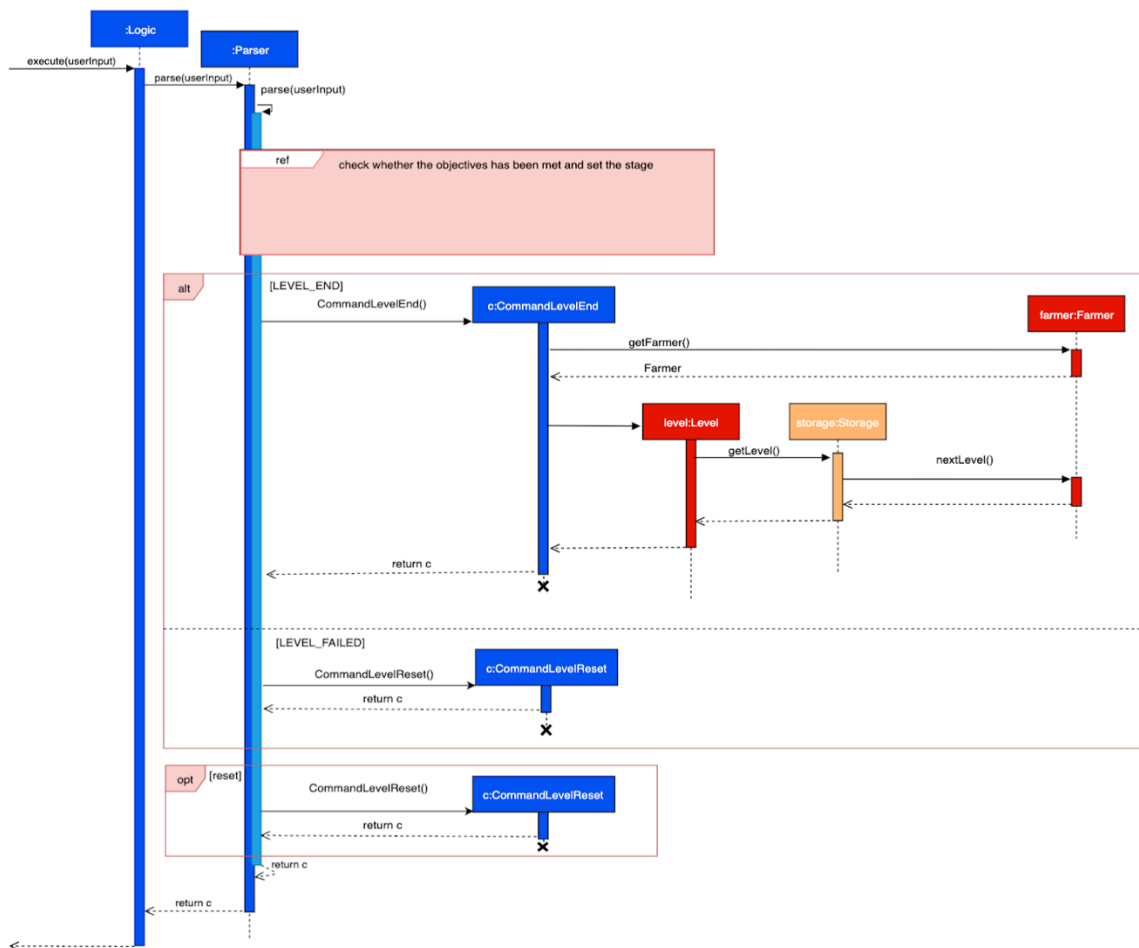


Figure 1.2.1.2 Sequence diagram for Level Progression and Reset Mechanism

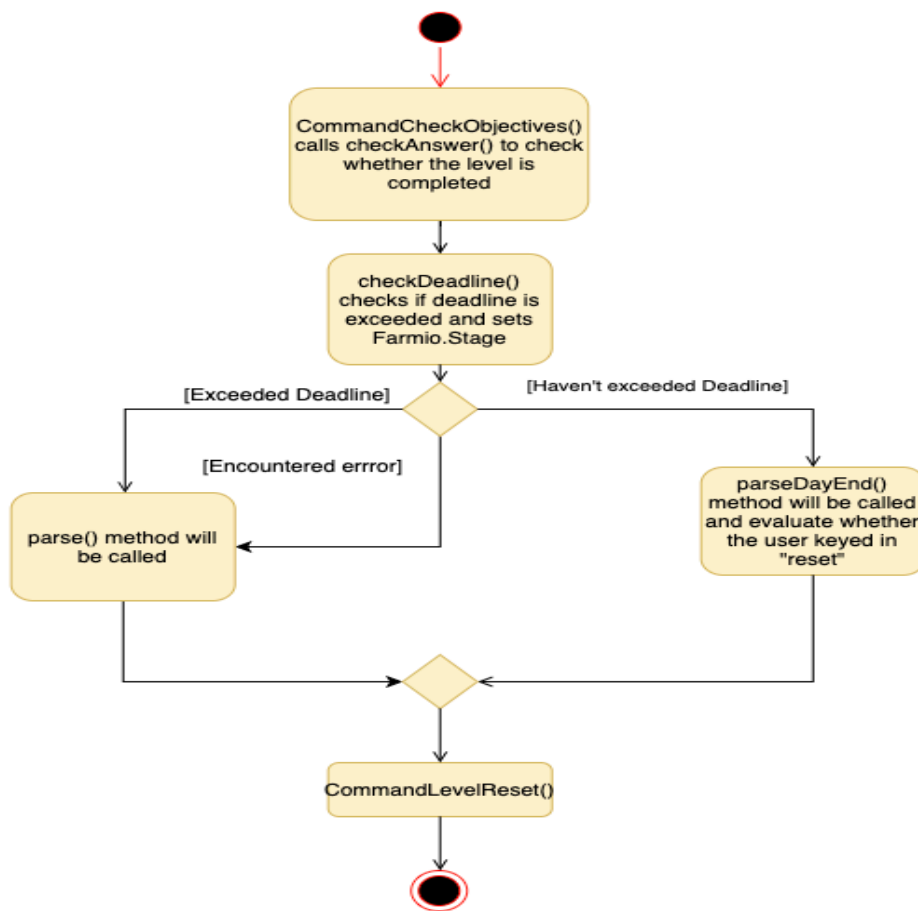


Figure 1.2.1.3. Activity Diagram for Reset Mechanism

2.1 Game Logging Feature

The Logging feature allows the user to see their previous runtime code execution.

2.1.1 Populating and Clearing of the Log

The log is populated at the start of each day when the tasks are executed and is cleared whenever the user keys in "start" during the `TASK_ADD` stage.

2.1.1.1 Implementation

The populating and clearing of the log data will utilise the following methods

- `Command#CommandTaskRun()` - used to determine when to add to the log
- `Farmer#startDay()` - where the log is populated
- `Parser#parseTaskAdd()` - used to obtain user input and check when to start the day
- `Command#CommandDayStart()` - used to determine when to clear the log
- `Log#clearLogList()` - Clears the data in the log

2.1.1.1.1 Clearing of Log

Step 1: `Parser#parseTaskAdd()` will obtain user input.

Step 2: If `Parser#parseTaskAdd()` determines that the user keyed in “start”, it will invoke the `Command#CommandDayStart()` method.

Step 3: The `Command#CommandDayStart()` method will then check on the current `STAGE` of Farmio. If the current stage is set to `TASK_ADD`, the `Log#clearLogList()` method will be called and the contents inside the log will be emptied.

2.1.1.1.2 Adding to the Log

Steps 1: When `Command#CommandTaskRun()` method is invoked, it calls the `Farmer#startDay()` method.

Steps 2: The `Farmer#startDay()` method will then add the tasks that are being executed for that day.

2.1.1.1.3 Viewing the Log

The feature will allow the user to view their log files

The View Logging feature will utilise the following methods

- `Parser#parseTaskLog()` - Parse the log command from the user
- `Command#CommandLog()` - executes the log command
- `Log#getLogTaskList()` - obtains the log details
- `Log#toStringSplitLogArray()` - modifies log data to appropriate format for UI

Steps 1: `Parser#parseTaskAdd()` will obtain user input. If the user enters log [Position], the `Command#CommandLog()` method will be invoked.

Steps 2: The `Command#CommandLog()` method will obtain the log data when calling the `Log#getLogTaskList()` method. The `Log#toStringSplitLogArray()` method will then be used to modify the log data to a certain format.

Step 3: The log data will then be passed to `FrontEnd` to display log details to the user.

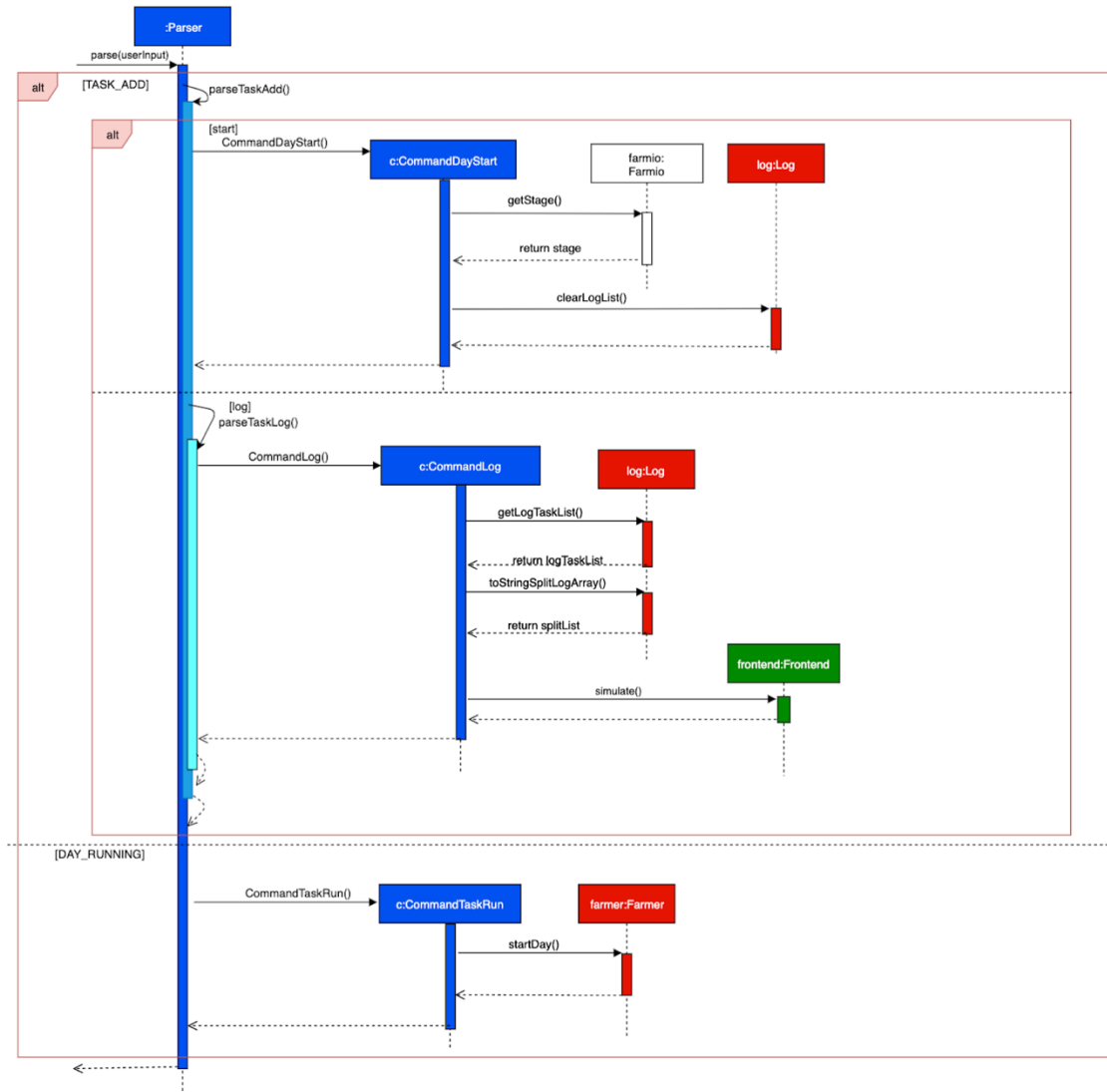


Figure 2.1.1.2 Sequence diagram for logging feature