CCPROG3 MCO1 Program Specifications

Program Overview – Verdant Sun Farming Simulator

Note: *MCO1 and MCO2 is to be done by pair, and by the same pair of students barring any reports of freeloading or decision by the pair to split. In the event of a split for any reason, the students who used to be in the pair would now need to work solo; they may not join an existing group or pair up with a new partner.*

Your task for MCO1 is to create a farming simulator called "Verdant Sun". You take on the role of a farmer planting and caring for crops that bear some yield after a number of days. At the end of a planting season (each season, for the sake of MCO brevity, is only 15 days), the total amount of money earned (savings) is tabulated and saved in a high score table (see **End Game**).

At the start of the game, give the player a starting savings of 1000. It will be day 1.

This farming simulator has following functionalities:

1. **Plant a Seed**
   Plants start out as seeds. Each **plant** has some properties, supplied via a json file (Plants.json) that will be uploaded alongside these specifications. These properties include:
   a. Plant name – The name of the plant. Plant names are typically composed of letters but may also contain special characters.
   b. Price – How much a seed of the specific plant costs.
   c. Yield – How much crop can be harvested from the plant once it has fully grown.
   d. Max Growth – An integer that indicates the growth stage of a plant. Alongside Current Growth (see next property), this is used to determine if the plant is already mature and is ready to be harvested.
   e. Current Growth – An integer that tracks which growth stage the plant currently is in. This starts at 0 when the plant seed is planted, and increases an amount per day (typically one per day if the plant has been watered that day, but may vary depending on certain factors) until it is greater than or equal to Max Growth, at which point it stops increasing.
   f. Preferred Soil – The Soil the plant prefers to grow on. If a seed is planted on its preferred soil, it grows 2 stages per day instead of one stage.
   g. Watered – A flag that determines if the plant has been watered for the day. Defaults to False and becomes True once the plant has been watered. Next day, becomes false again until the plant is watered.

   To plant a seed, players must select which tile in the field the seed is to be planted in. A **field** is composed of a 10 x 10 grid of tiles, each tile being a kind of **soil**. A **soil** has a name (either "loam", "sand" or "gravel"), a plant that has been planted on it (defaults to null until a plant is planted on the soil), and fertilizer used on the soil (defaults to null until fertilizer is applied to the soil; see **Apply Fertilizer**).

   Upon planting, the price of the plant is deducted from the player's savings. If the player does not have enough money, then they cannot plant. Once planted, the plant remains on the soil until it is harvested or removed.

2. **Remove/Harvest a Plant**
   This action may only be performed on a soil with a plant on it. Players select a tile (which, as described in the previous functionality, is a soil) and if the plant on that tile is not fully mature, it will be removed. Removed plants don't revert to being seeds, they are just destroyed upon removal.

   If the plant instead is fully mature, it is harvested and crop is collected instead. **Crops** have the following properties (the values for these properties are also found in Plants.json):
   a. Crop name – Name of the crop. This usually matches the name of the plant that produced the crop.
   b. Price – How much the crop sells for.

   Harvesting a plant automatically sells the crop it yields. The price of the crop is added to the player's savings (a tracker of how much money the player currently has). Note that plants aren't

automatically harvested the moment they mature; the player must manually harvest them. Also, mature plants do not grow or change growth stages anymore. They simply remain as mature plants on the soil until they are harvested.

Removing or harvesting a plant removes the plant from the soil, enabling the player to plant a seed on the soil again.

3. **Water a plant**
   Using a watering can, players can water soil with plants on them. A **watering can** has the following properties:
   a. Max water level – The maximum amount of water that can be placed in the watering can. For this game, this is simply a whole number. For MCO1, this will be set to 10.
   b. Current water level – Determines how much water the watering can currently has. This cannot be greater than the Max water level and cannot go below 0. The watering can starts full at the beginning of the game.

   Upon watering, the watering can loses one water level. The plant in the soil is also watered (i.e. the tracker for if the plant has been watered becomes true). Watering cannot be done if the watering can has a water level of zero; the watering can has to be refilled first.

   For the purposes of this game, soil without a plant on it cannot be watered.

4. **Refill Watering Can**
   Fills the watering can up to its max water level, regardless of how much water the can currently has. Using this action deducts 100 from the player's savings. Players cannot refill the watering can if they do not have enough money in their savings.

5. **Apply Fertilizer**
   Players can apply fertilizer to any soil, whether it has a plant on it or not. However, players cannot apply fertilizer on soil that already has fertilizer. **Fertilizer** has the following properties:
   a. Name – Name of the fertilizer.
   b. Price – How much each application of the fertilizer would cost.
   c. Effect days – How many days the fertilizer would have be in effect.

   Upon application of fertilizer to a soil (single instance), the price of the fertilizer is deducted from the player's savings. Players cannot apply fertilizer if they cannot pay for it.

   Fertilizer affects the growth of a plant that is planted on the soil that has the fertilizer. Instead of growing one stage per day (or two stages in case the plant is planted on a soil it prefers), it grows an additional stage (for a potential three stages if the plant is also on a soil it prefers).

   Each day that a fertilizer affects the growth rate of a plant, its effect days is reduced by one, until it eventually runs out. At that point, the soil it is applied on will no longer have fertilizer (which means fertilizer can be applied to it again). Note that this reduction of effect days only happens if a plant is planted on the soil the fertilizer is applied to; if there is no plant on the soil, the fertilizer remains in the soil.

   A json file (Fertilizers.json) will be provided that has the fertilizer variants.

6. **Next Day**
   When this action is invoked, the time of the game moves on to the next day. Several things happen as the current day ends:
   1. All watered plants grow one stage. They grown an additional stage if they were planted on preferred soil.
   2. All watered plants become no longer watered.
   3. All plants planted on soil with fertilizer grows an additional stage.
   4. Fertilizers placed on soil with plants on them have their effect days reduced by one (unless the soil has been fertilized by a meteorite, see **Meteorite Event**).
   5. The player will gain 50 in their savings.

End Game
The game ends after one planting season. The game saves the player's name and final savings in a high score table. This table can only contain 10 entries; if the player's final savings is less than the lowest savings in the high score table, then their entry will not be saved. The high score table must be persistent

(i.e. must be saved as a json file, see the included HighScores.json file for the sample format of the json file).

Meteorite Event
At the end of the seventh day, after the "Next Day" events have transpired, a meteorite hits the field and affects the grey colored tiles below:

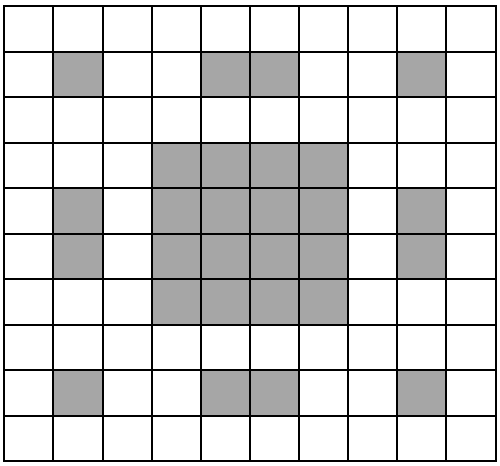

Figure 1.1: Tiles affected by the meteorite

If plants are on affected tiles, they are all removed/harvested. Plants cannot be planted on affected tiles. On the succeeding days, the player gets an additional action: **Excavate Meteorite**. This action costs 500 per tile, and only up to five affected tiles can be excavated per day (of course, only if the player has money to). Once excavated, the tile returns to the soil type it was before the meteorite event, and becomes permanently fertilized.

Additional Details
Here are a few clarifications to help better understand the limitations/scope of the specifications:

- The field is a 10 by 10 grid of tiles. The soil variants and the map layout will be provided via a json file (Map.json).
- Only if a plant is watered does it grow when the day ends. It doesn't matter if the plant was on preferred soil or if the soil is fertilized; the plant would not grow unless it is watered.
- Fertilizer only gets consumed if the plant planted on the soil it is applied to grows (remember that plants only grow if watered). You may fertilized soil with no plant on it; the fertilizer will just stay on the soil.
- Soil that has been fertilized cannot be fertilized anymore until the fertilizer that has been applied to it is consumed (remember that fertilizer gets consumed if the plant on the soil it was applied to grows).
- In a similar vein, you cannot water a plant that has already been watered. Remember also that you cannot water a tile with no plant on it.
- In a similar vein, you cannot plant on a tile that already has a plant.
- Removing and harvesting are variants of the same action. That is, when selected from a menu and when used on a plant, the game decides if the action is removal or harvesting.
- Removing a plant does not revert the plant to being a seed. It is just removed. Similarly, harvesting a plant does not yield seeds, it only yields crop which is immediately sold.
- The amount of crop created is dependent on the yield of the plant. There is no variation (i.e. it is not a maximum number, nor is the crop amount determined by a random number generator.
- Provide a reasonable text-based interface for the system. At the very least, the player's savings and the field should be displayed when the game starts and at the start of each day. Ideally, the updated field is also displayed after the player plants, waters, applies fertilizer or removes/harvests a plant. Make sure the interface is legible, and does not require you (the developer) to explain what the screen elements and symbols mean.
- When planting, watering, fertilizing or removing/harvesting, include an interface that allows the player to perform the action on multiple tiles. Make sure that the input strategy for the tiles are reasonable (i.e. does not require too much complex inputs). Perhaps a comma separated string of column or row labels may be used.

Other Requirements

1. The design and implementation of the solution should…
    - Conforms to the specifications described above
    - Exhibit proper object-based concepts, like encapsulation and information-hiding

2. Interaction with the user (i.e. input and output) should be through a command line interface (CLI). No graphical interface is expected for MCO1.
3. To allow for an easier time to validate the program, usage of libraries outside of what is available in the Java API is not allowed.

Please also note that MCO2 will look to extend some mechanisms of MCO1; however, MCO2's specifications will be delayed in order to simulate additional mechanics or modifications requested by a client after development of the base system. Additionally, a graphical user interface should be implemented only for MCO2.

Screenflow and Text-based User Interface

As the application loads, show the player the initial field (again, the layout will come from a json file that has been released alongside this specifications document). Also show the player how much savings they currently have (1000 at the start of the game).

Then, show the player a main menu of actions they can perform. Note that if an action cannot be performed (see the specifications above), the menu item should not be shown. The menu can have at most the following commands:

- Plant a seed
- Water a plant
- Refill watering can
- Apply fertilizer
- Remove a plant
- Excavate meteorite
- Next day

If the player picks Plant a seed, provide another menu that lists all plants that could be planted (remember that info for the plants are in a json file released alongside this document). If the player does not have enough money to plant a certain kind of plant, do not show it. Additionally, give the player the option to cancel the action. Cancelling the action reverts the player back to the main menu.

If the player selects water a plant, the player must then be given the chance to select either a single tile to water, or a list of tiles to water. If the former is selected, make sure to check if they selected a legal tile. If the tile they selected is not legal for watering, the water level of the watering can must not be reduced. Only if the tile is legal should the level of the watering can be reduced. If the latter option is selected, after inputting a list of tiles to water, the game should attempt to water the tiles one at a time. If the tile is illegal for watering, go to the next tile in the list. If the water level of the watering can becomes 0, skip the rest of the list. Of course, you must still provide the player the option to cancel and go back to main menu at any point in the watering action, (this could potentially result in the action being cancelled) except during the resolution of the list of tiles to water (the whole list must be resolved before the player is given the option to cancel).

When refill watering can is selected, 100 is deducted from the player's savings regardless of the watering can water level (i.e. even if the watering can is already full to begin with). The watering can's water level becomes its maximum water level.

When Apply fertilizer is selected, first have the player select a fertilizer to use (fertilizer information is provided in a json file released alongside this document). Then, have the player either select a single tile, or a list of tiles to apply fertilizer to, similar to the water a plant action. Similar to that action, check if the tile or tiles are legal before applying fertilizer to it. In the case of the latter, if the player's savings is not enough to fertilize the next tile, skip the rest of the list. Also like the water a plant action, provide the player an option to cancel the action at any point except when the list of tiles is being resolved.

When Remove a plant is selected, again similar to water a plant, the player must be given the option to either select a tile or a list of tiles. When resolving either action, make sure the tile is legal before the action is performed. When resolving a list of tiles, the tiles are resolved one at a time, skipping illegal tiles. If the plant on a selected tile is not fully grown yet, it is removed; if it is, it is harvested. As before, provide the player an option to cancel the action at any point except when the list of tiles is being resolved.

Excavate Meteorite works similar to removing a plant, in that the player is also given the option to either select a tile or list of tiles. Again, in resolving the selection, make sure the tile is legal. Illegal tiles are skipped when excavating multiple tiles. If there is not enough money left to excavate the rest of the list of

tiles, then the rest of the list is skipped. As before, provide the player an option to cancel the action at any point except when the list of tiles is being resolved.

Finally, if Next day is selected, next day actions (see **Next Day**) are performed. If it is the end of the seventh day, the **Meteorite Event** is also performed.

General Instructions

Deliverables
The deliverables for both MCOs include:

1. Signed declaration of original work (declaration of sources and citations may also be placed here)
    - See Appendix A for an example
2. Softcopy of the class diagram following UML notation.
    - Kindly ensure that the diagram is easy to read and well structured
3. Javadoc-generated documentation for proponent-defined classes with pertinent information
4. Zip file containing the source code with proper internal documentation
    - The program must be written in Java
    - Test script following the format indicated in Appendix A
5. A video demonstration of your program (this is under the discretion of the instructor; you may not be required to submit this)
    - While groups have the freedom to conduct their demonstration, a demo script will be provided closer to the due date to help with showing the expected functionalities.
    - The demonstration should also quickly explain key aspects of the program's design found in the group's class diagram
    - Please keep the demo as concise as possible and refrain from adding unnecessary information

Submission
All deliverables for the MCO are to be submitted via **Canvas**. Submissions made in other venues will not be accepted. Please also make sure to take note of the deadlines specified on Canvas. No late submissions will be accepted.

Grading
For grading of the MCO, please refer to the MCO rubrics indicated in the syllabus or the appendix.

Collaboration and Academic Honesty
This project is meant to be worked on as a pair (i.e. max of 2 members in a group). In exceptional cases, a student may be allowed by their instructor to work on the project alone; however, permission should be sought as collaboration is a key component of the learning experience. Under no circumstance will a group be allowed to work on the MCO with more than 2 members.

A student cannot discuss or ask about design or implementation with other persons, with the exception of the teacher and their groupmate. Copying other people's work and/or working in collaboration with other teams are not allowed and are punishable by a grade of 0.0 for the entire CCPROG3 course and a case may be filed with the Discipline Office. In short, do not risk it; the consequences are not worth the reward[1]. Comply with the policies on collaboration and AI usage as discussed in the course syllabus.

Documentation and Coding Standards
Do not forget to include internal documentation (comments) in your code. At the very least, there should be an introductory comment and a comment before every class and every method. This will be used later to generate the required External Documentation for your Machine Project. You may use an IDE or the appropriate command-based instructions to create the documentation, but it must be PROPERLY constructed.

Please note that we're not expecting you to add comments for each and every line of code. A well-documented program also implies that coding standards are adhered to in such a way that they aid in the documentation of the code. Comments should be considered for more complex logic.

Bonus Points
No bonus points will be awarded for MCO1. Bonus points will only be awarded for MCO2. To encourage the usage of version control, please note that a small portion of the bonus points for MCO2 will be the

[1] What is a measly passing grade compared to a life-long burden on your conscience?

usage of version control. Please consider using version control as early as MCO1 to help with collaborating within the group.

Resources and Citations
All sources should have proper citations. Citations should be written using the APA format. Examples of APA-formatted citations can be seen in the References section of the syllabus. You're encouraged to use the declaration of original work document as the document to place the citations.

Demonstration Delivery
The mode of delivery of the demo for MCO1 is left to the discretion of your instructor. All members are expected to be present in the video demonstration and should have relatively equal parts in terms of the discussion. Any student who is not present during the demo will receive a zero for the phase.

During the MP demo, it is expected that the program can be compiled successfully and will run. If the program does not run, the grade for that phase is 0. However, a running program with complete features may not necessarily get full credit, as implementation (i.e., code) will still be checked.

Other Notes
You are also required to create and use methods and classes whenever possible. Make sure to use Object-Based (for MCO1) and Object-Oriented (for MCO2) Programming concepts properly. No brute force solution. When in doubt, consult with your instructor.

Statements and methods not taught in class can be used in the implementation. However, these are left for the student to learn on his or her own.

Appendix A. Template for Declaration of Original Work

Declaration of Original Work

We/I, [Your Name(s)] of section [section], declare that the code, resources, and documents that we submitted for the [1st/2nd] phase of the major course output (MCO) for CCPROG3 are our own work and effort. We take full responsibility for the submission and understand the repercussions of committing academic dishonesty, as stated in the DLSU Student Handbook. We affirm that we have not used any unauthorized assistance or unfair means in completing this project.

[*In case your project uses resources, like images, that were not created by your group.*] We acknowledge the following external sources or references used in the development of this project:

1. Author. Year. Title. Publisher. Link.
2. Author. Year. Title. Publisher. Link.
3. Author. Year. Title. Publisher. Link.

By signing this declaration, we affirm the authenticity and originality of our work.

Signature and date	*Signature and date*
Student 1 Name ID number	Student 2 Name ID number

[*Note to students: Do not submit documents where your signatures are easily accessible. Ideally, submit a flattened PDF to add a layer of security for your digital signatures*]

Appendix B. Example of Test Script Format

Class: MyClass						
Method	**#**	**Test Description**	**Sample Input Data**	**Expected Output**	**Actual Output**	**P/F**
isPositive	1	Determines that a positive whole number is positive	74	true	true	P
	2	Determines that a positive floating point number is positive	6.112	true	true	P
	3	Determines that a negative whole number is not positive	-871	false	false	P
	4	Determines that a negative floating point number is not positive	-0.0067	false	false	P
	5	Determines that 0 is not positive	0	false	false	P

Appendix C. Rubrics for MCO1

Total: 100 points

Criteria	Exemplary	Satisfactory	Developing	Beginning	None
[Prerequisite] **100%**					Late submission of deliverables. OR Part or all of deliverable is plagiarized or not a product of student's output. OR No significant contribution to the group output. OR Did not appear during the demo. **0**
Program Correctness and Completeness	Program executes without errors, and properly provides all the functionalities of the object-based implementation. **40**	Program executes without errors, but is missing some minor features. **30-35**	Program executes with minor errors, or is missing significant features. **20 - 25**	Program executes with minor errors and is missing significant features. **10 - 15**	Program does not run OR Program does not produce any output. **0**
Designing Classes/Objects	Design decisions comply completely with the specs and all classes, attributes, behaviors, and relationships identified and shown in the UML class diagram are logical. **20**	Design decisions comply completely with the specs as shown in the UML class diagram, but include unnecessary or redundant classes OR there is incomplete information on the attributes, behaviors, or relationships. **15**	The design provides for most but not all of the original specs as shown in the UML class diagram. Most likely, include unnecessary or redundant classes AND there is incomplete information on the attributes, behaviors, or relationships. **10**	The UML class diagram does not include most information based on the specs. **5**	No submitted UML class diagram. OR Design of classes are not in compliance to a logical object-based design. **0**
Consistency of design and code	Program implementation corresponds correctly with the presented Object-based design.	There are minor inconsistencies in design or implementation of attributes or methods, but all	There are minor inconsistencies in design or implementation of attributes or methods, which leads to missing or unexpected features.	There are significant inconsistencies between design and implementation at the class level, but most	No submitted class diagram to compare with.

	10	necessary features are still provided. **8**	**5**	features are still apparent. **3**	**0**
Program Readability and Documentation	Coding standards prescribed in the course is followed. AND In-line comments are included for long codes, apart from proper documentation of methods (inclusive of pre-conditions, post-conditions, method parameters, and return data) via Javadoc. **10**	Coding standards prescribed in the course is followed. AND No in-line comments are included for long codes. But, there is proper documentation of methods (inclusive of pre-conditions, post-conditions, method parameters, and return data) via Javadoc. **8**	Coding standards prescribed in the course is followed. AND No in-line comments are included for long codes. There are method documentation via Javadoc, but are missing some information. **5**	Coding standards prescribed in the course is followed. AND No in-line comments are included for long codes. AND Method documentation is not via Javadoc annotation. **3**	No internal documentation. **0**
Test Case Design and Documentation	Apart from getters and setters, all methods in all classes are tested with at least 3 documented unique test cases. **10**	All methods in all classes are tested, but not all have at least 3 documented unique test cases. **8**	Most methods in all classes are tested with at least 3 documented unique test cases. **5**	Many methods do not have at least 3 documented unique test cases. **3**	No test script submitted. **0**
Delivery and Presentation	All members are present in the video presentation and have relatively equal parts to present. All members exhibit mastery of their project throughout the demo. **10**	All members are present in the video presentation and have relatively equal parts to present. All members exhibit familiarity of their project throughout the demo. **8**	All members are present in the video presentation; however, some members have little contribution to the demo. Most members exhibit familiarity of their project throughout the demo. **5**	All members are present in the video presentation; however, some members have little contribution to the demo. Most members exhibit some knowledge of their project throughout the demo. **3**	Did not appear in the demo. OR Member did not exhibit ample knowledge about the design AND implementation of the project. **0**