Alison Weiss
Rebecca Tang
Claire Taylor
James Maeder

Functionality 20 pts

Your program **MUST** successfully compile and run without errors, but…
How robust is it? What happens if I sit on the keyboard while it's running? Does it respond to user inputs the way you say in your writeup? Are there graphical glitches? What happens if I give it a PDF file when it's expecting a txt file?

Our program is very functional. Our graphics are clear and all buttons and fields work as they should. We have error messages that pop up and explain why certain actions, such as trying to plant a new plant in a plot that already has a plant, are not allowed. Likewise, a pop-up appears because you cannot water or fertilize an empty plot. The program smoothly handles the end of the game: when you are out of money and all plots are empty, or when you reach day 30, a pop-up message says that the game is over and gives the option to start a new game from the beginning. If you try to restore a file that is not in the folder, then an error pops up that says "cannot find file" and it prints the IO exception in the terminal. If the file you try to restore exists but does not meet the required specifications because the user manually created the file for some reason, then there will be an exception printed in the terminal, and the game will keep running as is. You can save the file as any name you want, the user does not need to worry about selecting a file type. As long as users type in the correct name of a file they saved previously, it will restore to that set-up. Our keylistener and mouselistener work well and we were careful to make sure the requestFocus() was called at the necessary times for ease of use. For the animations of watering we made it so that they reset, so that you can water in a row and the graphics will not get tripped up.

For functionality we suggest the 18-20 point range.

Design: 20 pts

Points here are assigned based on your documents, not your code.
How did you break the problems down? Did you over-engineer solutions? Were you particularly clever?

Our project is well designed. Our structure of having Vegetable and Flower extend Plant, and then the individual types extend either Vegetable or Flower cut down on a lot of duplicate code. Each plant type shares many methods, so this design is clear and allows for each type to be held in the garden, which is a 2D array of Plants. We can call methods, such as draw() and grow() for all non-null elements in garden[][], since they are all plants.

Using a 2D array for the Garden was a clever idea. In Main, we keep track of an x and y coordinate, which correspond to a plot and a position in garden[][]. Xpos and Ypos are static variables. This made it much simpler to implement the mouseListener that allows for selecting different plots and associating that plot with the Plant object in the 2D array. It also made the keylistener for the wasd toggle function better streamlined.

One problem we encountered is we wanted flowers to only be able to be harvested once, but wanted vegetables to have the capacity for a few harvests. We solved this with an int field called harvestCounter which all Plants have. For flowers it is initialized to 1 in the constructor, but for Vegetables it is randomly set to between 2-5. This way Vegetables behave differently from Flowers, but internally they have the same consistent structure.

Our program slickly allows the user to name and save files mid game, and reload old games by simply typing in the name they gave it when it was saved.

Another clever idea we used was making changeMoney return a boolean. If you have enough money, it changes the money and returns true. If you do not have enough money it returns false. This cuts down on a lot of if statements when items are bought, since we can just call to change money which will check if we have enough to buy the item.

For design we suggest the 18-20 point range.

Creativity: 20 pts

This is an original project. While there are farming and gardening games, this one's functionality and design is unique. We did not base the program off of any other programs. All ideas were our own. Having the user select to go to the next day, and having that trigger growth is a creative flow of game play. The goal of this game is to make it to day 30 with as much money as possible, which is not how other gardening games we have seen are designed. We have randomized external events which kill plants overnight. We created our own graphics for each state for every plant. We edited our pop-ups to have our own drawn graphic icons. Our animation of watering is quite creative. WaterAll covers all of the board, while watering an individual plot just waters that plot. For the save game functionality we create text files with an original way to read all necessary information numerically to recreate the game state.
For creativity we suggest the 18-20 point range.

Sophistication: 20 pts

Your project should be reasonably sophisticated. While this is a subjective measure, a rule of thumb is that your submission should be more than 6 times the effort of doing an average lab
(you have 2 people spending 3 weeks on it).

Note that sophistication does not equate to number of lines of code. You should, in your writeup, convey where and how your final product is complex. Do you develop or use some algorithm? Is there a data structure doing the brunt of the labor? Are there many different pieces which all have to fit together well?

This program is highly sophisticated. We implemented many different types of functionality in our project. We have a customized mouseListener, keyListener, and save/file reading functionality. We learned how to animate still images for the water animation sequence. We had to use a timer to get it to run at the correct time and for the proper duration. Our main class synthesizes all of these elements into a cohesive game. When using the game, all elements seem natural and useful. We worked hard on our graphic design and button layout. We thought deeply about how to make the game the most enjoyable and user-friendly we could. The user interacts with the program in many different ways, all of which we had to carefully code.

For sophistication we suggest 18-20 points.

Broadness: 20 pts

To obtain full credit, you should use at least 6 of the following in some justified way.

1. Java libraries that we haven't seen in class
    a. We use many java libraries we have not seen in class, such as MouseListener, ActionListener, ImageIO, FileReader, BufferedReader
2. Subclassing (`extends` a class you created)
    a. Vegetable and Flower both extend the parent class Plant, and our plant types extend either Vegetable or Flower, all of which were classes we created
3. Built in data structures
    a. Garden[][] is a 2D array which holds the plants in our game
    b. When reading from an input file, we make each line into an element of an arrayList.
4. File input/output
    a. The save As functionality outputs a text file which can then be input back in through the restore functionality to save progress in a game.
5. Randomization
    a. When you fertilize a plant, you have a 50% chance of getting double the money when you harvest and sell.
    b. Each day, the plants have a random chance of dying from our creative external events. This is necessary for added excitement in the game.
    c. HarvestCounter for vegetables is randomly set between 2 and 5, which is how many times you can harvest the vegetable before it dies.

Since we only use 5 of the categories of broadness, we deserve 16.67 points for this category.

Code quality: 20 pts

There are two categories to this criteria:
Code Quality

- Commenting throughout
- Code organization across files
- Code cleanness

Product Quality:
Does your final product have the look and feel of semi-professional software?

We followed the commenting style of the lab projects and were consistent with formatting throughout our program. The code is well organized conceptually and functionally between our files. We were thoughtful about minimizing duplicate code. We also removed all unnecessary commented out code to keep the code clean. We followed standard naming conventions such that the names give a sense of the purpose. Our indentation is standardized and our code is easy to read.

We tested our project thoroughly to ensure that all elements work as they should. We added instructions to help the user know how to play the game. Our various pop-ups help to explain the progression of the game and lend it a semi-professional feel. We are happy with the semi-professional look of our graphics and layout.

We suggest a code quality grade of 18-20