Connor Mahern, Tony Dattolo, Nolan Cauley

Team 15 - Russet

Assignment 03 Petition

April 29th, 2021

We, Team 15 - Russet, are presenting our dissatisfaction with the code base provided by the previous Team 15 - Russet for Assignment 02. We believe that much of what was provided by Team 15 is unsatisfactory in regards to the requirements provided in the Assignment 02 details. Listed below is our formal complaints and dissatisfaction with the program in general, as well as a compiled list of all functioning, as well as non functioning classes and methods.
We have tried for days to try and fix things with no success. We are unsure if you want us to follow their implementation and have it not work, or to change things to make it work but then we get points off for not following their implementation. There are all kinds of weird classes that were not supposed to be included according to the chosen team's work that was meant to be a starting place. The view does not work properly. The cells are not painted properly, there is not enough of them, and it goes off the screen. It does not grow correctly and does not grow according to where the player is located. The growth function is not executed on the server. It’s a mess, and as of this time, we do not believe we can successfully complete assignment 3 with this code.

We believe that Team 15 did not do a satisfactory job of generating resources within the game space, their updateResources() only seems to update resources starting at the very middle block and work its way outward throughout the duration of the game. Therefore, it seems that Team 15 does not randomly or even manually select resource blocks on the board to start with any resources when the board is instantiated. Additionally, Team 15 use of the Decorator Design

Pattern seems to be affecting the way that they are generating resources throughout the time of the game as well. Team 15 uses an Abstract class Decorator and two subsequent Decorator classes, AliveDecorator and DeadDecorator, to set the status (Resource Amount) of the Node (Resource Block). The AliveDecorator Checks if the status of the Node is less than or equal to 7 and then increments it by 1, whereas the DeadDecorator, sets the status of the Node to 0 and reinstanciates the Node. This causes a conflict with the resource generation algorithm because they use setNode after using the provided generation algorithm, and pass in True to the boolean alvie argument of setNode. setNode() provides a check to see if alive is true, if so they decorate it with the AliveDecorator, meaning the DeadDecorator only used to instantiate all the resources on the board as dead, and if not necessarily meaninfully used. Additionally with their provided growth parameter of 0.1, when applying this to g * size of neighbors / 4 > Math.random(), this inherently makes the chance of the state to be true extremely low, meaning that most of the Node will not increment in their status (Resource Amount), causing the previously mentioned center outward expansion pattern. As well as an increased amount of time complexity leading to continuous glitching during update periods.

Furthermore, they misused the Data Structure Pair provided by Team 05's RMI interface. Instead opting to use the built-in Java Structure Point for the initialization of the HashMap<Integer, Point> Players where it should be HashMap<Integer, Pair> Players. Which is conflicting to the Controller, because the getPlayerPosition() function returns a HashMap<Integer, Pair> and then converts that Pair into a Point on the Controller side by putting it into another HashMap with <Integer, Point>. For our team this was extremely difficult to understand and figure out, and additionally works around the interface provided by Team 05

which we feel is discrepancy to the instructions of using the Team 05 RMI implementation to complete the project.

Finally, The view for the Forager Game itself is sloppy and poorly designed. The biggest issue is that our team was unable to change the size of the board to increase the playability of multiple players, due to the poor use of a size variable, rather just hard coding in size “10” through all portions of the program and many values being arbitrarily multiplied by 100. Additionally, it seems as if the player and the resources are somehow being moved/generated on separate grids, meaning that since we could not really figure out how they were determining the size, everytime we changed the movement amount or scale of the board that the player would be moving in between two squares. Since we feel that Assignment 03 has a heavy emphasis on the GUI elements, we feel that the extremely small board with large players is hard for us to manipulate in a meaningful way and adequately complete the requirements of the project. We tried for several hours to reorganize the program in a way that would make the board larger, but every time we moved anything there was either the problem of players moving into half squares or the resource generation would glitch the View out to the point that causes the program to crash or make the player movement incredibly laggy.

Classes:

Node

What it has:

- has a Point for location and an int for “status”, which we assume means resource level.
- Getters/Setters and a method to obtain neighbors.

What it doesn’t have:

Cell

What it has:

- A constructor.

What it doesn’t have:

- Really unsure why there is both a node and cell class. It looks like cell extends node, and then adds a location, but then the location only uses a super call within the constructor. Not sure what to make of this. It looks like node and call are both referenced sporadically.

Decorator

What it has:

- Abstract class with variables for Node, “status”, and method getStatus()

What it doesn’t have:

AliveDecorator

What it has:

- Constructor and getter for “status.”

What it doesn't have:

- We believe these are an improper use of the decorator pattern and is more suited to game of life than forager. We are unsure how to conform this to meet forager requirements as our own implementations are much different, using either observable or other patterns.

DeadDecorator

What it has:

- The same as AliveDecorator

What it doesn't have:

- Same issues as alive

ForagerModel

What it has:

- setNode() -> given a boolean and a Point it makes the Node at that location the opposite (alive or dead)
- getPlayerPositions() -> given an int the method puts all of the players in the hashmap into a new one but removes the player at the given int
- sendPlayerPosition() -> replaces or puts a player into the HashMap with the specified ID and Point

What it doesn't have:

- Did not use Pair at all as instructed; changed the interface
- Resource update is broken. It's unclear how it's working and would need to be completely refactored. However, with the current implementation of other classes, we're

not sure how to do that without making significant changes to the overall structure. Not clear if this is preferred or if you'd rather have an incorrect program.

ForagerInterface

What it has: - it was the same one given to everyone to start

What it doesn't have: N/A

ForagerController

What it has:

- Movement handling, albeit seemingly disconnected somehow to the growth of the player

What it doesn't have:

- constructor takes in foragerInterface for some reason, which is weird because it's supposed to be used by the model.
- There is no way to update player location, and this is broken as is.
- Cells are being added using the decorator classes, which as we have previously mentioned do not conform to the required implementation for a03.

ForagerView

What it has:

What it doesn't have:

- Board is not properly setup

- Cells are not properly setup
- Part of the growth may be happening here?
- Two instances of cells are launched at the same time, and they both go off screen
- Required minimum space has not been added