Allie Brock

Andrew Chitester

(2/3 of Team Flamingo)

CSCI-C322 Spring 2021

Lab 08 Documentation Changes & Justification For Those Changes

It should first be said that the UML diagram itself inherently needed to be changed as the connections, datatypes, etc. were different. As in, completely wrong and incoherent in terms of the diagram as designed. It is obvious we were unable to finish the diagram and were rushing to finish in time. To my understanding, we need a paragraph for every single one of these changes. But as a general paragraph: the way of documenting datatypes, parameters, etc. were incorrect and needed to be corrected. For example: instead of "keyPressed(KeyEvent e): void" we needed "keyPressed(e: KeyEvent): void." If I wrote a paragraph for every single one of these, this document would be longer than it already is and very, very repetitive.

The changes will be documented as such: *UML Diagram*, *Cell*, *View*, and *Controller*, with *UML Diagram* documenting changes made to the diagram itself (connections, etc.) and the rest being changes respective to their classes. Some changes in the UML Diagram section explain the changes within the classes, so the class sections are smaller. (Class changes include changes inside their class boxes on the diagram.)

UML DIAGRAM:

The connecting arrow between Controller and Timer was incorrect because the dashed arrow means implementation of an interface, and every Controller *has* a Timer, it does not

implement some interface called Timer. For this reason, we changed the arrow connecting Controller and Timer into a solid-line with a white diamond end that points to Controller to symbolize aggregation. Timer can exist independently of Controller, but it is incorporated into Controller.

The arrow between Observable and Controller has been turned solid because that relationship is one of inheritance and not implementation. Our reasoning for this is that the original diagram was just wrong. Observable is a class, not an interface, and so for Controller to be able to be an Observable, it must inherit from Observable.

The ActionListener interface was added so that Controller could implement it. This was for our original design goals: we wanted Controller to generate the next stage of life with each tick of the Timer. To be able to do that (simply), we had to use the resources provided by ActionListener to listen to the Timer so it can act when it needs to act.

We had to add the Observer interface to be able to implement the Observer design pattern. As the diagram was, we only had Observable, which was wholly wrong. View itself implements Observer now, for it to be able to actually observe anything and change as needed. In this instance, it observes Controller as Controller moves the game of life forward.

For another View change: It extends JPanel now, so the JFrame addition to the original document was removed entirely. JFrame is what you use when you want to make an actual window that displays something to your program; JPanel is more for things that go in that panel. Once again, there isn't much to explain here except for the fact that the diagram was wrong and we needed to fix it. Also, the dashed line between View and JPanel (now JFrame) has been changed into a solid line to accurately represent inheritance.

CELL:

Cell was the one class we did not need to change at all. Probably because it was a miniscule part of the actual program.

CONTROLLER:

Controller implements ActionListener now, which means it has to listen to an action and call the actionPerformed() function. As such, the function was added, and it listens to the internal timer of the class. For more information as to why ActionListener was needed, refer to that section of the UML Diagram notes.

VIEW:

Instead of the weird “Board” variable in View, we start with a startingState that is a 2D array of Cells, much like the ones being created/analyzed in Controller. This was done for two reasons. One: Board was incoherent and had no real standing as a variable. Two: the View generates the actual representation of the board, so it needs a state that it reads from. Keeping that in line with the states/”boards” of Controller keeps everything easier to access and understand.