Michael Deng

Team 9

Cellular Automata: Analysis

Project Journal:

Time Review:

The amount of time I roughly spend on this project was roughly 70 hours. This is the time I spent created the initial plan, thinking about code design, writing the actual code, debugging the problems I had along the way, and helping others debug their code.

During the first week, I spent roughly 3 hours planning out how our code was going to end up being designed with my teammates and roughly 2 hours writing the initial design plan with them. The first week of coding, I spent roughly 23 hours physically writing code and around 2 hours refactoring what I had. During this time, I wrote the user interface, the game loop, the grid, and I helped write the 4 models we had to implement. After the first week finished and we looked at the completed implementation, we realized that our design implementation was decently flawed and was inflexible to allow a lot of changes to the code. One quick example was that it was incredibly difficult to display different images and shape on the gird, due to the fact that the gridpane class was so inflexible to adapting to different shapes. Throughout the week, we spent nearly 5 hours together redesigning different parts of the code and I spent around 25 rewriting code, implementing new features, debugging my own errors, and refactoring/commenting my code. The additional 10 hours was spent help my teammates write and debug their parts of the code. One of my teammates had not worked with coding for a very long time and had forgotten a lot of syntax and logic, so I spent the majority of that 10 hours pair programming with him, where he would type and try to implement his code while I looked over his shoulder and pointed out errors in his syntax as well as minimizing his code. In the end we got the WaTorWorld simulation working, a program that I was not able to get working by myself. I was very proud of him. Besides that, I would always help him debug his code and fix whatever errors he came across that he did not have the experience to fix quickly himself. My other partner was a lot more experienced, but I also worked with him for several hours over the past few weeks implementing the XML parser with my code as well as helping him debug small parts of his parser. On average I spent roughly 5 hours of work a day on this project, but as the due dates approached, I definitely spent more time on the code then in the earlier stages of the project. The shape of the linechart measuring my work hours per day was indeed a positive slope from Saturday to Thursday night.

I had the easiest time implementing the user interface for my project. Although I did not complete everything I wanted to complete, I believe that Main.java, which is responsible for the majority of the user interface, was excellently written code and at the same time, I did not have to spend a lot of time doing it. From the beginning of the project, I knew how I wanted to design the user interface and populate the modules on the screen, so I was also able to write most of my code without having to refactor it later. I also did not run into a lot of errors while writing this part of the code, so debugging was incredibly easy. The hardest part of the code that I probably spent the majority of my time doing was writing and designing how the grid worked. Initially we would pass the entire grid to each individual cell and have the cell interact with the rest of the grid, but we changed our implementation so that each cell only works with his neighbors. I had to re-design and rewrite the grid for our project multiple time and I would always run into null pointer exceptions for forgetting to initialize things and out of bounds exceptions for forgetting to check boundaries of my matrix. Good uses of my time was helping my teammates debug their code and writing my own. Bad uses of my time included waiting for my teammates to show up and trying to integrate the XML parser with my code because we tried it too late in the game.

Teamwork:

I cannot say how long the other members spent on the project, but we probably spent around 40-50 hours together working on this project. This included roughly 6-8 hours of code design and writing our initial plan and roughly 40 hours of writing code together and helping each other debug and refactor.

I was in charge of creating the user interface, the application loop, and the grid that would interact and change all of the cells. Pranava's job was to create the XML parser, the XML files that would be read into the program, and the integration of the XML parsing with the rest of the program. David's job was to create the abstract cell and to create all of the different scenarios that created super classes of the cell. I ultimately had to write the majority of the different cell super classes and populate the cell abstract class as well because David had difficulty keeping up with the pace of the project. The general trend was that he would attempt each of the simulations to the best of his abilities and then I would fix them up, refactor them, and sometimes rewrite them into order to get them working properly. David and I pair programed for the WaTorWorld simulation, a program which helped him to grow a lot as a programmer.

In the first week we met a couple of times to both design our code implementation and to write the initial plan. We would them take the week end and each try to write our own individual parts of the code. We would them begin to meet over the weekdays in order to refactor, debug, and bring the different parts of our code together. Problems began to arise due to the fact that we were all consistently busy at times others of us were not, so we would never be able to meet for a long time before the last couple of days. This left use scrambling to get our code integrated with each other. We also should have spent more time actually thinking about the code instead of just writing it. In this way, we may have been able to predict the extensions and how to better design our code to implement the extensions beforehand.

To be frank, our code was not ready for the majority of the extensions in the complete implementation. We had to spend some time adopting a different implementation before we could even attempt the other extensions. Due to our initial code design and our unpreparedness in the first week, we spent a lot of time just cleaning up code and fixing old problems, so we did not have much time to implement anything new. I was stuck creating the extensions to the user interface and the grid, thus David did not have anyone to help him with the simulations and we couldn't get any of them done. What we did get done were some of the user interface and XML additions as well as toroidal and hexagonal capabilities.

We all decided that this project was simple enough and we all worked on such different things that we did not have to work on different branches. This turned out to be correct. The way I managed my code was I would put a list of things I had to get done on Trello, and then I would implement each of those items one by one, pushing the code to GitHub every time I completed a task. Because I did not work with the XML parser, I built in test functions that would run my simulation, so that I could debug and refactor without having to wait on the XML capabilities. Whenever I had to make very large changes, I would check out a new branch and do all my coding there, pulling from master every once in a while to make sure I was writing code consistent with the most updated version. I would them let my team members review if and if they were ok with it, I would merge the branch with master.

I felt that the communication between us was a bit lacking at times, but we were all very conscious of each other's time frames, schedules and ideas. We were all super busy and so it was very difficult to schedule meeting times. Not every showed up at the appropriate times either, making collaboration difficult. But what was great was that none of us ever got mad at each other and we were all super patient and understanding. On top of that, once we were all there, work got done and no one slacked off a lot. As a team we worked well together, but our schedules and other commitments always pressured us and kept us working late.

Commits:

I had a total of 52 commits and the average size of each of the pushes, considering I had roughly 5,500 insertions and 2,500 deletions, was around 150 lines of code per commit. I can quite easily say that my amount of commits, insertions, and deletions are quite representative of my contribution to the project. I have roughly 5 times the number of insertions and deletions as the next largest contributor of code. I was the one who wrote the most amount of the code, not only because my part was one of the more code dense jobs, but because I effectively had to help David with his role in the project. The majority of the application, besides the XML capabilities, was all written, debugged and refactored by me. At the same time I was usually also responsible for organizing the times when we met and helping the other two debug their own code. I also lead the group in designing our code and planning whatever implementation we decided on.

In David's defense, the reason he does not have any apparent commits on his github is because, for some reason, his account is not set up properly to record his work. The record of commits definitely shows he has committed, but for some reason it does not show up in his graphical progress.

Three commits I had were "completed new neighbor implementation with toroidal capabilites", "added edit grid functionality", and "finished percentageCalc, cellMover needs work". The purpose of the first commit was to say that I had finished giving our newly implemented grid design the ability to be toroidal, where cells can wrap around the grid. The purpose of the second commit to say I gave our grid the capability to edit individual cell colors when the game is paused. The third commit referenced two methods in our segregationCell.java, and I said I finished one of the classes but did not finished the second.

I tried very hard to make sure that I did not give my other teammates problems with my commits and pushes. My two big rules about using github is one must always pull and fix merge conflicts before pushing and pushing broken code is not allowed. Because of these two rules, all of us were able to avoid created unnecessary amounts of merge conflicts. Some of my bigger commits where I sometimes completely redesign some parts of the implementation caused many conflicts, but I always helped my other two teammates resolve those conflicts. The fact that we all worked on different parts of the code also helped minimize conflicts.

Everytime I finished a big section of my own code, I would commit and push my work. It was was very important to me that my teammates always have my most recent code so they would not have to do any additional work figuring out what I had done. Thus, I would always push several times a day whenever I worked on the project. Whenever I had to make big changes that I knew would break a lot of other code implementations, I would always check out a branch and do work there so to not mess up my teammates work.

Conclusion:

We definitely underestimated the amount of work this project would need and thus, we were coding until the last second and yet did not get as much done as we would have liked. Our biggest down fall was that we did not take the project too seriously in the beginning and thus we were left with incomplete code later on. I believe what we underestimate the most was the amount of time it would take to write complete and correctly logic for some of the simulations. I know for a fact, WaTorWorld took us 15+ hours to figure out because the rules for the simulation written on the website did not produce results we wanted. In the future, I'm not going to make assumptions on how long the project is going to take. I'm just going to work until its done and them relax afterward.

I definitely felt that I took enough responsibility for my team. I felt that I was in more of a leadership role because I was the one who coordinated with everyone else to set up meeting times and I was ultimately the one would decide what everyone had to do. I also took on the role of a mentor, and helped one of teammates complete his role in the assignment. Every time I did something to add to or change the code, I would either text them or Facebook message them to let them know what's going on. This way, they knew exactly where I was so they would have to writing extra code for deprecated designs or implementations.

I believe the code that required the most amount of editing was the grid class. This is because the grid class is the one class that links everything together. Whenever a change was made to the user interface, XML capabilities, or the cell classes themselves, a change or addition would always have to be made to the grid class. This kept me constantly working, trying to adapt the class to take on new additions. I also spent quite some time refactoring the class in order to allow it to better take one different implementations and simulations.

As a designer, my biggest flaw is that I too frequently just jump into coding without first taking the time to sit down and think first. In the beginning I was impatient to start, so our initial code design was inflexible and un-modular. I need to stop jumping the gun and actually sit down and write my design out on paper and think critically first. This will help me not only better design solutions that adapt to many different scenarios, but it will also help me write my code faster because I'll know what I want to do. One of my greatest strengths is that I think modularly. When I write code, I think of the code in sections, which allows me to write code that implements a lot of small, useful methods. I am also very conscious of repetition and tried to avoid it as much as possible.

I thought my best traits as a team mate was that I was always very understanding of other people and willing to accept other opinions. When David was having trouble with his code, I would always help him and try to fix his problems, earning both his respect and his determination to continue working. I also am very willing to hear what other people have to say, so when Pranava suggested a new grid implementation, I was willing to try it out and because of that, we now have much more understandable adaptable code. My biggest flaw was that I was a little too nice and laid back sometimes. I would not keep up with the progress of my teammates and would not push them to work harder, and thus they did much less work that what I expected from them. They also left things to the last minute to do, giving all of us much unneeded stress.

If I could improve one thing in my own code to get a better grade, it would be my methods for finding a cells neighbors. My code was way to repetitive and used way too many if statements, two things which are fundamental problems with professor Duvall.

Design Review:

Status:

Throughout the project, my one coding style was very consistent with itself. I always tried to be very consistent with naming my methods with useful names, keeping my variables camel case, keeping my final static constants all uppercase, and all my methods commented.

I believe that in general my code is very readable and easy to understand. Most of my methods are named very descriptively and each of my larger methods are very shy, only calling other methods that are named well while at the same time not being cluttered with more logic. Even for my more logic based methods, they are much easier to understand. I personally feel that the majority of my methods are written in a format easy to follow and understand. Besides few block if trees and some in line comments that we forget to take out after testing, my code should be quite easy to read.

I tried very hard this time to use less global variables and less getters and setters. In my first project, each page had many global variables and the majority of those variables were passed from one class to the next using a multitude of setters and getters in each method. I have managed for this project to cut that number down to roughly 3 to 5 global variables per class and one only getter and one setter outside the cell super class. The way I dealt with constants and variables that have to span several classes was I put them all in one class that was created solely to store those commonly used globals. I am not sure if that was the correct approach to use, but I found that my code suddenly became a lot more readable because random variables suddenly had a name attributed to them, and it because very easy to change parameters instead of searching through the different classes to make all the changes I wanted to make. For example, changing the size of the grid while simulation testing became a breeze in the park because all I had to do was changes two variables in my constants class.

The majority of the additional features were not too terrible to extend and implement. One big thing I did not like about my code though was how I decided to handle additional implemented features in my code. What I ended up doing was creating more and more methods in each one of my classes for the different implementations, and then it was up to the XML parser to decide what method to use. For example, I had a bunch of different methods in my grid class that find a bunch of different neighbors. Such methods included finding only 4 neighbors, finding 8, finding used toroidal capabilities, and finding hexagonal neighbors. However, in order to call the correct method, the XML parser had to use a giant if tree and cycle through all the options to find the right method to use. I believe a much better design would have instead been to create a new, separate method for finding all the neighbors and then having the grid class call that class in order to find a neighbor. This would have made my code more modular, my grid class easier to understand, and my code generally more shy.

Some parts of my code are easier to test then others. Even though I believe the majority of my functions are useful and modular, they sometimes input and output variables that are difficult to replicate. Particular examples are certain methods in my cell subclasses, where I don’t even return variable some times. I personally believe that some of my cell subclasses would be more difficult to test without running the entire simulation and searching for a result. Yet there are some parts of my code that are much easier to test. One good example are my find neighbors method, which always take the same input and spit out different variations of the same output. One could easily write a test that passes in a cell and compares the returned hash map to a hardcoded value.

I did not find many bugs in Pranava code for a couple of big reasons. One reason was that I did not spend much time reviewing what he did because I trusted him to get his functionalities working and the other reason what that I had not experience with his code or the syntax behind XML parsing. I saw physical errors when we ran the simulation, but very rarely did I find big mistakes in his code. On the other hand, I saw many mistakes in David's code. Because I worked with him so much and basically taught him to be a better coder from the group up, I was privy to every single missed double equal sign and failed initialization and out of bounds error as I looked over his shoulder. However, I was able to point out his mistakes quickly and he became much better at coding in a very short amount of time.

I learned a bit about parsing and data streams by reading some of Pranava's code, but the majority of my time was spend helping out David and attempting to make my own code better.

In order to make my code more clear and maintainable, there are several things I can do. One of the short term solutions would simply be to try to remove the getters, setters, and global variables in my code. Another thing I could do is try to remove the larger if trees in my code. Yet the big long term technique I need to work on is make my code more modular through the use of more classes and inheritance. I am still in the phase of programming where I like to have all functionality in big classes, and I understand that is not the right way to go. What I need to do is abstract some common functionality away and make use of subclasses more, as well as make classes for basic functionalities that have many different variations, such as finding neighbors.

Design:

Our code was ultimately designed to function as a cohesive unit with four different parts: The graphics, the XML parsing, the grid manipulation, and the individual cells themselves. We wrote our code this way so that the interactions between each of the 4 parts would be as minimal as possible with very little interaction with each other's logic. This way changes could be made and their relationship to each other would not be affected. The XML parser would be responsible for reading in certain information and then attributing that information to certain methods in the other classes. My Main classes was my main graphics classes and was responsible for populating the stage with the modules I had created, such as buttons, a slider, and a graph. My application loop class was used to run the simulated with an on-click event was fired and was also used to connect on-click events to my grid. The application loop's main task was to help the graphics and cell matrix work together without intruding directly into each other's territories. My grid class, is reasonable for cycling through every single one of my cells and calling update on them. The grid is responsible for initializing the cells and the cell matrix, as well as populated the characteristics of each cell and giving the cell matrix to the grid pane to be displayed. Each of the cell sub classes were used to manipulate themselves based off of their neighbors. Each cell with be referenced by its position and given a list of its neighbors. The cell's job would be to look through that list of neighbors and decide how to change based off its surroundings.

In order to add a new simulation, all one has to do is add a new cell subclass to the package. All this subclass has to do is call the find neighbors method inherent to all cell subclasses and then write logic to manipulate itself based off of the array of neighbors and their states. We tried very hard to separate the graphics from the grid from each of the cells. In this way, if a change or addition needs to be made to one area, the others are not affected.

One big discussion we had as a team was deciding how the grid and cells would relate to each other. In the very beginning we were torn between giving the cells control over the grid or giving all the power to the grid and only letting the cells change themselves. If we let the grid control the cells, then the grid would have to do much more work changing the cells based off of the cell's neighbors. Each individual cell's rules would not help that either. At the same time it's not proper practice to give each individual cell control over the entire matrix and passing the matrix back and forth between the cells and the grid could have caused may unforeseen problems. Giving total control the cells over the grid would have allow the grid class be to incredibly shy because all the grid would have to do is call update, pass the grid the cell, and let the cell do all the work. Where this becomes dangerous is when the cells can start manipulating too much in the matrix, thus changing the performance of other cells and effectively making the solution incorrect. We chose to go with cell having the entire matrix because we felt that it keep our code more modular and shy and we were decently confident that we could keep the grid safe from individual cells.

Another big discussion we had was during the last week of the project, where we question the appropriateness of the interaction between the cell and the matrix. Pranava brought up an excellent point that instead of passing in all the entire matrix, it would be easier to just pass in a map of that particular cell's neighbors. We reasoned that this would not only protect our grid and not take any functionality away from our cell, but also that it would allow use to implement more changes and additions to the code without have to change the cells. For example, when we wanted to implement different neighbor checking, we only had to add a method to the grid without have to do anything to any of the cell classes, which made life so much easier.

As I said before I was in charge of the graphics, the application loop, and the grid that interacted with the cells. I did a lot of work with each of the cell classes as well, but I will focus on the three classes listed above. My Main class is responsible for the creation of each scene and the population of modules on each of the scenes. It deals with my three user pages: the splash page, the main game page, and the error page. For each page, it creates buttons, labels, sliders, and graphs. It also gives functionality to each of these modules which then call methods in other classes. My application loop class was responsible for running the application when the start button was pressed and updating the grid per frame. The application loop initializes both the scene for game and the gridpane that displays the cell matrix. Whenever a button is clicked that edits the gridpane, the application loop handles it and returns it to Main. If grid is the thing being edited, the application loop passes the command over to the grid class. The application loop also handle a lot of the interaction with the XML parser and passes the command over to the grid if need be. The grid class is what is responsible for manipulating each of the individual cells in a cell matrix and them passing the cell matrix back over to the application loop to. Whenever the application loop wants to initialize the cell matrix, it passes the command over to the grid to do so. The grid also has the responsibility of populated predefined characteristics of each of the cells such as the neighbors of each cell.

For my finding neighbor's methods in my grid class, I designed it so that there were several methods to call from in order to find a variety of different neighbor combinations. The XML parser would then chose which method to call based off of the user input. Each method has a couple of similarities in that that all check for bounds and populate a map based off of a pre-defined array of points. So what each method does is call upon the check bounds method and populate map method, while passing in an array of points to check that vary from one method to the next. Furthermore some of my methods feed off of other methods to avoid repetitive code. For example, when square neighbors (8 in total) need to be found instead of cardinal direction neighbors (4 in total), the square neighbor method calls the cardinal neighbors method as well as a corner neighbor method. And when toroidal capabilities are turned on, the toroidal methods first call their normal counter parts and then recheck the neighbors in order to find any toroidal exceptions. In this way I tried to both keep my code as minimal as possible and provide a lot of options for the code to implement.

The reason I designed the code the way I did was mainly to correct the mistakes I had made in the previous project, which included not isolated each of the classes from one other and having repetitive code. I made the choice to attempt to isolate each of the parts from other another so that not so many global variables would have to be used and less getters and setters would have to be implemented. And even though I may not have done the best job, I tried very hard to avoid repeated code so each of my classes have a lot more smaller, modular methods to call from. I also tried very hard to keep my code as shy as possible so many of my larger methods are just used to call other piece of logic. By designing my code this way, I depend on other people to design their code to specifically take my inputs and give back the outputs I need.

Ideal Design:

The original design we had was not the most ideal design to handle all of the implementations for a couple of reasons. We used a gridpane instead of a canvas to display our cell grid and because of that, we were not able to implement graphical hexagons, triangles, pictures, etc. We were able to easily implement the logic for hexagons quite easily, but the display is clearly not the best. The way our cells related to our grid, it would have been relatively easy to implement other scenarios. Added graphics additions to the code was a bit of a hassle because it had to span several class, so that as well was not optimal. I do not know too much about how the XML functionalities work, but I do know that many changes had to be made to my existing logic, so that was, once again, not optimal.

My ideal design would be to have a program that almost completely separates the graphics from the cell matrix and the cell subclasses. Where there is as little overlap between the three sections I mentioned as possible, that is when the code is I believe the best, because they is when you can add or change or delete anything in any section and it won't affect the others. My ideal implementation would not need to pass any variables from one class to the next. It would also give no control of any other cells in the matrix to any one cell. My ideal implementation would have better XML capabilities. It would also have more classes for a variety of different purposes instead of a few big classes with many methods. I would like to have used inheritance a little more as well.

In order to implement new simulations, I would do the same thing as in my current design.

My ideal design for this project would have been quite similar to what I current have, with more modulation and less repetitive code. First of all, I would have less to no global variables as well as no getter nor setters. All variable shared across multiple class would somehow find their way into the constants class. Instead of using a gridpane, I would use a canvas instead to allow for a variety of different shapes and styling. I would also try to give my cell super class more attributes so that each of the cell subclasses would not have to have as many methods as they do. I also did not like the amount of double for loops I used in my code and would have like to reduce the number I used and have one double for loop do many things. I also don't like I how o so statically positioned all the modules in my user interface. If I could have redone it I may have used a vBox instead or at least have relatively positioned all of my modules based off of the dimensions of the stage. I also wish I had paid more attention to how Pranava handled his XML because I felt that his code to integrate XML capabilities with my classes took away from the shyness, modularity, and correctness of my code.

Even though my ideal and current designs have many things that are different, they also have a lot in common as well. The general design is exact what I wanted it to be, where different sections were separated and did not depend on one another to work. I may not have done the best job is code that as effectively as I could have, but the main idea and logic is there. I also wanted to avoid giving the cells as little control over others as possible, which I believe I succeeded in doing well by only passing them a map of their neighbors. And even though my code still uses getters and setters and global variables, I believe I did a much better job to eliminate them than the last project.

Masterpiece:

For my masterpiece, I chose to rewrite how my code finds neighbors. I felt that the code did not belong in my grid class and that there were to many if trees in a couple of the methods. I decided to move all of the neighbor finding logic to a new class called NeighborFinder, which was created solely to find the neighbors pertaining to a certain cell given certain parameters. I added a few more classes that actually minimized my logic and got rid of all the large unnecessary if tree. Now, not only is my code for finding neighbors, more module and more shy, but there are big blocks of the code that are difficult to read and understand.

When testing my masterpiece, I would want to test that the logic for finding neighbors is actually finding the right cells as neighbors. Thus I would want to test the methods that actually calculate when the point is out of bounds and calculate the x and y values based off of where the current cell is.

I provided three tests that test three different functions, two of the functions do the same thing with different amounts of inputs. One tests checks to see where or not my check bounds method was working correctly. The other two run a method that returns a y value given a y value. This second method works in conjunction with toroidal to return correct y values depending on where the current cell is located and where it wants to move. I would really just test to make sure that my toroidal capabilities work because the logic for that is not intuitive.