Cell Society Project Analysis

Wenjun Mao

Wm56

Time review

It is quite hard to calculate the total hours on the project, but approximately 50 hours.

Start date: 9/14

Complete date: 9/26

Overall about 5 hours per day on average but may vary for each implement period.

Most time were spent on coding new features, since I haven't utilized xml with Java previously so the XML parser took most of the time for the first week's implementation.

And about 20%time planning and designing and 20%debugging or refactoring.

Coding the specific cell class and patch classes are easy since if we are familiar with the cell behavior and finished the inheritance for the abstract class and helper methods, all we need to do is to call needed methods and update the cell states.

A really bad point about time utilization happens during the first week, since I am a windows guy and windows does not have built-in merge tools or diff tool for GIT, so I have to spend a lot of time search in eclipse to resolve conflicts, and sometimes my teammate changes 3 lines and I got 100 merge conflicts.

This problem is resolved when I later tried the sourceTree and kDiff.

Teamwork

I actually feel kind of bad since I got another project and midterm so I missed the last two group meeting, but the overall the team worked pretty well in my view. And all the members contributed to the project.

We are planned to meet every night in Link and the responsible parts are listed below:

I think the role we split worked well since Davis was responsible for the overall game loop control and UI, so those two classes can interact better without any possible conflicts. What I needed to do is to parse the XML and generate a grid for them for further process. And as planned we created the cell abstract together and extended with each simulation individually so avoid possible conflicts.

Since the complete guide was not given until the second week, so we initially started with GridPane and Rectangle for Grid representation, then in order to extend the triangle and hexagon feature we have to give up those and calculate each cell blocks individually.

But overall our project have a pretty good extendibility.

Just as described above, since our responsibility split quite well so our code did not have too much conflicts. I commits and pushes each time I think my code is going to have possible conflicts with others.

I think the XML configuration part of our XML input can still be improved to handle more complicated simulations. Although I am aware that the current InfoSheet class is really ugly and has lots of setter and getters, I have not yet think of a better and simpler solution so I just kept that design.

The communication is just right since we tells each other when we have to do changes to their parts or adding anything that varies from our basic design.

Commits

I have a total of 88 commits.

The commits in the first week is kind of messy since I was not quite familiar with the GIT GUI and it auto generated a lot of merge commits without other comments.

Most of my other commits are merges new features or update my local branch with master and resolve conflicts.

I did not commit too much to master, what I did was push to my own branch and merge once I get any significant changes or possible conflicts.

I think this way of commit is clearer than pushing each time we got a trivia change to files.

I will take the example of my implementation for color picker:

My first commit is when I added the color picker class and the color chooser is functioning but can't change the cell color. This is only adding one new class so no conflicts occurred.

Then I created a commit called "modified rest of the cell types to enable color changing", which shows that I made changes to the subclasses of individual simulations.

At last since we have to allow modification of background colors, I made that change and committed as "enable color picker for background".

I don't have much debugging commits, mostly are new features and refactoring.

Conclusions

I think for some part I underestimated the workload. It took me quite long time to design and implement the XML parser since I don't have much previous experiences.

I think I have done planned works but I could have done more since we can all see that Davis have taken most of the works.

Since we mostly worked together, I think the communication is done pretty well. But for the last two days before the due date, what I felt is that since I was implementing a single simulation which does not interfere with the rest part of the code so we might have a little less communication during that period.

The code requires editing is the XML input format and the related InfoSheet class. Since as mentioned our XML has lots of tags and is not too flexible to more simulations added, and the InfoSheet class, although is pretty simple and easy to utilize, appeared to be messy and hard to extend.

We have to think ahead the extendibility and the design. I think we did pretty well this time.

Just as mentioned in this section, it should be the XML input and output.

Design Review

Status

The project overall follows the naming conventions and it has clearly separated GUI, processing and control classes.

The readability is good for most parts, since the naming of the classes and methods are self-explanatory and the overall hierarchy is strictly following our original design.

We don't have any global variables in the project and all the info of the grid is passed in a configuration class called InfoSheet. We did use some getter methods in that though.

Most of the implement features are not hard to extend based on our original design except the XML config saving since our XML format and the parsing is not good enough.

The XML input and grid generation can be easily tested but the functioning of the classes are harder to test.

I haven't found any bugs in their codes committed. - -.

Since I wasn't using gameLoop for controlling in my first project, I have learnt a lot in the MainController class for the whole program control.

I think the most crucial thing for the code to be readable is not the comments, it should be all classes have its own clear functionality and each methods' names are clear and explains what it does. So based on this standard I think most part of our project is readable.

Design

Our program is divided in two packages: controller and the simulation objects.

Our controller has a MainController class which controls the whole program flow with key handlers and calls UI to display the whole GUI.

The parser parses the input file and saves it in the format of an InfoSheet and a List of strings for each grid row.

Our grid class performs grid operations such as find neighbors, count cells or calls the update for the patches and move patches around.

The Rules class controls the rule operations.

The PatchFactory is used to create Patches.

And the rest classes ColorPicker and the SimulationChart are just implementing required features.

For the SimulationObjects package contains cell hierarchy, patch hierarchy and the patch body hierarchy.

Adding a new simulation requires a new Cell subclass and a possible Patch class depends on how complicated the rule of the simulation is.

In cell subclass that extends Cell, we need to implement the override methods of initial setting and update rules.

For most cases the Patch class does not need implementation and just using original Patch class. But for complicated simulations such as sugarscape that needs unique update operations or settings we need to extend patch and override with its own rules.

For the parts I designed,

The parser class called by MainController takes in a XML file selected by user and parse the config and the grid layout and process them to prepare for the creation of the simulation grid.

In the MainController, a method that takes the List of grid rows and the config info and create the grid. And the error handling based on the implement guide is also implemented in MainController, which I think could be refactored out to make the code more neat.

And two Cell subclasses, ForestCell and SugarCell.

Forest Cell Class is really easy to implement since the rules are simple and the cells instead of moving around the grid just changes its state and stay stationary.

For the SugarCell and the Sugar Patch class.

The hardest Part is what this simulation differs from others.

It has to take the parameter value to determine the color that filled the Patch.

This also involves in the Color Picking class since we need to update the colors as user changed them in the color Picker.

For the XML parsing and the infoSheet design part, I was intended to redesign the input XML format to support more complicated parameter inputs, and those changes requires the modification of the data storage structure, and I have been working on that for a few days but it turned out to lead to a kind of messy code of the parser and makes it even harder to access data from other classes.

The current parser used a simpler input with fewer tags or parameters and being able to support most of the provided simulation cases so that's why I chose the current design.

Ideal design

Our original design is able to handle most of the new feature requirements, with little change in Patch Bodies.

The feature of the Infinite Boundary is accessible with our current design although we haven't finished that part.

The major design is that once we have to extend the grid towards negative axis, we can move the whole grid towards opposite direction thus changes all the X,Y coordinates for the grid Patches.

But in my view, the optimum way is just keep a starting point (which kind of close to the head in the Linked List), and calculates and puts other Patches solely bases on calculation instead of giving each of the Patches a specific X,Y value.

Since the interaction mostly occurs between neighbors, this design should be better for most cases.

Adding simulation is similar to our current design with only one difference.

Since the ideal design does not contain coordinates, so the update methods and searching for new block methods should be slightly different from the original design.

Code Masterpiece

The code part I selected was Sugarscape Cell class.

Since this class represents all the designs for our cell hierarchy.

Test names are kind of self-explanatory so I won't list all of them over here.

I think I have already done refactoring before submitting the original project so I will leave the whole class as my masterpiece part.