Construction Experience

Project Name: RobotGo

Team: CMPT370-B3

Team Member: Cheng, Gong
Hounjet, Carlin
Lan, Shaoxiong
Xie, Joey
Yue, YiRui

Date: Nov 27, 2016

Table of Contents

Part 1: Pair Programming Experiences

The content of each pair programming experience includes three major parts: what's been achieved, problems and solution, and the learned skills.

1.1 Pair programming experiences of Carlin Hounjet

For my first pair programming session, Jason and I worked together to begin work on the robot language interpreter. Since this was our first time, we had no end game planned we just wanted to work as hard as we could for two hours and see what we accomplished, though I did have some expectations about finishing multiple methods. This turned out to be a positive experience despite us only finishing one method. Having two people made it more fun and productive, as the navigator could google technical questions (for example, how to handle casting data-types to the abstract class of Object in our Stack) and the added pressure of having another person working with you prevented all types of procrastination. It also helped to explain concepts that only I understood beforehand to make sure I wasn't flying high on my own hubris/insanity (it happens sometimes). Overall it was productive but also showed me that I needed to improve my ability to step back from problems and see the bigger picture instead of doing trial and error type of debugging. I also learned that I needed to relinquish control of the keyboard sooner as I lost track of time and spent most of the time as the pilot.

For my second session I worked with Joey on the model classes. This one felt highly productive but also slightly less fulfilling as the model classes required little to no thought and the most basic of tests. We finished and tested all of them within the two hours we spent together, this time with Joey being the pilot for the majority of the time. I felt that I helped less here as he had designed the models and knew them better than me. I did help a couple times, like when we were getting strange errors and having Joey explain the problem to me made the solution clear. Or when I helped remove some errors by making changed in the constructor. Mostly I helped by just reading off things from the design document and googling any questions Joey had. Doing this also helped me understand the model's relationship with the rest of the system I during subsequent meetings I was able to understand Yurui's explanation of his progress and help with troubles he was having.

For my third session I worked with Joey on the interpreter since we felt that working on the model wasn't a totally effective use of our skills and teamwork.

This session proved the effectiveness of pair programming to a much higher standard, as we were constantly swapping out keyboard control and covering for each other's mistakes. We rarely hit snags, but when we did it would just take a moment of discussion and testing to understand it and fix it. We also overcame many roadblocks that had to do with converting and handling conflicting data-types easily due to one person always being free to google while the other person made progress while trusting the other to find the solution before long. In the two hour session we created multiple functions and had a solid understanding of how to work with the stack and handle all necessary data-types.

The fourth pair programming session was with Joey again, making further progress on the Interpreter. This time we were handing the more dense and complicated functions of logical comparisons and loops. This session showed a flaw in my design of the interpreter as we struggled to implement my interpretation of how loops are handled, as well as a flaw in my part of the testing document that gave us the exact opposite of what we were expecting for the <> comparison. With Joey's help we quickly fixed the latter error, but spent probably a little too long trying to understand loops. We did get partway in and after we had called it a night, our furthered understanding of how we should handle it allowed me to come up with a new design. Another overall success that saved me a lot of potential frustration.

1.2 Pair programming experiences of Gong Cheng

Carlin and I worked on the forth interpreter class for a few sessions and the overall result is satisfactory. Compared with programming independently, it certainly added more fun and peer pressure which definitely prevented our minds from wandering. But I think it actually slowed things down a bit because the forth interpret part is what Carlin is familiar with and he spent a lot of time explaining how things should work to me. It did minimize the mistakes though, since sometimes we will point out some mistakes (for example String.equal() not ==) which helped avoid potential unnecessary future painful debugging. And also we could offer suggestions for each other like places we should use Stack.peek() instead of Stack.pop() and that kind of thing. Because this is the first time I did pair programming, it was not professional and there is a lot to improve for me. For example, I should talk more while coding to keep my navigator engaged.

Shaoxiong Lan and I worked on the GUI part together for about four hours in total. We enabled “Number of players” radio button, “Play” and “End” button in GUI, GUI_Initial Class. First of all, we changed the the GUI class main function to static regular method so the play Button ActionListener in GUI_Initial Class can invoke it. After that we enabled the end button by putting a dispose JFrame function in End button action listener. We ran into some bugs causing the static attribute in DrawingPanel Class to be changed, which we figured out together through discussion. Once that was complete, we used a loop in DrawingPanel to implement the number of Players radio button. So overall, we did a very efficient pair programming session that resulted in us learning a lot from each other. Because GUI is something we both understand fairly well, it made the pair programming go more smoothly. I think one of the main reasons why pair programming benefited us a lot is that we often found each other approaching a problem from very different perspectives, which enhanced our programming logic and problem solving ability.

1.3 Pair programming experiences of Joey Xie

In my first pair programming session I worked with Carlin on the model for about 2 hours. This part of our system was fairly simple as it mostly consisted of accessors and mutator methods but there were some complications when we tried to use maps and dictionary data structures. When I tried to test one of the objects I was creating instances of a completely different object which obviously led to errors. I only noticed the problem when I was explaining the problem to Carlin so this pair programming session was pretty helpful. In this session we were able to finish almost all of our model classes as specified by our design document.

In my second pair programming session I worked with Carlin on the interpreter, mainly with interpreting mathematical operations. This session was pretty productive as we took turns coding and thinking, usually when I am programming alone I would have to think ,then code, then get stuck on some small error that I can't find. We got confused a lot when we were pushing and putting things on stacks since sometimes we would put things in the wrong order if we weren't paying attention but we were able to sort things out by working together. This session we were able to finish all of the code that was related to interpreting mathematical operations and some common forth words.

In my third pair programming seesion I worked with Yirui for about 2 hours. We had to make the model and the controller compatible again becuase the other group members altered the design a bit. I worked on the model and Yirui worked on the controller so it made sense to work in a pair. We discussed a lot about the differences between what we had from our design and what we had now and we made the necessary changes.

In my fourth pair programming session I worked with Carlin on the interpreter again. The interpreter was already mostly complete, only the most complicated functions remained. We took turns coding everytime someone was stumped and I feel we were maybe at least three times as efficient as we would be if we were programming alone. There was a part where Carlin was stuck when he was testing a condition in an if then statement where if 1 1 <> it would print equal if it were true and not equal if it were false. The function was printing not equal which was wrong but I pointed out to him that he was using the <> operator and not the = operator so he would have to switch the true and false branches. He admitted that I probably saved him a couple of hours so this session was definitely helpful.
In this session we were able to finish the code interpreting if,else and then. We were also able to make some headway into interpreting loops.

1.4 Pair programming experiences of Shaoxiong Lan

Initial game interface (paired with Gong Cheng)
I paired with Gong Cheng on the initial interface part for total of about four hours. We constructed the class "GUI_initial" with radio button called "Number of players" to select the number of players and another radio button group to set the AI /Human mode for each team. For the "play" button, we added the action to open and switch to the Main Game Interface after setting the number and the mode of players. We also added the thumbnail of game board in the initial interface to show the robots deployment and team location based on users' selection to enhance the user experience. The difficulty we met in the pair programming session is how to change robot number and deployment on the real main game board and how to return to the initial interface and restart the game. We still took turns to complete the milestones one by one every 40 minutes. Gong helped a lot on the switch between the two interfaces. Compared to the main game board, the process of pair programming on initial interface went more smoothly.

Main game interface (paired with Yirui Yue):
I paired with Yirui Yue twice to construct and upgrade the GUI of game board. We have two pair programming sessions in about 6 hours.
During the GUI construction session, based on the tutorial online of hexagonal grids and a series of mathematic proof, we built the class called "hexmech_pointy" to get the single pointy-topped hexagon, which we would use as the cell in the game board. Then, based on the "hexmech_pointy" class, we built the game board class called "drawingPanel" to arrange the cells with offset coordinates. Adding the mouse click listener in this class is very helpful to get the coordinates of a mouse click and transform it to the hexagon coordinates. Then we built the GUI class to embed the drawingPanel class. In the GUI class, we also have the built-in class "table" to keep track of the data of current robot's turn. The 'start' button is to begin the game. Then 'end' button is to return to the initial interface. The 'switch' button is to end the current turn and begin the next turn.
During the GUI upgrading session, we set the milestones to complete one by one based on the basic function of GUI part, which included adding a new function or fixing the existing bugs. Firstly, we would brainstorm together to make sure the milestone is doable before constructing. We then took turns after completing these milestones, such as adding popup buttons or fixing overlapping bug. Pair programming is an efficient and helpful method to make two person work together. When I was the driver, I would focus on the implementation of a method. When I am the navigator, I would follow the YIrui Yue's implementation and try to find the potential bugs and boundary conditions. Until now, we still have some bugs on

the GUI, such as the incomplete merging sight range of robots in same team. It's one of the milestone we will complete as soon as possible. Although It took more time to discuss, it's more efficient and quicker to fix the bugs when we are stuck.

1.5 Pair programming experiences of Yirui Yue

I paired with Shaoxiong Lan twice to construct the main game interface. We spent in total 6 hours for the two pair programming sessions. First, we built a class called hexmech_pointy to draw a single regular hexagon, pointy topped. Then we built a class called drawingPanel which will use the class above to draw the entire game board filled with pointy hexagons. Lastly, together we decided the order that the robots will be stored in the robot list should be based on the types of robots. In this way, we solved the problem of which robot should be play in a specific turn. The major problem we met was how to convert a mouse click to a pointy topped hexagon coordinate. The online research helped us on understanding how to solve the case with a flatted topped hexagon. Then we took the math mechanism behind the online research result and succeeded in converting a mouse click to a pointy topped hexagon coordinate. I learned that the code contains less bugs with someone sitting next to you while programming and that the code will be more clear and more concise with more than just one person focusing on the same code. What's more, while solving a difficult tricky problem, it's more efficient to work with pairs.

I also paired with Joey Xie once for a total of two hours with the goal of modifying the robot model to help establish and build the connection between the controller and model. We also added two new attributes. The first was an attribute to record the robot's coordinate on the gameboard and the second was an attribute to store the forth code passed in from the robot library. Accordingly, we changed the part of the controller involved in using the robot model class for updating the robot list which stores the robot information. The small problem we had was that we built a redundant sub class inside the robot model class to record the robot coordinate, while there was already a sub class existing inside the interface class to finish that part of the work. The problem was pointed out by the team mate who focuses on the interface part. Different people have different coding styles, what I have learned from Joey is to plan first, then write the code with clear thought and logic. This way it will be more likely to produce bug-free code.

Part2 Code Review Summary

For our code review we decided to take an in depth analysis and review of the Robot Language Interpreter. We chose this because it was the hardest to understand and design conceptually, it was mainly designed by two of the members, and the class contained errors and it was helpful to work on them as a group. Because it was difficult to understand and design without implementing anything, it is necessary to review and ensure that what we had in mind is what we are ending up with. We also want to review it so everyone in the group can understand the processes and logic of this class since this was designed by a small portion of our group and it interacts with the other parts of our system. Once this was done, we could work together as a group to help solve a problem that was happening with our loop functionality.

The positives we have noticed in this class begin with the solid logic, the ease with which it can be understood, and the way it follows our design from previous documents. Most of the functionality has been implemented, well tested, and the logic has been reviewed by all group members, finding little to no problems. Another positive noted by many group members is that the methods are short and easy to understand. The case statements pf Play makes it easier to debug because they make searching for the functionality of FORTH words easy to understand at a glance. Tests are placed in a logical order from unit tests to functionality tests, as is outlined in our testing document. The structure and functionality of the methods follow our design document closely. Overall, we have concluded that the class is on the right track and we're happy with the progress so far.

The negatives we noticed during code review mainly had to do with formatting, documentation and some missing methods. Some of the variable names in this class do not currently adhere to our style guide. However, this has been done for ease of programming and can easily be changed with a simple “find/replace” of these variables. We also noticed a couple places with redundant code that can be optimized if time allows. The next problem is that this class is currently missing JavaDoc class and method headers, which will be added before the final version is complete. The last problem is that there are still some functionalities that have not been implemented and therefor could not be reviewed. These problems are all fixable and should not present any overwhelming difficulties (hopefully!).

What we spent the most time discussing was what the stacks and vague variables names represented. This got a little confusing when getting into the more dense

functions such as conditionals and nested loops. Overall, the logic of most functions were well understood and did not require a lot of discussion, despite the majority of the group spending little time on the design of this class. We are happy with the way we are implementing this class. As far as we understand, it is fulfilling what was intended in our design. We discussed the expected challenges (loops and conditionals), the parts we found surprisingly easy (arithmetic, comparisons, and tokenizing the code) as well as the unexpected difficulty of understanding where errors came up while using a stack. We all acknowledge that the design was helpful, as implementing all this from scratch would have been nearly impossible. Overall, we felt the process so far and this code review were positive and productive uses of group time.

Our plan for improvement is to continue utilizing pair programming and individual work to make the decided upon changes to the class. This will continue to be done by the main programmers despite the group all now understanding the class better. To be more specific about the main function we need to implement, we need to finish construction of the code that handles the nested loops now that we understand and have discussed the problem in depth. There is also the tasks of interpreting strings as outlined in the robot language document, the task of retrieving robot variables from the model, and the final step of connecting the interpreter to the main controller. These should only take a few more days of work at the pace we are going at.

Part 3 Changes made to previous design

3.1 Discussion of Construction activities.

The deign document certainly helped us a lot in the implementation stage. It serves as guideline for construction but still we find that a lot changes need to be made. On the one hand, some classes we included in the design document is unnecessary and on the other hand, we have to add some helper classes that we didn't think we need.
During the construction, the main tricky parts are GUI and Interpreter parts.
The GUI took a lot more time than we expected. We need to take account of every detail of every step and every situation the robots and users would face. What's more, the shading and the amount of robots certainly added a lot of difficulties to GUI class so a big chunk of time was spent debugging the GUI related Classes.

3.2 Changes Made between Design and the Implementation Stage

1)
Main change is we combined the initialization controller with the main controller. For the limited time, we find it will consume too much time if we use two controllers and also the initialization controller is too simple to make it independent.

2)
In GUI, we added hexmech_pointy Class as a GUI helper class to construct the single hexagon.
Based on the tutorial (http://www.redblobgames.com/grids/hexagons/) , we choose to construct the pointy-topped hexagon.

3)
we added DrawingPanel Class as another GUI helper class to arrange all the single hexagons to form the game board based on the offset coordinate. Because it makes it easier to understand. It would be too much for one class if realize those functionalities in it. Meanwhile, we can reuse it as the custom component to reduce the redundancy.

4) We need to add a string variable into the robot model class to store the FORTH code for the robots. This also necessitates change to the initialization where we download the code from the JSON file and store it in each robot.

5) We need to add a new property called “position” into the robot model class. The type is Point to store the current position in the game board based on the game board coordination.

6) We need to build mailboxes when we build the robots. Mail is for AI only. We also need three mailboxes for each robot instead of one.

7) We changed functionality of Read() and play slightly to adapt to newfound problems we missed during design.

8) We added extended functionality to the GUI due to the ease of use found in windows maker.

9) We changed how the interpreter handles loops, with added functions and two new stacks that handle nested loops and the storage of commands inside a loop for the next iteration.