Construction Experience Document

Brady Lang

Dylan Mcinnes

Parker Reese

Meagan Renneberg

Experience with Pair Programming
LEGEND: (**Author** *with partner***)**

Parker *with Meagan:*

Meagan and I started the BoardPosition entity in our system, one we both agreed would be good to work on together. I co-piloted first. Together we discussed and figured out some of the details we had neglected in our design document, such as variables for holding location and direction of our Robots. Being able to discuss this with another team member made conclusions come quicker. We also created a standard for the coordinate system of our board. Each hex has an (x,y) coordinate, but it took a couple trials to derive a system that has a pattern, consistent for each coordinate, to get to neighbouring coordinates. For example, my initial coordinate system was not consistent in what you did to the x and y values if you were to move down to the right. It was only after a brief oscillation of ideas that we found one that worked. Without being able to build off one another, a solution would have taken longer. I was also able – as a co-pilot - to correct errors such as spelling mistakes in comments, and could give a quick reminder on how to make a constructor. Both of us being a little out of practice with writing Java, being able to quickly talk through a line, without looking it up, increased efficiency.

When I was the pilot I was able to ascertain that my code was being edited as I went due to my consistent talking out loud with what I was thinking and typing. Meagan was able to do quick edits on the fly, something that has made going back to edit that code a redundancy. When making the tests for one of the methods in the BoardPosition I couldn't remember the proper syntax for a try and catch statement. Meagan was able to assist me with this. In the end, I felt more secure about the code I had written, for it was already edited and accepted by another peer.

Since Meagan and I are friends outside of class, I was met with the issue of staying focused. On several occasions, I got us off topic by introducing an irrelevant conversation starter. To continue pair programming in the future I first need to practice staying on topic while amongst friends.

Parker *with Brady:*

With Brady, I intentionally tried not to start any conversations that had nothing to do with the project. Something I learned to do from my first pair programming. It may also have helped that this pair programing was at the University, instead of a household.

Pair programming was a similar experience with Brady as it had been with Meagan. Syntax errors, spelling mistakes, and other small points were quickly rectified on the go. The creation of our Library Socket stubs as well as a mock JSON file went swiftly. Being able to talk about how our Library Socket was going to work right then and there was a benefit as it was quite some time since I last talked about our Robot Library Socket.

Something I did not have with Brady, but had with Meagan, was an extra monitor screen. In this instance, I would have liked one to duplicate what was on Brady's laptop as the angle on which I was viewing made it difficult to read the code. In future pair programming, I will be sure to have an extra monitor available to make the pair programming be more comfortable.

Dylan *with Meagan*:

In the paired programming session with Meagan, we added tokenizing functionality to the Forth Interpreter. I felt overall it was beneficial, as it kept me on task and focused. We worked fairly well together, and having someone pointing out things I needed to do during coding was useful. Explaining my thought processes while coding helped me keep track of what made sense and what did not, and prevented a few "d'oh" moments.

Dylan *with Brady*:

In the paired programming session with Brady, we added most of the functionality of NpcController. In it, we ended up finding some problems with the rest of the project. We changed some constructors and variables in the Robot defininiton, as well as changed the visibility of some methods in BoardPosition. I thought this was useful, as it increased both mine and Brady's understanding both of what we needed to do and what we were doing. It was also helpful to see how the ForthInterpreter interacted with our other classes.

Brady *with Parker*:

The first paired programming session I shared with Parker. We accomplished a few tasks and created an action map going forward. We shared a challenge that we reflected on for sessions moving forward. Firstly, the session was more productive than a solo coding session. We were able to exchange ideas, correct syntax and steer each other from distraction of other elements within the code. Secondly, code quality was higher. We kept adjusting with conformance to the style guide much easier as a group instead of treating the style guide as an afterthought. Things I had forgotten in are our chosen style guide were pointed out by Parker and vice-versa. This allowed us to be more honest in conformance to the style guide going forward. Also, programming out loud and bouncing ideas off one another solved a lot of errors before they had arisen, allowing for distraction free programming.

In our session we created method stubs for the RobotSocketLibrary, created a mock JSON file for testing, researched JSON integration, clarified gang and team definitions as described in the document and changed Robot id from INT to STRING to adhere to JSON "name." We identified a path going forward to create tests of RobotSocketLibrary and to start expressing the method stubs we had started. Also, creating a mock JSON file allowed us to start implementing JSON tests in the RobotSocketLibrary. A challenge that was noted in our session was a lack of goal setting for the session. We started juggling a few ideas, worked a little bit on each idea. We feel our efficiency would have increased if we had clearer end-goals in mind for the session.

Brady *with Dylan*:

The second paired programming session was shared with Dylan. We accomplished one very large task, identified future objectives and changed a few elements of previously written code to conform more closely to the original UML diagram as described in the Design Document. We created NpcController, and made changes to the Robot and RobotSocketLibrary classes. We also identified work needing to be done in the Team class and set TODO's within the code to assist team members with what elements need to be finished or changed in the near future.

Using the knowledge gained from my previous paired programming sessions, we first started the session with setting a specific end goal to work towards before anything had been done. We considered working on the Views in our system or the NpcController. Ultimately settling on the NpcController. This was doubly nice since Dylan had put a lot of individual time into the ForthInterpretter, which our NpcController works with very closely. Dylan was able to explain and share knowledge of the Forth Interpreter to me, helping me understand a component of the system I was less familiar with. This allows me to be more useful in other ForthInterpretter interactions within the system going forward and gave me a greater understanding of our system. We also made plenty of changes to previously programmed work in Robot and RobotSocketLibrary classes to adhere closer to the original design document which also unified the ForthInterpretter with the rest of the system.

Meagan *with Parker:*

For my first pair programming session I worked with Parker on BoardPosition which is a model entity in our system. It was a different experience for me, I was used to working around other people but they were always working on their own projects rather than working on the exact same one that I was. Instead of me bothering the person beside me for help and taking them away from their project I had someone right beside me watching everything I was doing and constantly offering criticism and having me explain what I was doing. I found this quite beneficial to my work ethic, not only was I more conscious of what I was doing but it also challenged me to support the decisions I made and have a strong argument for why I made these decisions. The main goal we accomplished during our programming session was figuring out a coordinates system for the game board. We both had different ideas about how we wanted the coordinates to be set, I wanted to have the centre of the board be (0,0) and have an even patter with negative numbers while Parker wanted to have one of the corners be (0,0) and have the entire outside of the board be (0,i) coordinates. We both discussed and defended our ideas until we came to the conclusion that a combination of our ideas would work best. His idea to start at (0,0) on a corner and my idea to have a consistent pattern worked together such that we were able to have our coordinate system mathematically work when figuring out the methods to turn and advance a player. Overall I found this experience valuable, I was more focused on the current task I was working on and I did not feel like I was bothering my partner when asking question because we were working together on the same thing.

Meagan *with Dylan:*

My second pair programming session I worked with Dylan on the Interpreter. This was more challenging than my first pair programming session since I did not have as strong of a grasp on how the interpreter worked. We mostly worked on a method that tokenizes input which I understood how it should work but was unsure of how to represent that in a code format. Once we started the method it became more clear the more I question why he did a certain thing. I think it was beneficial to both of us that we had to explain not only what we were doing but also why we were doing it that way. When the method working with the tests that had first been constructed Dylan did not just move onto the next part. He modified his tests in ways such that it seemed like he was trying to get the method to fail. It made me realize that when I tested some of my methods how I may not have tested them as aggressively as I should have. From just simply watching someone rewrite testing code, it gave me a new perspective on how I should be more diligent with the tests I design.

Code Review of the Forth Interpreter

LEGEND: <u>Class,</u> *methodName***.**

We first reviewed the code for the <u>ForthInterpreter</u>. We reviewed all the methods and tests in the <u>ForthInterpreter</u>. We read through the methods in the order they appeared and found in the first method *splitElse* that we needed to expand on the comments in the function because we were having trouble following what is was doing. To make future viewing of the method *spiltElse* easier we decided to add more detailed comments. The question about whether we needed to add a test for *splitElse* when there was an if statement that didn't have an else arose. After discussing if it was needed we decided that it was not needed because the exception that catches a malformed script would handle that. Some changes that needed to be done were minor changes such as fixing the method *exec*'s parameter to be correctly labeled as "token" when it had been mislabeled in the Javadoc comment as "word". It was also found that there were some TODO statements remaining that should have been removed when it was finished. These minor changes were not any trouble to fix and benefited us because our code was more readable with the proper comments. When reviewing the *push* method the question of whether the method should throw an exception if the stack ran out of space arose but it was decided that the test was not needed. Overall when reviewing the code we thought it was well done and tested efficiently. The only test that was decided to add was a test that would give an abnormal script and test that it would throw the appropriate exception. From reviewing the code and making small changes our code not only become easier to read but we also improved it so that it could handle more input even when the input may not be valid.

Code Review of the Board Position

LEGEND: Class, *methodName*.

The second code review we performed was the BoardPosition class. BoardPosition is a large, slightly complicated entity with many dependencies on it's functionality. When initially reviewing the class we saw that it lacked a javadoc description and needs to be updated.We also identified that most comments in the BoardPosition use the format for Javadoc comments when they were just regular comments. We decided to fix this so that the comments properly reflect the format described in our style guide. Next, we identified the helper methods *addOrientationToCurrent*, *addOrientationToCurrentReturn* and *subtractOrientationFromNum* were declared as either public or protected methods. As the helper methods are only used internally, we have changed these helper methods to be private.

Continuing through the methods, we found *immediateDirection*, *distanceTo* and *advance* are using an else statement for printing an error message. This could be covered better with throw statements, and the throw statements will need to be tested. The *addOrientationToCurrent*, *addOrientationToCurrentReturn* and *subtractOrientationFromNum* methods should have throw statements to ensure values are within the allowed range of zero to five. Using hardcoded tests would allow us to test the throw statements. We decided to create variables for magic numbers used in *addOrientationToCurrent*, *addOrientationToCurrentReturn* and *subtractOrientationFromNum.* The magic number of 5 will be changed to be MAX_ROTATION = 5. When then decided to remove the tests for valid positions in the *advance* method. The function *isValid* in GameBoard checks this so it is not necessary to test advancing into an invalid position in *advance*. For further testing as well in the tests for *setCoordinate*, we decided to create more tests using direction is facing one or two to cover all cases.

When reviewing code for the *DirectionTo* method, we found that we could use clone to reduce redundancy within the code and simplify the code. *DirectionTo* is a non-trivial solution, more internal comments to describe the function are necessary. Another issue we found was when reviewing the code for *getRelative*, we discovered the code was flawed and non functional. The method *getRelative* and its respective tests have been flagged for redesign and review. As we continued to review and critique BoardPosition, we decided to remove the coordinates class since it was simply holding two ints. We have moved the two ints describing the X,Y position into RobotController to reduce redundancy.

Things We Have Changed

LEGEND: Class, *methodName*, **'variable'**

In the class Robot we removed the ***'variables'*** variable because it will work better if it is stored in the ForthInterpreter.The *messages* are now stored as LiteralWord(s), indexed by the sender. The *messages* need to sorted by sender which is required by the robot language. And Send now takes in a LiteralWord instead of String to accommodate the change to *messages*.

In the class BoardPosition we added a variable '**orientationFromRight'** integer to BoardPosition. It is used to keep track of rotation, which will be a number 0-5, with 0 always being to the east.

In the class ForthInterpreter the variable ***'words'*** is now a map with string keys because they need to be accessible by their name. We added the method exec so that it can now take in a Script to execute rather than making a ForthInterpreter constructor that takes in two Scripts. This allows us to run more than one script without creating multiple interpreters.

In the class Script it no longer has a public ***'words'*** variable, our Script has no words and is token delimited. It only holds the indexes for the words classes (which are strings).

In the class GameBoard two more variables were added to GameBoard. First an integer ***'size'*** that is the size of the gameboard, the getter and setter methods were present but there was not a variable for storing the size. The other variable added is a Map called ***'Statistics'***, this map when given a string that was the robots identifier would lead to another map when given a string the was the statistic would lead to the value of that statistic. This was made so that there would be a place to store the statistics when updating the immediate statistics during the game.
The method getAllRobots was changed to have the return type of LinkedList rather than List. It made testing easier and adding lists of robots to a team became simpler.

Code Construction

Our code construction can be found under the tag CONSTRUCTION in our repository.