Group D4 Programming Document

Team D4

Nickolas Gough, Scott Hebert, Janelle Hindman, Yige Huang, and Tushita Patel

Project: Robo-Wars

Dr. Christopher Dutchyn

CMPT 370 Term I 2016/2017

Contents

[1.0 Pair Programming Experiences 3](#_Toc468027198)

[1.1 Janelle and Yige 3](#_Toc468027199)

[1.1.1 Janelle’s View 3](#_Toc468027200)

[1.1.2 Yige’s View 3](#_Toc468027201)

[1.2 Nick and Tushita 3](#_Toc468027202)

[1.2.1 Tushita’s View 3](#_Toc468027203)

[1.2.2 Nick’s View 4](#_Toc468027204)

[1.3 Tushita and Janelle 4](#_Toc468027205)

[1.3.1 Janelle’s View 5](#_Toc468027206)

[1.3.2 Tushita’s View 5](#_Toc468027207)

[1.4 Yige and Scott 5](#_Toc468027208)

[1.4.1 Scott’s View 5](#_Toc468027209)

[1.4.2 Yige’s View 6](#_Toc468027210)

[1.5 Scott and Nick 6](#_Toc468027211)

[1.5.1 Scott’s View 6](#_Toc468027212)

[1.5.2 Nick’s View 6](#_Toc468027213)

[2.0 Code Review Session Summary 7](#_Toc468027214)

[2.1 First Lesson Learned: Greater Shared Understanding of Code 7](#_Toc468027215)

[2.2 Second Lesson Learned: Very Efficient in terms of Energy 7](#_Toc468027216)

[2.3 Third Lesson Learned: Importance of Documentation 7](#_Toc468027217)

[2.4 Fourth Lesson Learned: Style and Syntax Importance 8](#_Toc468027218)

[2.5 Group Dynamics 8](#_Toc468027219)

[3.0 Reflection on Code Construction 8](#_Toc468027220)

[3.1 Time Required 8](#_Toc468027221)

[3.2 The Role of Design 8](#_Toc468027222)

[3.3 Integration Concerns 8](#_Toc468027223)

[3.4 Possible Changes 9](#_Toc468027224)

[4.0 Changes to the Design 9](#_Toc468027225)

[5.0 Final Thoughts 9](#_Toc468027226)

# Pair Programming Experiences

As part of the code construction, each of the group members worked together with another group member in a pair programming experience. This experience was repeated, so that a total of 10 pair programming sessions occurred. Below, broken down by session, is each person’s view of the session and its outcome.

## 1.1 Janelle and Yige

Janelle and Yige had their session on Friday, November 18th.

### 1.1.1 Janelle’s View

On Friday, November 18, Yige and I had a pair programming session where we set out to complete the remaining methods in our Interpreter component. The design and testing of the Interpreter was primarily my duty, so we selected these methods as a sort of "bus insurance" (someone else needed to understand the code clearly, in case I got hit by a bus). Yige understood the code quickly, and as I coded, she helped me catch some important errors that I hadn't considered before. In our session, she helped us focus on testing, and made some important adjustments that increased the number of passing tests from 6 to 220.

If both of us had had more time, it would have been nice for both of us to sit down and discuss details of the code for even more time before we even touched the keyboard. We caught some important mistakes, but might have made even more progress if we had caught them earlier.

### 1.1.2 Yige’s View

Before we started to do pair programming, we decided to do elements of the if-condition, for-loop and while-loop for the interpreter. At the beginning, Janelle informed me of some necessary information on scratch paper to ensure we were on the same page about what we were about to do. We started to code and talked about if-condition first, with our design document open on another monitor. Janelle wrote some comments based on what elements the if-condition needed and we both agreed on, and then coded after each comment, so the logic was quite clear. After coding if-condition, we tried to test if it worked since we were doing test driven development and tests were already done by Janelle before. After fixing the String comparison syntax, tests were working well, and this was the first time I saw people using regular expression to do find and replace in Eclipse, which was quite impressive! Later, we switched roles and I did another method in a similar way. Janelle really could hold such a huge java file all in her head, and I learnt a lot on different ways of thinking and coding. If we had more time, it would be nice to spend more time to do pair programming.

## 1.2 Nick and Tushita

Nick and Tushita had their pair programming session on Monday, November 21st.

### 1.2.1 Tushita’s View

My pair programming experience with Nickolas was both positive and educational. The session was initially expected to be for two to three hours and we were expected to be working on partially the game board, as the board was designed by Nick in the design stage but I am implementing it, so his input would be helpful. However, we ended up majorly working on the mathematical side of the Display unit, because we thought that my strengths in mathematics could make major contributions to the Display unit. One of the benefits of working with Nick was that because we are friends, we were both very comfortable talking to each other and critiquing the code, which meant fewer distractions and tangents and more powerful focus on the code. Moreover, my ‘forte’ (comparatively) in mathematics, and his knowledge in GUI allowed us to be more productive. Nick is also very prone to making silly mistakes, and my presence helped in pointing out the mistakes right away, which prevented major investments in debugging time.

Because this was the first time to experience pair programming for both of us, there certainly are possible areas of improvement. We are both not used to being vocal about what we are thinking, and often we would find ourselves not communicating our thoughts, which would leave the partner clueless. I had to remind Nick several times to explain how he was approaching a problem, and he had to do the same for me too. It is hard, although beneficial, to be expressive of our thinking process and I shall definitely learn from that and try to be vocal in the future. One of the other things that I didn’t like about the experience was that pair programming can be very tiresome, because it is an intense thinking session. During the session, we are working on two parts of our brain – the one that requires logical thinking and the one that requires good social behaviour. After two hours into the session, I felt very tired and wanted to take a nap! Therefore, in the next programming session with Janelle, I will make sure that I schedule it towards the end of the day to save myself from sluggishness for the rest of the day.

### 1.2.2 Nick’s View

The pair programming session involving Tushita and I focused on drawing the game board on the screen. The initial progress was slow, since neither of us have much prior experience with developing game screens. However, with both of our minds set on the task, the problem was solved within three hours or so. The productivity of the session was greater than my normal individual sessions, which is likely for two reasons. First, I am more experienced in graphical user interface development than Tushita is, but Tushita's mathematical skills are superior to my own, both of which were necessary for coding the drawing of the hexagonal grid of the game board to the screen. There were instances during the session when Tushita did not know how to proceed because she was not aware of the constructs provided by Java that would allow her to complete the task, which is when I was able to be of assistance to Tushita. Likewise, there were instances when I could not proceed myself because I couldn't resolve the math required to draw the hexagonal grid, which is where Tushita assisted me. Secondly, the second pair of eyes and mind reduced the number of mistakes and increased the code produced. Tushita caught many of my common mistakes, which likely decreased the amount of time I would have spent debugging my own code. Also, since Tushita is more familiar with some of the other components within the system than I am, and I am more familiar with other components in the system than she is, it was easy to determine how the game screen would need to be implemented without spending too much time referring to the design document. Overall, two pairs of eyes and minds increased the level of productivity since each of us possessed a different set of skills, were able to share thoughts and knowledge with each other, and were able to spot mistakes and flaws quicker.

## 1.3 Tushita and Janelle

Tushita and Janelle had their pair programming session on Thursday, November 24th.

### 1.3.1 Janelle’s View

For our pair programming session, Tushita and I worked on integrating some components with the Board code, in preparation for the team’s code review of the game logic. We handled communication between the Logger and the Board, so that the Board would report the correct events to the Logger with the correct scope. Working with Tushita on this code was very rewarding. She knows a lot more of the features of Eclipse than I do, and I learned all kinds of tricks and keyboard shortcuts by watching her code. The two of us were able to catch a lot of logical errors as we coded, rather than them going unnoticed and coming back to haunt us later.

In our session, we were able to tackle a lot of small problems in a short period of time, but it would have been more satisfying and potentially more productive for us (in value to man-hours spent) to grapple with a larger problem together. However, at this stage in the code construction there were not as many large tasks to handle, so we couldn’t take full advantage of the pair programming technique for deep work.

### 1.3.2 Tushita’s View

My second pair-programming session was with Janelle. This time, I made sure that we scheduled the session towards the end of the day to avoid feeling tired and worn out for the rest of the day. Working with Janelle was overall a great experience. She is very quick and intelligent, so she picked out many of my mistakes when I was typing the code. Although we got a little distracted for about ten minutes in the midst of our session due to the ongoing construction besides Spinks, we actually found that we were very productive by the end. I could not have produced the same amount of code by myself in many hours. Because she is very experienced with integration of games, we not only finished majority of the Board class (which is central to the system), I also got to learn a lot of tips about integration from her! This time, I focused on vocalizing what I was thinking, after learning from my past session with Nick. I believe (and hope) that I did a very good job in communicating what I was thinking. I am pretty happy with how I performed in the session, I learned a lot – in terms of techniques and communication – and felt confident about my produced code.

## 1.4 Yige and Scott

Yige and Scott had their Pair Programming session on Thursday, November 24th.

### 1.4.1 Scott’s View

Yige and I did our Pair Programming section mainly on the Flow component and related elements. I found it very interesting that, while Yige’s favored programming language is Python and mine is Java/Scala—about as different as you can be and still be within the C family—we had many similarities and greatly complementary styles. We both preferred to work on reducing Eclipse error messages and had very similar ‘thinking styles’ which sped our work. We greatly reduced the number of compiler errors and it felt like a very productive two hours.

I learned a great deal from Yige. She could very clearly articulate her thoughts and concerns; I felt very slow and clumsy in my coding compared to her. Coming from a non-programming background as I do, I have been quite concerned that my code construction skills were not up to the standard that the group desired. However, Yige never even hinted that this was an issue and she made me feel very much at ease and a contributing member of the group.

### 1.4.2 Yige’s View

Since flow was an important component and it was not finished yet, we decided to do part of it. We started by writing needed but non-existent methods like isValidMovement(). We wrote some helper methods for it and cleaned up some related methods by calling other methods to remove duplicate code. Later, we decided to get rid of errors since it was also important. We reduced Eclipse-complaining errors from 23 to 5 and other errors by sharing our knowledge: some were great logic mistakes, some were weird such as a statement ending with a period. We switched roles once in the middle. I was quite pleased coding with Scott. Scott’s coding style was similar to mine, and he had very clear logic, one of the best naming skills I have ever seen, and the ability to write very clean and understandable code. I happened to know a little more about Java built-in methods, Eclipse and vim, so our knowledge was complementary and the whole session went smoothly on communication and code. There were lot of things to learn from Scott, especially the variable and method naming, and the logic.

## 1.5 Scott and Nick

Scott and Nick had their Pair Programming session on Wednesday, November 23rd.

### 1.5.1 Scott’s View

We dealt mostly with the Flow component, with some overlap with the Board and the Display. We noted that the Flow component seemed underpowered, and wrote several methods to increase its ability to actually control the Flow of the game. It lasted just over two hours, and went by very quickly.

Nick was an excellent complement to my normal approach to code construction. He has a highly disciplined, methodical approach to code construction, while mine tends to be higher energy but more erratic. Given that I was the one typing the code, it felt very much as if he were directing me without bossing me, and the combination was highly successful. After doing both pair programming sessions, I hope to both do more for this project and also to use the technique more with my employees for company projects.

### 1.5.2 Nick’s View

The pair programming session involving Scott and I focused on implementing the Flow component within the system. Although the session produced a fair amount of code, the majority of the session was spent exposing and correcting some of the inherent flaws of the design of the flow component, since we were unable to produce the code we needed to perform many of the necessary tasks. Similar to my pair programming session with Tushita, I possessed more experience developing graphical user interfaces using Java than Scott and Scott's understanding of how the Flow component would be required to interact with the other components of the system both enabled us to produce more code than either of us could alone. At the beginning of the session, I could tell that Scott's experience with Java events is limited, but, partially due to my experience with Java events and Scott's ability to quickly learn, Scott was quickly able to quickly overcome this challenge. The majority of the flaws of the Flow component were exposed when one of us would ask the other how some piece of code should be written to correctly perform some task, and when neither of us were able to produce an answer, both of us began producing ideas and our individual understanding of the other components within the system to more quickly resolve the issue. Also, it was interesting to see how Scott would approach a problem by speaking to me about it. Scott would recognize a problem within the design and voice it or ask me a question, but he would almost immediately produce the answer himself, only to have me confirm it. I realized that perhaps speaking with others about challenges or problems while coding may actually lead to less time spent on a problem, as I normally spend much time trying to discover the solution myself. Finally, Scott made it clear that he preferred lists over arrays for multiple reasons, and even made changes to the code of the Flow component involving arrays to instead use lists because using lists reduced the amount of code written significantly. Again, this made me realize that stepping outside of my comfort zone of usual practices may result in improved results. Overall, the session discovered some problems and was enlightening.

# 2.0 Code Review Session Summary

After the pair programming sessions were complete, a section of code was chosen for an in-depth code review. The Board component was selected because it is the central component of the game from an architectural standpoint. If the Board component does not work, the game will not function properly. The session lasted over two (2) hours, and the results were useful if challenging. Several large logic errors were found, which is very useful, but it increased concern over the code base as a whole. As a result, more Code Reviews have been planned. The Flow component will be reviewed in depth, and the Interpreter component may also get a smaller review.

## 2.1 First Lesson Learned: Greater Shared Understanding of Code

The first lesson learned was that the level of knowledge about the Board’s actual design was less well-known than expected. Everyone asked at one time or another about the intent or execution of one or another field or method. As a result, the entire team felt that a better grasp and shared understanding of the Board component was reached as a result of the code review.

This led to a side discussion after the code review as to whether it would have been beneficial to do pair design session during that phase of the project. While the architecture of the game was decided largely at the group level, individual Classes of the components were detailed by individual group members. This led to the above issue with a lack of shared understanding.

## 2.2 Second Lesson Learned: Very Efficient in terms of Energy

Another lesson learned by the group was that code reviews by a group were less fatiguing to individuals when compared to individual reviews or even pair programming sessions. This allowed for more code to be reviewed in the same time frame. It may not be as efficient in terms of lines of code reviewed per group member, but it was not as inefficient in that area as might be expected, and it was certainly more efficient in terms of fatigue cost per group member. This is not a small concern in large projects.

## 2.3 Third Lesson Learned: Importance of Documentation

The next lesson is that documentation of code is of vital importance in a large group project. While the group has a style guide, the level of compliance to that guide varies greatly from one group member to another, and even with the same person. Every group member had comprehension issues multiple times during the code review, and many of them revolved around insufficient, missing, or misleading documentation. This may seem obvious, as code documentation is a constant refrain in computer science education. It was not the need for it that was so surprising but the degree to which it is needed.

## 2.4 Fourth Lesson Learned: Style and Syntax Importance

The fourth lesson dealt with stylistic and syntactical differences among the group members. Several issues related to this came up during the code review, and among them were whether exceptions should be thrown and when, maintaining component focus (e.g., keeping all user interfaces in Display), errors from state mutation, the need for modularity and to avoid ‘magic numbers’, and the like.

The largest discussion in this area dealt with when inputs should be validated. Several methods in the Board Class dealt with checking whether or not a Move or Shoot action was valid. This discussion hinged on whether this should occur in the Board Class or in some Class in the Flow Component.

## 2.5 Group Dynamics

The final lesson learned by the group was the importance of group dynamics in the code review process. This was learned by the lack of problems of this kind during the code review. There was no blame thrown out for any given error, and in general, group member egos never entered the discussion. There was a large sense that the code under review was the group’s code as a whole, and not any single member’s. (This was true even though one group member wrote the supermajority of the code for the Class.) Given that each group member had been through experiences that were much more adversarial, the lack of acrimony and task-oriented focus of the code review revealed the strong positive group dynamics exhibited by the group.

# 3.0 Reflection on Code Construction

The Code Construction has been very challenging. Several points, both positive and negative, have been raised regarding this phase by the group.

## 3.1 Time Required

Before Code Construction started, the group was confident that the time allotted was sufficient for the needs of construction. However, this has proven to take considerably more time than expected. Some of this may have been putting too much faith in the design, but another is that the group is still learning how to provide good time estimates for certain coding tasks. The Display elements in particular have proven to be challenging.

## 3.2 The Role of Design

While too much faith in the completeness of the design may be a negative point, the fact that a design existed at all has been a major benefit of this process. The existence of a shared reference document for the purpose of every Class has provided the basis for discussion for the group. It was also noted that writing the test plan from the design document, and the test code from the test plan, was very straightforward, and that has as its starting point the design document.

Due to the shared understanding provided by the design document, coding on the individual Class level has proceeded quite rapidly. Further, there has been no one person responsible for any particular Class, which is assisting in developing a shared understanding of the code, not just the design functionality driving that code.

## 3.3 Integration Concerns

Having said all of that, integration is proving to be quite the challenge. While the design has helped in that shared understanding, the code review revealed that the level of understanding was not as complete as originally thought. This was noted as a possible issue when choosing the Independent Component architecture, and it has turned into a reality.

## 3.4 Possible Changes

The group has already discussed all of these issues, and several changes to future projects as well as the current one have been decided upon.

The first change is that the design should be, if possible, more detailed in the connectivity of Classes. The Classes themselves were detailed quite well, but how the Classes connected was less detailed. This is directly impacting the ability to integrate the code.

The second change is to consider issues of constructability during the design phase. This is not meant to suggest that code should be constructed during design, but aside from certain enumerations, the design document did not consider the difficulty of actually implementing that design. This process has showed that this can cause implementation issues.

A third change that is being implemented is to create more fields in Classes to track states. Often, this is causing issues of simple reference, and creating more fields would seem a common-sense approach to this. If time permits, it is hoped that various aspects can be created to alleviate if not obviate this need. The Subject-Observer Pattern in particular would be helpful here.

# 4.0 Changes to the Design

Finally, there are some changes to the design both present and anticipated that should be documented.

As noted above, the code has many more fields that track system state than is shown in the design document. This may be replaced with aspects, but that will be done as time and understanding permits. A similar small implementation change is that, as expected from the testing document, the Interpreter has gained a timer to ensure AI turns do not cause massive problems.

The largest number of changes is in the Display. Probably due to the inability to create unit tests for user interface elements as noted in the testing document, the Display has undergone significant changes. First, each Screen has become its own Class, inheriting from the generic Screen Class. Second, there is only one game board panel, instead of treating each hex on the board as its own component. Third, the GameScreen Class received more methods revolving around updating user views.

The next largest number of changes occurred in the Flow component. The design for this component had few methods that performed a lot of functionality. This is being broken up into many smaller functions, each of whose function is much clearer. This resulted from a lack of ability to easily translate from the design document (‘what does this need to do?’) to the code construction (‘how does this accomplish it?’). The Flow component had the biggest issues with this knowledge mapping.

# 5.0 Final Thoughts

All in all, the code construction phase has gone well, even if not as smoothly as originally expected. Many valuable lessons were learned and, as it was noted that the code construction phase used everything that came before it, any issues were traced back to causes in previous phases when possible. The group is very optimistic about delivering the game on time and with no show stopping errors.