Construction Document
CMPT 370

Group C4
Jack Huang
Brandon Jamieson
Ixabat Lahiji
Daniel Morris
Kevin Noonan

Changes from our design:

- **Model:**

For every class we decided to make every field variable private since there are some fields that will not be changed after the creation of each class. We do not want any of these to be accidentally changed. Since each field value is private we implemented the appropriate getter and setter methods.

Board:

We decided to change the name of selectedHex to currentHex to ensure clarity within the methods of the class. We decided to add robot team creation into the board, so we added way to receive robot data from the view, like colours, team numbers. The rest we remained mostly the same.

Robot:

Since the range statistic is an integer we changed the datatype of range from Hex to an integer. We also changed the absDirection() method to being a field. It was not possible to calculate the absDirection so it will have to be stored in the robot instead. A constructor method was also created that takes in parameters for robotType and robotTeam to create a new robot.

RobotTeam:

We decided to implement a team number attribute. This would allow the creation of robots inside of this class to know what team number they are a part of. We also changed the datatype of colour from int to string to match the naming convention of colours within the view. A constructor class was also created that takes in parameters for isHuman, teamColour, and teamNumber to create a new robot team.

Spectator:

Our Spectator class has not strayed from our design specifications so far. It is possible this may change in the future, though we currently do not have any intentions of doing so.

Hex:

To make the creation of Hex objects easier, a constructor method was created that takes in the coordinates of a new hex to be created. Also the range and position fields were changed to positionX and positionY fields to match the naming system of our new coordinate system.

Interpreter:

In order for the robot to grab information that it needs about it's information, public getters for both string values and integers were created. These functions will be used within the interpreter as well, to obtain the code information for the robot. Words were not implemented as a separate class, and only handled as Strings. This is temporary, as modularity is important and will require the design choice we had previously agreed upon. The JSON parsing library used was the google library GSON.
Documentation on the library can be found here:
https://github.com/google/gson/blob/master/UserGuide.md
The repository main page can be found here:
https://github.com/google/gson

- **View:**

PlayMenu:

The PlayMenu is where users will decide which robot teams to play with, number of teams, as well as the board size to use. It has remained mostly the same as specified in our design, with minor changes to field types and names for implementation with java swing. Visually, it does not look like our requirements yet but more formatting and editing is planned to hopefully represent our requirements accurately.

GameMenu:

The GameMenu serves as the first menu presented during runtime, and has remained mostly the same as designed in our project. The functionality and fields all remained the same as specified in our design. However, visually, it is a much more basic version than our layout in our requirements document. This may be modified to represent our requirements more accurately as we make more progress though.

OptionsMenu:

Our OptionsMenu handles importing of robot teams from a remote server, and has remained the same as specified in our design, with no real changes made yet. It does not accurately represent the requirements document yet but it is intended that it will upon project completion.

InGameMenu:

The InGameMenu contained all visuals for the actual gameplay itself in our design. During construction, we found that splitting this into two classes would be an elegant way of implementation. We modified the original InGameMenu class to serve as a frame, then created

an InGameMenuPanel class for handling the actual display of the board. This allowed us to organize our code better while still keeping the core functionality of our design. Our InGameMenuPanel also required several private methods for drawing the board and robots, many of which were not originally in our design. For example, we implemented a drawHex() method that draws a hex based on a given position.

- **Controller:**

Our game's controller has remained mostly the same throughout construction. All methods will be implemented and will have functionality as outlined in our design document. However, there was one design change we made: the controller class is now a singleton. This way, the controller can be referenced at easy from anywhere, making button listeners for the view easy to implement. This also makes sense, as we will only ever have one controller during any point in runtime.

Pair programming reports:

Daniel Morris:

The pair programming session I had with Jack went very well. Jack and I had both been assigned to work on the model classes within our model view architecture. During the session Jack and I decided to work on the more difficult methods within the model such as searching for robots within a specified range of a location and moving robots between locations. At the start of the session it seemed as if we were both very involved in the problem at hand and at least in my mind we were both trying to be the pilot, instead of having one of one of us being the navigator. After a while I started to realize that when I am not the one at the computer I did not have to be totally involved in the task at hand. After realizing this I felt that when I was not the one sitting at the computer that I could think more lightly on the problem at hand and focus on the bigger picture. I definitely think this helped us become more efficient and ensure that the person at the computer was not too fatigued and could work very effectively.

At first I did not think the pair programming session with Kevin was going to work out very well but it turned out to be very successful. During the session Kevin and I decided to work on joining the model, view, and controller together. This turned out to be difficult for us since

that there were some classes that either of us hadn't worked with yet since they were made by other team members. Although with that being said we were able to decipher what needed to be done to join the classes together and we started working on that. The main problem that we came across was that the panels in the view had initially been constructed with separate listeners for every button or combobox they had displayed. Due to the architectural choice made in the design document we were not able to make changes to the model within these listeners. The solution to this problem was to make the controller an action listener and whenever a view panel received input from the user the panel would trigger the listener within the controller with a unique actionCommand. The controller would then use this command to make the necessary changes to the model and update the view accordingly. Once we had decided on this solution it got the ball rolling for us and it became a very efficient pair programming session. We were able to take turns as the pilot / navigator to maintain our focus and went on to accomplish the task at hand.

Kevin Noonan:

My pair programming session with Brandon went very well and was a great overall learning and working experience. We decided to implement the playMenu class that displays for the player the select the teams option, board size options and the ability to start the game. Going into the session it had been awhile since I had used java swing so I had prepared a little by looking up some of the basics. Because of this it was more beneficial for Brandon to be the driver first while I was the navigator. This way I was able to present my ideas while re-learning how java swing all works together. We worked this way for about an hour or so and then switched roles and at this point I was more prepared to take the role as the driver. During my time as the driver I felt a lot more focused in the the code I was writing and more confident as well. Brandon and I were able to get our ideas across to each other really well and with pair programming we did it easily and efficiently. Once we got the general idea of the playMenu going and I had been the driver for about an hour as well we start switching the roles whenever one of us wanted to take over. After multiple role switches and about three and a half hours we had a completed playMenu class with all of the functionality we wanted working. At the end of the session we put our time towards trying to solve a git issue so that we could start committing properly. Altogether pair programming was a very beneficial experience and it was a great time to. I would definitely do more pair programming in the future as you can learn and get work done at the same time.

Going into the second pair programming session with Daniel I was a little worried as I was not exactly sure how to implement code between the model, view and the controller. Because of this the start of the session was a little rough as we were both trying to figure out which code did what and how we were to implement the connection. Despite this the session was very successful in the end as a result of pair programming and our ability to work and share our ideas together. We started by going through the code of each class that we needed to create a

connection with and figure out how it all worked together in general. With both of us knowing this we were able to work out how to create a connection. We also realized during this that our original architecture was not optimal for creating the connection we wanted between the classes. Because we worked together to realize this we were able to come up with a solution relatively quickly. We noticed that we needed actionListeners in our controller class rather than having them in our view classes. We also needed to implement actionCommands in each class so that way the listeners could be called in the controller class. This way we could have the controller send the information of the model to the view. Because of this we were able to create a connection between the view and the controller and start the entire program from the controller. Once we had all of these ideas created and knew what we wanted to do we spent the next couple hours switching between driver and navigator implementing the connection between classes. With the knowledge that we had together this process went very smoothly and was easy to implement. Pair programming for this session was definitely a necessary addition to implementing this portion of the code and because of it we were able to complete the connection easily and efficiently. Overall it was a fantastic experience as we were able to work together in not just programming but figuring out how the code worked and working together to implement it all.

Brandon Jamieson:

My first pair programming session with Kevin was a great experience. We decided we would implement our PlayMenu, in which players will select teams and board size. I acted as the driver initially while Kevin served as the navigator. It took some getting used to but once we got in the groove of things, pair programming proved to be quite efficient. Implementing the GUI elements was simple with someone telling me exactly what listeners and elements needed to be made. On top of this, I found I was much less distracted when working with someone else and a lot more focused on the problems in front of us. After about an hour, Kevin and I switched roles and we continued to build the menu functionality, implementing board size and team selections.

During this, being the navigator also turned out to be a positive experience. Saying what I was thinking out loud really made me consider the reasoning behind my decisions in-depth. I also found it a lot easier to focus on the overall state of the menu, as well as what needed to be completed next without worrying about the actual coding of it. After a couple hours and several role switches, we finished the menu's functionality, as well as general formatting. We had been having troubles cloning the repository to commit, and also ended up solving the issue before ending our session (a directory name contained an invalid character and prevented any Windows users from cloning). Overall, I would say the session went smoothly with no real snags, and that it was a positive and fun experience.

My second pair programming session with Ix went just as well as my first session had. We met up on the weekend and promptly decided we would work on finishing up the InGameMenu and connecting it to the model through the controller. Again, I served as driver first while Ix navigated for me. We decided it would be best to change our design and create an InGameMenuPanel class for holding all gameplay visuals, and have the InGameMenu class serve as the frame for it. Coding this transition turned out to be quite easy with Ix's help, having no real issues that a couple minutes of thinking didn't solve. It was very helpful having a different viewpoint on the tasks, and made me closer consider alternative ways of doing things.

We switched after this, and decided we would implement a way to get robot images by loading them from our resource directory. After some quick research, we learned how to accomplish this and created a method for determining the correct image based on the robot type and team. We connected it where needed in the panel, as well as cleaned up a large amount of errors and loose code. After this we switched and began hooking up the InGameMenuPanel to the controller. During this, we needed to decide how each private listener class would get a reference to the controller, and concluded that modifying the Controller to use a singleton pattern would be an elegant solution. Implementing this concept was simple, and made the rest of the panel easy to make, taking about a half hour to implement. We also began planning out a skeleton for our option menu before ending our session. This session was both productive and informative; I learned new things and caught up on other parts of the development from Ix.

Jack Huang:

My First pair programming session with Daniel had been a great. We decided to work on the more difficult task within in the model, because both Daniel and I is assigned to work on the model part within our architecture. We worked on the search methods which search around the active robot for other robots within a given range and add each of the location to a single place for easy access. We also discuss the board itself and changed the coordinate system of the board. At the beginning of the session I was the Driver while Daniel had been the navigator. I felt that we didn't have a hard time getting used to it, we got right to work from the start. I find working on the code with someone questioning and discussing each part really made you focused and made less errors because of the second eye watching. It was a lot easier to just forces on the program at hand, while have the navigator making sure that the way working in the overall code. We switched roles around every hour continue working on the search and board.

When I was navigator it was also good, I didn't have to worry about the specific of the code and think about the overall effect the code will do to the code, and the functionality it should bring to the programs. I get to input different options and solution to the problems, while thinking about what is needed next. We finished the search and board after a couple hours. I think overall the session when great, we got done what we planned at the beginning of the session and tested the method as well. It was a productive and enjoyable experience.

My second pair programming session is with Ix. I wasn't really sure what we will do, but after it turned out to be an informative experience. Ix is assigned to work on the interpreter for AI, our goal was to figure exactly how the model is going to interact with the interpreter. For the first part of the programming session we worked the model part, with I being the driver, and Ix navigating me through on which part of the model the interpreter will call and which method and information would be important for the interpreter. While I code and make notes on future method that will be needed for the interpreter that have to be discuss more. We also tried to fix the github problem as I couldn't upload any file using the git terminal, because I can't checkout to any other Branch other than Master. For the second part of the session I switched to the navigator.

For the second part we figure out how the interpreter is sending instruction to the model, and me as the navigator tell and discussing what the name of method to call so the action would actually happen. Overall the session went well, I understood more on how the interpreter will interact with the model, it was really informative and positive experience.

Ix Lahiji:

My first pair programming session was done with Jack. We went over the structure of our individual sections, and talked about the interactions between our sections. I, being in charge of the interpreter, clarified some parts about how the interpreter would talk to a robot, and how they should be able to issue instructions to the board through the robot. As well, we talked briefly about robot librarian interaction and how that would be implemented. As well, we had attempted to fix an outstanding issue with our github repository. We were able to get it working on some computers, however there was a new issue where some computers were unable to detect the presence of the other branches.

The session was good, overall. It allowed us to communicate our possible upcoming issues beforehand, and helped us resolve some confusion. It also allowed us to get the repository working for us in some instances, which was better than the state it was in previously.

The second session was with Brandon. Because there wasn't much interaction between our individual sections (Brandon was working on the view, while i was working on the interpreter) we decided to give a quick explanation to one another about how our sections worked. It was a great refresher to see how the view worked, as I had not worked with java swing since CMPT 270. After the explanations we decided to try and switch to using a series of panels to hold sections of visual elements, and a frame in a separate class. We were experiencing an issue when trying to load the images for the robots, but after some research on the proper use of the functions (by looking at documentation of the particular function), we were able to get it working perfectly. Afterwards, we moved to fixing compiler errors and warning that were present in the code. We were able clear all of the errors, and most of the warnings in the view's

code. We then started work on the options menu, and planned out what exactly would happen in the individual functions, as well as the layout of the visual/interaction elements in the window.

Overall the session felt very productive and informative. I was able to brush up on my java swing and we were able to fix many issues in the code, as well as put skeleton code for future sections. I was able to learn a lot, as well as get a taste of working in a different section from my own.

Code review reports:

Our team met up over the weekend to review a couple pieces of code. We decided the first piece of code we would review would be the InGameMenuPanel, which handled all displaying of in-game elements like robots and the board. We chose this code because the way the view and model connect through the controller during gameplay is significant; any errors in this could lead to misrepresentation of the game's state. We booked a study pod in the library and loaded the code on the pod's monitor for all of us to see. As a team, we walked through the code starting with instantiation and then following method calls, with the code's writers explaining intended functionality before examining the actual implementation. This way, our group could ensure that implementation performed as intended gracefully, and that each team member properly understood how the code worked.

We discussed how the model's board representation maps to the view's board representation, and cleaned up some code related to the drawing of this. Doing the review also made it apparent where further commenting was needed, or where better variable names could have been used. Once we had finished reviewing the code, we discussed future functionality for the panel as well as changes to the rest of the system to support these changes. An example of such future functionality included a way to display a selected hex and display it's occupants. Overall, this review benefitted our project, resulting in a cleaner implementation, and solidified our team's knowledge of the panel's functionality.

After our first code review, our team took a brief break before regrouping and reviewing our second piece of code. We decided we would review the Board class, which contains most of the model's data for representing the state of the game, as well as interactions with the controller. It made sense to review this code because the board contains the game's state, meaning any errors here could result in invalid moves, gameplay imbalance, or a number of other undesirable consequences. We reviewed this code in the same manner as our previous code review: walking through the code from point of instantiation then following method calls and points of interaction until we had checked it all over. During this process, our team discovered a couple points in which additional commenting would be beneficial, as well as smoother ways of implementing

certain functions. For example, we found using our board's get hex function instead of referencing the hex array itself would result in much cleaner, condensed code at certain points.

We discussed how the board's coordinate system works and how robots are moved around through it, resulting in an overall greater understanding of the interactions and operations involved. We also discussed different ways of initializing the board so that each team has their own color, as well as discussing initial positions for each team. Once the review was done, we committed our work, discussed the status of our document, then ended our session. This review session was both informative and productive, with all team members now having full knowledge of how both the view and model work. On top of this, the session resulted in a more elegant board, as well as a clear idea of what features need further work or implementation.