Construction Experience Document

- Software Engineering Team: Arianne Butler, Michael Graham, Samuel Horovatin, Kristof Mercier, Christopher Mykota-Reid

Contents

Code Review Session

Code Review on the GameBoard class, implemented by Michael Graham

November 26, 2016 at 3:00pm – 5:00pm in SPINKS 372

Summary

The first thing that was established in our code review was that all questions and comments were valuable. We wanted to ensure that everyone was heard in order to promote an open work environment, ensuring clarity was our main priority.

As much of our system relies on how the coordinates are stored and accessed, we decided that it would be best to review the GameBoard class, as well as classes that interact with and manipulate it. Michael implemented the vast majority of this section, so the first step was to logically step through the code, explaining what each function is for, as well as why it was done that way. From there, each member opened the source code and began to read it line by line. Anytime someone had a question or a comment, we would discuss it as a group and ensure that it was appropriately addressed and a paraphrase of the question was recorded.

Note: The follow quotes are not verbatim from the meeting and have been rephrased to succinctly contain all of the important information that was discussed. The name attributed to the responses belongs to the group member who began answering the question. Our group made an effort to allow each member to give input on each of the topics.

Group Discussion:

- Q “Why are coordinates stored in a hash instead of something like a 2D array?”
  - A Michael: “A 2D array will have unused space. You would have to loop over coordinates that don’t exist and check for nullity every time. Also, if you don’t know the exact index of the coordinate you can just check if the hash contains it. “
- Q “Why are you setting coordinate’s (x, y) values to the physical position on the screen?”
  - A Michael: “By doing that, we can just draw the hexagon at exactly where it’s supposed to be. This reduces the amount of math that needs to be done every time the board is repainted because it’s only done when the game is initialized.”
- Q “Why do you use a hashmap keyed to a pair of integers with a value of a coordinate?
  - A Michael: “That’s… a good question. Now that you mention it, it’s completely pointless; the key is the exact same as the value. Thanks for pointing that out. Is there anything else you guys notice that needs patching up?”
  - A Kristof: “You could map a coordinate to its corresponding polygon/hex object. That way you reduce the number of hashmaps in the system, and the key (or the coordinate) maps to the value (or the hex) that needs to be drawn”
  - A Michael: “That’s a good idea. I’ll refactor the code after this meeting.”
- Q “How are the values determined in getNeighbors?”
  - A Michael: “They’re based on the absolute position of each coordinate. I can approximate locations between each variable thanks to the math in the function. I can only approximate things because integers automatically truncate values so I may be slightly off on each ‘guess’ of where the coordinate that I’m trying to get is. That’s one of the reasons I created an inner class to the GameBoard that determines if a coordinate is within range of a circle. “
- Q “Why do you check if a coordinate is in a range of a circle? We’re working with hexagons – not circles.”
  - A Michael: “Because it’s less computationally expensive to check if a given value is in range of a circle. It’s as simple as using the Pythagorean theory on the center coordinate.”
- Q “I thought running a square root operation was resource intensive, how is this better than doing the math to calculate the bounding box of a hex?”
  - A Michael: “In calculating the bounding box of a hex you have to use sin, cos, and tan quite a bit while working with radians. These operations combined are more complex than the single square root function

that I use. Also, this is a turn based game, and if a value takes a millisecond to calculate I don't think it will really make a difference."

Q "What does the drawCenteredCircle function do?"

A Sam: "That's for when Swing draws circles. Their central point isn't in the actual center of the circle. The math creates an offset coordinate and draws the circle so the center of the circle visually corresponds to what the physical coordinate is."

Q "Okay but where are you using it?"

A Michael: "It's just there to debug and visually see the bounding circles of each hexagon. It allowed us to find some of the problems occurring with bounding circles like bounding circle overlap."

Q "What exactly is going on inside of the buildGrid() function?"

A Michael: "I don't fully understand all of the math. I added a link to Stack Overflow where the one user wrote it. It may be best to check there."

Q "Why doesn't the origin of the Gameboard actually equal the central coordinate once the grid is built?"

A Sam: "It's largely to do with the same issue presented with drawCenteredCircle. The hex's have to be drawn at an offset to ensure that they're actually centered in the JPanel."

Q "How did you the corresponding hex to a mouse click; is that found by using a circular range?"

A Michael: "Yes, I made and keyed to a circular range with a corresponding hex/coordinate as its value. Then what I do is loop through all the possible ranges and find one that the current selected coordinate is a part of. Because this operation is done frequently it was put into its own function getKeysFromRange."

Q "If a coordinate is not in range of any of the ranges, the system loops through all possible ranges and checks if the coordinate is in range only to return null in its worst case? Is that a good solution? It seems like it would be a really intensive algorithm that is going to be called a lot.

A Michael: "Honestly, I couldn't think of a better one. If anyone has any ideas, please let me know."

A Chris: "You could do the same math that you did in buildGrid to get the exact location of a hex if it was a 2DArray. There wouldn't be any rounding errors due to it being an integer either then."

A Sam: "That would reduce the complexity quite a bit, cause the x and y offsets are calculated upon initialization and are stored for the duration of the Gameboard's life… So is it worthwhile to change the implementation at this point?"

A Michael: "Honestly, I won't have time to fix it if we decide to change it. I also don't think it's really a necessary refactor because I will still have to check if a mouse click falls within a certain range – the user won't click on the exact pixel coordinate of each hex."

A Kristof: "At this point it's not worthwhile. The solution works, and we don't really need to worry about that sort of thing with modern processors, but it's a nice thing to note."

A Michael: "Are we all okay with not changing it?"

All are in agreement.

Benefits and costs:

- **Benefit & cost:** We noticed that there were unnecessary hash maps that needed to be removed. This hardly affected our current work thanks to MVC's modular design. The change was completed in under an hour as it required only basic refactoring. The benefit of this was that less storage was occuring inside the system, and that there were less hashes to keep up to date (reduced coupling).
- **Cost:** Sam and Chris pointed out that the same math used in buildGrid( ) could be used to instantly calculate the pixel coordinate of each hexagon based on their index. This reduces the amount that bounding circles are relied upon in getNeighbors( ). It was not worthwhile to implement.
- **Cost:** We spent a significant amount of time reviewing the code when other features could have been implemented. The main benefit is that the entire team now understands the functionality of the Gameboard/GameScreen/Coordinate classes much better, but it wasn't necessary for the portions of the project that they are working on. Furthermore, all pieces of code written by other group members will not change as a result of this code review, because the interfaces that they are working with have not changed.

Pair Programming

Arianne Butler:

Pair Programming with Chris Mykota-Reid on Friday Nov 25 /16 at 4:30pm – 6:20pm:

For my first pair programming session, I worked with Chris. We spent our time developing aspects of the user interface. Our session was early on in interface development, so our first task was to figure out how to connect the views sequentially with one another and with the main function which runs on start-up. This was slightly trickier than expected. Having Chris's insight was helpful. He noticed aspects of the problem that I did not, and his attention on the work provided for a stronger level of understanding. One personal benefit that I experienced from pair programming is a higher level of confidence in my code. I sometimes struggle with trusting my own intuitions, and having Chris there to discuss my methods gave me confidence in my own understanding of the problem. I also experienced an increase in productivity. We stayed on task for longer periods of time and were able to solve problems faster and with higher efficiency. Overall, pair programming was a positive experience, and I now understand its benefits. I would use pair programming again in the future, but probably only for tricky implementation areas. I'm not confident that it saved a significant amount of time, or that I would not have been able to come up with a very similar solution on my own.

Pair Programming with Michael Graham on Sunday Nov 27 /16 at 5:30pm – 7:40pm:

For my second pair programming session, I worked with Mike. We worked together to integrate the GameBoard that he has been building with the GameScreen interface that I've been building. This task was the obvious choice for Mike and I, because we each created aspects of the code we would be working on. We worked together on improving the overall look and feel of the views, as well as to incorporate the GameBoard into the set of Views that make up the user interface. Mike has a lot of experience with figuring out various IDE's, so his knowledge in IDE navigation, especially with relation to the GUI's, was extremely helpful. We worked well together, using trial, error, and google to figure out a series of problems, including how to centre our GameBoard in the GameScreen view. I enjoyed my pair programming session with Mike, and felt it was valuable for this portion of the project. Because we were merging two separate sections, it made sense to have both of us there. This meant that all of the relevant knowledge was present between the two of us, which likely prevented errors that could have arisen as a result of not understanding another team member's code. I would definitely choose to pair program again in the future, especially when merging work from two different developers.

Michael Graham

Pair Programming with Samuel Horovatin Nov 22 /16 at 1:30pm to 5:00pm @ MLC

Me and Sam met up on Tuesday at the MLC. We decided on what coordinate system we would use, implemented it, created a functional prototype user interface, and encountered a problem that we couldn't figure out involving the hexes orientation. A large portion of this time was spent discussing how we were going to implement the functions that were outlined in the design document. I found that pair programming with Sam was for the most part great – it was nice to have someone who was completely on the same page as you to talk about things with. You didn't need to explain things if you were having a problem, because they were completely caught up with you and understood the sequence of events that led to that problem. The problem that I had was that I feel quite anxious when I am writing code and someone is watching. To be fair though, this isn't a problem with pair program, but instead a problem that I must resolve myself.

Pair Programming with Arianne Butler on Nov 27 /16 at 5:30pm to 7:40pm

Arianne approached me to work on the UI as she wasn't entirely sure how she was going to link what me and Sam created with what she had already made. First, we sat down and discussed exactly how she wanted it incorporated, and then we proceeded to implement it. This required a bit of research into how Intellij incorporates custom components. From there, we tidied up the user interface using some techniques that I learned from designing webpages. Really, a significant chunk of this pair programming session was used as a learning experience for both me and Arianne as neither of us had any idea how to incorporate custom components into the user interface that she had implemented. Pair programming with Arianne was insightful. It caused me to think about why we did something instead of just doing it. Because of this, I think if pair programmed more in the future it would be beneficial to not only my understanding of things, but also my ability as a programmer.

Samuel Horovatin:

Pair Programming with Michael Graham on Nov 22 /16 at 1:30pm to 5:00pm

Pair programming with mike was an extremely insightful and positive experience. It allowed me to understand the gameboard class to a higher degree than before. Through working through the math behind the hexagonal grid we were able to come to a consensus that axial coordinates best fit what we had in mind. Overall I would say pair programing was an excellent method of coding for this portion of the project and would do it again in a heartbeat.

Pair Programming with Kristof Mercier on Nov 25 /16 at 1:00pm 3:30pm

Pair programming with Kristof was an excellent way to gain understanding of part of the system I had very little to no understanding of prior. As Kristof had been the main creator of the interpreter class, it was immensely enlightening to get explanations of functions and walk through the process during coding. Due to us both thinking of the problem at the same time, we were able to come up with a novel solution to the storage and execution of parsed Json strings. Overall, pair programming was essential (for me at least) to gain a deeper understanding of the system as a whole.

Kristof Mercier:

Pair programming with Samuel Horvatin on Nov 25 /16 at 1:00pm - 3:30pm

During this session we focused on implementing parts of the Interpreter and IntepreterFunctions classes which deal with storing Forth variables and mapping Forth Words to internal command within the interpreterFunctions class. We also implemented and reviewed some of the individual functions within the InterpreterFunctions class. The session went fairly well and we got through a tricky implementation decision, storing known ForthWords in a dictionary, with relative ease. While we were still trying to conceptualize the full impact of this implementation decision, I noticed that I could focus on thinking from a higher abstraction, as I could rely on Sam to deal with the lower level. In the time that I spent figuring out how the flow of the system would benefit, he was considering how to load the dictionary at the beginning of the program's execution. For this instance I think that pair programming was a valuable asset, though I gained insight as to its drawback as well. As I had been previously implementing other parts of the interpreter, I had more knowledge of the subsystem coming in, which required for us to take some time to get Sam caught up. It would almost always be the case where one of the programmers has more experience in an area than the other, and this could be a considerable drawback in situations where this difference in understanding is more significant. Overall I would conclude that the session was a success, though I would not consider pair programming a practical method in all situations.

Pair programming with Chris Mykota-Reid on Nov 26 /16 at 9:00pm - 11:10pm

At this point, the Interpreter implementation was entirely planned out, if not implemented, with the exception of the conditional and loop functionality. This was one of the most difficult parts of the interpreter, and though we did not finish its implementation within the session, we planned out how it was to be done, making further implementation fairly simple. We ended up deciding on keeping stacks for each type of loop and if-statement, which held all of the necessary information for the loop or conditionals execution. This ended up being an efficient solution for a variety of reasons, and pair programming effectively streamlined the decision making process. Seeing as I was the one who had implemented almost the entire interpreter coming in, it was more difficult to get Chris caught up on the implementation details. This drawback was fairly insignificant, as most of the discussion was conceptual, and I could fill in the more specific implementation details on the way. I have always felt that explaining one's thinking forces one to consider the concept more succinctly and this is something that pair programming allows for. I think that if we had designed the implementation of the entire system in such a way as to minimize the difference in knowledge of the pairs, the process would have been more efficient. The group all had an overall understanding of each class from the design document, which held fairly well, but having implemented part of the component will always yield a greater understanding. In the future I would design the construction phase with this in mind, ensuring more efficient pair programming. In conclusion, this session was useful and allowed us to efficiently make important decisions about the implementation, though I feel that designing a construction phase oriented around pair programming would make the process more effective.

Christopher Mykota-Reid:

Pair Programming with Arianne on Nov 25 /16 at 4:30pm to 6:20pm

We pair programmed for around 2 hours creating GUI functions that transitions between the different views. I watched and commented while she coded. It worked well since I was finding solutions she wasn't and vice-versa so we were not getting stuck on one problem for too long. One thing that I didn't realize is that one person can be researching how to do something or fix a bug while the other person continues coding on another section. Or both people can be researching two different parts to a problem and sharing what they learn. This seems like a really great use of two people since not only do you have 2 unique perspectives working on the problem but now you also get near 100% efficiency in these situations. Another thing is that the person who doesn't have to type can be thinking about the problem in a much more open way since a chunk of their brain isn't tied down in typing out code and having to think about the problem in a more strict, concrete way. It's hard to describe, but when you're actually typing code into the computer your thought space is much narrower than when you are just sitting there, thinking. Generally the pair programming was a positive experience where we were taking each others suggestions with an open mind, both working towards the common goal of getting the program working. Pair programming seems like a very powerful tool in design of programs. It's also pretty fun too! Maybe to it's detriment; if you're having too much fun you're not getting as much work done as you could be.

Pair Programming with Kristof on Nov 26 /16 at 9:00pm - 11:10pm

We pair programmed implementation of if statements and loops in the interpreter. We didn't finish the implementation together but we did complete a significant chunk of it. The forth interpreter was kind of tricky because we hadn't taken as much care in the design phase so a lot of our time was spent thinking about the problem. Beyond the normal benefits of pair programming this session was helpful in that I gained an understanding of a part of the system I didn't have before. The interpreter was kind of an enigma and actually getting in and working on it got me to understand (at least somewhat) what is going on. On the other hand I felt less useful than usual while helping Kristof. I didn't know much about it going in so it took a while for me to understand what we were trying to do and most of it was just him dictating to me the aspects of the interpreter. I could still check for surface level logical errors but deeper ones might have eluded me. We also read over the robot language document together while trying to understand loops. Here the benefit of having two people was quite apparent. We were both picking up on different chunks, filling in each other's understanding of loops, and asking each other questions about our understandings when they were in conflict. Though not necessarily pair programming it did highlight the benefit of having another person thinking about the same problem which is a part of why pair programming is usually a successful strategy. Overall it was an O.K. session.

Changes from Design Document

View:

- **SetUpMenuPopUp View:**
  - We decided to implement the SetUpMenuPopUp as one window only, where robot selection for all three types occurs on the same screen. Initially, it was a set of three windows (see Design Document, page 11), one for each robot type (scout, sniper, and tank). We feel this layout is more readable and user-friendly, allowing the user to see the team they are creating all at once, as opposed to showing only one robot from a team at a time. It also makes it easier for the user to change their mind and select a different robot, because they do not have to navigate through views to reach the various robot types.
- **GameScreen View:**
  - The GameScreen view now contains one more button. The new button is the "End Turn" button. Upon discussion, our group realized that without an "End Turn" button, a player would not be capable of moving less than their maximum number of movements.

Model:

- **Robot Class:**
  - New Variables:
    - direction: integer – The direction that the robot is currently facing
    - hasAttacked: boolean – True if the robot has attacked, false otherwise
  - Function Changes:
    - Added:
      - + isValidJSON(Map json) : boolean – Checks if the JSON input is valid in the constructor
        - Added because it is important to verify that robots are valid
      - - makeRobotID( ) : void – Creates a RobotID as specified in the Design Document
        - An internal function that is only called by the constructor. It exists primarily to tidy up code and make it more readable
      - - initializeRobotAttributes( ) : void – Initializes robot attributes based on the robot type
        - An internal function that is only called by the constructor. It exists primarily to tidy up code and make it more readable
      - - initializeCoordinates( ) : void – Initializes starting position of the robot based on the GangID. GangID's 0 – 5 will each be associated with their own team colour and each colour will always begin on the same tile of the game map
        - An internal function that is only called by the constructor. It exists primarily to tidy up code and make it more readable
      - - addToHashMaps(Coordinate coordinate) : void – An internal function that places each robot onto both of the corresponding hash maps
        - Created to reduce the amount of repeated code
    - Changed:
      - + getStrength( ) is now getAttack( ) to match the initial project specifications in game.pdf

- **Gameboard Class:**
  - Variables:
    - Added:
      - + Coordinates : Hashmap – A hash of each Coordinate, keyed to a pixel point on the graph
        - This was added because a place to store the physical hexes was not actually specified in the design document.
      - - boundingCircles : Hashmap – A hash of bounding circles for each hex tile
        - Added for easy calculation of mouse click events.
      - - radius : integer – The radius of each hexagon & the hex board. This is used quite a bit in the calculation of where each coordinate will be on the grid.
      - - origin : Coordinate – The position that the Gameboard uses when calculating the offsets of each hexagon with relation to the center
        - Added to specify a starting position of where hexes are to be located around.
      - - center : Coordinate – The absolute center KEY of the hexagon hashmap
        - Added so that there is easy access to the key for the center of the board. It's necessary because each hex is drawn at an offset from the original position.
    - Changed:
      - + robotCoordinate <Robot, Coordinate> instead of robotCoordinate<RobotID, Coordinate>
        - It makes it quicker (just pass in this).
  - New Functions:
    - + getKeyFromRange(int x, int y) : Coordinate – Returns the key (a coordinate) that corresponds with a given range
      - Added so that there's an easy way of getting the corresponding key of an approximate coordinate.
    - + getKeyFromRange(Coordinate coordinate) : Coordinate – Returns the key (a coordinate) that corresponds with a given range
      - Added so that there's an easy way of getting the corresponding key of an approximate coordinate
    - + getNeighbors(Coordinate coordinate, int range) : Coordinate[] – Returns an array of all of the neighbors of a given coordinate
      - Added to easily access the surrounding tiles.
    - - getNeighbor(Coordinate start, Coordinate end) : Coordinate – Returns a neighbor coordinate, **for internal use only**. This is used as a helper function for getNeighbors.
      - Created for the sole purpose of making code more readable in getNeighbors.
    - + direction(Coordinate coordinate, int direction) : Coordinate – Returns the neighbor of a coordinate in a given direction
      - This was created to implement movement in the robot. It provides an interface between the direction that the robot is facing, and which way "forward" is.
    - + getRadius() : int – Returns the radius of the Gameboard
    - - buildGrid() : void – Builds a grid and adds all corresponding coordinates to the coordinates Hashmap
      - This is self explanatory.
  - New Internal Classes:
    - CircleRange class:
      - The purpose of this class is so that we can easily get the corresponding hex to where a player has clicked. Think of it as a bounding box for hexagons.
      - Variables:
        - - point : Point – The position that the circle is to be drawn around
        - - radius : int – The radius of the circular range
      - Functions:
        - + CircleRange(Point origin, int radius) : CircleRange – Creates a new circular range
        - - inRange(Point point) : boolean – Checks if a given point is in range

- **Coordinate Class:**
    - Variables:
        - Added:
            - hex : Polygon – stores the physical shape that is to be drawn at that coordinate
                - Added to reduce the amount of hash maps needed (we originally had one to lookup the corresponding shape).

    - New Functions:
        - + add(coordinate a, coordinate b) : Coordinate –Returns the coordinate that results from adding two coordinates

- **GameMaster Class:**
    - Implementation:
        - Our understanding of the implementation of turns was incorrect. We thought, for example, that a team with one robot would move three times in one turn (once each round). What we realized is that each robot moves once per turn, so our idea to use a circular linked list would not work. Instead, we chose to implement a queue of all robots, sorted into the proper order for the first round of the game. Then, turn order would be kept by removing and re-adding robots as each of their turns pass.

    - New Variables:
        - - gangs: an array of gangs participating in the match – we needed a method for enquiring whether a particular hang is an AI or not, so we gave the GameMaster a gang field.

- **StatsLogger Class:**
    - Variables:
        - +hashmap of robots and stats – At design time, we weren't sure how GSON would work specifically, but have since realized that it can be given a hashmap to generate a JSON with. To update variables, they must be edited through the hashmap.
    - Internal Classes:
        - RobotStats class:
            - An inner class specifically for logging updates with regards to the robots. The robot class doesn't have enough fields to hold all the data that needs to be logged so we created an inner class that would have fields only relavant to logging stats. This makes JSON generation very easy as we can, with our hashmap, map robotIDs to RobotStat classes and pass that to GSON to create us our JSON file.

Interpreter:

Most of these changes were inspired by the idea that each function should have a specific purpose. Many of the new functions contain a subset of the functionality described by a single function in the Design Document.

- **Interpreter Class:**
    - New Variables:
        - currentElement: a package private integer which contains the index of the current word in the forthWords list
        - robot: A private reference to the robot. Before, this was only stored in the InterpreterFunctions class but it was also included here to simplify references to it.
        - acceptedWords: A private dictionary of ForthWords as defined by the Forth Language document. This will allow us to check for code correctness, and allow us to easily refer to the function that each ForthWord corresponds to.

- New Functions:
    - - loadWords: A helper function to the play function which will deal with all Forth code parsing
    - -loadDict: A function to load the dictionary of accepted Forth words from words.txt

- InterpreterFunctions Class:
  - New Variables:
    - loopStack: Stack - A stack to store loop structs which contain all necessary information for loop execution
    - conditionalStack: Stack - A stack to store Booleans, used to determine which part of the conditional to run
    - Interpreter: Intepreter - The interpreter is stored in this variable so that it can be accessed from this class
  - New Functions:
    - + store(): void - Stores a value in the given variable
    - + load(): void - Loads a value from the given variable and pushes it to the stack
    - + begin(): void - Initializes a while loop struct and pushes it to the loopStack
    - + until(): void - Pops a boolean and decides whether or not to loop
    - + do(): void - Initializes a loop struct with a for loop and push it to loopStack
    - + loop(): void - Decide whether or not to loop based on the top element on the loopStack
    - + ifCondition(): void - Add a boolean to the conditionalStack and call seekTo to skip to the else branch if that boolean was false
    - + elseCondition(): void - Remove a boolean to the conditionalStack and call seekTo to skip to the then branch if that boolean was true
    - - checkBool(String element): void - throws an exception if the given element is not a boolean
    - - checkInt(String element): void - throws an exception if the given element is not an integer
    - - seekTo(String element): void - changed Intepreter.currentElement so that it points to the next occurrence of the given String

  - New Internal Class:
    - LoopStruct Class
      - Variables:
        - - start : int - The value at which the loop will start
        - - end: int - The value at which the loop will end
        - - i : int - The loop's current increment
        - - startIndex: int - The index of the word which begins the loop. This is the only field relevant to a guarded loop