**CHESS APPLICATION**

Charlie Mietzner -- clm133

Ritika Maknoor -- ritikamak

Peter Stamos -- petestamos

Rohan Patel (Scrum Master) --patelrohanv

10 November 2016

**CS 1530 – SPRINT 4 DELIVERABLE**

**DESCRIPTION OF SPRINT**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*We began this sprint with a strong idea and grasp on what we wanted to complete during it. By this 3rd sprint, as a team, we had gained a solid understanding of how the 2-week process ahead of us would most likely carry out. We had a general plan for what we would complete in the rest of the sprints that we had ahead of us, and therefore were more confident throughout this sprint, in comparison to the previous ones, about how exactly our work during it would lead to the completion of our final product. During our previous sprint’s retrospective, we noted what our strengths and weaknesses were- both individual and as a group. We noticed that during our last sprint, we had very effective communication but wanted to improve on being more decisive/clear on what we wanted to accomplish by the end of the sprint. We wanted for all of the teammates to have a good understanding of what our principle goal was for the sprint, and how each user story assigned to that sprint would contribute to the accomplishment of that principle goal.*

*Our sprint planning session for this sprint was effective and efficient. We discussed what we had completed so far- what components our “working software” had, and how close it was to meeting the end goal/all of the requirements for this product. During our first sprint, we broke down our backlog of user stories into 3 categories (Chess GUI; Game Mechanics; Backend), and had a vague idea of how we felt would be more logical to go about completing all of them. During our second sprint, using that initial backlog with the categories, we chose and completed user stories that were centered around setting up our Chess GUI and determining our graphics. Once we went over what we had completed, we discussed what that the most logical next step/principle goal for this sprint would be- to have our backend of game mechanics be all set up. We decided that by completing that this sprint, in the next sprint we would be able to synchronize the GUI and backend of game mechanics and finally have a Chess GUI that the user could begin to interact with by moving pieces around the board. Having this principle goal determined, we were able to choose our user stories rather quickly and divided them up amongst us by taking into consideration what domains and aspects of programming each of us felt we were strongest and most familiar with.*

*Throughout the entirety of the two-week span, we communicated very well. Due to the fact that the user stories that we chose to complete were all very much intertwined with one another, we were all in almost constant contact. The nature of the tasks that the user stories called for required manipulating one another’s code- i.e. abstracting away methods, organizing classes, synchronizing objects, etc. We all actively communicated and were available whenever another teammate needed anything. We did not have many disagreements. The only issue that arose was the fact that so many of our user stories overlapped in terms of the code we were working on. It was difficult when multiple team members were working on the same classes- such as a specific piece class and both members were re-organizing the code/methods in different ways. However, we resolved this for the most part pretty well by communicating with each other about who would modify what they needed to first and who would do so second. We were also particularly active with often merging our individual completed modifications to the master branch. By doing so, we all always had the most up-to-date version of the code subsets. After about the first week of the sprint, we, as a team, had a cohesive system for working on the code efficiently.*

*We did not interact with the customer during this sprint. After the previous sprints, we had a Chess GUI set up and most of the graphics determined. After this third/current sprint, we have finally completed setting up our backend of game mechanics (classes for board and piece movements that deal with the actual algorithmic implementations of the rules of chess). At the end of this third sprint we do indeed have completed and functional “working software”; however, this version of the software does not yet allow the user to interact with the board/Chess GUI. During our next sprint we plan to synchronize the GUI and the backend, and therefore during our next sprint (after we begin to synchronize the two), we plan to consult our customer and show them our interactive graphical display. We hope that doing so at that point in the near future, rather than now, will allow the customer to finally interact with the GUI physically. We will perform user observations as we take note of how they interact with the system by default. We hope that at that point, their feedback and our observations will provide us with a sense of direction- whether our understanding of how they expect to perform tasks (such as moving chess pieces on the board) is in line with theirs, or not. By having customer interaction occur near the beginning of the next sprint, we hope to solidify our understanding of the customer’s expectations before spending the entirety of the next sprint writing invalid software.*

*We faced many challenges when writing our code. During this sprint, we were focused on the backend of game mechanics. We already had created our graphical product and developed components of it, but now we had to actually write code that dealt with the algorithmic implementation of the rules of chess. This was not an easy task because our program had to be dynamic enough to handle user interaction, GUI interaction, and “computer opponent” interaction. It was not simply a program that would run once, take in user input for moves, and then complete. We began by writing a skeleton backbone for what our code would have to encompass- classes for different pieces and for the board itself. Then, we decided that abstraction was necessary. We created an abstract class Piece that was a generic Piece object; therefore, as we developed further, we could use it as a superclass to more specific piece types: kings, queens, knights, bishops, rooks, pawns. We decided that this was the most effective way to go about writing clean and useable object-oriented code. This helped us have uniformity amongst all of our pieces; we simply had to then write more specified code for each type of piece in terms of how exactly they could move on the board. Once we began to brainstorm how to effectively organize our code, we became much more familiar with what seemed to work best for us and would minimize duplication of code. We did not come across any challenged when writing our tests once we got started and figured out what exactly we needed to test for- both base cases (expected use cases) and edge cases (unexpected use cases). It was nice that we could write similar tests for the different piece types, since we were testing similar actions- valid/invalid movements, bounds on board, and occupancy of same/opposite pieces in squares.*

**LISTING OF COMPLETED USER STORIES & STORY POINTS**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

* As a player

I want the ability to flip the board,

So that I can view the game from the perspective of the other player.

STORY POINTS: 8

* As a player

           I want the ability to change the colors of my pieces,

           So that I can play the game with a pleasing color scheme.

STORY POINTS: 4

* As a player

I want the chess board to be at the top of the display window and any buttons to be at the bottom

           So that I see the chess board clearly while playing a game.

STORY POINTS: 2

* As a player

           I want to click anywhere in a square to execute a move to/from that square

           So that I am not restricted as to where I have to click.

STORY POINTS: 2

* As a player

I want to first click a piece, then click its destination square, and then see the piece move to the indicated destination

           So that I can visually see that a move has been made.

STORY POINTS: 8 **(not completed)**

* As a player

I want for there to be a list of all prior moves, on the side panel next to the game board

           So that moves can be kept track of during the game.

STORY POINTS: 4 **(not completed)**

* As a player

           I want white, and only white, to go first

           So that the program adheres to chess rules.

STORY POINTS: 2

* As a player

           I want turns to be taken (white/black/white/etc.),

So that the game follows standard chess rules.

STORY POINTS: 4

* As a player

I want pieces to not make illegal moves

So that neither player can cheat.

STORY POINTS: 16

* As a player

           I want an error message to appear if I try to make an illegal move

           So that illegal moves are blocked from occurring.

STORY POINTS: 8 **(not completed)**

* As a player

I want the board to be organized so that no 2 pieces can ever occupy the same square

So that the game follows standard chess rules.

STORY POINTS: 4

* As a player

I want special movements to be legal- such as “en passant”, “promotion”, and “castling”

So that the game follows standard chess rules.

STORY POINTS: 2 **(not completed)**

TOTAL VELOCITY = 42

**LINK TO CODE ON GITHUB**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

On GitHub: https://github.com/clm133/CS1530groupproject

**LISTING OF DEFECTS**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*SUMMARY:*

*The bishop class currently cannot check if its movement path is impeded.*

*DESCRIPTION:*

*When observing the bishop class’ implementation of movement, does not check if there is a piece that impedes the path between the current position and destination.*

*REPRODUCTION STEPS:*

*1. Refer to source code for the Bishop class.*

*2. Inspect the implementation for movePiece method.  
EXPECTED BEHAVIOR:*

*The method should return false if the path is impeded.*

*OBSERVED BEHAVIOR:*

*The method returns true if the destination is valid according to a bishop’s legal moves, not if it is valid in accordance to the board.*

*NOTES:*

*This sprint we focused on implementing movement for all pieces and setting up a relationship between all piece classes; we agreed to focus on the movement in regard to game mechanics in the next sprint. Will address this defect further then.*

*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*SUMMARY:*

*The rook class currently cannot check if its movement path is impeded.*

*DESCRIPTION:*

*When observing the rook class’ implementation of movement, does not check if there is a piece that impedes the path between the current position and destination.*

*REPRODUCTION STEPS:*

*1. Refer to source code for the Rook class.*

*2. Inspect the implementation for movePiece method.  
EXPECTED BEHAVIOR:*

*The method should return false if the path is impeded.*

*OBSERVED BEHAVIOR:*

*The method returns true if the destination is valid according to a rook’s legal moves, not if it is valid in accordance to the board.*

*NOTES:*

*This sprint focused on implementing movement for all pieces and setting up a relationship between all piece classes; we agreed to focus on the movement in regard to game mechanics in the next sprint. Will address this defect further then.*

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_