**CHESS APPLICATION**

Charlie Mietzner (Scrum Master) -- clm133

Ritika Maknoor -- ritikamak

Peter Stamos -- petestamos

Rohan Patel --patelrohanv

1 December 2016

**CS 1530 – SPRINT 5 DELIVERABLE**

***DESCRIPTION OF SPRINT***

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

*We went into this sprint with a strong idea and grasp on what we wanted/needed to complete during it. By this 4th sprint, we, as a team, had a general understanding on what we would most likely be capable of accomplishing in the two-week span. Being the second to last sprint (next would be our last two-week span to develop the software and meet all of the requirements of the customer), we were nearing the end and had to be particularly focused and confident with every development. Not only did we have our own goals, for this sprint ahead, that would help us continue on our path that led us through our progression of the project thus far; but also, we had new user stories from the customer that we had make our highest priority for our working version of software by the end of this sprint. The customer changing their mind and introducing new user stories meant that we had to re-architect our code/design to meet these new needs.*

*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 focused/committed to our specifically assigned user stories. This was a difficult task because so many of the user stories overlapped in terms of functionalities of the application that were being manipulated. Often times during the last sprint, time was wasted by multiple team members working on similar refactors of the same method. Therefore, we decided that we needed to improve on being more strategic in how we divided up the user stories for the sprint.*

*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). By the start of this 4th sprint, we had completed a large percentage of user stories from the Chess GUI category. We had the interface of the application (visual display and channel for user interaction) almost completely set up. We then discussed what the most logical next steps/our priorities should be for this sprint. We went through our updated backlog of remaining/ incomplete user stories and chose our stories for the sprint. We were able to do so rather quickly. We divided them up amongst us, taking into consideration what domains and aspects of the development needed that each of us felt we were strongest and best fit to take on.*

*Throughout the entirety of the two-week span, we communicated very well. Due to the fact that the group of user stories that we chose to complete were all very much connected/dependent on 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 were also better at commenting our code this sprint, which helped us all tremendously. 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 still difficult when multiple team members were working on the same classes (even though we tried to be more strategic in assigning user stories this sprint). The majority of our user stories this time dealt with further developing the functionality of the pieces on the GUI (expanding the visuals, allowing user interaction, movement), and with initializing the backend of the game mechanics (legality of moves, swapping turns, flipping of the visual board for each turn swap). However, we resolved this 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 individually 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/effectively.*

*We did interact with the customer during this sprint when they provided us with modifications to their requirements for the software. They provided us with 2 new and high priority user stories that in fact helped us understand what they, as the customer/user, wanted from the software in terms of graphics and visual capabilities. After this 4th/current sprint, we have finally connected the GUI (display/platform for user interaction) with our backend of game mechanics (board and piece movements that deal with the actual algorithmic implementations of the rules of chess).*

*At the end of this 4th sprint, we do indeed have completed/functional “working software”; however, this version of the software is not as functional in terms of GUI and game mechanics connection, as we had hoped for it to be by this point. This sprint, we did not complete the user stories that dealt with the visual movement of the pieces from one square to another on the visual board. We have the backend set-up so that it registers the clicks/desired movements of the user/player, but it is not yet programmed to support/display graphically a full game of chess. We faced many challenged when writing our code to connect these two subsystems. We were able to get the board to register all clicks/desired moves to be made on the board. We were also able to get the program to check the legality of the moves attempted, and notify the player if an attempted move was actually illegal. We were also able to get our application to take turns for which player’s moves would be logged next. However, we had a lot of trouble getting the display of the pieces moving on the board to follow the moves being attempted. We may have to refactor our code for the GUI during the next sprint to be able to support this extremely important functionality.*

*Before our next sprint, we plan to consult our customer and show them our interactive graphical display. We hope that in doing so, we will be able to gain an idea of the customer’s approval/disapproval of the development thus far. Also, the customer will finally have the chance to interact with the GUI physically, which will make them/us more confident in our near-future final-product delivery. 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 solid final understanding of the customer’s expectations before going into our final sprint.*

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

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

SUBSYSTEM 1 – CHESS GUI:

* As a player

I want an automated kibitzer,

So that I can pretend I am in a noisy environment.

STORY POINTS: 8

* 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: 2

* As a player

I want for all pieces currently out of play for each player, to be listed on the side panel next to the game board

     So that I can easily assess the progress of each game.

STORY POINTS: 4

* As a player

I want the title screen of the game to read “LaboonChess”

           So that I know that the correct program has been opened.

STORY POINTS: 1

SUBSYSTEM 2 – GAME MECHANICS:

* As a player

I want a chess game that follows FIDE regulations for Standard American Chess

So that I have an accurate chess simulator that abides by national tournament standards.

STORY POINTS: 8

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

So that illegal moves are blocked from occurring.

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: 8

* As a player

I want it to be apparently visible when a king is in check

So that I can see if a player could potentially lose.

STORY POINTS: 2

SUBSYSTEM 3 – BACKEND:

* As a player

I want to be able to save and load games

So that I can pause a game and continue playing where I had left off, whenever I desire.

STORY POINTS: 8

* As a player

I want to be prompted with the option to start a new game or load a previous one; with the default being to start a new game

So that I can go directly to a new game if I have no old ones saved.

STORY POINTS: 4

* As a player

I want to save the game as a text file in .pgn format (portable game notation)

So that I can copy and load the file elsewhere if I decide to play on another machine.

STORY POINTS: 8

* As a player

I want to check if the .pgn file is valid

So that the program does not throw exceptions when trying to read from it.

STORY POINTS: 1

TOTAL VELOCITY = 62

**LINK TO CODE ON GITHUB**

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

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

**LISTING OF DEFECTS**

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

SUMMARY:

The Stockfish engine would crash whenever opened.

DESCRIPTION:

The Stockfish engine adds functionality to the program by generating the best possible move for the computer to make.

REPRODUCTION STEPS:

1. Run “gradle test”
2. Observe failing test for StockfishTest.java

EXPECTED BEHAVIOR:

The test should not fail; the engine should start properly.

OBSERVED BEHAVIOR:

The engine would not start properly and thus was unusable.

NOTES:

This defect was fixed by editing the implementation of startEngine to first check what operating system the program was being run on. Then, the correct version of the engine was launched for that system. After this change, the defect was fixed.

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

SUMMARY:

The program can export data to a file but it cannot properly import data and populate the board.

DESCRIPTION:

When trying to load a saved game, the program can read in the file, but it cannot populate a graphical board with the given information.

REPRODUCTION STEPS:

1. Run “gradle run”
2. Attempt to click load game and resume a saved game

EXPECTED BEHAVIOR:

The program should load the saved game and populate the graphical chess board with all of the appropriate pieces in their exact old positions.

OBSERVED BEHAVIOR:

The board does not populate with the old positions; stays in “New Game” or current game’s format.

NOTES:

This defect was the result of a discrepancy between the output that the Stockfish engine gives back after making a move, and the input that it requires to make a move. A workaround for this problem was reached- where every move that was made to get up to the saved point is displayed on the terminal. While working off these individual saved points, we were able to properly edit our functions to implement a fully functioning load game that did in fact populate the graphical chess board appropriately.

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

SUMMARY:

The program can save a game partway through a game, but it cannot perform multiple tasks- cannot consecutively save a game and then also load a new game.

DESCRIPTION:

If one attempts to save a game and load a game consecutively in the same session, the program will throw an java.io.IOException: Broken pipe in addition to several more exceptions being thrown in JCL classes.

REPRODUCTION STEPS:

1. Run “gradle run”
2. Attempt to save the game; click “Save Game” button
3. Attempt to load a game; click “Load Game” button

EXPECTED BEHAVIOR:

The program should be able to save and load multiple times during the same session, without having to exit and restart the program between each action execution.

OBSERVED BEHAVIOR:

Program can only properly execute one action- save or load, per session.

NOTES:

This defect unfortunately cannot be fixed at this moment. However, we were able to recover from the error and allow the game to continue running without crashing. For the remainder of the session though, no saves or loads can be made.

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

SUMMARY:

The program can identify a “check” condition properly, but it cannot identify any “checkmate” conditions.

DESCRIPTION:

During a game, when a “checkmate” is reached, every attempt at a next move is rejected because the king is in a “check” condition no matter where they move (the program does not allow a player to deliberately move their pieces in a way that will result in immediate subsequent death of their king piece). The program, however, has no means of informing the user that “checkmate” has been reached.

REPRODUCTION STEPS:

1. Run “gradle run”
2. Reach a checkmate condition

EXPECTED BEHAVIOR:

The program should inform the user that either they or the computer are in “checkmate” and thus that the game is over.

OBSERVED BEHAVIOR:

Every attempted move is rejected and the user is forced to infer that checkmate has been reached.

NOTES:

We attempted to correct this defect by using the Stockfish engine to generate all legal moves. This, however, proved to be a poor approach as we were not able to modify our system to accept the output that Stockfish provided us with.

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

SUMMARY:

Colors for pieces were stored as booleans.

DESCRIPTION:

The representation of piece colors as booleans allowed for a simple implementation, but it also led to technical debt during later sprints.

REPRODUCTION STEPS:

1. Referring to source code

EXPECTED BEHAVIOR:

N/A

OBSERVED BEHAVIOR:

N/A

NOTES:

We had originally written our colors as booleans to easily keep track of black and white pieces. This, however, proved to be a poor design decision. As our program grew more complex, it became difficult to keep track of several other colors/boolean variables. Looking back, it would have made more sense to use an Enum or separate color class.

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

SUMMARY:

Several constant values are duplicated repeatedly throughout the various classes.

DESCRIPTION:

There exists a large number of variables that are final constants. These constants are the same and stay unchanged.

REPRODUCTION STEPS:

1. Referring to source code

EXPECTED BEHAVIOR:

N/A

OBSERVED BEHAVIOR:

N/A

NOTES:

This defect was a result of incorrect planning when designing classes. Many constant values are redefined across several classes. It would have made more logical sense to keep a static class of constants that would be used consistently across the various classes in our program.

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

SUMMARY:

When running test cases, the PlayerTest.java test file detects a defect where a popup alert message temporarily blocks the running program.

DESCRIPTION:

The program will not continue to test or build until “OK” button is clicked on the popup alert window.

REPRODUCTION STEPS:

1. Run “gradle test”

EXPECTED BEHAVIOR:

The tests should run without any interruption.

OBSERVED BEHAVIOR:

The tests cannot continue to run until “OK” button is clicked.

NOTES:

This defect is related to our implementation of checking for illegal moves and checking if a player’s king is in check. By constantly checking for these conditions throughout the entire run of the program, we have resulted in error messages occurring before we even want them to be checked for. Due to this incorrect implementation, we were unable to change the way that the tests function without first changing the methods that we were testing to no longer output warnings as popups.

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