Charlie Mietzner (Scrum Master)

Ritika Maknoor

Peter Stamos

Rohan Patel

29 September 2016

**CS 1530 – SPRINT 1 DELIVERABLE**

**BACKLOG OF USER STORIES**

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

SUBSYSTEM 1 – CHESS GUI:

* As a player

I want to play a graphical version of chess

So that I do not need to carry a board with me everywhere.

* As a player

I want a chess interface that is intuitive

So that I can easily play without the need to learn additional unnecessary game mechanics.

* As a player

I want a chessboard and chess pieces that accurately resemble that of their physical counterpart

So that I can easily distinguish both the chessboard and each chess piece on the board.

* As a player

           I want this game to run on a computer; not on mobile

           So that I can see the game on a larger screen.

* As a player

           I want the game to execute a jar file from the command line

           So that the program is easy to execute.

* As a player

           I want white, and only white, to go first

           So that the program adheres to chess rules.

* As a player

           I want to choose my player color and the AI difficulty

           So that I can customize my game experience.

* As a player

I want black pieces to be at the top of the board and white pieces to be at the bottom

           So that the game follows standard rules.

* As a player

           I want the rank and file to be displayed around the board

           So that I can see what location a piece is at.

* 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.

* 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.

* 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.

* 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.

* As a player

I want a 2-dimensional board

           So that I do not see the pieces at an angle.

* As a player

I want it to have a tutorial when the application is first loaded

           So that I can learn the rules of chess if I do not already know them.

* 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.

* As a player

I want for there to be a section for all pieces currently out of play, on the side panel next to the game board

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

* As a player

I want to time my games

So that I can see how long my moves take.

SUBSYSTEM 2 – GAME MECHANICS:

1. 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.

1. As a player

           I want I the game to end once checkmate is reached

           So that I can start a new game.

1. 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.

1. As a player

          I want no time limit on turns

So that I can take my time to think about moves.

1. As a player

I want to confirm that I want to make a move

So that I do not accidentally make the wrong move.

1. As a player

           I want to be able to undo a move

           So that I can undo/ not be penalized for accidental clicks.

1. As a player

           I want the game to suggest moves

           So that I can have options when I cannot see open moves.

1. As a player

I want an option to choose the game’s level of difficulty

So that I can improve my skill in the game.

1. 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.

1. As a player

           I want achievements

           So that I, as a player, can be motivated to continue playing and improve further.

SUBSYSTEM 3 – BACKEND:

* As a player

I want to play against a computer

So that I can simulate playing against a person.

* As a player

           I want the game to run on the JVM

           So that it can be portable.

* As a player

           I want the application to run on a 512mb machine

           So that it can run on a variety of systems.

* 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.

* 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.

* As a player

           I want the ability to start a new game even when I have one already running

           So that I can be playing multiple games simultaneously.

* 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.

* 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.

* As a player

I want consistent levels of difficulty during a game

So that the gameplay remains constant.

* As a player

I want a running timer to track the duration of the game

So that I can easily assess how much time has passed since the game began.

* As a player

           I want the system to determine my rating as a chess player

           So that I can see my overall progress.

**WALKING SKELETON**

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

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

**TEST & BRANCHING/INTEGRATION PLAN DESCRIPTION**

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

To ensure that our chess application meets the needs of our customer, we have to develop it in a way that guarantees that it looks and performs as the user initially envisioned it to. This includes following the customer’s initial set of requirements as well as ensuring that the application is thoroughly tested to prevent unintended conflicts. This testing plan will encompass both functional and nonfunctional methods of testing. Since the customer will be relying on our group to deliver software that functions are they intended for it to, the software must be reliable and be capable of handling both intended and unintended uses. Additionally, the program should be scalable without compromising performance.

Our software should be reliable because we do not want it to crash or throw unexpected errors. This would be unpleasant for the user and thus will negatively impact faith in the software. Our software should also be scalable because we do not want to limit the use of it to any one particular group of machines. By limiting the number of machines, we run the risk of alienating certain users. We can test for both of these attributes (non-functional requirements) by creating test cases that test for proper functionality of our application when edge cases occur (such as a lag in the AI and thus a delayed display for the user, or opening the application on different machines with different versions of Java). Our list of test cases in our test plan should be both specific and all-inclusive. They will test each requirement/user-story with multiple scenarios- both when expected and unexpected behavior/user interaction occurs.

As more features and advanced algorithms are implemented into the application, we will need to ensure that performance is not compromised while the user plays the game. In order to have the computer determine which moves to perform in response to the user’s last movement, the software will have to consist of numerous algorithms that will check all potential valid movements on the chessboard to determine which move is most suitable for each specific circumstance. Due to the myriad of potential movements calculations that must be made by the computer with each passing turn, there is a greater risk for a decline in performance. Therefore, we must ensure that we develop algorithms that minimize the execution time of the computer so that it does not take more than 10 seconds to make a move. Doing so will allow for a more pleasant and responsive experience for the customer.

Our team’s overall approach to branching will be to practice continuous integration with the master branch. We want to limit extended periods of development on a branch, as the differences between a development branch and the master branch will inevitably accumulate the longer that development on that branch continues on--increasing the potential for merge conflicts and the effort expended in resolving them.

With this in mind, branches will only be created for a user-story/feature and named for that user-story/feature. These branches should be focused on one feature that is manageable by one person in a short amount of time--the branch should *not* contain the entirety of a team member’s development during the course of a sprint. If the user-story is large or encompasses several features, it should be broken into smaller features and branches should only be created for those smaller features.

When development of a feature branch is done, the team member must submit a pull request for the branch before it can be merged with the master branch. When submitting a pull request, the team member must choose an assignee, or assignees, for the pull request other than themselves. The pull request assignee will review the changes, ensure that they work, and approve the merge with the master branch. The feature branch can then be deleted.

This plan is, obviously, subject to change over the course of development. However, we feel it keeps branch maintenance relatively simple and reinforces Agile’s philosophy of *kaizen*--whereby each short branch merged is a continual improvement of the overall project.

**USER STORIES/DESCISIONS DESCRIPTION**

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

Our initial discussions with the customer provided us with a list of requirements that they had in mind for the software. They began by stating a few very basic and vague requirements such as “I want to play chess”. After going back and forth, asking questions to gain more detailed and specific information that we thought would be pertinent for the development of the product, we ended up with a long master list of the user’s requirements. At the time of discussion, the user expressed which requirements they viewed as being absolutely necessary/vital, and which they viewed as being of lower importance/could instead be features added on later. We then prioritized the master list- those that the user indicated as being imperative were noted as being of highest priority and were placed at the top of our list.

Once the master list was prioritized by how much the user expressed each requirement’s importance/how often they mentioned the specific requirement, our development team then sat as a group and went through them all again. We decided that our overall system had 3 apparent categories/subsystems- Chess GUI; Game Mechanics; Backend. We once again filtered through the master list of all requirements; this time, we divided them up into these 3 categories.

* Chess GUI: Includes all requirements that deal with the actual interface of the game; specifics about display; board and pieces.
* Game Mechanics: Includes all requirements that deal with the actual algorithmic implementation of the rules of chess; rules and restrictions.
* Backend: Includes all requirements that deal with the programs functionalities not directly visible to the user; computer as opponent; stack trace; possible errors; saving and reloading games.

Our master list had a total of 50 requirements. Once divided up by category, each subsystem then had an average of 13 requirements in it.

We decided then that for our first sprint, we should focus on a set of requirements that covered all of the bases of our system- the 3 subsystems. We felt as though a large majority of our requirements were vital to the user. We had difficulty at this point narrowing down a set of user stories to focus on for this sprint. We knew, however, that we could not take on the responsibility of accomplishing all of the user stories that seemed to be vital components of our final product, in just our first sprint. We knew that according to Agile/Scrum methodology, we needed to have working software at the end of each sprint, rather than start working on the production of all of the needed components at once and having nothing truly complete and working at the end of our sprint. Therefore, after a lot of discussion over which user stories seemed to be the most logical and important to begin with in our first sprint, we put together a group of 11. We chose 4 user stories that had high priority in our Chess GUI subsystem, 3 that had high priority in our Game Mechanics subsystem, and 4 that had high priority in our Backend subsystem.

* Chess GUI:
  + As a player

I want a chess interface that is intuitive

So that I can easily play without the need to learn additional unnecessary game mechanics.

* + As a player

I want a chessboard and chess pieces that accurately resemble that of their physical counterpart

So that I can easily distinguish both the chessboard and each chess piece on the board.

* + 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.

* + As a player

I want the rank and file to be displayed around the board

So that I can see what location a piece is at.

* 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.

* + As a player

I want I the game to end once checkmate is reached

So that I can start a new game.

* + 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.

* Backend:
  + As a player

I want to play against a computer

So that I can simulate playing against a person.

* + 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.

* + 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.

* + As a player

I want a running timer to track the duration of the game

So that I can easily assess how much time has passed since the game began.

Once we discussed with the user and finalized the requirements list, we had to begin doing preliminary research on our domain. We decided that we will be using Gradle as our build tool. We found that Gradle would be best because of its flexibility, and powerful set of included tools. In addition, through our research, we found that Gradle appears to be the build tool of choice for the industry as a whole due to its compatibility with a wide variety of programming languages and its powerful Groovy language.

We decided on using Swing as our API because of its wide variety of standard components and features. Also, it has shown to have good performance and stability across platforms. Through our research, we have found that it works well for creating GUI’s, and that is a main component of our application.

We also gathered some research on the basic rules of chess, outlines by the FIDE regulations for Standard American Chess. In chess, the standard board is 8 by 8; 8 rows called “ranks”, numbered 1 through 8 and 8 columns called “files”, lettered a through h. On a standard chess board, white is on the bottom and black is on the top, with white making the first move of the game each and every time. Each player starts with 16 pieces on board; 1 king, 1 queen, 2 rooks, 2 knights, 2 bishops, 8 pawns. Players can move one piece per turn; an exception to this rule includes castling. When a pawn reaches the 8th rank, it is promoted to its player’s choice of higher piece (queen, bishop, knight, or rook). If a player’s king is taken, they lose and the game is over.

This first sprint actually went smoothly for our group. Our sprint planning session was very effective. We discussed which user stories were important to focus on during this first sprint. However, we did not divide them up as effectively as we could have. Throughout the entire sprint, we all contributed and worked together on each story and portion of work (which was helpful, especially as it was our first ever sprint); however, for upcoming sprints, we should be sure to clearly divide up the work and stories during the sprint planning session. As a group, we were all very communicative with any concerns we felt at any point. We were all very helpful/respectful and open to suggestions.

**PAPER PROTOTYPE DESCRIPTION**

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

When the user initializes the chess application, a central interface containing: a series of game options, chessboard, chess pieces, game timer, and game log, will appear. Three options are displayed at the top left corner of the interface that allow the user to either start a new game, save their current game’s data, or load a previously saved game. Directly under these options, the user will see an interactive 8 by 8 standard chessboard grid. In addition, they will also see a series of numeric (1-8) and alphabetical (A-H) characters along the western and southern borders of the board. These characters along the axes will help indicate specific grid locations on the chessboard.

Upon the start of a new game, chess pieces will be set into their standard initial positions on the chessboard; as the game progresses, these pieces can then be dragged/moved by the user to other positions/squares on the board. On the right-hand side of the interface a timer is displayed. This timer can be either initialized when the user begins a new game, or resumed when the user loads a previously saved game. This timer will allow the user to track the duration of their currently running game. Directly under the timer is a game log. This game log will display which chess pieces are out of play (both the user and computer’s taken pieces), and will have a list of all prior moves that have been made during the current game (both the user and computer’s moves). Black chess pieces that are out of play will be displayed in the upper half of the game log while white chess pieces that are out of play will be displayed in the lower half of the game log.

Once a game has reached completion, it will display a message informing the user if they have won or lost the game. If the user has beaten the computer, the user will be prompted to enter their name. This data, along with the user’s completion time, will be logged to a scoreboard that will display their 10 best completion times. The game will then reset all data from the previous game and initialize a new game.