Maigan Davey, Seiichi Nagai, Jeremy Pasimio

Project 1: Checkers Game

Iteration 6: Summary Report

1 Objectives and Testing Goals

Our team was given a checkers game application with seeded bugs, overly complex methods and little to no documentation and testing data. Our goal was to debug the program and test the entire system to prove that the system is fully functional and efficient. We used unit testing, integration testing, system testing and acceptance testing strategies and have constructed specific test cases to fulfill each of the testing strategies already mentioned to insure that we have tested the system to it's entirety.

2 Significant Challenges

There were a few challenges that we were confronted with during the extent of this project.
One challenge we were confronted with as a team was figuring out how the Checkers application was suppose to function, since the program given to us had many bugs and unnecessary code that we had to discover.

A second challenge was that we were not the original developers of the Checkers application; therefore some of the internal functionality of the program was not 100% coherent for us when we first began, and still some was a little unclear toward the end.

A third challenge we had was our unfamiliarity of the rules of checkers. This was a small setback; however because we were able to research and learn all of the ins and outs of the game of Checkers.

One last challenge among others, was adapting to a completely virtual team. Luckily we were all in the same time zone and scheduling work time was not a huge issue; however we did have to work with each other and notify one another if we were working on a certain section, so that no one was doing unnecessary work.

Maigan Davey, Seiichi Nagai, Jeremy Pasimio

3 Testing Results

3.1 Unit Testing

For the unit test cases, we looked at the methods in the program individually. It was necessary to insure that each of the individual methods did their own duties bug free before looking at the entire system as a whole. With that being said we tested each method individually by looking at the data that was used in that method and the outputs that the method would return. Gathering that data allowed us to record what the expected results should be, then we compared these expected results to the actual results and altered the method based on how the results are delivered. This process is known as white box testing because are testing the internal structures of the system and not the functionality at this step.

We found that most of our unit test cases passed, we had one of our unit test cases fail, this was because the undo() method in Checkers.java, was not accounting for the first play and not allowing the undo() function until the player had made two moves. Overall our unit testing was extremely successful. Our unit tests were able to achieve 33.8% code coverage. Our main goal with unit testing was to focus on the methods that focused on the logic of the game so we ignored classes and methods that were primarily focused on setup or updating of the GUI. We felt that system testing would be sufficient for testing GUI functionality given the scale of the project. In regard to the tests we were able to finish, our low coverage percentage was simply due to the limited amount of time for writing tests. The primary methods we focused on for calculating moves and board states involved many variables (Piece color, piece type, which color was to move, etc.) which resulted in too many paths for us to get complete path coverage in our tests.

3.2 Integration Testing

In the integration testing phase, we took the individual methods that we harshly tested individually in the unit testing phase and combined their functions. In order for our checkers application to run properly, all of the methods need to work together, because they use each other to reach the extent of the game known as checkers. For integration testing we gathered the methods that use each other and ran test cases on those groups of methods. Once the group or groups of methods were considered bug free, we moved on to the next phase. For integration testing it was made a point that all buttons be tested for class call functionality, it is therefore appropriate to state that complete component coverage was achieved.

Maigan Davey, Seiichi Nagai, Jeremy Pasimio

3.3 System Testing

System testing is the phase of testing where the integrated software that is considered complete is tested together. There are many different types of system testing, the following are the types that we used in our testing of the checkers program:

Performance Testing: For performance testing, we made sure that the system knew exactly how to respond to unexpected actions. The following are types of performance testing:

Quality Testing: For this, we tested for the reliability, maintainability and availability of the system. This means that we made sure that throughout the entire game of checkers the rules are always kept in check. The quality of the moves and images and sounds will always be maintained and never go out for an unknown reason.

Human Factors testing: For this we tested with end users. For the sake of this assignment, we were the end users. But the reason for this testing is to insure that all of the buttons have a clear purpose and it makes sense to have them on the screen. Along with that, the "how to" file will need to be explicitly clear, so there is no question that the end user understands the rules of the game.

Overall we confirmed that the system as a whole provides the expected functionality. We used the GUI heavily in System testing and ensured that all buttons and combinations of buttons/selections work appropriately via the GUI. It is therefore reasonable to state that complete coverage was achieved in System Testing.

3.4 Acceptance Testing

Acceptance testing was our final step in testing and was used to demonstrate that our checkers program is fully functional and ready for friendly checker matches.

To accomplish this, we needed to validate that all of our functions work as they are supposed to. Since this was the last phase, our goal was to create this summary report and list any bugs that we did not end up addressing.

4 Summary of Bug Fixes

4.1 Bug Fix 1 - Consecutive Undo

During our initial evaluation of the code, it was assumed the undo feature was set up to facilitate undoing consecutive moves (up to 3) in the game. Once the program was functioning properly and the undo button would only work for one move it was assumed that this was a bug. Upon closer evaluation of how the code was designed it was determined that only allowing

the undoing of a single move rather than consecutive moves was the original design of the program and our initial evaluation was faulty. The code has been left as is.

4.2 Bug Fix 2- Empty help window

The program was currently not calling an text file because it didn't exist. So, a new text file was created with full directions. Graphics were also added and the code was implemented in the Help class.
**See Bug Fix 2 for additional details.

4.3 Bug Fix 3- Availability of Undo function

The type of testing associated with this bug was Unit testing because the bug was ultimately occurring in the undo() method within Checkers.java. The Undo() method was only usable after the player had made two or more plays/moves. This was considered a bug because the undo() functionality should be available as long as any play has been made, so one or more plays/moves. To fix this bug, we looked at the variable undoCount, and came across the error. The variable was made to represent the wrong integer value causing it to continually skip over the first play and only allow the undo button to be used after two or more plays/moves.

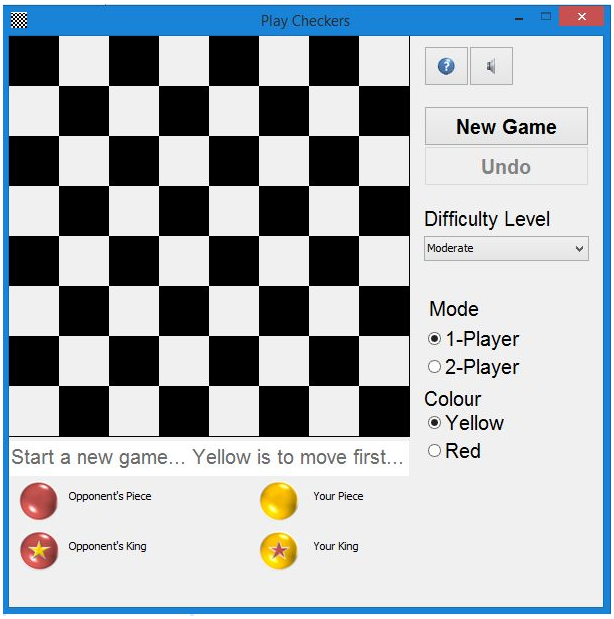
4.4 Bug Fix 4- Label change with radio button selection

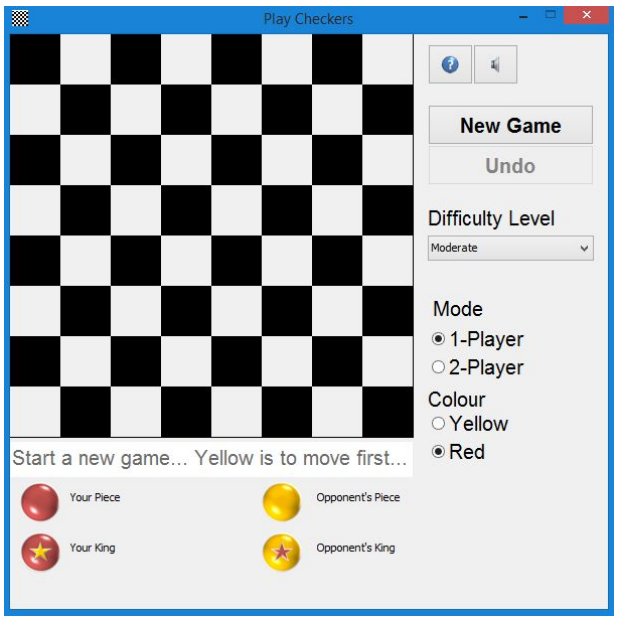
The labels next to the piece graphics were not changing with the corresponding radio button selection. Now, when the 'yellow' button is selected, the corresponding labels indicate that you are the yellow team. Similarly, when the 'red' button is selected, the labels indicate that you are the red team. When the '2-Player' button is selected, labels now indicate who is player 1 and who is player 2. This was accomplished by implementing the label changes in the appropriate radio button actionlisteners.

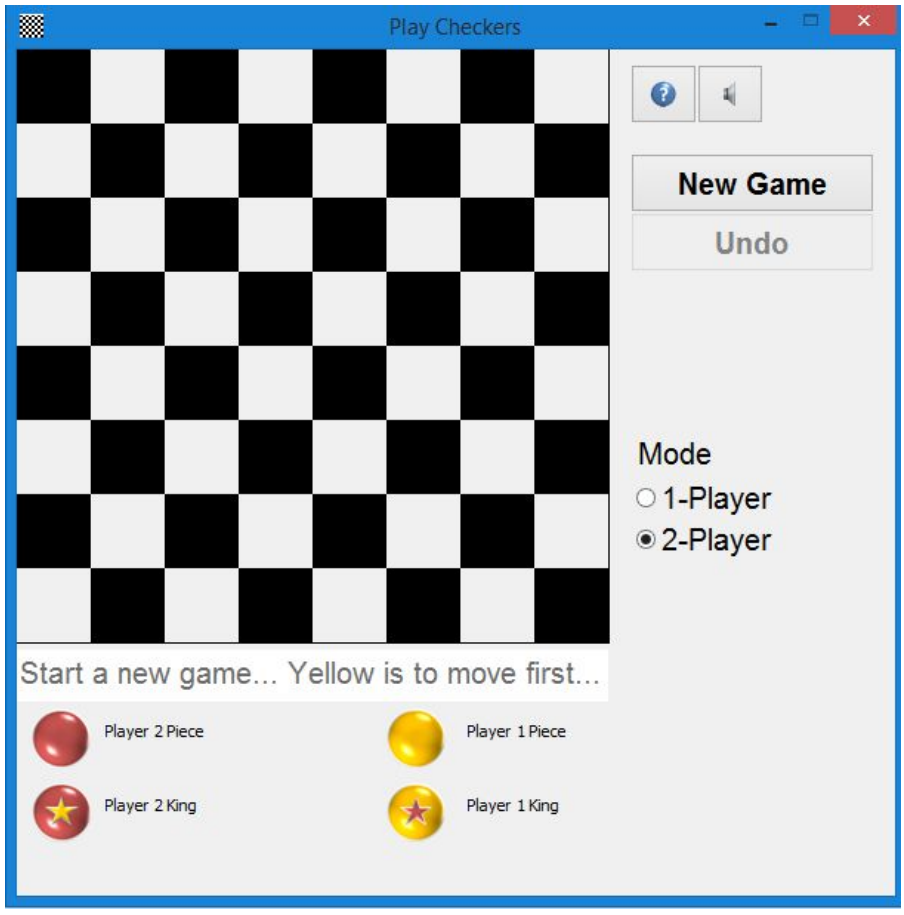
This fix had a very low-level impact on overall code. Only the Checkers class was changed and the implementation is not called from any other classes. Testing of the fix include selecting each radio button separately and making sure the labels change accordingly. Full coverage was accomplished.

Screenshots of the fix follow:

Maigan Davey, Seiichi Nagai, Jeremy Pasimio







Maigan Davey, Seiichi Nagai, Jeremy Pasimio

5 Summary of Enhancements

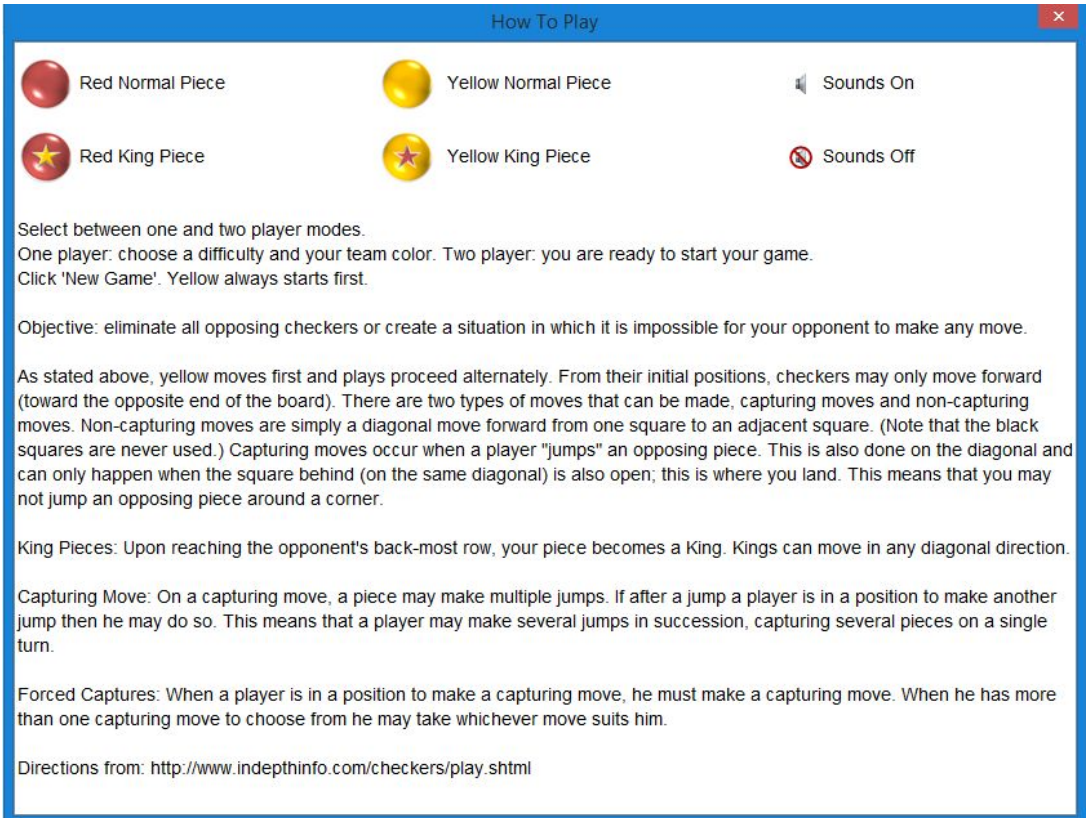
4.1 Enhancement 1- Gameplay directions unknown

Before this fix, no gameplay directions were present. Checkers directions are universal and were the same across almost all sources. The directions written in the 'How To Play' window are taken from the following website:

http://www.indepthinfo.com/checkers/play.shtml

These directions, along with directions on operating the UI were placed in the text file, "HowToPlay.txt". This enhancement caused the creation of a new .txt file being added to the project as well as a minor code change in the Help class. Overall this was a low level impact change. The change consisted of adding the .txt file as well as adding graphics to the "HowToPlay" window. Testing this enhancement consisted of making sure the integration of the txt file and the Help class functioned appropriately and that all graphics and text appeared on the window. Full testing coverage for this fix was achieved.

An image of the newly created 'How to Play' follows:



Maigan Davey, Seiichi Nagai, Jeremy Pasimio

6 What We Learned

This project was extremely insightful and we learned, as a group, a large amount. First we learned the severity of communication when working in an online team. It is necessary to continually check your form of communication in order to always be on the same page. Along with communication, being in an online team is about learning from one another and building off of each other's knowledge.

Second, we were able to become more comfortable with the various types of testing: unit testing, integration testing, system testing, and acceptance testing. We were able to use these testing strategies in a real life context while working with the checkers application and in turn were able to apply what we read about in class.

Third, we learned that developing software is very reliant on the users experience and making sure that everything is easy and efficient for the user. While many of the assignments in this class had detailed specifications, doing the bare minimum to cover the specifications is not always efficient enough and it is necessary to put yourself in the shoes of the user in order to make the system fully functioning and efficient.

7 What We Would Improve

If we were to do this project again there are a few areas that we could adapt. First we communicated through email and through Google Hangout (a chatbox), to be even more efficient with our meeting time, we could use skype and do voice or video calls. This would allow us to collaborate faster and we would be able to have real time conversations. With that being said, our current approach was very successful.

Another adaptation we could implement is that we could develop a schedule for when we met every week. For our project we simply worked when we could and would leave notes or emails to each other so that the others could pick up where one left of. We could have decided on a specific time and met at that time to work together. It is important to mention, that while we did not set a specific time, we all coincidentally would end up on the google docs at the same time and would build off each other when this was the case.