Four Row Solitaire

*Iteration 6: Implementation of bug fixes and enhancements*

# 1 Objectives and Testing Goals

The objective of this project is to review the Four Row Solitaire game, and perform software maintenance activities, fixing any faults found and implementing reasonable enhancements.

We approached the testing of this program with the idea that, since it’s a game, the vast majority of our testing should be focused on making sure that the user experience is as close to flawless as possible. A game is meant to be played, and as such, the way the underlying code works doesn’t necessarily matter as much as making sure that the gameplay is experienced by the user as intended. To that end, we chose to focus our efforts on more black box functionality tests rather than granular, line-by-line unit tests, because ultimately, if a line of code doesn’t get executed but the game plays like it’s supposed to, that missed line of code doesn’t matter to the user. Our end goal as a team was to have a product with as polished a gameplay experience as possible within the timeframe we had to work on it.

# 2 Testing Results

**2.1 Unit Testing**

Only one major Unit testing file was implemented – CardMoveTest.java – this file was implemented due to faults discovered in the basic card movement involved in the game. An extra method for testing was implemented in Deck.java to support creating a pre-determined deck to allow testing of various movement combinations. Various bugs were discovered by CardMoveTest.java and fixes were implemented.

**2.2 Integration & System Testing**

Note: To aid in testing, a complete script was developed to test as much of the codebase as possible. This has been included with this submission as a document called Test Guide included with this document.

Fixes implemented were tested against both the CardMoveTest.java file, and the included Test Guide script to ensure integration of implemented fixes and enhancements. Also, code to enable dealing a consistent play-field to allow easy testing of end-conditions of the game, and to insure the playfield is set up for certain testing methods was implemented. A Boolean trigger was added to Deck.java to enable and disable this testing environment. When merged, coverage from the execution of the Test Guide script and the CardMoveTest.java unit tests enabled us to achieve a 77% code coverage rate.

**2.3 Regression Testing**

Code was verified against CardMoveTest.java to ensure basic functionality was not affected by any program changes.

**2.4 Manual Black Box Testing**

Black box testing was used to test much of the GUI functionality that was difficult to test through automated processes. This would include both Game Menu, GM-XXX, test cases and Help Menu, HM-XXX, test cases. A manual test template was made in excel where the tester would follow each step provided and mark the results with a PASS/FAIL dropdown option to the right of the verification step. Any comments can be placed under the comments section for each test case covered. Once the test is completed the tester will then save their results under a different name. The template that was created was SolitaireBlackBoxTemplate.xlsx. Our black box test run results were saved in the excel spreadsheet called SolitaireBlackBoxTestRun.xlsx. These files have also been included with this document.

# 3 Summary of Bug Fixes

A complete issue tracker for this project can be found here:

<https://github.com/snesmith/Online-Team-9/issues>

3.1

Cards were not able to be stacked. This was most the critical fault encountered, and triggered the creation of the Unit Test file CardMoveTest.java. This unit test allowed us to identify additional problems involved with moving cards. Changes to various files to add missing isValidMove() methods to ensure valid moves were marked as such.

3.2

Kings were not able to be moved to empty columns, this was discovered by the unit test prompted by bug 3.1. Fix was added to mark KING rank cards as valid to be moved to an empty column.

3.3

Adding a non-ace card to an Ace pile triggered a nullpointerexception. This was also discovered by the CardMoveTest.java tests – fix was added to ensure that non-Ace cards were indicated as invalid moves when placed on an Ace pile.

3.4

Cards mis-identified. The cards with rank 5 were mis-identified as rank 6. An invalid constant was corrected. There was one instance of another card that appeared to be mis-identified, but the problem was unable to be replicated and may have been a spurious error.

3.5

Card clicking is erratic and often difficult. This bug was classified as an issue that time constraints would not allow a proper fix for. It was suggested that this problem is triggered by the single-threaded nature of the game, and that when the game was busy with another task in the background, or the Java VM was busy elsewhere, that mouse events would not be collected. There also seemed to be an issue when the mouse moved between the click and the release. Rewriting the program to allow UI event handlers to be run in a thread separate from the underlying code was determined to be outside the time constraints.

3.6

A console error was identified when playing sounds was potentially related to an underlying Java bug here: <http://bugs.java.com/bugdatabase/view_bug.do?bug_id=6790382> - as such no resolution was determined to be possible at this time.

# 4 Summary of Enhancements

**4.1**

A confirmation was added on starting a new game so that statistics could be tracked properly, and give the player an opportunity to return to the game and try additional moves. This involved adjusting the new game code to test if any moves have been made, and trigger a possible indicator of a game loss if any moves had been made.

**4.2**

Statistics were enhanced to track starting a new game, and having it be classified as a lost game if any moves had been made.

**4.3**

Squares were added to the playfield to allow a player to identify and easier select the empty columns to place a King. This was an entirely cosmetic issue.

# 5 Significant Challenges

Significant issues have been encountered in developing and implementing automated unit tests for UI elements. This impacted the ability to address the erratic card selection and event handler detection, preventing us from being able to spend any truly significant amount of time addressing this issue. Another major issue encountered was the complete absence of one of the team members for any of the group activities. This increased the work load placed upon the other team members, and cut down on the amount of time we were able to spend working on enhancements.

More time during the class exploring automated GUI testing methods would have assisted in being able to properly address GUI issues encountered in the project. This is the major opportunity that has been identified amongst team members as something that would be extremely useful in future endeavors and projects. The lack of foundation in this area is a potential cause for concern.

The greatest non-programming challenges for our team were communication and time management. In an online-only scenario with a group of people with differing schedules and priorities throughout the week, it took extra effort to make sure that we were all on the same page as we progressed through the project. Fortunately, I think we were fairly successful.

In terms of communication, we utilized a flexible online chat system that could be accessed from our computers and mobile devices to make sure we could be reached as easily as possible. Given our work schedules, it wasn’t uncommon for us to be exchanging notes late into the evenings and over the weekends. This posed a particularly steep challenge when we needed feedback from other group members on individual work, as sometimes you would complete something then have to wait hours or days for feedback before being able to proceed.

The final point above ties into the second major challenge we experienced, which was time management. Because communication was spotty at times and the amount of time required to complete portions of the project were unknown, it was difficult to plan out exactly when to work on things as a team. This was exacerbated by the fact that we lost a team member in the first week of the project, so we were handling the workload of four people with only three.

Ultimately, the team was successful in terms communicating and of utilizing the time we had available to us. If it were an ideal situation, the team could have completed the project much more efficiently had both of those factors been improved upon. Thankfully the team was a pretty smart group and we didn’t really struggle with the actual work itself. There were just opportunities to tackle that work in a more effective manner.

# 6 What We Learned

We’ve learned that one of the key aspects to ensuring that team members keeps coordinated is an effective group messaging platform, and an established source code repository. Tasking should be better shared, and a more well-rounded approach would be advantageous. We’ve also learned that an effective way of achieving reliable code coverage is to assemble and collate a testing script to ensure that as much functionality is explored as possible. Also, when time constraints are an issue, by the book testing may not be practical. Thus, implementing a triage approach by classifying issues at different severities can allow limited time resources to be allocated where best used.

Another lesson learned is that throwing together an automated unit test can help in understanding the underlying code in ways that may not be apparent prior to trying to develop the unit test. It is in analyzing the potential failure points, and attempting to develop tests to expose them that underlying logic and structure of a program can be revealed. This also works in reverse, for the GUI aspects of the program, the inability to establish some sort of baseline automated failure test was a big decision factor in not expending effort in tracking down the erratic card selection behavior.

Lastly, we learned that the collaborative software process requires a lot of patience and, tying back into the previous section, communication. When there are so many moving parts, making sure that what everyone is doing what they’re supposed to be doing without affecting someone else’s work is key. Everyone’s personal software process is a little different. So frequent communication, notation, comments, version control, and most of all, patience are invaluable to make sure that a project doesn’t become completely unmanageable.

# 7 What We Would Improve

We would suggest improvements in developing a skillset to analyze a program not just in the underlying code – but in the exploration of techniques on how to automate unit testing for different program aspects: GUI, API interfaces, program logic, user security, etc…

Another area of opportunity is in team management, delegation, and assignment of tasks. Testing is not just developing tests, it is also in building a team to discover and address issues. A team is only as good as its members can be applied to a problem, and without the opportunity to discover the strengths and weaknesses of the members can lead to uneven distribution of effort.

The last thing we that we could have improved on is time management. Because of the complete ‘time crunch’ involved in projects of this nature in such shortened class sessions. Time management is critical, and an exploration of establishing a baseline time commitment, along with being able to practice enough testing methodology as possible would be immensely helpful in future iterations of this course.