Connect Four Game

*Iteration 6: Implementation of bug fixes and enhancements*

# 1 Objectives and Testing Goals

The Testing Goals for the Connect Four game support the following objectives:

* Define the activities required to prepare for Performance Testing.
* Uncover bugs and implement fixes
* Enhancing program performance

While it was wishful to complete all testing categories, the team decided not to pursue Unit Testing, Integration Testing, System Testing, and Regression testing due to time constraints.

# 2 Testing Results

The only test results available are Performance test results, which are shown below.

**2.1 Unit Testing**

N/A

**2.1 Integration Testing**

N/A

**2.1 System Testing**

N/A

**2.1 Performance Testing**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Test Scenario** | **Test Case** | **Test Steps** | **Test Data** | **Expected Result** | **Actual Result** |
| Build & Run Project | **TC1** | 1. Open the Connect Four project in Eclipse IDE 2. Build the project 3. Run the project | src package  Image package | Project built and ran with Main panel open | Project built and ran in console mode |
| Main panel opens successfully | **TC1** | (1) Open the Connect Four project in Eclipse IDE  (2) Build the project  (3) Run the project | N/A | Eclipse IDE opened the Connect Four project successfully | Eclipse IDE opened the Connect Four project successfully |
| Player 1’s default value | **TC2** | 1. Open the Connect Four project in Eclipse IDE 2. Run the project 3. Click the “**PLAY**” button | N/A | Player 1’s default value is **Player 1** | Player 1’s default value is **Player 1** |
| Player 2’s default value | **TC3** | 1. Open the Connect Four project in Eclipse IDE 2. Run the project 3. Click the “**PLAY**” button | N/A | Player 2’s default value is **Player 2** | Player 2’s default value is **Player 2** |
| Player 2’s value becomes “**Computer**” | **TC4** | 1. Open the Connect Four project in Eclipse IDE 2. Run the project 3. Click the “**Play Against Computer**” button | N/A | Player 2’s default value is **Computer** | Player 2’s default value is **Computer** |
| Main panel transitions to Game Play panel | **TC5** | 1. Open the Connect Four project in Eclipse IDE 2. Run the project 3. Click the “**PLAY**” button | N/A | The **Main** panel transitioned to the **Game Play** panel | The **Main** panel transitioned to the **Game Play** panel |
| Proper player chip position | **TC6** | 1. Open the Connect Four project in Eclipse IDE 2. Run the project 3. Click the “**PLAY**” button 4. Place the colored chips into the empty board array | N/A | The players’ chips dropped into the board from bottom row upward | The players’ chips dropped into the board from bottom row upward |
| Connect Four diagonally | **TC7** | 1. Open the Connect Four project in Eclipse IDE 2. Run the project 3. Click the “**PLAY**” button 4. Win the game by arranging for four chips of the same color to line up linearly and diagonally. | N/A | A winning condition detected. The Game Play panel transitions to the Game Over panel with Winner found as Player 1, or Player 2, or Computer | A winning condition detected. The Game Play panel transitions to the Game Over panel with Winner found as Player 1, or Player 2, or Computer |
| Connect Four horizontally | **TC8** | 1. Open the Connect Four project in Eclipse IDE 2. Run the project 3. Click the “**PLAY**” button 4. Win the game by arranging for four chips of the same color to line up linearly and horizontally. | N/A | A winning condition detected. The Game Play panel transitions to the Game Over panel with Winner found as Player 1, or Player 2, or Computer | A winning condition detected. The Game Play panel transitions to the Game Over panel with Winner found as Player 1, or Player 2, or Computer |
| Connect Four vertically | **TC9** | 1. Open the Connect Four project in Eclipse IDE 2. Run the project 3. Click the “**PLAY**” button 4. Win the game by arranging for four chips of the same color to line up linearly and vertically. | N/A | A winning condition detected. The Game Play panel transitions to the Game Over panel with Winner found as Player 1, or Player 2, or Computer | A winning condition detected. The Game Play panel transitions to the Game Over panel with Winner found as Player 1, or Player 2, or Computer |
| Alternating players’ turns | **TC10**  **TC11**  **TC12**  **TC13** | 1. Open the Connect Four project in Eclipse IDE 2. Run the project 3. Click the “**PLAY**” button 4. Win the game by arranging for four chips of the same color to line up linearly and horizontally, or vertically, or diagonally. | N/A | The players turns alternates when the “**Play Again**” button clicks. | The players turns alternates when the “**Play Again**” button clicks. |
| Dynamic “**Round number**” | **TC14** | 1. Open the Connect Four project in Eclipse IDE 2. Run the project 3. Click the “**PLAY**” button 4. Play the game by dropping circular colored chips onto board. | N/A | The “**Round number**” value is updated every time the players make their move. | The “**Round number**” value is updated every time the players make their move. |
| Win/Lose/Tie logic | **TC15** | 1. Open the Connect Four project in Eclipse IDE 2. Run the project 3. Click the “**PLAY**” button 4. Win the game by arranging for four chips of the same color to line up linearly and horizontally, or vertically, or diagonally. | N/A | The game statistics are updated correctly to reflect players’ winning scores | The game statistics are updated correctly to reflect players’ winning scores |
| Win/Lose/Tie logic | **TC15** | 1. Open the Connect Four project in Eclipse IDE 2. Run the project 3. Click the “**PLAY**” button 4. Play the game and arranging for both players, either human against human or human against computer, to end the game in a tie. | N/A | The Game Play panel transitions to the Game Over panel with a “**tie**” message | The game cleared the board and threw a null-pointer exception |
| Win/Lose/Tie logic | **TC15**  **TC16** | 1. Open the Connect Four project in Eclipse IDE 2. Run the project 3. Click the “**PLAY**” button 4. Win the game by arranging for four chips of the same color to line up linearly and horizontally, or vertically, or diagonally. | N/A | The game statistics are updated correctly to reflect players’ winning scores.  The Game Play panel paused a few seconds before transitioning to the Game Over panel | The game statistics are updated correctly to reflect players’ winning scores.  The Game Play panel transitioned to the Game Over panel immediately |
| Last panel loops back to first panel | **TC17** | 1. Open the Connect Four project in Eclipse IDE 2. Run the project 3. Click the “PLAY” button 4. Win the game by arranging for four chips of the same color to line up linearly and horizontally, or vertically, or diagonally. 5. Click the “**Main Menu**” button | N/A | The **Game Over** panel transition to the **Main** panel immediately. | The **Game Over** panel transition to the **Main** panel immediately. |
| Panel’s ability to minimize, maximize, and close | **TC18**  **TC19** | 1. Open the Connect Four project in Eclipse IDE 2. Run the project 3. In either Main, or Game Play, or Game Over panel, click on the minimize button, or the maximize button, or the exit button 4. Verify panel behavior | The minimize button | The panel minimizes | The panel minimizes |
| Panel’s ability to minimize, maximize, and close | **TC18**  **TC19** | 1. Open the Connect Four project in Eclipse IDE 2. Run the project 3. In either Main, or Game Play, or Game Over panel, click on the minimize button, or the maximize button, or the exit button 4. Verify panel behavior | The maximize button | The panel maximizes | The panel maximizes |
| Panel’s ability to minimize, maximize, and close | **TC18**  **TC19** | 1. Open the Connect Four project in Eclipse IDE 2. Run the project 3. In either Main, or Game Play, or Game Over panel, click on the minimize button, or the maximize button, or the exit button 4. Verify panel behavior | The exit button | The Connect Four program terminates | The Connect Four program terminates |

**2.1 Regression Testing**

N/A

# 3 Summary of Bug Fixes

Sub-section descriptions of individual bug fixes (one per enhancement) that were implemented. Include operational description of the bug fix, high level software impact of the bug fix to the code base, and amount/type of testing associated with the bug fix. Include screen shots if deemed desired/appropriate by the team.

**4.1 Bug Fix 1**

The GUI would not run when the project was first delivered. It was found that the image folder was not declared as a source folder, and thus was not able to be found by the program. After setting the image file as a source folder in Eclipse, the game was able to run the GUI mode. There was not much testing involved in this process. Once Dasha pointed out the problem, it was a quick fix and the game was operational.

**4.2 Bug Fix 2**

The game was unable to process a tie condition. When a tie happened, a null-pointer exception was thrown. The code made calls to methods in such a way that a variable was being referenced after it had been set to null. By changing the order of the calls, the exception was avoided and the game processed the tie condition correctly. Being just a matter or reordering some code, there was no real impact to the software. Testing consisted of performance testing, by forcing tie conditions to happen and verifying that the game over panel was reached with a message of a tie condition, and no player scores were updated.

**4.3 Bug Fix 3**

There was not pause before the game jumped to the game over panel when a winning condition was met. This made it difficult to verify that a win had actually occurred. We decided that the players of the game would like to be able to see the board before a win was declared. Utilizing the Thread.sleep() command, we added a five second pause before changing to the game over screen when a win occurs. We tested this by running the program and verifying that the game did indeed pause on the game play screen before jumping to the game over screen when a win occurred. The sleep command pauses the entire thread for 5000ms, which will also free up processing time during the pause.

# 4 Summary of Enhancements

Sub-section descriptions of individual enhancements (one per enhancement) that were implemented. Include operational description of the enhancements, high level software impact of the enhancement to the code base, and amount/type of testing associated with the enhancement. Include screen shots if deemed desired/appropriate by the team.

**4.1 Enhancement 1**

The game also did not pause when a tie condition was met. As the board must be filled to meet the condition of a tie, this was not a necessary function. We decided that for consistency, it would be beneficial to add the pause to the tie condition as well as the win condition. The same Thread.sleep() command was used with a five second delay coded in. We tested it in the same way, forcing a tie condition, and verifying that the game paused for five seconds before changing to the game over panel. This also has the same effect of pausing the entire thread for five seconds.

**4.2 Enhancement 2**

The game does not clearly identify the winning pieces when a win condition is met. The game does not support this feature at all at this point. It does not even store the winning pieces, so a bit of functionality would have to be built in. A way to save a reference to the 4 connected pieces, and a method to change their appearance in some way, either through a color change, or highlighting, would need to be coded in. Testing could be completed by verifying that the correct 4 pieces were highlighted when a win condition is met. The impact to the software will depend on the implementation strategy that is undertaken. Due to time constraints, this enhancement has yet to be implemented.

# 5 Significant Challenges

We were assigned the project Connect Four Game and had a month to become familiar with it and test it. The first challenge that the team ran into was the fact that the teammates did not know each other or much about each other’s background. Not possessing that information, the teammates did not necessarily rely on the team for the first few days. It seemed easier to research the topic that stumbled you and then bring everything you know to the team table, rather than just simply ask the teammates for quick assistance. After the first meeting, the team members got a feel for everyone’s skill sets and the rest of the project went smoothly.

The next big challenge that we faced almost right away is the fact that there was no specifications or constraints provided for the project. During all the phases of the project, our team had to make many assumptions on how the game should work and why.

Another difficulty that we ran into was the lack of comments in the code and in some cases not so intuitive names for the variables. This issue made it extremely difficult to perform any white box testing; taking time constraints in the account made white box testing not feasible. For this same reason, bug fixing and enhancing the game took us a lot more time than expected.

# 6 What We Learned

What we learned while working as a team on this project is that with successful collaboration and responsible teammates the project tasks can be completed very quickly and efficiently. Our team found it fairly easy to set goals and achieve them together. At the testing phase, we discovered that in order to perform any white box testing one has to be very familiar with the code or the code has to be very readable with comments or documentation on how the units were supposed to work. During the whole project our team had to make many assumptions as well as consider various trade-offs. For example, is it reasonable to spend 2 business days on familiarizing ourselves with the code enough to perform unit testing with 80 % code coverage? Or would it be more useful to spend these 2 business days performing extensive black box testing? Not having any requirements or constraints for the game, it was important to perform extensive black box testing to make assumptions about the requirements and constraints and then figure out if the game fulfills them.

Our team found that documentation is a very vital part of any project at any stage. We collaborated using email for the most part, so we had a record of every discussion, idea and change that we had.

# 7 What We Would Improve

If our team were to redo this project, we would follow the work-flow below:

1. Employ source-control management system such as GitHub or similar.
2. Employ Continuous Integration methodology to automate testing.
3. Perform unit tests early on.
4. Refactor existing code using design patterns.
5. Perform subsequent tests automatically when there’s a code change.

We realized that the code size was quite large to enable us to tackle all required tests manually during the 7-week run. However, we believed the use of continuous integration would help us achieve a broader test coverage which would undoubtedly result in a much higher performance program.