## Unit Tests

Unit tests are run through the Eclipse IDE.

To run the unit tests, clone the github repository into eclipse, either by manually loading the files or by importing directly from github. From within the project structure pane, select the src/edu/colostate/cs/cs414/p3/testing package, right click, and run all as Junit tests. The results of the tests can then be seen in the right hand pane.

## System Testing

The goal of the system test is to ensure that every action initiated by the user of a client is correctly interpreted from the UI, sent to and processed by the server, and the correct response is sent and handled by the client.

As this is still a relatively early version of the software, the scope of the needed tests are much narrower than it will be in the full system. The system currently has two message/response pairs to test: Registering and logging in.

The system test is most concerned with the intermediate steps in the unit testable modules, such as ensuring that worker threads are able to properly utilize SQL queries, and that the UI can correctly use Client Connection to send messages to the server.

The test is best performed on a single computer to ensure that there are no network errors that may negatively impact the tests. The computer should be running JRE 8.0 or newer, or the equivalent OpenJDK implementation. The database should have a pre-built and populated table of known users to facilitate testing. Once started, if the client or server terminates, the other must be as well and the tests restarted.

### To Perform a System Test

Start the testing database, populated by known users

Start the server, noting the selected port and hostname of the computer, ensuring it is connected to the testing database and that debug messages are on.

In a separate process, start the client, ensuring that it is given the same hostname/port number to ensure connection.

Submit a registration request from the client for a valid new username, while monitoring the server debug console, ensuring that the information received is the same as that submitted, that the response is correctly created, and matches the response received by the client.

Repeat, using an already taken username

Repeat, instead sending a login request for a non-existent user

Repeat, instead trying to login with an incorrect username/password combination

Repeat, instead trying to login with a correct username/password combination.

Close the Client and terminate the server

Terminate the database

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| ID | Name | How To Test | Test Data | Expected Result | Actual Result | Pass/Fail |
| 01 | SQLDriver.testAddUser() | Eclipse | Username  “Test1”  Password  “TestPassword” | True | True | Pass |
| 02 | SQLDriver.deleteUser() | Eclipse | Username  “Test1”  Password  “TestPassword” | True | True | Pass |
| 03 | SQLDriver.testUsernameReturn() | Eclipse | Username  “ctunnell@rams.colostate.edu”  Password  “TestPassword” | “ctunnell@rams.colostate.edu  ” | “ctunnell@rams.colostate.edu” | Pass |
| 04 | SQLDriver.testPasswordReturn() | Eclipse | Username  “ctunnell@rams.colostate.edu”  Password  “TestPassword” | “TestPassword” | “TestPAssword” | Pass |
| 05 | SQLDriver.testValidLogin() | Eclipse | Username  “ctunnell@rams.colostate.edu”  Password  “TestPassword” | True | True | Pass |
| 06 | SQLDriver.testInvalidLogin() | Eclipse | Username  “ctunnell@rams.colostate.edu”  Password  “TestPassword2” | False | False | Pass |
| 07 | UserLogonResponsetest.test() | Eclipse | None | True | True | Pass |
| 08 | UserLogonTest.test() | Eclipse | None | True | True | Pass |
| 09 | UserRegistrationResponseTest.test() | Eclipse | None | True | True | Pass |
| 10 | UserRegistrationTest.test() | Eclipse | None | True | True | Pass |
| 11 | System Test | Manual | Testing Database | True | True | Pass |

RulesEngineTest.Java

-testProcessCaptures(): tests that the engine returns a new Game with an updated Board state where the respective Piece was removed.

- testGameHasEnded\_noKing(): tests that GameResult changes when the king has been captured. GameResult will be BLACK.

-testGameHasEnded\_kingEscaped(): tests that GameResult was correctly set to WHITE when a king reacher a corner cell.

-testGameHasEnded\_Continue(): tests that termination cases were not hit and that the Game is still in progress. GameResult should be CONTINUE.

PieceTest.Java:

-testPieceType(): tests if a rook is correctly evaluated

-testPieceKing(): tests if a king piece is correctly evaluated

-testPieceColor(): tests if a piece is evaluated to its respective color

-testColor(): tests string representation

CellTest.Java:

* testCellType(): test if type gets evaluated correctly
* testCellGetPiece(): test that getter performs correctly
* testCellSetPiec(): makes sure that the piece is updated
* testCellRemovePiece(): tests if the price is removed from the cell
* testCellEquals(): tests if a Cell object equals another Cell object

BoardTest.Java:

-testBoard(): test the cells are correctly initialized

-testCellAccessor(): test teh the Board can index into the array of Cells correctly based off x,y coordinates

GameTest.Java:

-testGameBoard(): tests that Board gets initialized correctly

-testGameResult(): test that GameResult is correctly set

-testGameTurn(): tests that the defender makes first move

-testGameTurnChange(): tests that the attacker will make the next move

-testGameAttacker(): tests that you can access the Player/Account for the game’s attacker

-testGameDefender(): tests that you can access the Player/Account for the game’s defender

AccountTest.Java:

-testAccountUsername(): tests that username is correctly set

-testAccountPassword(): test that password is set and accessed correctly

-testAccountInvites(): test that invites can be added and accessed for an Account

-testAccountPlayers(): test that the Account can access the Player object for games it is playing in

InvitationTest.Java:

-testInviteSender(): tests that sender is correct

-testInviteRecepient(): tests that recipient is correct

-testInviteEquals(): tests that invite objects are evaluated correctly when checking if they are equal

StatsTest.Java:

-testStatsWins(): test that wins is correctly set and accessed

-testStatsLosses(): test that losses is correctly set and accessed

-testStatUpdate(): tests that wins/losses are updated correctly

GameRecordTest.Java:

-testGameRecordAttacker(): tests that record has the correct attacker

-testGameRecordDefender(): tests that record has the correct defender

-testGameRecordWinner(): tests that record has the correct player set as the winner

-testGameRecordStartTime(): tests that record has the correct start time for the game

-testGameRecordEndTime(): tests that record has the correct end time for the game

PlayerTest.Java:

-testPlayerAccount(): tests that a Player account is correct

-testPlayerColor(): tests that the color for a player is correct