Team AMR

Requirements

Requirement 1

User: A user will be able to register an account by entering the command “newUser $username $password”, or login in using a registered username and password by entering the command “login $username $password”.

Priority:1

Testing: We will use “username” and “password” as the variables to pass, when neither of these are registered.

1. After the username and password have been entered using the login command, we expect
   1. “Either username or password are incorrect”
2. After the username and password have been entered using the newUser command, “User Registered” should appear.
3. To verify that the user was registered, we will
   1. Enter the username and password using the register command and expect
      1. “Username already taken”
   2. Enter the username and password using the login command and expect
      1. “User logged in”
   3. Manually check the user JSON file and expect to find
      1. “{’user’:’username’,’password’:’password’}”

Dev:

Flow:

Login:

1. Username exists or not
   1. If the username exist, then
      1. Password is correct
         1. If the password is correct, then the user will be logged in
         2. Else, the error message “Either username or password are incorrect” is returned
   2. Else the error message “Either username or password are incorrect” is returned

Registration:

1. Username exists or not
   1. If the username exists or contains illegal characters, following errors are returned
      1. Username taken
      2. Username contains illegal characters
   2. If the username does not exist and does not contain illegal characters, the information is saved in the user JSON file, and the user is returned the message “User Registered”

Implications:

* The user JSON file contains a JSON Array of user objects.

AMRChessServer

AMRChessProgram

Sends Username and password

Sends result in JSON format

Searches and Saves to

UserJSONFile

Requirement 2

User: Users should be able to move using the command “move $move”, where $move is the legal move expressed in algebraic chess notation.

Priority:2

Testing: We will use various scenarios to test various piece’s moves

1. After entering an illegal move, we will expect

a. “Illegal Move”

2. After entering a legal move, we will expect

a. ”$yourUser ‘s $piece moved to $location”

b. “$yourUser ‘s $piece took $opponentUser ‘s $opponentspiece at $location”

c. “$yourUser castled right/left”

d. “$yourUser ‘s pawn moved to $location and became a Queen”

Dev:

Flow:

1. Evaluate the command.

2. Move is legal for the piece

a. If legal, command correctly identifies the outcome(IE check or a move that takes a piece)

i. If the command correctly identifies the outcome, then the piece is moved, and the new board is sent to the server.

ii. Else “Illegal Move” is returned.

b. Else “Illegal Move” is returned.

Implications:

* There must be some representation of the chess pieces and board using classes.

Sends move to server

ChessServer

AMRChessProgram

Returns updated chess board

Checks legality

Class ChessBoard

Gives piece’s legal moves

Class ChessPiece

Requirement 3

User: Users should be able to monitor their wins/losses using the “stats” command

Priority:3

Testing:

1. After entering the stats command, we expect

a. “Wins:$wins” where $wins is the correct number of wins, and

b. “Losses:$losses” where $losses is the correct number of losses

Dev:

Flow:

1. Request stats

a. Stats exist

i. return stats

b.Stats do not exist

i. create stats and set to zero

ii. return new stats

Implications:

* Stats are stored with

Requests stat

ChessServer

AMRChessProgram

Returns stats