Team 101-1: Reality Bytes
Project Milestone 5

Members:
Clint Eisenzimmer
Will Loughlin
Ian Peterson
Jayden Tang
Ryan Kenfield

Application Name: Wacky Tanks

Feature 1:
Login System

Tests:
The login system has two key mechanics we want to test. Both the logging in process, and the signup system. In testing, we must check core functionality and edge cases of both of these. We start with the signup system.

- We want to ensure we can't take an already used username, so we start our database with the credentials:
  Username: Admin
  Password: password
  We then try to create a new account with the same username and ensure an error comes up.

- We want to ensure that the username meets the minimum of 2 characters and doesn't exceed the maximum of 8 characters. To do this we simply try to create an account with:
  Username: a (1 character)
  And
  Username abcdefghi (9 characters)
  If both come back prompting the user with errors we know everything has worked correctly.

- We want to ensure that account creation doesn't allow for weak passwords, so we require the following measures to make passwords secure:
  Passwords must have a minimum of 1 number, one uppercase character, one lowercase character, at least six characters, and a cap of 20 characters. Additionally, it will have a field requiring the password to be retyped, so that we can ensure the first typing contained no typos and the user can remember the password they typed correctly. This will be tested using various passwords that don't include all the criteria and several tests that do include all of the requirements to ensure errors come back at the right times.

- Final test for the signup is kind of like stress testing, where we consider the fact we don't want duplicates in our database, but we do not check for existing users and create users at the same time, so we try having many users sign up with the same credentials at near the same time to see if any errors occur allowing multiple users to add to the database without having duplicates. The reason we must test this is because we are not certain how all the database systems here work with synchronization, so it is possible we pass the check on one client saying there is no existing user with a username, and we pass another of the same checks before the first client can add the user to the database. If we find this to be a problem we will need to add some kind of lock to the database so only one client who is signing up can be accessing the database at a time, that way checks and additions will be done in a single swoop minimizing the possibility for error.

Next, we need to check that the login system is working properly. The following tests cover what we need:

- We want to check that logins are operating in a basic manner, so using our login from the signup tests we check that we can use Admin-password to log in.

- Assuming this is successful, we want to ensure we aren't getting false positives. To do this, we try the same username with a different password. This should return an error and prompt the user that they didn't enter the correct password.

- In a similar test, we want to be sure that only the correct password works on the associated username, so we make a second login on the database Admin2-password2, and we try using the login Admin-password2, and we should get an error again. (note that the passwords put into the database through the client do not follow the same criteria that a user must follow in the signup stage.)

- Finally, we don't want multiple instances on the same user in the server, so we need to check if the user is already logged in an error comes back to the user. This test can be done by logging in on our Admin account in one tab and attempt to log into the same account in a different tab right after. This should return an error telling the user that their account is in use.

Feature 2:
Damage, Death, and Respawn

Tests:

- We need to check that damage works properly, so we use two test accounts in the server, and ensure that 3 collisions on any point on the tank will always cause death. We can print the health of the user every time they get hit in the nodeJS console to help debug, and this should be easy to debug if any abnormalities are witnessed. Additionally, we must check that the bullet is deleted on the first contact with a tank, and

only counts as one collision on the tank, so damage isn't double-counted in edge cases such as the corner of the tank, or possibly when the tank is rotating.

- Upon the third hit on each tank, we must check that the firing tank's score increases by one, the attacked tank disappears and respawns elsewhere on the map with score reset, and ensure all of the dead tank's bullets are removed.

Feature 3:
Movement

Tests:

- The movement must just ensure that all movement keypresses operate, being w, a, s, d, and space for firing. As long as this works, we know basic movement is operational.

- We must test rotation in a manner as above, and as long as it is generally working, we have no problems with rotation to fix.

- The only edge cases to check here are the actual edges and corners of the map, to make sure interaction and collision with walls doesn't cause any problems with movements. This can be checked by colliding with walls at multiple different angles and rotating into walls and corners.

- Another test to perform is a collision with other tanks and how that affects movement. We want to test many angles of collision and rotation to make sure there is no way for tanks to land inside of each other or clip together in some way. As long as we cannot make that happen we shouldn't face any other issues with movement in regards to players.