Team GulDukat

Due: 10/25/2016

Ethan Wright

Zachary Fenton

Daniel Harding

Rachel Koenig

CP02 Testing Approach

Testing approach for the configuration portion of this iteration was only done on the Configuration Controller class, as most methods of the configuration GUI are private (and should remain so), but also because there was no real need to test the GUI. All input checks happen instantaneously and data is passed to the controller anyways. In general, and in honesty, no specific structured approach was used due to time constraints. Testing at integration did not happen, but key components of the controller were tested. A TDD approach will more than likely be used next time due to its usefulness in eliminating rigidness and fragility of code, which will be helpful for possible code re-usability on the next or third iteration.

Testing approach for the authentication portion of this iteration was pretty simplistic. Since this iteration does not contain a SQL database to store user credentials, we were able to hardcode a set of credentials. If the user logged in as a user, they would be directed towards the Welcome page and if they logged in using the admin credentials they would be sent to the admin panel. The only testing we implemented in the LoginGUITest.cs was to check if the fields were left blank and then printed out an error to the screen accordingly. In order to test you'll have to change the "errorLabel" access to public instead of private. Better test will be developed in the coming iterations.

Testing of the Combat Round Controller was done for Use Case 7. The tests focused on the two main public methods, spell attack and melee attack. These were used when characters attacked

monsters, such as when a user was choosing an action for a player-controlled character. Unit tests were set up for these two methods, where a random input was used in the form of character and monster attributes. These were random because that is how the character and monster constructor initialize them if no values are explicitly given by the user. The result after the attack method should be that the monster's health becomes lower than the health before. The problem with this approach is that we are dealing with random numbers. This means, there is a chance that the attack wouldn't hit, resulting in a test that would fail. Most of the time, the test would pass though.

Unfortunately, Use Case 2 did not get entirely implemented. Although, UC2 is very far along, a few things will be added in at the beginning of the next iteration. The only major aspects lacking as of now are character/monster initiatives and automatic attack decisions made by the system. This was simply due to a shortage of time. A lot of time was put into the rest of UC2. Character and monster stats are correctly calculated, the player can controller characters and deal damage to monsters, monsters die/leave the scene when they run out of health, and the player advances to the next level when all monsters are killed. The whole scenario plays through the correct number of levels as determined by the configuration that happens previously. Since we did not get around to implementing the entire use case, we have not yet made unit tests. However, much time was still spent testing the code. We spent a lot of time trying feed the simulator junk data and cause bugs and crashes. We fixed every bug that we discovered this way. Early in the next iteration, this use case will be finished up and proper unit tests will be implemented.