Nicole Lin, Analeah Real, Jolly Zheng
May 15, 2024
CS 3110 Final Project

Test Plan

As our project heavily relies on user-inputted numbers and letters, we found it difficult to test every single part of the code. However, because we had many mli/ml modules and files that we can test.

One of the biggest parts of our program is the inventory. The inventory stores the weapons, chest items as well as the health bar. This was extensively tested using OUnit testing where we tested based on glass box testing. We looked at every pattern-matching scenario and created a separate test for each. This makes us ensure the output is created in every scenario. After using forms of glass box testing, we also used some black box testing where we used arbitrary values that may or may not be inputted through the program. This is to ensure we have all grounds covered.

However, even with the inventory module, many functions return a unit or a print_endline statement. This made it hard for us to test this because the value changes each time and we could not test these functions using assert statements systematically. Instead, we used manual testing and continuously played our game.

In addition, for the other modules, we followed the same manner. For functions that return a unit (because they are print statements), we tested them manually and for those that returned strings, our constructed types, and boolean expressions we tested them with assert statements.

For our manual testing, it consisted of continuous play through the game until all of our branches were hit. This meant a lot of play testing, but we were doing this a lot while implementing.

Some parts were OUnit tested as well. For example, we made OUnit tests to make sure functions that were supposed to deduct health deducted health (and vice versa for adding health)

For battle manual testing:

- We manually tested every valid input that the player can input with keyboard to access an inventory space (1-5)
    - These items trigger either their own special event, or is a default punch in battle
- We manually tested that after player health reaches 0 or below, the game is over and exits
- We manually tested that after enemy health reaches 0 or below, the particular battle instance is over, and the next randomized event occurs
- We manually tested that weapon damage somewhat increases depending on weapon (but not too much of a difference, so we can make the game more balanced)
- We manually tested that after defeating an enemy, they would drop some kind of loot

- We manually tested text art and prompts were printing correctly, and that each prompt was reached

For chest manual testing:

- We manually tested the cases for
    - Chest with key
    - Chest with no key
    - Ignoring chest with key/no key

  to see if the correct prompt is given for each scenario
- We manually tested that there are different opening prompts (with repeats) for chests
- We manually tested that each kind of loot can be dropped (meat/key/weapon)

For utils manual testing:

- We manually tested for terminal screen flushing, so that it is somewhat a smooth and clean experience for the user/viewer
- We manually tested to see if each test prints the correct prompt from the json files

For scenario testing:

- We manually tested to see if each scenario choice correctly corresponds to the subsequent scenario event

Additional manual tests were conducted, but not explicitly stated since they are very trivial compared to the others to list on here.

The testing approach demonstrates the correctness of the system as parts that can be tested, we use systematic testing through assert statements. However, for parts that we were unable to do, we decided to do a lot of manual testing. A lot of manual testing allows us to be sure that everything works, looks good, and is fun to play under all scenarios.