MS3 Test Plan 5/16/24

Gusladag Technologies (Aakshay Gupta, Skye Slattery, Clarence Dagins)

Since our project extensively relies on GUIs, there was some functionality that we could only realistically test by actually playing the game, which led to some challenges with ensuring test suite extensivity. We wrote an OUnit test suite (found in test/test_camelgo) to test game logic and mechanics, including initial character and obstacle states, movement micro-adjustments, collision detection, and crash handling.

Our test suite verifies proper handling of individual events, such as making sure that the jump function makes the correct velocity and position adjustments to the player. We originally tested collision mechanics manually through playtesting, but we were able to get additional confirmation by having the test suite check that overlapping player and obstacle positions results in a game end. Since we knew whether or not each collision was supposed to occur, we used black-box testing to develop these tests; however, we later used glass-box testing to check specific properties of players and obstacles.

Most of our testing for the obstacles verified the randomized picking of the position and style (i.e., size) of each cactus, as well as confirming that the chances of each option was equally likely. Our get methods were tested more thoroughly, since many parts of our program rely on them. This section of our tests was developed using black-box testing and randomized testing. We later playtested visual representation in our GUI, such as intended movement paths and extra constraints which would make “impossible” cacti spacing situations less likely to occur.

We were able to put a lot of thought into the decoration of the player screen, since our game is displayed using a GUI. Like our testing for the obstacles, we tested decoration styles, velocity, position, and randomization, and we still relied on black-box and randomized testing to do so. We also playtested this functionality to make sure our decoration sprites were loading and moving across the screen as intended.

In addition to confirming start menu functionality, we also had to make sure player deaths were handled appropriately and that the game properly exited. However, because most of the work dealt with visual representation, we omitted many OUnit test cases in favor of manual playtesting. We were able to use black-box testing to confirm appropriate menu initialization and active states, as well as glass-box testing to check menu state transitions, but we weren’t able to write test cases for much more than that. For everything else, we used manual testing to make sure the menu and game were appearing as expected in the GUI window. Our game display compilation unit ran into a similar issue; both issues later hurt our overall Bisect test score.

Regardless, we were still able to verify almost all components of our game, via test cases or otherwise. In addition to achieving over 90% Bisect coverage in all of our other code files, in practice, our game plays just as intended based on our design. For these reasons, we believe that we have appropriately demonstrated correctness of our system.