Testing Plan: MS3

Petros, Grace, Arushi

Below denotes a very general purpose testing plan for the final project of CS 3110. The bulk of our test suite was made using OUnit tests, and we also use QCheck to test a randomized algorithm that we made. No tests are manual tests.

Pin: The Pin Module is a module dedicated to designing a report of feedback to the user/computer whenever they place a guess. It operates mostly as an array in the order from Red (correct spot), White (incorrect spot), and Null (not in answer). We tested as many of the functions exposed in the interface as we could. The tests we ran through OUnit guaranteed that we could convert a guess against an answer into an array of pins. We can then convert that into a string or an integer list. We can extract the multiplicity of all colors and more usefully get the number of reds, whites, and null values in the feedback.
Tests were developed using glass-box testing. These tests demonstrate the correctness of this module because it is meant to represent the feedback pins of the game, which are critical for the codebreaker player to be able to solve the game. Our tests ensure that the operations within the module act as intended, and also that the more easily understood representations that we return are correct. We also ensure that we maintain the order we've claimed the array should be in, where Red appears before White, and White appears before Null.

Game: Game is a module that simulates what the actual board looks like in Mastermind. We use a record type to store information about the guesses, pin responses, number of tries, the answer,

and other useful information. Some entries in the record are mutable, while others are not. In this OUnit testing suite, we ensured that we can instantiate a board, input an answer, update our guesses and its pin responses, ensure mutable fields are appropriately changed, and can clear the board to indicate a next round. Since we were dealing with mutable fields, it was especially important to ensure that we had code that correctly mutated values when necessary, and did not mutate values unexpectedly. This was done through glass box testing, because we needed to ensure that the helper functions that updated certain fields of the record were also done properly. Our tests ensure this module works because we were able to test the fact that each mutable field may not be simultaneously updated. Thus, we were able to find out which fields get updated with certain functions, and which fields don't.

Random Algorithm: The Random Algorithm module is a very general algorithm for a computer to solve Mastermind, which simply guesses random answers. Our implementation uses the linear congruential generator to generate random seeds and then create a list from the seed. We first started with writing some OUnit tests for this module, to test that we implemented the algorithm correctly; given a specific seed, we ran it through the algorithm and checked that our answer matched. This was glass-box testing, as we needed the actual implementation of the algorithm. We also checked that making a random guess returned the correct format. Then, to extend on this, we also used QCheck since the algorithm is non-deterministic. We tested that for 1000 random seeds, our algorithm returns a valid output for each one; outputted guesses needed to be of length 4, contain the values within the correct range, and contain no duplicates. Since we do not depend on the specific implementation, this was black box testing. Because our sample size

is so large, this helps prove that our algorithm code is safe and correct, and the postconditions of our function are satisfied.