CS 3110 Final Project Test Plan
Caml Capital

Our project was somewhat unique in that it was difficult to test using a traditional test case approach. This was because our project involves a game with many variables, and testing each of those variables through test cases would be quite tedious and time-consuming. Therefore, while we still included plenty of test cases, most of our testing was done “manually” by us playing the game.

Testing through user play:

As noted, most of our testing was done through us actually playing the game. This was mostly in the form of what would be considered glass box testing, with us playing the game with a focus on a new feature whenever we added it. One example is the implementation of railroads in our game. Railroads in the Monopoly game are treated differently from the other properties. Their rent depends on the number of them that you own, and they don’t offer the option to buy houses. Therefore when we implemented railroads, we played games with a range of players (2-4) where we would spread the ownership of the railroads differently across players to ensure that the proper rent was being charged when someone landed on them. It was playing trials like these that were our most effective testing method.

To help speed up this sort of testing, we created a sort of sandbox function that lets you move to whatever property you want to. This helped avoid the randomness of the dice rolls and would allow us to quickly test edge cases that were unlikely to occur in the actual game.

Testing through test case:

While most of our testing was done as described above, we also implemented OUnit test cases for some of the base functionality of the game. These test cases focused on fundamental things such as testing empty players and making sure players actually own the properties that they buy. While our testing through user play was our main active testing method as we implemented more features, we relied on OUnit testing to ensure the foundation of our game was solid. This would be considered glass box testing since we were specifically testing the base of the game.

Why our testing method demonstrates correctness:

The main thing that constitutes the correctness of our project was that the game functions the way we expected it to. Manual testing therefore made the most sense for this. By playing the game, especially in our sandbox environment, we were able to test edge cases and see the game from the player's perspective. Since the game worked perfectly when we played it, even in unlikely scenarios, that on its own is enough to demonstrate correctness.

On top of that, our OUnit testing exists to make sure that the background work is happening the way we intended it. Since most of the functions and extra implementations in the game relied on a few fundamental game loop functions (creating players, buying properties, etc.), testing these functions gave us the confidence that the game was operating the way we intended it to. This, combined with our manual testing, ensured that our program is correct.