**Unit Tests**

Feature One:

* Description: The second feature to be united tested is all the boolean methods in the WallDetection class. We decided to test this feature because without wall detection, we would not be able to create a maze map and therefore we need to ensure that the wall detection is working perfectly.
* Test case/class: We created four test methods, one for each direction that Player moves in.

Feature Two:

* Description: The first feature to be unit tested is the detectWin() method. This is an essential feature of the game because this method controls whether Player is still in a game, wins a game, or loses a game based on different conditions.
* Test case/class: To test this feature, a class called TestDetectWin is created with three methods that each contains different test cases. The first method is testStillInGame(), which has two test cases that test two scenarios where Player is still in a game. The second method is testLose(), which has three test cases that test three scenarios where Player loses a game. The third method is tests whether Player wins a game. It only has one test case, which is when Player reaches the exit cell and has collected all the regular rewards.

**Integration Tests**

* Interaction 1 between components, components involved, and test case/class
* Interaction 2 between components, components involved, and test case/class

**Test Quality**

**Findings**

1. The four private int methods for wall detection originally resided in the Panel class and returned an integer variable called “direction” that only the Panel has (Figure #). However, as we created unit tests for the wall detection methods, we discovered that private methods generally are not tested, and if they are so complex that they need testing, the methods should deserve their own class. Therefore, we created a separate class called WallDetection that holds these four methods and changed them to boolean functions (Figure #). By doing this, the code is more reusable because not only the Panel class can use the wall detection methods, other classes can also use them by creating a WallDetection object and calling the methods. This improved the quality of our code and made it easier to test wall detection.

A screenshot of a cell phone

Description automatically generated

Figure #. Old version of wall detection:

private int methods in Panel class.

A screenshot of a social media post

Description automatically generated

Figure #. New version of wall detection:

public boolean methods in

WallDetection class.

1. A bug that we found is that when we were testing the detectWin() method, a test case kept on returning error even though the logic seemed correct and the game was working perfectly fine (Figure #). One condition for Player to be still in game is when Player is on the exit cell but has not collected all the regular rewards. In this case, the string that holds the state of the game is supposed to return “GAME.” Yet, the string returned “LOSE” in the unit test (Figure #). We also made sure that Player and Moving Enemy did not collide but it was still returning “LOSE” instead of “GAME.” As a result, we modified the code for detectWin() by changing the if-else statements. It fixed the issue in that the string is finally returning the correct state for all winning and losing conditions (Figure #).

A screenshot of a cell phone

Description automatically generatedA screenshot of a cell phone

Description automatically generated

Figure #. Test case for testing if Player is still in game.

Figure #. Old version of detectWin(). Returned “LOSE” instead of “GAME”. Winning and losing conditions depend on class’ private variables.

A screenshot of a social media post

Description automatically generated

Figure #. New version of detectWin(). Fixed the testing issue. Winning and losing conditions depend on parameters that are passed to the method.

1. The third finding continues off the second finding. While modifying the if-else statements in detectWin() to carry out successful tests, we also added method parameters to detectWin() to make it less hard-coded, more reusable, and easier to test (Figure #). Now, the losing condition (if score < 0) does not depend the private score variable in the Cell class, but rather an integer variable that is being passed to the method. In addition, the variable that keeps track of how many regular rewards that Player has collected does not depend on the class’ private “collected” variable (Figure #), but a “collected” variable that is also being passed to the method. Lastly, the number of regular rewards that Player has to collect to win the game is not a constant that needs to be changed manually anymore (Figure #), but an integer variable called “rewardNum” (Figure #).