2048 Test Plan

Introduction

This test plan is to ensure the game 2048 meets all required functionality, and tests stability of the program under as many circumstances as possible. The plan is implemented throughout the development of the game, starting with designing for testing through to black-box testing of the final program. The tests will incorporate both code-based tests of function calls using a range of values, and documented procedures for user interface testing and final user acceptance testing.

Build to Test

The implementation of this plan begins at the design stage. This means that when designing the game, good practice such as selecting testable structures and using testable functions should be adhered to. Where possible, using classes, and limiting the number of global variables used within those classes is encouraged. When writing a function, make sure it is using the values from its parameters, and that it is returning a value when complete. This allows testing of the class and its functions as a black-box.

White Box Testing

To test classes and individual functions, a series of tests are programmed into a separate executable. For each testable function, a series values will be fed in, and the results checked against the expected output. The bounds of each component will be tested - for example, a test which creates the board size from zero through to the upper limit.

Test	Description	Pass/Fail?
Create board of different sizes	Attempt to create a board of size 0, 1, 2 and the board size upper limit	
Add numbers to the board	Add numbers to the board. Starting with 0 in each cell separately, then 0 in every cell, then the number upper limit in each cell separately, then in every cell. Test randomly adding numbers to the board.	
Slide board	Set up a known state on the board and attempt to slide in each direction. Check resulting board state against the next known state.	
Slide single line	Set up a known state on a single line and attempt to slide it. Check resulting line state matches the next known state.	
Numbers add up	Check that like values are added together when slid next to each other on a line	
Numbers cascade	Check that all like values are added together when line is slid.	
Game over	Set up a known game over condition and check that game is ended	

Black Box Testing

The user interface testing consists of entering data into the program through the user interface, and observing the output. The output is then checked against expected output. A high level black-box test plan could be as simple as

1 Start a new game
2 View game board
3 Enter a move
4 Check game board matches expected result
5 Enter another move
6 Check game board matches expected result
7 Play until game over.

However, to test the user interface in a more detailed manner, certain scenarios should be presented, where the result of each interaction will be known. The exact arrangement of the board is defined for each scenario, and the moves to be made by the user are predefined. These tests would be of similar nature to the white-box testing, in that known scenarios will be set up, however instead of programmatically validating the functionality, the user validates the functionality through the user interface. This will test the bounds of the user interface and identify any bugs which could occur in taking the users input and translating it into a move, or bugs in displaying the game board to the user. Where possible, the user interface tests should be kept to the minimal set required to test all bounds, as the user testing can be time consuming.

A typical testing scenario may look something like:

Test Scenario 1	Test movement in each direction				
Step	Directions	Expected Ouput			Pass/Fail?
1	Initialise scenario 1	2			
			2		
2	Enter ‘r’			2	
				2	
3	Enter ‘d’				
				4	
4	Enter ‘l’				
		4			
5	Enter ‘u’	4			