4 (a)

We chose to apply a two-prong approach as our testing methodology:

- Functional requirements were automatically tested with JUnit tests. These tests were large, integration-style tests which utilised all the necessary systems from the game, such as managers and components, to allow the creation of and interaction between game entities
- Playtesting was used to verify the functionality of UI elements and for the verification of correct rendering of game textures and animations

This methodology was appropriate due to both the constraints of game development in general and of the LibGDX framework in particular:

- correct rendering is not possible to test programmatically
- UI widgets used a lot of internal LibGDX machinery which would be prohibitively difficult to mock out or initialise correctly to write robust yet isolated tests
- game entities in general require a lot of setup and common state and are hence problematic to test with small, isolated unit tests

For most functional requirement tests, multiple assertions were written to ensure correct behaviours. For example, a test for FR_WASD, “WASD controls should control the ship’s movement”, was broken up into separate tests for each individual direction, each of which had separate assertions for desired behaviours such as “the ship should be located further along the desired direction”, “the ship has positive momentum in the desired direction”, and “the ship should have zero momentum in the orthogonal direction”.

4 (b)

A total of 27 tests were run to ensure functional requirement fulfilment, specifying a total of 60 assertions. All of these tests passed successfully.

The following items were excluded from automated testing for purposes of code coverage:

- com.ducks.screens module, since it contains UI elements which were tested via playtesting
- Debug class, since it is used for development and does not form part of the game
- DesktopLauncher, since it is used exclusively for launching the game and doesn't contain functionality

Automated testing resulted in 68% line coverage of the tested classes, with 79% of methods and 100% of classes being tested. Full detailed coverage report is available for viewing on the website at
https://stanbsky.com/How-Hard-Can-It-Be-Redux/code%20coverage/index.html

A summary of game functionality tested via automated testing is provided in tabulated form on the following page.

In terms of manual playtesting, 35 tests were run to test the UI, 11 for game rendering, and 9 for gameplay. Two tests for rendering failed: the game does not correctly resize, resulting in lack of aspect ratio preservation and issues drawing UI elements when switching between screens after resize. Fixing these issues would necessitate refactoring a lot of legacy code responsible for setting and sizing the viewport and camera. Unfortunately, we did not have sufficient time to resolve this issue prior to the deadline.

Full list of playtests performed on the game can be viewed at:
https://stanbsky.com/How-Hard-Can-It-Be-Redux/PlayTest.html

Summary of functionality under automated testing			
Type of Test	**Test File Name**	**Purpose of Test**	**Pass/Fail**
headless	test_FR_PLAYER_SPAWN	Player Ship spawns on the map	Pass
headless	test_FR_W	Ship moves upwards at the click of the 'W' key	Pass
headless	test_FR_A	Ship moves upwards at the click of the 'A' key	Pass
headless	test_FR_S	Ship moves upwards at the click of the 'S' key	Pass
headless	test_FR_D	Ship moves upwards at the click of the 'D' key	Pass
headless	test_FR_PLAYER_SHOOT	Ship shoots out cannonballs when the mouse is clicked	Pass
headless	test_FR_PLAYER_DAMAGE	Ship takes damage when in contact with an enemy cannonball	Pass
headless	test_FR_PLAYER_DIE	Ship dies when health reaches the bottom	Pass
headless	test_FR_ENTITY_SPAWNS	Powerups spawn on the map	Pass
headless	test_FR_BOSS_SPAWN	Pirate boss spawns and dies when health reaches 0	Pass
headless	test_FR_BOSS_DAMAGE	Pirate boss health goes down when shot at	Pass
headless	test_FR_BOSS_TRIPLE_SHOT	Pirate boss can do triple shots	Pass
headless	test_FR_PIRATE_SPAWN	Pirates spawn and die when shot	Pass
headless	test_FR_PIRATE_MOVE	Pirates move in random ways	Pass
headless	test_FR_PIRATE_COMBAT	Pirates shoot at player when nearby	Pass
headless	test_FR_COLLEGE_SPAWN	Colleges spawn and die when health reaches zero	Pass
headless	test_FR_COLLEGE_COMBAT	Colleges shoot at player when nearby	Pass
headless	test_FR_COLLEGE_DAMAGE	Colleges lose health when shot at	Pass
headless	test_FR_WHIRLPOOL_SPAWNING	Whirlpool spawns then despawns after 30 seconds, reappearing 10 seconds later	Pass
headless	test_FR_WHIRLPOOL_EFFECT	Whirlpool pulls ships in	Pass
headless	test_FR_CHEST	Chests spawns and opens once collided with	Pass
headless	test_FR_BULLET	Bullets spawn once mouse is clicked, slow down after certain time and disappear after a time	Pass
headless	test_FR_DIFFICULTY	Difficulty is set by buttons on main menu	Pass
headless	test_FR_SAVE_LOAD	Tests the ability to save data to a file then load data back into the game	Pass
headless	test_UI_ENDSCREEN	Tests that the end screen can display	Pass
headless	test_UI_SUBTITLE	Tests the ability to display information at the bottom of the screen in game	Pass
headless	test_B2_WORLDCREATOR	Tests that box2D can be initialised	Pass