Module	Engineering1 (Eng1) - COM00019
Assessment Title	Assessment 2, Cohort 2
Team	Quakthulu (Team 16)
Members	Aaron Price, Alec Coates, Charlie Curedale-Rayner, Eleanor Griffin-Smith
Deliverable	Software Testing Report

For testing the code that we inherited from Assessment 1, we are using unit tests for all of the testable methods in the classes that represent individual game objects (boats, obstacles, leaderboard, progress bar, etc.); this provides a good coverage of methods that have clear inputs and outputs and highlights any problems in changes made to those methods without needing a disproportionate amount of development time going into testing rather than the main game. For our unit tests we are using JUnit, because - being the most common Java unit testing framework - it has a wide range of resources and will most likely be well known to anybody working on this code in the future. We are running the JUnit tests using the libGDX headless backend in conjunction with an open source libGDX test-runner class using Mockito, which allows us to test code that relies on libGDX methods even though the actual game isn't running.

We didn't create unit tests for game screens or rendering functions because these have too many variables to practically create unit tests for that would cover every case, as well as that these screens contain libGDX objects that require skins/textures to be loaded to work - which libGDX doesn't support while running with the headless backend. Instead, for these game screens and game rendering, we are using system testing, which is more capable of spotting obvious errors in rendering or general game behaviour. System testing is also used to test the transition between game screens, because this is easy to identify in a user's view.

For the code that we are adding to the project, we are using more unit tests for the game save/load feature, because it has easy configurations to test the output of. We are also going to use integration testing for the game saving and loading to make sure the data is correctly written to and read from the save file in the computer's file system. We are going to use system testing for difficulty selection, because the main cause of concern with this feature is having the right difficulty levels for users, which is hard to test without user feedback. We are also going to use system testing for saving and loading to compliment the automated tests, so that we can make sure the whole system works; from saving data from game objects to the file and then loading the data back into live game objects again, which we can't do with unit/integration tests because of the aforementioned libGDX limitations.

Our team has unit tests, which are run before each build of the game, for 6 game object classes. Below is a list of classes and methods that have unit tests, and an explanation for what each one does.

Boat:

- SteerLeft - Tests that the boat moves left if in the river but does not move left if at or over the left bank.
- SteerRight - Tests that the boat moves right if in the river but does not move right if at or over the right bank.
- MoveForward - Tests the boat moves the correct distance forward.
- IncreaseSpeed - Tests that speed increases if the boat is not tired and is below the speed limit and stays the same otherwise.
- DecreaseSpeed - Tests that speed decreases if the boat is above the minimum speed and stays the same otherwise.
- CheckCollisions - Tests that when a boat collides with obstacles that it takes the correct damage, slows down and the obstacles are removed from the lane.
- ApplyDamage - Tests that the correct damage is applied to the boat.
- CheckIfInLane - Tests that this returns true if the boat is in the lane and false otherwise.
- UpdateFastestTime - Tests that the fastest time is updated correctly, including time penalties.
- IncreaseTiredness - Tests that tiredness is increased if below the maximum and stays the same otherwise.
- DecreaseTiredness - Tests that tiredness is decreased if above the minimum and stays the same otherwise.
- Reset - Tests that all affected values are reset to their defaults.
- ResetFastestLegTime - Tests that the fastest leg time is reset to default.
- GetProgress - Tests the correct progress is returned before and after finishing.

Goose:

- ChangeDirection - Tests that the goose always changes to a valid next direction.
- Move - Tests that the goose only moves if it has room in the lane to move forwards in the direction it is facing.

Lane:

- SpawnObstacle - Tests that obstacles are only spawned if the obstacle limit won't be passed.
- RemoveObstacle - Tests that an obstacle is always removed if it exists in the lane.

Leaderboard:

- UpdateOrder - Tests that the positions on the leaderboard are correctly updated based on leg time.
- GetTimes - Tests that the correct times are returned in the correctly sorted order.
- GetFinalists - Tests that the correct number of boats are returned from the top of the leaderboard in the correct sorted order.
- GetPodium - Tests that 3 boats are returned from the top of the leaderboard in the correct sorted order.

When we ran the unit tests on the code we inherited from assessment 1, 7 of the 27 unit tests failed, as can be seen in the links below.

The test for CheckCollisions fails because there is a return statement inside the for loop that removes hit obstacles, so only one obstacle will ever be removed. It also fails to check the obstacles' width, so are sometimes missed when they should be hit. To fix this problem, we can move the return to after the for loop in a collided boolean if statement and we can add the boat width to the collision check.

The test for Boat.SteerRight fails because it calls getHeight rather than getWidth. It also ignores river banks, so it can leave the river. To fix this problem, we can change getHeight to getWidth and we can add a check to make sure the boat isn't beyond the river bank.

The test for Boat.SteerLeft fails because it ignores river banks, so it can leave the river. To fix this problem, we can add a check to make sure the boat isn't beyond the river bank.

The test for Lane.SpawnObstacle fails because it adds obstacles when the current count is ≤ obstacleLimit, meaning there can be one more than the limit. To fix this problem, we can change the ≤ check to be a < check.

The test for Leaderboard.GetFinalists fails because the method doesn't sort the order of the boats before returning the list of boats. To fix this problem, we can call UpdateOrder at the start of the method.

The test for Leaderboard.GetPodium fails because it calls GetFinalists, which fails as explained above. This is fixed by fixing GetFinalists.

The test for Goose.Move sometimes fails because it changes direction (randomly) after checking that it can move in the original direction and before moving in the new direction using the original check. To fix this problem, we can move the random direction change to be after the goose has tried to move in the direction it checked.

We also have system tests listed below:

- Selecting a boat – Test that you can select every boat and that you are given the same boat that you selected.
- Steering Left – Test that you can steer left while within the confines of the river.
- Steering Right – Test that you can steer right while within the confines of the river.
- Acceleration – Test that you can accelerate while your stamina is below the cut-off point and you are not at top speed.
- Progress Bar – Test that all boats' icons are shown at the correct position along the progress bar.
- Leaderboard – Test that all boats are placed correctly on the leaderboard, with the correct times displayed.
- Leg progression – Test that you can progress to the next leg and that the next leg is the correct leg to be on.
- AI Boats – Test that AI boats are behaving as they should (avoiding obstacles, returning to lane, etc.) and not behaving abnormally (teleporting across the map, infinite acceleration, etc.).

When we ran the system tests on the program we inherited from assessment 1, 2 of the 8 system tests failed, shown in the links below.

The test for Steering Left fails because of the problem outlined in the unit test for Boat.SteerLeft.

The test for Steering Right fails because of the problem outlined in the unit test for Boat.SteerRight.

The tests that we wrote for our expansion of the original code (including unit tests for saving, integration tests for saving and system tests for saving and difficulty) all passed and can be seen in the links below.

Inherited code unit test results:
https://aleccoates.github.io/Dragon-Boat-Z/docs/tests/Test%20Results/00%20-%20Inherited%20Code/Unit%20Test%20Results.html

Inherited code system test results:
https://aleccoates.github.io/Dragon-Boat-Z/docs/tests/Test%20Results/00%20-%20Inherited%20Code/System%20Test%20Results.pdf

Fixed Inherited code unit test results:
https://aleccoates.github.io/Dragon-Boat-Z/docs/tests/Test%20Results/01%20-%20Fixed%20Inherited%20Code/Unit%20Test%20Results.html

Fixed Inherited code system test results:
https://aleccoates.github.io/Dragon-Boat-Z/docs/tests/Test%20Results/01%20-%20Fixed%20Inherited%20Code/System%20Test%20Results.pdf

Expanded code unit test results:
https://aleccoates.github.io/Dragon-Boat-Z/docs/tests/Test%20Results/02%20-%20Expanded%20Code/Unit%20Test%20Results.html

Expanded code integration test results:
https://aleccoates.github.io/Dragon-Boat-Z/docs/tests/Test%20Results/02%20-%20Expanded%20Code/Integration%20Test%20Results.html

Expanded code system test results:
https://aleccoates.github.io/Dragon-Boat-Z/docs/tests/Test%20Results/02%20-%20Expanded%20Code/System%20Test%20Results.pdf

System test videos:
https://www.youtube.com/playlist?list=PLwfw4fzdi5zy4Xx_zd3vPPIyR_jFxVrF4

Traceability matrix:
https://aleccoates.github.io/Dragon-Boat-Z/docs/tests/Test%20Results/02%20-%20Expanded%20Code/Traceability%20Matrix.pdf