Testing

Method Changes

Testing is a crucial part of the implementation phase, not only does it ensure a fully functioning final product, but also an easier collaborative effort, with bugs not being passed around. As a group we aim for 100% test pass rate; bugs may hinder the player experience, which is not what we want. A way to make sure that testing occurs is to use the test driven development process.

In the last assessment, a combination of unit testing (pre-written tests on individual elements of the game which can easily be automated) and system testing (tests carried out on the system as a whole by the development team), were used by our predecessors.

- Unit testing uses reusable, code defined tests which check for correct outputs of functionality. The data produced by this can be found here. The runnable unit tests can be found here.
- System testing involves the game being playtested, checking it against a criteria, only passing if the game follows the behaviour expected. The data produced by this can be found here.

The product was built using these testing methods, so we believed it to be appropriate to continue with these two methods, to maintain consistency.

Tests were originally written using JUnit, so we decided to continue using this as our testing framework. JUnit allows us to rerun tests when new features are added, to ensure that old functionality isn't negatively affected by new functionality unintentionally. It was agreed on that tests should be run before branches were merged with the master branch, to ensure that bugs are not carried through the product.

When we received the project, we found the test suite to be fairly comprehensive. We found the System tests table to be very useful, and therefore updated it as appropriate when new features were added. The unit tests covered a smaller amount of the product, and so required a greater deal of attention.

- As mentioned above, we found the system testing table to be very comprehensive. It covered all features needed according to requirements, and clear and appropriate descriptions for each. For these reasons, we continued to use this document.
- The unit tests were less comprehensive and we found many classes which lacked testing. The tests which did exist didn't run, due to syntax errors and references to architectural elements which did not seem to exist. The tests were also poorly organised, with only one method per test subject, with no clear indication of which specific tests passed or failed.

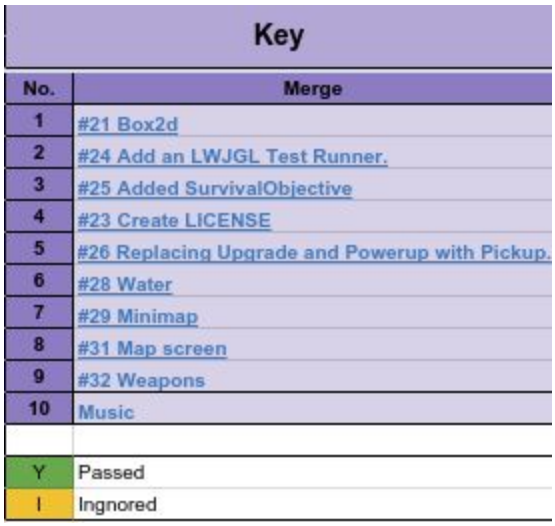
Therefore, although we utilised the existing system testing table with very few changes, a large amount of time was spent writing new unit tests that were based loosely on the previous tests. These tests were created with the architectural changes in mind, and covered a larger amount of the project. Tests are documented in the unit testing table, with a "Pass?" column for the state of the tests after each git merge.

Testing Report

Below is an image of the testing data gathered during the unit tests (the full version can be found here). It can be seen that 11 tests have been ignored, this is for two reasons:

- Some tests are ignored because the class has Round as a dependency. Round is a very heavy class that does a variety of complex tasks and has a large number of dependencies that cannot be easily mocked or separated.
- LibGDX generates an obscure error (java.lang.UnsatisfiedLinkError) when running under the testing conditions. This is probably because Box2D (the physics library) requires a display to be able to work.

It can be seen that, 56% of tests do run, which is a long way from our goal of 100%. That being said, if you consider the amount of code which is testable, you find that 100% of tests do in fact run.



Key	
No.	**Merge**
1	#21 Box2d
2	#24 Add an LWJGL Test Runner.
3	#25 Added SurvivalObjective
4	#23 Create LICENSE
5	#26 Replacing Upgrade and Powerup with Pickup.
6	#28 Water
7	#29 Minimap
8	#31 Map screen
9	#32 Weapons
10	Music
Y	Passed
I	Ingnored

Test Method	Test Condition	Expected Result	Pass?										Notes
			1	2	3	4	5	6	7	8	9	10	
LwjglTestRunnerTest#RunsWithoutExceptions	**LwjglTestRunner runs a test without exceptions.**	**The test passes.**	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	
LwjglTestRunnerTest#CausesAnExceptionWhenExpected	**LwjglTestRunner runs a test with exceptions.**	**The test fails.**	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	
CharacterTest#CanAccessMaxHealth	**A Character was created with a certain max health.**	**Character#getMaximumHealth() is equal to the original maximum health.**	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	
CharacterTest#CanAccessCurrentHealth	**A Character was created with a certain max health.**	**Without updating, Character#getCurrentHealth() is equal to the maximum health.**	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	
CharacterTest#CanBeDamaged	**A Character was damaged by a certain amount.**	**Character#getCurrentHealth() is equal to the maximum *health - damage*.**	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	
CharacterTest#CanBeKilled	**A Character is damaged more than their max health.**	**Character#isDead() is true.**	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	
CharacterTest#IsHealable	**A Character is damaged, and then healed.**	**Character#getCurrentHealth() is equal to the *maximum - damage + heal*.**	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	
CharacterTest#CannotBeHealedPastMax	**A Character is damaged, and healed past max health.**	**Character#getCurrentHealth() is equal to Character#getMaximumHealth().**	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	
ObjectiveTest#HasCorrectObjectiveString	**An Objective is created with a given name.**	**Objective#getObjectiveString() is the same as the name.**	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	
ObjectiveTest#IsUpdated	**An Objective is created.**	**Objective#update() lets the objective's logic update.**	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	
ObjectiveTest#IsOngoingByDefault	**An Objective is created.**	**Objective#getStatus() is ONGOING by default.**	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	
ParticleTest#NotRemovedBeforeDuration	**A Particle is created with a given duration.**	**Particle#isRemoved() is false after an update smaller than its duration.**	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	
ParticleTest#RemovedAfterDuration	**A Particle is created with a given duration.**	**Particle#isRemoved() is true after an update larger than its duration.**	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	
PathfindingAITest#ChasesPlayerHorizontally	**A Player and a Mob with PathfindingAI is created.**	**The mob is horizontally closer to the player.**	I	I	I	I	I	I	I	I	I	I	The tests are ignored because the class has Round as a dependency. Round is a very heavy class that does a variety of complex tasks and has a large number of dependencies that cannot be easily mocked or separated.
PathfindingAITest#ChasesPlayerVertically	**A Player and a Mob with PathfindingAI is created.**	**The mob is vertically closer to the player.**	I	I	I	I	I	I	I	I	I	I	
PathfindingAITest#StopsIfRangeProvided	**A Player and a Mob with PathfindingAI is created with a range.**	**The mob stops within a certain range of the player.**	I	I	I	I	I	I	I	I	I	I	
PathfindingAITest#TakesExpectedRouteAroundObstacle	**A Player and a Mob with PathfindingAI is created in a test level.**	**The mob goes around obstacles and reaches the player.**	I	I	I	I	I	I	I	I	I	I	
ProjectileTest#StartsAtCorrectPosition	**A Projectile is created with a given position.**	**Projectile#getPosition() is the original position.**	I	I	I	I	I	I	I	I	I	I	
PlayerTest#StartsAtCorrectPosition	**A Player is created with a given position.**	**Player#getPosition() is the original position.**	I	I	I	I	I	I	I	I	I	I	LibGDX generates an obscure error (java.lang.UnsatisfiedLinkError) when running under the testing conditions. This is probably because Box2D (the physics library) requires a display to be able to work.
PlayerTest#ScoreStartsAtZero	**A Player is created.**	**Player#getScore() is zero by default.**	I	I	I	I	I	I	I	I	I	I	
PlayerTest#CanAddScore	**A Player is created and some points are added.**	**Player#getScore() is equal to the points added.**	I	I	I	I	I	I	I	I	I	I	
PlayerTest#CanAddPickup	**A Player is created and a Pickup is added to it.**	**The Pickup is accessible from the Player.**	I	I	I	I	I	I	I	I	I	I	
PlayerTest#ScoreMultiplierWorks	**A Player is created, a Score Multiplier and the score is added.**	**Player#getScore() is PLAYER_SCORE_MULTIPLIER x score added.**	I	I	I	I	I	I	I	I	I	I	
PlayerTest#CannotBeDamagedWhenInvulnerable	**A Player is created, Invulnerability is added and is damaged.**	**Player#getHealth() does not change.**	I	I	I	I	I	I	I	I	I	I	