In setting and describing the test strategy we decided to **replace** the testing strategy utilised by the previous team. The previous team had used ***JUnit*** to test classes of their game. This was a useful, more automated testing method however the team had documented that they had experienced problems using *JUnit* which included "integration with LibGDX makes some unit tests difficult to implement for technical reasons" and "Issues arose in unit testing due to the heavy integration with *LibGDX*". These meant that testing may create issues or not be done as fully as it should be, leading to a product we can't say works properly if it hasn't been tested.

We also decided not to continue using *JUnit* as we felt that as a team, we had not used *JUnit* in the last assessment and based on the short-term deadlines, it is better to stick with what we knew works and we know how to do rather than to switch to a new strategy with some established downsides and the potential to be time consuming.

Our team were experienced from agile development from Assessment 2 and it allows more direct tests to be made, whilst ensuring that code is done properly. It also fits in with our Scrum development process as it allows tests to be conducted at each *sprint* (this is explained in detail in the Methods & Plans document).

Our use of Agile Development adopts some basic principles[1]:

- Testing is used to move the project forward: in agile development, testing isn't used towards the end of the project to discover bad software. Our testing method will follow the method that the product is built well from the beginning, using testing to provide feedback on an ongoing basis.
- Testing isn't a phase, it's a whole way of development: testing is a continuous process that is done continually to ensure that the features we planned to implement in each iteration get done. It is easier to fix now rather than to fix later.
- The whole team tests: rather than have one person in the team doing the testing, the whole implementation team does. Everyone on our team working on the implementation will develop and test.
- Feedback is provided regularly: software is tested during and after every implementation of the product and also makes use of automated tests that provide feedback very quickly.
- The code is kept clean with bugs fixed quickly: software with bugs is harder to test and modify and it slows the development process down. The code must therefore be kept clean with bugs fixed quickly and regularly.
- Documentation is kept light: rather than writing documentation which is long, verbose and superfluous we use checklists to suggest tests and keep the testing details light. This saves time and makes testing simpler, meaning it can be done more frequently.
- Coding isn't done until it's tested: when implementations and features are marked as done, it means that they are done and have been tested and necessary fixes made.
- Testing makes the project more efficient: it takes less time to discover and to fix errors when it is checked regularly, than to test the software at large intervals then need to make a lot of corrections as incorrect code has been used in different places.

There are however risks associated with testing:

- When testing small parts of code to ensure they work, the greater game with many of these parts linked and used together may not be given enough testing. Just because some code works, it doesn't mean it will necessarily work in terms of the game as a whole. This can be avoided by ensuring that the game as a whole is tested just as frequently as individual parts.
- Testing too much or allocating an excessive amount of time to testing can result in inadequate time given to actually build the product – if the product isn't completed on time or isn't made to be marketable then it causes serious issues for progress, in terms of the assessments and the swap progress.

Regression and Repeat Testing
Our testing process has to be expanded slightly for this assessment. There are sections of code that have already been tested and proved to work, however we are making large modifications to the code to complete the game which requires to be tested. However we also need to ensure any code that is changed, or used in a different way, is re-tested. All of our classes were tested again to ensure they still work correctly and we have retested **all** of our visual tests.

Testing Types
We have two types of tests - **Class Testing** and **Product Testing**. Class testing is on the code behind the game, usually done by checking or running code, or visually checking the product in a test run. Product testing performs visual tests on the final executable product to ensure the software is sound for the end user.

Class Testing
We performed numerous tests by running the classes and testing a small visual part of the game, the summary results are below. The full testing details and results are available at the website http://www.teampochard.co.uk/game-testing. Where needed, we have repeat tested functions which have been modified since the game was inherited, or used by another function.

Number of Tests	Number Passed	Number Failed		Number not implemented
		Initial	Retest	
10	4	5	1	0

Some of the tests we carried out failed - due to errors in the classes and functions they were running from. Where this was necessary though we took the action that was required, retested the game and the test passed. We did not to product tests until we had completed the class tests - as with errors in the game code it would be pointless to test the final game.

The only test which failed was part of the code which caused errors, we could not remove the erroneous code however, as it was essential for the game but we could handle the error in a **try-catch** block which ensured the game didn't crash with errors.

Product Testing
Product testing consisted of a large number of through, complete tests which tested every aspect of the visual play of the game. The full details of all these tests are available at http://www.teampochard.co.uk/game-testing including evidence screenshots - necessary to validate the tests.

Game Area	Number of tests	Number succeeded	Number failed	Number not implemented
Collision	24	24	0	0
Fighting	40	39	1	0
Movement	67	67	0	0
Non-Functional	3	3	0	0
Resource Continuity	36	36	0	0
Rounds	20	19	1	0

Two tests failed - the first one was the use of Melee to fight which did not work and caused a visual bug in the game. This was fixed and later passed as an additional, repeat test. The other test caused when the running total score did not display correctly, due to an error in adding up the scores. This was also since modified and passed later.

The following diagram cross-references our tests on the final, visual game against our requirements. This shows how the test was designed to check our requirements are correct and checked, and ensure that each requirement is tested in some way.

Testing our Requirements

	G1	G2	G3	G4	G5	G6	G7	G8	G9	G10	G11	G12	G13	I1	I2	I3	I5	I6	C1	C2	C3	C4	C5
SM1-SM16																			X	X	X	X	
SM17-SM20																					X	X	
SM21-SM36																			X	X			X
SM37-SM45																				X	X		
SM45-SM58																			X	X		X	
SM59-SM63																			X	X			X
SM64-SM67																			X	X	X	X	X
SC1-SC24							X																
SF1-SF8								X		X													
SF9								X										X					
SF10				X														X					
SF11-SF12										X								X					
SF13								X															
SF14				X																			
SF15-SF22					X			X															
SF23-SF24					X													X					
SF25-SF26					X																		
SF27-SF29								X										X					
SF30-SF32									X		X												
SF33-SF34									X														
SF35-SF39					X			X				X											
SR1																							
SR2	X	X																	X				
SR3	X																						
SR4															X								
SR5					X																		
SR6-SR7			X				X	X															
SR8					X							X											
SR9-SR11			X		X							X											
SR12-SR19	X		X										X										
SR20-SR22	X					X							X										
SR23-SR24		X										X					X						
SN1			X																				
SN2														X									
SN3																X							
ST1-ST63	X			X	X								X										

Available testing resources
The full document detailing **Class tests** is available at http://www.teampochard.co.uk/ClassTest3.pdf.
The full document detailing **Product tests** is available at http://www.teampochard.co.uk/ProductTest.pdf, with **accompanying screenshot evidence at** http://www.teampochard.co.uk/ProductScreenshots.pdf

References
[1]E. Hendrickson, "Agile Testing: Nine principles and six concrete practices for testing on agile teams", Quality Tree Software, 2008. [Online]. Available: http://testobsessed.com/wp-content/uploads/2011/04/AgileTestingOverview.pdf. [Accessed: 11- Jan- 2016].