In setting and describing the test strategy we decided to **continue** the testing strategy utilised by the previous team. The previous team had used unit testing for the game, and manual tests on the interface to ensure the integrity of the end product. We decided to continue these testing methods as the team have experience and and knowledge of this testing strategy, and it worked for the previous team; in their testing report they described the existing system testing documentation provided as "easy to follow and continue on", and that for the interface tests "it was easy to manually go through the tests and see if these worked". The previous team had also made the decision to re-do all of the tests, to ensure they were carried out correctly, as they found an incorrect test result. Therefore, we decided to do the same.

The previous team's inherited project was done with *JUnit* for testing, however we decided not to reinstate it 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 are experienced with agile development from Assessments 2 and 3 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 Project Review Report).

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. Again, this would be poorly justified - in a four week assessment it would be futile to test the product one week before submission then discover it doesn't work. 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, and a way we have done this is by extensively testing all the visual aspects of the game too.
- 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.
- The short deadline may encourage team members against their better judgement to hurry or skip some of the required testing, perhaps assuming "it's my code, I think this is right" - so it was essential that dedicated time was provided for testing with the ultimatum that development cannot continue until the existing set is fully tested.

Regression and Repeat Testing

Regression testing is the process of testing existing software applications, making sure that a change or addition hasn't broken already existing functionality[2]. It was important that whilst we ensured that certain tests were modified or repeated to ensure correct operation, an important part of regression testing is that it should not take an unfeasibly long amount of time that is wasted rather than doing other tasks[3]. We decided to retest the majority of the tests we inherited - these would be done either when we modified something that may affect that test or where a test refers to the overall playability of the game, when doing visual tests. This meant no time was spent specifically regression testing, making the process efficient.

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	
8	7	1	0	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 thorough, 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
Movement	40	40	0	0
Collision	40	40	0	0
Fighting	36	36	0	0
Rounds	58	58	0	0
Non-Functional	3	2	1	0
Other (menus etc)	11	10	1	0
Cheats	3	3	0	0

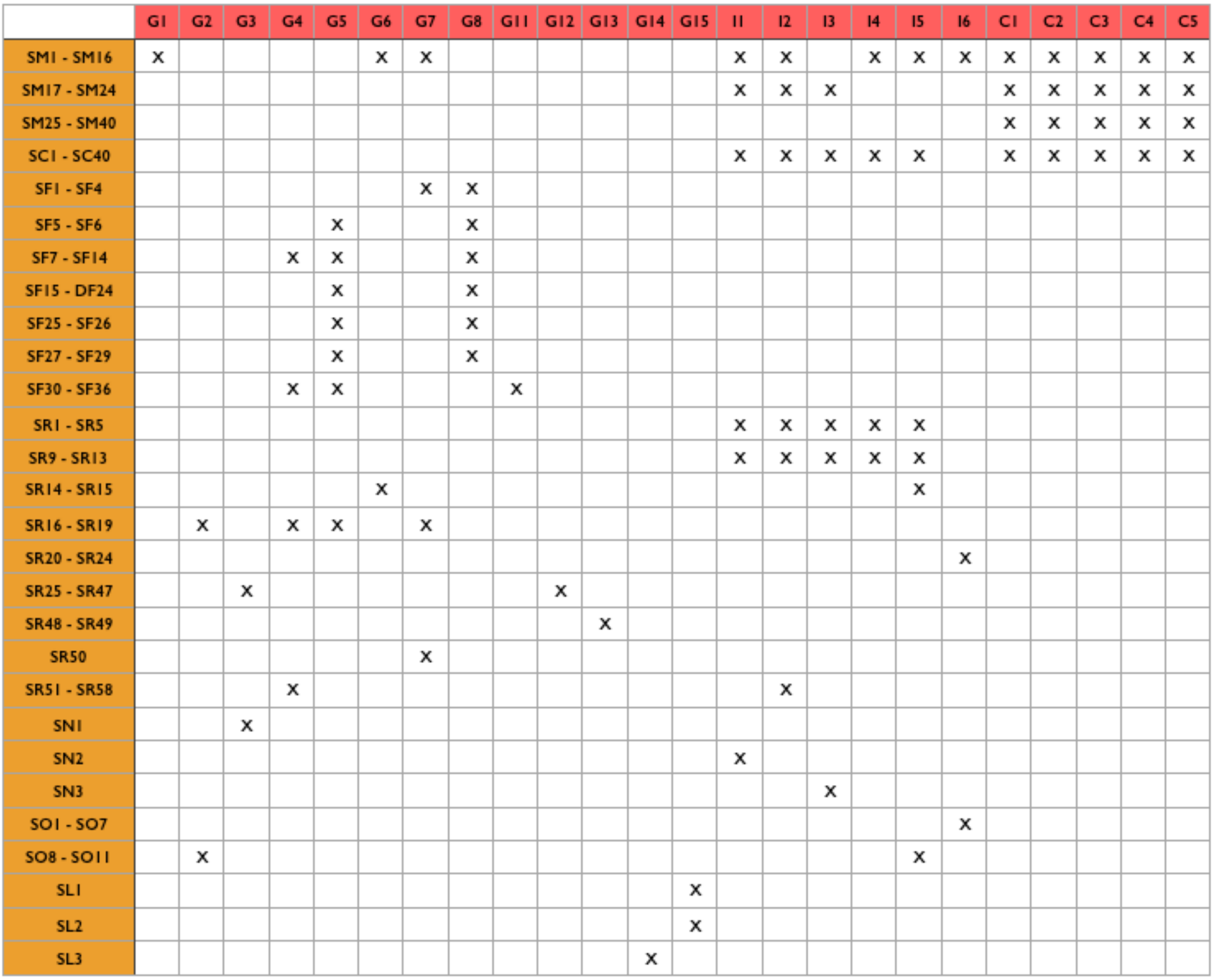
Two tests failed - the first one was a test which checked that the game, for the end user, took a rough time of five minutes to play each level. It is important to keep the game exciting, but not impossible and not so easy it was completed in five minutes. We found levels were too easy so decided to increase enemy characters' health slightly, in order to make the rounds a little bit more difficult. The second test that failed was loading a couple of levels from the select screen, but this was a small error in code which was then fixed and re-tested.

The diagram on the next page 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.

Available testing resources

The full document detailing **Class tests** is available at http://www.teampochard.co.uk/ClassTest4.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

Testing our Requirements



	G1	G2	G3	G4	G5	G6	G7	G8	G11	G12	G13	G14	G15	I1	I2	I3	I4	I5	I6	C1	C2	C3	C4	C5
SM1 - SM16	x					x	x							x	x		x	x	x	x	x	x	x	x
SM17 - SM24														x	x	x				x	x	x	x	x
SM25 - SM40																				x	x	x	x	x
SC1 - SC40														x	x	x	x	x		x	x	x	x	x
SF1 - SF4							x	x																
SF5 - SF6					x			x																
SF7 - SF14				x	x			x																
SF15 - DF24					x			x																
SF25 - SF26					x			x																
SF27 - SF29					x			x																
SF30 - SF36				x	x				x															
SR1 - SR5														x	x	x	x	x						
SR9 - SR13														x	x	x	x	x						
SR14 - SR15						x												x						
SR16 - SR19		x		x	x		x																	
SR20 - SR24																			x					
SR25 - SR47			x							x														
SR48 - SR49											x													
SR50							x																	
SR51 - SR58				x											x									
SN1			x																					
SN2														x										
SN3																x								
SO1 - SO7																			x					
SO8 - SO11		x																x						
SL1													x											
SL2													x											
SL3												x												

Bibliography

[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].
[2]T. Huston, "What Is Regression Testing?", Smartbear.com, 2016. [Online]. Available: https://smartbear.com/learn/automated-testing/what-is-regression-testing/. [Accessed: 19- Apr- 2016].
[3]"Regression Testing", Msdn.microsoft.com, 2003. [Online]. Available: https://msdn.microsoft.com/en-us/library/aa292167(v=vs.71).aspx. [Accessed: 17- Apr- 2016].