Testing

Testing method / approach

We used both unit testing and black box testing to test our game. Unit tests are usually written in code as assertions about what a unit code should do. Black box testing involves using the game and testing it without looking at the internal implementation, essentially play testing. We believe it is important to use more than one testing strategy as every strategy is suitable for different requirements. Unit tests are appropriate for our game as they can be run almost instantly, saving a lot of time during development. They also allow for better refactoring as you can assert that behaviour is the same in all the test cases, making the code more maintainable. Our unit tests will also be useful for anyone who continues to use our project in the future for regression testing. Our unit tests do not just cover 'the happy path' where nothing goes wrong, we also test that exceptions are thrown when they should be. To check test correctness we edited code to get expected failures for tests before replacing with the old code and checking that time the tests passed. Unit testing even with complete coverage of your code does not mean your code is perfect. There still could be cases where a user could provide bad input and cause your program to crash. Additionally, we used black box testing to try and expose any problems our unit testing missed as well as test things unit tests cannot, such as user experience.

Within the team those more familiar with testing used a test driven development[1] approach where they wrote tests before code. In our experience, TDD when performed correctly often produces higher quality and more comprehensive tests. However, for the team members who were new to unit testing the method was not as effective and took too long, so some of the team wrote tests after their code.

We used the Unity Test Tools[2] library for our unit tests. This was appropriate because it is the most used Unity compatible testing tool which means it has a large community for support, other developers are also more likely to be familiar with using it. Unity Test Tools[2] are a version of NUnit[3], one of the most used testing libraries for C#. This means that anyone who has written tests in C# before will likely be familiar with the library.

We are aware of the risks of testing. Incomplete or incorrect tests can give you a false sense of confidence in your software. This is why we chose to use more that one approach to testing. We also tried to reduce the likelihood of incorrect tests by checking each other's tests and checking tests were comprehensive by making this a mandatory part of code review. We made it clear that tests are as important as our code in order to be confident in our product.

Test Report

Examples of unit tests:

Test description	Related Requirements	Result	Notes
TileTest.SuccessfulInstallRoboticon **Given** that the player has a Roboticon to install and is installing it on an owned tile with no Roboticon already installed, the Roboticon **should** be installed onto the specified tile	#4	Pass	
TimeoutTest.InvalidConstructionTest **Given** that a Timeout is constructed with 0 or fewer seconds, the game **should** throw an ArgumentOutOfRangeException		Pass	This test is not required to pass a requirement, but does aid in development
TileTest.SuccessfulRemoveRoboticon **Given** that the player has a Roboticon already installed on the selected tile. **When** the player attempts to remove a Roboticon from the tile, the game **should** remove the Roboticon and place it back into the inventory	#22	Pass	
TimeoutTest.FinishedUnstartedTest **Given** that an unfinished timeout has started. **When** queried if the timeout has finished, the timeout **should** return false	#33 #18	Pass	
MarketTest.CreateNegativeMarket **Given** that a market is created with a negative price, the game **should** throw an ArgumentOutOfRangeException		Pass	This test is not required to pass a requirement, but does aid in development
MarketTest.SuccessfulBuy **Given** that an item is bought with a possitive quantity, and both the player and market have enough inventory to facilitate the buy, the game **should** transfer the item from the market stock to the player's inventory and deduct it's cost.	#8	Pass	
InventoryTest.SuccessfulItemTransaction **Given** that a positive quantity items are to be transferred and the current inventory has the quantity of the selected item, the game **should** transfer the quantity of the selected item from the current inventory to the selected inventory.	#8 #11	Pass	

Examples of black box tests:

Test description	Related Requirements	Result	Notes
Given the game is in phase one, and the player has not yet bought a tile, **When** the player successfully buys the tile, the game **should** not let the player buy another tile.	#20	Pass	The game immediately moves to the next player's turn.
Given that the player has a Roboticon to install and is installing it on an owned tile with no Roboticon already installed, the Roboticon **should** be installed on the tile and a Roboticon sprite added	#4	Pass	
Given the game is running, **when** the map is displayed, the map tiles **should** be of equal size	#3	Pass	
Given a player with some Roboticons installed **when** the player enters phase 4, the game **should** display the production of their colony that turn.	#17	Pass	
Given a player in phase 1, 2 or 3. **When** they reach a set time limit for that phase, the game **should** end their turn.	#21	Pass	The game automatically starts the other player's turn
Given that the game has just started and the menu screen has been completed, the map **should** display that no tile is owned by a player.	#5	Pass	No tile should have a blue or red border
Given that the game is in phase 1. **When** a tile is bought the selected tile **should** have a unique border according to which player bought the tile	#6	Pass	Player 1 should have a red border and player 2 should have a blue border
Given that all the tiles on the map have owners. **When** the turn ends, the game **should** end, producing a winner.	#25	Fail	This test failed as we prioritised other work as this was not one of the requirements the client asked for at this time.

Completeness and correctness of our tests

Our testing approach could be improved by automatic functional tests. These are tests which make assertions about the functionality of the program from the point of the user. Doing so would allow faster development of the GUI as developers could easily see if they make breaking changes. Our unit tests do not cover all edge cases and all possible states but we believe we have conducted enough black box tests and have enough unit tests for a satisfactory level of confidence in our code.

Link to testing material on the website
Contains the executable unit tests, image of GUI test results, and .xml of test results.

[1] Microsoft. “Guidelines for Test-Driven Development“ msdn.microsoft.com. [Online]. Available: https://msdn.microsoft.com/en-us/library/aa730844(v=vs.80).aspx [Accessed: Jan. 23, 2017]
[2] Unity. “Unity Test Tools - Asset Store” assetstore.unity3d.com. [Online]. Available: https://www.assetstore.unity3d.com/en/#!/content/13802 [Accessed: Jan. 23, 2017]
[3] NUnit. “NUnit - Home” nunit.org. [Online]. Available: https://www.nunit.org/ [Accessed: Jan. 23, 2017]