ENG1 Group 2 Marlin Studios

Part 2

Software Testing Report

SKYE FULLER

JONATHAN HAYTER

ALEXANDER MIKHEEV

MARKS POLAKOVS

FREDERICK SAVILL

HAOWEN ZHENG

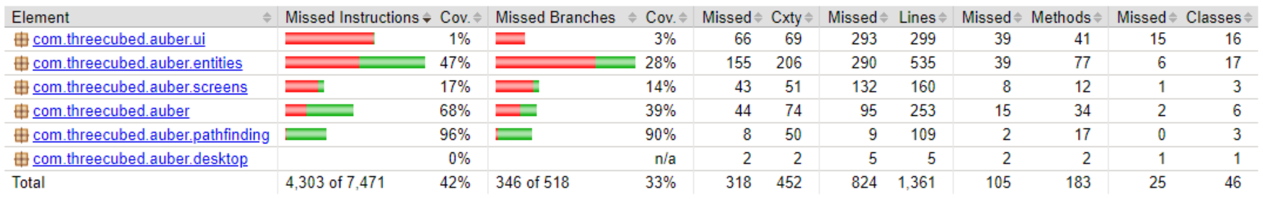
Testing Methods

In order to test the game, we decided to use JUnit tests as upon comparison with other testing methods, it was decided that JUnit tests would be the easiest to implement and write. Initially, we had some issues running these tests on our code due to the fact that both game logic and rendering were intertwined, however, after some research, we found that we could use the headless version of our library (libGDX), along with some changes to the code and adding the GdxTestRunner from the gdx-testing library, to get Junit tests working. We also used the Mockito library in order to mock certain classes necessary for tests to run without error, the JaCoCo library to calculate code coverage, GitHub Actions to automatically run the tests on commit (discussed further in continuous integration report), and the Coveralls service to track coverage between commits.
We felt this was appropriate as we were able to get tests to run automatically through GitHub Actions, and receive feedback from Coveralls, including pass % and line coverage. We used Coveralls as it was free for public GitHub repositories and did everything we needed.

When writing our tests, we designed them to cover as much of the game as possible. This included core functionalities, such as NPC movement, and also requirements. The table below shows the test name, what requirement it is related to, a description of what was being tested, and its result. Where possible, requirements have at least one test.

Tests Run

Our tests ran with a Pass % of 100%, and a line coverage of 42%. Full Testing report and Traceability Matrix are available on our website. (https://2020-eng1-team2.github.io/)



Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
com.threecubed.auber.ui		1%		3%	66	69	293	299	39	41	15	16
com.threecubed.auber.entities		47%		28%	155	206	290	535	39	77	6	17
com.threecubed.auber.screens		17%		14%	43	51	132	160	8	12	1	3
com.threecubed.auber		68%		39%	44	74	95	253	15	34	2	6
com.threecubed.auber.pathfinding		96%		90%	8	50	9	109	2	17	0	3
com.threecubed.auber.desktop		0%		n/a	2	2	5	5	2	2	1	1
Total	4,303 of 7,471	42%	346 of 518	33%	318	452	824	1,361	105	183	25	46

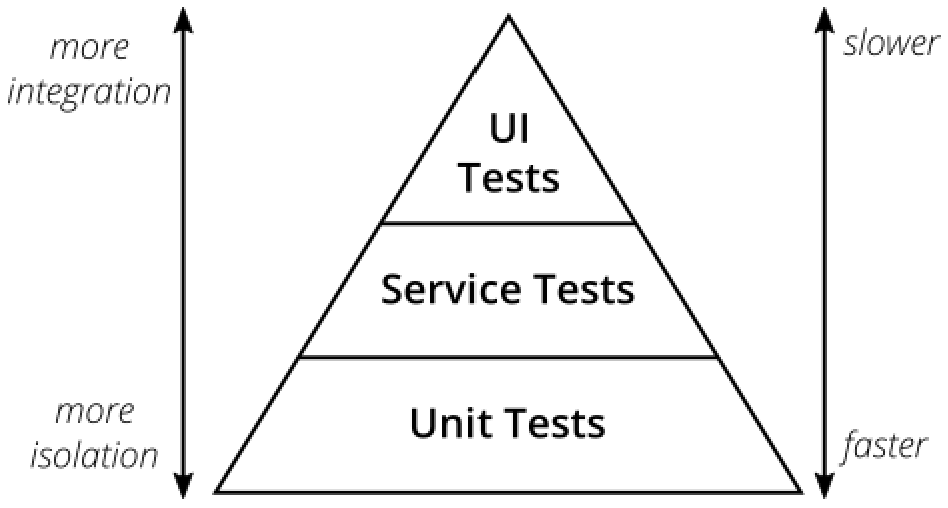
Jacoco Report

Test	Requirement Tested	Author	Description of test	Result
assetTest()	-	Marks	Tests if sprite atlas exists	Pass
easyNextDifficultyTest()	FR_DIFFICULY, FR_MENU	JJ	Tests if the next difficulty after Easy is Hard	Pass
hardNextDifficultyTest()	FR_DIFFICULY, FR_MENU	JJ	Tests if the next difficulty after Hard is Easy	Pass
winConditionTest()	FR_WIN_CONDITION	JJ	Tests if game ends in a win when infiltrator count <= 0	Pass
loseConditionTest()	FR_LOSE_CONDITION	JJ	Tests if game ends in a loss when all systems are destroyed	Pass
testEntityOnScreen()	-	Marks	Tests GameEntity.entityOnScreen() function	Pass
infiltratorsSpawnedTest()	FR_HOSTILES	JJ	Tests if all Infiltrators are spawned in on game start	Pass
playerInfiltratorSpeedTest()	FR_PLAYER_SPEED	JJ	Tests if player movement speed is the same as infiltrators	Pass
attackDamageTest()	FR_HOSTILES_ATTACK	JJ	Tests if Infiltrators attacks do 10% of the players max health	Pass
infiltratorSpawnTest()	FR_RANDOM_SPAWN	JJ	Tests if Infiltrators are all spawned in different positions	Pass
civilianSpawnTest()	FR_RANDOM_SPAWN	JJ	Tests if Civilians are all spawned in different positions	Pass
healTest()	FR_HEAL	JJ	Tests if the player gains health while in the Infirmary	Pass
updateMovedTest()	CON_REAL_TIME	Marks	Tests if an NPC moves to a different position without being instructed to	Pass
destinationReachedTest()	-	Marks	Tests if NPC status changes to IDLE once they have reached their destination	Pass
nonHostileTest()	FR_ALIENS_COUNT	JJ	Tests if 24 non hostile aliens are spawned in when a game starts	Pass
pauseTest()	UR_PAUSE	JJ	Tests if NPCs move when the game is paused	Pass

playerSpawnTest()	FR_PLAYER_SPAWN	JJ	Creates 2 games and ensure the spawn position of Auber is the same in both	Pass
invisibilityTest()	FR_POWER_UPS, FR_POWER_UPS_ABILITIES	JJ	Tests if invisible is set to true for player when power upis activated	Pass
invincibilityTest()	FR_POWER_UPS, FR_POWER_UPS_ABILITIES	JJ	Tests if player’s health gets set to max when power up is active	Pass
superSpeedTest()	FR_POWER_UPS, FR_POWER_UPS_ABILITIES	JJ	Tests if player speed is increased when power up is activated	Pass
visionTest()	FR_POWER_UPS, FR_POWER_UPS_ABILITIES	JJ	Tests if all Infiltrators are set to exposed when the power up is active	Pass
buffsOnAttackEasyTest()	FR_POWER_UPS	JJ	Tests if 5 power ups are spawned when a system is attacked (Easy mode)	Pass
buffsOnAttackHardTest()	FR_POWER_UPS	JJ	Tests if 5 power ups are spawned when a system is attacked (Hard mode)	Pass
saveLoadTest()	UR_SAVE	Marks	Tests if the World.read() function works correctly	Pass
randomFloatTest()	-	JJ	Tests if the Utils.randomFloatInRange() function works as intended	Pass
randomIntegerTest()	-	JJ	Tests if the Utils.randomIntegerInRange() function works as intended	Pass
keySystemsTest()	FR_KEY_SYSTEMS	JJ	Tests if there are at least 15 key systems in the world	Pass
spriteTest()	FR_ALIENS	JJ	Tests if two sprites exist for aliens	Pass
abilityTest()	FR_SPECIAL_ABILITIES	JJ	Tests if at least 3 special abilities exist for Infiltrators	Pass
prisonTest()	FR_PRISON, FR_ARREST	JJ	Tests if Infiltrators can move & are in the brig when arrested (hit by beam twice)	Pass
respawnTest()	FR_RESPAWN	JJ	Tests if Auber gets teleported to the Infirmary when they run out of health	Pass
telepadTest()	FR_TELEPADS	JJ	Check to see if there are 3 functional telepads on the map	Pass

Completeness & Correctness of our Tests

In the Jacoco Report (above), it can be seen that some aspects of our codebase were tested more than others. For example, only 1% of the UI classes were subject to tests, whilst 96% of pathfinding was tested. This is due to the nature of which some of these classes behave. UIs primarily draw to the screen, and don’t handle any game logic, this makes them difficult to run Junit Tests on. Perhaps a different approach such as System Testing would have allowed us to test these in a more realistic manner, however this would have required more integration, and would take longer to write and run; see image below[1]:



This meant that we were unable to test certain requirements, such as; **UR_UX**, **FR_MENU**, **FR_DEMO**, **FR_TUTORIAL**, and **FR_ATTACK_NOTIF**, as they relied heavily on UI aspects of the game. Furthermore, System tests would have allowed us to write tests more easily for requirements such as **FR_SABOTAGE**, which rely on real-world time.
The Traceability Matrix which can be found on our website under “Testing” shows the tests and what requirement they test in a graphical manner. It can be seen that for most requirements, we wrote just one test, this is solely due to the fact that the project we took over did not have any tests written, and time constraints meant that we had to focus our efforts on other parts of the project.

Increasing our line coverage or writing multiples of the same test with different inputs would have certainly increased the correctness of our tests, however it is difficult to measure correctness of tests. Methods exist to prove correctness of a unit test, however, these are very time consuming, and it is unrealistic for a small team on a limited timescale to use these methods, such as Formal verification[2].

Testing material can be found at the following:

- Full Jacoco Report can be found here: https://2020-eng1-team2.github.io/jacocoFullReport/html/index.html
- Traceability Matrix can be found here: https://2020-eng1-team2.github.io/pdfs/rtm.pdf
- Junit Test files can be found in desktop/test in the source code

[1] https://martinfowler.com/articles/practical-test-pyramid.html
[2] https://en.wikipedia.org/wiki/Formal_verification