Testing

Cohort 2 - Group 7 - ‘Escape from Uni’

Members: Louis Burdon, Varun Nayak, Alex Nevard, Sam Russell, Gaoman Zhu, Vivaan Kampani, Teva Geffen, George Overton

The testing process was continuously iterated throughout the development timeline so as to account for any significant changes made to existing functions, and to easily isolate code that introduced issues. Our project employs a multi-faceted testing strategy for the project, combining our automated testing implemented through our Continuous Integration pipeline and the custom manual testing schema we created, so as to ensure that our project delivers all of the desired functionality.

For the automated tests, our project leverages unit testing using JUnit, Integration testing and the Mockito framework. We collectively came up with this decision for the following reasons:

1) The isolation of each of the classes - the isolation of the classes meant that we were able to individually test the functionality of each of the methods associated with the classes we deemed appropriate for automated testing. It meant that we were able to highlight key bugs and errors we faced with each individual method before the entire game was built in the next phase of our CI pipeline.

2) Headless execution - The project has a key graphics component so we needed a way to test out interactions. Mockito enabled us to spin up mock objects and simulate interactions through our automated tests. It enabled us to write far more test cases than we would have been able to if we were to use JUnit alone, spanning more of our project. However, for classes like AudioManager and EventGreggs, where we knew there was a significant audio/visual aspect as well as user interaction for the requirements like UR_Audio, UR_University and UR_Usability, we collectively agreed that a solely manual testing schema would be more appropriate.

3) Seamless integration of new code with our existing files - Automated testing proved to be more effective for testing the core logic of our Shall and Should requirements as it would also save us a lot of time. It meant that when it came to adding new code to our repository, we were quickly able to identify any issues either with the implementation of the test itself or the implementation of the method in the class, owing to our schema. The logs produced from calling assertions in our tests proved to be extremely useful as well because it pointed out to us the points of failure extremely quickly when we had to debug our code. For instance, following the implementation of our Achievements section functionality, significant changes were made to our code so the testing schema was no longer appropriate for the state of our code. Through studying the logs, we were able to identify the issue that was causing our tests to fail and it was related to the implementation of the test itself. Recognising this then meant we were able to refine some of the tests and we managed to achieve a 100% success rate with our tests.

Following the completion of our automated testing schema, the pipeline would then build the game after completing a few other jobs. Once built, this is where our thorough manual testing schema would come in. We created a table covering all of the manual tests, which has been attached further down the document. It had been made much easier for us to design this schema because of the due diligence we carried out to study the code prior to implementing anything in the planning phase and thoroughly setting out our user requirements.

JUnit-Test-Report:https://vrun1506.github.io/team10project-continuance/Website/docs/reports/tests/index.html
JaCoCo-Test-Coverage-Report:https://vrun1506.github.io/team10project-continuance/Website/docs/reports/jacoco/index.html

Test statistics for the entire testing process

core > io.github.team10.escapefromuni

io.github.team10.escapefromuni

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
SettingsPage		0%		0%	26	26	113	113	16	16	1	1
MainMenu		0%		0%	27	27	111	111	18	18	1	1
LeaderboardPage		0%		0%	20	20	112	112	14	14	1	1
PauseMenu		0%		0%	22	22	103	103	16	16	1	1
NegativeEvent		43%		4%	17	26	54	106	5	14	0	1
NPCEvent		45%		5%	15	24	50	102	5	14	0	1
GameScreen		0%		0%	15	15	61	61	12	12	1	1
RoomManager		78%		31%	30	43	55	222	8	16	0	1
AchievementScreen		0%		0%	14	14	52	52	11	11	1	1
Player		29%		0%	27	31	48	70	9	13	0	1
GameOverScreen		51%		55%	9	21	38	90	3	11	0	1
AudioManager		41%		14%	31	39	39	72	10	14	0	1
UsernameInputScreen		0%		n/a	8	8	38	38	8	8	1	1
TutorialPage		0%		0%	12	12	27	27	10	10	1	1
HiddenEvent		66%		50%	11	20	20	59	3	9	0	1
EscapeGame		21%		n/a	4	6	22	27	4	6	0	1
UsernameInputScreen.new InputAdapter() {...}		0%		0%	5	5	16	16	2	2	1	1
AchievementManager		87%		86%	6	25	12	63	3	14	0	1
PositiveEvent		89%		64%	7	18	4	36	3	11	0	1
NegativeEventType		85%		n/a	4	7	4	17	4	7	0	1
Door		80%		0%	2	6	3	17	1	5	0	1
HiddenEventType.new HiddenEventType() {...}		47%		n/a	1	2	3	4	1	2	0	1
PositiveEventType.new PositiveEventType() {...}		41%		n/a	1	2	3	4	1	2	0	1
PositiveEventType.new PositiveEventType() {...}		41%		n/a	1	2	4	5	1	2	0	1
Room		92%		85%	3	21	3	30	1	11	0	1
NPCEventType		80%		n/a	3	6	3	13	3	6	0	1
PositiveEventType.new PositiveEventType() {...}		50%		n/a	1	2	3	4	1	2	0	1
PositiveEventType.new PositiveEventType() {...}		50%		n/a	1	2	3	4	1	2	0	1
Timer		95%		100%	1	11	1	17	1	8	0	1
HiddenEventType		94%		n/a	1	5	1	12	1	5	0	1
HiddenEventType.new HiddenEventType() {...}		90%		n/a	1	2	1	2	1	2	0	1
Event		100%		80%	2	10	0	28	0	2	0	1
AchievementType		100%		n/a	0	4	0	15	0	4	0	1
PositiveEventType		100%		n/a	0	3	0	10	0	3	0	1
Achievement		100%		n/a	0	7	0	10	0	7	0	1
EventType		100%		n/a	0	1	0	6	0	1	0	1
ScoreManager		100%		n/a	0	5	0	10	0	5	0	1
DoorDirection		100%		n/a	0	1	0	5	0	1	0	1
PositiveEventType.new PositiveEventType() {...}		100%		n/a	0	2	0	4	0	2	0	1
HiddenEventType.new HiddenEventType() {...}		100%		n/a	0	2	0	2	0	2	0	1
Total	5,087 of 8,522	40%	274 of 381	28%	328	505	1,006	1,690	177	310	9	40

Package io.github.team10.escapefromuni

all > io.github.team10.escapefromuni

246	0	0	1.670s	100%
tests	failures	ignored	duration	successful

Classes

Class	Tests	Failures	Ignored	Duration	Success rate
AchievementManagerTest	24	0	0	0.045s	100%
AchievementTest	14	0	0	0.009s	100%
AchievementTypeTest	10	0	0	0.002s	100%
DoorDirectionTest	4	0	0	0.001s	100%
DoorTest	14	0	0	0.044s	100%
EscapeGameTest	6	0	0	0.249s	100%
EventTypeTest	14	0	0	0.003s	100%
GameOverScreenTest	14	0	0	0.646s	100%
HiddenEventTest	16	0	0	0.026s	100%
HiddenEventTypeTest	12	0	0	0s	100%
NPCEventTest	8	0	0	0.015s	100%
NegativeEventTest	8	0	0	0.016s	100%
NegativeEventTypeTest	14	0	0	0.001s	100%
PositiveEventTest	16	0	0	0.052s	100%
PositiveEventTypeTest	16	0	0	0s	100%
RoomManagerTest	14	0	0	0.318s	100%
RoomTest	14	0	0	0.239s	100%
ScoreManagerTest	12	0	0	0.001s	100%
TimerTest	16	0	0	0.003s	100%

As part of the automated testing process, we executed 123 tests twice for sanity check to see whether the test really failed or whether it was an anomalous result (double execution was implemented through the pipeline).

All of the automated tests passed because they purely tested the logic of all of our code. The automated tests checked the initialisation of all of the class objects and their variables, the logic of the methods, the output of the methods through assertion logs, whether files exist or not, and interaction with the other objects. As a result, none of the automated tests really failed because they were used to cover basic interaction and logic. Where possible, we did execute an automated test as it would really make the development process far more efficient by enabling us to diagnose logic problems far faster than we would have been able to manually. Had we been manually looking through our code and testing each individual functionality independently, especially considering the size and complexity of the project, it would have taken us far longer to debug the issues associated with our code and would have meant a longer development timeline for our project.

As you can see from the JaCoCo report, the test coverage of the 123 tests that we have written only really tests 40.4% of the code that we have written and so we felt it was necessary to design a manual testing schema to cover the aspects of our code where there was a greater visual and audio aspect (playing the background music as an example) or a

higher level of interaction. This is because, as mentioned previously, the automated testing schema will predominantly test the logic of the game. Therefore, to ensure integrity of our code and the game produced, it was pivotal to create this manual schema to cover the missing areas.

The failed test for the functional requirement chasing dean can't be passed. We chose not to implement a physical dean as part of our game, however the dean will catch the user by the end of the time limit (as demonstrated through the message after the game is over). This meant that the requirement, which requires a dean to chase the user, can't be passed. For this test to pass we could implement a physical dean into our game to chase the player character around the map , as another way the user could lose.

Automated testing schema

Requirement	Test(s) related to the requirement	Test results
FR_Map_Limits	DoorInit(), setActive(), getActive(), DoorDirections() RoomManagerRef(), DoorPos(), DoorSize(), RoomManagerInit(), doorsInitialised(), doorDirections(), initialRoomSetup(), NoCollisionRoom(), exitRoom(), AddAdjacent(), allDirections(), getAdjacent()	Pass
FR_Score_Counter	InitialScore(), increaseScore(), multipleIncreases(), Reset(), CalculateFinalScore(), calculateFinalScoreNoTime()	Pass
FR_Pause	Pause(), Resume()	Pass
FR_Timer	checkTimerDone(), allEventsExist(), TimerInit(), TimerUpdate(), TimerMultipleUpdates(), TimeTicking(), TimeDone(), NegTime(), Reset(), TimerPrecision()	Pass
FR_Losing_Time	allEventsExist(), LoseScreenInit(), checkTimerDone()	Pass
FR_University	EventTypeLongboi(), EventTypeBob(), EventTypeTHE3(), EventTypeSYS2(), EventTypeENG1(), EventTypeGreggs(), EventTypeMonster(), EventTypeCupNoodles(), EventTypeNetworking(), EventTypePizza()	Pass
FR_Achievements	initialiseAchievements(), getAchievements(), saveAchievements(), loadAchievements(), loadAchievementsNoFile(), eventsPartial(), eventsFull(), checkTimerDone(), MultipleAchievementsCompleted(), PrintAchievements(), MultiAchievementSave(), achievementInit(), achievementInitComplete(), getDetails(), setCompleteToTrue(), toStringCheck(), allAchievementTypes(), toggleCompletion(), EnumValues(), allEventsExist(), getTypeNames(), getTypeDescriptions()	Pass
FR_Negative	allEventsExist(), EventTypes(), EventTypeNeg(), EventTypeNone(), InvalidEvent(), startEvent(), initialise(), initialiseUI(), endEvent(),	Pass

	storeReferences(), NegativeEventTypes(), EventTypeTHE3(), EventTypeSYS2(), EventTypeENG1(), EventTypeJob(), enumVals(), InvalidNegativeEvent(), initialise(), startEvent(), updateWhenPlayerFar(), UpdateWhenPlayerClose(), PickupOnlyOnce(), EndEventNotUsed(), EndEventUsed(), storeReferences(), setAndGetEvent(), NoEvent()	
FR_Positive	allEventsExist(), EventTypes(), EventTypePos(), EventTypeNone(), InvalidEvent(), PositiveEventTypes(), EventTypeGreggs(), EventTypeMonster(), EventTypeCupNoodles(), EventTypeNetworking(), EventTypePizza(), enumVals(), InvalidPositiveEvent(), setAndGetEvent(), NoEvent()	Pass
FR_Hidden	allEventsExist(), EventTypes(), EventTypeHidden(), EventTypeNone(), InvalidEvent(), updateWhenPlayerFar(), UpdateWithPlayerClose(), RevealOnlyOnce(), EndEventWhenRevealed(), EndEventWhenNotRevealed(), storeReferences(), HiddenEventTypes(), EventTypeLongboi(), EventTypeBob(), EventTypeInverseControls(), enumVals(), InvalidHiddenEvent(), setAndGetEvent(), NoEvent()	Pass

NFR_Usability	gameInit()	Pass
NFR_sys_reqs	viewportDims()	Pass
NFR_accessibility	viewportDims(), Resize()	Pass

As you can see from both this table and the provided reports the automated testing was fully passed as well as further manual testing which yielded one failed test.

Manual Testing Schema

Requirement	Test	Result
FR_Player _Displayed	Is the player displayed on the screen	Pass
FR_Player _Movement	Is the player able to be controlled by keys on the keyboard	Pass
FR_Settings	Does the settings page open and function	Pass
FR_Chasing_ Dean	Is there a dean that chases the character on screen	Fail
FR_ Leaderboard	Does the leaderboard display the top 5 players score	Pass
FR_Audio	Manual	Pass

Bibliography

https://docs.gradle.org/current/userguide/java_testing.html#test_reporting

https://github.com/mockito/mockito/wiki

https://javadoc.io/doc/org.mockito/mockito-core/latest/org.mockito/module-summary.html

https://www.freecodecamp.org/news/java-unit-testing/

https://www.vogella.com/tutorials/Mockito/article.html