Software Testing Report

Team 4 - Undercooked

Fin Cochrane
Sehran Ahmed
Sam Davis
Hamza Salman
Owen Thomas
Zhenyi Xu

a) Briefly summarise your testing method(s) and approach(es), explaining why these are appropriate for the project. (5 marks, $\leq$ 1 *page*)

b) Give a brief report on the actual tests, including statistics of what tests were run and what results were achieved, with a clear statement of any tests that are failed by the current implementation. If some tests failed, explain why these do not or cannot be passed and comment on what is needed to enable all tests to be passed. If no tests failed, comment on the completeness and correctness of your tests instead (12 marks, $\leq$ 3 *pages*).

c) Provide the precise URLs for the testing material on the website: this material should comprise the testing results and coverage report generated by your automated testing tooling, and descriptions of manual test-cases that you designed to test the parts of the code that could not be covered by your automated tests (5 marks).

Testing Methods

Our testing included both dynamic and static methods to verify the overall functionality of our code. We used unit tests in IntelliJ using JUnit for the dynamic tests, and used these to check the actual code was functioning as intended, which is a necessary part of any project. To test whether the game would “feel” functional, we used static tests that involved an expected outcome while playing the game and an actual outcome, along with any tests that weren’t able to be implemented using unit tests.

Test Report

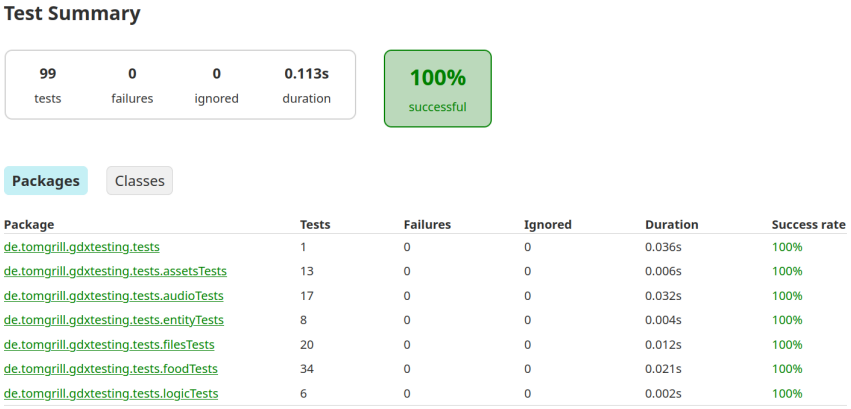
Important Dynamic Tests

Test ID	Description	Expected Result	Actual Result	Conclusion/ Actions Needed
DTEST_AUDIO_MANAGER_INIT	Initialise the AudioManager without crashing, with the correct default sounds.	No crashes, and default assets are loaded.	Same as expected results. Though the default sounds should not be any sound used in normal gameplay.	Replaced the default sound with a unique silent sound, so the user is not disrupted during regular gameplay when default sound is used.
DTEST_AUDIO_MANAGER_UPDATE_VOLUME	Change the volume of sound and music loaded into the AudioManager.	The volume values for all sound and music assets should change to the specified values.	They did not change.	Had to redesign the test as it was not representative of runtime. The changed test correctly reflected updates to volume in

				sound and music.
DTEST_TEXTURE_MANAGER_INIT	Initialise the texture manager without crashing, and load the default texture.	No crashes, and default assets are loaded.	Same as expected.	N/A
DTEST_AUDIO_SETTINGS_LISTENERS	Set the music volume, game volume settings and save them using listeners.	The listeners should set their repsective values for the Audio Settings and then the values should be saved to a settings.json file.	Same as expected.	N/A
DTEST_AUDIO_SLIDERS_SLIDER	Ensure that the slider can be correctly moved within bounds.	The slider should be able to update its properties to reflect the slide of its slider, but not allow sliding beyond its minimum maximum.	Same as expected.	N/A
DTEST_ENTITY_INIT	Load the entity texture and sprite, as well as default values for its properties like position.	All properties should have default values, except the texture and sprite, which should be updated with the correct texture.	Same as expected.	N/A
DTEST_ENTITY_COLLIDING	Detect if 2 entities collide properly.	Should not allow collision with entities of no width/ height. But should otherwise collide if the entities' hitboxes' overlap.	Did not detect collision when it should have.	A simple typo, where updating the width via the setter updated the height instead of width, preventing width being set to a value above 0, was fixed.
DTEST_MOVEABLEENTITY_MOVE	Can moveable entities move correctly?	Using the move function correctly moves the entities the correct distance.	Same as expected.	N/A
DTEST_FILECONTROL_SAVE_LOAD_DATA	Use the FileControl class to save and load data easily	Should save text and json data appropriately without crashing, and load the data back too.	Same as expected.	N/A
DTEST_INSTRUCTION	Can instructions load sprites, be serialised and deserialised.	Instructions should initiialise without crashing, and save and be loaded from a json file.	Same as expected.	N/A
DTEST_ITEMSTACK	Item stack should have all functionalities the stack memory	Should work like a regular stack memory structure, but with Items.	Same as expected.	N/A

	structure does, with the Item class.			

Here's a screenshot of the overview stats for all the unit testing:



Test Summary

99 tests	0 failures	0 ignored	0.113s duration	100% successful

Packages | Classes

Package	Tests	Failures	Ignored	Duration	Success rate
de.tomgrill.gdxtesting.tests	1	0	0	0.036s	100%
de.tomgrill.gdxtesting.tests.assetsTests	13	0	0	0.006s	100%
de.tomgrill.gdxtesting.tests.audioTests	17	0	0	0.032s	100%
de.tomgrill.gdxtesting.tests.entityTests	8	0	0	0.004s	100%
de.tomgrill.gdxtesting.tests.filesTests	20	0	0	0.012s	100%
de.tomgrill.gdxtesting.tests.foodTests	34	0	0	0.021s	100%
de.tomgrill.gdxtesting.tests.logicTests	6	0	0	0.002s	100%

Note: https://github.com/TomGrill/gdx-testing was used to run our unit tests.
For the full statistics and information on our testing, you can use the following link: https://htmlpreview.github.io/?https://raw.githubusercontent.com/undercooked-team/Piazza-Panic-UnderCooked/main/tests/build/reports/tests/test/index.html

No tests failed, and all implemented tests succeeded within reasonable time. However, the full game has not been fully covered by unit tests, with only 15 out of 108 classes being tested by unit tests. However, many of the other classes had functionalities similar to the classes we did test. For example almost all the entity classes loaded textures in the same way as Item.java, which was tested.
Also, many of the tests required user input to be fully tested, meaning static testing was used to cover functionality of the classes that weren't unit tested. For example the user interacting with the map is better tested statically, as there many potential runtime errors, which unit testing would not be very effective at picking up, eg. if the screen freezes, and the delta time supplied to the game is larger than usual, due to lag, then it was possible for characters/moveableEntities to clip out of the bounds of the map, by them moving more than usual in a single frame.
So even though the test coverage via unit tests is low, the fairly similar methods across classes and thorough static tests (using different people for testing, including members outside our team, to play the game) more than make up for this unit test coverage deficit.

Static Tests

Test ID	Description	Desired Result	Actual Result	Conclusion/ Actions Needed
STEST_ MENU	Navigation across the main menu screens is clear and easy	All buttons are clear as to what they do, every screen can be navigated back to again	Sliders in audio settings are unclear. Otherwise result as expected	Functional. Audio sliders need labelling
STEST_ DIFFICULTY	Scenario and endless difficulties	Different difficulties are noticeably more	Higher difficulties give less time from	Fully functional

	are noticeably different	challenging	customers, increasing the challenge	
STEST_ MODES	Different scenario and endless modes noticeably change gameplay	Modes introduce noticeable changes in gameplay	Modes change what recipes must be made, along with more challenging modes	Fully functional
STEST_ CONTROL	Controls work, make sense, feel good to use and can be learnt quickly	Controls work and are intuitive and can be learnt easily	Controls are slightly complicated but make sense	Functional. Controls could be simplified but not necessary
STEST_OOB	Player should not be able to get out of bounds	Player should not be able to get out of bounds	Player cannot get out of bounds due to collision	Fully functional
STEST_ CHEF	Currently selected chef is clearly displayed	Current chef is clearly identifiable	Camera focuses on current chef	Functional. Could be more clear
STEST_ INTERACT	All player interactions work as intended	Player should be able to interact with all necessary stations, and pick up and put down items	Player can interact with all necessary stations, and pick up and put down items	Fully functional
STEST_ CUSTOMER	Customer recipes and time left are clearly displayed	Customer recipes and time left are clearly displayed	Customer time is displayed with bar, recipe is displayed in bottom left, overall time is displayed at top	Fully functional
STEST_ POWERUP	Powerup effects are conveyed clearly	Powerup effects are easily identified	Powerup sprite says what it does on it in text, different colours for different powerups	Functional. Could be improved to be more visually appealing
STEST_COLLECT_POWERUP	Powerups can be collected	Powerups disappear and have an effect on the gameplay	Powerups disappear and have an effect on the gameplay	Fully functional
STEST_ STATION	The purpose of stations is conveyed clearly	Stations can be identified easily	Stations are visually distinct	Fully functional
STEST_ TUTORIAL	Tutorial is effective at conveying necessary information to the player	Tutorial tells player what they need to do	Tutorial mode effectively teaches the player how to play the game. Additional help is given for recipes during actual gameplay	Fully functional