

Software Testing Report

Group 5 :

Callum Watson

Jack Guan

Craig Slomski

Yaseen Khan

Chase Mo

Kamrul Islam

Testing method and approaches:

Automatic and manual testing is conducted on the project, with manual testing including playthroughs of the game and extensively testing each feature. Manual testing is particularly necessary due to the nature of the project being an interactive game; it can further uncover hidden visual flaws which are not detectable when using automatic tests. Certain tests are created for the pre-existing game, however they are mainly targeted towards the new features due to the current games architecture being heavily modular.

Automatic testing

For automatic testing unit testing will be conducted on each project class's methods where possible to singularly modulate the classes for maintainability. For example, the machine class is instantiated with arguments for each interactable workstation, therefore its return results are static meaning that automatic testing can be implemented to check the outputs of each method within the class; therefore only requiring an instance of the class and executing one of the classes methods. Whereas testing power ups featured in the game requires instantiating the entire gameMaster, featuring testing collision recognition between the chef and the item and the resultant code executed as a result.

The tests will be conducted using junit testing with test classes each containing related test methods.

The entire game or just a particular class method may be called or instantiated to conduct the test depending on the resources required to execute it, therefore we can test for modularity or entire connected systems within the project. The class GdxTestRunner sourced from the vle testing tutorial is further used with the tests, so that individual classes can be instantiated without running the gradle. These tests can further quickly test any breaks in code from new merges or additions to the codebase, or check their compatibility.

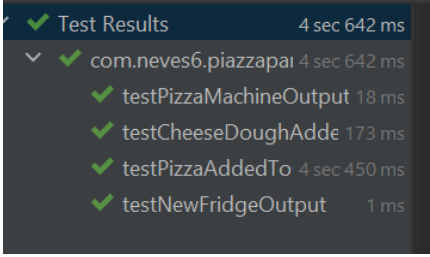

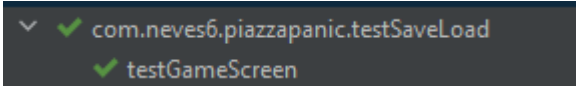
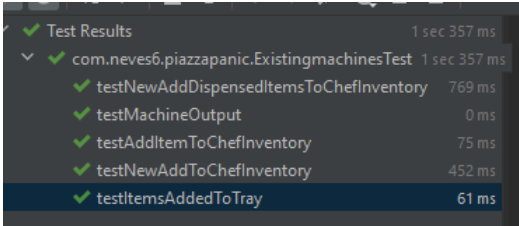
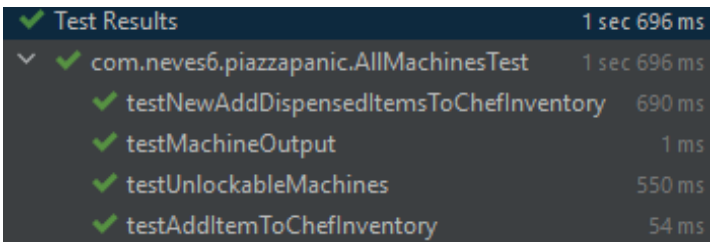

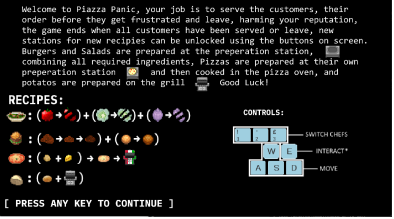
Furthermore we have incorporated testing on Github so that with each push the series of automatic tests are triggered to run on the latest version of the project. Highlighting any issues that may have been caused by the current changes.

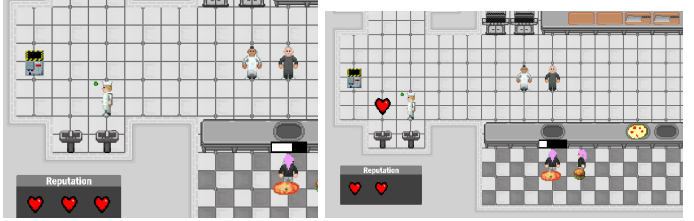
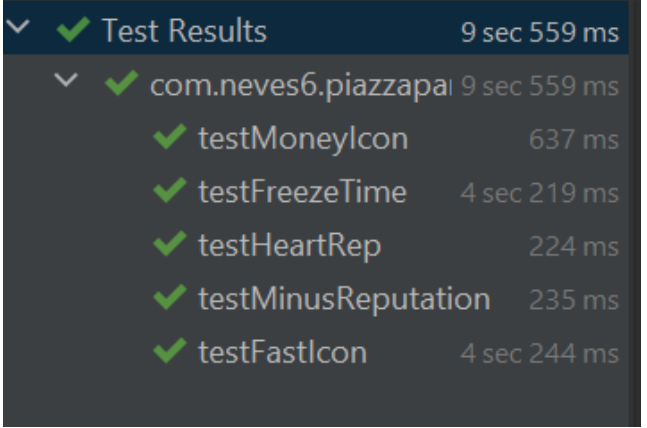
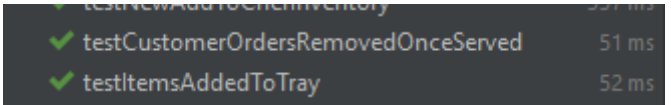
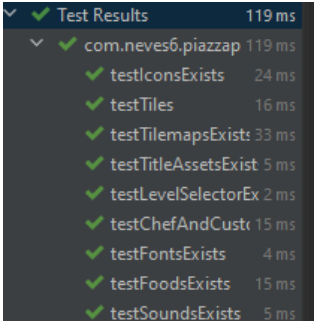
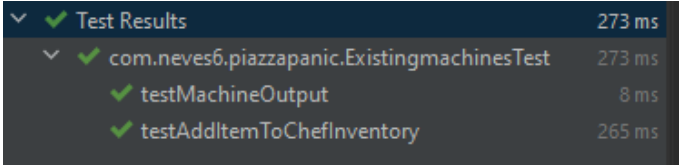
Manual user testing

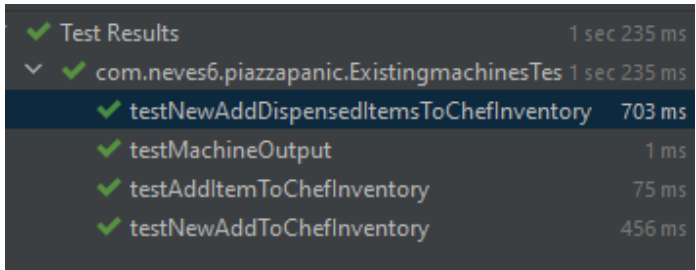
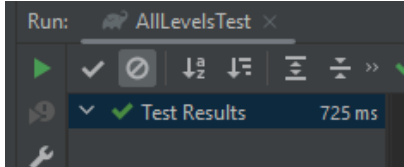
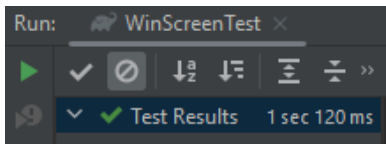
Manual testing is further used to gauge the user experience during gameplay as well as testing a combination of all the features into one scenario to check whether they cooperate smoothly in sync. For manual testing like automatic there is a set expected outcome, a method walkthrough (steps to follow) to test the feature at hand and the expected versus actual result. As manual testing is a more visual approach, screenshots of the gameplay will be documented with notes and explanations backing the results of each test where necessary.

For example checking for the correct screen changes when appropriate (level selector screen changing to the game scenario screen), or saving and loading the user's gameplay being able to visually see the positions of the sprites and making sure they are visually matching the correct positions set. As well as having updating sprites on screen change to the correct images not just numerically in the code but also visually at hand.

Report on actual tests

	Tests	Method	Result
1	<p>New pizza ingredient's fridge outputs are added to the chef stack.</p> <p>Pizza machine output added to chef stack</p> <p>Pizza creation machines output and dispenser output</p>	Automatic (pizzamachineTest.java)	 <pre> Test Results 4 sec 642 ms └─ com.neves6.piazzapari 4 sec 642 ms └─ testPizzaMachineOutput 18 ms └─ testCheeseDoughAdder 173 ms └─ testPizzaAddedTo 4 sec 450 ms └─ testNewFridgeOutput 1 ms </pre>
2	<p>Save and load game state</p> <p>(gameplay was saved before the exiting of the game, and when the load game is selected it returns to the last save file)</p>	Manual / Automatic	 <p>The game state loaded fully from the last save.</p>  <pre> com.neves6.piazzapanic.testSaveLoad testGameScreen </pre> <p>Unit test results</p>
3	<p>Resultant tray contents (foods) created once necessary ingredients have been deposited in the tray.</p> <p>(salad, burger etc are created as a result once all their necessary ingredients have been added)</p>	Automatic	 <pre> Test Results 1 sec 357 ms └─ com.neves6.piazzapanic.ExistingmachinesTest 1 sec 357 ms └─ testNewAddDispensedItemsToChefInventory 769 ms └─ testMachineOutput 0 ms └─ testAddItemToChefInventory 75 ms └─ testNewAddToChefInventory 452 ms └─ testItemsAddedToTray 61 ms </pre>
4	<p>Machines unlock after set requirements are met.</p> <p>(tests machines before they are unlocked seeking an error thrown, then after they are unlocked which returns a correct string.)</p>	Automatic	 <pre> Test Results 1 sec 696 ms └─ com.neves6.piazzapanic.AllMachinesTest 1 sec 696 ms └─ testNewAddDispensedItemsToChefInventory 690 ms └─ testMachineOutput 1 ms └─ testUnlockableMachines 550 ms └─ testAddItemToChefInventory 54 ms </pre> <p>Machines were properly unlocked and locked under the intended conditions</p>
5	<p>Passing screen states, changing the current game screen</p>	manual	  <p>In the example above after selecting the tutorial it moves onto the tutorial screen. After testing all of the options for the entire game the all of the screens passed the test, all being able to move onto another screen once the right button or condition was met.</p>




6	<p>Powerups correctly applied to user's when interacted with</p> <p>Tests the visual implementation of Power ups applied once collided with checking that the correct icon appears and is applied once the chef walks over its coordinates (collision)</p>	manual	 <p>As shown in the example, once a reputation power up is collected the reputation increases.</p>
7	<p>Power ups applied once collided with:</p> <p>Reputation point increase when heart powerup collected</p> <p>Time frozen when freeze time power up collected</p> <p>Machine processing speed increase when icon powerup collected</p> <p>Customer money increased when icon powerup collected</p> <p>Reputation decreased when minus rep power up collected</p>	Automatic (collected by chef) (powerupt est.java)	 <p>Furthermore tests the generating of icons onto the set screen coordinates and that they are interactable with the chef.</p>
8	<p>Customers are removed from the queue once their order has been served</p>	Automatic	 <p>The chef inventory is checked and emptied once the serving station is interacted with. If false nothing happens if it's the right order the customer and inventory item is removed.</p>
9	<p>Assets testing</p> <p>Testing that all game assets are present and accounted for. Therefore recognised by the system to exist.</p>	Automatic	
10	<p>Existing machine testing (tests machines before the code refactoring)</p>	Automatic	 <p>Machines were accessible in the machine list.</p>

11	Existing machines testing (Tests that the machines give the correct output and that when the correct item is in the chef inventory and an interaction is made the correct output is pushed into the chef stack) (tests machines after the code refactoring) (also tests the new pizza machine addition after code refactoring)	Automatic	 <p>Each machine can be fetched from their layers position correctly and interacted with to fetch the correct outputs for the indented inputs. Furthermore any incorrect inputs are not considered therefore it is robust.</p>
12	Testing that different levels present different game conditions	Automatic	
13	Testing GameWinScreen functionality	Automatic	 <p>Once the game win conditions were met the game ended and switched screens to the game end screen.</p>

Comments and Evaluation:

Out of 32 post code refactored automatic tests all 32 passed. These tests tested the functionality of the game features not just focusing on testing individual class methods, but interlinked game actions which included a variety of functions in order to operate. However, all test files were run and their results accounted for in order to determine the total number of tests passed.

Furthermore all tests tested functions as the game was running, therefore receiving real time results.

Build and Run Tests summary				
	Tests	Passed 	Skipped 	Failed 
JUnit Test Report	32 run	32 passed	0 skipped	0 failed

The github automatic testing which ran on each git push also passed the tests meaning that the new code integration did not break any functionality.

Failed Tests and Changes:

The tests which were made for the machines before code refactoring failed upon the refactoring of the code (10 testAddItemToChefInventory, 1 testPizzaAddedToChefInventory) as intended. The new machine tests for after code refactoring however passed.

Furthermore the pizza machine test had multiple iterations as it had to adapt to the changing additions to the game. Firstly it was a simple machine class instance which was inside a list containing all machines, which then after code refactoring. It required its tiled layer to be able to be accessed and (x,y) position found, and lastly it required checking the right game conditions for it to be unlocked and accessible.

Website test content URL : [Testing | Runtime Terrors @ UoY \(runtime-terrors-team-5.github.io\)](https://runtime-terrors-team-5.github.io)