PROJECT PHASE 3: TESTING REPORT

Unit Tests

Features that need to be tested

First, we built unit tests for our game that covered the core functionalities. We went through each of our classes and methods and unit tested those that represented single features of the game and that should be tested in isolation. Below is a list of the features that we unit tested:

A. Construction of the map with different images displayed in each cell
B. Player collision with regular rewards, bonus rewards, obstacles, punishment cells, and the end goal cell
C. Enemy collision with regular rewards, bonus rewards, obstacles, and punishment cells
D. Win/lose conditions following the above B and C features

How we tested them

Test class	Features covered	How
CellMapTest	A	• Created an instance of CellMap and tested if each cell is set to the correct "type." • This will better confirm that the correct image is displayed on screen for each cell.
detectCollisionTest	B, D	• Simulated character collision by spawning the character one cell away from the specified cell, then simulating a keypress that moves the character on top of the specified cell. • Check if the collision outcome is correct (score or screen is changed correctly). • Applied boundary test concepts such as setting variables to on, off, in, and out-points to test edge cases in which the player loses with regard to their score.
EnemyTest	C, D	• Simulated enemy collision by spawning enemies on top of obstacles and checking if a collision is detected. • Checking if bonus rewards/regular rewards are registered as non-collision cells since enemies can move through these rewards.

		• The above was achieved by testing if the collision value is true or false, where false represents no collision. • Tested the lose scenario where the player collides with an enemy and the appropriate lose screen is displayed.

Integration Tests

Interactions that need to be tested

After the unit tests, we conducted integration tests. These covered the instances where our game's logic was integrated with external libraries. These libraries aided in the construction of the game's GUI. Below is a list of the functionalities that we integration tested:

A. Keyboard input corresponding to the correct character movement on-screen
B. On-screen timer works as intended and bonus rewards are populated when necessary
C. Title, rules, and game windows being displayed when intended

How we tested them

Test class	Interactions covered	How
CharacterTest	A	• Instantiated the character and using the KeyEvent library, simulated keyboard movements such as up, down, left, and right. • Checked if the character's position was correctly transformed.
timerTest	B	• Ensured time is valid (0 <= seconds < 60). • Checked if bonuses are displayed in the correct positions with respect to time.
UITest	C	• Instantiated the UI and simulated button clicks to navigate through the various screens and check if the correct one was displayed.

Test Quality and Coverage

Measures taken to ensure the quality of tests

We first designed our tests by referring to the initial use cases we drafted. Following these allowed us to cover each of the core functionalities of our game at a high level. Then, we looked

for instances where we could incorporate boundary tests to test our edge cases. These were utilized in our tests that covered the win and lose conditions for the game. Finally, we measured our line and branch coverage to determine where additional testing was required. Initially, our coverage percentages were much lower than those displayed below; however, we pinpointed where the issues lay and further tested in those areas to bring the line and branch coverage up for our game.

Line and branch coverage

The below image shows our overall line and branch coverage:

Archetype - Phase-2

Archetype - Phase-2

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
com.group21.app.Screen		86%		52%	27	69	27	335	9	46	0	12
com.group21.app.Entity		91%		77%	26	94	18	218	2	22	1	7
com.group21.app		0%		n/a	4	4	7	7	4	4	2	2
com.group21.app.Cell		100%		100%	0	6	0	93	0	4	0	2
Total	358 of 5,478	93%	52 of 183	71%	57	173	52	653	15	76	3	23

We had a 93% line coverage and a 71% branch coverage. Further, the below images show the line and branch coverage breakdowns for each of our packages' (Screen, Entity, and Cell):

com.group21.app.Screen

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
MenuPanel.new ActionListener() {...}		29%		30%	10	12	17	30	0	2	0	1
ScreenPanel		99%		61%	8	18	2	83	1	9	0	1
RulePanel		95%		n/a	4	6	4	21	4	6	0	1
WinPanel		99%		n/a	1	3	1	30	1	3	0	1
MenuPanel		99%		n/a	1	3	1	29	1	3	0	1
LosePanel		99%		83%	2	6	1	31	1	3	0	1
TitlePanel		98%		n/a	1	3	1	20	1	3	0	1
UI		100%		100%	0	10	0	70	0	9	0	1
LosePanel.new ActionListener() {...}		100%		n/a	0	2	0	7	0	2	0	1
WinPanel.new ActionListener() {...}		100%		n/a	0	2	0	7	0	2	0	1
RulePanel.new ActionListener() {...}		100%		n/a	0	2	0	6	0	2	0	1
TitlePanel.new ActionListener() {...}		100%		n/a	0	2	0	6	0	2	0	1
Total	224 of 1,692	86%	22 of 46	52%	27	69	27	335	9	46	0	12

com.group21.app.Entity

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
Enemy		83%		61%	22	45	7	76	0	7	0	1
Music		0%		n/a	2	2	10	10	2	2	1	1
Character		98%		90%	2	19	1	47	0	7	0	1
detectCollision		100%		100%	0	24	0	71	0	2	0	1
Reward		100%		n/a	0	2	0	9	0	2	0	1
Bonus		100%		n/a	0	1	0	3	0	1	0	1
Entity		100%		n/a	0	1	0	2	0	1	0	1
Total	111 of 1,244	91%	30 of 133	77%	26	94	18	218	2	22	1	7

com.group21.app.Cell

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
CellMap		100%		100%	0	5	0	91	0	3	0	1
Cell		100%		n/a	0	1	0	2	0	1	0	1
Total	0 of 2,519	100%	0 of 4	100%	0	6	0	93	0	4	0	2

Methods/functionalities that were not covered

This section outlines the areas of our game that were not covered and why we felt it was acceptable to exclude them. Excluding these areas of code resulted in small impacts to our line and branch coverages shown above.

TitlePanel, RulePanel, MenuPanel, WinPanel, LosePanel classes

Each of these classes needed to be implemented as KeyListeners or ActionListeners. This meant they needed to include some stubs for some or all of actionPerformed(), keyPressed(), keyTyped(), and keyReleased(). However, these methods were not implemented and never used, and therefore were not covered in our tests.

ScreenPanel class

Likewise with the above, this class needed to be a KeyListener, however keyTyped() was not used in the game's logic and was not tested. keyPressed() and keyReleased() were used, but they were overridden by the Character class, so they were also not tested directly through ScreenPanel.

Music class

We did not test the Music class because it would require verifying that audio is heard, which is beyond the scope of the tests we were to create.

Findings

What we learned from writing and running tests

From designing and building our tests, we better understand the value of creating automated tests. Locating bugs in our code was far more efficient when testing in this manner, as opposed to manual testing, which we were using while building the game during initial construction. The line and branch coverage report was a very helpful tool because it allowed us to see which lines were not being tested and how we should alter our tests accordingly. We can definitely see how these automated tests would be beneficial in large-scale projects in a business. After new functionality is added to a program, the tests can be easily and quickly run again to see if any features were impacted by the new addition.

Changes made to the production code

One of the main things we added to our production code during the testing phase was a static variable in the UI class called gameState. During testing, we realized that it was very hard to test the user interface because the code did not have eyes and could not visualize if the correct screen was displayed. Hence, we added the gameState variable that updates dynamically depending on what screen the character is on. This made it possible to test whether the player was seeing the correct screen. For example, if the player's score drops below zero, then the player sees the lose screen, and the gameState changes to "losePanel".

We also refactored some minor features in the UI class to simplify the testing. For example, before testing, the replay button in the losePanel was instantiated in its constructor, making it inaccessible from outside the class. This was a problem for testing as it was not possible to simulate the button click in test functions. Therefore, we refactored the code by making the replay button public and instantiating it outside the constructor so the test functions could access it.