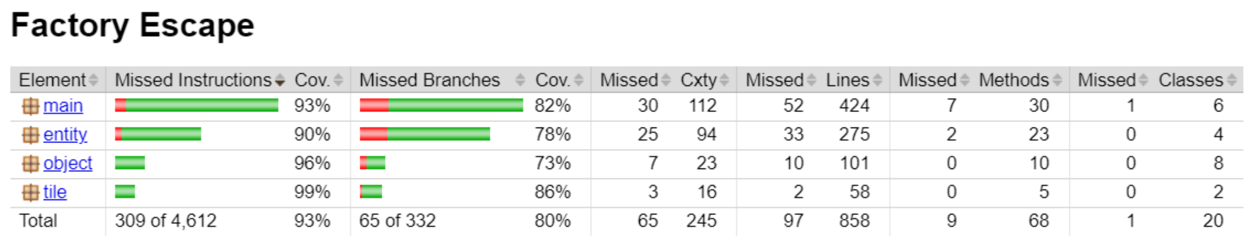
Ensuring Test Quality

To ensure adequate test-case quality we considered as many realistic scenarios as possible. When planning each test, we identified the relevant components along with their expected state when beginning the test, then identified the expected state of the relevant components after executing the test. After conducting the analysis, we ensured any non-relevant variables wouldn't interfere with test results by setting them to controlled values. We also employed mock Graphics2D objects with the Mockito framework to ensure that graphics were being rendered properly.

Test Coverage



Factory Escape

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
main		93%		82%	30	112	52	424	7	30	1	6
entity		90%		78%	25	94	33	275	2	23	0	4
object		96%		73%	7	23	10	101	0	10	0	8
tile		99%		86%	3	16	2	58	0	5	0	2
Total	309 of 4,612	93%	65 of 332	80%	65	245	97	858	9	68	1	20

Our test suite currently achieves 93% line coverage and 80% branch coverage. More details about coverage can be found in the automated coverage report, generated by following the directions in our readme.md file.

Code segments that weren't covered included:

- Driver code (main.Main, main.GamePanel)

Driver code that was responsible for initializing and running our full program in main.Main and main.GamePanel wasn't tested because this code was mainly responsible for initializing, invoking and storing the other components of our program. Any possible issues with initialization would be highly visible during manual testing and any other faults would be in the classes invoked by the driver code rather than the driver code itself.

- Catch statements in try-catch blocks

Many try-catch blocks were used in methods that loaded resources and the corresponding catch block simply printed a stack trace when a resource couldn't be found. We verified that resources could be loaded properly in try blocks, but felt it was not necessary to test the catch blocks since printing a stack trace is not prone to error. (examples include entity.Guard.loadGuardImage)

- Similar cases in control-flow statements

Certain if-else and switch cases had very similar logic for each case so we tested a subset of cases to verify the behavior for all other cases. These similar cases were structured in such a way that we couldn't refactor control-flow statements into a single statement that could handle all cases more generally. An example of this is in entity.Guard.draw where a switch statement covers various cases for drawing an image in a similar way but can't be decomposed in a single statement.

- entity.GuardManager

GuardManager is a wrapper class that holds multiple Guard objects and calls the same method on each Guard. Verifying that individual Guard objects function as expected is sufficient to ensure the GuardManager will function as expected.

- While branches in tile.TileManager

The TileManager class contains several while loops that iterate through map tiles in methods such as TileManager.draw() and TileManager.loadMap. Some branch coverage in this class may be missing because cases where the while loop is not executed are not tested because a case where no tiles are present on the map is not possible.

Test Class Summary

Class	**Purpose**	**By**	**Unit or Integration**	**Bugs identified/code refactored/changes made**
TileManager Test	Test TileManager methods	Alex	U	None
CollisionTest	Test CollisionChecker methods	Alex	U	Added handling for out of bounds checks in CollisionChecker
ObjectSpawnerTest	Test expected behavior of ObjectSpawner	Alex	U	Fixed bug that caused premature object deletion in ObjectSpawner
ObjectSpawnerIntegrationTest	Test interaction between ObjectSpawner and blocking tiles and ObjectSpawner and other preset SuperObjects	Alex	I	Removed hard coded spawn boundaries in ObjectSpawner, instead basing it off of the TileManager object in the GamePanel
GuardPlayer CollisionTest	Test behavior when guard and player collide	Alex	I	Update behavior of isCollidingWithPlayer method in guard class to also consider the player's collision box
RenderAfter ChangeTest	Check that adding or removing SuperObjects is reflected when	Alex	I	None

	drawing the next frame			
PlayerMovementTest	Checks that WASD keys being pressed leads to change in player direction and player speed changes in correct direction.	Brandon	U	None
EnemyCollisionIntegrationTest	Checks guard's direction changes after a collision and "collisionOn" status is set to true	Brandon	I	None
PlayerObjectIntegrationTest	Checks player colliding with all types of objects result in correct change score, game state, and keycards collected	Brandon	I	None
KeyHandler Test	Test the Correct Implementation of the KeyHandler Class	Aaron	U	None
ScoreSystemTest	Test the Correct Implementation of the Score Keeping System	Aaron	U	None
ImageRenderingTest	Test image rendering and sprites	Nathan	U	None
UI	UI unit test	Nathan	U	None
KeyboardIntegrationTest	Check the different keyboard input for ui navigation that will change the game state	Nathan	I	None
KeyboardAndCollisionITest	Test player collide with object or walls by different keyboard input	Nathan	I	None

Findings

For the most part, our tests confirmed that our program was functioning as expected. Only one bug was identified during testing relating to the ObjectSpawner class where batteries were being deleted prematurely after 20s instead of 25s as specified. Besides fixing this bug, other minor adjustments were made such as modifying the main.CollisionChecker class to also trigger collisions when attempting to move outside of the map.