Aaron Lopez (301434253)
Brandon Quon (301545786)
Sepehr Borji (301372450)
Jordan McKenzie (301551839)

Unit and Integration Tests

Feature to test and a brief description of each:

- Game boot-up and Initialization tests that GUI, map, grid, and sprites load properly on startup
- GUI navigation and state, testing to confirm that the user can navigate to all available screens
- Game Over/Victory conditions to test that victory conditions and game over conditions are working
- Collision tests to test the functionality of collisions between characters
- Scoreboard tests to test the functionality of processing, updating, and receiving the score as well as writing this score to a text file to be saved

Manual Tests:

- Button Testing
    - Boundary Testing: Testing the extreme positions of the button to ensure that clicking within the expected area triggers the intended action.
    - Negative Testing: Testing scenarios where clicking outside the designated clickable area does not trigger any action, ensuring that accidental clicks do not result in unintended behaviour.
    - Multiple Button Interaction: Testing scenarios involving interactive elements in close proximity, ensuring that adjacent buttons do not interfere with each other's functionality and that each button responds correctly to user input without unintended side effects.
    - Collision feedback: as part of collision testing, manual testing was also done to ensure that all types of collisions are registered on top of the conditions and edge cases tested in our unit tests for collisions

Aaron Lopez (301434253)
Brandon Quon (301545786)
Sepehr Borji (301372450)
Jordan McKenzie (301551839)

Test Quality and Coverage

- Game boot-up and initialization cover the proper booting of the app using the processing Java library, the startup of the GUI manager, the setup of screens for GUI navigation, and the setup of the map and sprites for the core game loop. This will cover a lot of functions involved in the GameManager and GUIManager for drawing sprites, buttons, maps, etc. This test has full coverage.
- GUI navigation simply tests that the GUI can enter different states properly, and if so, we can navigate to and see the different screens such as for seeing the scoreboard. When the user changes state, this also tests small accessory functions such as the hover effect and its associated helper functions when a user hovers their mouse over a button on the GUI. This test covers all conditions and states of the GUI and its navigation
- The game win and loss test confirms that when the loss/win conditions are met, the game ends in a way that reflects the player's performance and the conditions met.
- Instantiation testing tests that the objects/characters that must be instantiated for the core game loop are instantiated properly and in the right quantity
- Collision tests force collisions to occur between player and non-player characters and test the result of this. In collisions between players and enemies, the player will lose oxygen. In collisions with collectibles, the player will collect the collectible, and when there is no space to collide (edge case), no collision can occur. As such, the collision tests have full conditional coverage and even test for edge cases such as if there is no space for any possible collision
- Enemy patrol tests ensure that the enemy is correctly moving towards the player in all directions.
- The scoreboard test covered all functionality involved in the scoreboard and managing the high score. It also tested this functionality checking boundary conditions such as if scores can be negative, 0, or what happens if we try to get the score when the score file is empty, etc.

Aaron Lopez (301434253)
Brandon Quon (301545786)
Sepehr Borji (301372450)
Jordan McKenzie (301551839)

Coverage:

Element	Class, %	Method, %	Line, %
all	88% (24/27)	65% (52/80)	54% (320/591)
Battery	100% (1/1)	100% (2/2)	100% (3/3)
Blackhole	100% (1/1)	66% (2/3)	83% (10/12)
Cell	100% (1/1)	50% (1/2)	91% (11/12)
Character	100% (2/2)	100% (4/4)	96% (28/29)
Collectable	100% (1/1)	50% (1/2)	12% (1/8)
Directions	100% (1/1)	100% (2/2)	100% (2/2)
EndTile	100% (1/1)	33% (1/3)	14% (1/7)
Enemy	100% (1/1)	100% (1/1)	100% (1/1)
GameEndListener	100% (0/0)	100% (0/0)	100% (0/0)
GameManager	100% (2/2)	100% (11/11)	98% (92/93)
Gameobject	100% (1/1)	75% (3/4)	58% (10/17)
GUIManager	100% (2/2)	14% (2/14)	13% (27/198)
GUIState	100% (1/1)	100% (2/2)	100% (2/2)
Main	0% (0/1)	0% (0/1)	0% (0/1)
Map	100% (1/1)	100% (1/1)	100% (51/51)
Mapobject	100% (1/1)	100% (1/1)	100% (1/1)
Objects	100% (1/1)	100% (2/2)	100% (2/2)
OxygenTank	100% (1/1)	66% (2/3)	57% (4/7)
Player	100% (1/1)	66% (2/3)	76% (20/26)
Processing	0% (0/2)	0% (0/7)	0% (0/61)
Scoreboard	100% (1/1)	100% (6/6)	88% (32/36)
Spike	100% (1/1)	100% (2/2)	100% (2/2)
Transform	100% (1/1)	100% (1/1)	100% (7/7)
WalkingAlien	100% (1/1)	100% (3/3)	100% (13/13)

Note: There are some methods that cannot be tested by JUnit, and thus had to be manually tested. This includes all draw methods in each of the classes, the overridden methods in our processing class, as well as a lot of the methods in the GUIManager class. These methods were entirely visual (i.e, dealt with drawing sprites to the screen), and had no logic that could be tested numerically.

Aaron Lopez (301434253)
Brandon Quon (301545786)
Sepehr Borji (301372450)
Jordan McKenzie (301551839)

Findings

Bugs/Code quality improved:

- The scoreboard functionality had an edge case that had to be accommodated regarding non-negative scores, so we added functionality to ensure that our score never goes below 0 and that if it is 0, we don't store this value, because a score of 0 is not something any player will want to brag about on the high score scoreboard
- The collision-checking functionality had an edge case that required solving. This edge case checked for a collision on a non-existent game grid. To avoid a null referencing error (because the game grid is null), we simply needed to check if we had a game grid. If we didn't, a collision is impossible, so there is no need to check for a collision!