CMPT 276 Project Phase 3 (Testing) Report

Group 10
Ewan Brinkman, Ariel Lin, Taaibah Malik, Hoi Chun Hogan Mok

Unit Tests

SpriteTest.java

- Assert the sprite getWidth method gives the correct width
- Assert the sprite getHeight method gives the correct height
- Assert the sprite getCenter method gives the correct center
- Assert the sprite getCurrentImageWidth method gives the correct current image width
- Assert the sprite getCurrentImageHeight method gives the correct current image height

AppTest.java

- Assert that the initial game state is correct (starts on the title screen)
- Assert that the settings button works: makes settings panel visible and hides title screen panel
- Assert that the choose avatar button works: makes avatar panel visible and hides the title screen panel
- Assert that the audio button properly turns game audio on/off
- Assert that the how to play panel is made visible when button is pressed, and that the settings panel is hidden
- Assert that the back button from the how to play screen makes the settings panel visible and hides the how to play panel
- Assert that the back button from the settings page makes the title screen panel visible and hides the settings panel
- Assert that each avatar button correctly selects each avatar
- Assert that the back button from the avatar page hides the choose avatar panel and makes the title panel visible
- Assert that the game panel is made visible when the game button is pressed, and that the title screen panel is hidden
- Assert that the game over screen panel is made visible when the game ends/player dies
- Assert that the return to title screen button works: hides the game over screen panel and makes the title screen panel visible

VectorTest.java

- Assert that the getVectorScaledToMax method returns a Point2D.Double with correct X and Y values
- Assert that the getVectorAngle method returns the correct arctangent angle for the vector
- Assert that the getVectorMagnitude method returns the correct distance of the vector to the origin

GameTest.java

- Assert that the size of enemy sprite array list is empty while starting the game loop
- Assert that the player position spawned in the correct position in testSetUpGame
- Assert that the game is not running in after the game is ended
- Assert that the game score is correctly modified
- Assert that the screen pixel height is correct
- Assert that the screen pixel width is correct
- Assert the the enemy sprite id array list is empty after deleting all the enemies
- Assert that isEneteredDoor is true after entering the door
- Assert that the player type is correct
- Assert that the game duration is starting from zero
- Assert the the regular reward id is empty after deleting all the regular reward

InputManagerTest.java

- Assert all valid key presses registers the correct change
- Assert an invalid key press does not register any change
- Assert all valid key releases registers the correct change
- Assert an invalid key release does not register any change
- Assert typing a key has no effect (our game only detects key presses and releases, key typing is a different type of event)
- Assert holding down a key keeps triggering the correct update
- Assert resetting stored key input actually clears stored key input

AudioManagerTest.java

- Assert that the audio is not null in testGetAudio
- Assert that the audio isMuted
- Assert that the audio is not muted
- Assert that is playing once
- Assert that is playing on loop

LevelTest.java

- Assert that the level name is correct
- Assert that the tile height and width is correct
- Assert that the size of enemies sprite id is correct

TextGenerator.java

- Assert the the durationText is correct

CollisionCheckerTest.java

- Try creating the class
- Assert collisions are correctly determined without any overlap
- Assert collisions are correctly determined with overlap
- Assert intersections are correctly determined without any overlap
- Assert intersections are correctly determined with overlap

CameraManagerTest.java:

- Includes a bunch of assertions for correctly applying an offset for the nontarget sprite, the target sprite, and points with and without entities that they are related to (attached to). A point is only drawn if it is on the screen and if its attached entity (if it has one) is on the screen. Also, make sure a value of null is returned to signal no drawing if the drawing result will not appear on the screen.

DrawOffsetTest.java

- Assert getting the offset without flipping is the correct result
- Assert getting the offset with flipping is the correct result

All factories in the factory package in the entity package: test creating the given entity, making sure it is created with the correct position

EntityTest.java: test adding and removing the entity from the game

DoorTest.java: test removing the door from the game, and opening the door

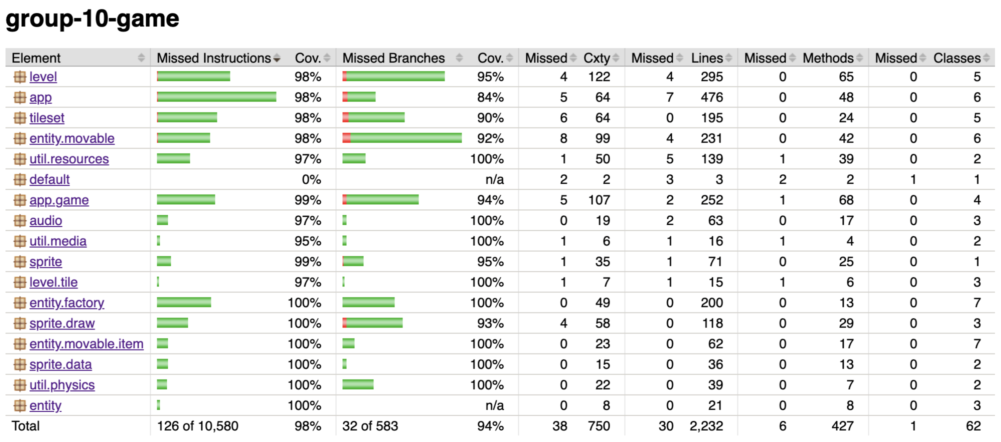
Integration Tests

- These integration tests are in AppIntegrationTest.java*:*
    - These integration tests all begin by starting the game from the main menu and clicking play:
        - Four tests each moving in a direction (one test for moving up, another for down, another for left, and another for right) and hitting a wall

- Testing the player collides with the regular reward and changing the game score
- Testing the player collides with the bonus reward and changing the game score
- Testing the player collides with the enemies and drawing out the game over screen
- Testing the draw debug mode doesn't crash
- Testing once the player collects all regular reward and they enter the door
- Testing if the flip level map layout works, when the player flips onto a wall and the flip is rejected, and when an enemy flips onto a wall and should freeze (and thus not be able to reach the player and end the game)

Test Quality and Coverage

An important part of testing is ensuring to create high quality test cases. As a team, we created a list of features to cover by discussing everything we implemented in Phase 2. After writing out what needed to be tested, we divided the work amongst ourselves. To monitor coverage as we developed tests, we used the JaCoCo plugin for Maven. This plugin tracks code coverage, and displays it visually for easy analysis. The screenshot below was taken after completing all of our tests - as you can see, we managed to achieve 94% branch coverage, and approximately 98.6% line coverage, calculated with the formula (((2232-30)/2232)*100)% using the total lines (2232) and lines missed (30). The exact percentage of branch coverage and the number of lines covered are also displayed. To access this page, you can navigate to ./target/site/jacoco/index.html after running the tests.



group-10-game

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
level		98%		95%	4	122	4	295	0	65	0	5
app		98%		84%	5	64	7	476	0	48	0	6
tileset		98%		90%	6	64	0	195	0	24	0	5
entity.movable		98%		92%	8	99	4	231	0	42	0	6
util.resources		97%		100%	1	50	5	139	1	39	0	2
default		0%		n/a	2	2	3	3	2	2	1	1
app.game		99%		94%	5	107	2	252	1	68	0	4
audio		97%		100%	0	19	2	63	0	17	0	3
util.media		95%		100%	1	6	1	16	1	4	0	2
sprite		99%		95%	1	35	1	71	0	25	0	1
level.tile		97%		100%	1	7	1	15	1	6	0	3
entity.factory		100%		100%	0	49	0	200	0	13	0	7
sprite.draw		100%		93%	4	58	0	118	0	29	0	3
entity.movable.item		100%		100%	0	23	0	62	0	17	0	7
sprite.data		100%		100%	0	15	0	36	0	13	0	2
util.physics		100%		100%	0	22	0	39	0	7	0	2
entity		100%		n/a	0	8	0	21	0	8	0	3
Total	126 of 10,580	98%	32 of 583	94%	38	750	30	2,232	6	427	1	62

We implemented a couple of features to help with increasing our test quality as well. First, we made the game automatically adjust its size to be larger or smaller than the level size for testing the camera. Another feature is that in the input manager test class, we simulated key presses by directly calling the input manager key press event method. There were a few features that we could not implement tests for. Firstly, we could not test lines 49 and 50 of Audio.java, which only execute if the audio loaded has an error. Similarly, lines 73, 74, and 80 of App.java only run if the app icon does not successfully load.

Findings

As we worked on this phase of the project we learned a lot, particularly in regards to testing. We learned that any messy code makes writing tests harder, and that some methods would benefit from refactoring. We made some changes to improve our code quality and fix the bugs we found. Below are a list of the changes found from testing and refactoring:

- InputManager: Could always set IS_PRESSED to true without having to check an if statement
- Sprite: Position hitboxes were being updated more than they needed to be
- CameraManager: Changed to only need the game in its constructor, having a window passed in too was redundant, since the window was the game
- Level.java: loadCurrentLevel had the levelId as a parameter when it shouldn't have
- CameraManager.java: Already knew drawing on the boundary wasn't working entirely (would not draw if near the top boundary when actually should draw). However, from testing it was also realized that it was drawing off the right side of the screen when it should not have been doing so.
- The MapManager class onLastLevel method was checking if all levels were complete if the current level index of the list of levels in the map manager equaled the maximum level number set in a JSON config file. This is problematic if more levels get added during the game, since the levels array size changes and the comparison to a preset value will no longer be accurate. Technically, this wasn't causing problems in our code, but from testing it was realized we should instead check if the current level index is at the end of the list of levels. This avoids further problems.
- Found by refactoring: Found unnecessary passing of owner position when creating a Hitbox, since the owner sprite is passed, so the position can be accessed directly from the owner sprite instance