CMPT 276 - Phase 3 Report

Group 22:

Michael Plunkett (mplunket@sfu.ca)
Dina Zeng (dhz1@sfu.ca)
Sameer Hossain (sameerh@sfu.ca)
Salman Rafiei (salman_rafiei@sfu.ca)

Unit tests:

PlayerTest - (Sameer)

- We made 10 unit tests using *assertEquals*:
    - 1 player creation test ,
    - 4 player movement tests in each direction,
    - 2 tests to check player death when his health or points hits or goes below 0,
    - 2 tests for checking the result when player health is adjusted,
        - max and non-max health;
    - 1 updating player sprites test.

ComponentFactoryTest - (Michael)

- We made 4 unit tests:
    - One of them is an empty file,
    - One contains a standard 9x10 map,
        - Describes use of all 9 types of tiles defined in our program;
    - One contains a triangular shaped 18x18 map,
        - Describes use of all 9 types of tiles defined in our program;
    - One contains a maximally sized 50x50 map;

- Tests the loadMap method on 4 different test map text files found in the test resources folder.

EnemyFactoryTest - (Michael/Dina)

- We made 2 unit tests:
    - One of them contains one of each enemy,
    - One of them is empty;

- Tests the createEnemies method on 2 different test enemies text files;
- Asserts that each enemy loaded was of the correct class.

SkeletonTest - (Salman)

- We made 3 Unit tests.
- The first test confirms creation works.
- Second test confirms two functions
  - Adding to the path array works properly
  - Class checks the validity of a path correctly; examined by comparing the path before and after it is completed
- Last test verifies setting images for skeleton type works as expected

ItemFactoryTest - (Michael/Dina)

- CreateItems method:
  - We made 2 unit tests:
    - One of them contains one of each type of item,
    - One of them is empty;
  - 2 different test item text files;
  - Asserted that each item loaded was of the correct class, and that the ItemFactory counted the correct number of keys;

- spawnPotions method:
  - Used the big 50x50 map of floor tiles from the ComponentFactoryTest unit test;
  - Placed a player in the center (25, 25);
  - Called spawnPotion;
  - Asserted that the resulting potion is indeed within 10 squares of the player,
    - not on the player;
  - looped 100 times within the unit test to account for the random nature of the spawnPotion method.

MapComponentTest - (Salman)

- We made 4 unit tests:
  - One of them shows tile creation works;
  - One of them examines the modification of the tiles collision status;
  - One of them creates an item (key) and attaches it to tile;
  - One of them attaches an image to the tile.

UITest - (Dina)

- 2 Tests were made for this Unit:
  - The first test checks if the new instance of the UI class is not null;

- The second test checks if the getters and setters of the UI class are working as expected.

Integration tests:

PlayerItemCollisionTest - (Michael)

- We made 7 integration tests;
- Tests the playerInteraction method in the Player class;
- One test for each of the items:
    - Key,
    - Spikes,
    - Door,
    - Potion (four for this particular object);
- For each item:
    - Ensured that the player's health, points, and keyCount are adjusted appropriately;
    - For the potion tests:
        - Had 4 different cases of health adjustment:
            - The player's health is full,
            - Missing less than 50,
            - Missing exactly 50,
            - Missing more than 50 health;
        - Note: the potion restores 50 health.

CharacterCharacterCollisionTest - (Michael)

- We made 6 integration tests:
    - Player placed at (5, 5):
        - Enemy placed at (5, 5),
        - Enemy placed at (5, 10),
        - Enemy placed at (10, 5),
        - Enemy placed at (10, 10);
    - Enemy placed at (10, 10):
        - Second Enemy placed at (5, 5),
        - Second Enemy placed at (10, 10);
- Tests the checkEntity method in the CollisionChecker class;
- Asserts checkEntity returns the correct index of the enemy the player or other enemy collides within the enemies array,

    - or -1 in the case of no collision;
- Asserts player or enemy's collision is appropriately set;
- Uses an assortment of enemies across all tests to cover more cases.

PlayerMovementTest - (Sameer)

- We made 4 integration tests using *assertTrue*:
    - Up,
    - Down,
    - Left,
    - Right;
- Verifies player movement in each direction depending on what keys are pressed.

Test Quality + Coverage:

- For our project we started out by listing the main features of our game that would be tested to ensure we have the most coverage with the least amount of test cases. For example, the player itself has many features, movement, interactions, etc.

- Since our game is written in object oriented patterns and many of our methods did not take any patterns we couldn't use a functional approach. Hence, instead of testing out different input values, we designed unit tests based on specific class attributes like score and player health.

- To ensure the quality of our tests, we first cross checked our tests with the number of features in our code. That is, movement, collision, UI, class creation, etc. We then ensured that we had at least a few tests for each of the features. Then, we ran through our code, following the flow of the program, and added tests to check variables and methods throughout. We also checked all of the conditions in our program whenever possible (i.e. the condition was not in a private void method). Hence, we were able to conclude that we had good quality tests.

Findings:

- There were many findings, changes, and moments of eureka while we were creating tests for our projects.

- For starters, it was difficult for us to test some classes/methods since many of our classes/methods had the private modifier and returned void. We had to test methods that called these private methods in order to test them. During this process, we discovered we had a lot of dead code - such as unused getters, or variables. We were

able to declutter our code by removing this dead code. This also improved the quality of our code since we refactored the dead methods out of our other methods/classes.

- We also discovered the source of some bugs in our program. For instance, whenever the player is immobile, that is, the user is not pressing any arrow keys, and an enemy collides with the player, the program does not call the method to check for collision and update the player/game state from that collision. We discovered that there was a condition statement in our update() method that only checks for collision when the player is pressing arrow keys. We resolved the bug by removing that condition statement.

- Additionally, we added the options to restart the game or return to the main menu whenever the game ends. Originally, this was added to help us with testing our code. However, we also realized that adding the options improved our game flow. Before, the player was unable to play the game again without rerunning the program/closing and reopening the game. Now, the player can play again when the game ends much easier.

Changes made:

- Added option to allow player to restart game or return to main menu after game ends
    - Press R to restart the game
    - Press Q to return to the main menu
    - Makes testing easier
- Level 1 created
- Problematic skeleton on level 2 replaced
- Fixed bug where player only collides with an enemy if they are moving
    - If the player is not moving (no key pressed), the slime will not collide with player and the game will end when the player moves if the slime is within the player's hit box.