CMPT 276 – Group 2

Phase 3 - Report

Sahba Hajihoseini, Dylan Kurath, David Ligocki, John Mitton

Unit and Integration Tests

Our main intention in the 3rd phase was testing the game and fixing the bugs. First we identified all the futures that needed to be unit tested and ran automated tests:

- AppTest.java: appTest makes sure the app can be constructed and that the state starts on the main menu mapsTest tests if the maps class is created, tests if the mapHeight and mapWidth functions return positive values, and tests if the initmap function adds entities to the entity manager spritesheettest tests if it can load the image in the constructor and tests all possible values for grabbing an image from the spritesheet.
- To test entityManager, first, we implemented a constructor test. Then, we made another test to check if the code successfully creates and finds the entities (newEntityTest() and findEntityTest()). We also tested removeFromList() for removing specific entities from our list of entities, and then we tested clearListTest() for removing all entities from the list. updateTest() ensures that entityManager interacts with bonus rewards to remove bonus rewards from the list of entities after they have been through a duration of 10 updates (equivalent to 10 ticks).
- We designed entityTest.java to test the methods of the entity class. First we made the constructorTest() and then, implemented other simple value manipulating tests such as setPositionTest() and addScoreTest(). We also checked interactions with the entityList from entityManager in findCharacterInListTest(). Then we checked that entities detect and respond to other entities in nearby tiles correctly with moveTest() and collisionTest(), as well as using moveTest() to check entity.java's private function canMove().
- We made a constructor test case for our barriers (BarrierTest.java).
- Bonus rewards needed a constructor test and a test for tiktimer. bonusTikTimerTest() checks if the bonus rewards spawn and disappear correctly.

- MovingEnemyTest.java checks the methods of MovingEnemy class. In addition to the constructor test, we added characterCollisionTest() to test the interaction where the enemy and character exist on the same tile, by confirming that the game sets the score to 0 and the user loses. findShortestPathTest() checks that the behaviour of the moving enemies are consistent with what is expected from the implementation of findShortestPath().
- PunishmentTest.java and RegularRewardsTest.java have constructor tests for Punishment class and RegularRewards class respectively.
- userCharacterTest.java has a constructor test and other methods to test if the game gets and interprets the correct values from userInput.java (rightHeldTest(), leftHeldTest(), upHeldTest(), downHeldTest() and, moveTest()). This class also tests the character's score interactions with other entities through updates after encountering enemies, punishments, regular rewards and bonus rewards (testUpdate()).

Test Quality and Coverage

To enhance the quality of our code, we made multiple test cases for functions to be specific to certain executions for more branch and condition coverage. We made assertions based on the expected results we calculated and compared them with the actual results. This process helped us detect the existence of bugs in certain series of events and fix them by working backwards to the cause of the problem. We tried to keep test cases simple so we could isolate issues with specific functions, but also complex enough to catch cases which are not the most expected action (trying to prevent developer bias).

We didn't use automated testing for our rendering functions because we thought it was more sensible to test UI by running the app and observing through random functional testing. We found that testing this was important because some bugs that show up in the UI may not prevent code from functioning but would still create issues for the user. We were able to detect issues we saw through the UI testing and fix them through more thorough analysis of the code.

Findings

From this phase of testing we learned that in cases like this app where there is not a lot of complexities occurring behind the scenes, we can detect most issues through testing the functionality manually.

Moreover, we made a few changes to the production code during the testing phase:

- changing implementation of clearList() in entityManager
- Deleting stateManager.java
- Fixing GAMEOVER or GAMEWON by clearing entitylist after state change
- Updating levelfinish, mainmenu, character sprite changes, removing pause functions from the entityManager,
- Adding getters for upheld, downheld, rightheld, and leftheld in userCharacter class

We also edited the main menu. The new menu is more detailed and helps the player understand the game. Further, We added 4 more characters - 5 characters in total. 4 of the characters are named after us (Sahba, David, Dylan, John). Besides, we tried to design them based on our own appearance! However, we have a guest character which allows the user to play with it if they do not want to play with our characters.

We made changes to our code before implementing automated tests, because we discovered problems through random functional and UI testing. Due to the modification of our maps top row being shifted down one to prevent UI clashing with the score and timer, we adjusted our bonusReward spawning algorithm to spawn only within the playable area. We also prevented issues where too many userCharacters and movingEnemies would exist because the list of entities was not being cleared after each execution of the game.

We were able to find only a few bugs through unit and integration tests and we solved them and confirmed their functionality was improved. In canMove() we removed some elseif statements because their result caused other more important conditions in the chain of if statements to be skipped, and since these removed statements had the same output as our default output we did not need to include them. We improved the quality of the code through our testing and debugging, but we did not include the refactoring from Assignment 3 into our code.