CMPT 276: Phase 3 Report

Efe Erhan

Brianna Espena

Ruochun (Jasmine) Liu

Yiwen (Moon) Wang

AppTest:

Every complex method was tested, which consisted of the ScoreTracker updater and the constructor. An integrated test of AppWindow and Board was also implemented, which checks the interactions between the two classes.

Coverage:

updateScoreTrackerTest:

- Line - 100%
- Branch - 100%

integratedWindowAndBoardTest:

- Line - 99%
- Branch - 100%

BoardTest

The tests consisted of all the complex methods that weren't included in the integrated tests with AppWindow. Every class that inherits baseElement was added to a Board, and interaction between them and the trackers was tested. The Board's collision checkers were tested as well. There was some overlap with AppTest's integrated test, so coverage wasn't very complete.

Coverage:

enemyCollisionTest

- Line - 29%
- Branch - 4%

playerCollisionTest

- Line - 29%
- Branch - 4%

collectTest:

- Line - 21%
- Branch - 25%

addTest:

- Line - 92%
- Branch - 100%

ScoreTrackerTest

We tested the method “addMark()” and the method “addRealMark()” in unit tests. Increasing and decreasing marks work in both methods. Both methods pass the tests. The methods “getMarks()” and “getRealMarks()” are tested in the method “addMarksTesting()” and “addRealMarksTesting()”. If these two methods “getMarks()” and “getRealMarks()” are not working, tests will fail. All the corner cases were tested.

Coverage:

- Line - 100%
- Branch - 100%

TimeTrackerTest

The methods “addSeconds()” and “getSeconds()” were tested in the class “TimeTrackerTest” and passed the test. All the corner cases were tested. The methods “stopTimer()”, “startTimer()”, and “actionPerformed()” were not tested here, since they were tested in the integration test.

Coverage:

- Line - 50%
- Branch - 100%

ScoreTrackerViewTest

Test whether the class “ScoreTracker” ‘s object still runs well in the class “ScoreTrackerView”.

The method “ScoreTrackerViewTest()” tests whether “ScoreTracker” is initialized in the position we want, and the two variables of the “ScoreTracker” have the correct initialized value. The test method “addMarkTest()” tests whether the function of the class “ScoreTracker”, “addMarks()” still works in the function of the class “ScoreTrackerView()”. The method “addRealMarks” in class “ScoreTrackerView” was tested in method “addRealMarksTest”. The two getter methods are also included in this test.

Coverage:

- Line - 94.4%
- Branch - 100%

TimeTrackerViewTest

Test whether the “TimeTrackerView” object initialized correctly in its expected position.The position and initialization of the object were tested.
Coverage:

- Line - 33.3%
- Branch - 100%

TimeTrackerIntegrationTest
“TimeTrackerView” integrated into the class “Board”. The class “Board” called the “TimeTrackerView” object to start and stop the timer. The class “TimeTrackerIntegrationTest”
tests whether “TimeTrackerView” works correctly in the class “Board”. We let it start time, wait 5 seconds, and stop. When running the test, User Interface shows the timer on the right corner and changes every second. After 5 seconds, the timer in the user interface shows 5, and the window is closed. In that way, we test the timer’s user interface. The user interface, logic and all the corner cases were tested as well.

RegularRewardsTest and BarrierTest (Unit test):
The unit test for RegularRewards class and Barrier class are quite similar. To make sure the class can be logically isolated in a system, we needed to test that it would be drawn and that it’d be drawn at the specified position. We built a positionTest to check the position. We called getX and getY to check whether they are what we input. Also, a drawTest was built to check if the draw succeeded at a new board with all-null spots.
Coverage:

RegularRewardsTest:

- Line:66%
- Branch:100%

BarrierTest:

- Line: 86.36%
- branch:100%

BonusIntegrationTest and regularRewardsIntegrationTest:
Methods and attributes from the “bonus” and “regularRewards” are integrated into the classes “AppWindow" and “scoreTracker”. Other classes would call the “addMarks()” methods to add marks when the player touches the bonus cell or regular reward cell. To test whether the image is drawn successfully we produced a new test board which is null at all positions. We drew the bonus or regular reward cell at a certain position then checked that the position was no longer null. We also tested whether it would print out

the “Grid spot already occupied” message when the position is not null. To test whether the marks are added when the player touched the bonus or regular reward cell, we used addMarks method, which is defined in ScoreTracker class. We added the mark first and checked if the scoreTracker returned the correct values.
Coverage (BonusIntegrationTest):

- Line: 81%
- Branch: 100%

Coverage (regularRewardsIntegrationTest):

- Line:66%
- Branch:100%

BarrierIntegrationTest:
It is integrated into the class “App” when a barrier needs to be drawn at a certain position. So the test needs to make sure the image is drawn at the right position. The main interaction between the barrier class and other classes was the drawing of the image and that right message was printed if the position to be drawn at was full. Similar to the bonus and regularRewards class we drew a barrier at a position on an all-null board, and checked that the position was no longer null. The checkGridSpot() method was implemented to test that the right message for that event was printed.
Coverage:

- Line: 86.36%
- Branch:100%

For the following three tests, we tested the complex methods, and the simpler methods that are called within them. We also tested the constructors for each. The actionPerformedTests in the character, spriteAnimListener (inner class of character) and MovingEnemy classes have lower coverage as they have multiple branches with very similar functionality and equal importance. In addition, the branches in those actionPerformed methods have very specific conditions. The class attributes accessed by actionPerformed methods were set to access and test specific branches. Not all values and attributes modified in the tested branch were checked since those methods were tested separately.

CharacterTest:

The animFrameUpdater method has only 50% coverage because the logic of the method is simple and the branch with the specific condition was tested.

Coverage:

shouldGetDirection:
- Line: 100%
- Branch: 100%

testKeyReleased:
- Line: 100%
- Branch: 100%

testKeyPressed:
- Line: 100%
- Branch: 100%

testActionPerformed:
- Line: 10%
- Branch: 25%

testSetLocation: setLocation method is from AWT library
- Line: 100%
- Branch: 100%

shouldAdjust:
- Line: 100%
- Branch: 100%

testAnimFrameUpdater:
- Line: 50%
- Branch: 50%

testActionPerformed_sAL:
- Line: 20%
- Branch: 14%

MovingEnemyTest:

Coverage:

adjustTest:
- Line: 100%
- Branch: 100%

shouldGetDirection:
- Line: 100%
- Branch: 100%

shouldGetIdentifier:

- Line: 100%
- Branch: 100%

shouldSetLocation: setLocation method is from AWT library

- Line: 100%
- Branch: 100%

testActionPerformed:

- Line: 47%
- Branch: 33%

NonAnimatedEnemyTest:

Coverage:

shouldGetIdentifier:

- Line: 100%
- Branch: 100%

testActionPerformed:

- Line: 100%
- Branch: 100%

Findings

Changes made to production code:

- certain attributes and methods in some classes were changed from private to protected so that they could be accessed and set during testing of methods in their respective class
- Before testing, the original classes, “ScoreTracker” and“TimeTracker”, contain both views and models which makes unit testing impossible, so we put the models into the class “ScoreTracker” and the class “TimeTracker” and put the views into the class “ScoreTrackerView” and the class “TimeTrackerView”.

The tests uncovered a missing feature: The game did not end when the player’s score fell below zero. There was also a bug that threw an error when checking for the moving enemy’s collision with the wall/door slots. Both of these issues are now fixed.