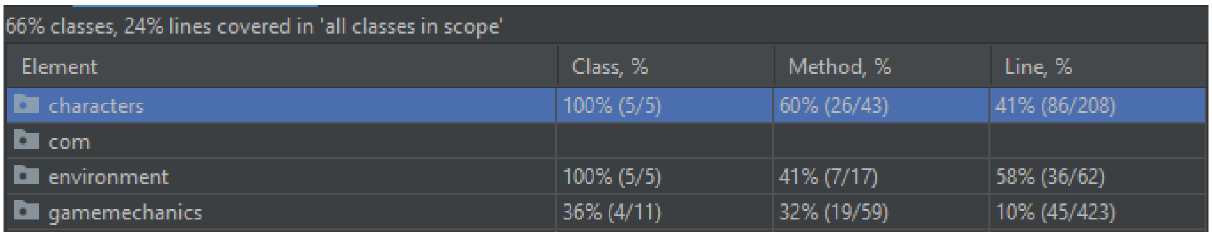
CMPT 276: Group Project

Phase 3

Group 3: Ethan Rowat, Luna Sang, Pavel Jordanov, Tai chuan david Png

Test Quality and Coverage

Measure, report, and discuss line and branch coverage of your tests. Document and explain the results in your report.



66% classes, 24% lines covered in 'all classes in scope'

Element	Class, %	Method, %	Line, %
characters	100% (5/5)	60% (26/43)	41% (86/208)
com			
environment	100% (5/5)	41% (7/17)	58% (36/62)
gamemechanics	36% (4/11)	32% (19/59)	10% (45/423)

- **Characters**

  Tests conducted on Wilbur included movement tests to see if he could move in all directions including up, down, left, and right. This was tested by comparing his actual x or y coordinate after performing a single move, with the expected x or y coordinate. Moreover, a conceptual test was conducted to test and see if Wilbur can move in a circle which would verify that all the move methods can work together. This was performed by performing the first four tests on movement all together into one test and then comparing both the x and y coordinates of Wilbur with the expected coordinates. Wilbur's ability to die was also tested. This was done by simply calling the kill method to try to kill Wilbur followed by a conditional test to see if his boolean value which determines his aliveness was changed. Lastly, his invincibility was tested in two different ways. The first test tests if Wilbur can die while being invincible. This was done by setting Wilbur to be invincible then calling the kill method. It was then checked if he was killable by checking if the state of his liveness changed. Finally, the last test was to make sure that the invincibility does last 5 seconds. Although the method used was probably not the most efficient method, it did confirm that the invincibility does last atleast 5 seconds. The way this was done was by timing how long the main thread would be asleep for while Wilbur was still invincible. This test does have some delays though since it takes extra time to calculate the time length for the invincibility. Due to this, the test would not always pass as there are a number of things that could slow down the program and cause the time to be longer than 5 seconds. The longest seen was just around 6 seconds. As we could not find a better solution we changed the assertion from an assertEquals to an assertTrue in which we made sure the invincibility was atleast 5 seconds and would not have to worry about the delay of the program.

Tests on Butcher included the tracking tests, movements around barriers and damage on Wilbur (the player) when butcher catches Wilbur. Tracking function is the most important feature of the Butcher class as the butchers are supposed to chase the player very efficiently and kill the player. Every branch of the tracking function was tested based on the relative position of Wilbur and Butcher. The tracking function also deals with interactions with barriers as when the next move is a barrier, the butcher needs to choose a second direction. So the move function that used to determine interactions with barriers was tested and all the barrier types were covered. Another important feature of the butcher is that when the butcher catches Wilbur, Wilbur gets killed. Test on whether Wilbur gets killed when butcher catches him was tested by assigning the same location as Wilbur to the butcher and check whether the isDead function of Wilbur returns true.

- **Game mechanics**

The tests created for the game mechanics objects included, tests on the coordinate class, and tests on the interactions that would happen on the Board class using the BoardSpace. The interaction with the MagicCorn, MagicTruffle, BearTrap, Butcher, the Board's exit and Wilbur were tested by moving Wilbur into a space containing the item and testing for an interaction. The outcome of these tests lead to revising how Wilbur's class modified points and achieved the results fo these interactions. The test tried to observe if the detection methods detected that the right type of interaction took place and that the correct result would be linked with the interaction. A set of tests were added to check if the coordinates were capable of detecting (and not falsely detecting) barriers from different directions around a moving character (namely Wilbur). Tests were also added to confirm that our game could indeed tell the difference or similarity between two sets of coordinates regardless of type of object that held them.

- **Environment**

The tests conducted on the Environment objects include: Magic Corn, Magic Mushroom and Bear Trap. These fall under the Stationary Modifier category and as such, a JUnit test was performed on the StationaryModifier class and each of the Stationary Modifier objects were tested in this class. Tests on the Magic Corn object were performed by testing if the Magic Corn would increment Wilbur's points by the correct amount. This was done by setting a

Wilbur object to have points set to 0 and then have it interact with the corn. An assert equal would then be made on the expected integer value that Wilbur will have once he interacts with the corn, and the actual score of the Wilbur object once interacting with the Magic Corn object. Tests for Bear Trap were similar in the sense that we'd assert the final points of the Wilbur object with an expected integer value. However, additional tests had to be performed on Wilbur's state after interacting with the Bear Trap as the Bear Trap kills Wilbur by setting his state from alive to dead. The tests on Wilbur's dead/alive state was done by asserting if the Wilbur object's state and an expected Boolean variable named wilburIsDead were equal. The Boolean variable wilburIsDead was initialised to false as Wilbur is not initially dead when the game is started. An interesting note about the Bear Trap tests is that we also tested for point decrements as the Bear Trap should simultaneously kill Wilbur and also decrement his points. Two tests were made in regards to this, one for when the expected points value after the interaction is positive, and one for when it is negative. This was to make sure that the Bear Trap was correctly decrementing Wilbur's points. A similar procedure was followed in testing the Magic Mushroom in that assertions were made between the actual Wilbur object points after interacting with the Magic Mushroom object and an expected integer value of Wilbur's final points after the interaction. However, further assertions had to be made on whether Wilbur would transition to an invincible state once interacting with the Magic Truffle. This was done by setting a Boolean variable wilburInvincibilityState to true, and then asserting equal if the Wilbur object's invincibility state was in fact true by calling the wilbur.isInvincible() method.

Discuss whether there are any features or code segments that are not covered and why.

There are some sections of code that are not covered by our JUnit tests, this is due to a number of reasons, one of them mainly being the logic being contained in a non instantiatable grouping of code. For example, the Board class that contains our GUI does not have reachable logic since we cannot run and inject keystrokes into the game easily. This is permissible since these sections of code are testable via black box tests (or are otherwise visible when running the game) for instance, our exit tile becomes uncovered once the player collects all of the MagicCorn on the board. We tested for this by running the game and collecting all the corn on the board. This also allowed us to test whether the timer, score and total number of rewards to collect were consistent with what was expected. Additionally the movement of Wilbur is bounded by the placement of barriers on the board. This is testable by trying to navigate Wilbur out of bounds and observing whether or not this is possible. The movement of Butchers is set up to be one cell for each tick, and could also be tested by running the game and observing whether the butchers are moving for each tick.

Findings

Briefly discuss what you have learned from writing and running your tests. Did you make any changes to the production code during the testing phase? Were you able to reveal and fix any bugs and/or improve the quality of your code in general? Discuss your more important findings in the report.

From writing the tests, we have learned that it is important to take a step back from the code to question not only how the code works but whether it truly accomplishes the tasks it sets out to do, and whether or not it is robust to errors. It is helpful to write tests in order to ensure that the particular code functions in the expected manner, and questioning whether there could be cases where the code does not function in the intended manner.

Writing the tests also allowed us to determine how cohesive and coupled our code was, there were a few cases where functionality was placed into the incorrect class and could be separated out into a more suited class. One example of this was the logic of the interactions with Wilbur (our main character) being dispatched by the Board, this was not suitable for our purposes since the effects of the interactions were almost exclusively applied to Wilbur. This allowed us to separate out the functionality of the interactions into the Wilbur class.

The testing also allowed us to discover when some code was too complicated or did not follow the law of Demeter. For instance, when replicating the comparison of the Coordinate class. The Coordinate class contains the x and y coordinates of our game grid, and since every character and item has a coordinate, we had to do calls such as character.getCoordinates().getxCoordinate() in order to obtain the x coordinate, and similarly for the y coordinate. This is not optimal since it involves two method calls every time we need one of these coordinates. The code could be simplified to create a getxCoordinate() within the StationaryModifier and MovingCharacter classes that reduces the calls to character.getxCoordinate() and similar for y.

Testing also required that the person writing the tests understood how the code worked. This translated to better documentation, and increased code clarity. For example, at first our main character used several methods to modify the points. This made the code more complicated than it had to be and made it difficult to follow the execution path. We decided that reducing the code to a single method that passed a parameter to instruct the code what execution to complete, was a simpler method to follow and reduced the complexity of our code by simplifying execution paths.

Testing is also a good way to examine the effectiveness of our code. For the butcher's tracking function, we found that the tracking was not effective enough to move towards a direction that makes the butcher closest to the current position of the main character. So we redesigned the algorithm by changing the movement condition based on the slope of the line between butcher and Wilbur. When |slope|<1, the movement priority will be given to horizontal directions; and when |slope|>1, the movement priority will be given in the vertical directions. Since there are barriers on the map, we also added the checking conditions when the next move is a barrier, a second choice direction will be chosen. The probability of the random move feature (the butchers move randomly for a given probability based on the probability parameter in the tracking function) is still maintained because we found that if the butcher keeps chasing Wilbur all the time, the game is extremely hard and therefore makes the player lose the fun of playing the game.

During our testing, a few bugs were revealed in our JUnit tests as well as our black box tests. We found that when Wilbur collected a magic truffle he would become invincible for 5 seconds (as intended) but if he collected another magic truffle during this 5 second period his invincibility would not be extended and instead he would return to his original state after the first 5 seconds. We were able to fix this bug by restarting the invincibility task as soon as another truffle was collected.

Another major issue that was discovered was attached to the movement of Wilbur. Wilbur was able to move very quickly on the board (due to his movement being triggered on key presses, which meant the key could be held down to move more than once). In addition, the Butcher's moving on periodic "ticks" meant that the Butcher's "tick" would sometimes miss the period of time in which Wilbur was on the grid tile the Butchers moved onto. This meant that Wilbur would not register the Butcher's attack and therefore would not cause Wilbur to die.. This was fixed by making the event that moved Wilbur a key release instead of a key press, which means that holding down a movement key would only register as a movement when the key is released. For example, if the left movement key was pressed then Wilbur will only move left once the User's finger is removed from the left movement key.

One change that had to be done to make all the Wilbur tests pass, was to add a condition to the kill method. This was due to the fact that Wilbur was able to die if the kill method was called even while being invincible. To fix this, a condition was added to the kill method which checks if Wilbur is invincible or not first, then based upon this boolean condition the kill method would actually execute.