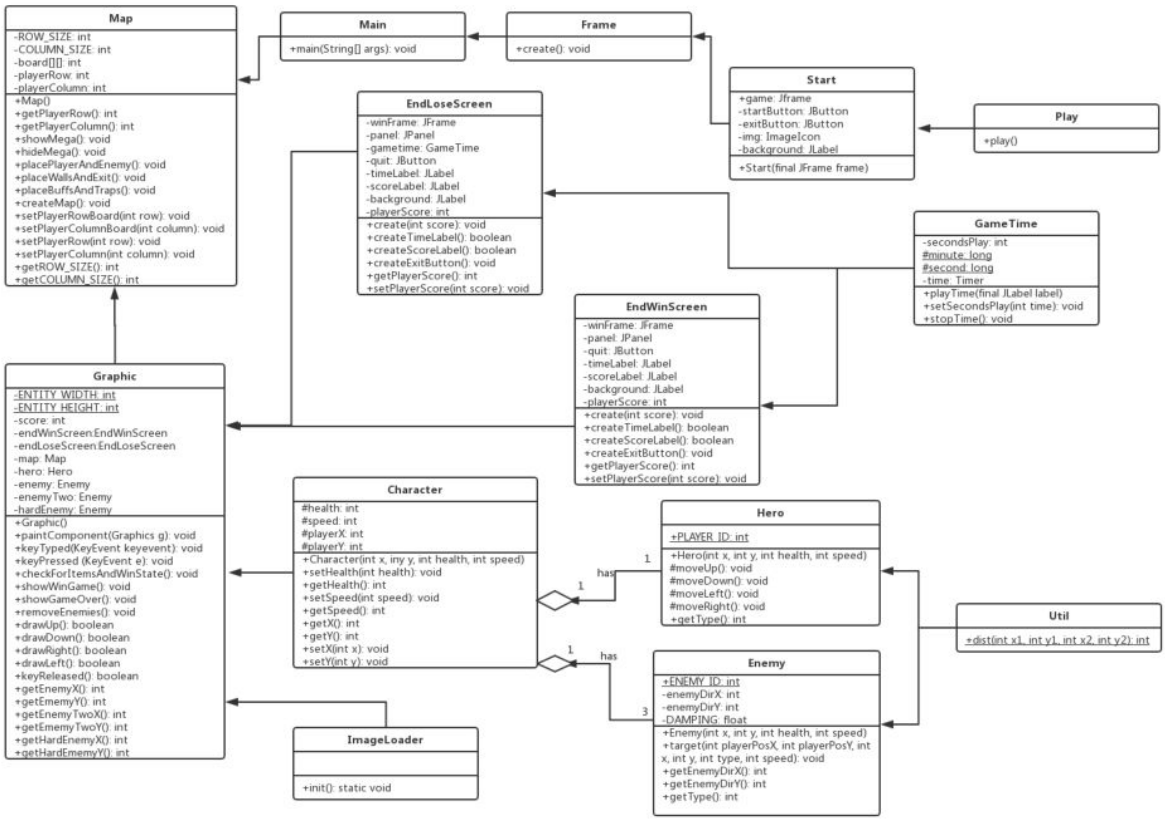
Phase 3 Report

Jovan Bapla, Dennis Zhang
Josh Amato, Kris Zhang

Refactor For Testing

Before beginning the testing phase, the group decided to refactor our production code for better readability and improve overall quality. This was done by creating new methods to encapsulate repeated code for better reusability and enhance better code design. Bugs appeared during this stage, but were quickly managed due to the easy accessibility that the methods provided. New classes were also constructed to load in all of the images required. This restricted us to only manipulating this class when there are alterations in the images while all other classes dependent on the images will update themselves.



Integration Testing

This part was difficult for the group because it was hard to come up with a way to get the actual tests to work for simulating the key presses. Using this integration testing in the Graphic class and testing key presses to make sure the graphics were moving to where they are supposed to and to see if the grid in Map class is also corresponding to the key presses and moving the character in the 2d array.

Tested Classes

The JUnit framework is the main source in helping us validate the functionality of the methods in the project. The first stage of JUnit testing was testing single features in isolation. The features that were targeted for testing were the classes that included individual code components to ensure the code works the way it is intended to. These included getter and setter methods, small utility methods, and any other methods that had a return value or take in parameters. The classes that covered these components were.

- Hero class
- Character class
- Enemy class
- Util class
- Map class
- ImageLoader class
- EndLoseScreen class
- EndWinScreen class
- Graphic class
- GameTime class

Hero class: This class consists of the player's ability to move around the map. It holds the hero's moveUp(), moveDown(), moveRight(), and moveLeft() methods which uses speed to illustrate the direction they are moving at. When each of these methods were called, they were tested to see if the player's position had changed according to the move by getting their X and Y positions. This class also carries a getType() method which just grabs the hero's type ID. This was tested by setting the expected value and comparing it with the actual value that the hero type ID holds.

Character class: This class makes sure all the character statistics and location is set. The tests that were made for this class made sure that the setters and getters were

working correctly for the player's character and that its location was also set to the given values.

Enemy class: This class deals with the enemy class targeting the hero. The way testing was approached with this class was to check the placement of the enemies in the beginning of the game as well as checking the enemies location later on with different cases. Checking the type of the object was also tested to make sure that they are the correctly desired type.

Util class: This class tests the Util class method which only includes the dist() method. The dist() method uses the distance formula to compare and find the distance between two objects. The test will individually evaluate if the dist() method will output the correct values when inputting them into the distance formula.

Map class: This class holds the logic behind the grid to the game. One of the tests makes sure that the wall detection is correct. Another test makes sure the traps are in the correct location by putting the player over them and testing to see if the trap reacts. Making sure the traps and bonus items disappear is also tested.

ImageLoader class: This class was used for better organization of the images. Where instead of having to put the URL links and directory trees inside different classes there was just one class that had them all for ease of access. The tests for this class were making sure that the images loaded correctly and to make sure that the images were found as well as the size of the images to make sure they are the proper dimensions.

EndLoseScreen class: This class is used to see the results of when a player loses the game with the time and score of the player. The tests created for this class was to verify that the time showed the same time as in the game and verified that different cases were to work such as a score of zero or a score greater than zero. Multiple times were tested as well such as having a time less than or greater than a minute to see if it was displaying correctly.

EndWinScreen class: This class is also used to see the results of a player but instead of for when the player loses it is for when the player wins. The same tests are applied with this class as were with EndLoseScreen. Which was checking the time both before and after the one minute mark to make sure everything displayed correctly. As well as checking that the score was correctly showing on the screen.

Graphic class: This is the class where all the visuals are being displayed. Testing for this class was done by checking if the win and lose condition of the player reaching zero health is met. As well as making sure the buffs are changing to disappear. The key pressed method is also tested which is talked more about in the integration testing section.

GameTime class: This class holds the logic for the timer that is used in the game. It uses the Swing Timer class to increment an integer variable called secondsPlay, in seconds, to keep track of time. There were two tests done for this class. The first was to test whether secondsPlay was properly being converted into the proper time. A test was made to display a time above 1 minute to check whether it was properly converted into minutes and seconds. Another test was made but this time for less than 1 minute, to ensure that only the seconds, and not minutes, were being calculated.

Not Tested Classes

-Play Class
-Start Class
-Frame Class

After having a back and forth with the professor and T.A, the talk inspired us that testing most UI classes such as Play, Start, and Frame were not needed because it was either not able to test properly using JUnit or mostly in this case because most things were not needed to test functionality wise. Such as checking for button placement or checking if a new screen would pop up. All of these types of instruction could be checked by just running the main game and seeing if they are doing what they are supposed to be or are showing on the screen in the right place. Overall, all the beginning JFrame and JPanel were just all visual with a few buttons that did simple tasks like exiting the program which can be seen through terminal so that having them be tested was redundant because it worked already. With the screens being mostly visual, the buttons and background images can be seen which means that they were being loaded in and felt writing tests for them would also be unnecessary.

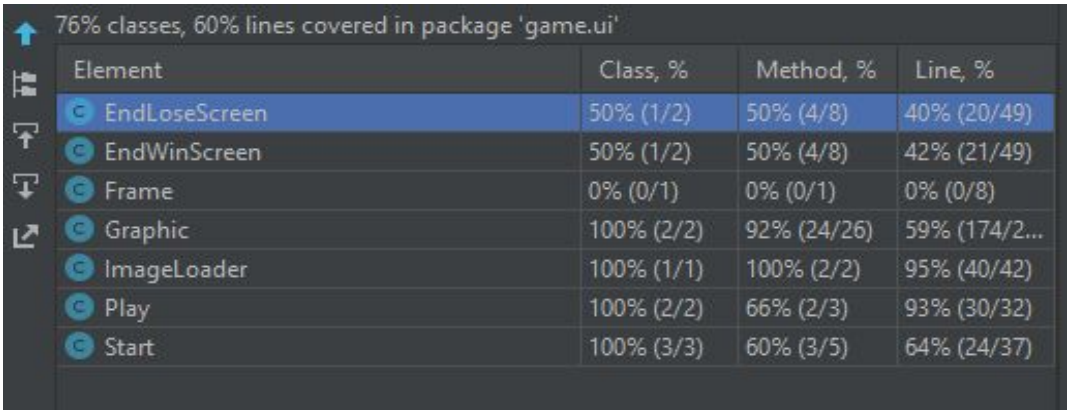
Coverage

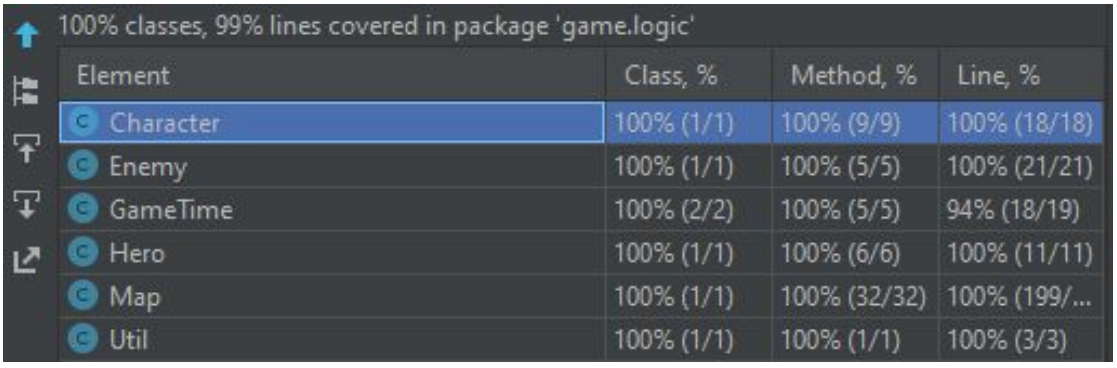
There are 4 main chunks of code that are called in the constructor of the Graphic class that make up the main game logic. The first chunk of code is for hit detection, while the second, third and fourth chunks are for the enemies' line of sight. They were difficult to test as they were simply if-else statements. These statements either call the

showGameOver method (which will be touched on) or increments an enemy's x and y values.

In the showGameOver method, the tests cover the majority of the method with the exception of the call to the method created from EndLoseScreen. This was not getting coverage because of a conscious choice to not have tests for the GUI components of the project. While showGameOver gets some coverage, showWinGame gets none at all, despite having almost the same code in their methods.

As previously mentioned, some of the classes were not tested due to them consisting mainly of UI elements. This was the case for the EndWinScreen class and the EndLoseScreen class. There were tests developed to test the score and time labels but not to test whether the Swing components were displaying properly. These components, along with a single button that quits the game offers no game logic and results with no coverage in the tests.

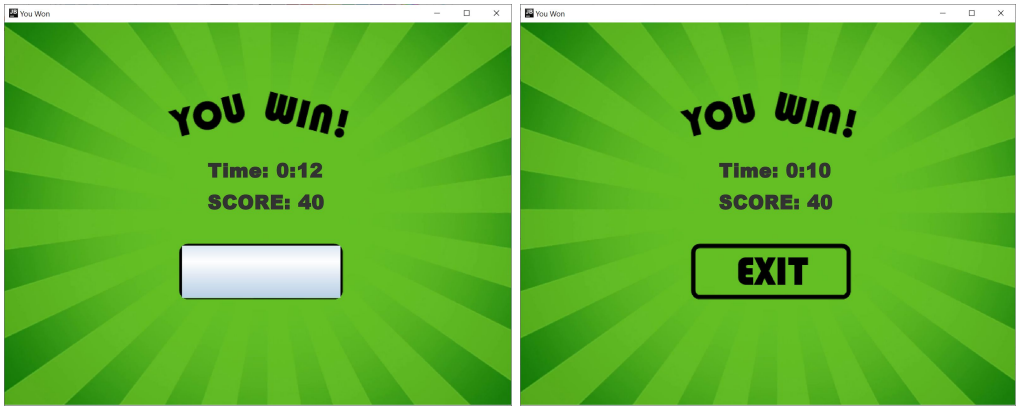




Bugs

There weren't many bugs to be found when actually performing tests. However there were two being noticed while in the testing process. They were about how the

EndLoseScreen and EndWinScreen class fill the exit buttons. On those screens the background of the button appeared differently for the group members. In conclusion, it was because the team was working on different types of computers like a mac or windows. So that required a fix to it and so the decision was to implement the solution of making sure that the fill of the button was set to false so that it was certain that no one would be able to see the background of the button. Below are pictures of the EndWinScreen class with and without the bug.



Another bug found was with the ImageLoader class which was causing errors when running the game and when creating the build with Maven. That issue had us stumped and with the T.A's help, the ImageLoader class was no longer causing issues with our final product.

Final Thoughts

One of the things that was brought to attention when testing was making sure all team members had good coding practices. For example, a few getters and setters needed to be added to create decent tests for items and replaced a few lines of the production code with these new methods, which helped encapsulate the code better. This was also a good learning experience in using JUnit where only one person prior had a little bit of experience but now the whole team has learned a new method in making sure their code in the future will be viable and can trust that they created decent code that can be reassured with JUnit testing and checking different test cases with their methods that they have created.