2.1.0 Unit and Integration Tests

GameObjects

BonusReward.java

Unit Test

- *start():handles the running of BonusReward thread, runs if running is true*
- *stop():handles stopping of BonusReward thread, stops if running is false*
- *getRandomTime():gets a randomtime that gives a value between 5 and 30*
- *Tests starting and stopping threads, player on the reward, and player not on the reward.*

Integration Test

- *Public void run (): handles bonus reward's location update; interact with the bonus reward object, player object, Asset object*

Enemy.java

Unit Test

- *run(): keeps running update() until running becomes false, sleeps for 1 second after performing update()*
- *start():handles the running of enemyThread until running is false*
- *stop():handles stopping of enemyThread when running is false*

Integration Test

- *isOnPlayer(): Checks if enemy is on same location; interacts with player objects*
- *update(): Checks with player's location by using player object and update its location in terms of player's location; interacts with player object*

GameObject.java

Unit Test

- *getX(): gets the x location of the game object*
- *getY(): gets the y location of the game object*
- *isWalkable(): checks if the next tile is walkable*

MainCharacter.java

Unit Test

- *run():keeps running the player thread until running is false, also updates timepassed that keeps track of time of game session*
- *stop():stops the player thread when running is false*
- *getTime():gets the timepassed variable*

- *getScore():gets the player's score*
- *addRewardCount(): adds one to player's reward count*
- *getRewardCount(): gets player's reward count*
- *setScore(): sets player's score*
- *getPlayerThread(): returns the player thread*

Integration Test

- *update():checks for user input by interacting with keyInput on whether they pressed W A S or D, depending on button press move player to that location, also checks if the location they are moving to is walkable or not before changing player location; interacts with keyInput object*
- *getLevel1Win(): returns a boolean depending on player's current condition, uses reward count and checking if player is on exit before returning true but returns false if conditions are not met; interacts with level1State object.*

RegularReward.java

Integration Test

- *update(): checks if player is on the same location as the reward, if it is reward's value becomes zero and adds uses adds player's reward count by one; interacts with MainCharacter object*

RewardCell.java

Integration Test

- *Constructor(): creates the rewardCell object: interacts with MainCharacter Object*

Trap.java

Integration Test

- *update(): updates the player score when player's location is same as the trap's location, after doing so trap is removed from the game and its boolean steppedOn becomes true: interacts with MainCharacter object*

Input

KeyInput.java

Unit Test

- *setKeyPressed(boolean pressed): set keyPressed. Used by keyPressed(KeyEvent e).*
- *keyPressed(KeyEvent e): summoned when a key is pressed; it sets keyPressed to true.*

MouseInput.java

Unit Test

- *mousePressed(MouseEvent e):summoned when mouse is pressed, if left clicked, set leftPressed to true; else if right clicked, set rightPressed to true.*
- *mouseReleased(MouseEvent e): summoned when mouse is pressed, if left clicked, set leftPressed to false; else if right clicked, set rightPressed to false.*
- *mouseMoved(MouseEvent e): sets the mouse coordinate.*

State

GameOverState.java

Integration Test

- *update():sets the game's running to false: interacts with game object*

Game.java

Unit Test

- *setRunning():sets the games running boolean to the one given in parameter*
- *getRunning(): gets the running boolean*

Integration Test

- *setMouseInput():sets the mouseInput to the MouseInput given in parameter: interacts with MouseInput object and Jpanel Object*
- *getMouseInput(): gets the mouseInput object; interacts with Jpanel and KeyInput object*
- *setKeyInput(): sets the keyInput to the KeyInput given in parameter: interacts with KeyInput object and Jpanel object*
- *getKeyInput(): gets the keyInput object, interacts with KeyInput and JPanel object*

Level1State.java

Unit Test

- *startThreads():starts the player, enemy, and bonus reward threads.*

Integration Test

- *Constructor: creates a Level1State object, populates the map with objects; interacts with MainCharacter, BonusReward, Trap, Enemy and Point, and ArrayList objects*
- *update(): checks with player and see if their score dropped to zero or interacted with ghost, if so the state becomes gameOverState, and transitions to winstate if player has collected all rewards and*

reached exit cell; interacts with Enemy, Player, Trap, BonusReward and ArrayList objects

MenuState.java

Integration Test

- *update(): updates the game's state to gamestate if user left clicks with mouse; interacts with MouseInput and Jpanel object*

State.java

Unit Test

- *getstate(): gets the current state*
- *setState(): sets the current state*

Integration Test

- *update(): updates state: interacts with Game object*

WinState.java

Integration Test

- *update(): sets the game's setRunning to false: interacts with Game object*

World

Tile.java

Unit Test

- *Tile(BufferedImage texture, int id): Constructor - it creates a tile. Tested by checking if fields have been assigned correctly.*

World.java

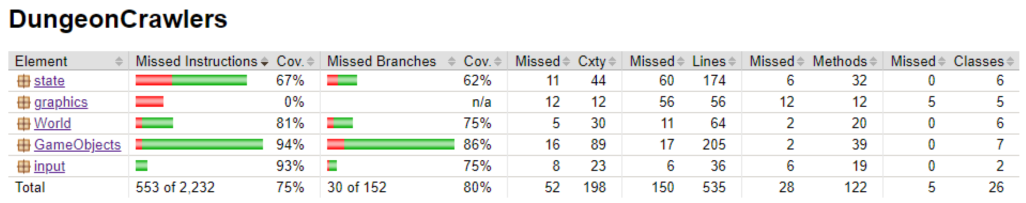
Unit Test

- *getTile(int x, int y): It returns a Tile. Depending on the coordinate, the corresponding tile will be returned, or the default tile would be returned.*
- *loadFile(String path): loads the path then converts it into a readable string.*
- *loadWorld(String path): uses loadFile(String path), then stores the corresponding world information. Acts like a constructor.*
- *stringToInt(String number): converts strings into int. Used by loadWorld.*

2.1.2 Test Quality and Coverage

Coverage

Our group used the Jacoco plugin to report our line and branch coverage. Our results are as follows:



DungeonCrawlers

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
state		67%		62%	11	44	60	174	6	32	0	6
graphics		0%		n/a	12	12	56	56	12	12	5	5
World		81%		75%	5	30	11	64	2	20	0	6
GameObjects		94%		86%	16	89	17	205	2	39	0	7
input		93%		75%	8	23	6	36	6	19	0	2
Total	553 of 2,232	75%	30 of 152	80%	52	198	150	535	28	122	5	26

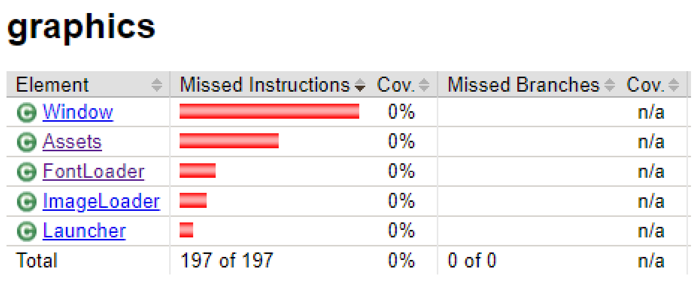
75% Line coverage; 80% Branch coverage

Sources of Uncovered Code

1) Graphics

Due to limited knowledge and course restrictions, our team did not write any tests covering the rendering of UI or the game window. These methods account for a large portion of our missing coverage.

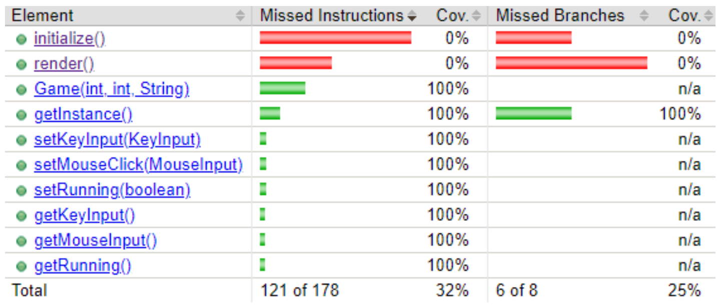
One of our directories is the Graphics directory. Classes here handle loading in assets, fonts, and other operations related to UI. This directory was not covered at all in our tests.



graphics

Element	Missed Instructions	Cov.	Missed Branches	Cov.
Window		0%		n/a
Assets		0%		n/a
FontLoader		0%		n/a
ImageLoader		0%		n/a
Launcher		0%		n/a
Total	197 of 197	0%	0 of 0	n/a

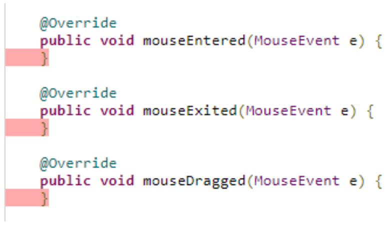
In addition to dedicated graphics, many classes also have their own rendering methods. World renders the graphic of the tiles, each menu renders its own graphic, and each GameObject also renders its own graphic. Not testing these account for a large portion of our missing line/branch coverage. For example, GameTest does not cover the initialize() and render() methods, and these account for 68% of line tests and 75% of branch coverage of this class.



Element	Missed Instructions	Cov.	Missed Branches	Cov.
initialize()		0%		0%
render()		0%		0%
Game(int, int, String)		100%		n/a
getInstance()		100%		100%
setKeyInput(KeyInput)		100%		n/a
setMouseClick(MouseInput)		100%		n/a
setRunning(boolean)		100%		n/a
getKeyInput()		100%		n/a
getMouseInput()		100%		n/a
getRunning()		100%		n/a
Total	121 of 178	32%	6 of 8	25%

2) Empty methods

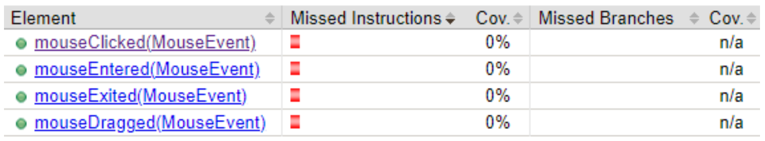
Some methods which implement abstract classes, such as the KeyInput and MouseInput classes, inherit methods which are not used. These methods were not tested and hence, are read as lines not covered. These do not affect branch coverage as they do not have conditional statements.



```
@Override
public void mouseEntered(MouseEvent e) {
}

@Override
public void mouseExited(MouseEvent e) {
}

@Override
public void mouseDragged(MouseEvent e) {
}
```



Element	Missed Instructions	Cov.	Missed Branches	Cov.
mouseClicked(MouseEvent)	🟥	0%		n/a
mouseEntered(MouseEvent)	🟥	0%		n/a
mouseExited(MouseEvent)	🟥	0%		n/a
mouseDragged(MouseEvent)	🟥	0%		n/a

3) Logical Constraints

Our team tested the majority of all logical code, but some do not allow for full coverage. For example, the MainCharacter class has an update function where the player is able to move in a direction when: 1) the correct key is pressed, 2) the next tile is not a barrier, 3) the next tile is out of bounds

We tested all combinations except: 1) the correct key is pressed, 2) the next tile is a barrier, 3) the next tile is out of bounds

This situation is impossible to test as there exist no barriers outside the map bounds. Outside of these constraints, our team found and tested all possible logical tests.

<u>**2.1.3 Findings**</u>

Our code behaved as expected logically as expected while running the tests and no bugs were found as we did many testing during the development process. However, we did find that many classes were missing setters and getters, which were needed for testing for many classes. We added them to not only allow for easier testing, but for better development practices and improving our code quality. These changes allow for more flexibility and ease of use of our code in the future.