Group 12 Phase 3 Report

Tests:

Sound(Done by Sahil):

- Check if sound is played
- Check if the sound manager is created
- Check if a played sound can be stopped
- Check if a specified sound can be played

Menu(Done by Jon):

- Check if the menu has been created
- Check if all menu components have been loaded in
- Check if all menu buttons have been loaded in
- Check if the menu can be updated(various tests)
- Check if you can exit the menus settings
- Check if you can stop the game from the menu
- Check if the menus images has been drawn

Scoreboard(Done by Dylan):

- Check if score is updated
- Check if timer is updated
- Check if a message is shown
- Check if various different messages are shown
- Check if the scoreboard is painted

GameTime(Done by Dylan):

- Check if timer is created
- Check if timer is started
- Check if timer is stopped
- Check if timer is paused
- Check if timer is resumed after pausing
- Check if the displayed time is formatted correctly

Movement(Done by Dylan):

- Check if player can move in all 4 directions (UP, DOWN, LEFT, RIGHT)
- Check if enemy moves
- Check if all movement keys (WASD) are pressed

Enemy(Done by Dylan):

- Check if enemy has damage
- Check if enemy is drawn
- Check if enemy can activate/deactivate ability
- Check if enemy updates every frame

Map(Done by Dylan)

- Check if tile coordinates are correct
- Check if map is loaded correctly
- Check if collision between tiles and player works
- Check if map array can be obtained
- Check if map is drawn
- Check if pizza updates every frame

Player(Done by Max)

- Check if the player's screen position is unchanged
- Check if the player moves when movement keys are pressed
- Check if the player doesn't move when no keys are pressed
- Check if the player becomes invincible when colliding with enemy

GamePanel(Done by Max)

- Check if game thread is started
- Check if game initially loads correctly
- Check if game's timer runs after resuming from pause
- Check if game's timer is paused and scoreboard is hidden after pausing
- Check if game's timer stops and scoreboard is hidden after game over
- Check if game's score updates after item is collected

When making tests, we realized that many tests can be separated into different categories, as evident by our project structure. These categories are entity handling, map handling, core gameplay handling, and handling of miscellaneous utilities. As well, some tests can fit into different sub-categories, such as menu tests in the core gameplay. Knowing this, we were able to write specific tests for the different features of the game. Most tests were made to be as cohesive as possible. For example, both player and enemy got their own test classes, as well as every utility class. If a specific test relied on using multiple classes different from other tests, it would be tested in the class that would be affected the most.

Making tests for our game is not as easy as it initially seemed. It was pretty apparent that multiple classes did not need to be tested for input. As such, most (but not all) tests were conducted on class attributes rather than different inputs. Many classes also used void methods and private or protected attributes which were an even greater challenge to test. These would usually test whether or not some accessible attribute is correct or updated instead. A different and more difficult example was testing the correctness of draw or paint methods for any classes that would visually appear in the game. This was solved by obtaining an image from the class, then after drawing a desired object using the image, check to see if its attributes match with expected values.

Overall our tests achieve 89% line coverage and 79% branch coverage, meaning that most of the game is included within the tests.

Coverage:

Package	Line Coverage	Branch Coverage
Entity	89%	62%
Enemy (in Entity)	63%	30%
Util	94%	82%
Game	86%	85%
Menu (in Game)	97%	94%
Map	93%	86%
Overall	89%	79%

Nearly every category was able to have fairly high coverage, as we wanted to test as many outcomes as possible and with reasonable accuracy. However, enemies were by far the most difficult to achieve thorough coverage with. This is because actions that enemies take are randomized, and dependent on multiple frame updates, meaning that testing all of these interactions would be possible in the limited testing space. So while we could not test every line and branch for each choice an enemy could make, we could test whether it was making a choice in the first place. Other test classes had a similar issue to the enemies test but to a lesser extent, where testing update methods fully was not feasible, such as the ones in GamePanel or ItemManager. Fortunately, these updates are usually fairly quick or easy to spot in game, and any flaws would be easy to identify, and playtesting the game has allowed us to be sure of these functions working properly.

Specific coverage:

Util:

Sound:

- Decided that testing main functionality was most important
- Wanted to cover if the sounds worked as well as obscure things like can all of the sounds be looped
- Cases didn't cover all scenarios where the player picks up an item and then plays the sound, instead we just tested if the specified sound can be played at will.

Scoreboard:

- Main functionality was prioritized but also integration with items
- Wanted to cover if the score and time display worked properly, as well as extra messages in various different scenarios
- Cases didn't cover font exceptions or when no message is displayed

GameTime:

- Decided that testing main functionality was most important
- Wanted to cover if the timer was able to be started and stopped, as well as if the time was formatted correctly
- Cases didn't cover branches where the time tries to update while the timer isn't running and where resumeTimer is called when the timer is running

Movement (CollisionDetector and KeyHandler):

- Tested main functionality alongside integration with player and enemy
- Wanted to cover if entities are able to move, as well as if key inputs work correctly for all movement keys
- Cases didn't cover all possibilities for movement, such as the player being able to move in a specific direction.

☐ ImageLoader and Config were not directly tested but had uses in other tests that covered most of their methods.

Map:

- Testing MapManager alone covers the majority of functions in the map package
- Wanted to cover loading and getting the map, collision with tiles on the map, getting the correct map coordinates, and if objects on the map update
- Cases didn't cover map loading exceptions, all branches where onCollide returns false, and when pizza items changed collectability status

Game:

GamePanel:

- Main functionality was prioritized but also integration with items
- Wanted to cover game thread starting, as well as behaviour in different game states with different items
- Cases didn't cover paint method as well as updates during PLAY game state

Menu package:

- Decided that testing main functionality was most important
- Wanted to cover all menu states and inputs, as well as correct behaviour, appearance, and position of buttons
- Cases didn't cover font exceptions, thread sleep exceptions, and pressing escape when menu state is not SETTINGS (closing the game)

Entity:

Player:

- Main functionality was prioritized but also integration with enemies
- Wanted to cover getting correct screen position, player input via keyboard making player move, and if player becomes invincible on enemy collision
- Cases didn't cover every single input case, forcing the game to pause if needed, and if player isn't invincible on enemy collision

Enemy package:

- Decided that testing main functionality was most important
- Wanted to cover activating and deactivating ability, getting damage, and checking if enemy updates
- Does not cover every possible movement update, as well as ability activation/deactivation on update

Findings:

While we made sure to minimize as many bugs as possible when making our game, we still did find a few problems from testing. One of these bugs was entities being created with incorrect coordinates, and this was fixed fairly quickly by correcting the input values. Another problem involved odd behaviour with item collision, as the item's hitbox would still be present even after being picked up, and would also block entity movement. Additionally, enemies would also be able to pick up items which were not intended. This was fixed by changing the input parameters of the onCollide function to correctly determine if the player is picking up an item. Another item-related problem found was the score not being displayed correctly when picking up an item, which was easily solved by making sure the actual score variable is updated rather than the score parameter in the updateScore method.

As we added more and more tests, another bug was encountered where the testing suite would stop functioning. This was because java was running out of heap space due to audio data being too big. This led to the restructuring of the SoundManager class to use a hashmap rather than an arraylist to prevent huge accesses of data, only loading sounds when needed. This lets us run more tests without any more space issues.

When the sound class was being tested, Sahil ran into an issue where he couldn’t run the code since his Java was outdated. After fixing this issue he was able to create and test the sound test cases. The main issue was that the “clamp” function was being treated as an error since his java was outdated. Initially he created a clamp function of his own to temporarily bypass the issue but eventually reverted that change and then updated his Java.