Phase 3 Report

Unit Tests:

GameController:

- Pause Game: This unit of functionality is responsible for pausing the game when the selects pause. All player movement, enemy movement, time count down, and reward pop-ups stop till the game resumes.
    - Function: pauseGame()
    - Class: GameControllerTest
    - Assertion: assertTrue on controller.isPause()
- Resume Game: This unit of functionality resumes the game after a pause. So all functionality that stopped when the pause functionality was selected resumes.
    - Function: resumeGame()
    - Class: GameControllerTest
    - Input/Action: game pauses
    - Assertion: assertFalse on controller.isPause()
- Time Level: This unit of functionality times the level. If the player does not get all the keys before the time runs out, then the game is over.
    - Function: getTime()
    - Class: GameControllerTest
    - Input/Action: wait for 60 secs
    - Assertion: assertTrue on controller.getTime().equals("0:59")
- Time Runs Out: This unit of functionality ends the game once the 2 minutes timing is over. The game at this point totally stops
    - Function: gameOver()
    - Class: GameControllerTest
    - Action: time runs out
    - Assertion: assertTrue on controller.isEnd()
- User Ends the Game: This unit of functionality ends the game immediately terminating all existing threads.
    - Function: endGame()
    - Action: user ends the game
    - Assertion: assertTrue on controller.isEnd()
- Game Reset: This unit of functionality resets the game to the beginning level restoring the timer back to 2 minutes and player and enemy positions.
    - Function: resetGameByAssertingOnTimer() and resetGameByAssertingOnPlayerPosition()
    - Action: user resets the game
    - Assertion: assertTrue on time and player's position
- Game Start up: This unit of functionality is responsible for starting the game by loading the map and starting the timer
    - Function: startGame()
    - Action: user starts the game
    - Assertion: assertTrue on timer

- Treasure Scheduler: This unit of functionality takes care of scheduling the treasures on the map that appears randomly and disappears
    - Function: scheduleTreasureAssertingWhenNoTreasure() && scheduleTreasureAssertingWhenThereIsTreasure()
    - Action: wait for some time to test interval for no treasure and with treasure
    - Assertion: assertTrue when a treasure pops up or doesn't in the right interval
- MedKit Scheduler: This unit of functionality handles scheduling the appearances and disappearances of medkit in the map
    - Function: scheduleMedKitAssertingNoMedKit() and scheduleMedKitAssertingWhenThereIsMedKit()
    - Action: wait for some time to test if interval has a medkit
    - Assertion: assertTrue when a medkit pops up or doesn't in the right interval

Player Test:

- Test if player is alive: This unit tests if the player is alive when set alive. Moreover, player status is checked to see if alive after losing only 1 health (2 remaining).
- Test if player is dead: This unit of functionality tests to see if player's status is dead after losing 3 health.
- Test adding and removing health: This unit of functionality is responsible for adding and removing health points.
- Test adding key: This unit of functionality is responsible for adding key collected to key count.
- Test player collision with entities: These tests check to see if colliding with either enemy, health, lavapit, key, bonus reward, and bonus health produces expected outcome.
    - Collision with enemy: expected outcome is to reduce health by 1
    - Collision with key: expected outcome is to increase key count by 1
    - Collision with lavapit: expected outcome to kill player (reduce health by 3)
    - Collision with bonus reward: expected outcome is to increase bonus score by 1
    - Collision with bonus health: expected outcome is to increase health by 1

Enemy test:

- Enemy attribution: Enemy has its own attribution like speed, direction. This unit of functionality is reponsible for construct a enemy unit. These attribution will be use in other function, The test is checking if we give a right value to enemy.
- Icon load: The enemy should have its own Icon in the frame. This test is checking if the icon successfully load into the game.
- AI path finder: This unit of functionality is responsible for moving eneny and chasing player. Enemy will keep tracking player in a certain area untill player leave. The test will check whether enemy goes on the right way.
- Enemt direction: Check if enemy shows right picture when it change the direction. Enemy has 4 type of Icon which is go right, left, up ,and down.
- Making a Intergration test for enemy and trap with player and frame. Check if they can work successfully with each other.

Integration Test:

- File System Integration With Game: This test how well the rest of my system integrates with reading game set up from file using class LevelTest.
  - readFromFileWall()
    - Test case: read from file that contains index 3
    - Assertion: assert on whether the cell at index 3 does contain a wall
  - readFromFileLavaPit()
    - Test case: read from file that contains index 95
    - Assertion: assert on whether the cell at index 95 does contain a lavapit
  - readFromFileReward()
    - Test case: read from file that contains index 56
    - Assertion: assert on whether the cell at index 56 is empty so a reward can appear on it
  - readFromFileEntity()
    - Test case: read from file that contains index 146
    - Assertion: assert on whether the cell at index 146 does contain an enemy at the start of the game

Findings:

- Fixed enemy movement bug: I found an ArrayIndexOutOfBoundException when testing the enemy move functionality. I fixed it my implementing strict boundary checks
- Fixed Rewards pop ups: implementing Thread.sleep() in timer.scheduleAtFixedRate() leads to random behaviout so I had to fix this bug by removing Thread.sleep()
- Improved the quality of code in Level1, Level2, and Level3 classes. I abstracted common functionaliies of these levels to maintain cleaner code
- Improved the quality of my code my placing try and catch statements in functions that are responsible for handling these errors
- Fixed file system bug by using correct file path for set up files
- Fix player collision class to collide with different entities on the square.
- Move PlayerCollideWithEnemy into Player class
- Change the Playercollide structure, adding the Enemy collide function.
- Initial enemy speed to 1;
- Enemy cannot move correctly as it place incorrectly set by -1.
- Add a new function isactive() to enemy file to determine if the enemy is alive or not.
- In the playerCollison part, we added the function enemy.getSquare().setEntity(), we set this parameter to null, so that if the enemy touches the player, it will disappear.
- In phase2, the key in the game will flash everywhere, to solve this bug, we added a function:
- square.isWalkable()&&square.getentity()==null){}, through this function we can make these keys appear randomly in the place where they should appear (not on the wall, traps)

- In the enemy move function, our image was down. This time I changed the image to up and down so that the player would look very comfortable.
- The player will have five choices after each move. I calculate the distance between the enemy and the player by calcDist(), and the enemy will always choose to move in the direction of the player closest to ensure the game's difficulty.
- Before the enemy leaves a shadow when moving, resulting in many enemies, I solved this problem by setting setEntity to null in the move function.
- Previously, enemies did not drop blood when colliding with player, I solved this by adding a constraint
- if(player.getYIndex()==square.getY()&&player.getXIndex()==square.getX()){GameController.collision.collide(player,this) to achieve player and enemy collide with enemies.
- Add a new function mouseclicked() to achieve some basic button operations. When the mouse clicks on that area, it will pop up a different interface, such as click restart, which will restart the game, and the exit will exit the software.

Coverage:

PlayerTest:
- Line Coverage:
  - Lines = 27
  - Covered = 39
  - Coverage = (27/39)*100 = 69%
- PlayerTest Branch Coverage:
  - Branches = 2
  - Covered = 2
  - Coverage = (2/2)*100 = 100%

Player Collision:
- Line Coverage:
  - Lines = 31
  - Covered = 30
  - Coverage = (30/31)*100 = 96%
- PlayerTest Branch Coverage:
  - Branches = 16
  - Covered = 4
  - Coverage = (4/16)*100 = 25%

Demon:
- Line Coverage:
  - Lines = 84
  - Covered = 67
  - Coverage = (67/84)*100 = 79%
- PlayerTest Branch Coverage:
  - Branches = 0
  - Covered = 0
  - Coverage = (0/0)*100 = 100%

GameController:

- Line Coverage:
    - Lines = 169
    - Covered = 68
    - Coverage = (68/169)*100 = 40%
- PlayerTest Branch Coverage:
    - Branches = 0
    - Covered = 0
    - Coverage = (0/0)*100 = 100%

Level Test:

- Line Coverage:
    - Lines = 719
    - Covered = 290
    - Coverage = (290/719)*100 = 40%
- PlayerTest Branch Coverage:
    - Branches = 35
    - Covered = 19
    - Coverage = (19/35)*100 = 54%