Blob 2D Game
Report 3

Unit Tests:

Features Required:

- Spawn-able Entities: -> void testSpawn() -> GameBoardTest.java
    - Created unit tests to ensure each entity being spawned is placed at the correct point position associated with them using .getPos().
    - Tests to ensure the constructors are being created accordingly
- Bonus Visibility: -> void testBonusVisibility() -> GameBoardTest.java
    - Created unit test to check if the bonus visibility is appearing and disappearing at correct random times. This test was conducted applying true and false assertions to determine whether the visibility Boolean was activated and deactivated accordingly to the event.
- getPos() : -> void testSpawn() -> GameBoardTest.java
    - Created unit test to check if instantiated object has correctly assigned point position
- setPos(): -> testCheckWalls() -> GameBoardTest.java
    - Created unit test to assign instantiated object with point position
- getScore(): -> testCollectRewards() -> GameBoardTest.java
    - Created unit test to check if method is returning correct player's current Score
- addScore(): -> testCollectRewards() -> GameBoardTest.java
    - Created unit test to check if player score is correctly adding assigned Score Value
- deductScore(): -> testHitPunishment() -> GameBoardTest.java
    - Created unit test to check if player score is being deducted correctly by assigned Score Value
- FrozenTimer()-> testFrozenTimer() -> PlayerTest.java
    - Created unit test to assert if freeze duration is being held to the accurate length

Integration Tests:

Interactions Required:

GameBoardTest.java - Integration Tests

- Void testPlayerWin():
  - Interaction between player, exit tile and gate to determine if player has met requirements to win the game.
- Void testPlayerLose():
  - Interaction between player and system to determine the lost state using player's score.
- Void testEnemyKillPlayer():
  - Interaction between moving enemies and player, asserting to see if player and enemy point positions are equal and asserting the Boolean isDead is the condition is met to ensure that the player is being killed if they have the same positions.
- Void testHitPunishment():
  - Interaction between player and non movable enemies, asserting to see if player score is being deduced correctly by enemies associated Penalty value, as well as asserting to see if playerHit boolean returns true indicating the player has been hit.
- Void testHitTelePunishment():
  - Interaction between player and different type of non movable enemy, asserting to see if player position is being reset to the correct reset point position. In addition, asserting to check if element of ArrayList of non moveable enemies is an instanceof TelePunishment, so that player will only teleport if in contact with this object class type.
- Void testEnemyDirection():
  - Interaction between Enemy position and Player position, assertions are used to indicate Player estimate player position based on Enemy position as origin. This allows the enemy to scan for the player and tests to ensure that the player position is accurately scanned.
- Void testEnemyCheckWalls():
  - Interaction between Enemy and wall objects, asserting to see if wall Boolean variables are being activated when enemy is next to a wall according to the directional Boolean aWallNorth, aWallSouth, aWallEast and aWallWest. Also asserting to test if enemy is position is being spawned correctly.
- Void testCheckWalls():
  - Interaction between Player and wall objects, asserting to see if wall Booleans variables: aWallNorth, aWallSouth, aWallEast and aWallWest are being activated

according to the direction indicators. Also asserting to see if player spawned at correct point position.

- Void testEnemyCheckEnemy():
  - Interaction between Enemies and other Enemies to avoid overlapping of enemies on one tile. Applies similar functionality concept to EnemyCheckWalls, except replacing wall object with Enemy object.
- Void testCollectRewards():
  - Interaction between Player and cakes, asserting to see if player point position and cake point positions are equal, then asserting to see if player score is correctly adding the assigned score value.

PlayerTest.java – Integration Tests

- Void testKeyPressed ():
  - Interaction between the User (System) with the keypress simulated, and the Player. Asserting to see whether the player moved or not when the key was pressed (assuming game was not paused, and user is pressing W specifically).

EnemyTest.java – Integration Tests

- Void testEnemyMovement():
  - Interaction between the Enemies and the Player, asserting to see whether an enemy will move in the direction of the player when called (Test method starts 1 tile away and tests enemy movement (north) onto player tile, but this can be generalized to any position for enemies and player).

Features Not Included in Tests:

- LoadImages() feature was not tested, as it is implemented with a try – catch error exception to notify us in cases where the images were not loaded in correctly. Furthermore, the functionality depends on ImageIO to read the file from the resources folder, as a result, we do not need to test the ImageIO feature.
- gameBoundary() feature was not tested, as it is a simple feature we are able to visually test, as the boundary is the border of the window frame.
- JButton, JFrame, JWindow and Paint Components were not tested as well, as they are all visual aspects to the game and can be tested visually.
- gameInit() feature was not tested, as it is redundant, if spawn test fails, this method would fail as well.
- stateSwitch(int gameState) was not tested, as its utility was based for visual representation, we would know if the stateSwitch fails if the window does not change gameState correctly.
- gameReset() simply clears the game entities, nothing to test except to see if they got cleared; however can be seen visually during the next game reset.

- timeElapsed() not tested because its based on real time, so we require the program to run and count the game running time using System.currentTimeMillis().

Branch Coverage:

The following branch possibilities were tested:

- 19 Enemy movement possibilities
- 8 Enemy checkWall() possibilities
- 8 Enemy checkEnemy() possibilities
- 1 Enemy kill Player possibility
- 2 Possibilities for teleporting tile
- 8 player checkWall() possibilities
- 2 Bonus Visibility conditions
- 1 Condition for reward collection
- 1 Player hit punishment condition
- 1 Player Loose condition
- 2 Player win conditions
- 1 Frozen check condition

The following branches were not tested:

- 3 Different buttons
- 4 Possible game States
- 1 Condition for time elapsed
- 6 Paint component possibilities
- 2 Pause and Resume game possibilities
- 1 Action performed condition
- 1 Map boundary condition
- 4 Player movement possibilities

Total Branches covered: 54
Total Branches not covered: 23
Total Branches: 54 + 23 = 77
Branch Coverage Ratio: 70%

Line Coverage:

Total Lines tested: 620 (approx.)
Total Lines not tested: 255 (approx.)
Total Lines: 875 (approx.)
Line Coverage Ratio: 70%

Findings:

- While testing the testPlayerWin() under GameBoardTest.java, the assertions allowed us to find a small bug, where the bonus reward was acting as a regular reward. This resulted in the requirement gate to unlocking after collecting 9 pieces of regular and 1 bonus reward, while the requirements were to be 10 regular pieces to unlock the gate.
    - Changed the production code, to fix the minor technical bug.
- Method reusability became an issue, as a result, we parameterized some functions to allow us to reuse them for test purposes.
- Found some methods to be redundant, such as the enemyCheckWalls() and checkWalls(), as they are both subclasses of entities, and can be simplified into one method and inherited/overridden by the children classes; however, we decided to not change this until phase 4, where we must refactor as we are focused on prioritizing testing this phase.
- Spawn Methods for several of one entity type is being inefficiently instantiated, needs to be refactored.