CMPT 276 Group Project: Phase 3

Thomas Kingsman, 301424023
Owen Liu, 301369997
Gurnoor Chahal, 30142771
Yoobin Nam, 301538106

Barrier:

- Made Barrier constructor public
- Added function to return the BarrierType (Wall, etc.)

Line coverage:
4/4 in constructor
7/7 in setSprite
=100%

Branch coverage:
3/4 in setSprite
=75%

Things skipped:

- setSprite: skipped case where option is not one of the listed; we used an enum so there are currently no other options anyways

Controller:

- Added getSpeed() method
- Added get methods for all direction booleans

Line coverage:
14/14 lines tested in keyPressed
14/14 lines tested in keyReleased
26/26 lines testing in keyProcess
=100%

Branch coverage:
5/5 branches tested in keyPressed
5/5 branches tested in keyReleased
16/16 branches tested in keyProcess
=100%

Things skipped:

- Sets and gets: as outlined by TA
- Constructor: as outlined by TA

Entity:

- Made constructors public

Line coverage:
2/2 in default constructor
0/2 in entity sanity constructor
0/2 in entity position constructor
0/2 in entity position and sanity constructor

1/1 in getSanityImpact
1/1 in setSanityImpact
= 40%

Branch coverage:
100% coverage since no conditional executions in Entity class.

Things skipped:
Skipped other constructors to reduce redundancy testing; most of it is inherited from the gameObject class, which tests it more extensively.

ExitTile:
- Made constructor public

Line coverage:
14/14 in checkForFinish
=100%

Branch coverage:
4/4 in checkForFinish
=100%

Things skipped:
- Constructor: as outlined by TA

gameFrame:
- Made constructor public

Line coverage:
N/A

Branch coverage:
N/A

Things skipped:
- All of it; it only serves to create a graphic, which cannot be easily computer tested

GameObject:
- Overloaded getDimensions() method to allow adding an ImageObserver
    - Helps prevent issue where dimensions return as array with contents {-1, -1}
- Made constructor public
- Created a method to return the sprite image for testing
- Moved BarrierType to its own class file in order to change it to public for testing

Line coverage:
2/2 in Constructor
0/2 in setDimensions
0/1 in getDimensions
1/1 in getLocation
1/1 in setX
1/1 in getX

1/1 in setY
1/1 in getY
1/1 in getGoType
1/1 in setGoType
0/1 in getCurrBoundaries
=69.23%

<u>Branch coverage:</u>
0/2 in setDimensions
=0%

<u>Things skipped:</u>

- Sets and gets: as outlined by TA

gamePanel:

- Made most functions to create stage objects static
    - Allows creation of stage objects within internal lists, without actually creating a visible panel
- Added parameter to force an ImageObserver in checkNoBarrier
    - JUnit tests need an observer to check when an image has loaded to prevent dimensions of {-1, -1}; however, in deployment, this is not a concern. In fact, deployment explicitly cannot use an observer as it adds several milliseconds of delay per frame update, which isn't tenable
- Added several get functions
- Added function to reset player, enemies, and pickups
- Added function to preload images when getting dimensions

<u>Line coverage:</u>
42/77 in game creation functions
18/18 in checkNoBarrier
84/88 in checkCollision
14/14 in noOverlappingEntities
0/23 in createObjects
0/4 in drawSpecialPickups
0/15 in playBackgroundMusic
0/5 in paintComponent
0/44 in actionPerformed
12/12 in prepImgForViewing
11/11 in resetGameAttributes
=58.2%

<u>Branch coverage:</u>
0/0 game creation functions
5/6 in checkNoBarrier
17/26 in checkCollision
2/4 in noOverlappingEntities
0/2 in createObjects
0/2 in drawSpecialPickups
0/2 in playBackgroundMusic
0/2 in paintComponent
0/12 in actionPerformed

0/4 in prepImgForViewing
0/0 in get functions
0/0 in resetGameAttributes
=40%

Things skipped:

- Non-static functions: these require instantiation of a gamePanel object, which creates a graphic which cannot be easily computer tested
- Functions to create objects: these are trivially working as they repeatedly create new objects; the correctness of the creation of these objects is left to their respective class's tests
- Branches to not force observers: these branches are for deployment only; in JUnit tests, they execute quickly enough that they *must* force an observer to function correctly
- Branches for case where the image fails to load: cannot guarantee that the image will fail to load, so impossible to consistently test

Main:

- N/A

Line coverage:
N/A

Branch coverage:
N/A

Things skipped:

- All of it; it only serves to create a graphic, which cannot be easily computer tested

MobileEnemy:

- N/A

Line coverage:
1/1 in getMoveSpeed
33/33 in moveToPlayer
= 100%

Branch coverage:
16/16 in moveToPlayer
=100%

Things skipped:

- Constructor: as outlined by TA

Pickup:

- Made constructors public

Line coverage:
2/2 in createCoffeeSprite
=100%

Branch coverage:

0/0 in createCoffeeSprite
=indeterminate

<u>Things skipped:</u>

- Constructor: as outlined by TA

Player:

- Made constructor public

<u>Line coverage:</u>
0/3 in constructor
1/1 in getSanity
1/1 in getPickupsCollected
1/1 in getSpecialPickupsCollected
1/1 in updateSanity
1/1 in incrementPickup
1/1 in incrementSpecialPickup
=66.67%

<u>Branch coverage:</u>
100% Coverage since no conditional executions in Player Class.

<u>Things skipped:</u>

- Constructor: as outlined by TA instruction

SpecialPickup:

- Made constructors public

<u>Line coverage:</u>
0/4 in constructors
0/2 in createSpecialSprite
=0%

<u>Branch coverage:</u>
0/0 in constructors
0/0 in createSpecialSprite
=indeterminate

<u>Things skipped:</u>

- All of it; the logic is highly similar to the Pickup class which performs 100% testing

startFrame:

- N/A

<u>Line coverage:</u>
N/A

<u>Branch coverage:</u>
N/A

<u>Things skipped:</u>

- All of it; it only serves to create a graphic, which cannot be easily computer tested

StaticEnemy:

- Moved several constructor parameters into the super() call

<u>Line coverage:</u>
3/3 in constructor
=0%

<u>Branch coverage:</u>
0/0 in constructor
=indeterminate

<u>Things skipped:</u>

- N/A

Interactions:
All interactions are self documenting within the code.

Findings:
In writing tests, we learned about several important aspects of our code.
Most notably, we had to implement ImageObservers for sprite size detection. Previously, we referenced the same sprites often enough that they weren't needed to ensure correct behaviour of dimension get methods. However, in unit testing, they began to be necessary as we would create new objects frequently and would reference them very soon after creation.
Further testing of functionality required us to make more get methods as, although we achieved our goal in encapsulation, we often found we needed to implement more as we had no way of accessing the variables otherwise. Often we would ourselves be unable to unit test certain classes as their functionality relied on other classes. Their constructors or implementations all relied on parent classes as well, so that allowed for less test cases; however, each covers a wide range of aspects in our game.
Finally, we had to reformat several methods of gamePanel to be static, as creating a new gamePanel would create the graphic window, which we wanted to avoid. This required a bit of adjustment to the general code for that class.
Overall, this refactoring and testing phase helped us ensure the quality of our code, and also served as a window of insight into industry testing standards.