Testing Write-Ups

placeTowerDuringWaveTest():

- This test tests whether a tower can be placed while the wave is occurring as being able to adjust attack power to changing circumstances or difficulty would be useful.
- This will test the interaction within the GameController class between the startGame() functionality and the buyWizard/placingWizard functionality as starting the game initiates the waves of monsters and buying the Wizard is the only function that enables us to place it.
- If the startGame() method is called and monsters are spawned and travelling across the screen, we are in mid-wave mode of the game. We would try to test if we can place a tower during this phase.

towerAttacksTest()

- This test tests whether a tower will attack during the duration of the wave.
- This will test the interaction between the GameController, Wizard class, and the Enemy class as the wizards are the towers that attack the enemy, and the GameController enables this interaction (specifically during the update method).
- During the course of the wave from monsters being spawned to their end when they attack the monolith, there should be an indication of the tower influencing the health of the enemy. This would be indicated by the health of the enemy decreasing which should only be influenced by the tower and implemented by the enemy or GameController.

towerProximityAttackTest():

- This test tests whether a tower will attack when the enemy is in proximity of the tower itself and not when it is instantly spawned in.
- This will test the interaction between the GameController, Wizard class, and the Enemy class as the wizards are the towers that attack the enemy within proximity, and the GameController enables this interaction (specifically during the update method).
- To differ from the towerAttacksTest, the test will simulate the interaction between 1 tower and the enemies, and the test will check the health of 2 towers: 1 within the “proximity” of the tower and one outside of it. If the healths of the two towers differ with the one inside the proximity being lower while the one outside should be full health, then the functionality of the tower proximity should be verified (specifically during the update method).

enemyKilledTest()

- This test tests whether an enemy will be killed and removed from the path.
- This will test the interaction between the GameController, Wizard class, and the Enemy class as the wizards are the towers that are capable of killing the enemy, and the GameController enables this interaction.
- During the update method of the GameController, the towers will continuously attack the enemy, and when the health of the Enemy hits 0 (implemented in Enemy class while called in GameController), the enemy should die indicated by its sprite removal from the game screen/path, and this should be indicated by the decreased number of enemies in the wave described in the internal count of monsters (totalMonsters/spawnedMonsters)

within the GameController class. This would be additionally supported by the UI change of monsters.

distinctTowerBehaviorTest():

- This test will test if the towers have differing behaviors. Specifically, the Water Wizard should have the faster speed. The Fire Wizard should have the higher damage. The Nature Wizard should have the larger range.
- This would test the interaction between the Wizard class and its subclasses, the Enemy class, and the Game Controller class. The differing behaviors of the towers should be indicated by the changes in health of the enemies given the location. And the GameController enables that interaction.
- The test would have multiple set ups for each of the towers. For the speed test, three game scenarios would be set up where a Water, Fire, and Earth Wizard are each individually set to face a wave of monsters at the same exact location. An in-test timer would count the time elapsed between the tower's first and second attack, so the test with the specific tower with the shortest time between attacks should correspond with the towers having the faster attack speed. The test should also count the differing health decreases of the monster attacked on the first attack. The test with the specific tower with the largest health decrease should correspond with the tower with the higher attack. The test should also time the time elapsed between the start of the wave and the first attack on the first monster. The test with the specific tower with the shortest time elapsed should correspond with the tower with the larger range as the tower with the larger range in the same location should attack first.

distinctEnemyBehaviorTest():

- This test will test if the enemies of different types will have different behaviors. The enemies should be: 1 enemy with a larger amount of health, 1 enemy with faster speed, 1 enemy with stronger attack
- This will test the interaction between the Enemy class and its subclasses and the towers and the Monolith class. The interaction between enemy and towers will test the attack of the monsters while the interaction between the enemy and tower will test the health and speed.
- The test would have 3 setups each testing 3 times (1 for each distinct enemy). Each setup will have specific conditions which will be tested on 3 waves of monsters: 1 for each type. 1 test will have a tower involved. That test will time the count the number of hits it takes to kill the enemy for each type. The test with the most hits taken should correspond with the monster type with the largest health. This test will involve the health decrease and "killing" of monsters. The next setup can combine the other 2 circumstances. For each wave of specific monster, a timer will be set to time how long the monster takes to get to the monolith, and once the monster gets to the monolith, the amount of health decreased from the first hit will be recorded. The test with the shorter time taken to travel should correspond with the monster type with the fastest speed, and the test with the largest health decrease of the tower should correspond with the monster type with the strongest attack

moneyGainedTest(): (general case):

- This test will test if the amount of money properly increases.

- This will test the interaction between the Game Model and the Game Controller as the game model controls the data the amount of money the player has and the game controller controls the sequence of actions leading to increasing the money.
- Through whatever means the player earns money, the test should test if a specific method to increase money is called for the Game Model class, and as long as that method is only used by the Game Controller and implemented by the Game Model, then we can be sure of the functionality of the method.

moneyGainedWhenEnemyDefeatedTest():

- This test expands upon moneyGainedTest() as it tests whether the only case where a player earns money is when a monster is defeated and not randomly during the wave.
- This will test the interaction between the Game Model, Game Controller, and Enemy class. The Game Model should be the only object able to directly call the method to increase money. The Game Controller controls what happens after the specific sequence of events of when an Enemy is defeated by a Tower. The enemy class holds information of the “worth” of the monster which is to be added to total money.
- The Game Controller dictates the logic of updating the health and number of enemies (update()) when attacked by towers, so when the number of enemies decrease (killing) at the hands of a tower (controlled by a boolean), the defeated enemy should return its worth to the Game Controller which will pass it to the Game Model to increase the money.