M6Test.java

testPlaceAndUpgradeSingleTowerSuccessBad

- This test ensures that when a single “bad” tower is placed at a specified location on the map and is then upgraded by purchasing an upgrade and then applying it on the same tower, the corresponding positions on the path the enemies take are updated correctly to deal the right amount of damage to enemies that pass over it. This is done by calling the testPlaceTower() method in the LandscapeController class, then the purchaseUpgrade() method in the LandscapeController class, and then the testUpgradeTower() method in the LandscapeController class, and then verifying that the HashMap object returned by the getTestingPathDamage() method contains the correct keys and values for the positions adjacent to where the tower is placed

testPlaceAndUpgradeSingleTowerSuccessNormal

- This test ensures that when a single “normal” tower is placed at a specified location on the map and is then upgraded by purchasing an upgrade and then applying it on the same tower, the corresponding positions on the path the enemies take are updated correctly to deal the right amount of damage to enemies that pass over it. This is done by calling the testPlaceTower() method in the LandscapeController class, then the purchaseUpgrade() method in the LandscapeController class, and then the testUpgradeTower() method in the LandscapeController class, and then verifying that the HashMap object returned by the getTestingPathDamage() method contains the correct keys and values for the positions adjacent to where the tower is placed

testPlaceAndUpgradeSingleTowerSuccessElite

- This test ensures that when a single “elite” tower is placed at a specified location on the map and is then upgraded by purchasing an upgrade and then applying it on the same tower, the corresponding positions on the path the enemies take are updated correctly to deal the right amount of damage to enemies that pass over it. This is done by calling the testPlaceTower() method in the LandscapeController class, then the purchaseUpgrade() method in the LandscapeController class, and then the testUpgradeTower() method in the LandscapeController class, and then verifying that the HashMap object returned by the getTestingPathDamage() method contains the correct keys and values for the positions adjacent to where the tower is placed

testPlaceAndUpgradeSingleTowerWithoutPurchasingUpgradeBad

- This test ensures that when a single “bad” tower is placed at a specified location on the map, and is then attempted to be upgraded without purchasing an upgrade, the corresponding positions on the path the enemies take are updated correctly to deal the right amount of damage to enemies that pass over it without any upgrades being applied. This is done by calling the testPlaceTower() method in the LandscapeController class, then the testUpgradeTower() method in the LandscapeController class and then verifying that the HashMap object returned by

the getTestingPathDamage() method contains the correct keys and values for the positions adjacent to where the tower is placed

testPlaceAndUpgradeSingleTowerWithoutPurchasingUpgradeNormal

- This test ensures that when a single “normal” tower is placed at a specified location on the map, and is then attempted to be upgraded without purchasing an upgrade, the corresponding positions on the path the enemies take are updated correctly to deal the right amount of damage to enemies that pass over it without any upgrades being applied. This is done by calling the testPlaceTower() method in the LandscapeController class, then the testUpgradeTower() method in the LandscapeController class and then verifying that the HashMap object returned by the getTestingPathDamage() method contains the correct keys and values for the positions adjacent to where the tower is placed

testPlaceAndUpgradeSingleTowerWithoutPurchasingUpgradeElite

- This test ensures that when a single “elite” tower is placed at a specified location on the map, and is then attempted to be upgraded without purchasing an upgrade, the corresponding positions on the path the enemies take are updated correctly to deal the right amount of damage to enemies that pass over it without any upgrades being applied. This is done by calling the testPlaceTower() method in the LandscapeController class, then the testUpgradeTower() method in the LandscapeController class and then verifying that the HashMap object returned by the getTestingPathDamage() method contains the correct keys and values for the positions adjacent to where the tower is placed

testPlaceAndUpgradeMultipleTowersSuccessBad

- This test ensures that when multiple “bad” towers are placed at distinct locations on the map and are then upgraded by purchasing upgrades and then applying them on each tower in sequence (making sure to purchase a new upgrade each time), the corresponding positions on the path the enemies take are updated correctly to deal the right amount of damage to enemies that pass over it. This is done by repeatedly calling the following methods in the LandscapeController class: testPlaceTower(), purchaseUpgrade(), and testUpgradeTower(). We then verify that the HashMap object returned by the getTestingPathDamage() method contains the correct keys and values for the positions adjacent to where the towers are placed

testPlaceAndUpgradeMultipleTowersSuccessNormal

- This test ensures that when multiple “normal” towers are placed at distinct locations on the map and are then upgraded by purchasing upgrades and then applying them on each tower in sequence (making sure to purchase a new upgrade each time), the corresponding positions on the path the enemies take are updated correctly to deal the right amount of damage to enemies that pass over it. This is done by repeatedly calling the following methods in the LandscapeController class: testPlaceTower(), purchaseUpgrade(), and testUpgradeTower(). We then verify that the HashMap

object returned by the getTestingPathDamage() method contains the correct keys and values for the positions adjacent to where the towers are placed

testPlaceAndUpgradeMultipleTowersSuccessElite

- This test ensures that when multiple “elite” towers are placed at distinct locations on the map and are then upgraded by purchasing upgrades and then applying them on each tower in sequence (making sure to purchase a new upgrade each time), the corresponding positions on the path the enemies take are updated correctly to deal the right amount of damage to enemies that pass over it. This is done by repeatedly calling the following methods in the LandscapeController class: testPlaceTower(), purchaseUpgrade(), and testUpgradeTower(). We then verify that the HashMap object returned by the getTestingPathDamage() method contains the correct keys and values for the positions adjacent to where the towers are placed

testPlaceAndUpgradeMultipleTowersSuccessMixed

- This test ensures that when multiple towers of different level (“bad”, “normal”, and “elite”) are placed at distinct locations on the map and are then upgraded by purchasing upgrades and then applying them on each tower in sequence (making sure to purchase a new upgrade each time), the corresponding positions on the path the enemies take are updated correctly to deal the right amount of damage to enemies that pass over it. This is done by repeatedly calling the following methods in the LandscapeController class: testPlaceTower(), purchaseUpgrade(), and testUpgradeTower(). We then verify that the HashMap object returned by the getTestingPathDamage() method contains the correct keys and values for the positions adjacent to where the towers are placed

testPlaceAndUpgradeMultipleTowersOnlyOneUpgradePurchasedBad

- This test ensures that when multiple “bad” towers are placed at distinct locations on the map and are then attempted to be upgraded by purchasing just one upgrade and trying to apply it to each tower, the corresponding positions on the path the enemies take are updated correctly to deal the right amount of damage to enemies that pass over it such that only the first tower on which testUpgradeTower() is called is upgraded, while the others are not upgraded. This is done by calling testPlaceTower() for each tower, then calling purchaseUpgrade() only once, and then testUpgradeTower() for each tower. We then verify that the HashMap object returned by the getTestingPathDamage() method contains the correct keys and values for the positions adjacent to where the towers are placed

testPlaceAndUpgradeMultipleTowersOnlyOneUpgradePurchasedNormal

- This test ensures that when multiple “normal” towers are placed at distinct locations on the map and are then attempted to be upgraded by purchasing just one upgrade and trying to apply it to each tower, the corresponding positions on the path the enemies take are updated correctly to deal the right amount of damage to enemies that pass over it such that only the first tower on which testUpgradeTower() is called

is upgraded, while the others are not upgraded. This is done by calling testPlaceTower() for each tower, then calling purchaseUpgrade() only once, and then testUpgradeTower() for each tower. We then verify that the HashMap object returned by the getTestingPathDamage() method contains the correct keys and values for the positions adjacent to where the towers are placed

testPlaceAndUpgradeMultipleTowersOnlyOneUpgradePurchasedElite

- This test ensures that when multiple “elite” towers are placed at distinct locations on the map and are then attempted to be upgraded by purchasing just one upgrade and trying to apply it to each tower, the corresponding positions on the path the enemies take are updated correctly to deal the right amount of damage to enemies that pass over it such that only the first tower on which testUpgradeTower() is called is upgraded, while the others are not upgraded. This is done by calling testPlaceTower() for each tower, then calling purchaseUpgrade() only once, and then testUpgradeTower() for each tower. We then verify that the HashMap object returned by the getTestingPathDamage() method contains the correct keys and values for the positions adjacent to where the towers are placed

testPlaceAndUpgradeMultipleTowersOnlyOneUpgradePurchasedMixed

- This test ensures that when multiple towers of different level (“bad”, “normal”, and “elite”) are placed at distinct locations on the map and are then attempted to be upgraded by purchasing just one upgrade and trying to apply it to each tower, the corresponding positions on the path the enemies take are updated correctly to deal the right amount of damage to enemies that pass over it such that only the first tower on which testUpgradeTower() is called is upgraded, while the others are not upgraded. This is done by calling testPlaceTower() for each tower, then calling purchaseUpgrade() only once, and then testUpgradeTower() for each tower. We then verify that the HashMap object returned by the getTestingPathDamage() method contains the correct keys and values for the positions adjacent to where the towers are placed

testPlaceSingleTowerOutOfRangeAndUpgradeTowerBad

- This test ensures that when a single “bad” tower is placed at a specified location on the map (which is out of range of any of the path locations that enemies traverse on their way to the monument) and is then upgraded by purchasing an upgrade and then applying it on the same tower, no positions on the path the enemies take are updated (i.e. no damage is done to any enemies). This is done by calling the testPlaceTower() method in the LandscapeController class, then the purchaseUpgrade() method in the LandscapeController class, and then the testUpgradeTower() method in the LandscapeController class, and then verifying that the HashMap object returned by the getTestingPathDamage() method is of size zero (which would mean that no damage is done to any enemies on the path)

testPlaceSingleTowerOutOfRangeAndUpgradeTowerNormal

- This test ensures that when a single “normal” tower is placed at a specified location on the map (which is out of range of any of the path locations that enemies traverse on their way to the monument) and is then upgraded by purchasing an upgrade and then applying it on the same tower, no positions on the path the enemies take are updated (i.e. no damage is done to any enemies). This is done by calling the testPlaceTower() method in the LandscapeController class, then the purchaseUpgrade() method in the LandscapeController class, and then the testUpgradeTower() method in the LandscapeController class, and then verifying that the HashMap object returned by the getTestingPathDamage() method is of size zero (which would mean that no damage is done to any enemies on the path)

testPlaceSingleTowerOutOfRangeAndUpgradeTowerElite

- This test ensures that when a single “elite” tower is placed at a specified location on the map (which is out of range of any of the path locations that enemies traverse on their way to the monument) and is then upgraded by purchasing an upgrade and then applying it on the same tower, no positions on the path the enemies take are updated (i.e. no damage is done to any enemies). This is done by calling the testPlaceTower() method in the LandscapeController class, then the purchaseUpgrade() method in the LandscapeController class, and then the testUpgradeTower() method in the LandscapeController class, and then verifying that the HashMap object returned by the getTestingPathDamage() method is of size zero (which would mean that no damage is done to any enemies on the path)

testPlaceOneTowerInRangeOneTowerOutOfRangeUpgradeWrongTowerBad

- This test ensures that when two “bad” towers are placed at distinct locations on the map (one of which is out of range of any of the path locations that enemies traverse on their way to the monument, and the other of which is in range of the path) and the “out-of-range” tower is upgraded by purchasing an upgrade and then applying it on the same tower, no positions on the path the enemies take are updated (i.e. no damage is done to any enemies). This is done by calling the testPlaceTower() method twice in the LandscapeController class, then the purchaseUpgrade() method once in the LandscapeController class, and then the testUpgradeTower() method once on the “out-of-range” tower in the LandscapeController class, and then verifying that the HashMap object returned by the getTestingPathDamage() method is of size zero (which would mean that no damage is done to any enemies on the path)

testPlaceOneTowerInRangeOneTowerOutOfRangeUpgradeWrongTowerNormal

- This test ensures that when two “normal” towers are placed at distinct locations on the map (one of which is out of range of any of the path locations that enemies traverse on their way to the monument, and the other of which is in range of the path) and the “out-of-range” tower is upgraded by purchasing an upgrade and then applying it on the same tower, no positions on the path the enemies take are updated (i.e. no damage is done to any enemies). This is done by calling the testPlaceTower() method twice in the LandscapeController class, then the purchaseUpgrade() method once in the LandscapeController class, and then the testUpgradeTower() method

once on the “out-of-range” tower in the LandscapeController class, and then verifying that the HashMap object returned by the getTestingPathDamage() method is of size zero (which would mean that no damage is done to any enemies on the path)

testPlaceOneTowerInRangeOneTowerOutOfRangeUpgradeWrongTowerElite

- This test ensures that when two “elite” towers are placed at distinct locations on the map (one of which is out of range of any of the path locations that enemies traverse on their way to the monument, and the other of which is in range of the path) and the “out-of-range” tower is upgraded by purchasing an upgrade and then applying it on the same tower, no positions on the path the enemies take are updated (i.e. no damage is done to any enemies). This is done by calling the testPlaceTower() method twice in the LandscapeController class, then the purchaseUpgrade() method once in the LandscapeController class, and then the testUpgradeTower() method once on the “out-of-range” tower in the LandscapeController class, and then verifying that the HashMap object returned by the getTestingPathDamage() method is of size zero (which would mean that no damage is done to any enemies on the path)