CS2340 team ohio squids <3

Directions: Write a brief writeup describing your tests and testing process for the milestone. Explain which implementation components were chosen for testing and why. Additionally, explain how your tests verify that the code functions as expected.

M2 Testing Writeup

Alyse Kwok
I wrote a JUnit test that first checks to see if the start button goes to the configuration screen successfully, and that the difficulty options successfully show up on the drop down to choose the check difficulty. When the player clicks start, the button takes the player to a new screen with a name box and drop down menu. The drop down menu should register a difficulty level that is later tested with other tests.

Anna Zhu
I wrote a JUnit that checks to make sure that the user does not enter a null or a whitespace name. I set a couple of conditions in the code that created a popup and sent a warning to the user that the name was invalid, so the test makes sure that the popup is created and that the screen does not move on to the initial game screen with invalid names.

Ephraim Kwon
I wrote a JUnit that checks to make sure that the map displayed correctly corresponds to its difficulty. This needs to be tested because every difficulty has a different map. My test verifies this by checking if the map corresponds with the correct image and difficulty level.

Carolyn Yuan
I wrote a JUnit that checks to make sure the monument health is correct based on the chosen difficulty. This was chosen for testing because the health needs to be higher for easy level, average for medium level, and lower for the hard level. My test verifies that the code functions as expected by checking if the value of the score is equal to the score that corresponds to the level.

Andy Ngyuen
I wrote a JUnit that checks to make sure the starting money is labeled correctly depending on the difficulty level chosen. This needs to be tested because the starting

money needs to be higher than the other two difficulties when it's easy and lower when it's hard. The starting money of the player will be 500, 750, and 1000 for the easy, medium, and hard levels, respectively. My test makes sure that the label is correct by checking that the value of the money is equal to the amount of money that each difficulty level should have.

M3 Testing Writeup

Andy:
buyTower1 - This test was chosen to see if it's possible to buy the first tower, the Corn Farmer. It clicks the play button, chooses a difficulty, clicks enter, and then buys the Corn Farmer with the buy button. This verifies that the code functions correctly if a Corn Farmer is able to be moved.

buyTower2 - This test was chosen to see if it's possible to buy the first tower, the Corn Bomber. It clicks the play button, chooses a difficulty, clicks enter, and then buys the Corn Bomber with the buy button. This verifies that the code functions correctly if a Corn Bomber is able to be moved.

Alyse:
buyTower 3 - This test was chosen to see if it's possible to buy the first tower, the Corn Ninja. It clicks the play button, chooses a difficulty, clicks enter, and then buys the Corn Ninja with the buy button. This verifies that the code functions correctly if a Corn Ninja is able to be moved onto the map.

easyCost - This test was chosen to see if Corn Farmer costs less, which is $, when the difficulty is easy. It clicks the play button, chooses a difficulty, clicks enter, and then checks the price of the Corn Farmer. This verifies that the code functions correctly if the price is equal to $.

Carolyn:
mediumCost - This test was chosen to see if Corn Farmer costs less, which is $, when the difficulty is medium. It clicks the play button, chooses a difficulty, clicks enter, and then checks the price of the Corn Farmer. This verifies that the code functions correctly if the price is equal to $.

hardCost - This test was chosen to see if Corn Farmer costs less, which is $, when the difficulty is hard. It clicks the play button, chooses a difficulty, clicks enter, and then

checks the price of the Corn Farmer. This verifies that the code functions correctly if the price is equal to $.

Anna:
placeOnMap - We are testing the areas on the map that we can place the tower on. When we drag and drop our towers, we should only be able to place it in certain areas.

deductMoney - When we buy a tower, the cost of the tower should be subtracted from the total amount of money that we have as long as we have sufficient funds.

Ephraim:
description -
easy difficulty tower price - We are testing to see if the starting money and health are equivalent in the easy difficulty stage. Since all of the towers have different prices amongst different towers, we should be able to have the highest health and money when choosing the easy mode.

medium difficulty tower price - We are testing to see if the starting money and health are equivalent in the easy difficulty stage. Since all of the towers have different prices amongst different towers, we should be able to have the second highest health and money when choosing the medium mode.

M4 Testing Writeup

Anna:

enemyAttackMonument - We are testing to see if once the enemy reaches the monument and begins attacking it whether or not the health decreases or not. This is an important functional feature of the game because our goal is to defend the monument and if we fail to protect it, the monument's health will decline and we lose the game.

enemyPath - We are testing to see if the enemies follow the path to the monument. They are not allowed to go anywhere on the screen that is not the path so we need to make sure that it follows the path accordingly.

Alyse:
checkGameOverScreen - We are testing to see if the game over screen occurs when the monument health reaches zero. This is important so that the game can end.

checkRestart - This tests to see whether the restart button on the game over button works. The button should reset the game and return the player to the welcome screen. This is important so that the player can restart the game if desired.

Ephraim:

checkStartCombat - This test is to see if the start combat button or start round works as intended. This button should start the round, and enemies should only start appearing if this button is pushed. This is important because the player should only expect the round to start after they click the button, and not be surprised by enemies appearing without their knowledge.

checkGameOverScreenElements: - This test checks to see if there are elements within the game over screen such as the consoling message and buttons. This is important because you not only need the game over window to appear, but also to have the necessary elements inside to be present and not null.

Carolyn:

checkEnemyStart - This test will verify that enemies appear at the beginning of the map path after the “start combat” button has been clicked. This is important because the enemies should appear at the beginning of the map and advance from there.

checkCloseApplication- This test is for when the player is brought to the Game Over Screen/Menu and the player chooses the option to close the application. It will verify that the application is closed. This is important because the player should be able to exit the game after it ends.

Andy:

checkStartCombatButtonWorksOnce - This test will verify that the start combat button only works once per round. After pressing the button the first time, any other subsequent press should do nothing while the round is still going. This is important because the player shouldn’t be able to start combat multiple times.

checkEnemiesAfterAttack - This test will verify that enemies disappear after attacking the monument. This is important because enemies should only be able to attack and do damage to the monument once before disappearing.

M5 Testing Writeup

1) Cornninja
    a) Attack enemy when it enters tower proximity
    b) Player's money will increase
    c) Enemy loses health
2) Cornfarmer
    a) Different damage
3) Corn
    a) Different speed, range, or area of effect
4) Player
    a) Place new towers after enemies come in

Anna:
checkTowerProximity- This test will check to see if the tower attacks the enemy only when it is in the proximity.
checkDamageAmount- This test will check to see if the different towers cause different damage when attacking enemies.

Alyse:
checkEnemyHealth - This test will check to see if the enemy being attacked by the tower actually loses health. This is important as the enemy needs to take damage and eventually die for the player to win.

checkTowerHealthMonument - This test will check to see if the tower used to heal the monument will actually restore its health. This is important as the tower must implement its functionality

Ephraim:

checkEndOfRoundMoney - This test will check that once the round ends, the player earns money. This is important because players need passive income on top of the money they'll get from killing enemies.

checkTowerShoot - This test will check to see that when a tower is placed, it shoots the enemies. This is important because towers need to eliminate the enemies.

Carolyn:
checkPlaceTower- This test will check to see that the player will be able to place new towers after the enemy comes in.
checkEnemyHealthVisibility- This test will check to see if the enemy visibly loses health on the map screen when attacked by a tower.

Andy:
checkMoneyGainDuringRound - This test will check that the player's money amount is incremented during normal gameplay. This is important because players need to gain money in order to buy more towers.

checkDifferentEnemies - This test will check that multiple different types of enemies spawn during the game. This is important because different enemy types should have distinct gameplay effects that either do different amounts of damage or move with different speeds.