AvatarTest Explanation (T10-G02)

Constructors and Copy Constructors

Testing the initial spawn position of the Avatar object. An error should occur if the object is trying to spawn inside one of the bordering walls.
Copy constructors should clone the original item properly, keeping all variables intact.

test_constructor_left(): testing the constructor's parameters by creating the player inside of the left-most wall. If this error occurs, should default the player's spawn to the INITIAL_X and INITIAL_Y, as found in the ConstantVariables.java file. This error-check occurs in the base class, Item.java.

test_constructor_top(): testing the constructor's parameters by creating the player inside of the uppermost wall. If this error occurs, should default the player's spawn to the INITIAL_X and INITIAL_Y, as found in the ConstantVariables.java file. This error-check occurs in the base class, Item.java.

test_constructor_right(): testing the constructor's parameters by creating the player inside of the right-most wall. If this error occurs, should default the player's spawn to the INITIAL_X and INITIAL_Y, as found in the ConstantVariables.java file. This error-check occurs in the base class, Item.java.

test_constructor_bottom(): testing the constructor's parameters by creating the player inside of the bottom row walls. If this error occurs, should default the player's spawn to the INITIAL_X and INITIAL_Y, as found in the ConstantVariables.java file. This error-check occurs in the base class, Item.java.

test_constructor_proper(): attempts to create the Avatar at a valid spawn point, such as at (6, 3). No error should be caught, and the object should be created at x = 6 and y = 3.

test_constructor_defaultStart(): checks the default spawn position. No errors should occur if the object is created at the default (x, y), as defined by ConstantVariables.java.

test_copyConstructor_xy(): copies an existing Avatar and checks to see if the x and y positions were properly cloned.

test_copyConstructor_score(): copies an existing Avatar and checks for proper cloning of the score.

Getters and Setters

Getters and setters should be tested separately of each other, to ensure that each one works independently. For example, if addScore() and getScore() were to be tested within the same test, an error may occur calling out addScore() when getScore() could be the problem.

test_getScore(): creates an object, then adds 12 points to it. Then, see if getScore() returns 12.

test_addScore(): similar to the first, an object is created and 5 points are added. Then check if the object has a score of 5.

test_setScore_validUpper(): since the game ends once all 606 coins have been collected, the score can not be set to greater than 606. To test the boundary, a score of 606 was given to the test Avatar object. Since this is valid, the object should return a score of 606.

test_setScore_invalidUpper(): it is not possible to collect more than 606 coins, meaning the player's score cannot be above 606. Testing the boundary by setting the item's score to 607 should be caught, and have the score default to 0.

test_setScore_validLower(): score cannot be negative, but it can be 0. An object was created, and its score was set to 0. This is a valid value, and the score should remain unchanged.

test_setScore_invalidLower(): an invalid score would be a negative value. If the score is negative, it should default to the default value, which is 0. An object's score was set to -5, but it should be changed to 0.

test_changeDirection_invalidButton(): only the "wasd" keys can be used to generate a player move. To ensure no invalid keys also allowed for moving of the Avatar, an ArrayList of Strings was created and filled with invalid keys. The changeDirection() method grabs only the first character of a String that is given, and attempts to use this to move. It also sets a direction for the player's x and y, which can be equal to 0, 1, or -1. The list of invalid test Strings to try:

"e" - a key that should not be recognized for movement
"Bop" - tests a longer String, none of which should be recognized
"A" - although "a" is a valid key, "A" is not. Tests uppercase invalids
"ow" - "w" is a valid key, but "o" is not. This tests to see if the method only grabs the first character of the String, not the whole String
"rasdw" - similar to the the "ow" test, this checks if the method only uses the first character, as it should

An Avatar object is created at (1, 1) with an initial direction of "00". This list of invalid Strings is then used inside of a FOR loop, where each element is tested. Every time, the Avatar is expected to remain at its (1, 1) position with no change in either the x or y ("00"), since no valid moves were given.

test_changeDirection_validButton(): similar the the previous test, but this time trying out valid moves. Again, an ArrayList is created with the following Strings:

"d" - valid key, used to move to the right: "10"
"dab on them" - since this String leads with a "d", it is a valid move. Should move to the right
"a" - valid key, used to move left: "-10"
"w" - valid key, used to move up: "0-1"
"s" - valid key, used to move down: "01"

An Avatar is created. This is used inside of a FOR loop, such that each value can be tested separately. Inside of this loop, a variable dir is created, which is set to the expected output for each test. For example, if the key "a" is being tested, x = -1 and y = 0 because it is moving to the left. This means that the expected output would be dir = "-10". This is done for every String in the ArrayList.