CSC-260: Testing Report For Word Find Game

Suites

Most tests were just refactored from last time. There are three major testing suites, each dedicated to the main functionality of the applications. The Parser, the puzzle, and the game.

We decided that visual displays can be "tested" by verifying that it works properly visually. There are no suites for these displays.

BoardParseTests

- parsingTest
  - This test verifies that the parser can read level files properly and throw the correct error messages. The two major cases tested are the following:
    - A correctly formatted file is properly parsed
      - Here I expect to have no errors or exceptions to be thrown so I wrap the parser in a try and except block in the case that it doesn't find the test txt file. In this case I use **junit.Assert.fail**
      -
    - Incorrectly formatted file properly parse
      - I expect an assertionError so I wrap it in a try and except block where I expect only this error to be thrown. I use **junit.Assert.fail;**

PuzzleTests

- arrayListConstructorTest
  - This test verifies that the puzzle and wordlist are properly generated. **junit.Assert.AssertTrue** is used to make sure each object is the same as the one used to initialize.
- stringConstructorTest
  - This test verifies that the puzzle as a string and wordlist are properly created in the puzzle. **junit.Assert.AssertTrue** is used to make sure each object is the same as the one used to initialize.
- containsTest
  - This test verifies that the logic inside the puzzles contains a function that can determine whether a valid word exists inside the puzzle definition.

junit.Assert.AssertFalse and **junit.Assert.AssertTrue** was used to verify this.

- emptyTests
  - This test verifies that the constructors cannot initialize using empty parameters. Try except blocks are used to catch expected failures using **junit.Assert.fail**
- getBoardTest
  - This verifies that the board initialized is the same board stored after it's returned. **junit.Assert.AssertTrue** is used extensively to verify that any edits made to the 2d array won't affect the internal structure in any way.

WordFindGameTests

- attemptTest
  - This test was made to make sure points were given out properly. It verifies that a valid word and invalid words change the internal point count accordingly. **junit.Assert.AssertTrue** is used extensively again.
- getPuzzleTest
  - This verifies that the puzzle that it was initialized with is returned ensuring that it's keeping track of the data properly. **junit.Assert.AssertTrue** is used to verify this
- getTotalPuzzlePointsTest
  - Check that the game can see the objective/goal. Verifies the number of points required to win the game is stored in the object; **junit.Assert.AssertTrue** is used for this
- getWordsFoundTest
  - This checks if certain attempts keep track of the right type of words and stores only those that correlate to points; where **junit.Assert.AssertFalse** is used for this

WordTypeEmojiTests

- noEmojiAssociatedTest
  - This test was made to validate the case where a given word has no emoji associated with it by checking an Emoji library API for possible tags or aliases. It verifies that a given word does not contain an emoji associated and does not have an Emoiji WordType in its list of types. **junit.Assert.AssertEquals** is used for

this purpose to check if it contains the Emoiji WordType or not.

- emojiAssociatedWithTagTest
  - This verifies the case where a given word has an emoji associated with it by having a valid tag equaling the given word. **junit.Assert.AssertTrue** is used to validate that the Emoiji WordType is found within the given list of types.
- emojiAssociatedWithAliasTest
  - This verifies the case where a given word has an emoji associated with it by having a valid alias equaling the given word. **junit.Assert.AssertTrue** is used to validate that the Emoiji WordType is found within the given list of types for this case.

BoardDisplayTests:

- Tested initialization for different size boards.
- Tested reset board to correctly enable all buttons back.
- Tested currentWordDisplay which updates after each button press to add the associated letter to the attempted String.
- Tested button counters after every submission to correctly display the count by getting it from the game.
- Tested the Update function to correctly update buttons with counter and grey-out features.

WordTests:

- equals
  - A simple test that just compares words. Although words may have different word types contained inside we are only really interested in checking if the word is the same
- constructorTest
  - This just verifies that all the constructor makes a word correctly

TileTests:

- equals
  - A simple test that just compares words. Although words may have different word types contained inside we are

only really interested in checking if the word is the same

- constructorTest
  - This just verifies that all the constructor makes a word correctly

14	0	0	0.288s
tests	failures	ignored	duration

100%
successful