Tests Included in This Build

EnumUtilsTest.java

Tests the EnumUtils.java class which is meant to convert EnumSets (full or partial sets of enums) into the Java ObservableList which can then be used to feed FXGL graphical elements with the raw string representations of the specific types associated with an enumerator.

testEmptyEnum()

This tests whether the EnumUtils.java class successfully converts an empty EnumSet into an empty ObservableList. This is the desired behavior since we do not want to either populate UI elements or have errors attributed to a list being empty (an empty category early or late in case progression could be needed).

testSingleElementEnum()

This tests whether the EnumUtils.java class successfully converts an EnumSet containing only one element into a single-element ObservableList. This test confirms appropriate translation from a populated EnumSet to an ObservableList.

testElementOrderFullEnum()

This tests whether the EnumUtils.java class successfully converts an EnumSet containing all elements of an Enum into an Observable list in the natural Enum order. This is the desired outcome as we want the enumerations to work as our baseline order and for our methods to not modify that without explicit purpose.

PlayerComponentTest.java

Tests the PlayerComponent.java class which is meant to be a representation of the game's player. It contains various properties like health, strength, and is a subclass of FXGL's Component class meant to represent game entities.

checkAttributes()

This tests whether the attributes that the PlayerComponent object was initialized with get correctly returned when calling get[Attribute]() and also checks whether they are of the correct value.

checkMovement()

This tests whether the translate() methods of the PlayerComponent class correctly move the player by the default distance of 10. Tests up, down, left, and right directions. At the end of the test. The player is back in their original position.

MoneyTest.java

This tests the money system that is embedded in FXGL vars as well as in the PlayerComponent class. The interdependence between the PlayerComponent object that we are using for the player stats as well as the FXGL vars allows us to tie the player object with the UI systems in game, per the requirements of FXGL's UI system. There is also a check for the constants function to show that they are returning the expected enumerated values.

getDifficultyAndGear()

This tests whether or not the type DifficultyLevel and WeaponType have properly been set up, as well as checking that the functions to return the default selection (via enum) correctly returns the needed values.

checkAddSubtractFunds()

This tests the playerComponent.addFunds() method. It takes the current value of the player's money and updates both it and the FXGL var that mirrors the local value. It works for both positive and negative values. This function is important because it updates the interdependence between the player and the UI level.

checkShowFunds()

This function shows the current amount of money that the player has, and does not update or modify this value. This is useful for calls to check how much money the player has currently. The value that the player has is shared to FXGL vars.

InitGameTest.java

This tests for after the player has clicked “start” in the player configuration screen. It checks that the right values for the player’s money and the correct sprite are used.

testSprite()

This tests that the player’s selected sprite is loaded from the correct image. To use this test you must change the png to the one corresponding with the weapon the user will select in the game:

```
Texture texture = FXGL.texture("ninja.png");
```

You then exit the game and the test will run.

testMoney()

This tests that the player selected difficulty initializes the correct starting money. To use this test you must change the money value to the one corresponding to the difficulty:

```
int money = 10;
```

You then exit the game and the test will run.