**Assignment 3 - Iteration 5**

**Task:** Add drag and drop coin, sound, animations, and other JavaScript functions for working vending machine

**WHERE**

**Class Diagram**

A picture containing chart

Description automatically generated

**WHAT**

**PLAN**

**Goal of the iteration:** I will be adding animations, sound effects and other functionality in this final iteration is to have a full functional complete vending machine.

|  |  |  |
| --- | --- | --- |
| TASK | TIME ESTIMATED | TIME TAKEN |
| 1. Research sounds, animations | 30 Hour | 25 Minutes |
| 2. Planning Class Diagram | 30 Minutes | 30 minutes |
| 3. Create sound and animations effects in JavaScript and CSS | 1 Hour | 1 Hour |
| 4. Create functions inside vending machine class to get a working vending machine | 2 Hours | 4 Hour |

**HOW**

**Unit Tests**

**Add and display vending machine**



import unittest  
from vending\_machine import VendingMachine  
from controller import setup  
  
  
class TestVendingMachinePropertiesExist(unittest.TestCase):  
 def test\_vm\_name\_exists(self):  
 the\_vm = VendingMachine()  
 self.assertTrue(hasattr(the\_vm, 'name'))  
  
 def test\_vm\_location\_exists(self):  
 the\_vm = VendingMachine()  
 self.assertTrue(hasattr(the\_vm, 'location'))  
  
  
class TestVendingMachineGetsCorrectValuesFromControllerSetup(unittest.TestCase):  
 def test\_vm\_name\_correct(self):  
 the\_vm = setup()  
 self.assertEqual(the\_vm.name, 'Ara Vending Machine')  
  
 def test\_vm\_location\_correct(self):  
 the\_vm = setup()  
 self.assertEqual(the\_vm.location, 'Madras Street')  
  
  
class TestVendingMachineGetsCorrectTypesFromControllerSetup(unittest.TestCase):  
 def test\_vm\_name\_has\_no\_leading\_spaces(self):  
 the\_vm = setup()  
 self.assertFalse(the\_vm.name.startswith(' '))  
  
 def test\_vm\_name\_has\_no\_trailing\_spaces(self):  
 the\_vm = setup()  
 self.assertFalse(the\_vm.name.endswith(' '))  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 unittest.main(verbosity=3)

**Add and Display Coins – BEFORE FAILED**

While building the functional vending machine, I realised more variables and values needed to be added to some of the classes, this broke my unit test but has now been fixed (see below)

import unittest  
from vending\_machine import VendingMachine  
from controller import setup  
from money import Money  
  
  
class TestVendingMachinePropertiesExist(unittest.TestCase):  
 def test\_name\_property\_exists(self):  
 the\_money = Money('fiveDollar', 500, 10)  
 self.assertTrue(hasattr(the\_money, 'name'))  
  
 def test\_value\_property\_exists(self):  
 the\_money = Money('fiveDollar', 500, 10)  
 self.assertTrue(hasattr(the\_money, 'value'))  
  
 def test\_quantity\_property\_exists(self):  
 the\_money = Money('fiveDollar', 500, 10)  
 self.assertTrue(hasattr(the\_money, 'quantity'))  
  
  
class TestVendingMachineGetsCorrectValuesFromControllerSetup(unittest.TestCase):  
 def test\_vm\_name\_correct(self):  
 the\_money = Money('fiveDollar', 500, 10)  
 self.assertEqual(the\_money.name, 'fiveDollar')  
  
 def test\_vm\_value\_correct(self):  
 the\_money = Money('fiveDollar', 500, 10)  
 self.assertEqual(the\_money.value, 500)  
  
 def test\_vm\_quantity\_correct(self):  
 the\_money = Money('fiveDollar', 500, 10)  
 self.assertEqual(the\_money.quantity, 10)  
  
  
class TestVendingMachineGetsCorrectTypesFromControllerSetup(unittest.TestCase):  
 def test\_vm\_name\_has\_no\_leading\_spaces(self):  
 the\_vm = setup()  
 self.assertFalse(the\_vm.name.startswith(' '))  
  
 def test\_vm\_name\_has\_no\_trailing\_spaces(self):  
 the\_vm = setup()  
 self.assertFalse(the\_vm.name.endswith(' '))  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 unittest.main(verbosity=3)

**Add and Display Coins – AFTER PASSED**

As you can see below with the yellow highlighting, this is something I added in while completing the last iteration.



import unittest  
from vending\_machine import VendingMachine  
from controller import setup  
from money import Money  
  
  
class TestVendingMachinePropertiesExist(unittest.TestCase):  
 def test\_name\_property\_exists(self):  
 the\_money = Money('fiveDollar', 500, 10, '5-dollar.png')  
 self.assertTrue(hasattr(the\_money, 'name'))  
  
 def test\_value\_property\_exists(self):  
 the\_money = Money('fiveDollar', 500, 10, '5-dollar.png')  
 self.assertTrue(hasattr(the\_money, 'value'))  
  
 def test\_quantity\_property\_exists(self):  
 the\_money = Money('fiveDollar', 500, 10, '5-dollar.png')  
 self.assertTrue(hasattr(the\_money, 'quantity'))  
  
  
class TestVendingMachineGetsCorrectValuesFromControllerSetup(unittest.TestCase):  
 def test\_vm\_name\_correct(self):  
 the\_money = Money('fiveDollar', 500, 10, '5-dollar.png')  
 self.assertEqual(the\_money.name, 'fiveDollar')  
  
 def test\_vm\_value\_correct(self):  
 the\_money = Money('fiveDollar', 500, 10, '5-dollar.png')  
 self.assertEqual(the\_money.value, 500)  
  
 def test\_vm\_quantity\_correct(self):  
 the\_money = Money('fiveDollar', 500, 10, '5-dollar.png')  
 self.assertEqual(the\_money.quantity, 10)  
  
  
class TestVendingMachineGetsCorrectTypesFromControllerSetup(unittest.TestCase):  
 def test\_vm\_name\_has\_no\_leading\_spaces(self):  
 the\_vm = setup()  
 self.assertFalse(the\_vm.name.startswith(' '))  
  
 def test\_vm\_name\_has\_no\_trailing\_spaces(self):  
 the\_vm = setup()  
 self.assertFalse(the\_vm.name.endswith(' '))  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 unittest.main(verbosity=3)

**Add and Buy Product – PASSED**



import unittest  
from vending\_machine import VendingMachine  
from controller import setup  
from product import Product  
  
  
class TestVendingMachinePropertiesExist(unittest.TestCase):  
 def test\_name\_property\_exists(self):  
 the\_product = Product('Cola', 250, 'a1', 5)  
 self.assertTrue(hasattr(the\_product, 'name'))  
  
 def test\_price\_property\_exists(self):  
 the\_product = Product('Cola', 250, 'a1', 5)  
 self.assertTrue(hasattr(the\_product, 'price'))  
  
 def test\_position\_property\_exists(self):  
 the\_product = Product('Cola', 250, 'a1', 5)  
 self.assertTrue(hasattr(the\_product, 'position'))  
  
 def test\_quantity\_property\_exists(self):  
 the\_product = Product('Cola', 250, 'a1', 5)  
 self.assertTrue(hasattr(the\_product, 'quantity'))  
  
  
class TestVendingMachineGetsCorrectValuesFromControllerSetup(unittest.TestCase):  
 def test\_vm\_alloy\_correct(self):  
 the\_product = Product('Cola', 250, 'a1', 5)  
 self.assertEqual(the\_product.name, 'Cola')  
  
 def test\_vm\_diameter\_correct(self):  
 the\_product = Product('Cola', 250, 'a1', 5)  
 self.assertEqual(the\_product.price, 250)  
  
 def test\_vm\_weight\_correct(self):  
 the\_product = Product('Cola', 250, 'a1', 5)  
 self.assertEqual(the\_product.position, 'a1')  
  
 def test\_vm\_value\_correct(self):  
 the\_product = Product('Cola', 250, 'a1', 5)  
 self.assertEqual(the\_product.quantity, 5)  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 unittest.main(verbosity=3)

**Buy and Calculate change - PASSED**



import unittest  
from vending\_machine import VendingMachine  
from controller import setup  
from product import Product  
  
  
class TestVendingMachinePropertiesExist(unittest.TestCase):  
 def test\_name\_property\_exists(self):  
 the\_product = Product('Cola', 250, 'a1', 5)  
 self.assertTrue(hasattr(the\_product, 'name'))  
  
 def test\_price\_property\_exists(self):  
 the\_product = Product('Cola', 250, 'a1', 5)  
 self.assertTrue(hasattr(the\_product, 'price'))  
  
 def test\_position\_property\_exists(self):  
 the\_product = Product('Cola', 250, 'a1', 5)  
 self.assertTrue(hasattr(the\_product, 'position'))  
  
 def test\_quantity\_property\_exists(self):  
 the\_product = Product('Cola', 250, 'a1', 5)  
 self.assertTrue(hasattr(the\_product, 'quantity'))  
  
  
class TestVendingMachineGetsCorrectValuesFromControllerSetup(unittest.TestCase):  
 def test\_vm\_name\_correct(self):  
 the\_product = Product('Cola', 250, 'a1', 5)  
 self.assertEqual(the\_product.name, 'Cola')  
  
 def test\_vm\_price\_correct(self):  
 the\_product = Product('Cola', 250, 'a1', 5)  
 self.assertEqual(the\_product.price, 250)  
  
 def test\_vm\_position\_correct(self):  
 the\_product = Product('Cola', 250, 'a1', 5)  
 self.assertEqual(the\_product.position, 'a1')  
  
 def test\_vm\_quantity\_correct(self):  
 the\_product = Product('Cola', 250, 'a1', 5)  
 self.assertEqual(the\_product.quantity, 5)  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 unittest.main(verbosity=3)

**Sequence Diagram**

A picture containing graphical user interface

Description automatically generated

**Timeline**

Timeline

Description automatically generated

**Testing**

**JSLINT**

Text

Description automatically generated

As you can see the JSLINT errors above,

* onMoneyDrag() is outside the vending machine class and doesn’t recognise it is being used under certain conditions
* “Promised returned from play is ignored” this is coming from a sound file using .play(). This error can be because certain browsers don’t yet support this. To fix this I used .then and. catch function to allow the audio to play or return an automatic playback failure.
* Final warnings are just typos – nothing to do here as it is the way I want the words

**JSLINT – ITERATION 5 COMPLETE**

Text

Description automatically generated

**PYLINT**

**PYLINT – FROM ITERATION 4**

Text

Description automatically generated

* Fixed all “Line too long” by breaking down my formatting strings
* Fixed all methods that had “no-self-use” by using the arrays (self.all\_my\_products):
* def get\_vending\_product(self):  
   append\_product = ""  
   for product in self.all\_the\_vending\_products:  
   append\_product += f"{product.name}: {{name: '{product.name}', price: {product.price}," \  
   f" position: '{product.position}', quantity: {product.quantity}}}, "  
   return append\_product

**PYLINT – ITERATION 5 - PASSED**



**EVALUATION**

**Performance Review:**

**What happened vs what was planned**

* In this final iteration I found that I may not have evenly distributed the iterations as I found the final one has a few unexpected surprises and was very time consuming to add all the functionality that I wanted.
* Unit tests in Python failed unexpectedly, which I should have expected considering I changes a few classes and added more parameters to accommodate functionality issues.
* Had some seriously problems with my CSS be got this resolved after working out positions (relevant and absolute) and z-index’s to allow elements to slide down over (on-top) of other elements.

**Personal Reflection:**

This final iteration taught me a lot about planning and preparation. Many issues arise due to me thinking I could create most of the functionality in the very last iteration, this made this iteration around three times longer to complete than the previous iteration. Overall I think I completed my vending machine to a high standard with a nice easy to use user interface, however there are more features I would have liked to add but I had already gone far past the planned timeframe I was expecting to complete it in.