**Task 4.1 – ShapeDrawer**

**Shape.cs**

using System;

using SplashKitSDK;

namespace ShapeDrawer;

public class Shape

{

//Fields

private Color \_color; //changed from string to Color

private float \_x;

private float \_y;

private int \_width;

private int \_height;

//Create constructor

public Shape(int param)

{

\_color = Color.Chocolate; // As my name is Min Thu Kyaw Khaung, the first letter 'M' which is after A-L.

\_x = 0.0f;

\_y = 0.0f;

\_width = param;

\_height = param;

}

//Draw the shape

public void Draw()

{

SplashKit.FillRectangle(\_color, \_x, \_y, \_width, \_height); //changed from Console.WriteLine statements

}

//Check if the shape is at the position (xInput,yInput)

//IsAt method

public bool IsAt(Point2D pt)

{

return pt.X >= \_x && pt.X <= (\_x + \_width) &&

pt.Y >= \_y && pt.Y <= (\_y + \_height);

}

//Properties

public Color Color

{

get { return \_color; }

set { \_color = value; }

}

public float X

{

get { return \_x; }

set { \_x = value; }

}

public float Y

{

get { return \_y; }

set { \_y = value; }

}

public int Width

{

get { return \_width; }

set { \_width = value; }

}

public int Height

{

get { return \_height; }

set { \_height = value; }

}

}

**Program.cs**

using System;

using SplashKitSDK;

namespace ShapeDrawer;

public class Program

{

public static void Main()

{

Window window = new Window("Shape Drawer", 800, 600);

Shape myShape = new Shape(181);

do

{

SplashKit.ProcessEvents();

SplashKit.ClearScreen();

if (SplashKit.MouseClicked(MouseButton.LeftButton))

{

myShape.X = SplashKit.MouseX();

myShape.Y = SplashKit.MouseY();

}

if (SplashKit.KeyTyped(KeyCode.SpaceKey))

{

Point2D mousePos = SplashKit.MousePosition();

if (myShape.IsAt(mousePos))

{

myShape.Color = SplashKit.RandomColor();

}

}

myShape.Draw();

SplashKit.RefreshScreen();

} while (!window.CloseRequested);

}

}

**Task 4.2 – SwinAdventure Iteration 3 Inventory**

**Inventory.cs**

using System;

using System.Collections.Generic;

namespace SwinAdventure

{

public class Inventory

{

// Fields

private List<Item> \_items;

//Constructor

public Inventory()

{

\_items = new List<Item>();

}

//Methods

public bool HasItem(string id)

{

foreach (Item item in \_items)

{

if (item.AreYou(id))

{

return true;

}

}

return false;

}

public void Put(Item itm)

{

\_items.Add(itm);

}

public Item? Take(string id)

{

for (int i = 0; i < \_items.Count; i++)

{

if (\_items[i].AreYou(id))

{

Item item = \_items[i];

\_items.RemoveAt(i);

return item;

}

}

return null;

}

public Item? Fetch(string id)

{

foreach (Item item in \_items)

{

if (item.AreYou(id))

{

return item;

}

}

return null;

}

//Property

public string ItemList

{

get

{

string result = "";

foreach (Item item in \_items)

{

result = result + "\t" + item.ShortDescription + "\n";

}

return result;

}

}

}

}

**InventoryTests.cs**

using NUnit.Framework;

using SwinAdventure;

namespace SwinAdventure.Tests

{

[TestFixture]

public class InventoryTests

{

private Inventory \_inventory;

private Item \_testItem1;

private Item \_testItem2;

[SetUp]

public void Setup()

{

\_inventory = new Inventory();

\_testItem1 = new Item(new string[] { "sword", "axe" }, "bronze sword", "A basic bronze sword");

\_testItem2 = new Item(new string[] { "gem", "ruby" }, "red gem", "A shiny red ruby");

}

[Test] //The Inventory has items that are put in it.

public void TestFindItem()

{

// Arrange

\_inventory.Put(\_testItem1);

\_inventory.Put(\_testItem2);

// Act & Assert

Assert.That(\_inventory.HasItem("sword"), Is.True, "Should find sword in inventory");

Assert.That(\_inventory.HasItem("axe"), Is.True, "Should find weapon identifier for sword");

Assert.That(\_inventory.HasItem("gem"), Is.True, "Should find gem in inventory");

Assert.That(\_inventory.HasItem("ruby"), Is.True, "Should find ruby identifier for gem");

}

[Test] //The Inventory does not have items it does not contain.

public void TestNoItemFind()

{

// Arrange

\_inventory.Put(\_testItem1);

// Act & Assert

Assert.That(\_inventory.HasItem("shield"), Is.False, "Should not find shield in inventory");

Assert.That(\_inventory.HasItem("potion"), Is.False, "Should not find potion in inventory");

Assert.That(\_inventory.HasItem("gold"), Is.False, "Should not find gem when not in inventory");

}

[Test] //Returns items it has, and the item remains in the inventory.

public void TestFetchItem()

{

// Arrange

\_inventory.Put(\_testItem1);

\_inventory.Put(\_testItem2);

// Act

Item? fetchedSword = \_inventory.Fetch("sword");

Item? fetchedGem = \_inventory.Fetch("gem");

// Assert

Assert.That(fetchedSword, Is.Not.Null, "Should return a valid item");

Assert.That(fetchedGem, Is.Not.Null, "Should return a valid item");

Assert.That(fetchedSword, Is.SameAs(\_testItem1), "Should return the same sword item");

Assert.That(fetchedGem, Is.SameAs(\_testItem2), "Should return the same gem item");

// Verify items are still in inventory after fetch

Assert.That(\_inventory.HasItem("sword"), Is.True, "Sword should still be in inventory after fetch");

Assert.That(\_inventory.HasItem("gem"), Is.True, "Gem should still be in inventory after fetch");

}

[Test] // Returns the item, and the item is no longer in the inventory.

public void TestTakeItem()

{

// Arrange

\_inventory.Put(\_testItem1);

\_inventory.Put(\_testItem2);

// Act

Item? takenSword = \_inventory.Take("sword");

Item? takenGem = \_inventory.Take("gem");

// Assert

Assert.That(takenSword, Is.Not.Null, "Should return a valid item");

Assert.That(takenGem, Is.Not.Null, "Should return a valid item");

Assert.That(takenSword, Is.SameAs(\_testItem1), "Should return the same sword item");

Assert.That(takenGem, Is.SameAs(\_testItem2), "Should return the same gem item");

// Verify item is no longer in inventory after take

Assert.That(\_inventory.HasItem("sword"), Is.False, "Sword should not be in inventory after take");

Assert.That(\_inventory.HasItem("gem"), Is.False, "Gem should not be in inventory after take");

}

[Test] //Returns a string containing multiple lines. Each line contains a tab-indented short description of an item in the Inventory.

public void TestItemList()

{

// Arrange

\_inventory.Put(\_testItem1);

\_inventory.Put(\_testItem2);

// Act

string itemList = \_inventory.ItemList;

// Assert

Assert.That(itemList.Contains("\ta bronze sword (sword)"), Is.True, "Item list should contain tabbed sword description");

Assert.That(itemList.Contains("\ta red gem (gem)"), Is.True, "Item list should contain tabbed gem description");

}

}

}