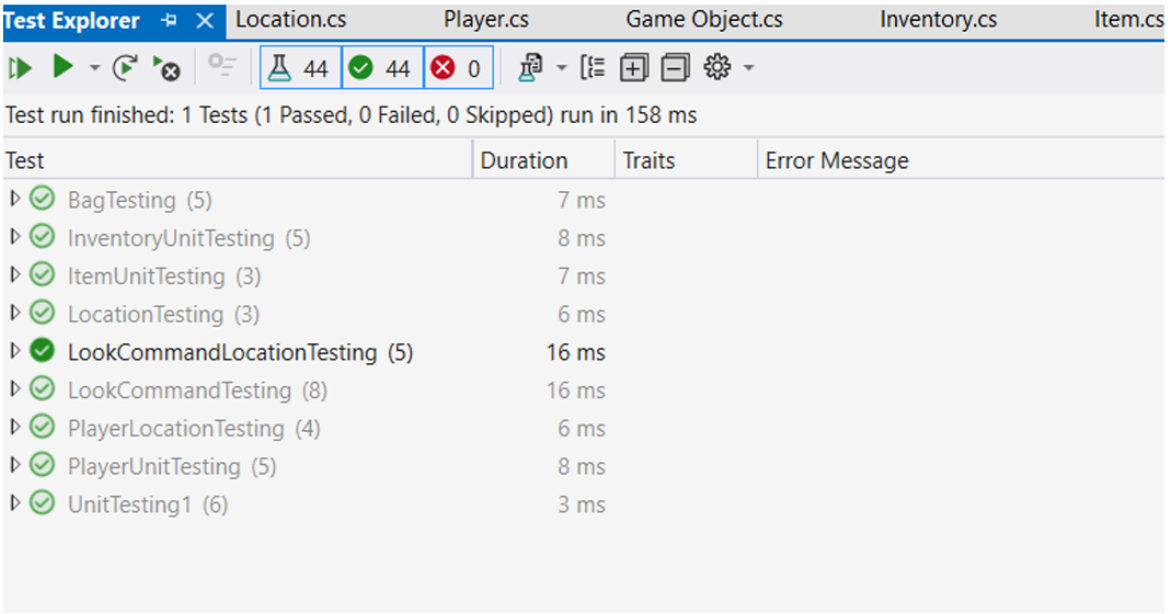
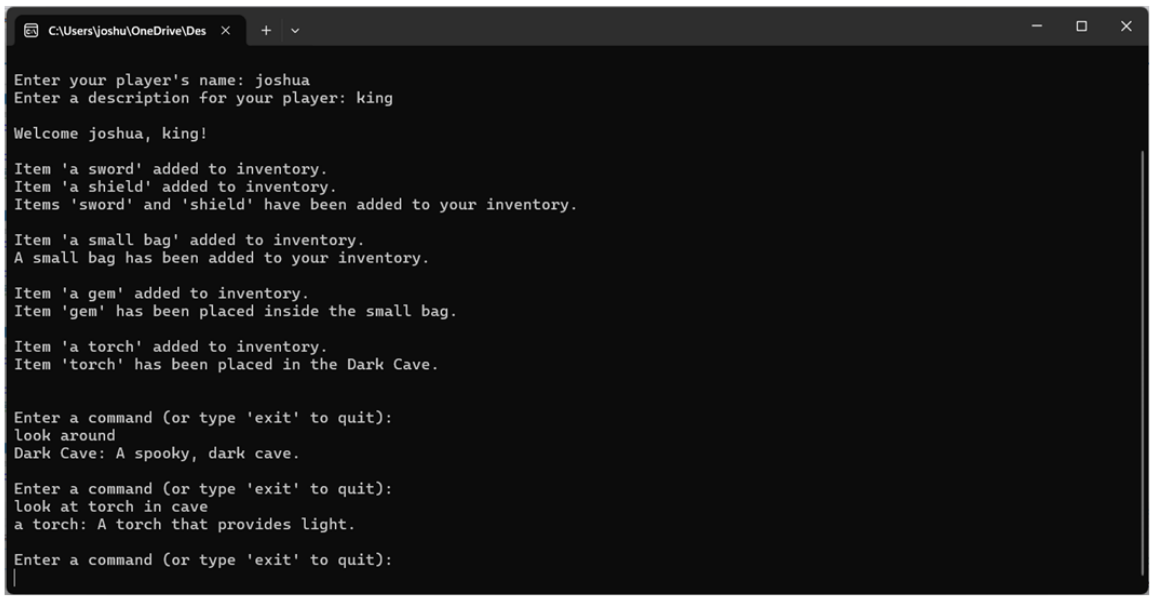
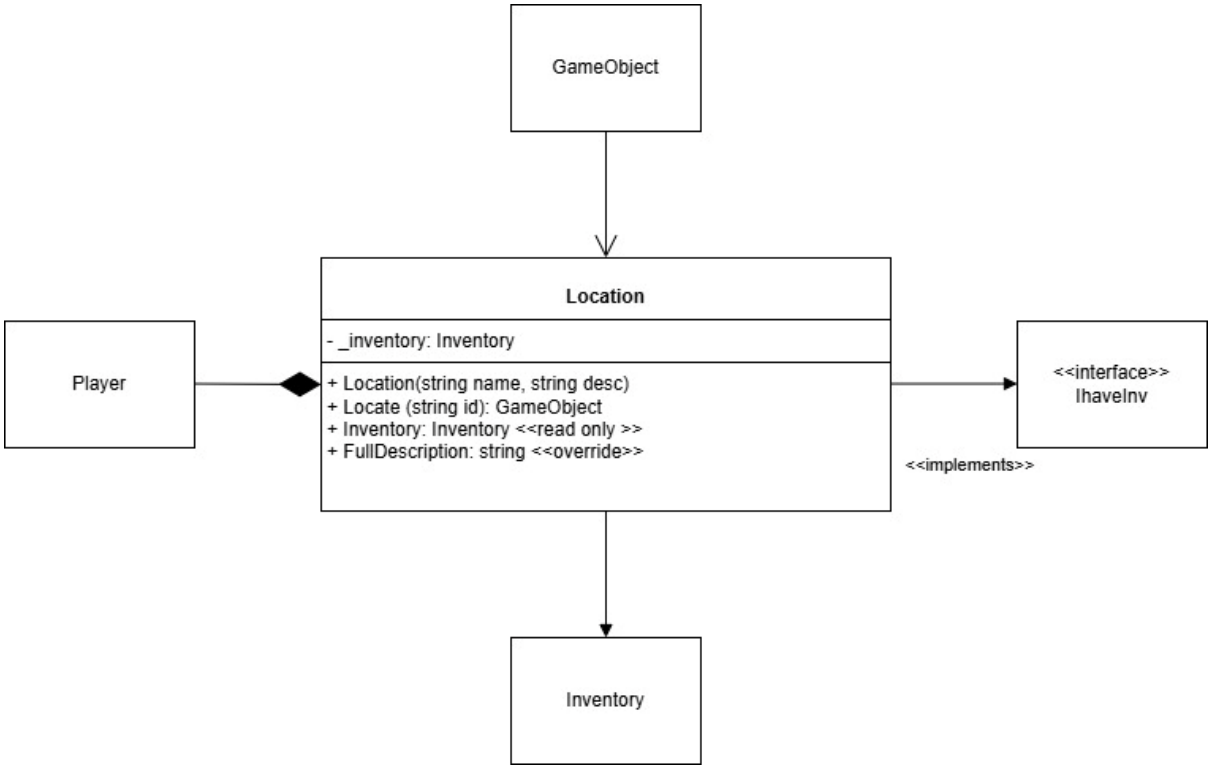
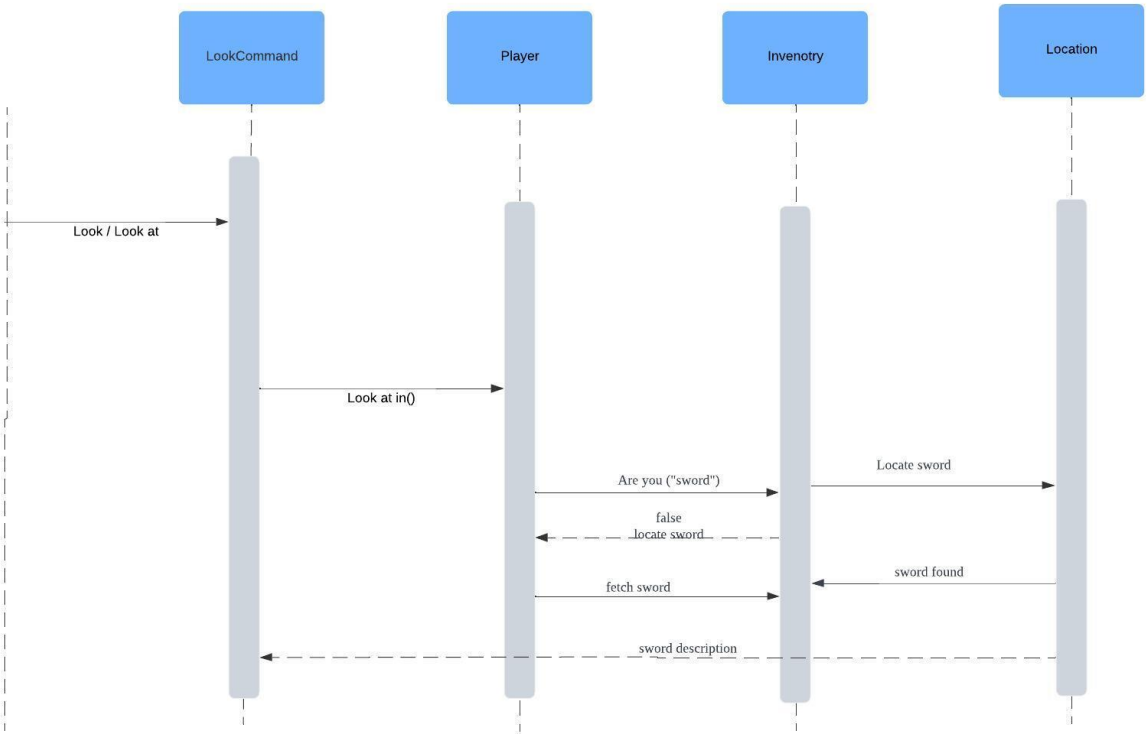
7.2 Corrections






GameObject
Location
- _inventory: Inventory
+ Location(string name, string desc)
+ Locate (string id): GameObject
+ Inventory: Inventory <<read only >>
+ FullDescription: string <<override>>
Player
<<interface>>
IhaveInv
<<implements>>
Inventory


LookCommand
Player
Invenotry
Location
Look / Look at
Look at in()
Are you ("sword")
Locate sword
false
locate sword
sword found
fetch sword
sword description

Location.cs:

```
namespace SwinAdventure
{
    public class Location : GameObject, IHaveInventory
    {
        private Inventory _inventory;

        public Location(string[] ids, string name, string description) : base(ids, name, description)
        {
            _inventory = new Inventory();
        }

        public Inventory Inventory
        {
            get { return _inventory; }
        }

        public GameObject Locate(string id)
        {
            // Check if the location itself is being looked for
            if (AreYou(id))
            {
                return this;
            }

            // Check if the item is in the location's inventory
            return _inventory.Fetch(id);
        }

        public override string GetFullDescription()
        {
```

```
            return $"{Name}: {ShortDescription}";
        }
    }
}
```

LookCommand.cs:

```
namespace SwinAdventure
{
    public class LookCommand : Command
    {
        public LookCommand() : base(new string[] { "look", "inventory" }) { }

        public override string Execute(Player player, string[] text)
        {
            // Case 1: "inventory" to display player's inventory
            if (text.Length == 1 && text[0].ToLower() == "inventory")
            {
                return player.Inventory.ItemList.Length > 0
                    ? $"You are carrying:\n{player.Inventory.ItemList}"
                    : "You are carrying nothing.";
            }

            // Case 2: "look" command to describe the current location
            if (text.Length == 1 && text[0].ToLower() == "look")
            {
                return player.Location?.GetFullDescription() ?? "You are nowhere.";
            }

            // Case 3: "look around"
            if (text.Length == 2 && text[0].ToLower() == "look" && text[1].ToLower() == "around")
            {
                return player.Location?.GetFullDescription() ?? "You are nowhere.";
```

```
}

// Case 4: "look at inventory in <location>"
if (text.Length == 5 && text[0].ToLower() == "look" && text[1].ToLower() == "at"
    && text[2].ToLower() == "inventory" && text[3].ToLower() == "in")
{
    string locationId = text[4].ToLower();
    IHaveInventory location = FetchContainer(player, locationId);

    if (location == null)
    {
        return $"I cannot find the {locationId}.";
    }

    if (location is Location loc)
    {
        return loc.Inventory.ItemList.Length > 0
            ? $"The {loc.Name} contains:\n{loc.Inventory.ItemList}"
            : $"The {loc.Name} is empty.";
    }

    return $"The {location.Name} does not contain an inventory.";
}

// Case 5: "look at <item>" or "look at <item> in <container>"
if (text.Length == 3 || text.Length == 5)
{
    if (text[0].ToLower() != "look" || text[1].ToLower() != "at")
    {
        return "Error in look input.";
    }
```

```
string itemId = text[2].ToLower();
IHaveInventory container;

if (text.Length == 5)
{
   if (text[3].ToLower() != "in")
   {
      return "What do you want to look in?";
   }

   string containerId = text[4].ToLower();
   container = FetchContainer(player, containerId);

   if (container == null)
   {
      return $"I cannot find the {containerId}.";
   }
}
else
{
   container = player; // Default to player's inventory
}

GameObject item = container.Locate(itemId);

if (item == null)
{
   return $"I cannot find the {itemId} in the {container.Name}.";
}
```

```
            return item.GetFullDescription();
        }

        // Case 6: "look at me"
        if (text.Length == 3 && text[0].ToLower() == "look" && text[1].ToLower() == "at" && text[2].ToLower() == "me")
        {
            return player.GetFullDescription();
        }

        return "I don't understand what you want to do.";
    }

    private IHaveInventory FetchContainer(Player player, string containerId)
    {
        // Search for the container in the player's location or inventory
        return player.Locate(containerId) as IHaveInventory ?? player.Location?.Locate(containerId) as IHaveInventory;
    }
  }
}
```

Player.cs:

```
using MiNET.Utils.Skins;
using System;

namespace SwinAdventure
{
  public class Player : GameObject, IHaveInventory
  {
    private Inventory _inventory;
    private Location _location;
```

```
    public Player(string name, string description) : base(new string[] { "me", "inventory" }, name, description)
    {
        _inventory = new Inventory();
    }

    public override string GetFullDescription()
    {

        return $"You are {Name}, {ShortDescription}. You are holding " + _inventory.ItemList;
    }
    public Location Location
    {
        get { return _location; }
        set { _location = value; }
    }

    public GameObject Locate(string id)
    {
        if (AreYou(id)) return this;

        GameObject item = Inventory.Fetch(id);
        if (item != null) return item;

        return _location?.Locate(id);
    }


    public Inventory Inventory
    {
        get { return _inventory; }
```

```
        }
    }
}
```

Program.cs:

```
using System;
using SwinAdventure;

class Program
{
    static void Main(string[] args)
    {
        // Display list of available commands at the beginning
        Console.WriteLine("Welcome to Swin-Adventure!");
        Console.WriteLine("Here are the commands you can use:");
        Console.WriteLine(" - 'look at [item]': Look at an item in your inventory or in the location.");
        Console.WriteLine(" - 'look at [item] in [container]': Look at an item inside a container (e.g., a bag).");
        Console.WriteLine(" - 'look': Look around your current location and see items there.");
        Console.WriteLine(" - 'exit': Exit the game.\n");

        // Step 1: Get the player's name and description from the user.
        Console.Write("Enter your player's name: ");
        string playerName = Console.ReadLine();

        Console.Write("Enter a description for your player: ");
        string playerDescription = Console.ReadLine();

        // Create a Player object with the user's input.
        Player player = new Player(playerName, playerDescription);
        Console.WriteLine($"\nWelcome {playerName}, {playerDescription}!\n");
```

```
// Step 2: Create items and add them to the player's inventory.
Item sword = new Item(new string[] { "sword" }, "a sword", "A sharp, shining sword.");
Item shield = new Item(new string[] { "shield" }, "a shield", "A sturdy wooden shield.");
player.Inventory.Put(sword);
player.Inventory.Put(shield);
Console.WriteLine("Items 'sword' and 'shield' have been added to your inventory.\n");

// Step 3: Create a bag and add it to the player's inventory.
Bag smallBag = new Bag(new string[] { "bag", "small bag" }, "a small bag", "A small leather bag.");
player.Inventory.Put(smallBag);
Console.WriteLine("A small bag has been added to your inventory.\n");

// Step 4: Create another item and add it to the bag.
Item gem = new Item(new string[] { "gem" }, "a gem", "A shiny, valuable gem.");
smallBag.Inventory.Put(gem);
Console.WriteLine("Item 'gem' has been placed inside the small bag.\n");

// Step 5: Set up the location and add items to it
Location darkCave = new Location(new string[] { "cave" }, "Dark Cave", "A spooky, dark cave.");
Item torch = new Item(new string[] { "torch" }, "a torch", "A torch that provides light.");
darkCave.Inventory.Put(torch);  // Place the torch in the dark cave
Console.WriteLine("Item 'torch' has been placed in the Dark Cave.\n");

// Set the player's initial location
player.Location = darkCave;

// Step 6: Create the LookCommand
LookCommand lookCommand = new LookCommand();
```

```
        // Step 7: Main command loop
        while (true)
        {
            Console.WriteLine("\nEnter a command (or type 'exit' to quit):");
            string command = Console.ReadLine();

            if (command.ToLower() == "exit")
            {
                break; // Exit the loop and end the program.
            }

            // Split the command into an array of words
            string[] commandWords = command.Split(' ');

            // Execute the LookCommand with the player and command words
            string result = lookCommand.Execute(player, commandWords);
            Console.WriteLine(result);
        }

        Console.WriteLine("Goodbye!");
    }
}
```

Player Location testing:

```
using NUnit.Framework;
using System.Numerics;
using SwinAdventure;
namespace SwinAdventure.Tests
{
    [TestFixture]
    public class PlayerTests
```

```
{
    private Player _player;
    private Location _location;
    private Item _inventoryItem;
    private Item _locationItem;

    [SetUp]
    public void SetUp()
    {
        _player = new Player("Hero", "A brave adventurer.");
        _location = new Location(new string[] { "forest" }, "Dark Forest", "A spooky dark forest.");
        _inventoryItem = new Item(new string[] { "sword" }, "Sword", "A sharp blade.");
        _locationItem = new Item(new string[] { "shield" }, "Shield", "A sturdy shield.");

        _player.Inventory.Put(_inventoryItem);
        _location.Inventory.Put(_locationItem);
        _player.Location = _location;
    }

    [Test]
    public void PlayerCanLocateItself()
    {
        Assert.AreEqual(_player, _player.Locate("inventory"));
    }

    [Test]
    public void PlayerCanLocateItemInInventory()
    {
        Assert.AreEqual(_inventoryItem, _player.Locate("sword"));
    }
```

```
        [Test]
        public void PlayerCanLocateItemInLocation()
        {
            Assert.AreEqual(_locationItem, _player.Locate("shield"));
        }

        [Test]
        public void PlayerCannotLocateNonexistentItem()
        {
            Assert.IsNull(_player.Locate("potion"));
        }
    }
}
```

LookCommand testing:

```
using NUnit.Framework;

namespace SwinAdventure.Tests
{
    [TestFixture]
    public class LookCommandTests
    {
        private Player _player;
        private LookCommand _lookCommand;
        private Item _gem;
        private Bag _bag;

        [SetUp]
        public void Setup()
        {
            // Initialize LookCommand
```

```
        _lookCommand = new LookCommand();

        // Initialize player, gem, and bag
        _player = new Player("Player", "A brave adventurer");
        _gem = new Item(new string[] { "gem" }, "a gem", "A shiny red gem");
        _bag = new Bag(new string[] { "bag" }, "a small bag", "A small leather bag");

        // Add gem to the player's inventory
        _player.Inventory.Put(_gem);

        // Add the bag to the player's inventory
        _player.Inventory.Put(_bag);

        // Add another gem to the bag
        Item anotherGem = new Item(new string[] { "gem" }, "another gem", "Another shiny gem");
        _bag.Inventory.Put(anotherGem);
    }

    [Test]
    public void TestLookAtMe()
    {
        string result = _lookCommand.Execute(_player, new string[] { "look", "at", "inventory" });
        Assert.AreEqual("You are Player, A brave adventurer. You are holding \tA shiny red gem
(gem)\n\tA small leather bag (bag)\n", result);
    }

    [Test]
    public void TestLookAtGem()
    {
        string result = _lookCommand.Execute(_player, new string[] { "look", "at", "gem" });
        Assert.AreEqual("a gem: A shiny red gem", result);
```

```
    }

    [Test]
    public void TestLookAtUnk()
    {
        string result = _lookCommand.Execute(_player, new string[] { "look", "at", "unknown" });
        Assert.AreEqual("I cannot find the unknown in  Player", result);
    }

    [Test]
    public void TestLookAtGemInMe()
    {
        string result = _lookCommand.Execute(_player, new string[] { "look", "at", "gem", "in",
"inventory" });
        Assert.AreEqual("a gem: A shiny red gem", result);
    }

    [Test]
    public void TestLookAtGemInBag()
    {
        string result = _lookCommand.Execute(_player, new string[] { "look", "at", "gem", "in", "bag"
});
        Assert.AreEqual("another gem: Another shiny gem", result);
    }

    [Test]
    public void TestLookAtGemInNoBag()
    {
        string result = _lookCommand.Execute(_player, new string[] { "look", "at", "gem", "in",
"nonexistentbag" });
        Assert.AreEqual("I cannot find the nonexistentbag", result);
    }
```

```
        [Test]
        public void TestLookAtNoGemInBag()
        {
            Player player = new Player("Player", "A brave adventurer");
            Bag smallBag = new Bag(new string[] { "small bag" }, "small bag", "A small leather bag");

            Item anotherGem = new Item(new string[] { "another gem" }, "another gem", "Another shiny gem");
            player.Inventory.Put(smallBag);
            smallBag.Inventory.Put(anotherGem);

            LookCommand lookCommand = new LookCommand();
            string result = lookCommand.Execute(player, new string[] { "look", "at", "gem", "in", "small bag" });

            Assert.AreEqual("I cannot find the gem in  small bag", result);
        }


        [Test]
        public void TestInvalidLook()
        {
            string result = _lookCommand.Execute(_player, new string[] { "hello" });
            Assert.AreEqual("I don't know how to look like that", result);
        }
    }
}
```

Location testing:

```
using NUnit.Framework;
using SwinAdventure;
```

```
namespace SwinAdventure.Tests
{
    [TestFixture]
    public class LocationTests
    {
        private Location _location;
        private Item _item;

        [SetUp]
        public void SetUp()
        {
            _location = new Location(new string[] { "cave" }, "Dark Cave", "A dimly lit cave.");
            _item = new Item(new string[] { "gem" }, "Shiny Gem", "A sparkling gem lies here.");
            _location.Inventory.Put(_item);
        }

        [Test]
        public void LocationCanIdentifyItself()
        {
            Assert.AreEqual(_location, _location.Locate("cave"));
        }

        [Test]
        public void LocationCanLocateItemInInventory()
        {
            Assert.AreEqual(_item, _location.Locate("gem"));
        }

        [Test]
        public void LocationCannotLocateNonexistentItem()
        {
```

```
            Assert.IsNull(_location.Locate("sword"));
        }
    }
}
```