Selected files

15 printable files

```
Week_9\9.2\SwinAdventure\Bag.cs
Week_9\9.2\SwinAdventure\Command.cs
Week_9\9.2\SwinAdventure\GameObject.cs
Week_9\9.2\SwinAdventure\IdentifiableObject.cs
Week_9\9.2\SwinAdventure\IHaveInventory.cs
Week_9\9.2\SwinAdventure\Inventory.cs
Week_9\9.2\SwinAdventure\Item.cs
Week_9\9.2\SwinAdventure\Location.cs
Week_9\9.2\SwinAdventure\LookCommand.cs
Week_9\9.2\SwinAdventure\MoveCommand.cs
Week_9\9.2\SwinAdventure\Path.cs
Week_9\9.2\SwinAdventure\Player.cs
Week_9\9.2\SwinAdventure\Program.cs
Week_9\9.2\TestPath\TestMove.cs
Week_9\9.2\TestPath\TestPath.cs
```

Week_9\9.2\SwinAdventure\Bag.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Security.Cryptography;
using System.Threading.Tasks;

namespace SwinAdventure
{
    public class Bag : Item, IHaveInventory
    {
        private Inventory _inventory;
        public Bag(string[] idents, string name, string desc) : base(idents, name, desc)
        {
            _inventory = new Inventory();
        }

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

            if (_inventory.HasItem(id))
                return _inventory.Fetch(id);

            return null;
        }

        public Inventory Inventory => _inventory;
        public override string FullDescription
        {
            get
            {
                return $"In the {Name} you can see:\n{_inventory.ItemList}";
            }
        }
    }
}
```

Week_9\9.2\SwinAdventure\Command.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace SwinAdventure
{
    public abstract class Command : IdentifiableObject
    {
        public Command(string[] ids) : base(ids)
        {
        }

        public abstract string Execute(Player p, string[] text);
    }
}
```

Week_9\9.2\SwinAdventure\GameObject.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace SwinAdventure
{
    public abstract class GameObject : IdentifiableObject
    {
        private string _description, _name;

        public GameObject(string[] idents, string name, string desc) : base(idents)
        {
            _name = name;
            _description = desc;
        }

        public string Name => _name;

        public string ShortDescription => $"{Name} ({FirstId})";

        public virtual string FullDescription => _description;
    }
}
```

Week_9\9.2\SwinAdventure\IdentifiableObject.cs

```csharp
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace SwinAdventure
{

    public class IdentifiableObject
    {
        private List<string> _identifiers = new List<string>();

        public IdentifiableObject(string[] idents)
        {
            foreach (string id in idents)
            {
                AddIdentifier(id);
            }
        }

        public bool AreYou(string id)
        {
            return _identifiers.Contains(id.ToLower());
        }

        public string FirstId
        {
            get
            {
                if (_identifiers.Count > 0)
                {
                    return _identifiers[0];
                }

                return "";
            }
        }

        public void AddIdentifier(string id)
        {
            _identifiers.Add(id.ToLower());
        }

        public void PrivilegeEscalation(string pin)
        {
            if (pin != "4794")
                return;

            if (_identifiers.Count == 0)
            {
                AddIdentifier("12");
            }
            else
            {
                _identifiers[0] = "12";
            }
        }
    }
}
```

Week_9\9.2\SwinAdventure\IHaveInventory.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace SwinAdventure
{
    public interface IHaveInventory
    {
        public GameObject? Locate(string id);
        public string Name { get; }
    }
}
```

Week_9\9.2\SwinAdventure\Inventory.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace SwinAdventure
{
    public class Inventory : GameObject
    {
        private List<Item> _items;

        public Inventory() : base(new string[] { "inventory" }, "inventory", "The player's inventory")
        {
            _items = new List<Item>();
        }

        public string ItemList
        {
            get
            {
                if (_items.Count == 0)
                {
                    return "\tNothing here!";
                }

                List<string> itemsDesc = new List<string>();
                foreach (Item item in _items)
                {
                    itemsDesc.Add("\t" + item.ShortDescription);
                }
                return string.Join("\n", itemsDesc);
            }
        }

        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)
        {
            foreach (Item item in _items)
            {
                if (item.AreYou(id))
                {
                    _items.Remove(item);
                    return item;
                }
            }
            return null;
        }

        public Item? Fetch(string id)
        {
            foreach (Item item in _items)
            {
                if (item.AreYou(id))
                {
                    return item;
                }
            }
            return null;
        }
    }
}
```

Week_9\9.2\SwinAdventure\Item.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace SwinAdventure
{
    public class Item : GameObject
    {
        public Item(string[] idents, string name, string desc) : base(idents, name, desc)
        {
        }
    }
}
```

Week_9\9.2\SwinAdventure\Location.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace SwinAdventure
{
    public class Location : GameObject, IHaveInventory
    {
        private Inventory _inventory;
        private List<Path> _paths;

        public Location(string name, string desc) : base(new string[] { "location", "room", "here" }, name, desc)
        {
            _inventory = new Inventory();
            _paths = new List<Path>();
        }

        public Location(string name, string desc, List<Path> paths) : this(name, desc)
        {
            _paths = paths;
        }

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

            foreach (Path path in _paths)
            {
                if (path.AreYou(id))
                {
                    return path;
                }
            }

            return _inventory.Fetch(id);
        }

        public override string FullDescription
        {
            get
            {
                return $"You are in the {Name}.\n" +
                        $"{base.FullDescription}\n" +
                        $"{PathList}\n" +
                        $"In this location, you can see:\n{_inventory.ItemList}";
            }
        }

        public string PathList
        {
            get
            {
                if (_paths.Count == 0)
                {
                    return "There are no paths to other locations";
                }

                string paths = "There are exits to ";

                for (int i = 0; i < _paths.Count; i++)
                {
                    paths += _paths[i].Name;
                    if (i < _paths.Count - 1)
                    {
                        paths += ", ";
                    }
                }

                return paths;
            }
        }

        public Inventory Inventory
        {
            get
            {
                return _inventory;
            }
        }

        public void AddPath(Path path)
        {
            _paths.Add(path);
        }
    }
}
```

Week_9\9.2\SwinAdventure\LookCommand.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

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

        public override string Execute(Player p, string[] text)
        {
            // If text length is not 1,3,5
            if (text.Length != 1 && text.Length != 3 && text.Length != 5)
                return "I don't know how to look like that";

            if (text[0] != "look")
                return "Error in look input";

            if (text.Length != 1 && text[1] != "at")
                return "What do you want to look at?";

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

            string containerId = "";
            string itemId = "";
            switch (text.Length)
            {
                case 1:
                    containerId = "location";
                    itemId = "location";
                    break;
                case 3:
                    containerId = p.FirstId;
                    itemId = text[2];
                    break;
                case 5:
                    containerId = text[4];
                    itemId = text[2];
                    break;
            }

            IHaveInventory? container = FetchContainer(p, containerId);
            if (container == null)
                return $"I can't find the {containerId}";

            return LookAtIn(itemId, container);
        }

        public IHaveInventory? FetchContainer(Player p, string containerId)
        {
            return p.Locate(containerId) as IHaveInventory;
        }

        public string LookAtIn(string thingId, IHaveInventory container)
        {
            GameObject? thing = container.Locate(thingId);
            if (thing == null)
                return $"I can't find the {thingId}";

            return thing.FullDescription;
        }
    }
}
```

Week_9\9.2\SwinAdventure\MoveCommand.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace SwinAdventure
{
    public class MoveCommand : Command
    {
        public MoveCommand() : base(new string[] { "move", "go" })
        {
        }

        public override string Execute(Player p, string[] text)
        {
            if (text.Length != 2 && text.Length != 3)
                return "I don't know how to move like that";

            if (text[0] != "move" && text[0] != "go")
                return "Error in move input";

            if (text.Length == 3 && text[1] != "to")
                return "Where do you want to go?";

            string destinationId = "";
            switch (text.Length)
            {
                case 2:
                    destinationId = text[1];
                    break;
                case 3:
                    destinationId = text[2];
                    break;
            }

            Path? path = p!.Locate(destinationId) as Path;
            if (path == null)
                return $"I can't find the path to {destinationId}";

            if (path.IsBlocked)
                return $"The path to {destinationId} is blocked";

            p.Location = path.Destination;
            return $"You have moved to {path.Destination.Name}";
        }
    }
}
```

Week_9\9.2\SwinAdventure\Path.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace SwinAdventure
{
    public class Path : GameObject
    {
        private Location _source, _destination;
        private bool _isBlocked;

        public Path(string[] ids, string name, string desc, Location source, Location
destination) : base(ids, name, desc)
        {
            _source = source;
            _destination = destination;
            _isBlocked = false;
        }

        public bool IsBlocked
        {
            get
            {
                return _isBlocked;
            }
            set
            {
                _isBlocked = value;
            }
        }

        public Location Source
        {
            get
            {
                return _source;
            }
        }

        public Location Destination
        {
            get
            {
                return _destination;
            }
        }
    }
}
```

Week_9\9.2\SwinAdventure\Player.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

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

        public Player(string name, string desc) : base(new string[] { "me", "inventory" }, name,
desc)
```

```
        {
            _inventory = new Inventory();
        }

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

            GameObject? obj = _inventory.Fetch(id);
            if (obj != null)
                return obj;

            if (_location != null)
                return _location.Locate(id);

            return null;
        }

        public override string FullDescription
        {
            get
            {
                return $"You are {Name}, {base.FullDescription}\n" +
                       $"You are carrying:\n{_inventory.ItemList}";
            }
        }

        public Inventory Inventory => _inventory;
        public Location? Location { get => _location; set => _location = value; }
    }
}
```

Week_9\9.2\SwinAdventure\Program.cs

```
namespace SwinAdventure
{
    class Program
    {
        static void Main()
        {
            string playerName, playerDesc;
            while (true)
            {
                Console.Write("Enter player name: ");
                playerName = Console.ReadLine() ?? string.Empty;
                Console.Write("Enter player description: ");
                playerDesc = Console.ReadLine() ?? string.Empty;
                if (string.IsNullOrEmpty(playerName) || string.IsNullOrEmpty(playerDesc))
                {
                    Console.WriteLine("Player name and description cannot be empty.");
```

```csharp
                }
                else
                {
                    break;
                }
            }
            Player player = new Player(playerName, playerDesc);

            // Create items and put them in the player's inventory
            Item item1 = new Item(new string[] { "shovel" }, "a shovel", "a wooden shovel");
            Item item2 = new Item(new string[] { "sword" }, "a sword", "a steel sword");
            player.Inventory.Put(item1);
            player.Inventory.Put(item2);

            // Create a bag and put it in the player's inventory
            Bag bag = new Bag(new string[] { "bag" }, "a bag", "a leather bag");
            player.Inventory.Put(bag);

            // Create items and put them in the bag's inventory
            Item item3 = new Item(new string[] { "coin" }, "a coin", "a shiny coin");
            bag.Inventory.Put(item3);

            // Create location and put some items in its inventory
            Location location = new Location("forest", "A dark forest with tall trees");
            Item item4 = new Item(new string[] { "rock" }, "a rock", "a big rock");
            Item item5 = new Item(new string[] { "flower" }, "a flower", "a red flower");
            location.Inventory.Put(item4);
            location.Inventory.Put(item5);

            // Create another location and a path between the two locations
            Location location2 = new Location("cave", "A dark cave with bats");
            Path path = new Path(new string[] { "north" }, "north", "a path from forest to cave", location, location2);
            location.AddPath(path);

            // Set player's location
            player.Location = location;

            LookCommand look = new LookCommand();
            MoveCommand move = new MoveCommand();

            while (true)
            {
                Console.Write("> ");
                string command = Console.ReadLine() ?? string.Empty;

                if (string.IsNullOrEmpty(command))
                    continue;
                if (command == "quit")
                    break;

                string response;
                if (command.StartsWith("move") || command.StartsWith("go"))
                {
                    response = move.Execute(player, command.Split(" "));
                    Console.WriteLine(response);
                    Console.WriteLine();
                    continue;
                }

                response = look.Execute(player, command.Split(" "));
                Console.WriteLine(response);
                Console.WriteLine();
            }
        }
    }
}
```

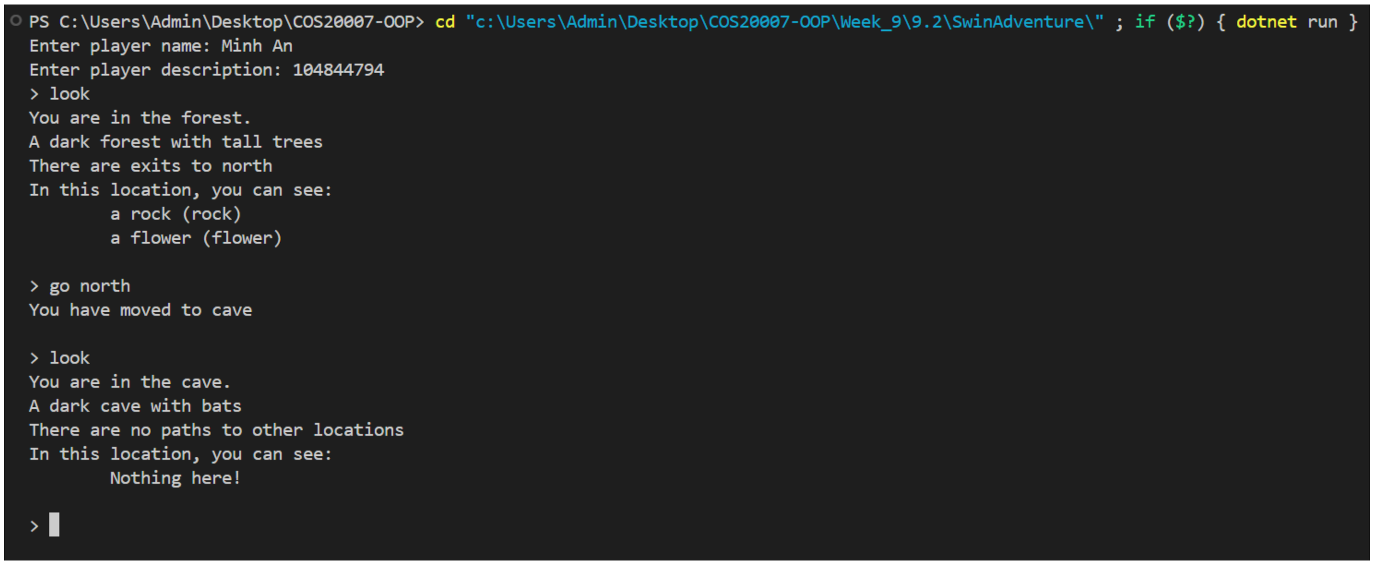
Week_9\9.2\TestPath\TestMove.cs

```
using NUnit.Framework;
using SwinAdventure;
using Path = SwinAdventure.Path;

namespace TestMoveCommand
{
    public class TestMoveCommand
    {
        private Player _player;
        private Location _location1;
        private Location _location2;
        private Path _path;
        private MoveCommand _moveCommand;

        [SetUp]
        public void Setup()
        {
            _player = new Player("Minh An", "104844794");
            _location1 = new Location("forest", "A dark forest with tall trees");
            _location2 = new Location("cave", "A dark cave with bats");
            _path = new Path(new string[] { "north" }, "north", "a path from forest to cave",
_location1, _location2);
            _location1.AddPath(_path);
            _player.Location = _location1;
            _moveCommand = new MoveCommand();
        }

        [Test]
        public void TestMoveToBlockedPath()
        {
            _path.IsBlocked = true;
            Assert.That(_moveCommand.Execute(_player, new string[] { "go", "north" }), Is.EqualTo("The path to north is blocked"));
        }

        [Test]
        public void TestMoveToNonExistentPath()
        {
            Assert.That(_moveCommand.Execute(_player, new string[] { "go", "south" }), Is.EqualTo("I can't find the path to south"));
        }

        [Test]
        public void TestMoveToDestination()
        {
            Assert.That(_moveCommand.Execute(_player, new string[] { "go", "north" }), Is.EqualTo("You have moved to cave"));
        }

        [Test]
        public void TestInvalidMoveCommand()
        {
            Assert.That(_moveCommand.Execute(_player, new string[] { "move", "north" }), Is.EqualTo("You have moved to cave"));
            Assert.That(_moveCommand.Execute(_player, new string[] { "go", "north", "to" }), Is.EqualTo("Where do you want to go?"));
            Assert.That(_moveCommand.Execute(_player, new string[] { "go", "north", "to", "cave" }), Is.EqualTo("I don't know how to move like that"));
            Assert.That(_moveCommand.Execute(_player, new string[] { "go", "north", "to", "cave", "now" }), Is.EqualTo("I don't know how to move like that"));
        }
    }
}
```

Week_9\9.2\TestPath\TestPath.cs

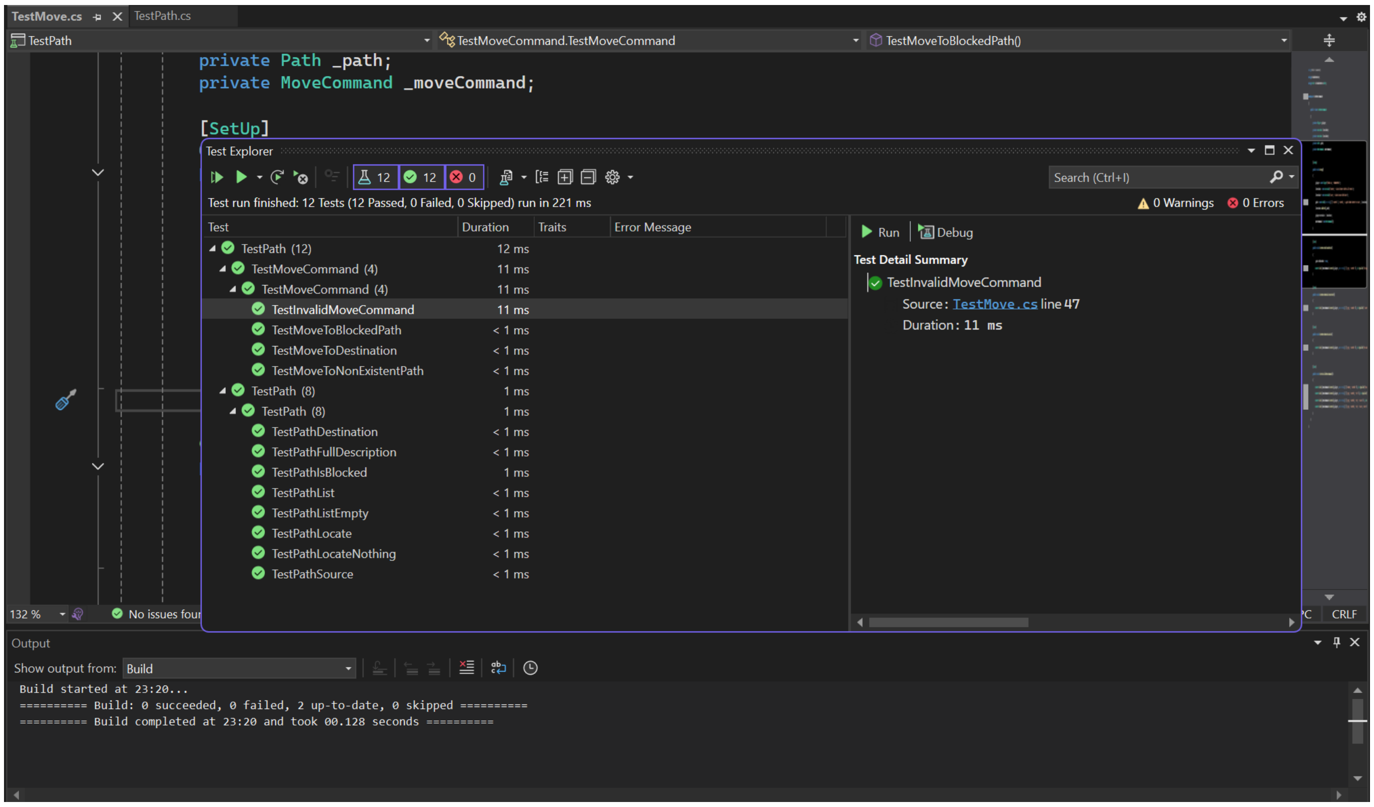
```
using NUnit.Framework;
using SwinAdventure;
using Path = SwinAdventure.Path;

namespace TestPath
{
    public class TestPath
    {
        private Player _player;
        private Location _location1;
        private Location _location2;
        private Path _path;

        [SetUp]
        public void Setup()
        {
            _player = new Player("Minh An", "104844794");
            _location1 = new Location("forest", "A dark forest with tall trees");
            _location2 = new Location("cave", "A dark cave with bats");
            _path = new Path(new string[] { "north" }, "north", "a path from forest to cave",
_location1, _location2);
            _location1.AddPath(_path);
            _player.Location = _location1;
        }

        [Test]
        public void TestPathIsBlocked()
        {
            Assert.That(_path.IsBlocked, Is.False);
            _path.IsBlocked = true;
            Assert.That(_path.IsBlocked, Is.True);
        }

        [Test]
        public void TestPathSource()
        {
            Assert.That(_path.Source, Is.EqualTo(_location1));
        }

        [Test]
        public void TestPathDestination()
        {
            Assert.That(_path.Destination, Is.EqualTo(_location2));
        }

        [Test]
        public void TestPathLocate()
        {
            Assert.That(_player.Locate("north"), Is.EqualTo(_path));
        }

        [Test]
        public void TestPathLocateNothing()
        {
            Assert.That(_player.Locate("south"), Is.Null);
        }

        [Test]
        public void TestPathFullDescription()
        {
            Assert.That(_path.FullDescription, Is.EqualTo("a path from forest to cave"));
        }

        [Test]
        public void TestPathList()
        {
            Assert.That(_location1.PathList, Is.EqualTo("There are exits to north"));
        }

        [Test]
        public void TestPathListEmpty()
        {
            Location location = new Location("desert", "A hot desert with sand dunes");
            Assert.That(location.PathList, Is.EqualTo("There are no paths to other locations"));
        }
    }
}
```

Screenshot of program running:



```
PS C:\Users\Admin\Desktop\COS20007-OOP> cd "c:\Users\Admin\Desktop\COS20007-OOP\Week_9\9.2\SwinAdventure\" ; if ($?) { dotnet run }
Enter player name: Minh An
Enter player description: 104844794
> look
You are in the forest.
A dark forest with tall trees
There are exits to north
In this location, you can see:
        a rock (rock)
        a flower (flower)

> go north
You have moved to cave

> look
You are in the cave.
A dark cave with bats
There are no paths to other locations
In this location, you can see:
        Nothing here!

>
```

Screenshot of test case passed



```
private Path _path;
private MoveCommand _moveCommand;

[SetUp]
```

Test run finished: 12 Tests (12 Passed, 0 Failed, 0 Skipped) run in 221 ms

0 Warnings 0 Errors

Test	Duration	Traits	Error Message
TestPath (12)	12 ms		
TestMoveCommand (4)	11 ms		
TestMoveCommand (4)	11 ms		
TestInvalidMoveCommand	11 ms		
TestMoveToBlockedPath	< 1 ms		
TestMoveToDestination	< 1 ms		
TestMoveToNonExistentPath	< 1 ms		
TestPath (8)	1 ms		
TestPath (8)	1 ms		
TestPathDestination	< 1 ms		
TestPathFullDescription	< 1 ms		
TestPathIsBlocked	1 ms		
TestPathList	< 1 ms		
TestPathListEmpty	< 1 ms		
TestPathLocate	< 1 ms		
TestPathLocateNothing	< 1 ms		
TestPathSource	< 1 ms		

Test Detail Summary

TestInvalidMoveCommand

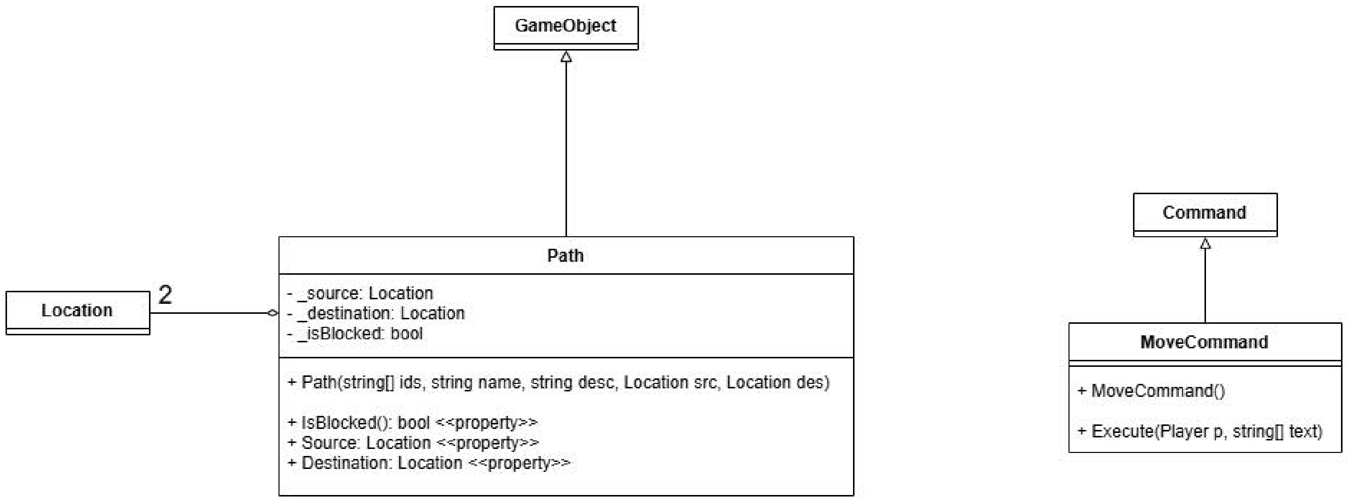
Source: TestMove.cs line 47

Duration: 11 ms

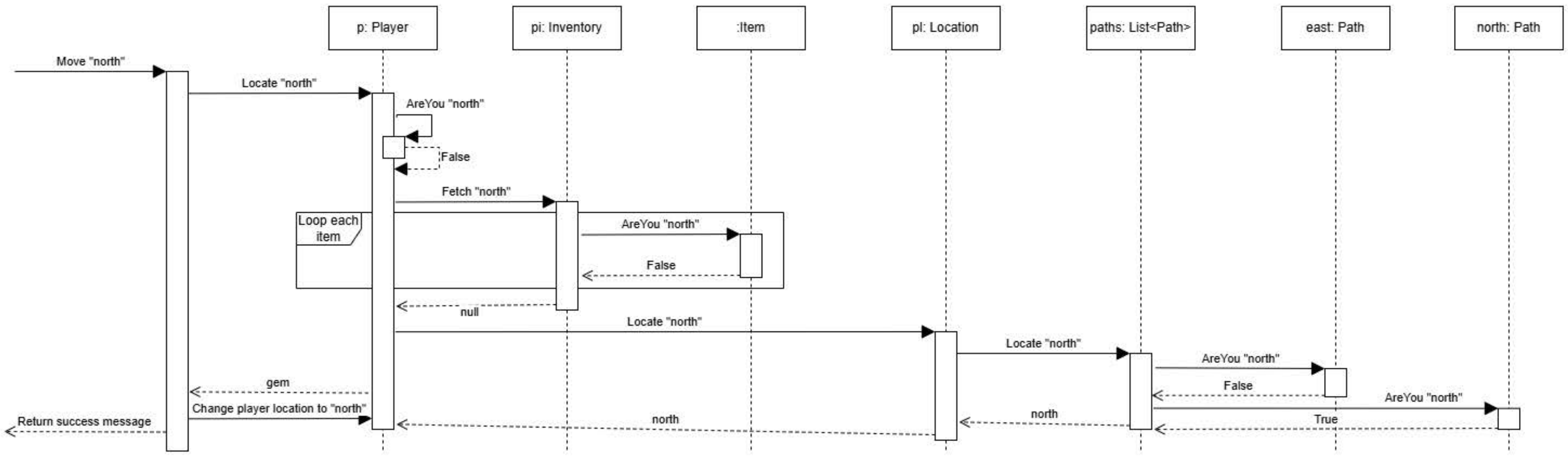
Show output from: Build

```
Build started at 23:20...
========== Build: 0 succeeded, 0 failed, 2 up-to-date, 0 skipped ==========
========== Build completed at 23:20 and took 00.128 seconds ==========
```

UML Diagram


GameObject
Command
Path
Location
2
- _source: Location
- _destination: Location
- _isBlocked: bool
+ Path(string[] ids, string name, string desc, Location src, Location des)
+ IsBlocked(): bool <<property>>
+ Source: Location <<property>>
+ Destination: Location <<property>>
MoveCommand
+ MoveCommand()
+ Execute(Player p, string[] text)

Sequence Diagram


p: Player
pi: Inventory
:Item
pl: Location
paths: List<Path>
east: Path
north: Path
Move "north"
Locate "north"
AreYou "north"
False
Fetch "north"
Loop each item
AreYou "north"
False
null
Locate "north"
Locate "north"
AreYou "north"
False
AreYou "north"
True
north
north
gem
Change player location to "north"
Return success message