Path.cs

using Swin\_Adventure;

using System;

using System.Xml.Linq;

namespace Swin\_Adventure

{

public class Path : GameObject

{

private bool \_isBlocked;

private Location \_source, \_destination;

public Path(string[] idents, string name, string desc, Location source, Location destination) : base(idents, name, desc)

{

\_source = source;

\_destination = destination;

\_isBlocked = false;

AddIdentifier("path");

foreach (string s in name.Split(" "))

{

AddIdentifier(s);

}

}

public string Move(Player player)

{

if (\_destination != null)

{

player.Location = \_destination;

}

return $"You have moved to {\_destination.Name}";

}

public Location Source { get { return \_source; } }

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

public bool IsBlocked

{

get { return \_isBlocked; }

set { \_isBlocked = value; }

}

}

}

PathTest.cs

using System;

using System.Numerics;

using System.Reflection.Metadata;

using NUnit.Framework;

using Swin\_Adventure;

namespace SwinAdventureTest

{

public class PathTest

{

Player \_playerTest;

Location \_roomATest;

Location \_roomBTest;

Swin\_Adventure.Path \_pathTest;

Move \_commandTest;

[SetUp]

public void Setup()

{

\_roomATest = new Location(new string[] { "roomA" }, "Room A", "Room A");

\_roomBTest = new Location(new string[] { "roomB" }, "Room B", "Room B");

\_playerTest = new Player("thuan", "dan choi", \_roomATest);

\_pathTest = new Swin\_Adventure.Path(new string[] { "north" }, "Door", "A test door", \_roomATest, \_roomBTest);

\_roomATest.AddPath(\_pathTest);

\_commandTest = new Move();

}

[Test]

public void TestPathLocation()

{

Location \_expected = \_roomBTest;

Location \_actual = \_pathTest.Destination;

Assert.AreEqual(\_expected, \_actual);

}

[Test]

public void TestPathNameFullDescription()

{

string \_expected = "A test door";

string \_actual = \_pathTest.FullDescription;

Assert.AreEqual(\_expected, \_actual);

}

[Test]

public void TestPathNameShortDescription()

{

string \_expected = "Door (north)";

string \_actual = \_pathTest.ShortDescription;

Assert.AreEqual(\_expected, \_actual);

}

[Test]

public void TestPathNotBlocked()

{

bool \_actual = \_pathTest.IsBlocked;

Assert.IsFalse(\_actual);

}

[Test]

public void TestPathBlocked()

{

\_pathTest.IsBlocked = true;

bool \_actual = \_pathTest.IsBlocked;

Assert.IsTrue(\_actual);

}

[Test]

public void TestLocatePathfromLocation()

{

Assert.AreEqual(\_pathTest, \_playerTest.Location.Locate("north"));

}

[Test]

public void TestLocateNoPath()

{

Assert.AreNotEqual(\_pathTest, \_roomATest.Locate("south"));

}

[Test]

public void TestMovePlayer()

{

\_commandTest.Execute(\_playerTest, new string[] { "Move", "north" });

Assert.AreEqual(\_playerTest.Location, \_roomBTest);

}

}

}

MoveCommand.cs

using System;

using Swin\_Adventure;

namespace Swin\_Adventure

{

public class Move : Command

{

public Move() : base(new string[] { "move", "go", "head", "leave" })

{

}

public override string Execute(Player player, string[] text)

{

string \_moveDirection;

if (text.Length > 0 && !AreYou(text[0].ToLower()))

{

return "Invalid move command. You remain in the same location.";

}

switch (text.Length)

{

case 1:

return "Go: ???";

case 2:

\_moveDirection = text[1].ToLower();

break;

case 3:

\_moveDirection = text[2].ToLower();

break;

default:

return "Error in move input.";

}

GameObject \_path = player.Location.Locate(\_moveDirection);

if (\_path != null)

{

if (\_path.GetType() == typeof(Path))

{

return (\_path as Path).Move(player) + ".\r\n\n" + player.Location.FullDescription;

}

else

{

return "Could not find the " + \_path.Name;

}

}

else

{

return "Invalid path identifier. You remain in the same location.";

}

}

}

}

MoveCommandTest.cs

using NUnit.Framework;

using System.Collections.Generic;

using System;

using Swin\_Adventure;

namespace SwinAdventureTest

{

public class MoveTests

{

private Location \_location1;

private Location \_location2;

private Swin\_Adventure.Path \_path;

private Player \_player;

private Move \_moveCommand;

[SetUp]

public void Setup()

{

\_location1 = new Location(new string[] { "location1" }, "Location1", "Location1");

\_location2 = new Location(new string[] { "location2" }, "Location2", "Location2");

\_path = new Swin\_Adventure.Path(new string[] { "path1" }, "a path", "long path", \_location1, \_location2);

\_player = new Player("TestPlayer", "Player 1", \_location1);

\_location1.AddPath(\_path);

\_moveCommand = new Move();

}

[Test]

public void GetPathFromLocation()

{

Assert.AreEqual(\_path, \_player.Location.Locate("path1"));

}

[Test]

public void PathCanMovePlayerToDestination()

{

string result = \_moveCommand.Execute(\_player, new string[] { "move", "path1" });

Assert.AreEqual("You have moved to Location2.\r\n\n" + \_location2.FullDescription, result);

}

[Test]

public void PlayersCanLeaveLocationWithValidPathIdentifier()

{

Move moveCommand = new Move();

string result = moveCommand.Execute(\_player, new string[] { "move", "path1" });

Assert.AreEqual("You have moved to Location2.\r\n\n" + \_location2.FullDescription, result);

}

[Test]

public void PlayersInSameLocationWithInvalidPathIdentifier()

{

Move moveCommand = new Move();

string result = moveCommand.Execute(\_player, new string[] { "move", "invalidpath" });

Assert.AreEqual("Invalid path identifier. You remain in the same location.", result);

Assert.AreEqual(\_location1, \_player.Location);

}

}

}

Location.cs

using System;

using System.Collections.Generic;

using Swin\_Adventure;

namespace Swin\_Adventure

{

public class Location : GameObject, IHaveInventory

{

private Inventory \_inventory;

\_paths = new List<Path>();

public Location(string [] idents,string name, string description) : base(idents, name, description)

{

\_inventory = new Inventory();

}

public Inventory Inventory

{

get { return \_inventory; }

}

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 at the " + Name + ". " + base.FullDescription + "\n" +

"You can see:\n" + Inventory.ItemList;

}

}

public string PathList

{

get

{

string \_pathList = "";

foreach (Path path in \_paths)

{

\_pathList += "\t" + path.ShortDescription + "\n\t" + path.FullDescription;

}

return \_pathList;

}

}

public void AddPath(Path path)

{

\_paths.Add(path);

}

}

}

LocationTest.cs

using NUnit.Framework;

using Swin\_Adventure;

using System;

namespace SwinAdventureTest

{

[TestFixture]

public class LocationTest

{

public Location \_locationTest;

public Item \_itemTest;

public Player \_playerTest;

Location \_roomBTest;

Swin\_Adventure.Path \_pathTest;

Move \_commandTest;

[SetUp]

public void Setup()

{

\_locationTest = new Location(new string[] { "location" }, "forest", "a rain forest");

\_itemTest = new Item(new string[] { "gem" }, "ruby", "a bright red ruby");

\_playerTest = new Player("thuan", "dan choi", \_locationTest);

\_roomBTest = new Location(new string[] { "roomB" }, "Room B", "Room B");

\_pathTest = new Swin\_Adventure.Path(new string[] { "north" }, "Door", "A test door", \_locationTest, \_roomBTest);

\_locationTest.AddPath(\_pathTest);

\_commandTest = new Move();

\_locationTest.Inventory.Put(\_itemTest);

}

[Test]

public void TestLocationIsIdentifiable()

{

Assert.IsTrue(\_locationTest.AreYou("location"));

}

[Test]

public void TestLocationCanLocateItems()

{

Assert.IsTrue(\_locationTest.Locate("gem") == \_itemTest);

Assert.IsTrue(\_locationTest.Inventory.HasItem("gem"));

}

[Test]

public void TestLocationLocateItself()

{

Assert.IsTrue(\_locationTest == \_locationTest.Locate("location"));

}

[Test]

public void TestLocationLocateNothing()

{

Assert.IsNull(\_locationTest.Locate("sword"));

}

[Test]

public void TestLocationFullDescription()

{

Assert.IsTrue(\_locationTest.FullDescription.Contains("a rain forest"));

Assert.IsTrue(\_locationTest.FullDescription.Contains("a ruby (gem)"));

}

[Test]

public void TestLocationHasPath()

{

Assert.AreEqual(\_pathTest, \_playerTest.Location.Locate("north"));

}

[Test]

public void TestLocationHasNoPath()

{

Assert.IsNull(\_locationTest.Locate("south"));

}

[Test]

public void TestLocationPathList()

{

string \_expected = "\tDoor (north)\n\tA test door";

string \_actual = \_locationTest.PathList;

Assert.AreEqual(\_expected, \_actual);

}

[Test]

public void TestMovePlayer()

{

\_commandTest.Execute(\_playerTest, new string[] { "Move", "north" });

Assert.AreEqual(\_playerTest.Location, \_roomBTest);

}

[Test]

public void TestSameLocation()

{

\_commandTest.Execute(\_playerTest, new string[] { "Move", "south" });

Assert.AreEqual(\_playerTest.Location, \_locationTest);

}

}

}

Program.cs

using System;

using Swin\_Adventure;

namespace Swin\_Adventure;

class Program

{

static void Main(string[] args)

{

Console.WriteLine("Welcome to SwinAdventure, designed by Thuan!");

Console.WriteLine("Enter Player Name: ");

string playerName = Console.ReadLine();

Console.WriteLine("Enter your description: ");

string playerDescription = Console.ReadLine();

Item item1 = new Item(new string[] { "weapon" }, "sword", "this is an Excalibur");

Item item2 = new Item(new string[] { "armor" }, "shield", "this is a shield");

Bag bag = new Bag(new string[] { "bag" }, "bag", "This is a bag.");

Item itemInBag = new Item(new string[] { "gem" }, "ruby", "This is a beautiful gem");

Location location1 = new Location(new string[] { "roomA" }, "Room A", "You are in Room A.");

Location location2 = new Location(new string[] { "roomB" }, "Room B", "You are in Room B.");

Path path = new Path(new string[] { "north" }, "Door", "A test door", location1, location2);

Player player = new Player(playerName, playerDescription, location1);

player.Inventory.Put(item1);

player.Inventory.Put(item2);

player.Inventory.Put(bag);

bag.Inventory.Put(itemInBag);

location1.Inventory.Put(itemInBag);

location1.AddPath(path);

Move moveCommand = new Move();

LookCommand lookCommand = new LookCommand();

while (true)

{

Console.WriteLine("\nItem at this location:");

Console.WriteLine($"\n{player.Location.Inventory.ItemList}");

Console.WriteLine("\nItem in Inventory:");

Console.WriteLine($"{player.Inventory.ItemList}");

Console.WriteLine("\nCommands: look at <item>, look at <item> in <container>, quit");

Console.WriteLine("Commands: move <path>, exit");

Console.Write("Enter a command: ");

string command = Console.ReadLine().ToLower();

string[] commandParts = command.Split(' ');

switch (commandParts[0])

{

case "look":

string result = lookCommand.Execute(player, commandParts);

Console.WriteLine("\n" + result);

break;

case "move":

if (commandParts.Length >= 2)

{

string direction = commandParts[1];

Console.WriteLine(moveCommand.Execute(player, new string[] { "move", direction }));

}

else

{

Console.WriteLine(moveCommand.Execute(player, new string[] { "move" }));

}

break;

case "exit":

return;

default:

Console.WriteLine("Invalid command. Type 'move <direction>' or 'exit' to quit.");

break;

}

}

}

}