Inventory.cs

using System;

using System.Collections.Generic;

namespace Iteration7

{

public class Inventory

{

private readonly List<Item> \_items;

public Inventory()

{

\_items = [];

}

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 Fetch(string id)

{

foreach (Item item in \_items)

{

if (item.AreYou(id))

{

return item;

}

}

return null;

}

public Item Take(string id)

{

Item takeitem = Fetch(id);

\_items.Remove(takeitem);

return takeitem;

}

public string ItemList

{

get

{

string list = "";

foreach (Item item in \_items)

{

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

}

return list;

}

}

}

}

Path.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Iteration7

{

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

{

private readonly Location \_source = source;

private readonly Location \_destination = destination;

public Location Source

{

get

{

return \_source;

}

}

public Location Destination

{

get

{

return \_destination;

}

}

}

}

GameObject.cs

using System;

using System.Collections.Generic;

using System.Text;

namespace Iteration7

{

public class GameObject(string[] idents, string name, string desc) : IdentifiableObject(idents)

{

private readonly string \_description = desc;

private readonly string \_name = name;

public string Name

{

get

{

return \_name;

}

}

public string ShortDescription

{

get

{

return "a " + \_name + " " + "(" + FirstID + ")";

}

}

public virtual string FullDescription

{

get

{

return \_description;

}

}

}

}

Bag.cs

using System;

using System.Xml.Linq;

namespace Iteration7

{

public class Bag(string[] idents, string name, string desc) : Item(idents, name, desc), IHaveInventory

{

private readonly Inventory \_inventory = new();

public Inventory Inventory

{

get

{

return \_inventory;

}

}

public GameObject Locate(string id)

{

if (this.AreYou(id))

{

return this;

}

else if (\_inventory.HasItem(id))

{

return \_inventory.Fetch(id);

}

return null;

}

public override string FullDescription

{

get

{

string InventoryDescription = "In the " + Name + " you can see:\n";

InventoryDescription += \_inventory.ItemList;

return InventoryDescription;

}

}

}

}

Location.cs

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Iteration7

{

public class Location(string[] idents, string name, string desc) : GameObject(idents, name, desc), IHaveInventory

{

readonly Inventory \_inventory = new();

readonly List<Path> \_paths = [];

public Location(string[] idents, string name, string desc, List<Path> paths) : this(idents, name, desc)

{

\_paths = paths;

}

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

{

if (\_inventory != null)

{

return $"{base.FullDescription}\nItems available:\n{\_inventory.ItemList}";

}

return "There are no items here.";

}

}

public void AddPath(Path path)

{

\_paths.Add(path);

}

public string PathList

{

get

{

if (\_paths.Count == 0)

{

return "\nThere are no exits.";

}

else

{

string list = "\nThere are exits to the ";

foreach (Path path in \_paths)

{

list += path.FirstID + ", ";

}

list = list.TrimEnd(',', ' ') + ".";

return list;

}

}

}

}

}

Players.cs

using System;

using System.Collections;

using System.Collections.Generic;

using System.Linq;

using System.Xml.Linq;

namespace Iteration7

{

public class Player(string name, string desc) : GameObject(["me", "inventory"], name, desc), IHaveInventory

{

private readonly Inventory \_inventory = new();

private Location \_location;

public GameObject Locate(string id)

{

if (AreYou(id))

{

return this;

}

GameObject obj = \_inventory.Fetch(id);

if (obj != null)

{

return obj;

}

if (\_location != null)

{

obj = \_location.Locate(id);

return obj;

}

else

{

return null;

}

}

public override string FullDescription

{

get

{

return "You are " + Name + ", a " + base.FullDescription + ".\nYou are carrying:\n" + \_inventory.ItemList;

}

}

public Inventory Inventory

{

get

{

return \_inventory;

}

}

public Location Location

{

get

{

return \_location;

}

set

{

\_location = value;

}

}

public void Move(Path path)

{

if (path.Destination != null)

{

\_location = path.Destination;

}

}

}

}

Move.cs

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Iteration7

{

public class Move : Command

{

public Move() : base(["move", "go", "head", "leave"])

{

}

private static bool IsMoveCommand(string command)

{

return command == "move" || command == "go" || command == "head" || command == "leave";

}

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

{

string location;

string error = "Where do you want to go?";

string error1 = "Error in move command.";

if (text.Length < 2 && IsMoveCommand(text[0].ToLower()))

{

return error;

}

if (text.Length == 2 && IsMoveCommand(text[0].ToLower()))

{

location = text[1].ToLower();

}

else

{

return error1;

}

return MoveTo(location, p);

}

private static string MoveTo(string location, Player p)

{

GameObject path = p.Location.Locate(location);

string nopath = "The exit is not available.";

if (path == null)

{

return nopath;

}

else

{

p.Move((Path)path);

return $"You have moved {path.FirstID} through a {path.Name} to {p.Location.Name}, {p.Location.FullDescription}{p.Location.PathList}";

}

}

}

}

Look.cs

using System;

namespace Iteration7

{

public class Look : Command

{

public Look() : base(["look"])

{

}

private static IHaveInventory? FetchContainer(Player p, string containerId)

{

return p.Locate(containerId) as IHaveInventory;

}

private static string LookAtIn(string thingId, IHaveInventory container)

{

GameObject foundItem = container.Locate(thingId);

if (foundItem == null)

{

if (container == container.Locate("inventory"))

{

return $"I can't find {thingId}";

}

else

{

return $"I can't find {thingId} in the {container.Name}";

}

}

return foundItem.FullDescription;

}

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

{

IHaveInventory? container = p;

string error = "I don't know how to look like that";

string error1 = "Error in look input";

string error2 = "What do you want to look at?";

string error3 = "What do you want to look in?";

if (text.Length == 1 && text[0].Equals("look", StringComparison.CurrentCultureIgnoreCase))

{

return $"You are in the {p.Location.Name}, {p.Location.FullDescription}{p.Location.PathList}";

}

if (text.Length != 3 && text.Length != 5)

{

return error;

}

if (!text[0].Equals("look", StringComparison.CurrentCultureIgnoreCase))

{

return error1;

}

if (!text[1].Equals("at", StringComparison.CurrentCultureIgnoreCase))

{

return error2;

}

if (text.Length == 5)

{

if (!text[3].Equals("in", StringComparison.CurrentCultureIgnoreCase))

return error3;

container = FetchContainer(p, text[4]);

if (container == null)

return $"I can't find the {text[4]}";

}

return LookAtIn(text[2], container);

}

}

}

Items.cs

using System;

using System.Collections.Generic;

using System.Text;

namespace Iteration7

{

public class Item(string[] idents, string name, string desc) : GameObject(idents, name, desc)

{

}

}

IhaveInventory.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Iteration7

{

interface IHaveInventory

{

GameObject Locate(string id);

string Name

{

get;

}

}

}

IdentifiableObject.cs

using System;

using System.Collections.Generic;

namespace Iteration7

{

public class IdentifiableObject

{

private readonly List<string> \_idents = [];

public IdentifiableObject(string[] idents)

{

foreach (string s in idents)

{

AddIdentifier(s);

}

}

public bool AreYou(string id)

{

return \_idents.Contains(id.ToLower());

}

public string FirstID

{

get

{

if (\_idents.Count == 0)

{

return "";

}

else

{

return \_idents[0];

}

}

}

public void AddIdentifier(string id)

{

\_idents.Add(id.ToLower());

}

}

}

Command.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Iteration7

{

public abstract class Command(string[] ids) : IdentifiableObject(ids)

{

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

}

}

Program.cs

using System;

using System.IO;

namespace Iteration7

{

public class Interface

{

static void Main(string[] args)

{

Player player;

Bag bag;

Bag backpack;

Item sword;

Item shield;

Item potion;

Item gem;

Location garage;

Location bedroom;

Location gamingroom;

Location livingroom;

Item monitor;

Item computer;

Item phone;

Command lookcommand;

Command movecommand;

string input = "";

Console.WriteLine("Press Q to Exit \n");

Console.WriteLine("What is your name?");

string name = Console.ReadLine();

Console.WriteLine($"Hi {name}, What is your occupation?");

string description = Console.ReadLine();

player = new Player(name, description);

bedroom = new(["bedroom"], "private bedroom", "a room for private stuff");

player.Location = bedroom;

Console.WriteLine($"You are {name}, a {description}. Welcome to the {player.Location.Name}");

bag = new Bag(["bag"], "leather bag", "a light bag, suitable for short trips");

backpack = new Bag(["backpack"], "fabric backpack", "a medium-sized backpack, suitable for abroad travelling");

gem = new Item(["gem"], "gem", "A gem that could be used to trade items.");

sword = new Item(["sword"], "diamond sword", "a diamond sword which has not broken once");

shield = new Item(["shield"], "gold shield", "a gold shield that lasts a lifetime");

potion = new Item(["potion"], "healing potion", "a healing potion which is needed for adventurers");

garage = new Location(["garage"], "big garage", "a room where items are stored");

Path bedroomTogarage = new(["east"], "rolling door", "Walk through rolling door", bedroom, garage);

Path garageTobedroom = new(["west"], "rolling door", "Walk through rolling door", garage, bedroom);

bedroom.AddPath(bedroomTogarage);

garage.AddPath(garageTobedroom);

monitor = new Item(["monitor"], "new monitor", "a brand new monitor");

computer = new Item(["computer"], "public computer", "a computer which is suitable for students");

phone = new Item(["phone"], "mobile phone", "a phone that is recently sold");

Item tablet = new (["tablet"], "IMac", "an expensive tablet");

Item mouse = new(["mouse"], "wireless mouse", "a mouse that is bought on EBay");

Item TV = new(["TV"], "Samsung TV", "a TV that hasn't been used much");

garage.Inventory.Put(phone);

player.Inventory.Put(computer);

lookcommand = new Look();

movecommand = new Move();

player.Inventory.Put(sword);

player.Inventory.Put(shield);

player.Inventory.Put(bag);

player.Inventory.Put(potion);

player.Inventory.Put(backpack);

bag.Inventory.Put(potion);

backpack.Inventory.Put(sword);

backpack.Inventory.Put(bag);

gamingroom = new(["gamingroom"], "large gamingroom", "a room for relaxation");

Path bedroomTogamingroom = new(["north"], "door", "Go through door", bedroom, gamingroom);

Path gamingroomTobedroom = new(["south"], "door", "Go through door", gamingroom, bedroom);

bedroom.AddPath(bedroomTogamingroom);

gamingroom.AddPath(gamingroomTobedroom);

livingroom = new(["livingroom"], "family livingroom", "a room for family talk");

Path bedroomTolivingroom = new(["down"], "door", "Go through door", bedroom, livingroom);

Path livingroomTobedroom = new(["up"], "door", "Go through door", livingroom, bedroom);

bedroom.AddPath(bedroomTolivingroom);

livingroom.AddPath(livingroomTobedroom);

gamingroom.Inventory.Put(monitor);

gamingroom.Inventory.Put(computer);

bedroom.Inventory.Put(gem);

bedroom.Inventory.Put(mouse);

livingroom.Inventory.Put(tablet);

livingroom.Inventory.Put(TV);

while (input != "q")

{

Console.Write("Command: \n");

input = Console.ReadLine().Trim();

if (input == "q")

{

break;

}

List<string> checkmovecommand = ["move", "go", "head", "leave"];

List<string> checkcommand = ["move", "go", "head", "leave", "look"];

string error = "Error in command input.";

if (input.Split()[0].Equals("look", StringComparison.CurrentCultureIgnoreCase))

{

Console.WriteLine(lookcommand.Execute(player, input.Split()));

}

if (checkmovecommand.Contains(input.Split()[0].ToLower()))

{

Console.WriteLine(movecommand.Execute(player, input.Split()));

}

if (!checkcommand.Contains(input.Split()[0].ToLower()))

{

Console.WriteLine(error);

}

}

}

}

}

InventoryTest.cs

using System;

using System.Collections.Generic;

using NUnit.Framework;

namespace Iteration7

{

[TestFixture]

public class TestInventory

{

Inventory inventory;

Item sword;

Item shield;

Item potion;

[SetUp]

public void SetUp()

{

inventory = new Inventory();

sword = new Item(["sword"], "diamond sword", "a diamond sword which has not broken once");

shield = new Item(["shield"], "gold shield", "a gold shield that lasts a lifetime");

potion = new Item(["potion"], "healing potion", "a healing potion which is needed for the adventurers");

inventory.Put(sword);

inventory.Put(shield);

}

[Test]

public void TestFoundItem()

{

Assert.Multiple(() =>

{

Assert.That(inventory.HasItem("sword"), Is.True);

Assert.That(inventory.HasItem("shield"), Is.True);

});

}

[Test]

public void TestNoItemFound()

{

Assert.That(inventory.HasItem("potion"), Is.False);

}

[Test]

public void TestFecthItem()

{

Assert.Multiple(() =>

{

Assert.That(inventory.Fetch("sword"), Is.EqualTo(sword));

Assert.That(inventory.HasItem("sword"), Is.True);

});

}

[Test]

public void TestTakeItem()

{

Assert.Multiple(() =>

{

Assert.That(inventory.Take("sword"), Is.EqualTo(sword));

Assert.That(inventory.HasItem("sword"), Is.False);

Assert.That(inventory.HasItem("shield"), Is.True);

Assert.That(inventory.HasItem("potion"), Is.False);

});

}

[Test]

public void TestItemList()

{

Assert.That(inventory.ItemList, Is.EqualTo("\ta diamond sword (sword)\n\ta gold shield (shield)\n"));

}

}

}

PlayersTest.cs

using System;

using System.Collections.Generic;

using NUnit.Framework;

namespace Iteration7

{

[TestFixture]

public class TestPlayer

{

Player player;

Item sword;

Item shield;

[SetUp]

public void Setup()

{

player = new Player("ruchan", "member of a chess club");

sword = new Item(["sword"], "diamond sword", "a diamond sword which has not broken once");

shield = new Item(["shield"], "gold shield", "a gold shield that lasts a lifetime");

player.Inventory.Put(sword);

player.Inventory.Put(shield);

}

[Test]

public void TestPLayerIsIdentifiable()

{

Assert.Multiple(() =>

{

Assert.That(player.AreYou("me"), Is.True, "True");

Assert.That(player.AreYou("inventory"), Is.True, "True");

});

}

[Test]

public void TestPlayerLocatesItems()

{

var result = false;

var itemLocated = player.Locate("sword");

if (sword == itemLocated)

{

result = true;

}

Assert.That(result, Is.True);

\_ = player.Locate("shield");

if (shield == itemLocated)

{

result = true;

}

Assert.That(result, Is.True);

}

[Test]

public void TestPlayerLocatesItself()

{

Assert.Multiple(() =>

{

Assert.That(player.Locate("me"), Is.EqualTo(player));

Assert.That(player.Locate("inventory"), Is.EqualTo(player));

});

}

[Test]

public void TestPlayerLocatesNothing()

{

Assert.That(player.Locate("plate"), Is.EqualTo(null));

}

[Test]

public void TestPlayerFullDescription()

{

Assert.That(player.FullDescription, Is.EqualTo("You are ruchan, a member of a chess club.\nYou are carrying:\n\ta diamond sword (sword)\n\ta gold shield (shield)\n"));

}

}

}

LookTest.cs

using System;

using System.ComponentModel;

using System.Linq;

namespace Iteration7

{

[TestFixture]

public class TestLook

{

Look look;

Player player;

Bag bag;

Item sword;

Item shield;

Item potion;

[SetUp]

public void SetUp()

{

look = new Look();

player = new Player("ruchan", "a member of a chess club");

bag = new Bag(["bag"], "leather bag", "a light bag, suitable for short trips");

sword = new Item(["sword"], "diamond sword", "a diamond sword which has not broken once");

shield = new Item(["shield"], "gold shield", "a gold shield that lasts a lifetime");

potion = new Item(["potion"], "healing potion", "a healing potion which is needed for the adventurers");

}

[Test]

public void TestLookAtMe()

{

Assert.That(look.Execute(player, ["look", "at", "me"]), Is.EqualTo(player.FullDescription));

}

[Test]

public void TestLookAtSword()

{

player.Inventory.Put(sword);

Assert.That(look.Execute(player, ["look", "at", "sword"]), Is.EqualTo(sword.FullDescription));

}

[Test]

public void TestLookAtUnknownItems()

{

Assert.That(look.Execute(player, ["look", "at", "plate"]), Is.EqualTo($"I can't find plate"));

}

[Test]

public void TestLookAtSwordInMe()

{

player.Inventory.Put(sword);

Assert.That(look.Execute(player, ["look", "at", "sword", "in", "me"]), Is.EqualTo(sword.FullDescription));

}

[Test]

public void TestLookAtSwordInBag()

{

bag.Inventory.Put(sword);

bag.Inventory.Put(shield);

player.Inventory.Put(bag);

Assert.That(look.Execute(player, ["look", "at", "sword", "in", "bag"]), Is.EqualTo(sword.FullDescription));

}

[Test]

public void TestLookAtPotionInNoBag()

{

bag.Inventory.Put(potion);

Assert.That(look.Execute(player, ["look", "at", "potion", "in", "bag"]), Is.EqualTo("I can't find the bag"));

}

[Test]

public void TestLookAtNoShieldInBag()

{

bag.Inventory.Put(sword);

player.Inventory.Put(bag);

Assert.Multiple(() =>

{

Assert.That(look.Execute(player, ["look", "at", "shield", "in", "bag"]), Is.EqualTo("I can't find shield in the leather bag"));

Assert.That(look.Execute(player, ["look", "at", "potion", "in", "bag"]), Is.EqualTo("I can't find potion in the leather bag"));

});

}

[Test]

public void TestInvalidLook()

{

Assert.Multiple(() =>

{

Assert.That(look.Execute(player, ["look", "down"]), Is.EqualTo("I don't know how to look like that"));

Assert.That(look.Execute(player, ["stare", "at", "plate"]), Is.EqualTo("Error in look input"));

Assert.That(look.Execute(player, ["look", "at", "potion", "on", "bag"]), Is.EqualTo("What do you want to look in?"));

Assert.That(look.Execute(player, ["look", "for", "shield"]), Is.EqualTo("What do you want to look at?"));

});

}

}

}

LocationTest.cs

using Iteration7;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Iteration7

{

[TestFixture]

public class LocationTest

{

Player player;

Location location;

Item sword;

[SetUp]

public void SetUp()

{

player = new Player("ruchan", "member of a chess club");

sword = new Item(["sword"], "diamond sword", "a diamond sword which has not broken once");

location = new Location(["garage"], "big garage", "a place where you store stuff");

}

[Test]

public void TestLocationIdentifyItself()

{

GameObject result = location.Locate("garage");

Assert.That(result, Is.EqualTo(location));

}

[Test]

public void TestLocationLocateItemTheyHave()

{

location.Inventory.Put(sword);

GameObject expected = sword;

GameObject actual = location.Locate("sword");

Assert.That(actual, Is.EqualTo(expected));

}

[Test]

public void TestPlayerCanLocateItemInTheirLocation()

{

location.Inventory.Put(sword);

player.Location = location;

GameObject expected = sword;

GameObject actual = player.Location.Locate("sword");

Assert.That(actual, Is.EqualTo(expected));

}

}

}

BagTest.cs

using System;

using System.Collections.Generic;

using NUnit.Framework;

namespace Iteration7

{

[TestFixture]

public class TestBag

{

Item sword;

Item shield;

Bag bag;

Bag backpack;

[SetUp]

public void SetUp()

{

sword = new Item(["sword"], "diamond sword", "a diamond sword which has not broken once");

shield = new Item(["shield"], "gold shield", "a gold shield that lasts a lifetime");

bag = new Bag(["bag"], "leather bag", "a light bag, suitable for short trips");

backpack = new Bag(["backpack"], "fabric backpack", "a medium-sized backpack, suitable for abroad travelling");

bag.Inventory.Put(sword);

backpack.Inventory.Put(shield);

backpack.Inventory.Put(bag);

}

[Test]

public void TestBagLocateItems()

{

Assert.Multiple(() =>

{

Assert.That(bag.Locate("sword"), Is.EqualTo(sword));

Assert.That(backpack.Locate("shield"), Is.EqualTo(shield));

});

}

[Test]

public void TestBagLocatesItself()

{

Assert.Multiple(() =>

{

Assert.That(bag.Locate("bag"), Is.EqualTo(bag));

Assert.That(backpack.Locate("backpack"), Is.EqualTo(backpack));

});

}

[Test]

public void TestBagLocatesNothing()

{

Assert.That(bag.Locate("Nothing"), Is.EqualTo(null));

}

[Test]

public void TestBagFullDesc()

{

Assert.That(bag.FullDescription, Is.EqualTo("In the leather bag you can see:\n\ta diamond sword (sword)\n"));

}

[Test]

public void TestBagInBag()

{

Assert.Multiple(() =>

{

Assert.That(backpack.Locate("bag"), Is.EqualTo(bag));

Assert.That(bag.Locate("sword"), Is.EqualTo(sword));

Assert.That(bag.Locate("shield"), Is.EqualTo(null));

});

}

}

}

PathTest.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Iteration7

{

public class PathTest

{

Player? \_testPlayer;

Location? \_testRoomA;

Location? \_testRoomB;

Path? \_testPath;

[Test]

public void TestLocatePathDestination()

{

\_testPlayer = new Player("ruchan", "student");

\_testRoomA = new Location(["bedroom"], "bedroom", "Room for decoration");

\_testRoomB = new Location(["bathroom"], "bathroom", "Room for illustration");

\_testPlayer.Location = \_testRoomA;

\_testPath = new Path(["north"], "Door", "A test door", \_testRoomA, \_testRoomB);

\_testRoomA.AddPath(\_testPath);

Location \_expected = \_testRoomB;

Location \_actual = \_testPath.Destination;

Assert.That(\_actual, Is.EqualTo(\_expected));

}

[Test]

public void TestLocatePathName()

{

\_testPlayer = new Player("ruchan", "student");

\_testRoomA = new Location(["bedroom"], "bedroom", "Room for decoration");

\_testRoomB = new Location(["bathroom"], "bathroom", "Room for illustration");

\_testPlayer.Location = \_testRoomA;

\_testPath = new Path(["north"], "Door", "A test door", \_testRoomA, \_testRoomB);

\_testRoomA.AddPath(\_testPath);

string \_expected = "A test door";

string \_actual = \_testPath.FullDescription;

Assert.That(\_actual, Is.EqualTo(\_expected));

}

[Test]

public void TestLocatePath()

{

\_testPlayer = new Player("ruchan", "student");

\_testRoomA = new Location(["bedroom"], "bedroom", "Room for decoration");

\_testRoomB = new Location(["bathroom"], "bathroom", "Room for illustration");

\_testPlayer.Location = \_testRoomA;

\_testPath = new Path(["north"], "Door", "A test door", \_testRoomA, \_testRoomB);

\_testRoomA.AddPath(\_testPath);

GameObject \_expected = \_testRoomA.Locate("north");

GameObject \_actual = \_testPath;

Assert.That(\_actual, Is.EqualTo(\_expected));

}

}

}

MoveTest.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Iteration7

{

public class MoveTest

{

Move? \_moveCommand;

Player? \_testPlayer;

Location? \_testRoomA;

Location? \_testRoomB;

Path? \_testPath;

[Test]

public void TestPlayerCanMove()

{

\_moveCommand = new Move();

\_testPlayer = new Player("ruchan", "student");

\_testRoomA = new Location(["bedroom"], "bedroom", "Room for decoration");

\_testRoomB = new Location(["bathroom"], "bathroom", "Room for illustration");

\_testPlayer.Location = \_testRoomA;

\_testPath = new Path(["north"], "Door", "A test door", \_testRoomA, \_testRoomB);

\_testRoomA.AddPath(\_testPath);

\_moveCommand.Execute(\_testPlayer, ["move", "north"]);

string \_expected = \_testRoomB.Name;

string \_actual = \_testPlayer.Location.Name;

Assert.That(\_actual, Is.EqualTo(\_expected), "Testing that player can move");

}

[Test]

public void TestMoveDescription()

{

\_moveCommand = new Move();

\_testPlayer = new Player("ruchan", "student");

\_testRoomA = new Location(["bedroom"], "bedroom", "Room for decoration");

\_testRoomB = new Location(["bathroom"], "bathroom", "Room for illustration");

\_testPlayer.Location = \_testRoomA;

\_testPath = new Path(["north"], "door", "A test door", \_testRoomA, \_testRoomB);

\_testRoomA.AddPath(\_testPath);

string \_expected = "You have moved north through a door to bathroom, Room for illustration.\nItems available:\n\nThere are no exits.";

string \_actual = \_moveCommand.Execute(\_testPlayer, ["move", "north"]);

Assert.That(\_actual, Is.EqualTo(\_expected), "Testing that move description is correct");

}

[Test]

public void TestInvalidMoveNoDirection()

{

\_moveCommand = new Move();

\_testPlayer = new Player("ruchan", "student");

\_testRoomA = new Location(["bedroom"], "bedroom", "Room for decoration");

\_testRoomB = new Location(["bathroom"], "bathroom", "Room for illustration");

\_testPlayer.Location = \_testRoomA;

\_testPath = new Path(["north"], "Door", "A test door", \_testRoomA, \_testRoomB);

string \_expected = "Where do you want to go?";

string \_actual = \_moveCommand.Execute(\_testPlayer, ["move"]); ;

Assert.That(\_actual, Is.EqualTo(\_expected), "Testing invalid move: no path specified");

}

[Test]

public void TestInvalidMoveNoPath()

{

\_moveCommand = new Move();

\_testPlayer = new Player("ruchan", "student");

\_testRoomA = new Location(["bedroom"], "bedroom", "Room for decoration");

\_testRoomB = new Location(["bathroom"], "bathroom", "Room for illustration");

\_testPlayer.Location = \_testRoomA;

\_testPath = new Path(["north"], "Door", "A test door", \_testRoomA, \_testRoomB);

string \_expected = "The exit is not available.";

string \_actual = \_moveCommand.Execute(\_testPlayer, ["move", "east"]); ;

Assert.That(\_actual, Is.EqualTo(\_expected), "Testing invalid move: non-existent path");

}

[Test]

public void TestInvalidMoveNoLocation()

{

\_moveCommand = new Move();

\_testPlayer = new Player("ruchan", "student");

\_testRoomA = new Location(["bedroom"], "bedroom", "Room for decoration");

\_testPlayer.Location = \_testRoomA;

string \_expected = "The exit is not available.";

string \_actual = \_moveCommand.Execute(\_testPlayer, ["move", "east"]); ;

Assert.That(\_actual, Is.EqualTo(\_expected), "Testing invalid move: no destination location specified");

}

}

}

ItemsTest.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Iteration7

{

[TestFixture]

public class TestItem

{

Item shield;

[SetUp]

public void SetUp()

{

shield = new Item(["shield"], "gold shield", "a gold shield that lasts a lifetime");

}

[Test]

public void TestItemIdentifiable()

{

Assert.That(shield.AreYou("shield"), Is.True, "True");

Assert.That(shield.AreYou("sword"), Is.False, "True");

}

[Test]

public void TestShortDesc()

{

Assert.That(shield.ShortDescription, Is.EqualTo("a gold shield (shield)"));

}

[Test]

public void TestFullDesc()

{

Assert.That(shield.FullDescription, Is.EqualTo("a gold shield that lasts a lifetime"));

}

}

}

IdentifiableObjectTest.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using NUnit.Framework;

namespace Iteration7

{

[TestFixture]

public class TestIdentifiableObject

{

[Test]

public void TestAreYou()

{

string[] testArray = ["Fred", "Bob"];

IdentifiableObject testIdentifiableObject = new(testArray);

Assert.That(testIdentifiableObject.AreYou("fred"), Is.True);

}

[Test]

public void TestNotAreYou()

{

string[] testArray = ["Fred", "Bob"];

IdentifiableObject testIdentifiableObject = new(testArray);

Assert.That(testIdentifiableObject.AreYou("wilma"), Is.False);

}

[Test]

public void TestCaseSensitive()

{

string[] testArray = ["Fred", "Bob"];

IdentifiableObject testIdentifiableObject = new(testArray);

Assert.That(testIdentifiableObject.AreYou("bOB"), Is.True);

}

[Test]

public void TestFirstID()

{

string[] testArray = ["Fred", "Bob"];

IdentifiableObject testIdentifiableObject = new(testArray);

StringAssert.AreEqualIgnoringCase("fred", testIdentifiableObject.FirstID);

}

[Test]

public void TestFirstIDWithNoIDs()

{

string[] testArray = [];

IdentifiableObject testIdentifableObject = new(testArray);

StringAssert.AreEqualIgnoringCase("", testIdentifableObject.FirstID);

}

[Test]

public void TestAddID()

{

string[] testArray = ["Fred", "Bob"];

IdentifiableObject testIdentifiableObject = new(testArray);

testIdentifiableObject.AddIdentifier("Wilma");

Assert.Multiple(() =>

{

Assert.That(testIdentifiableObject.AreYou("fred"), Is.True);

Assert.That(testIdentifiableObject.AreYou("bob"), Is.True);

Assert.That(testIdentifiableObject.AreYou("wilma"), Is.True);

});

}

}

}

Ảnh có chứa ảnh chụp màn hình, văn bản, phần mềm, máy tính

Mô tả được tạo tự động

Ảnh có chứa văn bản, ảnh chụp màn hình, phần mềm, Phần mềm đa phương tiện

Mô tả được tạo tự động

Ảnh có chứa văn bản, biểu đồ, hàng, Kế hoạch

Mô tả được tạo tự động

Ảnh có chứa văn bản, ảnh chụp màn hình, Phông chữ, biên lai

Mô tả được tạo tự động