Swinburne University of Technology

COS20007 Object Oriented Programming

Case Study - Iteration 7 - Paths

PDF generated at 21:58 on Wednesday 22nd November, 2023

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

namespace MazeGame
{
    public class Paths : GameObject
    {
        private Location _location;
        public Paths(string[] ids, string name, string desc, Location location) :
↪   base(ids, name, desc)
        {
            _location = location;
        }

        public Location LinkedArea
        {
            get { return _location; }
        }

    }
}
```

```
1
2  namespace MazeGame.nUnitTests
3  {
4      public class PathsTests
5      {
6          private Player _player { get; set; } = null!;
7          private Location garden { get; set; } = null!;
8          private Item water { get; set; } = null!;
9          private Item pearl { get; set; } = null!;
10         private Location area51 { get; set; } = null!;
11         private Location library { get; set; } = null!;
12         private Item book { get; set; } = null!;
13         private Paths gardenPath1 { get; set; } = null!;
14         private Paths gardenPath2 { get; set; } = null!;
15         private Paths area51Path1 { get; set; } = null!;
16         private MoveCommand move { get; set; } = null!;
17         private Paths libraryPath1 { get; set; } = null!;
18
19         [SetUp]
20         public void SetUp()
21         {
22             //Location 1
23             garden = new Location(new string[] { "garden" }, "green garden", "A
   ↪ garden blooming with natural plants, trees, and flowers");
24             water = new Item(new string[] { "water" }, "a bottled water", "A 1 Litres
   ↪ bottle of spring water to keep you hydrated");
25             pearl = new Item(new string[] { "pearl" }, "a pearl", "A pearl picked
   ↪ from pearl tree. A fruit great for snack");
26             garden.Inventory.Put(water);
27             garden.Inventory.Put(pearl);
28
29             //Location 2
30             area51 = new Location(new string[] { "area51" }, "area 51", "Special
   ↪ labratory for aliens");
31
32             //Location 3
33             library = new Location(new string[] { "library" }, "archive library",
   ↪ "area that contains old history book");
34             book = new Item(new string[] { "book" }, "a history book", "A book that
   ↪ captures the history of this city");
35             library.Inventory.Put(book);
36
37             //Paths in Garden
38             gardenPath1 = new Paths(new string[] { "n" }, "north",
39                 "You got in your car and travelled through the road up North",
   ↪ area51);
40             gardenPath2 = new Paths(new string[] { "e" }, "east",
41                 "You walked for a kilometer to a library in East", library);
42             garden.PathList.Add(gardenPath1);
43             garden.PathList.Add(gardenPath2);
44
45             //Paths in Area51
46             area51Path1 = new Paths(new string[] { "s" }, "south",
```

```
47                  "You got in your car and travelled through the road down South",
    ↪ garden);
48              area51.PathList.Add(area51Path1);
49
50              //Paths in Library
51              libraryPath1 = new Paths(new string[] { "w" }, "west",
52                  "You walked for a kilometer to a garden in East.", garden);
53              library.PathList.Add(libraryPath1);
54
55              //Command
56              move = new MoveCommand(new string[] { "move", "go", "head", "leave" });
57
58              //Player
59              _player = new Player("Hoang An", "the comtemplator of infinity");
60              _player.ChangeLocation(garden);
61          }
62
63          [Test]
64          public void Test_PathIdentifiable()
65          {
66              var sut1 = gardenPath1.AreYou("n");
67              var sut2 = gardenPath2.AreYou("e");
68              var sut3 = area51Path1.AreYou("s");
69              var sut4 = libraryPath1.AreYou("w");
70
71              Assert.Multiple(() =>
72              {
73                  Assert.IsTrue(sut1);
74                  Assert.IsTrue(sut2);
75                  Assert.IsTrue(sut3);
76                  Assert.IsTrue(sut4);
77              });
78          }
79
80
81          [Test]
82          public void Test_InsensitiveCase()
83          {
84              Console.WriteLine("Input: mOvE");
85              Console.WriteLine($"Before moving north:
    ↪ {_player.CurrentLocation.Name}");
86              var sut = move.Execute(_player, new string[] { "mOvE", "n" });
87              string expectedPath = $"You travelled towards {gardenPath1.Name}\n" +
88                                    $"{gardenPath1.Description}\n" +
89                                    $"Items available in this
    ↪ area:\n{gardenPath1.LinkedArea.Inventory.ItemList}\n";
90
91              Assert.Multiple(() =>
92              {
93                  Assert.That(sut, Is.EqualTo(expectedPath));
94                  Assert.That(_player.CurrentLocation, Is.EqualTo(area51));
95              });
96              Console.WriteLine($"After moving north: {_player.CurrentLocation.Name}");
```

```
97              Console.WriteLine(sut);
98          }
99
100         [Test]
101         public void Test_MoveByPathID()
102         {
103             Console.WriteLine($"Before moving east: {_player.CurrentLocation.Name}");
104             var sut = move.Execute(_player, new string[] { "move", "e" });
105             string expectedPath = $"You travelled towards {gardenPath2.Name}\n" +
106                             $"{gardenPath2.Description}\n" +
107                             $"Items available in this
    ↪ area:\n{gardenPath2.LinkedArea.Inventory.ItemList}\n";
108
109             Assert.Multiple(() =>
110             {
111                 Assert.That(sut, Is.EqualTo(expectedPath));
112                 Assert.That(_player.CurrentLocation, Is.EqualTo(library));
113             });
114             Console.WriteLine($"After moving east: {_player.CurrentLocation.Name}");
115             Console.WriteLine(sut);
116         }
117
118         [Test]
119         public void Test_MoveByPathName()
120         {
121             Console.WriteLine($"Before moving east: {_player.CurrentLocation.Name}");
122             var sut = move.Execute(_player, new string[] { "move", "east" });
123             string expectedPath = $"You travelled towards {gardenPath2.Name}\n" +
124                             $"{gardenPath2.Description}\n" +
125                             $"Items available in this
    ↪ area:\n{gardenPath2.LinkedArea.Inventory.ItemList}\n";
126
127             Assert.Multiple(() =>
128             {
129                 Assert.That(sut, Is.EqualTo(expectedPath));
130                 Assert.That(_player.CurrentLocation, Is.EqualTo(library));
131             });
132             Console.WriteLine($"After moving east: {_player.CurrentLocation.Name}");
133             Console.WriteLine(sut);
134         }
135
136         [Test]
137         public void Test_InvalidPath()
138         {
139             Console.WriteLine($"Before moving south:
    ↪ {_player.CurrentLocation.Name}");
140             var sut = move.Execute(_player, new string[] { "move", "south" });
141             string expectedPath = "I can't find path: south";
142             Assert.That(sut, Is.EqualTo(expectedPath));
143             Console.WriteLine(sut);
144         }
145     }
146 }
```

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

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

        public Location(string[] ids, string name, string desc) : base(ids, name,
↪   desc)
        {
            _inventory = new Inventory();
            _paths = new List<Paths>();
        }

        public GameObject Locate(string id)
        {
            List<GameObject> items = new List<GameObject>();

            if(_inventory.HasItem(id))
            {
                var itm = _inventory.Fetch(id);
                items.Add(itm);
            }
            else
            {
                foreach(Paths path in _paths)
                {
                    if(id == path.FirstId || String.Equals(id, path.Name,
↪   StringComparison.OrdinalIgnoreCase))
                    {
                        var itm = path;
                        items.Add(itm);
                    }
                }
            }

            if(items.Count > 0)
            {
                var result = items.ElementAt(0);
                items.Clear();
                return result;
            }
            else
            {
                return null;
            }
        }
```

```
52
53          public override string FullDescription
54          {
55              get
56              {
57                  List<string> ways = new List<string>();
58                  foreach (var way in PathList)
59                  {
60                      ways.Add(way.Name);
61                  }
62                  string[] arr = ways.ToArray();
63                  string availWays = string.Join("\n", arr);
64
65
66                  return $"You are in a {Name}\n" +
67                  $"{Description}\n" +
68                  $"There are {PathList.Count} available pathways: \n{availWays}\n" +
69                  $"Items available in this area:\n" +
70                  $"{Inventory.ItemList}";
71              }
72          }
73
74          public Inventory Inventory
75          {
76              get { return _inventory; }
77          }
78
79          public List<Paths> PathList
80          {
81              get { return _paths; }
82          }
83      }
84  }
```

```
using System.ComponentModel;
using System.Diagnostics.Metrics;
using System.Security.Cryptography;

namespace MazeGame.nUnitTests
{
    public class LocationTests
    {
        private Location garden { get; set; } = null!;
        private Paths gardenPath1 { get; set; } = null!;
        private Paths gardenPath2 { get; set; } = null!;
        private Location area51 { get; set; } = null!;
        private Location library { get; set; } = null!;
        private Item water { get; set; } = null!;
        private Item pearl { get; set; } = null!;
        private Player _player { get; set; } = null!;
        private Item sword { get; set; } = null!;
        private Item shovel { get; set; } = null!;
        private Item pickaxe { get; set; } = null!;
        private LookCommand look { get; set; } = null!;
        private Bag _bag1 { get; set; } = null!;
        private Item gem { get; set; } = null!;

        [SetUp]
        public void SetUp()
        {
            garden = new Location(new string[] { "garden" }, "green garden", "A garden blooming with natural plants, trees, and flowers");
            water = new Item(new string[] { "water" }, "a bottled water", "A 1 Litres bottle of spring water to keep you hydrated");
            pearl = new Item(new string[] { "pearl" }, "a pearl", "A pearl picked from pearl tree. A fruit great for snack");
            garden.Inventory.Put(water);
            garden.Inventory.Put(pearl);

            area51 = new Location(new string[] { "area51" }, "area 51", "Special labratory for aliens");
            library = new Location(new string[] { "library" }, "archive library", "area that contains old history book");
            gardenPath1 = new Paths(new string[] { "n" }, "north",
            "You got in your car and travelled through the road up North", area51);
            gardenPath2 = new Paths(new string[] { "e" }, "east",
                "You walked for a kilometer to a library in East", library);
            garden.PathList.Add(gardenPath1);
            garden.PathList.Add(gardenPath2);


            _player = new Player("Hoang An", "the comtemplator of infinity");
            sword = new Item(new string[] { "sword" }, "a bronze sword", "A short sword cast from bronze");
            shovel = new Item(new string[] { "shovel" }, "a shovel", "A durable shovel borrowed from the village");
            pickaxe = new Item(new string[] { "pickaxe" }, "an obsidian pickaxe", "A pickaxe made of obsidian");
```

```
47              _player.ChangeLocation(garden);
48              _player.Inventory.Put(sword);
49              _player.Inventory.Put(shovel);
50              _player.Inventory.Put(pickaxe);
51              look = new LookCommand(new string[] { "look", "Look" });
52          }
53
54          [Test]
55          public void Test_LocationIsIdentifiable()
56          {
57              var sut = garden.AreYou("garden");
58              Assert.That(sut, Is.True);
59              Console.WriteLine(sut);
60          }
61
62          [Test]
63          public void Test_LocationLocatesPathID()
64          {
65              var sut = garden.Locate("n");
66              Assert.That(sut, Is.EqualTo(gardenPath1));
67              Console.WriteLine(sut);
68          }
69
70          [Test]
71          public void Test_LocationLocatesInvalidPath()
72          {
73              var sut = garden.Locate("s");
74              Assert.That(sut, Is.Not.EqualTo(gardenPath1));
75              Console.WriteLine(sut);
76          }
77
78          [Test]
79          public void Test_LocationLocatesPathName()
80          {
81              var sut = garden.Locate("east");
82              Assert.That(sut, Is.EqualTo(gardenPath2));
83              Console.WriteLine(sut);
84          }
85
86          [Test]
87          public void Test_LocationSelfLocate()
88          {
89              string command = "Look";
90              string[] array = command.Split(' ');
91              var sut = look.Execute(_player, array);
92              Assert.Multiple(() =>
93              {
94                  Assert.That(sut, Is.Not.Null);
95                  Assert.That(sut, Is.EqualTo(garden.FullDescription));
96              });
97              Console.WriteLine(sut);
98          }
99
```

```
100         [Test]
101         public void Test_LocationLocatesItem()
102         {
103             var sut1 = garden.Locate("water");
104             var sut2 = garden.Locate("pearl");
105             Assert.Multiple(() =>
106             {
107                 Assert.That(sut1.Description, Is.EqualTo(water.Description));
108                 Assert.That(sut1.Description, Is.Not.Null);
109                 Assert.That(sut2.Description, Is.EqualTo(pearl.Description));
110                 Assert.That(sut2.Description, Is.Not.Null);
111             });
112             Console.WriteLine($"Water: {sut1.Description}");
113             Console.WriteLine($"Pearl: {sut2.Description}");
114         }
115 
116         [Test]
117         public void Test_LocationLocatesUnkItem()
118         {
119             var sut = garden.Locate("chair");
120             Assert.That(sut, Is.Null);
121             if(sut == null)
122             {
123                 Console.WriteLine("Item was not found");
124             }
125         }
126 
127         [Test]
128         public void Test_PlayerLocatesItemInArea()
129         {
130             string command1 = "look at water";
131             string command2 = "look at pearl";
132             string[] array1 = command1.Split(' ');
133             string[] array2 = command2.Split(' ');
134             var sut1 = look.Execute(_player, array1);
135             var sut2 = look.Execute(_player, array2);
136             Assert.Multiple(() =>
137             {
138                 Assert.That(sut1, Is.EqualTo(water.Description));
139                 Assert.That(sut1, Is.Not.Null);
140                 Assert.That(sut2, Is.EqualTo(pearl.Description));
141                 Assert.That(sut2, Is.Not.Null);
142             });
143             Console.WriteLine($"Water: {sut1}");
144             Console.WriteLine($"Pearl: {sut2}");
145         }
146 
147         [Test]
148         public void Test_LocationLocatesPlayerItem()
149         {
150             var sut1 = garden.Locate("sword");
151             var sut2 = garden.Locate("shovel");
152             var sut3 = garden.Locate("pickaxe");
```

```
153
154             Assert.Multiple(() =>
155             {
156                 Assert.That(sut1, Is.Null);
157                 Assert.That(sut2, Is.Null);
158                 Assert.That(sut3, Is.Null);
159             });
160             if(sut1 == null && sut2 == null &&  sut3 == null)
161             {
162                 Console.WriteLine("Sword, shovel, and pickaxe is not found");
163             }
164         }
165
166         [Test]
167         public void Test_LocatesLocationItemInBag()
168         {
169             _bag1 = new Bag(new string[] { "bag1" }, "brown bag", "a bag made and
    ↪ stiched with leather.");
170             gem = new Item(new string[] { "gem" }, "a green gem", "A rare type of gem
    ↪ that can only be obtained through trade");
171             _bag1.Inventory.Put(gem);
172             _player.Inventory.Put(_bag1);
173
174             string command1 = "look at water in bag1";
175             string command2 = "look at pearl in bag1";
176             string[] array1 = command1.Split(' ');
177             string[] array2 = command2.Split(' ');
178
179             var sut1 = look.Execute(_player, array1);
180             var sut2 = look.Execute(_player, array2);
181
182             Assert.Multiple(() =>
183             {
184                 Assert.That(sut1, Is.EqualTo($"I can't find the {water.FirstId} in
    ↪ {_bag1.Name}"));
185                 Assert.That(sut2, Is.EqualTo($"I can't find the {pearl.FirstId} in
    ↪ {_bag1.Name}"));
186             });
187
188             Console.WriteLine(sut1);
189             Console.WriteLine(sut2);
190         }
191
192         [Test]
193         public void Test_LocatesBagItemInLocation()
194         {
195             _bag1 = new Bag(new string[] { "bag1" }, "brown bag", "a bag made and
    ↪ stiched with leather.");
196             gem = new Item(new string[] { "gem" }, "a green gem", "A rare type of gem
    ↪ that can only be obtained through trade");
197             _bag1.Inventory.Put(gem);
198             _player.Inventory.Put(_bag1);
199
```

```
            var sut = garden.Locate("gem");
            Assert.That(sut, Is.Null);
            if (sut == null)
            {
                Console.WriteLine("Item was not found");
            }
        }
    }
}
```

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

namespace MazeGame
{
    public class LookCommand : Command
    {
        public LookCommand(string[] ids) : base(ids)
        {

        }

        public override string Execute(Player p, string[] text)
        {
            IHaveInventory container;
            string itemID;

            if (text[0] == "look")
            {
                if (text.Length == 3 || text.Length == 5)
                {
                    if (text[1] == "at")
                    {
                        if (text.Length == 3)
                        {
                            container = p;
                            itemID = text[2];

                            return LookAtIn(itemID, container);
                        }
                        else if (text.Length == 5 && text[3] == "in")
                        {
                            string containerID = text[4];
                            container = FetchContainer(p, containerID);
                            string itmReturn;
                            if(container != null)
                            {
                                itemID = text[2];
                                itmReturn = LookAtIn(itemID, container);

                                if(itmReturn == ("I can't find the " + itemID))
                                {
                                    return $"I can't find the {itemID} in
↪   {container.Name}";
                                }
                                else
                                {
                                    return itmReturn;
                                }
                            }
```

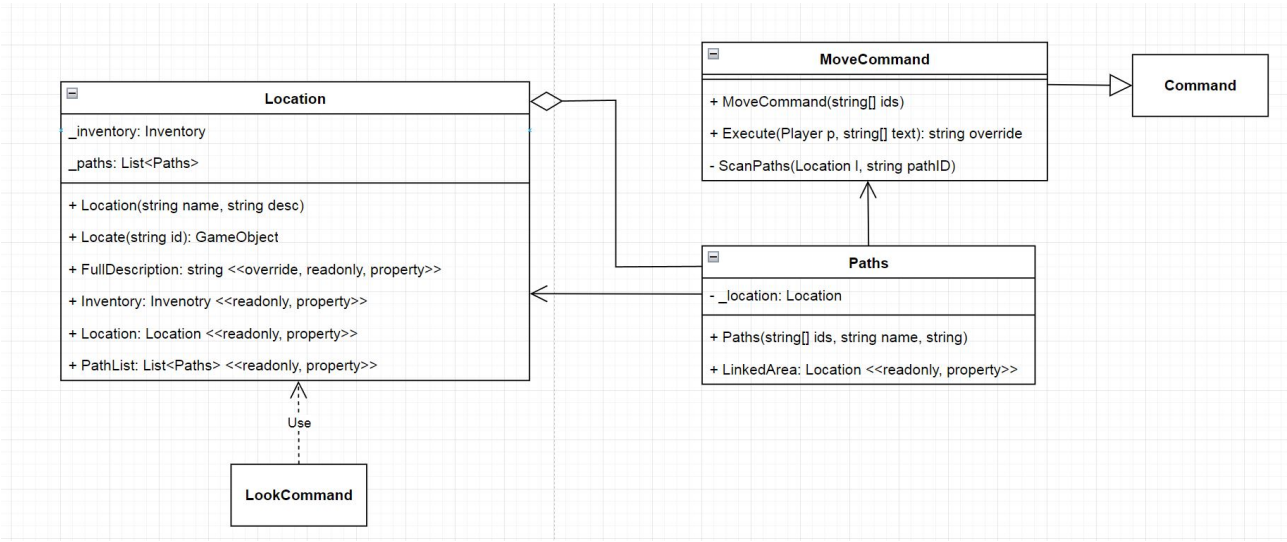
```
                            else
                            {
                                return "I can't find the " + containerID;
                            }
                        }
                        else
                        {
                            return "What do you want to looking in?";
                        }
                    }
                    else
                    {
                        return "What do you want to look at?";
                    }
                }
                else
                {
                    return "I don't know how to look like that";
                }
            }
            else if (text[0] == "Look")
            {
                return p.CurrentLocation.FullDescription;
            }
            else
            {
                return "Error in look input";
            }
        }

        private IHaveInventory FetchContainer (Player p, string containerID)
        {
            var result = p.Locate(containerID);
            return (IHaveInventory)result;
        }

        private string LookAtIn(string thingId, IHaveInventory container)
        {
            var result = container.Locate(thingId);

            if(result != null)
            {
                return result.FullDescription;
            }
            else
            {
                return "I can't find the " + thingId;
            }
        }
    }
}
```

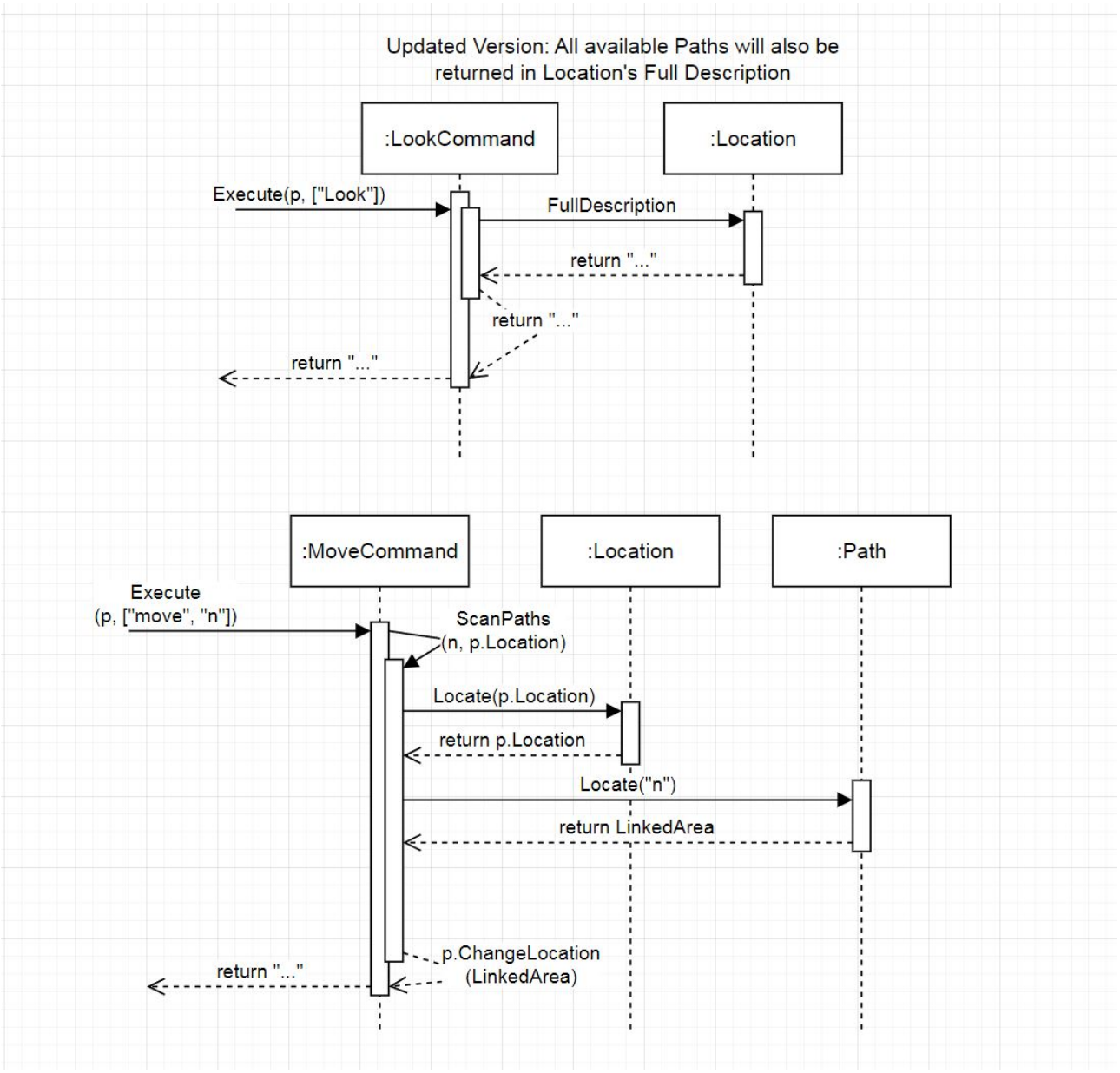
```
namespace MazeGame.nUnitTests
{
    public class MoveCommandTests
    {
        private Player _player { get; set; } = null!;
        private Location garden { get; set; } = null!;
        private Item water { get; set; } = null!;
        private Item pearl { get; set; } = null!;
        private Location area51 { get; set; } = null!;
        private Location library { get; set; } = null!;
        private Item book { get; set; } = null!;
        private Paths gardenPath1 { get; set; } = null!;
        private Paths gardenPath2 { get; set; } = null!;
        private Paths area51Path1 { get; set; } = null!;
        private MoveCommand move { get; set; } = null!;
        private Paths libraryPath1 { get; set; } = null!;

        [SetUp]
        public void SetUp()
        {
            //Location 1
            garden = new Location(new string[] { "garden" }, "green garden", "A garden blooming with natural plants, trees, and flowers");
            water = new Item(new string[] { "water" }, "a bottled water", "A 1 Litres bottle of spring water to keep you hydrated");
            pearl = new Item(new string[] { "pearl" }, "a pearl", "A pearl picked from pearl tree. A fruit great for snack");
            garden.Inventory.Put(water);
            garden.Inventory.Put(pearl);

            //Location 2
            area51 = new Location(new string[] { "area51" }, "area 51", "Special labratory for aliens");

            //Location 3
            library = new Location(new string[] { "library" }, "archive library", "area that contains old history book");
            book = new Item(new string[] { "book" }, "a history book", "A book that captures the history of this city");
            library.Inventory.Put(book);

            //Paths in Garden
            gardenPath1 = new Paths(new string[] { "n" }, "north",
                "You got in your car and travelled through the road up North", area51);
            gardenPath2 = new Paths(new string[] { "e" }, "east",
                "You walked for a kilometer to a library in East", library);
            garden.PathList.Add(gardenPath1);
            garden.PathList.Add(gardenPath2);

            //Paths in Area51
            area51Path1 = new Paths(new string[] { "s" }, "south",
                "You got in your car and travelled through the road down South", garden);
```

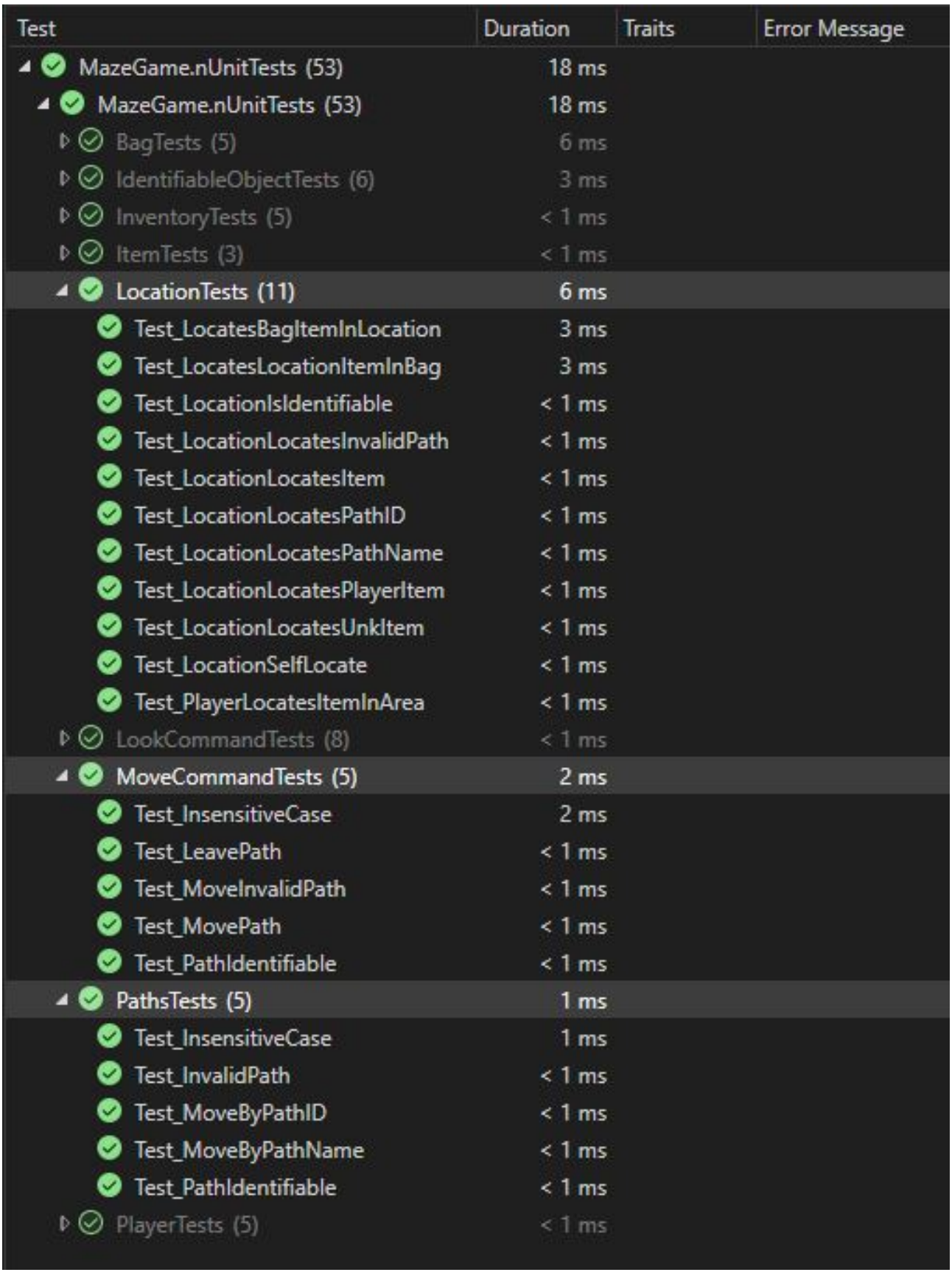
```
47              area51.PathList.Add(area51Path1);
48
49              //Paths in Library
50              libraryPath1 = new Paths(new string[] { "w" }, "west",
51                  "You walked for a kilometer to a garden in East.", garden);
52              library.PathList.Add(libraryPath1);
53
54              //Command
55              move = new MoveCommand(new string[] { "move", "go", "head", "leave" });
56
57              //Player
58              _player = new Player("Hoang An", "the comtemplator of infinity");
59              _player.ChangeLocation(garden);
60          }
61
62          [Test]
63          public void Test_PathIdentifiable()
64          {
65              var sut1 = gardenPath1.AreYou("n");
66              var sut2 = gardenPath2.AreYou("e");
67              var sut3 = area51Path1.AreYou("s");
68              var sut4 = libraryPath1.AreYou("w");
69
70              Assert.Multiple(() =>
71              {
72                  Assert.IsTrue(sut1);
73                  Assert.IsTrue(sut2);
74                  Assert.IsTrue(sut3);
75                  Assert.IsTrue(sut4);
76              });
77          }
78
79          [Test]
80          public void Test_MovePath()
81          {
82              Console.WriteLine($"Before moving east: {_player.CurrentLocation.Name}");
83              var sut = move.Execute(_player, new string[] { "move", "e"});
84              string expectedPath = $"You travelled towards {gardenPath2.Name}\n" +
85                              $"{gardenPath2.Description}\n" +
86                              $"Items available in this
    ↪  area:\n{gardenPath2.LinkedArea.Inventory.ItemList}\n";
87
88              Assert.Multiple(() =>
89              {
90                  Assert.That(sut, Is.EqualTo(expectedPath));
91                  Assert.That(_player.CurrentLocation, Is.EqualTo(library));
92              });
93              Console.WriteLine($"After moving east: {_player.CurrentLocation.Name}");
94              Console.WriteLine(sut);
95          }
96
97          [Test]
98          public void Test_LeavePath()
```

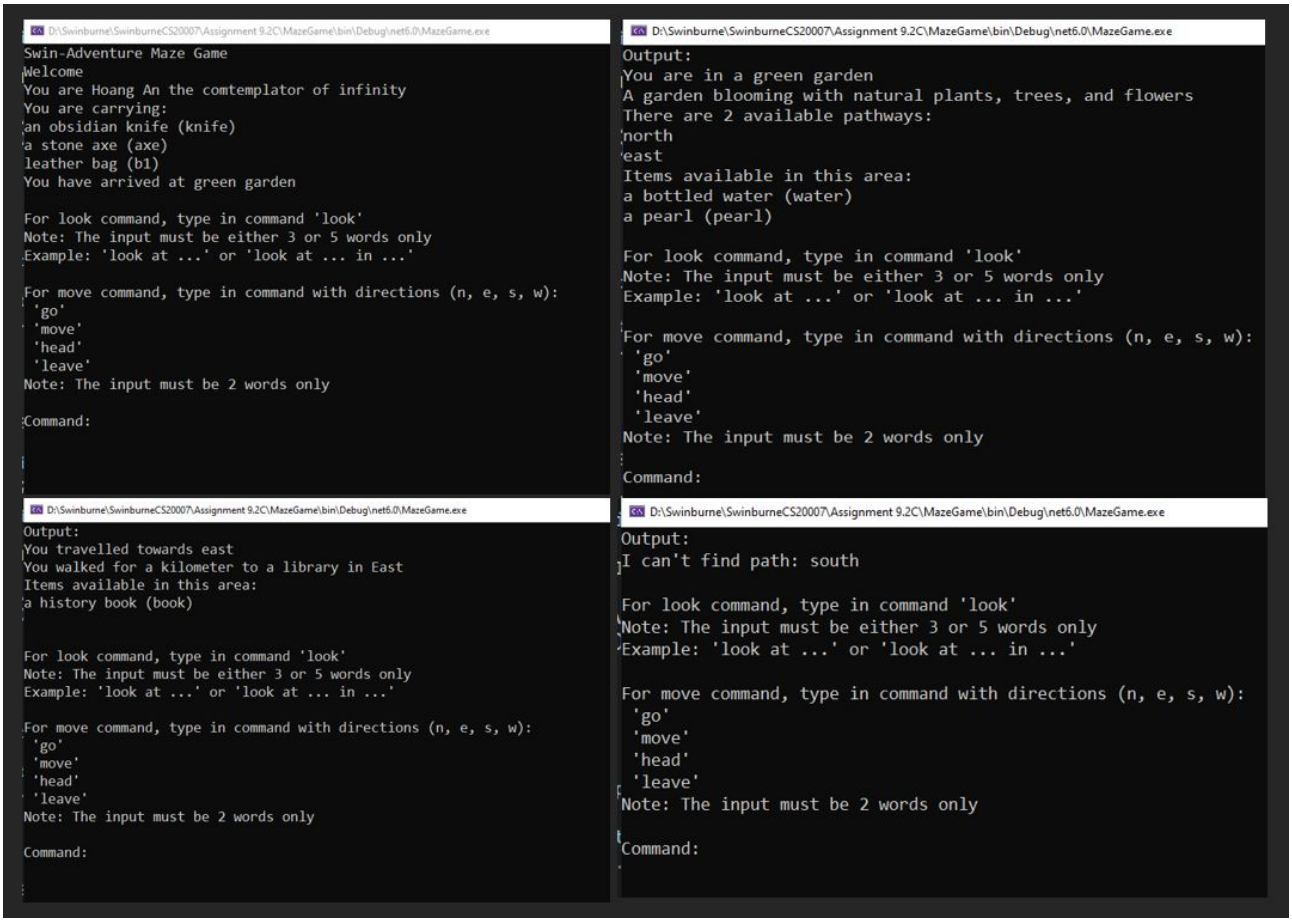
```
99          {
100             Console.WriteLine($"Before moving north:
    ↪ {_player.CurrentLocation.Name}");
101             var sut = move.Execute(_player, new string[] { "leave", "n" });
102             string expectedPath = $"You travelled towards {gardenPath1.Name}\n" +
103                             $"{gardenPath1.Description}\n" +
104                             $"Items available in this
    ↪ area:\n{gardenPath1.LinkedArea.Inventory.ItemList}\n";
105
106             Assert.Multiple(() =>
107             {
108                 Assert.That(sut, Is.EqualTo(expectedPath));
109                 Assert.That(_player.CurrentLocation, Is.EqualTo(area51));
110             });
111             Console.WriteLine($"After moving north: {_player.CurrentLocation.Name}");
112             Console.WriteLine(sut);
113         }
114
115         [Test]
116         public void Test_MoveInvalidPath()
117         {
118             Console.WriteLine($"Before moving west: {_player.CurrentLocation.Name}");
119             var sut = move.Execute(_player, new string[] { "move", "west" });
120             string expectedPath = "I can't find path: west";
121
122             Assert.Multiple(() =>
123             {
124                 Assert.That(sut, Is.EqualTo(expectedPath));
125                 Assert.That(_player.CurrentLocation, Is.EqualTo(garden));
126             });
127             Console.WriteLine($"After moving west: {_player.CurrentLocation.Name}");
128             Console.WriteLine(sut);
129         }
130
131         [Test]
132         public void Test_InsensitiveCase()
133         {
134             Console.WriteLine("Input: mOvE");
135             Console.WriteLine($"Before moving east: {_player.CurrentLocation.Name}");
136             var sut = move.Execute(_player, new string[] { "mOvE", "e" });
137             string expectedPath = $"You travelled towards {gardenPath2.Name}\n" +
138                             $"{gardenPath2.Description}\n" +
139                             $"Items available in this
    ↪ area:\n{gardenPath2.LinkedArea.Inventory.ItemList}\n";
140
141             Assert.Multiple(() =>
142             {
143                 Assert.That(sut, Is.EqualTo(expectedPath));
144                 Assert.That(_player.CurrentLocation, Is.EqualTo(library));
145             });
146             Console.WriteLine($"After moving east: {_player.CurrentLocation.Name}");
147             Console.WriteLine(sut);
148         }
```

```
149         }
150 }
```


Location
_inventory: Inventory
_paths: List<Paths>
+ Location(string name, string desc)
+ Locate(string id): GameObject
+ FullDescription: string <<override, readonly, property>>
+ Inventory: Invenotry <<readonly, property>>
+ Location: Location <<readonly, property>>
+ PathList: List<Paths> <<readonly, property>>
MoveCommand
+ MoveCommand(string[] ids)
+ Execute(Player p, string[] text): string override
- ScanPaths(Location l, string pathID)
Command
Paths
- _location: Location
+ Paths(string[] ids, string name, string)
+ LinkedArea: Location <<readonly, property>>
Use
LookCommand


Updated Version: All available Paths will also be returned in Location's Full Description
:LookCommand
:Location
Execute(p, ["Look"])
FullDescription
return "..."
return "..."
return "..."
:MoveCommand
:Location
:Path
Execute (p, ["move", "n"])
ScanPaths (n, p.Location)
Locate(p.Location)
return p.Location
Locate("n")
return LinkedArea
p.ChangeLocation (LinkedArea)
return "..."


Test
Duration
Traits
Error Message
MazeGame.nUnitTests (53) 18 ms
MazeGame.nUnitTests (53) 18 ms
BagTests (5) 6 ms
IdentifiableObjectTests (6) 3 ms
InventoryTests (5) < 1 ms
ItemTests (3) < 1 ms
LocationTests (11) 6 ms
Test_LocatesBagItemInLocation 3 ms
Test_LocatesLocationItemInBag 3 ms
Test_LocationIsIdentifiable < 1 ms
Test_LocationLocatesInvalidPath < 1 ms
Test_LocationLocatesItem < 1 ms
Test_LocationLocatesPathID < 1 ms
Test_LocationLocatesPathName < 1 ms
Test_LocationLocatesPlayerItem < 1 ms
Test_LocationLocatesUnkItem < 1 ms
Test_LocationSelfLocate < 1 ms
Test_PlayerLocatesItemInArea < 1 ms
LookCommandTests (8) < 1 ms
MoveCommandTests (5) 2 ms
Test_InsensitiveCase 2 ms
Test_LeavePath < 1 ms
Test_MoveInvalidPath < 1 ms
Test_MovePath < 1 ms
Test_PathIdentifiable < 1 ms
PathsTests (5) 1 ms
Test_InsensitiveCase 1 ms
Test_InvalidPath < 1 ms
Test_MoveByPathID < 1 ms
Test_MoveByPathName < 1 ms
Test_PathIdentifiable < 1 ms
PlayerTests (5) < 1 ms


D:\Swinburne\SwinburneCS20007\Assignment 9.2C\MazeGame\bin\Debug\net6.0\MazeGame.exe
Swin-Adventure Maze Game
Welcome
You are Hoang An the comtemplator of infinity
You are carrying:
an obsidian knife (knife)
a stone axe (axe)
leather bag (b1)
You have arrived at green garden
For look command, type in command 'look'
Note: The input must be either 3 or 5 words only
Example: 'look at ...' or 'look at ... in ...'
For move command, type in command with directions (n, e, s, w):
'go'
'move'
'head'
'leave'
Note: The input must be 2 words only
Command:
D:\Swinburne\SwinburneCS20007\Assignment 9.2C\MazeGame\bin\Debug\net6.0\MazeGame.exe
Output:
You are in a green garden
A garden blooming with natural plants, trees, and flowers
There are 2 available pathways:
north
east
Items available in this area:
a bottled water (water)
a pearl (pearl)
For look command, type in command 'look'
Note: The input must be either 3 or 5 words only
Example: 'look at ...' or 'look at ... in ...'
For move command, type in command with directions (n, e, s, w):
'go'
'move'
'head'
'leave'
Note: The input must be 2 words only
Command:
D:\Swinburne\SwinburneCS20007\Assignment 9.2C\MazeGame\bin\Debug\net6.0\MazeGame.exe
Output:
You travelled towards east
You walked for a kilometer to a library in East
Items available in this area:
a history book (book)
For look command, type in command 'look'
Note: The input must be either 3 or 5 words only
Example: 'look at ...' or 'look at ... in ...'
For move command, type in command with directions (n, e, s, w):
'go'
'move'
'head'
'leave'
Note: The input must be 2 words only
Command:
D:\Swinburne\SwinburneCS20007\Assignment 9.2C\MazeGame\bin\Debug\net6.0\MazeGame.exe
Output:
I can't find path: south
For look command, type in command 'look'
Note: The input must be either 3 or 5 words only
Example: 'look at ...' or 'look at ... in ...'
For move command, type in command with directions (n, e, s, w):
'go'
'move'
'head'
'leave'
Note: The input must be 2 words only
Command: