[The World of Yucca Moths]

Ecosystem PROG 201 Final Project

[Andrada Iorgulescu, May 8, 2022]

Contents

Project

Overview

A simulation of the Mojave Desert, and the relationship between yucca moths (Tegeticula yuccasella) and yucca plants (Yucca schidigera) who are in an obligate mutualism relationship.

Framework

In total, I have 9 classes:

Person Class: The base class of Player and Vendor class. I used this to create the Name and Inventory properties they both use. This class also holds the Buy() (add desired item to inventory) and Sell() (remove desired item from inventory) functions.

Player Class: Inherits Name and Inventory properties from Person class. Holds the LoadPlayerInventory() method which pulls the information from an XML that is then used to populate the player's start-up inventory. Also holds Harvest() (collect seeds from yucca plant), Plant() (use seeds to grow more yucca plants), and Feed() (give food from inventory to yucca moths and increase their population) methods. Class also inherits from the Trade interface, and holds the TradeItems() (Buy and Sell simultaneously) method.

Vendor Class: Inherits Name and Inventory properties from Person class. Holds the LoadVendorInventory() method which pulls the information from an XML that is then used to populate the vendor's inventory which the player will buy from and sell to.

Utility Class: Holds Print() (writes a message to the screen), Pause() (asks the player for their input to continue), and AllItemsInList() (outputs the items in a list) methods.

Trade Class: An interface which holds the TradeItems() method.

Entity Class:

Environment Class: Holds an enum which lists the 3 different statuses in the environment (balanced, unbalanced, unsound). Defines the Name parameter ("Mojave Desert Yucca Moth Ecosystem"). Instantiates the Entities list, which is populated using the LoadEntityData() method within the Entity class.

Item Class: Creates the name, description, and amount parameters that are used to explain the items in the player's and vendor's inventories

WeatherEvent Class: Holds the WeatherType enum, which lists the 3 different weather types in the game (sunny, cloudy, rainy). Holds GetWeather() method, which is used to generate a new weather condition for the current day.

Experience

Describe the experience for a potential player. (About 500-700 words).

Systems

Obligate Mutualism Relationship

Describe the elements, inputs, outputs, and results in your system. You may want to add diagrams to show how they work. (About 500-700 words each).

[Mojave Desert Yucca Moth Ecosystem]

Describe the elements, inputs, outputs, and results in your system. You may want to add diagrams to show how they work. (About 500-700 words each).

Playtest Sessions

Test Session (April 18, 2022)

1) *Alexandra Dimitrovici (outside of class)*

Most Successful Project Aspects

- *can harvest items from the yucca plant*
- *can trade with the vendor*

Biggest Project Issues

- *can't craft any items*
- *no use for items outside of trading with vendor*

How did you use what you learned to improve your application?

- *I still need to add in some required features and give the game more layers.*
- *I also want to make the interface prettier and add some better images*

Test Session (May 2, 2022)

1) *Alexandra Dimitrovici (outside of class)*

Most Successful Project Aspects

- *can see vendor's inventory list*
- *can see player's inventory list*

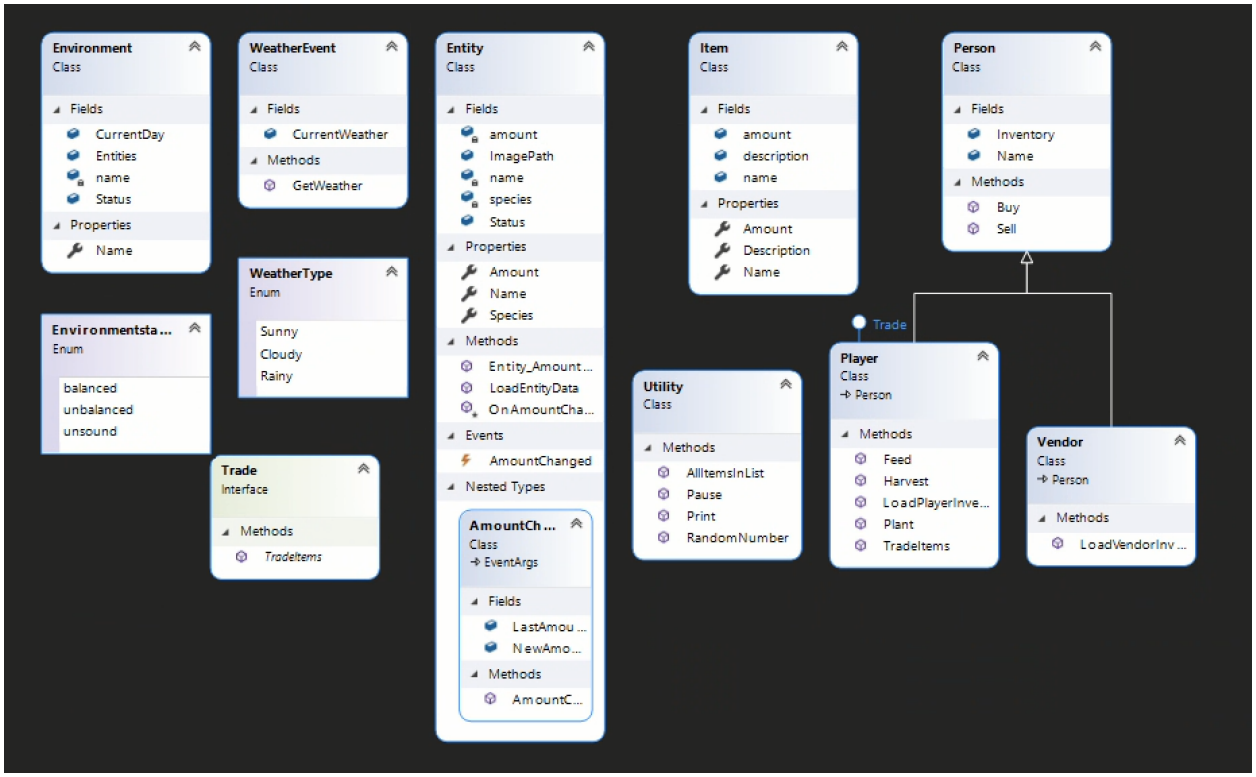
Biggest Project Issues

- *can't plant seeds*
- *can't feed yucca moths*

How did you use what you learned to improve your application?

- *fix the buttons for planting and feeding*
- *fix the output box so it shows updates about the ecosystem*

UML Diagrams



UML Diagram Explanation

Describe your structure and explain your design decisions (i.e., the rationale). Include information about the types of structures that you chose (classes, enums, etc.), inheritance, and class relationships.

Object-Oriented Programming Concepts

Separation of Concern

Definition: The idea that pieces of software should be separated based on the kind of tasks it is created to do.

Code excerpts from your project:

```
1 reference
public static void Print(string message)
{
    WriteLine(message);
}
0 references
public static void Pause()
{
    Print("Press any key to continue...");
    ReadKey();
}
```

Explain usage in your project: I have these two methods in my utility class. The Print method uses the Console.Writeline function to print a message to the screen. The Pause method uses the print method to give the player a prompt, and then the Console.ReadKey function to wait for their response. The Pause method could not function without the Print method, however it does not make sense for them to be one because they were created to accomplish different tasks. The Print method exists to show the player something, while the Pause method exists to wait for their input.

Polymorphism

Definition: The process of redefining methods in derived classes. It is most often used in late binding, where an object's response to the other members in its method is determined based on the object's type at run time.

Code excerpts from your project:

Explain usage in your project:

Inheritance ("is a")

Definition: Using a new class to reuse, extend, and modify the behavior defined in another(s). The class that is being inherited from is called the *base class*. The class that inherits is called a *derived class*.

Code excerpts from your project:

```
4 references
public class Person
{
    public string Name = "";
    public List<Item> Inventory = new List<Item>();
```

```
3 references
class Vendor : Person
{
```

```
1 reference
private void SetUpVendor()
{
    vendor.Name = "Vendor";
    vendor.Inventory = Vendor.LoadVendorInventory();
    VendorInventory.DataContext = vendor.Inventory;
    VendorNameInventory.DataContext = $" {vendor.Name} 's Inventory";
}
```

```
class Player : Person, Trade
{
```

```
private void SetUpPlayer()
{
    player.Name = "Anonymous Player";
    player.Inventory = Player.LoadPlayerInventory();
    PlayerInventory.DataContext = player.Inventory;
    PlayerNameInventory.DataContext = $" {player.Name} 's Inventory ";
}
```

```
public partial class MainWindow : Window
{
    Environment Desert = new Environment();
    Player player = new Player();
    Vendor vendor = new Vendor();
```

Explain usage in your project: The person class is where I created the Name and inventory properties. Vendor and Player both inherit from the person class as they both require names and an inventory. When the player and the vendor are instantiated at the beginning of the game, I am able to create names and inventories for them even though I did not set those up as parameters in their respective classes. Because they both inherit from a class that has those parameters, I do not have to add them into those classes again, and am able to just give a name and inventory to their objects when they are instantiated.

Containment (“has a”)

Definition:

Code excerpts from your project:

Explain usage in your project:

General Association (“uses a”)

Definition:

Code excerpts from your project:

Explain usage in your project:

C# Programming Skills

Delegate

Definition: A type that refers to methods with specific parameters and return types. It is called through the delegate instance.

Code excerpts from your project:

Explain usage in your project:

Enum

Definition: This is a value type that holds a set of named constants, which can be called upon by their numeric order (0-#).

Code excerpts from your project:

```
2 references
enum Environmentstatus
{
    balanced,
    unbalanced,
    unsound
}
```

```
8 references
enum WeatherType
{
    Sunny,
    Cloudy,
    Rainy
}
```

Explain usage in your project: The first enum exists within the Environment class and is used to hold different statuses for the environment. Balanced is for when everything is in order, unbalanced is for when things aren't perfect but could be recovered with player interaction, and unsound is for when the environment is in danger and will be hard or impossible to recover. The second enum is used to hold different weather types, which are different on each day. Different weather days have different effects on the yucca moths and plants. Sunny and rainy days improve plant and moth growth, while cloudy days don't.

External Data

Description: (i.e., plain text, XML, etc.) – note that a definition is not asked for – instead describe the type of data you will be reading into your application. For this application, I used XML files to read in external data. I used XML files to populate the player's starting inventory, the vendor's inventory, and the entities within the environment. The XML files supplied data such as: names, amounts, species, descriptions, and imagePaths.

Code excerpts from your project:

```
1 reference
public static List<Entity> LoadEntityData()
{
    List<Entity> tempEntities = new List<Entity>();
    string path = "../../data/entities.xml";
    XmlDocument doc = new XmlDocument();
    doc.Load(path);
    XmlElement root = doc.DocumentElement;
    XmlNodeList entities = root.ChildNodes;

    foreach (XmlElement x in entities)
    {
        tempEntities.Add(new Entity()
        {
            Name = x.GetAttribute("name"),
            Species = x.GetAttribute("species"),
            ImagePath = "media/" + x.GetAttribute("imagepath") + ".png",
            Amount = Convert.ToInt32(x.GetAttribute("amount"))
        });
    }

    return tempEntities;
}
```

```
public List<Entity> Entities = Entity.LoadEntityData();
```

1 entities.xml | 2 playerInventory.xml | 3 vendorInventory.xml

```
1   <environment type = "Desert" name = "Mojave Desert Yucca Moth Ecosystem">
2     <entity name="Yucca Moth" species = "Prodoxidae" imagepath = "yuccaMoth" amount = "1000">
3     </entity>
4     <entity name="Yucca Plant" species = "Yucca" imagepath = "yuccaPlant" amount = "1000">
5     </entity>
6      <entity name="Anonymous Player" species = "human">
7     </entity>
8      <entity name="Vendor" species = "human">
9     </entity>
10  </environment>
11
```

```
1 reference
public static List<Item> LoadPlayerInventory()
{
    List<Item> tempInventory = new List<Item>();
    string path = "../../data/playerInventory.xml";
    XmlDocument doc = new XmlDocument();
    doc.Load(path);
    XmlElement root = doc.DocumentElement;
    XmlNodeList playerInventory = root.ChildNodes;

    foreach (XmlElement x in playerInventory)
    {
        tempInventory.Add(new Item()
        {
            Name = x.GetAttribute("name"),
            Description = x.GetAttribute("description"),
            Amount = Convert.ToInt32(x.GetAttribute("amount"))
        });
    }

    return tempInventory;
}
```

```
1 reference
private void SetUpPlayer()
{
    player.Name = "Anonymous Player";
    player.Inventory = Player.LoadPlayerInventory();
    PlayerInventory.DataContext = player.Inventory;
    PlayerNameInventory.DataContext = $" {player.Name} 's Inventory ";
}
```

1 entities.xml | 2 playerInventory.xml | 3 vendorInventory.xml

```
1  <playerInventory>
2  <item name="Seeds" description ="Used to increase yucca plant population" amount = "10"/>
3  <item name="Food" description ="Used to increase yucca moth population" amount = "5"/>
4  <item name="Basket" description ="Used to hold seeds" amount = "1"/>
5  <item name="Fire" description ="Used to increase temperature" amount = "1"/>
6  <item name="Glacier" description ="Used to decrease temperature" amount = "1"/>
7  </playerInventory>
```

```
1 reference
public static List<Item> LoadVendorInventory()
{
    List<Item> tempInventory = new List<Item>();
    string path = "../../data/vendorInventory.xml";
    XmlDocument doc = new XmlDocument();
    doc.Load(path);
    XmlElement root = doc.DocumentElement;
    XmlNodeList vendorInventory = root.ChildNodes;

    foreach (XmlElement x in vendorInventory)
    {
        tempInventory.Add(new Item()
        {
            Name = x.GetAttribute("name"),
            Description = x.GetAttribute("description"),
            Amount = Convert.ToInt32(x.GetAttribute("amount"))
        });
    }

    return tempInventory;
}
```

```
1 reference
private void SetUpVendor()
{
    vendor.Name = "Vendor";
    vendor.Inventory = Vendor.LoadVendorInventory();
    VendorInventory.DataContext = vendor.Inventory;
    VendorNameInventory.DataContext = $" {vendor.Name} 's Inventory";
}
```

1 entities.xml | 2 playerInventory.xml | 3 vendorInventory.xml

```
1  - <vendorInventory>
2    <item name="Seeds" description="Used to increase yucca plant population" amount="50"/>
3    <item name="Food" description="Used to increase yucca moth population" amount="50"/>
4    </vendorInventory>
```

Explain usage in your project: I have 3 Lists that get their information from XML files. The first method exists within the Entity class. It creates a “tempEntities” list that gathers the Name, Species, ImagePath, and Amount of each entity in the game. This temporary list is then loaded into the ‘Entities” list within the Environment class, which gets instantiated when the Environment is instantiated at the start of the game. The next method exists within the Player class. It creates a temporary list of the player’s starting inventory; it grabs the items’ names, descriptions, and amounts from the XML file. This list becomes the player’s starting inventory when they are instantiated at the start of the game (in the MainWindow.xaml.cs file) , and then gets written to the “PlayerInventory” text block. The third method works in the same way as the one for the player, however, this one pertains to the vendor.

Interface

Definition: This holds functionalities to be implemented by a class or group. CLasses will inherit an interface to gain access to its functionalities.

Code excerpts from your project:

```
1 reference
public interface Trade
{
    1 reference
    void TradeItems();
}
```

```
3 references
class Player : Person, Trade
{
```

```
1 reference
public void TradeItems()
{
    Person.Buy(item);
    Person.Sell(item);
    throw new NotImplementedException();
}
```

Explain usage in your project: I have created a Trade interface to be used by the Player class. This will allow the player to perform the Buy (add an item to their inventory) and Sell (remove an item from their inventory) methods simultaneously.

Credits

- Environment class code from Prog 201 Data Binding class demo
- Entity class populated with code from Prog201 Event Handlers in-class demo
- Utility class code from Prog 201 in-class demo
- Code for loading from XML files based on Prog 201 Week 9 in-class demo
- Code for WeatherEvent class based on Prog 201 in-class demo

Research

- *https://www.britannica.com/animal/yucca-moth*
- *https://www.gardeningknowhow.com/ornamental/foliage/yucca/caring-for-and-landscaping-with-yuccas-outdoors.htm*

Instructor Provided Research

Blog.Nature.org. “The Yucca and its Moth.” Cool Green Science: Smarter by Nature. https://blog.nature.org/science/2013/03/21/the-yucca-and-its-moth (Accessed February 4, 2022).

En.wikipedia.org. "Mojave Desert." Wikipedia. https://en.wikipedia.org/wiki/Mojave_Desert (Accessed February 4, 2022).

Fs.fed.us. “Fire Effects Information System (FEIS).” Forest Service U.S. Department of Agriculture. https://www.fs.fed.us/database/feis/plants/shrub/yucsch/all.html (Accessed February 4, 2022).

Fs.fed.us. “Yucca Moths (Tegeticula sp.).” Forest Service U.S. Department of Agriculture. https://www.fs.fed.us/wildflowers/pollinators/pollinator-of-the-month/yucca_moths.shtml (Accessed February 4, 2022).

Ir.lib.uwo.ca. “Evolutionary GEM: Coevolution of Yuccas and Yucca Moths.” Western Libraries. https://ir.lib.uwo.ca/wurjhns/vol8/iss1/10 (Accessed February 4, 2022).

Nwf.org. “Yucca Moths.” The National Wildlife Federation. https://www.nwf.org/Educational-Resources/Wildlife-Guide/Invertebrates/Yucca-Moths (Accessed February 4, 2022).

WorldWildlife.org. “Mojave Desert.” World Wildlife Fund. https://www.worldwildlife.org/ecoregions/na1308 (Accessed February 4, 2022).