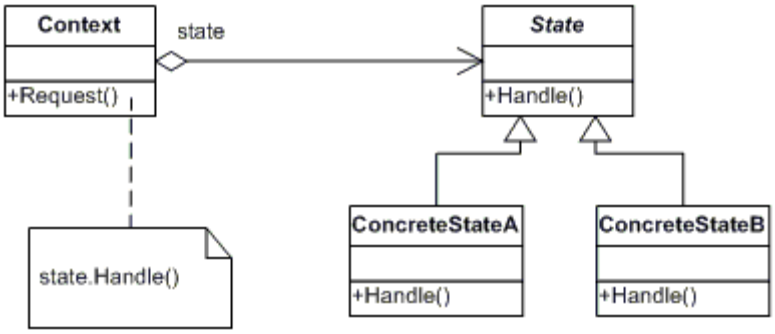
The State Pattern
Design Patterns
Andreya Grzegorzewski – Ohio Northern University
Fall 2016-2017

Introduction

This assignment requires an application that makes use of the state pattern. This pattern allows an object to alter its behavior based on the state that the object is in. My app is a girlfriend simulator. The girlfriend object has variables to represent her hunger, happiness, and anger. These variables are used to determine the girlfriend's state; she may be happy, sad, mad, or feeling fat. The user can choose to feed, antagonize, compliment, or ignore his girlfriend, and she will react differently to each of those actions based on her state. After each action, a label is updated displaying the girlfriend's mood as well as her hunger, happiness, and anger levels.

The UML Diagram



To the left is the UML diagram for the state pattern, courtesy of dofactory.com. The Context object contains a reference to an abstract State object, which can handle requests from the Context class. The State object may be instantiated with an object from any one of the concrete state classes. Below is a table displaying the classes I used to implement this pattern.

Context	My Context class is called Girlfriend. It contains a State object and handles state changes. The Girlfriend class contains the methods feed, compliment, antagonize, and ignore described above.
State	My State interface is an abstract class called Mood. Mood creates prototypes to handle the feed, compliment, antagonize, and ignore methods defined in the Girlfriend class.
Concrete States	My Concrete States inherit from the Mood class. They are Happy, Sad, Mad, and FeelsFat. The girlfriend handles actions differently based on her state, and the concrete state is changed based on the girlfriend's hunger, happiness, and anger levels.

Narrative and Code

I started this code by creating the Mood class, which is shown below in its entirety.

```
public abstract class Mood // This is the State interface
{
    /* Each of the following methods returns an array of 3 ints.
     * The first one represents the change in hunger,
     * the second is the change in happiness,
     * and the third is the change in anger.
     * Hunger, happiness, and anger are all represented
     * in the Girlfriend class as ints in the range [0, 10].
     */

    public abstract int[] reactToBeingFed();
    public abstract int[] reactToCompliment();
    public abstract int[] reactToBeingAntagonized();
    public abstract int[] reactToBeingIgnored();
}
```

Next, I wrote the four concrete state classes. These are all very similar, so only one will be shown here. The complete code for this app can be found in my Design-Patterns repository on GitHub at https://github.com/andreya-grzegorzewski/Design-Patterns.

```
public class Happy : Mood // This is a concrete state
{
    public override int[] reactToBeingAntagonized()
    {
        return new int[3] { 1, -1, 1 };
    }

    public override int[] reactToBeingFed()
    {
        return new int[3] { -2, 2, -2 };
    }

    public override int[] reactToBeingIgnored()
    {
        return new int[3] { 1, -1, 1 };
    }

    public override int[] reactToCompliment()
    {
        return new int[3] { 1, 4, -4 };
    }

    public override string ToString()
    {
        return "is happy";
    }
```

As described in the Mood class, the three ints returned in each function represent, in order, the change to hunger, the change to happiness, and the change to anger. For

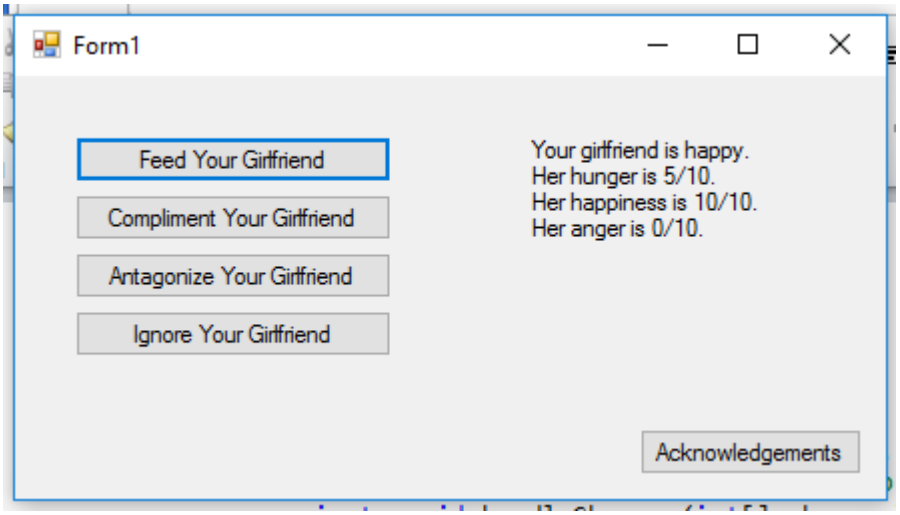
example, in the reactToBeingAntagonized method, hunger is increased by one (because time has passed), happiness is decreased by one, and anger is increased by one.

Once I wrote the four Concrete State classes, I moved on to the Context class, which is called Girlfriend in my app. As shown in the UML diagram, the Girlfriend class contains a pointer to a Mood object. Additionally, it defines three variables for hunger, happiness, and anger, it overrides the ToString method, and it handles changes of state. The class is shown below.

```
class Girlfriend // This is the Context class
{
    // Start off not too hungry; very happy; not mad. Let's see how long it lasts.
    private int hunger = 5;     // 0 is very full; 10 is very hungry
    private int happiness = 10; // 0 is very sad; 10 is very happy
    private int anger = 0;      // 0 is not angry; 10 is very angry

    private Mood mood = new Happy();

    public void feed()
    {
        int[] changes = mood.reactToBeingFed();
        handleChanges(changes);
        switchMood();
    }

    public void compliment()
    {
        int[] changes = mood.reactToCompliment();
        handleChanges(changes);
        switchMood();
    }

    public void antagonize()
    {
        int[] changes = mood.reactToBeingAntagonized();
        handleChanges(changes);
        switchMood();
    }

    public void ignore()
    {
        int[] changes = mood.reactToBeingIgnored();
        handleChanges(changes);
        switchMood();
    }

    public override string ToString()
    {
        return "Your girlfriend " + mood.ToString()
            + ".\nHer hunger is " + hunger + "/10.\nHer happiness is "
            + happiness + "/10.\nHer anger is " + anger + "/10.";
    }
```

```
        private void switchMood()
        {
            if (hunger <= 3)
                mood = new FeelsFat();
            else if (happiness <= 3)
                mood = new Sad();
            else if (anger >= 7)
                mood = new Mad();
            else if (happiness >= 7)
                mood = new Happy();
        }

        // Handles changes to hunger, happiness, and anger.
        // The array of ints contains changes to each of those variables, in that order.
        private void handleChanges(int[] changes)
        {
            hunger += changes[0];
            happiness += changes[1];
            anger += changes[2];

            if (hunger > 10)
                hunger = 10;
            if (hunger < 0)
                hunger = 0;
            if (happiness > 10)
                happiness = 10;
            if (happiness < 0)
                happiness = 0;
            if (anger > 10)
                anger = 10;
            if (anger < 0)
                anger = 0;
        }
    }
```

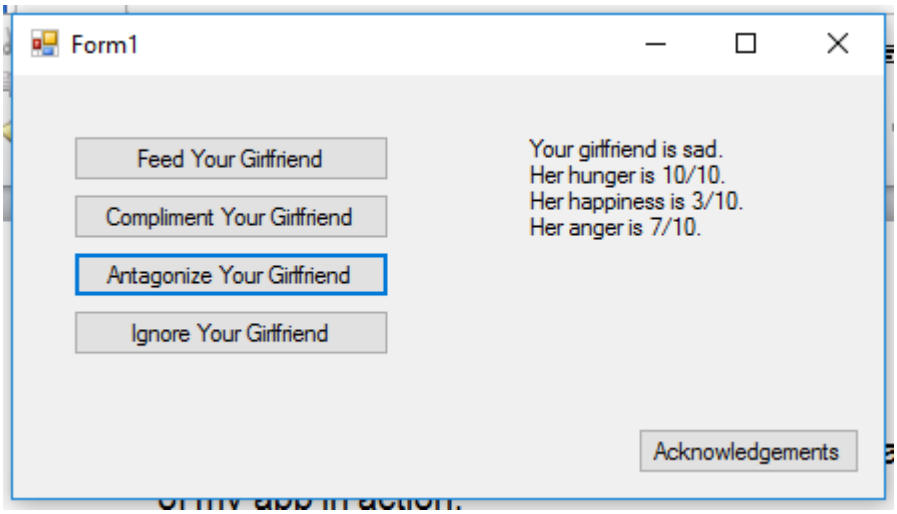
Finally, I designed my form and wrote my form code. The form as shown upon starting the app is shown below.



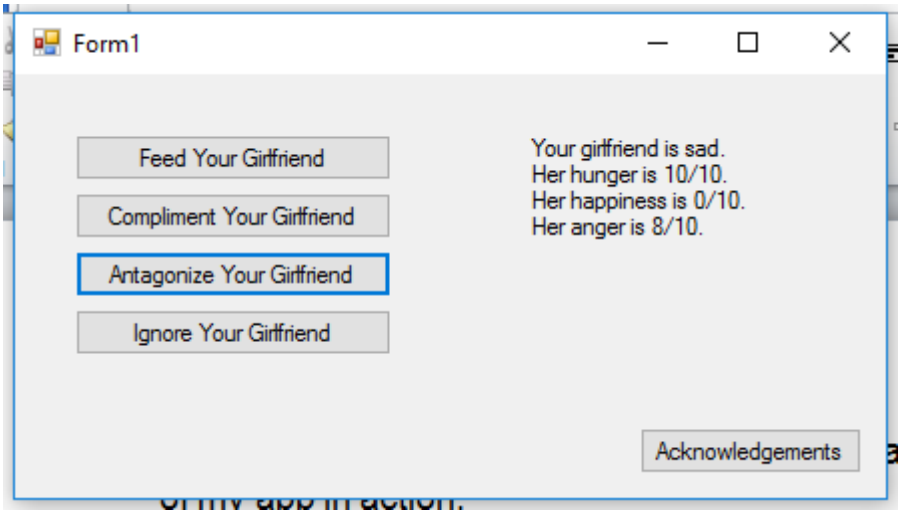
The user can click any of the four action buttons on the left side of the form, and the results will be shown in the label to the right of the buttons. With my form designed, I started writing the form code. I started by creating a Girlfriend object, which I named Andreya. In the Form1 constructor, I set up the state label. Finally, I handled the button click events. The code is shown below.

```
public partial class Form1 : Form
{
    Girlfriend Andreya = new Girlfriend();

    public Form1()
    {
        InitializeComponent();
        girlfriendStateLabel.MaximumSize = new Size(400, 0);
        girlfriendStateLabel.AutoSize = true;

        girlfriendStateLabel.Text = Andreya.ToString();
    }

    private void feedButton_Click(object sender, EventArgs e)
    {
        Andreya.feed();
        girlfriendStateLabel.Text = Andreya.ToString();
    }

    private void antagonizeButton_Click(object sender, EventArgs e)
    {
        Andreya.antagonize();
        girlfriendStateLabel.Text = Andreya.ToString();
    }

    private void complimentButton_Click(object sender, EventArgs e)
    {
        Andreya.compliment();
        girlfriendStateLabel.Text = Andreya.ToString();
    }

    private void ignoreButton_Click(object sender, EventArgs e)
    {
        Andreya.ignore();
        girlfriendStateLabel.Text = Andreya.ToString();
    }

    private void acknowledgementsButton_Click(object sender, EventArgs e)
    {
        string message = "This app is dedicated to my boyfriend,\n"
            + "who has learned how to play this game in real life.\n"
            + "I wish the best of luck to those of you who have not mastered that
              ability yet.";
        MessageBox.Show(message, "Acknowledgements");
    }
}
```

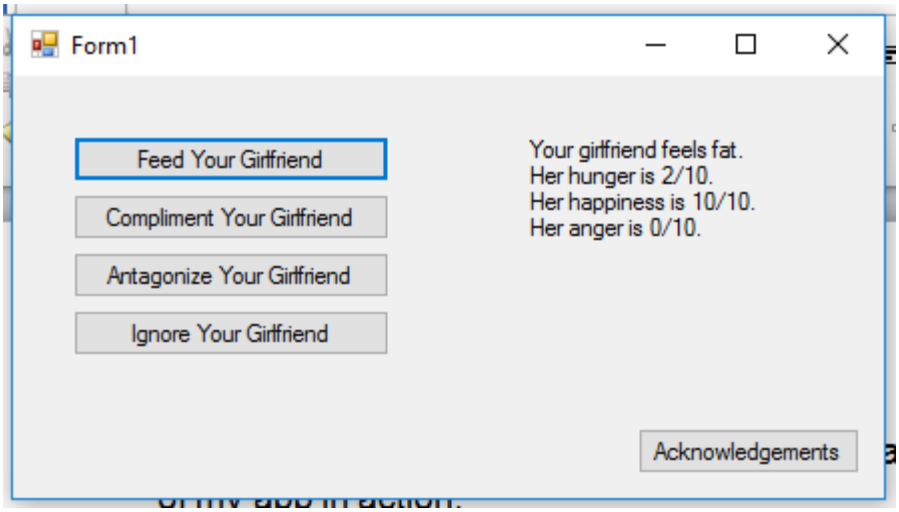
This concludes the code for my girlfriend simulator app. Below are several screenshots of my app in action.



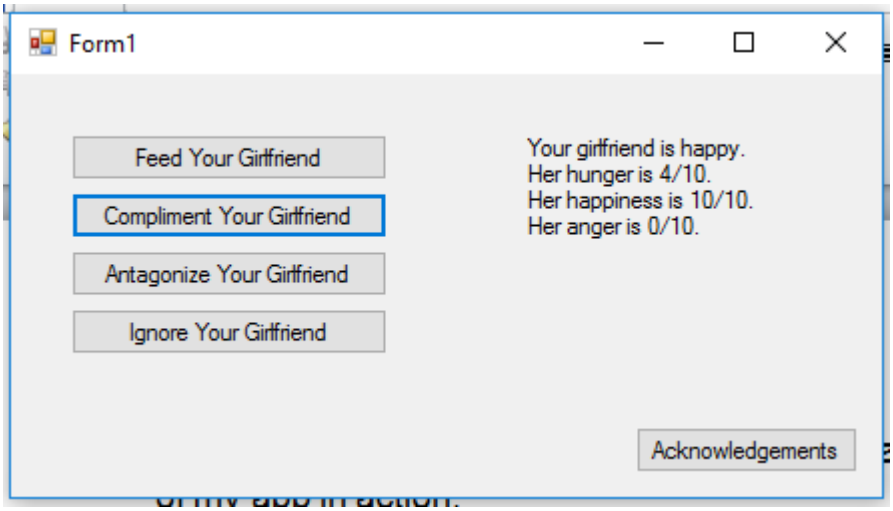
In this screenshot, I antagonized my girlfriend until she became sad. This took seven clicks of the Antagonize button. Each click increased anger by one and decreased happiness by one.



After she was in the sad state, I antagonized her one more time. This time, her anger increased by one and her happiness decreased by three. This shows that the same actions are handled differently based on what state the girlfriend object is in.



In this screenshot, I complimented my girlfriend several times until she was happy, and then I fed her until her hunger was very low and she felt fat.



Finally, I complimented her until her hunger increased enough that she no longer felt fat; instead, she became happy.

Observations

The hardest part about this program was coming up with an idea. Once I decided that I would do this, the coding was very simple and straightforward, and I didn't have any problems with functionality. This pattern makes a lot of sense to me, and I could see it being used a lot. When I was doing some research about the state pattern, I found that it could be used in Pacman, and from there I could understand lots of other games that could make use of the state pattern; for example, in some MMORPGs, battle pets can be in an aggressive state, a defensive state, or a passive state. I find it very interesting to see how the state pattern can apply so well to games that I have played for years.