Design Patterns
Prof. Retterer

Jeremy Martin
10/20/16

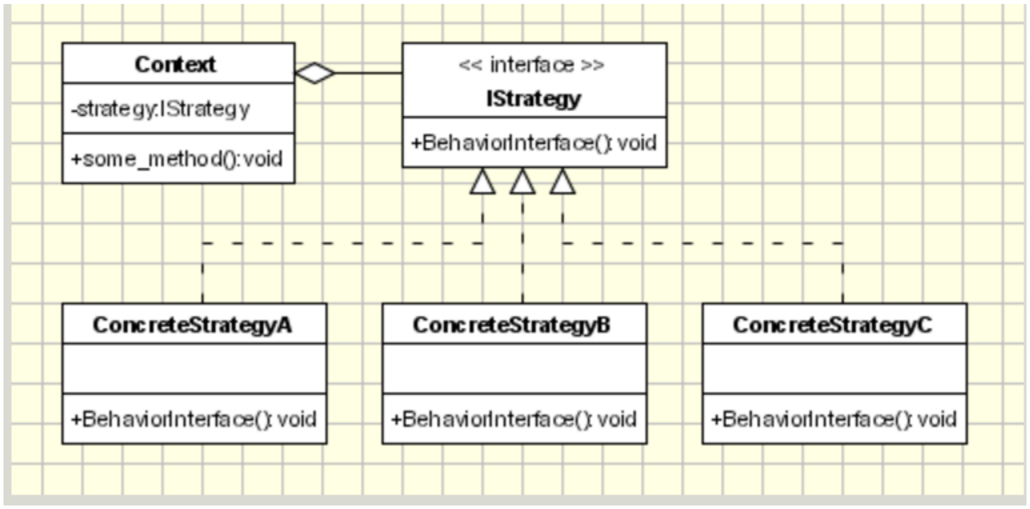
Strategy Pattern

Introduction

For this project, we were tasked with making a program that demonstrated the Strategy pattern. For my project, I decided to make a program that would make different strategies for a RPG party based on which button was pressed, or a random composition when the random button was pressed.

Strategy UML

To the right, you will find a picture of the UML diagram for the Strategy pattern.



Below, you will find the code that was used to make the program.

First we will start with the abstract class Party_Tactics.

```
public abstract class Party_Tactics
    {
        public abstract void partyCommand();
    }
```

The next pieces of code are all implementing Party_Tactics, they are Power_Move, Normal_Nature, and Utility_Mode classes.

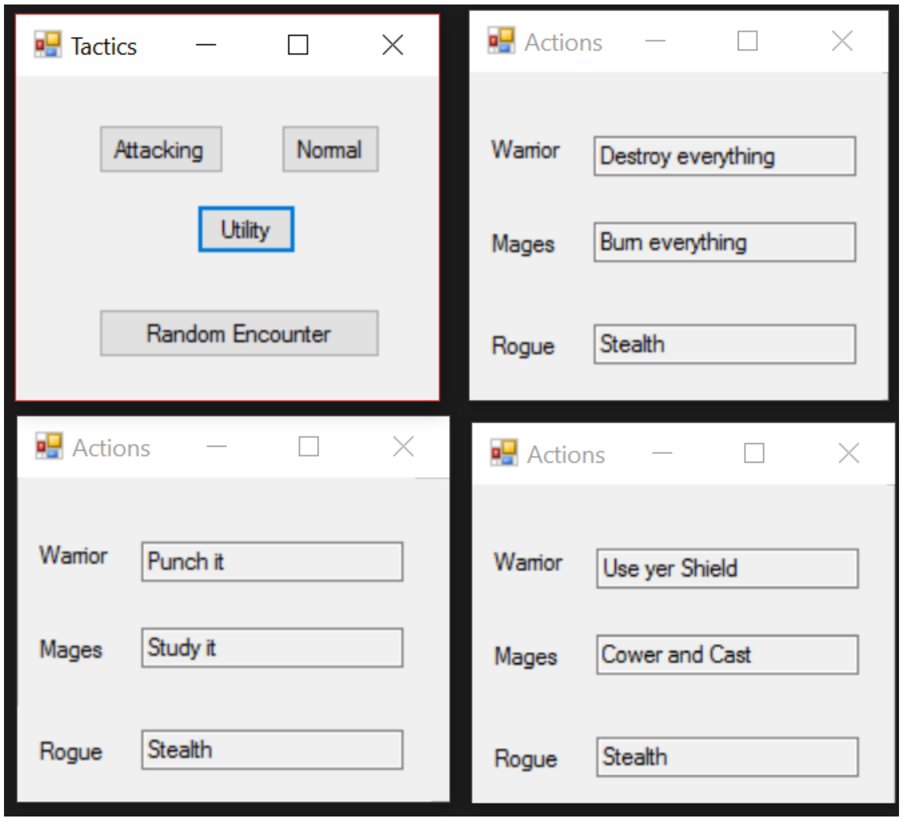
```
public class Power_Move : Party_Tactics
    {
        Form2 f2 = new Form2();

        public override void partyCommand()
        {
            f2.m_tbWarrior.Text = "Destroy everything";
            f2.m_tbMages.Text = "Burn everything";
            f2.m_tbRogue.Text = "Stealth";
            f2.Visible = true;
        }
    }

public class Normal_Nature : Party_Tactics
    {
        Form2 f2 = new Form2();

        public override void partyCommand()
        {
            f2.m_tbWarrior.Text = "Punch it";
            f2.m_tbMages.Text = "Study it";
            f2.m_tbRogue.Text = "Stealth";
            f2.Visible = true;
        }
    }

public class Utility_Mode : Party_Tactics
    {
        Form2 f2 = new Form2();

        public override void partyCommand()
        {
            f2.m_tbWarrior.Text = "Use yer Shield";
            f2.m_tbMages.Text = "Cower and Cast";
            f2.m_tbRogue.Text = "Stealth";
            f2.Visible = true;
        }
    }
```

The last piece of code is for the form.

```
public partial class Form1 : Form
    {
        Power_Move pm;
        Normal_Nature nn;
        Utility_Mode um;
        Random rand = new Random();

        public Form1()
        {
            InitializeComponent();
            pm = new Power_Move();
            nn = new Normal_Nature();
            um = new Utility_Mode();
        }

        private void m_btnAttack_Click(object sender, EventArgs e)
        {
            pm.partyCommand();
        }

        private void m_btnNormal_Click(object sender, EventArgs e)
        {
            nn.partyCommand();
        }

        private void m_btnUtility_Click(object sender, EventArgs e)
        {
            um.partyCommand();
        }

        private void m_btnRandom_Click(object sender, EventArgs e)
        {
            int randNum;
            randNum=rand.Next() % 10;

            if(randNum<=3)
```

```
                {
                    um.partyCommand();
                }
                if(randNum>3 && randNum<=6)
                {
                    nn.partyCommand();
                }
                if(randNum>7 && randNum<=9)
                {
                    pm.partyCommand();
                }
            }
        }
```

Here is a picture showing all of the possible combinations of tactics that are created from the program.



Warrior	Destroy everything
Mages	Burn everything
Rogue	Stealth

Warrior	Punch it
Mages	Study it
Rogue	Stealth

Warrior	Use yer Shield
Mages	Cower and Cast
Rogue	Stealth

Conclusion

Overall, this was an interesting pattern to make. It seems like a very simple pattern to implement into quite a variety of programs. The strategy pattern seems to be a perfect example of polymorphism, each object has its own way of taking care of itself by using a method similar to the other objects related to it.