Design Patterns
Prof. Retterer

Jeremy Martin
11/29/16

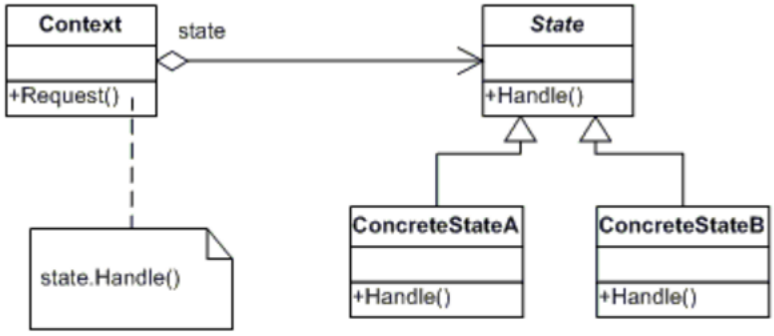
State Pattern

Introduction

For this project, we were tasked with making an application that demonstrated the State Pattern. My application shows the different states that you can be under during a fight in a role-playing game. The different states are Normal, Stunned, Poisoned, Burned, and Paralyzed. Each state has a different way of calculating an attack on the enemy.

UML Diagram

To the right, you will see the UML diagram for the State Pattern.



The State class in my code is the State class shown on the diagram. The five different states that the character can be in are all the ConcreteState classes on the diagram. The Form code handles the Context class.

Now, we will move on to the code for the application. All the ConcreteState classes handle the handleAttack method differently. But I will only show a few of them. First, we will start out with the State class.

```
public abstract class State // this is the State class
    {
        public abstract string handleAttack();
    }
```

Now we will move onto the ConcreteState classes. The Normal state just picks a random number between 1-100 to act as the damage to the enemy.

```
public class Normal : State // these are the ConcreteState classes
    {
        Random rand = new Random();
        int Attack;

        public override string handleAttack()
        {
            Attack = rand.Next() % 100 + 1;
            return "You deal " + Attack.ToString() + " damage to the enemy.";
        }
    }
```

The Poisoned and Burned states pick a random number between 1-100 to act as the damage. On top of this there is a method that is called along with these to determine the damage that is done to your character, this method is found in the Form1 code.

```
public class Poisoned : State
    {
```

```
        Random rand = new Random();
        int Attack;

        public override string handleAttack()
        {
            Attack = rand.Next() % 100 + 1;
            return "You deal " + Attack.ToString() + " damage to the enemy.";
        }
    }
```

The final ConcreteState class that we will look at is the Paralyzed class. This class has a random number between 1-100 that is assigned to AttackChance, if the number is bigger than 75, then the attack will actually go through.

```
public class Paralyzed : State
    {
        Random rand = new Random();
        int Attack;
        int AttackChance;

        public override string handleAttack()
        {
            AttackChance = rand.Next() % 100 + 1;

            if(AttackChance>75)
            {
                Attack = rand.Next() % 100 + 1;
                return "You deal " + Attack.ToString() + " damage to the enemy.";
            }
            else
            {
                return "You are paralyzed and cannot attack this turn.";
            }
        }
    }
```

Finally, we will move onto the Form1 code. This code checks to see which of the radio buttons is checked to determine which state the character is in. If the health variable is reduced below 0, the text box will only display that you have died and prompt you to try again. There is also the calculateDamage method that I mentioned earlier for the Poisoned and Burned classes.

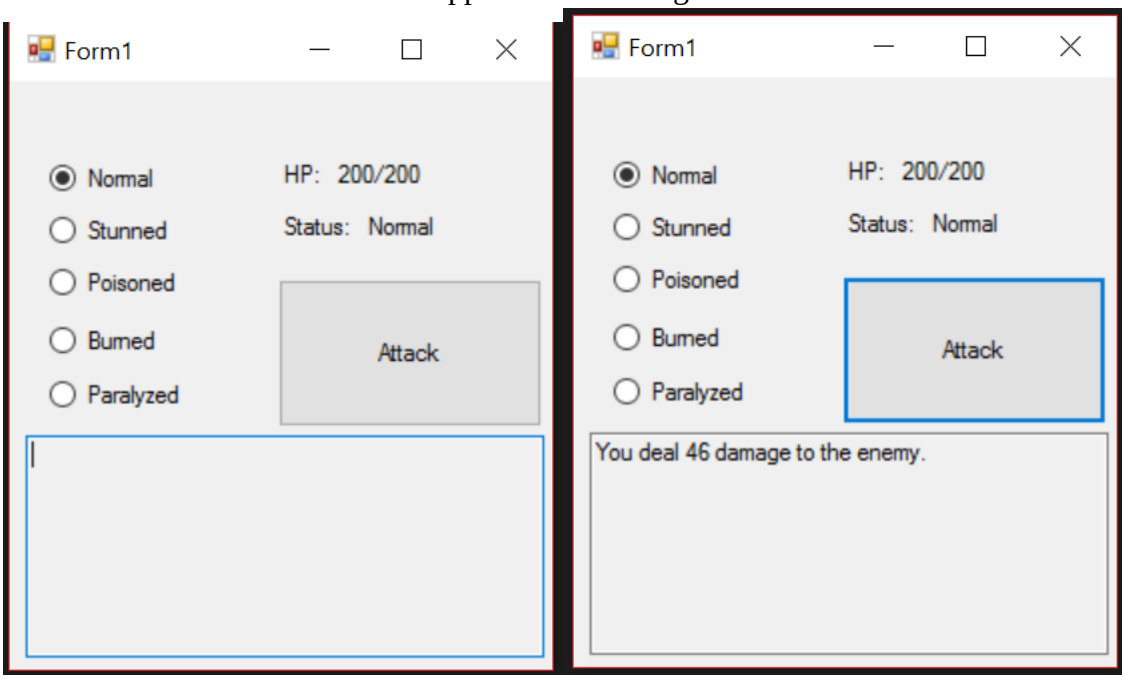
```
public partial class Form1 : Form
    {
        string State;
        State s;
        Random rand = new Random();
        int Health;
        int Damage;

        public Form1()
        {
            InitializeComponent();
            m_lblHealth.Text = "HP:   200/200";
            m_lblStatus.Text = "Status:   Normal";
            Health = 200;
        }
```

State Pattern

```
        private void m_btnAttack_Click(object sender, EventArgs e)
        {
            if(m_rbNormal.Checked)
            {
                s = new Normal();
                State = "Normal";
                m_lblStatus.Text = "Status:    " + State;
                m_tbBattleLog.Text = s.handleAttack() + Environment.NewLine +
m_tbBattleLog.Text;
            }
            else if(m_rbStunned.Checked)
            {
                s = new Stunned();
                State = "Stunned";
                m_lblStatus.Text = "Status:    " + State;
                m_tbBattleLog.Text = s.handleAttack() + Environment.NewLine +
m_tbBattleLog.Text;
            }
            else if(m_rbPoisoned.Checked)
            {
                s = new Poisoned();
                State = "Poisoned";
                m_lblStatus.Text = "Status:    " + State;
                m_tbBattleLog.Text = s.handleAttack() + "You take " +
calculateDamage() + " damage." + Environment.NewLine + m_tbBattleLog.Text;
                if (Health <= 0)
                {
                    State = "Dead";
                    m_lblStatus.Text = "Status:    " + State;
                    m_tbBattleLog.Text = "You have died. Try again mortal!";
                }
            }
            else if(m_rbBurned.Checked)
            {
                s = new Burned();
                State = "Burned";
                m_lblStatus.Text = "Status:    " + State;
                m_tbBattleLog.Text = s.handleAttack() +"You take "+
calculateDamage()+" damage." + Environment.NewLine + m_tbBattleLog.Text;
                if (Health <= 0)
                {
                    State = "Dead";
                    m_lblStatus.Text = "Status:    " + State;
                    m_tbBattleLog.Text = "You have died. Try again mortal!";
                }
            }
            else if(m_rbParalyzed.Checked)
            {
                s = new Paralyzed();
                State = "Paralyzed";
                m_lblStatus.Text = "Status:    " + State;
                m_tbBattleLog.Text = s.handleAttack() + Environment.NewLine +
m_tbBattleLog.Text;
            }

            if(State=="Dead")
            {
                m_tbBattleLog.Text = "You can't do that while you're dead." +
Environment.NewLine + m_tbBattleLog.Text;
```

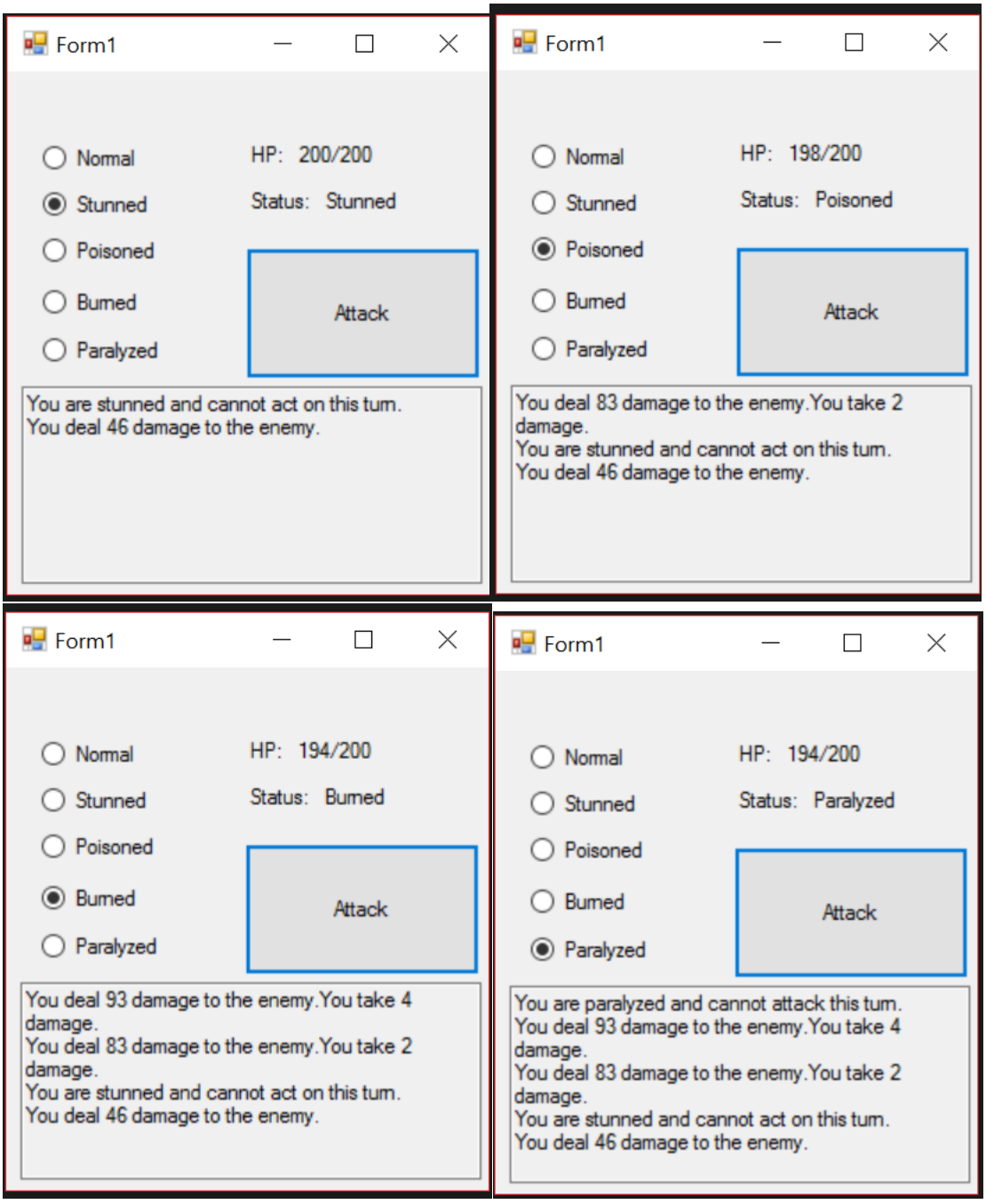
```
            }
        }

        private string calculateDamage()
        {
            if(State=="Poisoned")
            {
                Damage = rand.Next() % 20 + 1;
            }
            if(State=="Burned")
            {
                Damage = rand.Next() % 10 + 1;
            }

            Health -= Damage;
            if (Health>-19)
            {
                m_lblHealth.Text = "HP:   " + Health.ToString() + "/200";
            }
            return Damage.ToString();
        }
    }
}
```

Here are some screenshots of the application working.



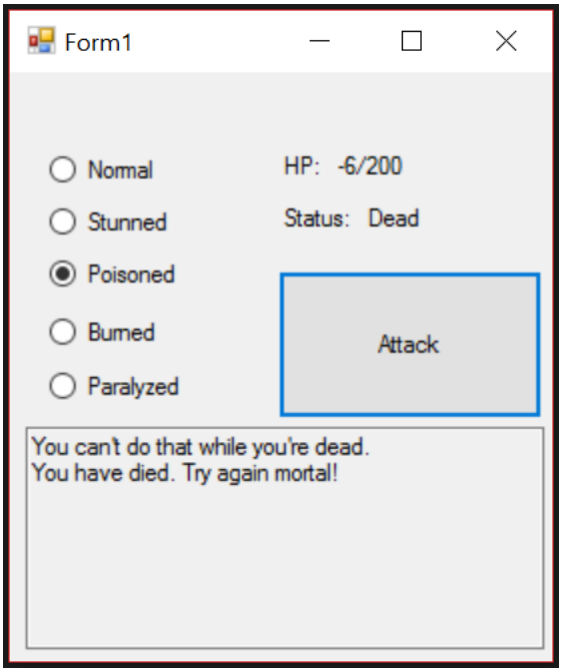
Jeremy Martin

State Pattern



Jeremy Martin

State Pattern



Conclusion

Overall, this was an easy pattern to make. The only real problem that I ran into, was the fact that I couldn't modify the labels in the form from the classes. That is why I had to make the calculateDamage method in the Form1 code, the original application had the Poisoned and Burned classes calculating their own damage.