Design Patterns
Prof. Retterer
Facade Pattern

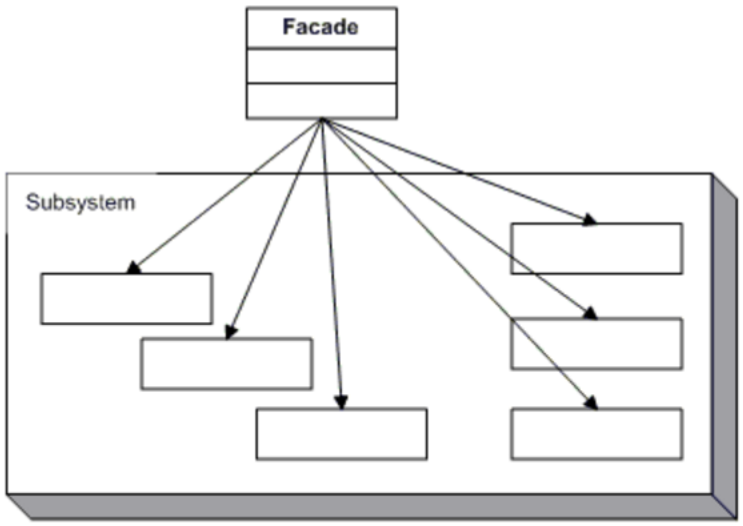
Jeremy Martin
9/13/16

Introduction

This assignment asked us to make a simple application that demonstrated the Facade Pattern. The facade pattern is a class within the program that has the ability to change multiple subclasses and change things within these subclasses. My program uses this pattern to create a simple tactics engine for AI characters within a turn based role playing game.

UML Diagram for Facade

The UML diagram on the right shows the classes that are needed within the program. The diagram here is very generic due to the fact that you only need to have one class that is able to control the workings of the other classes. In my program, I have chosen Facade to be the facade class and it will be controlling aspects from the Warrior class, the Rogue class, the White_Mage class, and the Black_Mage class.



Now I will start showing the code that I used. Below is the code for the Facade class.

```
public delegate void TacticsStateChangedEventHandler(object sender, EventArgs e);

     public class Facade //this is the facade class
    {
        public enum Warrior {Attack,Defend,Abilites,Escape}
        public enum Rogue {Attack,Defend,Steal,Escape}
        public enum White_Mage {Whtie_Magic,Heal,Buff}
        public enum Black_Mage {Black_Magic,Defend,Debuff,Escape}
        private Warrior _WarriorState;
        private Rogue _RogueState;
        private White_Mage _WhiteMageState;
        private Black_Mage _BlackMageState;

        public Warrior m_WarriorState
        {
            get { return _WarriorState; }
            set
            {
                _WarriorState = value;
                if (TacticsStateChanged != null)
                    TacticsStateChanged(this, new EventArgs());
            }
        }
        public Rogue m_RogueState
        {
            get { return _RogueState; }
            set
            {
                _RogueState = value;
                if (TacticsStateChanged != null)
                    TacticsStateChanged(this, new EventArgs());
            }
        }

        public White_Mage m_WhiteMageState
        {
            get { return _WhiteMageState; }
            set
            {
                _WhiteMageState = value;
                if (TacticsStateChanged != null)
                    TacticsStateChanged(this, new EventArgs());
            }
        }

        public Black_Mage m_BlackMageState
        {
            get { return _BlackMageState; }
            set
            {
                _BlackMageState = value;
                if (TacticsStateChanged != null)
                    TacticsStateChanged(this, new EventArgs());
            }
        }

        public event TacticsStateChangedEventHandler TacticsStateChanged;

        public bool Offensive()
        {
            m_WarriorState = Warrior.Attack;
            m_RogueState = Rogue.Attack;
            m_WhiteMageState = White_Mage.Whtie_Magic;
            m_BlackMageState = Black_Mage.Black_Magic;

            return true;
        }

        public bool Defensive()
        {
            m_WarriorState = Warrior.Defend;
            m_RogueState = Rogue.Defend;
            m_WhiteMageState = White_Mage.Heal;
            m_BlackMageState = Black_Mage.Defend;

            return true;
        }

        public bool Utility()
        {
            m_WarriorState = Warrior.Abilites;
            m_RogueState = Rogue.Steal;
            m_WhiteMageState = White_Mage.Buff;
            m_BlackMageState = Black_Mage.Debuff;

            return true;
        }

        public bool Survive()
        {
            m_WarriorState = Warrior.Escape;
            m_RogueState = Rogue.Escape;
            m_WhiteMageState = White_Mage.Heal;
            m_BlackMageState = Black_Mage.Escape;

            return true;
        }
    }
```

The next pieces of code come from the Warrior, Rogue, White_Mage, and Black_Mage classes.

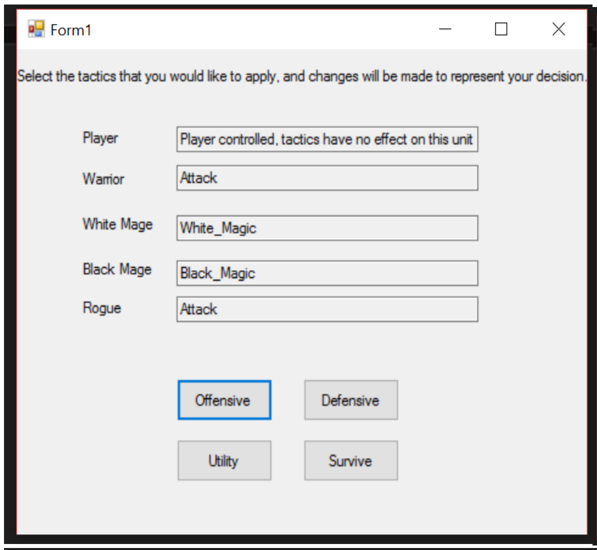
```
public delegate void WarriorStateChangedEventHandler(object sender, EventArgs e);

     public class Warrior
    {
        public event WarriorStateChangedEventHandler WarriorStateChanged;
        public enum WarriorState {Attack,Defend,Abilities,Escape}

        private WarriorState _WarriorState;

        public WarriorState m_WarriorState
        {
            get
            {
                return _WarriorState;
            }
            set
            {
                _WarriorState = value;
                if (WarriorStateChanged != null)
                    WarriorStateChanged(this, new EventArgs());
            }
        }
    }


    public delegate void RogueStateChangedEventHandler(object sender, EventArgs
e);

     public class Rogue
    {
        public event RogueStateChangedEventHandler RogueStateChanged;
        public enum RogueState { Attack, Defend, Steal, Escape }

        private RogueState _RogueState;

        public RogueState m_RogueState
        {
            get
            {
                return _RogueState;
            }
            set
            {
                _RogueState = value;
                if (RogueStateChanged != null)
                    RogueStateChanged(this, new EventArgs());
            }
        }
    }

public delegate void WhiteMageStateChangedEventHandler(object sender, EventArgs
e);

     public class White_Mage
    {
        public event WhiteMageStateChangedEventHandler WhiteMageStateChanged;
        public enum WhiteMageState { White_Magic,Heal,Buff}

        private WhiteMageState _WhiteMageState;

        public WhiteMageState m_WhiteMageState
        {
            get
            {
                return _WhiteMageState;
            }
            set
            {
                _WhiteMageState = value;
                if (WhiteMageStateChanged != null)
                    WhiteMageStateChanged(this, new EventArgs());
            }
        }
    }

public delegate void BlackMageStateChangedEventHandler(object sender, EventArgs
e);

     public class Black_Mage
    {
        public event BlackMageStateChangedEventHandler BlackMageStateChanged;
        public enum BlackMageState { Black_Magic,Debuff,Defend,Escape }

        private BlackMageState _BlackMageState;

        public BlackMageState m_BlackMageState
        {
            get
            {
                return _BlackMageState;
            }
            set
            {
                _BlackMageState = value;
                if (BlackMageStateChanged != null)
                    BlackMageStateChanged(this, new EventArgs());
            }
        }
    }
```

The last piece of code comes from the form that is displayed to the user.

```
public partial class Form1 : Form
    {
        Warrior war;
        Rogue rg;
        White_Mage wm;
        Black_Mage bm;

        public Form1()
        {
            InitializeComponent();
            war = new Warrior();
            rg = new Rogue();
            wm = new White_Mage();
            bm = new Black_Mage();
            war.WarriorStateChanged += new
WarriorStateChangedEventHandler(war_WarriorStateChanged);
            rg.RogueStateChanged += new
RogueStateChangedEventHandler(rg_RogueStateChanged);
            wm.WhiteMageStateChanged += new
WhiteMageStateChangedEventHandler(wm_WhiteMageStateChanged);
            bm.BlackMageStateChanged += new
BlackMageStateChangedEventHandler(bm_BlackMageStateChanged);
        }

        void war_WarriorStateChanged(object sender, EventArgs e)
        {
            UpdateStatus();
        }

        void rg_RogueStateChanged(object sender, EventArgs e)
        {
            UpdateStatus();
        }

        void wm_WhiteMageStateChanged(object sender,EventArgs e)
        {
            UpdateStatus();
        }

        void bm_BlackMageStateChanged(object sender,EventArgs e)
        {
            UpdateStatus();
        }

        public void UpdateStatus()
        {
            m_tbWarrior.Text = war.m_WarriorState.ToString();
            m_tbRogue.Text = rg.m_RogueState.ToString();
            m_tbWhiteMage.Text = wm.m_WhiteMageState.ToString();
            m_tbBlackMage.Text = bm.m_BlackMageState.ToString();
        }

        private void m_btnOffense_Click(object sender, EventArgs e)
        {
            war.m_WarriorState = Warrior.WarriorState.Attack;
            rg.m_RogueState = Rogue.RogueState.Attack;
            wm.m_WhiteMageState = White_Mage.WhiteMageState.White_Magic;
            bm.m_BlackMageState = Black_Mage.BlackMageState.Black_Magic;
        }

        private void m_btnDefense_Click(object sender, EventArgs e)
        {
            war.m_WarriorState = Warrior.WarriorState.Defend;
            rg.m_RogueState = Rogue.RogueState.Defend;
            wm.m_WhiteMageState = White_Mage.WhiteMageState.Heal;
            bm.m_BlackMageState = Black_Mage.BlackMageState.Defend;
        }

        private void m_btnUtility_Click(object sender, EventArgs e)
        {
            war.m_WarriorState = Warrior.WarriorState.Abilities;
            rg.m_RogueState = Rogue.RogueState.Steal;
            wm.m_WhiteMageState = White_Mage.WhiteMageState.Buff;
            bm.m_BlackMageState = Black_Mage.BlackMageState.Debuff;
        }

        private void m_btnSurvive_Click(object sender, EventArgs e)
        {
            war.m_WarriorState = Warrior.WarriorState.Escape;
            rg.m_RogueState = Rogue.RogueState.Escape;
            wm.m_WhiteMageState = White_Mage.WhiteMageState.Heal;
            bm.m_BlackMageState = Black_Mage.BlackMageState.Escape;
        }
    }
```

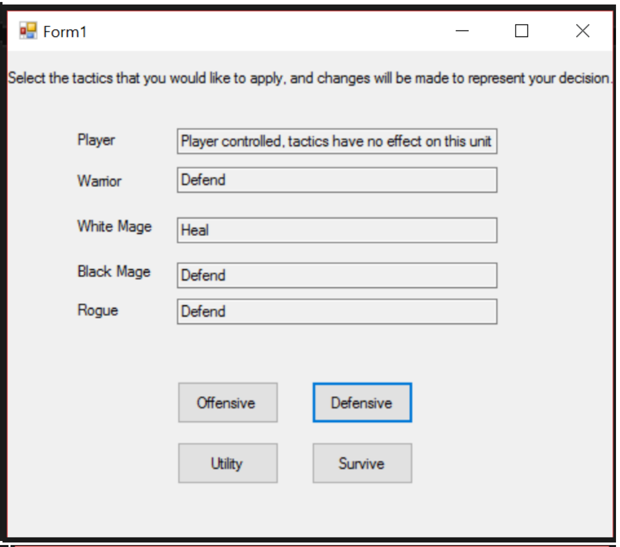
Now here are some screenshots of my code working. The first picture shows the tactics for pressing the “Offensive” button, the second shows the “Defensive” button, the third shows the “Utility” button, and the fourth one shows the “Survive” button.

Design Patterns Jeremy Martin
Prof. Retterer 9/13/16
Facade Pattern


Form1

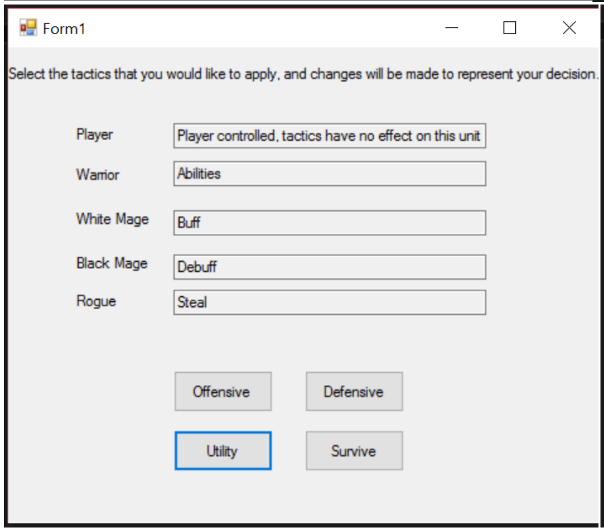
Select the tactics that you would like to apply, and changes will be made to represent your decision.

Player	Player controlled, tactics have no effect on this unit
Warrior	Attack
White Mage	White_Magic
Black Mage	Black_Magic
Rogue	Attack


Form1

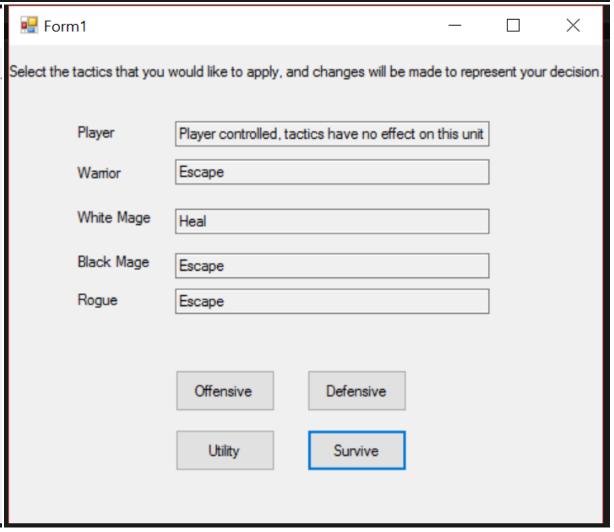
Select the tactics that you would like to apply, and changes will be made to represent your decision.

Player	Player controlled, tactics have no effect on this unit
Warrior	Defend
White Mage	Heal
Black Mage	Defend
Rogue	Defend


Form1

Select the tactics that you would like to apply, and changes will be made to represent your decision.

Player	Player controlled, tactics have no effect on this unit
Warrior	Abilities
White Mage	Buff
Black Mage	Debuff
Rogue	Steal


Form1

Select the tactics that you would like to apply, and changes will be made to represent your decision.

Player	Player controlled, tactics have no effect on this unit
Warrior	Escape
White Mage	Heal
Black Mage	Escape
Rogue	Escape

Conclusion

Overall, I thought this was fun project and I really appreciated the ability to use things related to video games. I had other ideas that I could've used, but this one just felt really natural to me. The UML diagram is clear and to the point about what to do. This was an interesting project and I hope that I am able to work with the code that I have already put together here for the Factory Method pattern.