A Summoner's Tale
Chapter 12
Hulk Smash!

Welcome to my Summoner's Tale tutorial series on creating a Pokémon-inspired game with MonoGame. I'm writing these tutorials for the MonoGame 3.8.1 framework using Visual Studio 2022. The code should work on previous versions of MonoGame and Visual Studio. I plan on creating the editors on macOS and Windows. I'm unfamiliar with Linux, so a few projects may not be done for that platform.

The tutorials will make more sense if they are read in order. You can find the list of tutorials on my web blog, A Summoner's Tale page. In addition to the PDFs, I will make the code for each tutorial available on GitHub here: https://github.com/Synammon/summoners-tale. It will also be included on the page that links to the tutorials.

When last we saw our hero… Oops, wrong story, sorry. Still, our hero was about to smash something. So, I will leave it in.

Kidding aside, in this tutorial I will cover combat in earnest. First up, I want to make two revisions. I want to add a new property to shadow monsters and moves. Replace the MoveData class with the following version.

```
public class MoveData
{
    public string Name { get; set; }
    public int Elements { get; set; }
    public TargetType Target { get; set; }
    public TargetAttribute TargetAttribute { get; set; }
    public Point Mana { get; set; }
    public Point Range { get; set; }
    public int Status { get; set; }
    public int Power { get; set; }
    public bool Hurts { get; set; }
    public bool IsTemporary { get; set; }
}
```

I just added a new property, Power, which is the power of the move. Next, replace the ShadowMonsterData class with this version.

```
public class ShadowMonsterData
{
    public const int Normal = 0;
    public const int Asleep = 1;
    public const int Confused = 2;
    public const int Poisoned = 4;
    public const int Paralyzed = 8;
    public const int Burn = 16;
```

```
        public const int Frozen = 32;

        public string Name { get; set; }
        public int Elements { get; set; }
        public int Level { get; set; }
        public int Experience { get; set; }
        public Point Health { get; set; }
        public int Attack { get; set; }
        public int Defence { get; set; }
        public int SpecialAttack { get; set; }
        public int SpecialDefence { get; set; }
        public int Speed { get; set; }
        public int Accuracy { get; set; }
        public float CriticalRate { get; set; }
        public int AttackMod { get; set; }
        public int DefenceMod { get; set; }
        public int SpecialAttackMod { get; set; }
        public int SpecialDefenceMod { get; set; }
        public int SpeedMod { get; set; }
        public int AccuracyMod { get; set; }
        public int Status { get; set; }
        public List<MoveData> Moves { get; set; }
    }
```

I added a new property CriticalRate, which has much as you expected, the rate at which a shadow monster deals a critical hit. It will range from .01 to .25 normally. A value of .25 is probably too high, causing a critical hit one in four times on average. A better value will probably be between .05 and .10. Some experimentation is in order.

It is now time to implement the battle engine. As I mentioned in previous tutorials, the battle engine is a turn-based battle engine, much like the old Pokémon games. To start, we want to add a new game state. In the SummonersTale project, right-click the GameStates folder, select Add and then Class. Name this new class DamageState. Replace the code with this version.

```
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;
using Psilibrary.ShadowMonsters;
using SummonersTale.Forms;
using SummonersTale.ShadowMonsters;
using System;

namespace SummonersTale.StateManagement
{
    public enum CurrentTurn
    {
        Players, Enemies
    }

    public interface IDamageState
    {
        void SetShadowMonsters(ShadowMonster player, ShadowMonster enemy);
        void SetMoves(MoveData playerMove, MoveData enemyMove);
        void Start();
    }
    public class DamageState : BaseGameState, IDamageState
    {
        #region Field Region

        private CurrentTurn turn;
        private Texture2D combatBackground;
        private Rectangle playerRect;
        private Rectangle enemyRect;
        private TimeSpan cTimer;
        private TimeSpan dTimer;
        private ShadowMonster player;
        private ShadowMonster enemy;
        private MoveData playerMove;
        private MoveData enemyMove;
        private bool first;
        private Rectangle playerBorderRect;
        private Rectangle enemyBorderRect;
        private Rectangle playerMiniRect;
        private Rectangle enemyMiniRect;
        private Rectangle playerHealthRect;
        private Rectangle enemyHealthRect;
        private Rectangle healthSourceRect;
        private float playerHealth;
        private float enemyHealth;
        private Texture2D avatarBorder;
        private Texture2D avatarHealth;
        private Vector2 playerName;
        private Vector2 enemyName;

        #endregion

        #region Property Region
        #endregion

        #region Constructor Region

        public DamageState(Game game)
            : base(game)
        {
            Game.Services.AddService<IDamageState>(this);

            playerRect = new Rectangle(10, 90, 300, 300);
            enemyRect = new Rectangle(Settings.BaseWidth - 310, 10, 300, 300);

            playerBorderRect = new Rectangle(10, 10, 300, 75);
            enemyBorderRect = new Rectangle(Settings.BaseWidth - 310, 320, 300, 75);

            healthSourceRect = new Rectangle(10, 50, 290, 20);
            playerHealthRect = new Rectangle(playerBorderRect.X + 12,
playerBorderRect.Y + 52, 286, 16);
            enemyHealthRect = new Rectangle(enemyBorderRect.X + 12,
enemyBorderRect.Y + 52, 286, 16);

            playerMiniRect = new Rectangle(playerBorderRect.X + 11,
playerBorderRect.Y + 11, 28, 28);
            enemyMiniRect = new Rectangle(enemyBorderRect.X + 11, enemyBorderRect.Y
+ 11, 28, 28);
```

```
            playerName = new Vector2(playerBorderRect.X + 55, playerBorderRect.Y +
5);
            enemyName = new Vector2(enemyBorderRect.X + 55, enemyBorderRect.Y + 5);
        }

        #endregion

        #region Method Region

        public override void Initialize()
        {
            base.Initialize();
        }

        protected override void LoadContent()
        {
            combatBackground = new Texture2D(GraphicsDevice, 1280, 720);
            combatBackground.Fill(Color.White);

            avatarBorder = new Texture2D(GraphicsDevice, 300, 75);
            avatarHealth = new Texture2D(GraphicsDevice, 300, 25);

            avatarBorder.Fill(Color.Green);
            avatarHealth.Fill(Color.Red);

            base.LoadContent();
        }

        public override void Update(GameTime gameTime)
        {
            if ((cTimer > TimeSpan.FromSeconds(3) ||
                enemy.Health.X <= 0 ||
                player.Health.X <= 0) &&
                dTimer > TimeSpan.FromSeconds(2))
            {
                if (enemy.Health.X <= 0 || player.Health.X <= 0)
                {
                    manager.PopState();
                    manager.PushState((BattleOverState)BattleOverState);
                    BattleOverState.SetShadowMonsters(player, enemy);
                }
                else
                {
                    manager.PopState();
                }
            }
            else if (cTimer > TimeSpan.FromSeconds(2) && first && enemy.Health.X <=
0 && player.Health.X <= 0)
            {
                first = false;
                dTimer = TimeSpan.Zero;

                if (turn == CurrentTurn.Players)
                {
                    turn = CurrentTurn.Enemies;
                    enemy.ResolveMove(enemyMove, player);
                }
                else
                {
                    turn = CurrentTurn.Players;
                    player.ResolveMove(playerMove, enemy);
                }
            }
            else if (cTimer == TimeSpan.Zero)
            {
                dTimer = TimeSpan.Zero;

                if (turn == CurrentTurn.Players)
                {
                    player.ResolveMove(playerMove, enemy);
                }
                else
                {
                    enemy.ResolveMove(enemyMove, player);
                }
            }

            cTimer += gameTime.ElapsedGameTime;
            dTimer += gameTime.ElapsedGameTime;

            base.Update(gameTime);
        }

        public override void Draw(GameTime gameTime)
        {
            base.Draw(gameTime);

            GraphicsDevice.SetRenderTarget(renderTarget);
            GraphicsDevice.Clear(Color.Black);

            SpriteBatch.Begin();

            SpriteBatch.Draw(combatBackground, Vector2.Zero, Color.White);

            Vector2 location = new(25, 475);

            if (turn == CurrentTurn.Players)
            {
                SpriteBatch.DrawString(
                    ControlManager.SpriteFont,
                    player.Name + " uses " + playerMove.Name + ".",
                    location,
                    Color.Black);

                if (playerMove.Target == TargetType.Enemy && playerMove.Hurts)
                {
                    location.Y += ControlManager.SpriteFont.LineSpacing;

                    if (ShadowMonster.GetMoveModifier(playerMove, enemy) < 1f)
                    {
                        SpriteBatch.DrawString(
                            ControlManager.SpriteFont,
                            "It is not very effective.",
                            location,
                            Color.Black);
                    }
                    else if (ShadowMonster.GetMoveModifier(playerMove, enemy) > 1f)
                    {
                        SpriteBatch.DrawString(
                            ControlManager.SpriteFont,
                            "It is super effective.",
                            location,
                            Color.Black);
                    }
                }
            }
            else
            {
                SpriteBatch.DrawString(
                    ControlManager.SpriteFont,
                    "Enemy " + enemy.Name + " uses " + enemyMove.Name + ".",
                    location,
                    Color.Black);

                if (enemyMove.Target == TargetType.Enemy && playerMove.Hurts)
                {
                    location.Y += ControlManager.SpriteFont.LineSpacing;

                    if (ShadowMonster.GetMoveModifier(enemyMove, player) < 1f)
                    {
                        SpriteBatch.DrawString(
                            ControlManager.SpriteFont,
                            "It is not very effective.",
                            location,
                            Color.Black);
                    }
                    else if (ShadowMonster.GetMoveModifier(enemyMove, player) > 1f)
                    {
                        SpriteBatch.DrawString(
                            ControlManager.SpriteFont,
                            "It is super effective.",
                            location,
                            Color.Black);
                    }
                }
            }

            SpriteBatch.Draw(avatarBorder, playerBorderRect, Color.White);

            SpriteBatch.Draw(avatarBorder, playerBorderRect, Color.White);

            playerHealth = (float)player.Health.X / (float)player.Health.Y;
            MathHelper.Clamp(playerHealth, 0f, 1f);
            playerHealthRect.Width = (int)(playerHealth * 286);

            SpriteBatch.Draw(avatarHealth, playerHealthRect, healthSourceRect, 
Color.White);

            SpriteBatch.Draw(avatarBorder, enemyBorderRect, Color.White);

            enemyHealth = (float)enemy.Health.X / (float)enemy.Health.Y;
            MathHelper.Clamp(enemyHealth, 0f, 1f);
            enemyHealthRect.Width = (int)(enemyHealth * 286);
```

```
            SpriteBatch.Draw(avatarHealth, enemyHealthRect, healthSourceRect, 
Color.White);
            SpriteBatch.DrawString(ControlManager.SpriteFont, player.Name, 
playerName, Color.White);
            SpriteBatch.DrawString(ControlManager.SpriteFont, enemy.Name, enemyName, 
Color.White);

            SpriteBatch.End();

            GraphicsDevice.SetRenderTarget(null);

            SpriteBatch.Begin();
            SpriteBatch.Draw(renderTarget, new Rectangle(new Point(), 
Settings.Resolution), Color.White);
            spriteBatch.End();
        }

        public void SetShadowMonsters(ShadowMonster player, ShadowMonster enemy)
        {
            this.player = player;
            this.enemy = enemy;

            if (player.Speed + player.SpeedMod >= enemy.Speed + enemy.SpeedMod)
            {
                turn = CurrentTurn.Players;
            }
            else
            {
                turn = CurrentTurn.Enemies;
            }
        }

        public void SetMoves(MoveData playerMove, MoveData enemyMove)
        {
            this.playerMove = playerMove;
            this.enemyMove = enemyMove;
        }

        public void Start()
        {
            cTimer = TimeSpan.Zero;
            dTimer = TimeSpan.Zero;
            first = true;
        }

        #endregion
    }
}
```

This is a really busy class. First, there is an enumeration that tells whose turn it is. Like all of our states, this state has an interface. Unlike other states, it defines some methods that must be implemented. The first methods to be implemented is SetShadowMonsters, which takes as parameters the player's shadow monster and the shadow monster that the player will be battling. SetMoves is similar to SetShadowMonsters. It takes moves rather than shadow monsters. Finally, the Start method will start the timers that control how text is rendered.

There are a lot of fields. CurrrentTurn tells whose turn it is. The combatBackground field is the background to be drawn. The playerRect and enemyRect fields are the destination for the player and enemy sprites, respectively. The cTimer and dTimer fields are for calculating how long it has been since they started. The player and enemy fields are the player's and the enemy's shadow monsters. The playerMove and enemyMove fields are the player and enemy moves. Next, there are pairs of fields for the player and enemy shadow monsters. They are for the border, thumbnail and current health. There is a source rectangle for the health bar. The next fields are the percent of health the player and enemy shadow monsters have left. After that are the textures for the borders for the shadow monsters, then where to draw the player and enemy shadow monster names.

First, the constructor registers itself as a service in the game. It then initializes the position fields. I settled on the values after playing around.

Rather than create graphics in Photoshop, or GIMP, I create the textures for the screen manually. I use the overload of the Texture2D class that takes a GraphicsDevice, the width of the texture and the height of the texture. I use the Fill extension method to fill them. I do this for the three textures.

The Update method is where damage is applied based on time. If cTimer is greater than three seconds, the enemy is not alive, the player is not alive or dTimer is greater than two seconds, then if the enemy is not alive or the player is not alive, then one of the shadow monsters has fainted, and the state is popped off the stack, the BattleOverState is pushed on the stack, and the BattleOverState is passed the player and enemy shadow monsters. I will get to the new state shortly. If cTimer and dTimer are greater than three and two seconds, respectively, the damage has been applied to both shadow monsters, and the state is popped off the stack.

Else, if cTimer is greater than two seconds and both shadow monsters are alive, and the turn is true it is time to apply damage and switch turns. First, set to false, and dTimer is set to zero. If the turn is the player's turn, the turn is switched to the enemy and the enemy's ResolveMove method is called. Otherwise, it's the enemy's turn, so the turn is switched to the player, and the player's ResolveMove method is called.

If cTimer is zero, then this is the first time through. The dTimer is set to zero. If the turn is the player's turn, I call the player's ResolveMove method. Otherwise, I call the enemy's ResolveMove method.

The Draw method does a lot. First, it sets the render target like other states. Then, it calls base.Draw to draw any children. It then draws the background for the state. The location local variable is where text will be drawn. If it is the player's turn, it draws what move the player's shadow monster just used. Next, there is an if statement that tests that the target is the enemy and the move is an attack. It then calls GetMoveModifier to see if the modifier is less than one and appends that the move is not very effective. It then checks if it is greater than zero and appends that it is super effective. I do the same thing based on the enemy and its move. The rest of the method is drawing the various components. The interesting code is where I calculate the health. The health is current health divided by current maximum health. I then clamp it between zero and one. The width of the health bar is then 286 times the percent health.

The SetShadowMonsters method sets the player and enemy fields to the values passed in. It then determines who attacks first based on the speed attribute of the shadow monsters, siding with the player if they are equal. SetMoves just sets the fields to the values passed in. Start resets the timers to zero and sets first to true.

I need to add a couple of method stubs to the ShadowMonster class, GetMoveModifier and ResolveMove. Add these two methods to the ShadowMonster class. The first returns a float that calculates the effectiveness of a move based on elements. It currently returns one for testing purposes. Once I've added more elements, I will fill this out. The other method is ResolveMove, and it will apply the damage to the target shadow monster.

Before getting to the next state, there are a few things that I need to add to the Player class. Most importantly, they need shadow monsters. After all, that is the hole point of this game. Update the Player class to the following.

```
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;
using SummonersTale.ShadowMonsters;
using SummonersTale.SpriteClasses;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace SummonersTale
{
    public class Player : DrawableGameComponent
    {
        private SpriteBatch _spriteBatch;

        public Player(Game game, string name, bool gender, AnimatedSprite sprite)
            :base(game)
        {
            _spriteBatch = game.Services.GetService<SpriteBatch>();

            Name = name;
            Gender = gender;
            Sprite = sprite;
        }

        public string Name { get; private set; }
        public bool Gender { get; private set; }
        public AnimatedSprite Sprite { get; private set; }

        public List<ShadowMonster> ShadowMonsters { get; private set; } = new();

        public List<ShadowMonster> BattleMonsters { get; private set; } = new();
        public override void Update(GameTime gameTime)
        {
            if (Sprite != null)
            {
                Sprite.Update(gameTime);
            }

            base.Update(gameTime);
        }

        public override void Draw(GameTime gameTime)
        {
            base.Draw(gameTime);
            if (Sprite != null)
            {
                _spriteBatch.Begin();
                Sprite.Draw(_spriteBatch);
                _spriteBatch.End();
            }
        }

        internal bool Alive()
        {
            return BattleMonsters.Where(x => x.Health.X > 0).Any();
        }
    }
}
```

I added two properties, ShadowMonsters and BattleMonsters. The first is all shadow monsters that the player owns. The second is a list of the player's current team of six battle monsters. There is also a new method, Alive, that checks to see if any of the player's battle monsters have not fainted, thanks to a little LINQ. It is always more efficient to use LINQ when you need to iterate over a list whenever possible.

Sorry to be bouncing around so much, but add these two method stubs to the ShadowMonster class. One will be called when a shadow monster wins a battle. The other will be called to check to see if a shadow monster has levelled up.

```
        internal long WinBattle(ShadowMonster enemy)
        {
            return 0;
        }

        internal bool CheckLevelUp()
        {
            return true;
        }
```

Now, I am going to add the BattleOverState. It will be placed on the stack once the battle is over. It will redirect to either the battle won or battle lost state. Right-click the StateManagement folder, select Add and then Class. Name this new class BattleOverState. Here is the code.

```
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using SummonersTale.Forms;
using SummonersTale.ShadowMonsters;

namespace SummonersTale.StateManagement
{
    public interface IBattleOverState
    {
        void SetShadowMonsters(ShadowMonster player, ShadowMonster enemy);
    }

    public class BattleOverState : BaseGameState, IBattleOverState
    {
        #region Field Region

        private ShadowMonster player;
        private ShadowMonster enemy;
        private Texture2D combatBackground;
        private Rectangle playerRect;
        private Rectangle enemyRect;
        private Rectangle playerBorderRect;
        private Rectangle enemyBorderRect;
        private Rectangle playerMiniRect;
        private Rectangle enemyMiniRect;
        private Rectangle playerHealthRect;
        private Rectangle enemyHealthRect;
        private Rectangle healthSourceRect;
        private Vector2 playerName;
        private Vector2 enemyName;
        private float playerHealth;
        private float enemyHealth;
        private Texture2D avatarBorder;
        private Texture2D avatarHealth;
        private readonly string[] battleState;
        private Vector2 battlePosition;
        private bool levelUp;

        #endregion

        #region Property Region
        #endregion

        #region Constructor Region

        public BattleOverState(Game game)
            : base(game)
        {
            Game.Services.AddService<IBattleOverState>(this);

            battleState = new string[3];

            battleState[0] = "The battle was won!";
            battleState[1] = " gained ";
            battleState[2] = "Continue";

            battlePosition = new Vector2(25, 475);
            playerRect = new Rectangle(10, 90, 300, 300);
            enemyRect = new Rectangle(game.Window.ClientBounds.Width - 310, 10, 300,
300);

            playerBorderRect = new Rectangle(10, 10, 300, 75);
            enemyBorderRect = new Rectangle(game.Window.ClientBounds.Width - 310,
320, 300, 75);

            healthSourceRect = new Rectangle(10, 50, 290, 20);
            playerHealthRect = new Rectangle(playerBorderRect.X + 12,
playerBorderRect.Y + 52, 286, 16);
            enemyHealthRect = new Rectangle(enemyBorderRect.X + 12,
enemyBorderRect.Y + 52, 286, 16);

            playerMiniRect = new Rectangle(playerBorderRect.X + 11,
playerBorderRect.Y + 11, 28, 28);
            enemyMiniRect = new Rectangle(enemyBorderRect.X + 11, enemyBorderRect.Y
+ 11, 28, 28);

            playerName = new Vector2(playerBorderRect.X + 55, playerBorderRect.Y +
5);
            enemyName = new Vector2(enemyBorderRect.X + 55, enemyBorderRect.Y + 5);
        }

        #endregion

        #region Method Region

        protected override void LoadContent()
        {
            combatBackground = new Texture2D(GraphicsDevice, 1280, 720);
            Color[] buffer = new Color[1280 * 720];

            for (int i = 0; i < buffer.Length; i++)
            {
                buffer[i] = Color.White;
            }

            combatBackground.SetData(buffer);

            avatarBorder = new Texture2D(GraphicsDevice, 300, 75);
            avatarHealth = new Texture2D(GraphicsDevice, 300, 25);

            buffer = new Color[300 * 75];

            for (int i = 0; i < buffer.Length; i++)
            {
                buffer[i] = Color.Green;
            }

            avatarBorder.SetData(buffer);

            buffer = new Color[300 * 25];

            for (int i = 0; i < buffer.Length; i++)
            {
                buffer[i] = Color.Red;
            }

            avatarHealth.SetData(buffer);

            base.LoadContent();
        }

        public override void Update(GameTime gameTime)
        {
            if (Xin.WasKeyReleased(Keys.Space) || Xin.WasKeyReleased(Keys.Enter))
            {
                if (levelUp)
                {
                    LevelUpState levelUpState = 
Game.Services.GetService<LevelUpState>();
                    manager.PushState(levelUpState);
                    levelUpState.SetShadowMonster(player);

                    this.Visible = true;
                }
                else if (Player.Alive())
                {
                    manager.PopState();
                    manager.PopState();
                }
                else
                {
                    manager.PopState();
                    manager.PopState();
                }
            }

            base.Update(gameTime);
        }

        public override void Draw(GameTime gameTime)
        {
            Vector2 position = battlePosition;

            base.Draw(gameTime);

            SpriteBatch.Begin();

            SpriteBatch.Draw(combatBackground, Vector2.Zero, Color.White);

            for (int i = 0; i < 2; i++)
            {
                SpriteBatch.DrawString(
                    ControlManager.SpriteFont,
                    battleState[i],
                    position,
                    Color.Black);
                position.Y += ControlManager.SpriteFont.LineSpacing;
            }

            SpriteBatch.DrawString(
                ControlManager.SpriteFont,
                battleState[2],
                position,
                Color.Red);

            SpriteBatch.Draw(player.Sprite, playerRect, Color.White);
            SpriteBatch.Draw(enemy.Sprite, enemyRect, Color.White);

            SpriteBatch.Draw(avatarBorder, playerBorderRect, Color.White);

            playerHealth = (float)player.Health.X / (float)player.Health.Y;
            MathHelper.Clamp(playerHealth, 0f, 1f);
            playerHealthRect.Width = (int)(playerHealth * 286);

            SpriteBatch.Draw(avatarHealth, playerHealthRect, healthSourceRect, 
Color.White);

            SpriteBatch.Draw(avatarBorder, enemyBorderRect, Color.White);

            enemyHealth = (float)enemy.Health.X / (float)enemy.Health.Y;
            MathHelper.Clamp(enemyHealth, 0f, 1f);
            enemyHealthRect.Width = (int)(enemyHealth * 286);

            SpriteBatch.Draw(avatarHealth, enemyHealthRect, healthSourceRect, 
Color.White);
            SpriteBatch.DrawString(
                ControlManager.SpriteFont,
                player.Name,
                playerName, Color.
                White);
            SpriteBatch.DrawString(
                ControlManager.SpriteFont,
                enemy.Name,
                enemyName,
                Color.White);

            SpriteBatch.Draw(player.Sprite, playerMiniRect, Color.White);
            SpriteBatch.Draw(enemy.Sprite, enemyMiniRect, Color.White);

            SpriteBatch.End();
        }

        public void SetShadowMonsters(ShadowMonster player, ShadowMonster enemy)
        {
            levelUp = false;
            this.player = player;
            this.enemy = enemy;


            long expGained;
            if (player.Health.X >= 0)
            {
                expGained = player.WinBattle(enemy);

                battleState[0] = player.Name + " has won the battle!";
                battleState[1] = player.Name + " has gained " + expGained + " 
experience";

                if (player.CheckLevelUp())
                {
                    battleState[1] += " and gained a level!";

                    foreach (Move move in player.LockedMoves)
                    {
                        if (player.Level >= move.Level)
                        {
                            player.UnlockedMoves.Add(move);
                            battleState[1] += " " + move.Name + " was unlocked!";
                        }
                    }

                    levelUp = true;
                }
                else
                {
                    battleState[1] += ".";
                }
            }
            else
            {
                battleState[0] = player.Name + " has lost the battle.";
            }
        }

        #endregion
    }
}
```

There is an interface like the other states. It has one method, SetShadowMonsters. It will be used to pass the shadow monsters to the state.

The fields of the class are basically identical to DamageState. The difference being there is a battleState field that is an array of strings for outputting the outcome of the battle. There is a field, levelUp, that checks if the player's shadow monster has levelled up. and there are no Move fields.

The constructor works the same as DamageState with the addition of assigning some values to the battleState field. LoadContent creates the content the same way as in DamageState.

Update checks to see if space or enter has been released. If they have, it checks to see if the player's shadow monster levelled up. If it has, the level up state is pushed on the stack, but this state remains visible, or it would if we weren't using render targets. That is something I will solve in a future tutorial. It also sets the shadow monster in the level up state. If the player is alive, the states are popped off the stack. If the player is not alive, the states are popped off the stack. Here we should warp to a location but currently don't have access to the world.

The Draw method is similar to the Draw method of the DamageState. The difference being that it writes out the outcome of the battle rather than what moves were used. It loops over the first two parts and writes the third part directly.

The SetShadowMonsters method sets levelUp to false then sets the shadow monster fields.

There is a local variable that holds the experience gained from the battle. It checks if the player is alive. If it is it calls the WinBattle method of the ShadowMonster class passing in the enemy shadow monster. It then sets the first two parts of the display string. If the player has levelled up I append that message. I then loop over the known moves. If the move is locked and the player's level is greater than or equal to the unlocked level, I unlock the move and append the move was unlocked. The levelUp field is set to true. If the shadow monster did not level up, I just append a period. The case for the player not alive works the same as the first, just different strings and calling LoseBattle instead of WinBattle.

The next state that I'm going to implement is the LevelUpState. As mentioned, it is called when a shadow monster levels up. It is a pop-up and doesn't cover the full screen like the other states that have been implemented so far. That is why I set the battle over state to visible when I push the state on the stack.

```
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using SummonersTale.Forms;
using SummonersTale.ShadowMonsters;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace SummonersTale.StateManagement
{
    public interface ILevelUpState
    {
        void SetShadowMonster(ShadowMonster playerShadowMonster);
    }

    public class LevelUpState : BaseGameState, ILevelUpState
    {
        #region Field Region

        private Rectangle destination;
        private int points;
        private int selected;
        private ShadowMonster player;
        private readonly Dictionary<string, int> attributes = new Dictionary<string,
int>();
        private readonly Dictionary<string, int> assignedTo = new Dictionary<string,
int>();
        private Texture2D levelUpBackground;

        #endregion

        #region Property Region
        #endregion

        #region Constructor Region

        public LevelUpState(Game game)
```

```
            : base(game)
        {
            Game.Services.AddService<ILevelUpState>(this);

            attributes.Add("Attack", 0);
            attributes.Add("Defense", 0);
            attributes.Add("Speed", 0);
            attributes.Add("Health", 0);
            attributes.Add("Done", 0);

            foreach (string s in attributes.Keys)
                assignedTo.Add(s, 0);
        }

        #endregion

        #region Method Region

        public override void Initialize()
        {
            base.Initialize();
        }

        protected override void LoadContent()
        {
            levelUpBackground = new Texture2D(GraphicsDevice, 500, 400);
            levelUpBackground.Fill(Color.Gray);

            destination = new Rectangle(
                (TargetWidth - levelUpBackground.Width) / 2,
                (TargetHeight - levelUpBackground.Height) / 2,
                levelUpBackground.Width,
                levelUpBackground.Height);

            base.LoadContent();
        }

        public override void Update(GameTime gameTime)
        {
            int i = 0;
            string attribute = "";

            if (Xin.WasKeyReleased(Keys.Down) || Xin.WasKeyReleased(Keys.S))
            {
                selected++;

                if (selected >= attributes.Count)
                {
                    selected = attributes.Count - 1;
                }
            }
            else if (Xin.WasKeyReleased(Keys.Up) || Xin.WasKeyReleased(Keys.W))
            {
                selected--;

                if (selected < 0)
                {
                    selected = 0;
                }
            }

            if (Xin.WasKeyReleased(Keys.Space) || Xin.WasKeyReleased(Keys.Enter))
            {
                if (selected == 4 && points == 0)
                {
                    foreach (string s in assignedTo.Keys)
                    {
                        player.AssignPoint(s, assignedTo[s]);
                    }

                    manager.PopState();
                    manager.PopState();
                    manager.PopState();
                    return;
                }
            }

            int increment = 1;

            if ((Xin.WasKeyReleased(Keys.Right) || Xin.WasKeyReleased(Keys.D)) &&
points > 0)
            {
                foreach (string s in assignedTo.Keys)
                {
                    if (s == "Done")
                    {
                        return;
                    }

                    if (i == selected)
                    {
                        attribute = s;
                        break;
                    }

                    i++;
                }


                if (attribute == "Health")
                {
                    increment *= 5;
                }

                points--;
                assignedTo[attribute] += increment;

                if (points == 0)
                {
                    selected = 4;
                }
            }
            else if ((Xin.WasKeyReleased(Keys.Left) || Xin.WasKeyReleased(Keys.A))
&& points <= 3)
            {
                foreach (string s in assignedTo.Keys)
                {
                    if (s == "Done")
                    {
                        return;
                    }

                    if (i == selected)
                    {
                        attribute = s;
                        break;
                    }

                    i++;
                }

                if (assignedTo[attribute] != attributes[attribute])
                {
                    if (attribute == "Health")
                    {
                        increment *= 5;
                    }

                    points++;
                    assignedTo[attribute] -= increment;
                }
            }

            base.Update(gameTime);
        }

        public override void Draw(GameTime gameTime)
        {
            base.Draw(gameTime);


            SpriteBatch.Begin();
            SpriteBatch.Draw(levelUpBackground, destination, Color.White);

            Vector2 textPosition = new Vector2(destination.X + 5, destination.Y +
5);

            SpriteBatch.DrawString(ControlManager.SpriteFont, player.Name,
textPosition, Color.White);
            textPosition.Y += ControlManager.SpriteFont.LineSpacing * 2;

            int i = 0;

            foreach (string s in attributes.Keys)
            {
                Color tint = Color.White;

                if (i == selected)
                    tint = Color.Red;

                if (s != "Done")
                {
                    SpriteBatch.DrawString(ControlManager.SpriteFont, s + ":",
textPosition, tint);
```

```
                    textPosition.X += 125;

                    SpriteBatch.DrawString(ControlManager.SpriteFont,
attributes[s].ToString(), textPosition, tint);
                    textPosition.X += 40;

                    SpriteBatch.DrawString(ControlManager.SpriteFont,
assignedTo[s].ToString(), textPosition, tint);
                    textPosition.X = destination.X + 5;

                    textPosition.Y += ControlManager.SpriteFont.LineSpacing;
                }
                else
                {
                    SpriteBatch.DrawString(ControlManager.SpriteFont, "Done",
textPosition, tint);
                    textPosition.Y += ControlManager.SpriteFont.LineSpacing * 2;
                }
                i++;
            }

            SpriteBatch.DrawString(
                ControlManager.SpriteFont,
                points.ToString() + " point left.",
                textPosition,
                Color.White);
            SpriteBatch.End();
        }

        public void SetShadowMonster(ShadowMonster playerShadowMonster)
        {
            player = playerShadowMonster;

            attributes["Attack"] = player.Attack;
            attributes["Defense"] = player.Defence;
            attributes["Speed"] = player.Speed;
            attributes["Health"] = player.Health.X;

            assignedTo["Attack"] = random.Next(0, 2);
            assignedTo["Defense"] = random.Next(0, 2);
            assignedTo["Speed"] = random.Next(0, 2);
            assignedTo["Health"] = random.Next(5, 11);

            points = 3;
            selected = 0;
        }

        #endregion
    }
}
```

Again, there is an interface that the class will implement. This time the single method is SetShadowMonster which requires the shadow monster to level up. The class inherits from BaseGameState and implements the interface.

There are several fields in the class. The destination field is where we will be drawing the level up state. The points field is the number of points that can be assigned to attributes. Applying a point to health increases health by five. The selected field is what attribute is currently selected to have a point assigned to it. The player field is the shadow monster that is levelling up. The attributes field is the attribute, and assignedTo is the values being assigned. Finally, levelUpBackground is the background that will be drawn.

First, the constructor registers itself as a service. Then, the constructor initializes the two collections. The first collection is done manually, adding the attributes one at a time. The other is initialized using the first. The LoadContent method creates the background texture as we did with the other textures earlier in the tutorial.

There is a lot going on in the Update method. First, there are local variables i and attribute. The i is a counter and attribute is the attribute being assigned. First, there is a check if the down or S keys have been pressed. If they have, the selected field is incremented. If it is greater than or equal to the number of elements, it is set to the number of elements minus one. If those keys have not been released, there is a check for the up or W keys. In this case, the selected field is decremented, and if it is less than zero, it is set to zero.

Next, there is a check if the space or enter keys have been released. If the selected field is four, the Done entry, and there are no points left, I loop over the keys in assignedTo and call the AssignPoint method on the shadow monster. Then I pop the three states off the stack: level up, battle over and damage.

Next, there is a local variable increment that tells how many points are assigned to an attribute. Now there is a check to see if the right or D keys have been released, meaning that the player wants to assign a point to the attribute and that there are points to spend. There is then a foreach loop that loops over the assignedTo collection. If the key is Done, we return out of the method. If i is selected, we grab the key in the attribute field and break out of the loop. Then we increment the i counter. If the attribute is Health, the increment is multiplied by five. The points field is decremented, and assignedTo for the attribute is incremented by the value of the increment variable. If there are no points left, selected is set to Done.

The case for the left and A keys works basically the same way. The difference is it checks to see that points are less than or equal to 3, and it decrements instead of increments.

The Draw method draws the background first. Then there is a local variable textPosition that determines where to draw the text. It offsets the X and Y position of the background by 5 pixels. It draws the name of the shadow monster and then increments the Y coordinate of the text position by the line spacing plus two. There is a local variable i that counts the attribute that is being drawn. I then loop over the attribute keys. I set a local variable tint to White, and if i is selected, set it to red. If the key is not Done, I draw the attribute name, the shadow monster's current attribute and the points being assigned. If it is Done, I just draw done. Both cases increment the Y value of textPosition to move to the next line. Then I increment the counter. After drawing the attributes, I draw the number of points left.

The SetShadowMonster method sets the attributes to the attributes of the shadow monster, It then adds some random values to the assignedTo values. This will make a trained shadow monster much tougher than a wild shadow monster of the same level. It then initializes points to three and selected to zero.

Now, back to the ShadowMonster class. We need to add the AssignPoint method. Add this method to the ShadowMonster class.

```
public void AssignPoint(string s, int p)
{
    Type type = typeof(ShadowMonster);
    PropertyInfo info = type.GetProperty(s);

    if (info.PropertyType != typeof(Point))
    {
        info?.SetValue(this, p + (int)info.GetValue(this, null), null);
    }
    else
    {
        info?.SetValue(this, new Point(0, p) + (Point)info.GetValue(this, null),
null);
    }
}
```

Time for a little more reflection rather than building out an if-else series or switch statement. In this case, it is probably the same amount of code in this case. However, if we wanted to add more attributes, this requires no additional changes. They will be handled automatically, just like our additions to the data classes. We do not have to go into the ToString and FromString methods. The changes just work. I will mention that this does not work for saved files. For those, you are best to create a temporary class, or classes, that will be your new save format. Use the existing code to open the file, and transfer the data to the new classes. Write out the data using the new classes. Update the existing code to the new classes. Finally, modify the loading code to match the new format.

The last state that I'm going to implement is the battle state. Right-click the StateManagement folder in the Solution Explorer, select Add and then Class. Name this new class BattleState. Here is the code for that class.

```
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Psilibrary.ConversationComponents;
using SummonersTale.Forms;
using SummonersTale.ShadowMonsters;
using System.Collections.Generic;
using System.Linq;

namespace SummonersTale.StateManagement
{
    public interface IBattleState
    {
        void SetShadowMonsters(ShadowMonster player, ShadowMonster enemy);
        void StartBattle();
        void ChangePlayerShadowMonster(ShadowMonster selected);
    }

    public class BattleState : BaseGameState, IBattleState
    {
        #region Field Region

        private ShadowMonster player;
        private ShadowMonster enemy;
        private GameScene combatScene;
        private Texture2D combatBackground;
        private Rectangle playerRect;
        private Rectangle enemyRect;
        private Rectangle playerBorderRect;
        private Rectangle enemyBorderRect;
        private Rectangle playerMiniRect;
        private Rectangle enemyMiniRect;
        private Rectangle playerHealthRect;
        private Rectangle enemyHealthRect;
        private Rectangle healthSourceRect;
        private Vector2 playerName;
        private Vector2 enemyName;
        private float playerHealth;
        private float enemyHealth;
        private Texture2D avatarBorder;
        private Texture2D avatarHealth;

        public ShadowMonster EnemyShadowMonster { get { return enemy; } }

        #endregion

        #region Property Region
        #endregion

        #region Constructor Region

        public BattleState(Game game)
            : base(game)
        {
            Game.Services.AddService<IBattleState>(this);

            playerRect = new(10, 90, 300, 300);
            enemyRect = new(TargetWidth - 310, 10, 300, 300);

            playerBorderRect = new(10, 10, 300, 75);
            enemyBorderRect = new(TargetWidth - 310, 320, 300, 75);

            healthSourceRect = new(10, 50, 290, 20);
            playerHealthRect = new(
                playerBorderRect.X + 12,
                playerBorderRect.Y + 52,
                286,
                16);
            enemyHealthRect = new(enemyBorderRect.X + 12, enemyBorderRect.Y + 52,
286, 16);
```

```
            playerMiniRect = new(playerBorderRect.X + 11, playerBorderRect.Y + 11,
28, 28);
            enemyMiniRect = new(enemyBorderRect.X + 11, enemyBorderRect.Y + 11, 28,
28);

            playerName = new(playerBorderRect.X + 55, playerBorderRect.Y + 5);
            enemyName = new(enemyBorderRect.X + 55, enemyBorderRect.Y + 5);
        }

        #endregion

        #region Method Region

        protected override void LoadContent()
        {
            if (combatScene == null)
            {
                combatBackground = new(GraphicsDevice, 1280, 720);
                combatBackground.Fill(Color.White);

                avatarBorder = new(GraphicsDevice, 300, 75);
                avatarHealth = new(GraphicsDevice, 300, 25);

                avatarBorder.Fill(Color.Green);
                avatarHealth.Fill(Color.Red);

                combatScene = new(Game, "", new List<SceneOption>());
            }

            base.LoadContent();
        }

        public override void Update(GameTime gameTime)
        {
            if (Xin.WasKeyReleased(Keys.P))
            {
                manager.PopState();
            }

            combatScene.Update(gameTime);

            if (Xin.WasKeyReleased(Keys.Space) ||
                Xin.WasKeyReleased(Keys.Enter) ||
                (Xin.WasMouseReleased(MouseButton.Left) &&
                combatScene.IsOver))
            {
                DamageState damageState =
(DamageState)Game.Services.GetService<IDamageState>();

                manager.PushState(damageState);
                damageState.SetShadowMonsters(player, enemy);

                Move enemyMove = null;

                int move = random.Next(0, enemy.Moves.Count);
                enemyMove = (Move)enemy.Moves[move];
                damageState.SetMoves(
                    player.Moves.Where(x => x.Name == 
combatScene.OptionText).FirstOrDefault(),
                    enemyMove);
                damageState.Start();
            }

            Visible = true;

            base.Update(gameTime);
        }

        public override void Draw(GameTime gameTime)
        {
            base.Draw(gameTime);

            SpriteBatch.Begin();

            SpriteBatch.Draw(combatBackground, Vector2.Zero, Color.White);
            combatScene.Draw(gameTime, SpriteBatch, combatBackground);

            SpriteBatch.Draw(player.Sprite, playerRect, Color.White);
            SpriteBatch.Draw(enemy.Sprite, enemyRect, Color.White);

            SpriteBatch.Draw(avatarBorder, playerBorderRect, Color.White);

            playerHealth = (float)player.Health.X / (float)player.Health.Y;
            MathHelper.Clamp(playerHealth, 0f, 1f);
            playerHealthRect.Width = (int)(playerHealth * 286);

            SpriteBatch.Draw(avatarHealth, playerHealthRect, healthSourceRect, 
Color.White);

            SpriteBatch.Draw(avatarBorder, enemyBorderRect, Color.White);

            enemyHealth = (float)enemy.Health.X / (float)enemy.Health.Y;
            MathHelper.Clamp(enemyHealth, 0f, 1f);
            enemyHealthRect.Width = (int)(enemyHealth * 286);

            SpriteBatch.Draw(avatarHealth, enemyHealthRect, healthSourceRect, 
Color.White);
            SpriteBatch.DrawString(
                ControlManager.SpriteFont,
                player.Name,
                playerName,
                Color.White);
            SpriteBatch.DrawString(
                ControlManager.SpriteFont,
                enemy.Name,
                enemyName,
                Color.White);

            SpriteBatch.Draw(player.Sprite, playerMiniRect, Color.White);
            SpriteBatch.Draw(enemy.Sprite, enemyMiniRect, Color.White);

            SpriteBatch.End();
        }
        public void SetShadowMonsters(ShadowMonster player, ShadowMonster enemy)
        {
            this.player = player;
            this.enemy = enemy;

            player.StartCombat();
            enemy.StartCombat();

            List<SceneOption> moves = new();

            if (combatScene == null)
            {
                LoadContent();
            }

            foreach (Move move in player.Moves)
            {
                SceneOption option = new(move.Name, move.Name, new SceneAction());
                moves.Add(option);
            }

            combatScene.Options = moves;
        }

        public void StartBattle()
        {
            player.StartCombat();
            enemy.StartCombat();
            playerHealth = 100f;
            enemyHealth = 100f;
        }

        public void ChangePlayerShadowMonster(ShadowMonster selected)
        {
            this.player = selected;

            List<SceneOption> moves = new();

            foreach (Move move in player.Moves)
            {
                SceneOption option = new(move.Name, move.Name, new SceneAction());
                moves.Add(option);
            }

            combatScene.Options = moves;
        }

        #endregion
    }
}
```

Surprise, surprise, there is another interface. It has three methods: SetShadowMonsters, StartBattle and ChangePlayerShadowMonster. The methods set the combatants, start a battle and change the player's shadow monster. The class inherits from BaseGameState and implements the interface.

The class has the same fields as the DamageState with the addition of combatScene, which is a GameScene that displays the moves known by the shadow monster. The constructor and LoadContent methods work the same way as DamageState. There is a property that returns the enemy shadow monster.

There is a check to see if the P key was released that pops the state off the stack. This is for debugging purposes to get out of a battle quickly. It should be removed in production. Now the Update method of the combatScene is called to update the scene. There is then a check to see if the space or enter keys have been released or the left mouse button and the mouse is over a scene option. If it has, I push the damage state on top of the stack and call its SetShadowMonsters method to set that shadow monsters. The next thing to do is determine the enemy's move. I grab a random move from the list. You may want to apply a little intelligence here, making the opponents a little harder to predict and more of a challenge. I then call the SetMoves method and start the damage state. I then update the player and enemy. I make sure that the state is always visible.

The Draw method works the same as in the other states. The only difference is that I draw the scene.

SetShadowMonsters sets the player and enemy fields the calls StartCombat, the last new method for shadow monsters, in this tutorial at least. Next, there is a local variable that holds the moves known by the player's shadow monster. If combatScene is null, I call the LoadContent method. I now loop over the known moves and it as a scene option. Since this is the start of combat, the selected index of the scene is set to zero, and the Options are set to moves. StartBattle calls the StartCombat methods of the objects and initializes the playerHealth and enemyHealth fields. You may want to go the Pokémon route and not heal between battles. It is a simple matter of no setting the health to full.

ChangePlayerShadowMonster works the same way as SetShadowMonsters. It just initializes the field and creates the scene.

That is it for the new states. Before I get to implementing these new classes in the game, I need to add a method stub for StartCombat to the ShadowMonster class. Add the following method stub to the ShadowMonster class.

```
        internal void StartCombat()
        {
        }
```

Now it is time to implement them in the game. The first step is to create fields and properties in the Desktop class and initialize them Update the Desktop class to the following.

```
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Psilibrary.ShadowMonsters;
using Psilibrary.TileEngine;
using SummonersTale;
using SummonersTale.ShadowMonsters;
using SummonersTale.StateManagement;
using System;
using System.Collections.Generic;

namespace SummonersTaleGame
{
    public class Desktop : Game
    {
        private readonly GameStateManager _manager;
        private readonly GraphicsDeviceManager _graphics;
        private SpriteBatch _spriteBatch;

        private GamePlayState _playState;
        private TitleState _titleState;
        private MainMenuState _mainMenuState;
        private NewGameState _newGameState;
        private ConversationState _conversationState;
        private DamageState _damageState;
        private BattleState _battleState;
        private BattleOverState _battleOverState;
        private LevelUpState _levelUpState;

        private ConversationManager _conversationManager;

        public SpriteBatch SpriteBatch => _spriteBatch;

        public TitleState TitleState => _titleState;
        public GamePlayState PlayState => _playState;
        public MainMenuState MainMenuState=> _mainMenuState;
        public NewGameState NewGameState => _newGameState;
        public ConversationState ConversationState => _conversationState;
        public DamageState DamageState => _damageState;
        public BattleState BattleState => _battleState;
        public BattleOverState BattleOverState => _battleOverState;
        public LevelUpState LevelUpState => _levelUpState;

        public Desktop()
        {
            Settings.Load();

            _graphics = new GraphicsDeviceManager(this)
            {
                PreferredBackBufferWidth = Settings.Resolution.X,
                PreferredBackBufferHeight = Settings.Resolution.Y,
            };

            _graphics.ApplyChanges();
            _manager = new GameStateManager(this);

            Services.AddService(typeof(GraphicsDeviceManager), _graphics);

            Content.RootDirectory = "Content";
            IsMouseVisible = true;

            Components.Add(_manager);
        }
        protected override void Initialize()
        {
            Components.Add(new FramesPerSecond(this));
            Components.Add(new Xin(this));

            _graphics.ApplyChanges();

            base.Initialize();
        }

        protected override void LoadContent()
        {
            _spriteBatch = new SpriteBatch(GraphicsDevice);
            Services.AddService(typeof(SpriteBatch), _spriteBatch);

            _conversationManager = new(this);
            _conversationManager.LoadConverstions(this);
            _conversationManager.WriteConversations();

            Components.Add(_conversationManager);

            _conversationState = new(this);
            _playState = new(this);
            _titleState = new(this);
            _mainMenuState = new(this);
            _newGameState = new(this);
            _damageState = new(this);
            _battleOverState = new(this);
            _battleState = new(this);
            _levelUpState = new(this);

            _manager.PushState(_titleState);

            Move smash = new()
            {
                Name = "Smash",
                Range = new(1, 6),
                Mana = new(40, 40),
                Target = TargetType.Enemy,
                TargetAttribute = TargetAttribute.Health,
                IsTemporary = false,
                Elements = 0
            };

            string m = smash.ToString();

            smash = Move.FromString(m);

            List<MoveData> moves = new()
            {
                smash
            };

            Move bash = new()
            {
                Name = "Bash",
                Range = new(2, 8),
                Mana = new(30, 30),
                Target = TargetType.Enemy,
                TargetAttribute = TargetAttribute.Health,
                IsTemporary = false,
                Elements = 0
            };

            ShadowMonster monster = new()
            {
                Name = "Goblin",
                Moves = moves,
                Elements = 0,
                Health = new(25, 25),
            };

            monster.LockedMoves.Add(bash);
            monster.UnlockedMoves.Add(smash);

            m = monster.ToString();

            monster = ShadowMonster.FromString(m);
        }

        protected override void Update(GameTime gameTime)
        {
            if (GamePad.GetState(PlayerIndex.One).Buttons.Back ==
ButtonState.Pressed || Keyboard.GetState().IsKeyDown(Keys.Escape))
                Exit();

            // TODO: Add your update logic here

            base.Update(gameTime);
        }

        protected override void Draw(GameTime gameTime)
        {
            GraphicsDevice.Clear(Color.CornflowerBlue);

            base.Draw(gameTime);
        }
    }
}
```

Pretty much the same as implementing other game states. Create the private fields for them, create the properties to expose them, and initialize them in the LoadContent method.

There are a few things. One, this tutorial is already thirty pages long. Two, I need to develop a new type of character to be able to battle. We have no shadow monsters to work with, other than the test goblin. Finally, I have not touched iOS or Android in some time. Those are a lot of outstanding items to try and fit into one tutorial. For those reasons, I'm going to end this tutorial here and pick it up here next time.

So, you can build and run now, but nothing has changed visibly. I will tackle that in the next tutorial, as I'm not going to dive any further in this tutorial. I think I've fed you more than enough for one day. I encourage you to keep visiting my blog for the latest news on my tutorials.

Good luck with your game programming adventures.
Cynthia