Shadow Monsters – MonoGame Tutorial Series
Chapter 8
Battling Shadow Monsters

This tutorial series is about creating a Pokemon style game with the MonoGame Framework called Shadow Monsters. The tutorials will make more sense if you read them in order as each tutorial builds on the previous tutorials. You can find the list of tutorials on my blog: Shadow Monsters. The source code for each tutorial will be available as well. I will be using Visual Studio 2019 Community for the series. The code should compile on the 2013, 2015 and 2017 versions as well.

I want to mention though that the series is released as Creative Commons 3.0 Attribution. It means that you are free to use any of the code or graphics in your own game, even for commercial use, with attribution. Just give credit to Cynthia McMahon and add a link to my site, https://mygameprogrammingadventures.blogspot.com. Screenshots of your project and/or a video of game play would be appreciated.

I also want to mention that I assume you have a basic understanding of C# and MonoGame. If you don't I recommend that you learn basic C# and work with MonoGame a little. Enough to know the basics of fields, properties, methods, classes and the MonoGame framework.

In this tutorial we will be battling shadow monsters. The first thing to do is add in some states to handle the battle There is the battle state, the damage state, the battle over state and the level up state. There are references between them so I'm going to just start with one then implement them all.

First I need to fix a glitch in the FromString method of the Character class. When assigning shadow monsters I need to use the field instead of the property. Update the FromString method of the Character class to the following.

```
public static Character FromString(Game game, string characterString)
{
    if (gameRef == null)
    {
        gameRef = game;
    }

    Character character = new Character();
    string[] parts = characterString.Split(',');

    character.name = parts[0];
    character.textureName = parts[1];
    character.sprite = new AnimatedSprite(
        game.Content.Load<Texture2D>(@"CharacterSprites\" + parts[1]),
        Game1.Animations)
    {
        CurrentAnimation = (AnimationKey)Enum.Parse(typeof(AnimationKey), parts[2])
    };
    character.conversation = parts[3];
    character.currentMonster = int.Parse(parts[4]);
```

```
    for (int i = 5; i < 11 && i < parts.Length - 1; i++)
    {
        ShadowMonster monster =
ShadowMonsterManager.GetShadowMonster(parts[i].ToLowerInvariant());
        character.monsters[i - 5] = monster;
    }

    character.givingMonster = ShadowMonsterManager.GetShadowMonster(parts[parts.Length -
1].ToLowerInvariant());
    return character;
}
```

First state to add is the damage state. The damage state applies the chosen moves to the shadow monsters. Right click the GameStates folder in the solution explorer, select Add and then class. Name this new class DamageState. Here is the code.

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

namespace ShadowMonsters.GameStates
{
    public enum CurrentTurn
    {
        Players, Enemies
    }

    public interface IDamageState
    {
        void SetShadowMonsters(ShadowMonster player, ShadowMonster enemy);
        void SetMoves(IMove playerMove, IMove 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 IMove playerMove;
        private IMove 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)
        {
            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

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

        protected override void LoadContent()
        {
            combatBackground = new Texture2D(GameRef.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 ((cTimer > TimeSpan.FromSeconds(3) ||
                !enemy.Alive ||
                !player.Alive) &&
                dTimer > TimeSpan.FromSeconds(2))
            {
                if (!enemy.Alive || !player.Alive)
                {
                    manager.PopState();
                    manager.PushState((BattleOverState)GameRef.BattleOverState);
                    GameRef.BattleOverState.SetShadowMonsters(player, enemy);
                }
                else
                {
                    manager.PopState();
                }
            }
            else if (cTimer > TimeSpan.FromSeconds(2) && first && enemy.Alive && player.Alive)
            {
                first = false;
                dTimer = TimeSpan.Zero;

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

```
            else if (cTimer == TimeSpan.Zero)
            {
                dTimer = TimeSpan.Zero;

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

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

            base.Update(gameTime);
        }

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

            GameRef.SpriteBatch.Begin();

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

            Vector2 location = new Vector2(25, 475);

            if (turn == CurrentTurn.Players)
            {
                GameRef.SpriteBatch.DrawString(
                    FontManager.GetFont("testfont"),
                    player.DisplayName + " uses " + playerMove.Name + ".",
                    location,
                    Color.Black);

                if (playerMove.Target == Target.Enemy && playerMove.MoveType ==
MoveType.Attack)
                {
                    location.Y += FontManager.GetFont("testfont").LineSpacing;

                    if (ShadowMonster.GetMoveModifier(playerMove.MoveElement, enemy.Element) <
1f)
                    {
                        GameRef.SpriteBatch.DrawString(
                            FontManager.GetFont("testfont"),
                            "It is not very effective.",
                            location,
                            Color.Black);
                    }
                    else if (ShadowMonster.GetMoveModifier(playerMove.MoveElement,
enemy.Element) > 1f)
                    {
                        GameRef.SpriteBatch.DrawString(
                            FontManager.GetFont("testfont"),
                            "It is super effective.",
                            location,
                            Color.Black);
```

```
                    }
                }
            }
            else
            {
                GameRef.SpriteBatch.DrawString(
                    FontManager.GetFont("testfont"),
                    "Enemy " + enemy.DisplayName + " uses " + enemyMove.Name + ".",
                    location,
                    Color.Black);

                if (enemyMove.Target == Target.Enemy && playerMove.MoveType == MoveType.Attack)
                {
                    location.Y += FontManager.GetFont("testfont").LineSpacing;

                    if (ShadowMonster.GetMoveModifier(enemyMove.MoveElement, player.Element) <
1f)
                    {
                        GameRef.SpriteBatch.DrawString(
                            FontManager.GetFont("testfont"),
                            "It is not very effective.",
                            location,
                            Color.Black);
                    }
                    else if (ShadowMonster.GetMoveModifier(enemyMove.MoveElement,
player.Element) > 1f)
                    {
                        GameRef.SpriteBatch.DrawString(
                            FontManager.GetFont("testfont"),
                            "It is super effective.",
                            location,
                            Color.Black);
                    }
                }
            }

            GameRef.SpriteBatch.Draw(avatarBorder, playerBorderRect, Color.White);
            GameRef.SpriteBatch.Draw(player.Texture, playerRect, Color.White);
            GameRef.SpriteBatch.Draw(enemy.Texture, enemyRect, Color.White);

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

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

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

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

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

            GameRef.SpriteBatch.Draw(avatarHealth, enemyHealthRect, healthSourceRect,
Color.White);
            GameRef.SpriteBatch.DrawString(FontManager.GetFont("testfont"), player.DisplayName,
playerName, Color.White);
            GameRef.SpriteBatch.DrawString(FontManager.GetFont("testfont"), enemy.DisplayName,
```

```
enemyName, Color.White);

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

            GameRef.SpriteBatch.End();
        }

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

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

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

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

        #endregion
    }
}
```

A lot going on here. There is an enumeration that will be used to determine who's turn it is. There is a public interface that the class will be implementing. The methods to be implemented are SetShadowMonsters that takes as parameters the player's shadow monster and the enemy's shadow monster. SetMoves is similar to SetShadowMonsters in that it takes two parameters: the player's desired move and the enemy's desired move. Start will start the damage state.

There are a lot of fields. CurrrentTurn tells who's 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 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.

The constructor initializes the position fields. The interesting parts is positioning the enemy pieces. I use the Width property of Window.ClientBounds on the Game object to position them on the right side of the screen.

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 then create an array of Colors that is the width of the texture times the height of the texture. I then fill it with a color. Finally I call the SetData method passing in the array. 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 enemy is not alive or 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. If cTimer and dTimer are greater than three and two seconds 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 turn is true it is time to apply damage and switch turns. First is set to false and dTimer is set to zero. If 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 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. It is absolute and will be made relative in a future tutorial. If it is the player's turn it draws what move the player's shadow monster just used. Next there is an if that tests that the target is the enemy and the move is an attacks. 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.

The next state that I want to implement is the battle over state. It is placed on the stack when one of the shadow monsters faints. Right click the GameStates folder in the Solution Explorer,

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 ShadowMonsters.ShadowMonsters;

namespace ShadowMonsters.GameStates
{
    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)
        {
            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(GameRef.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.CheckKeyReleased(Keys.Space) || Xin.CheckKeyReleased(Keys.Enter))
    {
        if (levelUp)
        {
            manager.PushState((LevelUpState)GameRef.LevelUpState);
            GameRef.LevelUpState.SetShadowMonster(player);

            this.Visible = true;
        }
        else if (Game1.Player.Alive())
        {
            manager.PopState();
            manager.PopState();
        }
        else
        {
            manager.PopState();
            manager.PopState();
            // should warp to a location since the player has no avatars
            // with no access to the world it is hard to say where to warp
            // to
        }
    }

    base.Update(gameTime);
}

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

    base.Draw(gameTime);

    GameRef.SpriteBatch.Begin();

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

    for (int i = 0; i < 2; i++)
    {
        GameRef.SpriteBatch.DrawString(
            FontManager.GetFont("testfont"),
            battleState[i],
            position,
            Color.Black);
        position.Y += FontManager.GetFont("testfont").LineSpacing;
    }

    GameRef.SpriteBatch.DrawString(
        FontManager.GetFont("testfont"),
        battleState[2],
        position,
        Color.Red);

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

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

```
            playerHealth = (float)player.CurrentHealth / (float)player.GetHealth();
            MathHelper.Clamp(playerHealth, 0f, 1f);
            playerHealthRect.Width = (int)(playerHealth * 286);

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

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

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

            GameRef.SpriteBatch.Draw(avatarHealth, enemyHealthRect, healthSourceRect,
Color.White);
            GameRef.SpriteBatch.DrawString(
                FontManager.GetFont("testfont"),
                player.DisplayName,
                playerName, Color.
                White);
            GameRef.SpriteBatch.DrawString(
                FontManager.GetFont("testfont"),
                enemy.DisplayName,
                enemyName,
                Color.White);

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

            GameRef.SpriteBatch.End();
        }

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


            long expGained;
            if (player.Alive)
            {
                expGained = player.WinBattle(enemy);
                battleState[0] = player.DisplayName + " has won the battle!";
                battleState[1] = player.DisplayName + " has gained " + expGained + "
experience";

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

                    foreach (string s in player.KnownMoves.Keys)
                    {
                        if (player.KnownMoves[s].Unlocked == false &&
                            player.Level >= player.KnownMoves[s].UnlockedAt)
                        {
                            player.KnownMoves[s].Unlock();
                            battleState[1] += " " + s + " was unlocked!";
                        }
                    }
```

```
                    levelUp = true;
                }
                else
                {
                    battleState[1] += ".";
                }
            }
            else
            {
                expGained = player.LoseBattle(enemy);

                battleState[0] = player.DisplayName + " has lost the battle.";
                battleState[1] = player.DisplayName + " has gained " + expGained + "
experience";

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

                    foreach (string s in player.KnownMoves.Keys)
                    {
                        if (player.KnownMoves[s].Unlocked == false && player.Level >=
player.KnownMoves[s].UnlockedAt)
                        {
                            player.KnownMoves[s].Unlock();
                            battleState[1] += " " + s + " was unlocked!";
                        }
                    }

                    levelUp = true;
                }
                else
                {
                    battleState[1] += ".";
                }
            }
        }

        #endregion
    }
}
```

There is an interface like the other state. It has just 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 differences 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 was as in DamageState.

Update checks to see if space or enter have 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. 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 DamageStatel The difference being is 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 player's level is greater than or equal the unlocked at 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.

Right click the GameStates folder, select Add and then Class. Name this new class LevelUpState. Here is the code.

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

namespace ShadowMonsters.GameStates
{
    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)
{
    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(GameRef.GraphicsDevice, 500, 400);

    Color[] buffer = new Color[500 * 400];

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

    levelUpBackground.SetData(buffer);

    destination = new Rectangle(
        (GameRef.Window.ClientBounds.Width - levelUpBackground.Width) / 2,
        (GameRef.Window.ClientBounds.Height - levelUpBackground.Height) / 2,
        levelUpBackground.Width,
        levelUpBackground.Height);

    base.LoadContent();
}

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

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

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

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

        if (Xin.CheckKeyReleased(Keys.Space) || Xin.CheckKeyReleased(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.CheckKeyReleased(Keys.Right) || Xin.CheckKeyReleased(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.CheckKeyReleased(Keys.Left) || Xin.CheckKeyReleased(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);


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

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

            GameRef.SpriteBatch.DrawString(FontManager.GetFont("testfont"), player.DisplayName,
textPosition, Color.White);
            textPosition.Y += FontManager.GetFont("testfont").LineSpacing * 2;

            int i = 0;

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

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

                if (s != "Done")
                {
                    GameRef.SpriteBatch.DrawString(FontManager.GetFont("testfont"), s + ":",
textPosition, tint);
                    textPosition.X += 125;

                    GameRef.SpriteBatch.DrawString(FontManager.GetFont("testfont"),
attributes[s].ToString(), textPosition, tint);
                    textPosition.X += 40;

                    GameRef.SpriteBatch.DrawString(FontManager.GetFont("testfont"),
assignedTo[s].ToString(), textPosition, tint);
                    textPosition.X = destination.X + 5;

                    textPosition.Y += FontManager.GetFont("testfont").LineSpacing;
                }
                else
                {
                    GameRef.SpriteBatch.DrawString(FontManager.GetFont("testfont"), "Done",
textPosition, tint);
                    textPosition.Y += FontManager.GetFont("testfont").LineSpacing * 2;
                }
                i++;
            }

            GameRef.SpriteBatch.DrawString(
                FontManager.GetFont("testfont"),
                points.ToString() + " point left.",
                textPosition,
                Color.White);
            GameRef.SpriteBatch.End();
        }

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

            attributes["Attack"] = player.BaseAttack;
            attributes["Defense"] = player.BaseDefense;
            attributes["Speed"] = player.BaseSpeed;
            attributes["Health"] = player.BaseHealth;

            assignedTo["Attack"] = random.Next(1, 7);
            assignedTo["Defense"] = random.Next(1, 7);
            assignedTo["Speed"] = random.Next(1, 7);
            assignedTo["Health"] = random.Next(5, 21);

            points = 3;
            selected = 0;
        }

        #endregion
    }
}
```

Again there is an interface that the class will implement. This time the single method is SetShadowMonster that takes 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.

The constructor initializes the two collections. The first collection is done manually adding in the attributes one at a time. The other is initialized using the first, The LoadContent method creates the background texture like 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 key 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 key. 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 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 is 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 that X and Y position of the background by 5 pixels. It draws the name of the shadow monster 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 monsters 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.

The last state that I'm going to implement is the battle state. Right click the GameStates 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 ShadowMonsters.ConversationComponents;
using ShadowMonsters.ShadowMonsters;
using System.Collections.Generic;

namespace ShadowMonsters.GameStates
{
    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)
        {
```

```
            playerRect = new Rectangle(10, 90, 300, 300);
            enemyRect = new Rectangle(GameRef.Window.ClientBounds.Width - 310, 10, 300, 300);

            playerBorderRect = new Rectangle(10, 10, 300, 75);
            enemyBorderRect = new Rectangle(GameRef.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()
        {
            if (combatScene == null)
            {
                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);

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

    base.LoadContent();
}

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

    combatScene.Update();

    if (Xin.CheckKeyReleased(Keys.Space) ||
        Xin.CheckKeyReleased(Keys.Enter) ||
        (Xin.CheckMouseReleased(MouseButtons.Left) &&
        combatScene.IsMouseOver))
    {
        manager.PushState((DamageState)GameRef.DamageState);
        GameRef.DamageState.SetShadowMonsters(player, enemy);

        IMove enemyMove = null;

        do
        {
            int move = random.Next(0, enemy.KnownMoves.Count);
            int i = 0;

            foreach (string s in enemy.KnownMoves.Keys)
            {
                if (move == i)
                {
                    enemyMove = (IMove)enemy.KnownMoves[s].Clone();
                }

                i++;
            }

        } while (!enemyMove.Unlocked);

        GameRef.DamageState.SetMoves(
            (IMove)player.KnownMoves[combatScene.OptionText].Clone(),
            enemyMove);
        GameRef.DamageState.Start();

        player.Update();
        enemy.Update();
    }

    Visible = true;

    base.Update(gameTime);
}
```

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

            GameRef.SpriteBatch.Begin();

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

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

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

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

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

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

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

            GameRef.SpriteBatch.Draw(avatarHealth, enemyHealthRect, healthSourceRect,
Color.White);
            GameRef.SpriteBatch.DrawString(
                FontManager.GetFont("testfont"),
                player.DisplayName,
                playerName,
                Color.White);
            GameRef.SpriteBatch.DrawString(
                FontManager.GetFont("testfont"),
                enemy.DisplayName,
                enemyName,
                Color.White);

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

            GameRef.SpriteBatch.End();
        }

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

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

            List<SceneOption> moves = new List<SceneOption>();

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

```
            foreach (string s in player.KnownMoves.Keys)
            {
                if (player.KnownMoves[s].Unlocked)
                {
                    SceneOption option = new SceneOption(s, s, new SceneAction());
                    moves.Add(option);
                }
            }

            combatScene.SelectedIndex = 0;
            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 List<SceneOption>();

            foreach (string s in player.KnownMoves.Keys)
            {
                if (player.KnownMoves[s].Unlocked)
                {
                    SceneOption option = new SceneOption(s, s, new SceneAction());
                    moves.Add(option);
                }
            }

            combatScene.SelectedIndex = 0;
            combatScene.Options = moves;
        }

        #endregion
    }
}
```

First, 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 then loops through known moves to get the move. I do this until I find an unlocked move. 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 on them. 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 if the move is unlocked I add 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 is set to moves. StartBattle calls the StartCombat methods of the objects and initializes the playerHealth and enemyHealth fields.

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

That is it for the new scenes, Now it is time to implement them in the game. The first step is to create fields and properties in the Game1 class and initialize them Update the Game1 class to the following,

```
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using ShadowMonsters.ConversationComponents;
using ShadowMonsters.GameStates;
using ShadowMonsters.TileEngine;
using System;
using System.Collections.Generic;

namespace ShadowMonsters
{
    /// <summary>
    /// This is the main type for your game.
    /// </summary>
    public class Game1 : Game
    {
        public static Player Player;
        public static Random Random = new Random();
        private static Dictionary<AnimationKey, Animation> animations = new
Dictionary<AnimationKey, Animation>();
        private readonly GraphicsDeviceManager graphics;
        private SpriteBatch spriteBatch;
        private readonly GamePlayState gamePlayState;
        private readonly ConversationState conversationState;
        private readonly LevelUpState levelUpState;
        private readonly BattleOverState battleOverState;
        private readonly DamageState damageState;
        private readonly BattleState battleState;
        private readonly GameStateManager stateManager;
```

```
public SpriteBatch SpriteBatch => spriteBatch;
public GamePlayState GamePlayState => gamePlayState;
public ConversationState ConversationState => conversationState;
public LevelUpState LevelUpState => levelUpState;
public BattleOverState BattleOverState => battleOverState;
public DamageState DamageState => damageState;
public BattleState BattleState => battleState;

public static Dictionary<AnimationKey, Animation> Animations => animations;

public Game1()
{
    graphics = new GraphicsDeviceManager(this)
    {
        PreferredBackBufferWidth = 1280,
        PreferredBackBufferHeight = 720
    };

    graphics.ApplyChanges();

    Content.RootDirectory = "Content";

    stateManager = new GameStateManager(this);
    Components.Add(stateManager);

    gamePlayState = new GamePlayState(this);
    conversationState = new ConversationState(this);
    levelUpState = new LevelUpState(this);
    damageState = new DamageState(this);
    battleOverState = new BattleOverState(this);
    battleState = new BattleState(this);

    stateManager.PushState(gamePlayState);
    ConversationManager.Instance.CreateConversations(this);
    IsMouseVisible = true;
}

/// <summary>
/// Allows the game to perform any initialization it needs to before starting to run.
/// This is where it can query for any required services and load any non-graphic
/// related content.  Calling base.Initialize will enumerate through any components
/// and initialize them as well.
/// </summary>
protected override void Initialize()
{
    // TODO: Add your initialization logic here

    Animation animation = new Animation(3, 32, 36, 0, 0);
    animations.Add(AnimationKey.WalkUp, animation);

    animation = new Animation(3, 32, 36, 0, 36);
    animations.Add(AnimationKey.WalkRight, animation);

    animation = new Animation(3, 32, 36, 0, 72);
    animations.Add(AnimationKey.WalkDown, animation);

    animation = new Animation(3, 32, 36, 0, 108);
    animations.Add(AnimationKey.WalkLeft, animation);
```

```
            Components.Add(new Xin(this));
            Components.Add(new FontManager(this));

            Game1.Player = new Player(this, "Bonnie", true, @"Sprites\mage_f");
            base.Initialize();
        }

        /// <summary>
        /// LoadContent will be called once per game and is the place to load
        /// all of your content.
        /// </summary>
        protected override void LoadContent()
        {
            // Create a new SpriteBatch, which can be used to draw textures.
            spriteBatch = new SpriteBatch(GraphicsDevice);

            // TODO: use this.Content to load your game content here
        }

        /// <summary>
        /// UnloadContent will be called once per game and is the place to unload
        /// game-specific content.
        /// </summary>
        protected override void UnloadContent()
        {
            // TODO: Unload any non ContentManager content here
        }

        /// <summary>
        /// Allows the game to run logic such as updating the world,
        /// checking for collisions, gathering input, and playing audio.
        /// </summary>
        /// <param name="gameTime">Provides a snapshot of timing values.</param>
        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);
        }

        /// <summary>
        /// This is called when the game should draw itself.
        /// </summary>
        /// <param name="gameTime">Provides a snapshot of timing values.</param>
        protected override void Draw(GameTime gameTime)
        {
            GraphicsDevice.Clear(Color.CornflowerBlue);

            // TODO: Add your drawing code here

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

Now it is time to trigger a battle with Paul. We will start a battle with an NPC if the player presses the B key. Change the Update method of the GamePlayState to the following.

```
        public override void Update(GameTime gameTime)
        {
            engine.Update(gameTime);
            frameCount++;
            if (Xin.KeyboardState.IsKeyDown(Keys.W) && !inMotion)
            {
                motion.Y = -1;
                Game1.Player.Sprite.CurrentAnimation = AnimationKey.WalkUp;
                Game1.Player.Sprite.IsAnimating = true;
                inMotion = true;
                collision = new Rectangle(
                    (int)Game1.Player.Sprite.Position.X,
                    (int)Game1.Player.Sprite.Position.Y - Engine.TileHeight * 2,
                    Engine.TileWidth,
                    Engine.TileHeight);
            }
            else if (Xin.KeyboardState.IsKeyDown(Keys.S) && !inMotion)
            {
                motion.Y = 1;
                Game1.Player.Sprite.CurrentAnimation = AnimationKey.WalkDown;
                Game1.Player.Sprite.IsAnimating = true;
                inMotion = true;
                collision = new Rectangle(
                    (int)Game1.Player.Sprite.Position.X,
                    (int)Game1.Player.Sprite.Position.Y + Engine.TileHeight * 2,
                    Engine.TileWidth,
                    Engine.TileHeight);
            }
            else if (Xin.KeyboardState.IsKeyDown(Keys.A) && !inMotion)
            {
                motion.X = -1;
                Game1.Player.Sprite.CurrentAnimation = AnimationKey.WalkLeft;
                Game1.Player.Sprite.IsAnimating = true;
                inMotion = true;
                collision = new Rectangle(
                    (int)Game1.Player.Sprite.Position.X - Engine.TileWidth * 2,
                    (int)Game1.Player.Sprite.Position.Y,
                    Engine.TileWidth,
                    Engine.TileHeight);
            }
            else if (Xin.KeyboardState.IsKeyDown(Keys.D) && !inMotion)
            {
                motion.X = 1;
                Game1.Player.Sprite.CurrentAnimation = AnimationKey.WalkRight;
                Game1.Player.Sprite.IsAnimating = true;
                inMotion = true;
                collision = new Rectangle(
                    (int)Game1.Player.Sprite.Position.X + Engine.TileWidth * 2,
                    (int)Game1.Player.Sprite.Position.Y,
                    Engine.TileWidth,
                    Engine.TileHeight);
            }


            if (motion != Vector2.Zero)
            {
                motion.Normalize();
                motion *= (Game1.Player.Sprite.Speed *
(float)gameTime.ElapsedGameTime.TotalSeconds);
```

```
    Rectangle pRect = new Rectangle(
            (int)(Game1.Player.Sprite.Position.X + motion.X),
            (int)(Game1.Player.Sprite.Position.Y + motion.Y),
            Engine.TileWidth,
            Engine.TileHeight);

    if (pRect.Intersects(collision))
    {
        Game1.Player.Sprite.IsAnimating = false;
        inMotion = false;
        motion = Vector2.Zero;
    }

    foreach (Point p in engine.Map.CharacterLayer.Characters.Keys)
    {
        Rectangle r = new Rectangle(
            p.X * Engine.TileWidth,
            p.Y * Engine.TileHeight,
            Engine.TileWidth,
            Engine.TileHeight);

        if (r.Intersects(pRect))
        {
            motion = Vector2.Zero;
            Game1.Player.Sprite.IsAnimating = false;
            inMotion = false;
        }
    }

    Vector2 newPosition = Game1.Player.Sprite.Position + motion;
    newPosition.X = (int)newPosition.X;
    newPosition.Y = (int)newPosition.Y;

    Game1.Player.Sprite.Position = newPosition;
    motion = Game1.Player.Sprite.LockToMap(
        new Point(
            map.WidthInPixels,
            map.HeightInPixels),
        motion);

    if (motion == Vector2.Zero)
    {
        Vector2 origin = new Vector2(
                Game1.Player.Sprite.Position.X + Game1.Player.Sprite.Origin.X,
                Game1.Player.Sprite.Position.Y + Game1.Player.Sprite.Origin.Y);
        Game1.Player.Sprite.Position = Engine.VectorFromOrigin(origin);
        inMotion = false;
        Game1.Player.Sprite.IsAnimating = false;
    }
}

if ((Xin.CheckKeyReleased(Keys.Space) ||
    Xin.CheckKeyReleased(Keys.Enter)) && frameCount >= 5)
{
    frameCount = 0;
    foreach (Point s in engine.Map.CharacterLayer.Characters.Keys)
    {
        Character c = engine.Map.CharacterLayer.Characters[s];

        AnimationKey animation = Game1.Player.Sprite.CurrentAnimation;
```

```
            if (animation == AnimationKey.WalkLeft &&
                ((int)c.Sprite.Position.X > (int)Game1.Player.Sprite.Position.X ||
                    (int)c.Sprite.Position.Y != (int)Game1.Player.Sprite.Position.Y))
            {
                continue;
            }

            if (animation == AnimationKey.WalkUp &&
                ((int)c.Sprite.Position.X != (int)Game1.Player.Sprite.Position.X ||
                    (int)c.Sprite.Position.Y > (int)Game1.Player.Sprite.Position.Y))
            {
                continue;
            }

            if (animation == AnimationKey.WalkRight &&
                ((int)c.Sprite.Position.X < (int)Game1.Player.Sprite.Position.X ||
                    (int)c.Sprite.Position.Y != (int)Game1.Player.Sprite.Position.Y))
            {
                continue;
            }

            if (animation == AnimationKey.WalkDown &&
                ((int)c.Sprite.Position.X != (int)Game1.Player.Sprite.Position.X ||
                    (int)c.Sprite.Position.Y < (int)Game1.Player.Sprite.Position.Y))
            {
                continue;
            }

            float distance = Vector2.Distance(
                Game1.Player.Sprite.Origin + Game1.Player.Sprite.Position,
                c.Sprite.Origin + c.Sprite.Position);

            if (Math.Abs(distance) < Engine.TileWidth + Engine.TileWidth / 2)
            {
                manager.PushState(
                    (ConversationState)GameRef.ConversationState);

                GameRef.ConversationState.SetConversation(c);
                GameRef.ConversationState.StartConversation();
                break;
            }
        }
    }
    if ((Xin.CheckKeyReleased(Keys.B)) && frameCount >= 5)
    {
        frameCount = 0;
        foreach (Point s in engine.Map.CharacterLayer.Characters.Keys)
        {
            Character c = engine.Map.CharacterLayer.Characters[s];

            AnimationKey animation = Game1.Player.Sprite.CurrentAnimation;

            if (animation == AnimationKey.WalkLeft &&
                ((int)c.Sprite.Position.X > (int)Game1.Player.Sprite.Position.X ||
                    (int)c.Sprite.Position.Y != (int)Game1.Player.Sprite.Position.Y))
            {
                continue;
            }
```

```
                    if (animation == AnimationKey.WalkUp &&
                        ((int)c.Sprite.Position.X != (int)Game1.Player.Sprite.Position.X ||
                            (int)c.Sprite.Position.Y > (int)Game1.Player.Sprite.Position.Y))
                    {
                        continue;
                    }

                    if (animation == AnimationKey.WalkRight &&
                        ((int)c.Sprite.Position.X < (int)Game1.Player.Sprite.Position.X ||
                            (int)c.Sprite.Position.Y != (int)Game1.Player.Sprite.Position.Y))
                    {
                        continue;
                    }

                    if (animation == AnimationKey.WalkDown &&
                        ((int)c.Sprite.Position.X != (int)Game1.Player.Sprite.Position.X ||
                            (int)c.Sprite.Position.Y < (int)Game1.Player.Sprite.Position.Y))
                    {
                        continue;
                    }

                    float distance = Vector2.Distance(
                        Game1.Player.Sprite.Origin + Game1.Player.Sprite.Position,
                        c.Sprite.Origin + c.Sprite.Position);

                    if (Math.Abs(distance) < Engine.TileWidth + Engine.TileWidth / 2)
                    {
                        GameRef.BattleState.SetShadowMonsters(
                            Game1.Player.Selected,
                            c.BattleMonster);
                        manager.PushState(GameRef.BattleState);
                        break;
                    }
                }
            }

            Engine.Camera.LockToSprite(map, Game1.Player.Sprite, new Rectangle(0, 0, 1280,
720));
            Game1.Player.Update(gameTime);

            base.Update(gameTime);
        }
```

The new code works like if we were starting a conversation. If the B key is pressed we loop over all of the characters on the map. If we are facing the character and we are within distance we set the shadow monsters then push the battle state onto the stack,

I'm going to wrap this tutorial up here. It was quite a long one and we covered a lot. I will be starting work on the next tutorial shortly. Keep checking back on the blog for news on that tutorial. I hope to have it up in the next week or so.

I wish you the best in your MonoGame Programming Adventures!