Shadow Monsters – MonoGame Tutorial Series
Chapter 7
Player Component

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 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.

Before I get to the player component I want to address the conversation state briefly. There can be the same cascade problem with input as in the last tutorial. For that reason I implemented the frame count. Add the following field and change the Update method of ConversationState to the following.

```
private int frameCount = 0;

public override void Update(GameTime gameTime)
{
    if (conversation.CurrentScene == null)
    {
        return;
    }

    frameCount++;

    if ((Xin.CheckKeyReleased(Keys.Space) ||
        Xin.CheckKeyReleased(Keys.Enter) ||
        (Xin.CheckMouseReleased(MouseButtons.Left) &&
        conversation.CurrentScene.IsMouseOver)) && frameCount > 5)
    {
        frameCount = 0;
        switch (conversation.CurrentScene.OptionAction.Action)
        {
            case ActionType.Teach:
                BuyShadowMonster();
                break;
            case ActionType.Change:
                speaker.SetConversation(conversation.CurrentScene.OptionScene);
                manager.PopState();
```

```
                break;
            case ActionType.End:
                manager.PopState();
                break;
            case ActionType.GiveItems:
                break;
            case ActionType.GiveKey:
                if (conversation.CurrentScene.OptionAction.Parameter != null)
                {
                    bool success = int.TryParse(
                        conversation.CurrentScene.OptionAction.Parameter,
                        out int key);

                    if (success)
                    {
                    }

                    conversation.ChangeScene(conversation.CurrentScene.OptionScene);
                }
                break;
            case ActionType.Quest:
                CheckQuest();
                break;
            case ActionType.Rest:
                conversation.ChangeScene(conversation.CurrentScene.OptionScene);
                break;
            case ActionType.Shop:
                break;
            case ActionType.Talk:
                conversation.ChangeScene(conversation.CurrentScene.OptionScene);
                break;
        }
    }

    conversation.Update();
    base.Update(gameTime);
}
```

In this tutorial I will be creating a component that will house all of the information needed for the player. To get started right click the ShadowMonsters project in the Solution Explorer, select Add and then Class. Name this new class Player. Here is the code for the player class.

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

namespace ShadowMonsters
{
    public class Player : DrawableGameComponent
    {
        public const int MaxShadowMonsters = 6;

        #region Field Region
```

```
        private Game1 gameRef;
        private string name;
        private bool gender;
        private string mapName;
        private Point tile;
        private AnimatedSprite sprite;
        private Texture2D texture;
        private string textureName;
        private float speed = 256f;

        private Vector2 position;

        private readonly List<ShadowMonster> shadowMonsters = new List<ShadowMonster>();
        private int currentShadowMonster;
        private readonly ShadowMonster[] battleShadowMonsters = new
ShadowMonster[MaxShadowMonsters];
        private int selected;
        private readonly Dictionary<string, string> _charactersMet = new Dictionary<string,
string>();
        private readonly Dictionary<int, string> _keysFound = new Dictionary<int, string>();

        #endregion

        #region Property Region

        public ShadowMonster[] BattleShadowMonsters
        {
            get { return battleShadowMonsters; }
        }

        public ShadowMonster Selected
        {
            get { return battleShadowMonsters[selected]; }
        }

        public Dictionary<int, string> KeysFound => _keysFound;

        public Dictionary<string, string> CharactersMet => _charactersMet;

        public int Gold { get; internal set; }
        public Vector2 Position
        {
            get { return sprite.Position; }
            set { sprite.Position = value; }
        }

        public AnimatedSprite Sprite
        {
            get { return sprite; }
        }

        public float Speed
        {
            get { return speed; }
            set { speed = value; }
        }

        public string MapName
        {
```

```
    get { return mapName; }
    set { mapName = value; }
}

#endregion

#region Constructor Region

private Player(Game game)
    : base(game)
{
}

public Player(Game game, string name, bool gender, string textureName)
    : base(game)
{
    gameRef = (Game1)game;
    this.name = name;
    this.gender = gender;

    this.textureName = textureName;
    sprite = new AnimatedSprite(
        game.Content.Load<Texture2D>(textureName),
        Game1.Animations)
    {
        CurrentAnimation = AnimationKey.WalkDown
    };
    Gold = 1000;
}

#endregion

#region Method Region
public virtual void AddShadowMonster(ShadowMonster shadowMonster)
{
    shadowMonsters.Add(shadowMonster);
}

public void SetCurrentShadowMonster(int index)
{
    if (index < 0 || index >= MaxShadowMonsters)
        throw new IndexOutOfRangeException();

    if (battleShadowMonsters[index] != null)
        selected = index;
}

public ShadowMonster GetShadowMonster(int index)
{
    if (index < 0 || index >= MaxShadowMonsters)
        throw new IndexOutOfRangeException();

    return shadowMonsters[index];
}
internal bool Alive()
{
    for (int i = 0; i < MaxShadowMonsters; i++)
        if (battleShadowMonsters[i] != null && battleShadowMonsters[i].Alive)
            return true;
```

```
            return false;
        }

        public ShadowMonster GetBattleShadowMonster(int index)
        {
            if (index < 0 || index > MaxShadowMonsters)
                throw new IndexOutOfRangeException();

            return battleShadowMonsters[index];
        }

        internal void HealBattleShadowMonsters()
        {
            foreach (ShadowMonster a in battleShadowMonsters)
            {
                if (a != null)
                {
                    a.Heal(a.GetHealth());
                    a.IsAsleep = false;
                    a.IsConfused = false;
                    a.IsFainted = false;
                    a.IsParalyzed = false;
                    a.IsPoisoned = false;
                }
            }
        }

        internal void AddKey(int key, string name)
        {
            if (!KeysFound.ContainsKey(key))
                KeysFound.Add(key, name);
        }

        public override void Update(GameTime gameTime)
        {
            sprite.Update(gameTime);
            base.Update(gameTime);
        }

        public override void Draw(GameTime gameTime)
        {
            sprite.Draw(gameTime, gameRef.SpriteBatch);
            base.Draw(gameTime);
        }

        #endregion
    }
}
```

This class is a component so it inherits from DrawableGameComponent. It gives access to an Update and Draw method so we can update and draw the sprite.

There are a lot of fields in this class. gameRef is a reference to the Game1 class so that we have access to a SpriteBatch object. The name field is the name of the player. gender is the gender of the player. False will be male and true will be female. The mapName field is the name of the map the player is on. The tile field is what tile the player is on. Next is the AnimatedSprite, sprite. The texture field is the texture of the sprite and textureName is its name. speed is the speed of the player. The position field is the position of the player. The

next field, shadowMonsters, is the shadow monsters that the player has. The battleShadowMonsters field is the, up to six, shadow monsters that the player is carrying. The selected field is the index of the current shadow monster. The _charactersMet field is for future use when we need to track what characters the player has met. Finally, _keysFound is the keys that the player has found.

There are properties to expose the values of the fields. The interesting one is the Selected property. It returns a ShadowMonster instead of an int.

The AddShadowMonster method is used to add a new shadow monster to the list of shadow monsters the player owns. In this case I do not clone because we want that exact shadow monster. The SetCurrentShadowMonster method is to set the active shadow monster for battle. The GetShadowMonster method gets a shadow monster from the list of shadow monsters the player owns. The HealBattleShadowMonsters method is used to heal the current shadow monsters the player is carrying. It loops over the array of battle shadow monsters. If the shadow monster is not null it calls the Heal method then rests all of the status fields. The AddKey method adds a key to the player's key ring. The Update method calls the Update method of the sprite and the Draw method calls the Draw method of the sprite.

The next step in adding the component is to create a Player object. I placed it in the Game1 class as a static object. This way all classes can reference it using the class name. Add the following field to the Game1 class and update the Initialize method to create the component.

```
public static Player Player;

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();
}
```

The next step is to update the GamePlayState class to use the new component. There were a number of changes so I will give you the code for the entire class.

```
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;
```

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

namespace ShadowMonsters.GameStates
{
    public class GamePlayState : BaseGameState
    {
        private readonly Engine engine = new Engine(new Rectangle(0, 0, 1280, 720));
        private TileMap map;
        private Vector2 motion;
        private bool inMotion;
        private Rectangle collision;
        private ShadowMonsterManager monsterManager = new ShadowMonsterManager();
        private int frameCount = 0;

        public GamePlayState(Game game) : base(game)
        {
        }

        protected override void LoadContent()
        {
            MoveManager.FillMoves();
            ShadowMonsterManager.FromFile(@".\Content\ShadowMonsters.txt", content);
            TileSet set = new TileSet();
            set.TextureNames.Add("tileset1");
            set.Textures.Add(content.Load<Texture2D>(@"Tiles\tileset16-outdoors"));

            TileLayer groundLayer = new TileLayer(100, 100, 0, 1);
            TileLayer edgeLayer = new TileLayer(100, 100);
            TileLayer buildingLayer = new TileLayer(100, 100);
            TileLayer decorationLayer = new TileLayer(100, 100);

            for (int i = 0; i < 1000; i++)
            {
                decorationLayer.SetTile(random.Next(0, 100), random.Next(0, 100), 0,
random.Next(2, 4));
            }

            map = new TileMap(set, groundLayer, edgeLayer, buildingLayer, decorationLayer,
"level1");

            Character c = Character.FromString(GameRef,
"Paul,ninja_m,WalkDown,PaulHello,0,fire1,,,,,,,");
            c.Sprite.Position = new Vector2(2 * Engine.TileWidth, 2 * Engine.TileHeight);

            map.CharacterLayer.Characters.Add(new Point(2, 2), c);

            engine.SetMap(map);
            base.LoadContent();
        }

        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;
                }
            }
        }

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

        base.Update(gameTime);
    }

    public override void Draw(GameTime gameTime)
    {
        base.Draw(gameTime);
        engine.Draw(gameTime, GameRef.SpriteBatch);
        GameRef.SpriteBatch.Begin(
            SpriteSortMode.Deferred,
            BlendState.AlphaBlend,
            SamplerState.PointClamp,
            null,
            null,
            null,
```

```
                Engine.Camera.Transformation);
            Game1.Player.Draw(gameTime);
            GameRef.SpriteBatch.End();
        }
    }
}
```

The old sprite field was removed. References to sprite.Position have been replaced with the new component, Game1.Player.Sprite.Position. The calls to Update and Draw have been replaced with Game1.Player.Update and Game1.Player.Draw.

The last thing that I'm going to tackle is updating the BuyShadowMonster method to add the new shadow monster to the player's list of shadow monsters. Update that code to the following.

```
private void BuyShadowMonster()
{
    Game1.Player.AddShadowMonster(speaker.GiveMonster);

    for (int i = 0; i < Player.MaxShadowMonsters; i++)
    {
        if (Game1.Player.BattleShadowMonsters[i] == null)
        {
            Game1.Player.BattleShadowMonsters[i] = speaker.GiveMonster;
            break;
        }
    }

    string scene = conversation.CurrentScene.OptionScene;

    conversation.CurrentScene.Options.RemoveAt(conversation.CurrentScene.SelectedIndex);
    conversation.CurrentScene.SelectedIndex--;
    conversation.ChangeScene(scene);
}
```

The first step is to add the shadow monster to the list of shadow monsters the player owns. Next there is a for loop that loops over the battle shadow monster. If there is no shadow monster at that spot I assign it to be the giving shadow monster and break out of the loop.

I'm going to wrap this tutorial up here. I know it is a bit shorter but I don't want to start something new. 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!