Shadow Monsters – MonoGame Tutorial Series
Chapter 22
Character Generator

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. 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.

So, there were problems with the definition form that missed vetting. I used the wrong symbol when converting text to strings. I missed adding a couple fields when creating the string. Open the code view for the DefinitionForm by right clicking it in the Solution Explorer and select View Code. Replace the code with the following.

```
using ShadowMonsters.ShadowMonsters;
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace ShadowEditor
{
    public partial class DefinitionForm : Form
    {
        public string ShadowMonster { get; private set; }
        public bool OkPressed { get; private set; }

        public DefinitionForm()
        {
            InitializeComponent();

            this.Load += DefinitionForm_Load;

            foreach (var v in Enum.GetNames(typeof(ShadowMonsterElement)))
            {
                cboElement.Items.Add(v);
```

```
    }

    cboElement.SelectedIndex = 0;

    BtnOK.Click += BtnOK_Click;
    BtnCancel.Click += BtnCancel_Click;
    BtnAdd.Click += BtnAdd_Click;
    BtnRemove.Click += BtnRemove_Click;
}

public DefinitionForm(ShadowMonster m) : this()
{
    TxtKey.Text = m.Name;
    TxtName.Text = m.DisplayName;
    cboElement.SelectedItem = m.Element;
    TxtCost.Text = m.Cost.ToString();
    TxtLevel.Text = m.Level.ToString();
    TxtAttack.Text = m.BaseAttack.ToString();
    TxtDefense.Text = m.BaseDefense.ToString();
    TxtSpeed.Text = m.BaseSpeed.ToString();
    TxtHealth.Text = m.BaseHealth.ToString();

    foreach (IMove move in m.KnownMoves.Values)
    {
        LBMoves.Items.Add(move.Name + ":" + move.UnlockedAt);
    }
}

private void BtnOK_Click(object sender, EventArgs e)
{
    if (string.IsNullOrEmpty(TxtName.Text))
    {
        MessageBox.Show("You must enter a name for the shadow monster.");
        return;
    }

    if (string.IsNullOrEmpty(TxtKey.Text))
    {
        MessageBox.Show("You must enter the name of the shadow monster.");
        return;
    }

    if (string.IsNullOrEmpty(TxtCost.Text)
        || !int.TryParse(TxtCost.Text, out int cost))
    {
        MessageBox.Show("You must enter a numeric value for cost");
        return;
    }

    if (string.IsNullOrEmpty(TxtLevel.Text)
        || !int.TryParse(TxtLevel.Text, out int level))
    {
        MessageBox.Show("You must enter a numeric value for level,");
        return;
    }

    if (string.IsNullOrEmpty(TxtAttack.Text)
        || !int.TryParse(TxtAttack.Text, out int attack))
    {
        MessageBox.Show("You must enter a numeric value for attack,");
```

```
                return;
            }

            if (string.IsNullOrEmpty(TxtDefense.Text)
                || !int.TryParse(TxtDefense.Text, out int defense))
            {
                MessageBox.Show("You must enter a numeric value for defense.");
                return;
            }

            if (string.IsNullOrEmpty(TxtSpeed.Text)
                || !int.TryParse(TxtSpeed.Text, out int speed))
            {
                MessageBox.Show("You must enter a numeric value for speed,");
                return;
            }

            if (string.IsNullOrEmpty(TxtHealth.Text)
                || !int.TryParse(TxtHealth.Text, out int health))
            {
                MessageBox.Show("You must enter a numeric value for health,");
                return;
            }

            StringBuilder sb = new StringBuilder();

            sb.Append(TxtKey.Text + "," + TxtName.Text + "," + cboElement.SelectedItem + "," +
cost + ",");
            sb.Append(level + "," + attack + "," + defense + "," + speed + "," + health +
",0,0");

            foreach (object o in LBMoves.Items)
            {
                string s = o.ToString();
                sb.Append("," + s);
            }

            ShadowMonster = sb.ToString();
            OkPressed = true;
            Close();
        }

        private void BtnCancel_Click(object sender, EventArgs e)
        {
            Close();
        }

        private void BtnAdd_Click(object sender, EventArgs e)
        {
            LBMoves.Items.Add(CboMoves.SelectedItem.ToString() + ":" + TxtMoveLevel.Text);

        }

        private void BtnRemove_Click(object sender, EventArgs e)
        {
            if (LBMoves.SelectedIndex < 0)
                return;

            LBMoves.Items.RemoveAt(LBMoves.SelectedIndex);
        }
```

```
        private void DefinitionForm_Load(object sender, EventArgs e)
        {
            MoveManager.FillMoves();

            foreach (IMove move in MoveManager.Moves.Values)
            {
                CboMoves.Items.Add(move.Name);
            }

            CboMoves.SelectedIndex = 0;
        }
    }
}
```

As I mentioned on the blog there was a flaw with my scaling method. I used integer division instead of floating point division. This will have most resolutions map 1 to 1. I remedied this by adding a Scale property to the Settings class. It just casts to a float when calculating the scaling ratio. Add the following property to the Settings class.

```
        public static Vector2 Scale
        {
            get
            {
                return new Vector2(
                    (float)resolution.X / 1280,
                    (float)resolution.Y / 720);
            }
        }
```

Just a few minor changes. When creating the form when a definition is passed in I set the Text property of TxtLevel to the level in the definition. When doing the TryParse I not the value. I also updated the code where I create the string from the definition. I added the level after cost, as per the FromString method, and I also append the element. I also set the initial value of the drop downs to 0.

What I'm going to work on in this tutorial is creating a character for the player to control in a character generator rather than hard coded values. I will allow the player to enter a name so I added a text box control. Right click the Controls folder in the Solution Explorer, select Add and then Class. Name this new class TextBox. Here is the code.

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

namespace ShadowMonsters.Controls
{
    public class TextBox : Control
    {
        #region Event Region
```

```
        public event EventHandler Changed;

        #endregion

        #region Field Region

        private readonly Texture2D _background;

        #endregion

        #region Property Region

        #endregion

        #region Constructor Region

        public TextBox(Texture2D background)
        {
            _background = background;
            _text = "";
        }

        #endregion

        #region Method Region

        public override void Draw(SpriteBatch spriteBatch)
        {
            spriteBatch.Draw(_background, Position, Color.White);
            spriteBatch.DrawString(FontManager.GetFont("testfont"), _text, Position +
Vector2.One * 3, Color.Black);
        }

        public override void HandleInput()
        {
            if (Xin.CheckKeyPressed(Keys.A))
            {
                if (!(Xin.IsKeyDown(Keys.LeftShift) || Xin.IsKeyDown(Keys.RightShift)))
                {
                    _text += "a";
                }
                else
                {
                    _text += "A";
                }
                OnChange();
            }
            if (Xin.CheckKeyPressed(Keys.B))
            {
                if (!(Xin.IsKeyDown(Keys.LeftShift) || Xin.IsKeyDown(Keys.RightShift)))
                {
                    _text += "b";
                }
                else
                {
                    _text += "B";
                }
                OnChange();
            }
            if (Xin.CheckKeyPressed(Keys.C))
```

```
{
    if (!(Xin.IsKeyDown(Keys.LeftShift) || Xin.IsKeyDown(Keys.RightShift)))
    {
        _text += "c";
    }
    else
    {
        _text += "C";
    }
    OnChange();
}
if (Xin.CheckKeyPressed(Keys.D))
{
    if (!(Xin.IsKeyDown(Keys.LeftShift) || Xin.IsKeyDown(Keys.RightShift)))
    {
        _text += "d";
    }
    else
    {
        _text += "D";
    }
    OnChange();
}
if (Xin.CheckKeyPressed(Keys.E))
{
    if (!(Xin.IsKeyDown(Keys.LeftShift) || Xin.IsKeyDown(Keys.RightShift)))
    {
        _text += "e";
    }
    else
    {
        _text += "E";
    }
    OnChange();
}
if (Xin.CheckKeyPressed(Keys.F))
{
    if (!(Xin.IsKeyDown(Keys.LeftShift) || Xin.IsKeyDown(Keys.RightShift)))
    {
        _text += "f";
    }
    else
    {
        _text += "F";
    }
    OnChange();
}
if (Xin.CheckKeyPressed(Keys.G))
{
    if (!(Xin.IsKeyDown(Keys.LeftShift) || Xin.IsKeyDown(Keys.RightShift)))
    {
        _text += "g";
    }
    else
    {
        _text += "G";
    }
    OnChange();
}
if (Xin.CheckKeyPressed(Keys.H))
```

```
{
    if (!(Xin.IsKeyDown(Keys.LeftShift) || Xin.IsKeyDown(Keys.RightShift)))
    {
        _text += "h";
    }
    else
    {
        _text += "H";
    }
    OnChange();
}
if (Xin.CheckKeyPressed(Keys.I))
{
    if (!(Xin.IsKeyDown(Keys.LeftShift) || Xin.IsKeyDown(Keys.RightShift)))
    {
        _text += "i";
    }
    else
    {
        _text += "I";
    }
    OnChange();
}
if (Xin.CheckKeyPressed(Keys.J))
{
    if (!(Xin.IsKeyDown(Keys.LeftShift) || Xin.IsKeyDown(Keys.RightShift)))
    {
        _text += "j";
    }
    else
    {
        _text += "J";
    }
    OnChange();
}
if (Xin.CheckKeyPressed(Keys.K))
{
    if (!(Xin.IsKeyDown(Keys.LeftShift) || Xin.IsKeyDown(Keys.RightShift)))
    {
        _text += "k";
    }
    else
    {
        _text += "K";
    }
    OnChange();
}
if (Xin.CheckKeyPressed(Keys.L))
{
    if (!(Xin.IsKeyDown(Keys.LeftShift) || Xin.IsKeyDown(Keys.RightShift)))
    {
        _text += "l";
    }
    else
    {
        _text += "L";
    }
    OnChange();
}
if (Xin.CheckKeyPressed(Keys.M))
```

```
{
    if (!(Xin.IsKeyDown(Keys.LeftShift) || Xin.IsKeyDown(Keys.RightShift)))
    {
        _text += "m";
    }
    else
    {
        _text += "M";
    }
    OnChange();
}
if (Xin.CheckKeyPressed(Keys.N))
{
    if (!(Xin.IsKeyDown(Keys.LeftShift) || Xin.IsKeyDown(Keys.RightShift)))
    {
        _text += "n";
    }
    else
    {
        _text += "N";
    }
    OnChange();
}
if (Xin.CheckKeyPressed(Keys.O))
{
    if (!(Xin.IsKeyDown(Keys.LeftShift) || Xin.IsKeyDown(Keys.RightShift)))
    {
        _text += "o";
    }
    else
    {
        _text += "O";
    }
    OnChange();
}
if (Xin.CheckKeyPressed(Keys.R))
{
    if (!(Xin.IsKeyDown(Keys.LeftShift) || Xin.IsKeyDown(Keys.RightShift)))
    {
        _text += "p";
    }
    else
    {
        _text += "P";
    }
    OnChange();
}
if (Xin.CheckKeyPressed(Keys.Q))
{
    if (!(Xin.IsKeyDown(Keys.LeftShift) || Xin.IsKeyDown(Keys.RightShift)))
    {
        _text += "q";
    }
    else
    {
        _text += "Q";
    }
    OnChange();
}
if (Xin.CheckKeyPressed(Keys.R))
```

```
{
    if (!(Xin.IsKeyDown(Keys.LeftShift) || Xin.IsKeyDown(Keys.RightShift)))
    {
        _text += "r";
    }
    else
    {
        _text += "R";
    }
    OnChange();
}
if (Xin.CheckKeyPressed(Keys.S))
{
    if (!(Xin.IsKeyDown(Keys.LeftShift) || Xin.IsKeyDown(Keys.RightShift)))
    {
        _text += "s";
    }
    else
    {
        _text += "S";
    }
    OnChange();
}
if (Xin.CheckKeyPressed(Keys.T))
{
    if (!(Xin.IsKeyDown(Keys.LeftShift) || Xin.IsKeyDown(Keys.RightShift)))
    {
        _text += "t";
    }
    else
    {
        _text += "T";
    }
    OnChange();
}
if (Xin.CheckKeyPressed(Keys.U))
{
    if (!(Xin.IsKeyDown(Keys.LeftShift) || Xin.IsKeyDown(Keys.RightShift)))
    {
        _text += "u";
    }
    else
    {
        _text += "U";
    }
    OnChange();
}
if (Xin.CheckKeyPressed(Keys.V))
{
    if (!(Xin.IsKeyDown(Keys.LeftShift) || Xin.IsKeyDown(Keys.RightShift)))
    {
        _text += "v";
    }
    else
    {
        _text += "V";
    }
    OnChange();
}
if (Xin.CheckKeyPressed(Keys.W))
```

```
{
    if (!(Xin.IsKeyDown(Keys.LeftShift) || Xin.IsKeyDown(Keys.RightShift)))
    {
        _text += "w";
    }
    else
    {
        _text += "W";
    }
    OnChange();
}
if (Xin.CheckKeyPressed(Keys.X))
{
    if (!(Xin.IsKeyDown(Keys.LeftShift) || Xin.IsKeyDown(Keys.RightShift)))
    {
        _text += "x";
    }
    else
    {
        _text += "X";
    }
    OnChange();
}
if (Xin.CheckKeyPressed(Keys.Y))
{
    if (!(Xin.IsKeyDown(Keys.LeftShift) || Xin.IsKeyDown(Keys.RightShift)))
    {
        _text += "y";
    }
    else
    {
        _text += "Y";
    }
    OnChange();
}
if (Xin.CheckKeyPressed(Keys.Z))
{
    if (!(Xin.IsKeyDown(Keys.LeftShift) || Xin.IsKeyDown(Keys.RightShift)))
    {
        _text += "z";
    }
    else
    {
        _text += "Z";
    }
    OnChange();
}
if (Xin.CheckKeyPressed(Keys.D1))
{
    if (!(Xin.IsKeyDown(Keys.LeftShift) || Xin.IsKeyDown(Keys.RightShift)))
    {
        _text += "1";
    }
    else
    {
        _text += "!";
    }
    OnChange();
}
if (Xin.CheckKeyPressed(Keys.D2))
```

```
{
    if (!(Xin.IsKeyDown(Keys.LeftShift) || Xin.IsKeyDown(Keys.RightShift)))
    {
        _text += "2";
    }
    else
    {
        _text += "@";
    }
    OnChange();
}
if (Xin.CheckKeyPressed(Keys.D3))
{
    if (!(Xin.IsKeyDown(Keys.LeftShift) || Xin.IsKeyDown(Keys.RightShift)))
    {
        _text += "3";
    }
    else
    {
        _text += "#";
    }
    OnChange();
}
if (Xin.CheckKeyPressed(Keys.D4))
{
    if (!(Xin.IsKeyDown(Keys.LeftShift) || Xin.IsKeyDown(Keys.RightShift)))
    {
        _text += "4";
    }
    else
    {
        _text += "$";
    }
    OnChange();
}
if (Xin.CheckKeyPressed(Keys.D5))
{
    if (!(Xin.IsKeyDown(Keys.LeftShift) || Xin.IsKeyDown(Keys.RightShift)))
    {
        _text += "5";
    }
    else
    {
        _text += "%";
    }
    OnChange();
}
if (Xin.CheckKeyPressed(Keys.D6))
{
    if (!(Xin.IsKeyDown(Keys.LeftShift) || Xin.IsKeyDown(Keys.RightShift)))
    {
        _text += "6";
    }
    else
    {
        _text += "^";
    }
    OnChange();
}
if (Xin.CheckKeyPressed(Keys.D7))
```

```
    {
        if (!(Xin.IsKeyDown(Keys.LeftShift) || Xin.IsKeyDown(Keys.RightShift)))
        {
            _text += "7";
        }
        else
        {
            _text += "&";
        }
        OnChange();
    }
    if (Xin.CheckKeyPressed(Keys.D8))
    {
        if (!(Xin.IsKeyDown(Keys.LeftShift) || Xin.IsKeyDown(Keys.RightShift)))
        {
            _text += "8";
        }
        else
        {
            _text += "*";
        }
        OnChange();
    }
    if (Xin.CheckKeyPressed(Keys.D9))
    {
        if (!(Xin.IsKeyDown(Keys.LeftShift) || Xin.IsKeyDown(Keys.RightShift)))
        {
            _text += "9";
        }
        else
        {
            _text += "(";
        }
        OnChange();
    }
    if (Xin.CheckKeyPressed(Keys.D0))
    {
        if (!(Xin.IsKeyDown(Keys.LeftShift) || Xin.IsKeyDown(Keys.RightShift)))
        {
            _text += "0";
        }
        else
        {
            _text += ")";
        }
        OnChange();
    }
    if (Xin.CheckKeyPressed(Keys.Back) && _text.Length > 0)
    {
        _text = _text.Substring(0, _text.Length - 1);
        OnChange();
    }
}

private void OnChange()
{
    Changed?.Invoke(this, null);
}
public override void Update(GameTime gameTime)
{
```

```
            HandleInput();
        }

        #endregion
    }
}
```

Wow, that was a wall of code. Fortunately most of it is the same code over and over again with different values. The class inherits from the Control class so it has access to all of its protected and public members. There is an event handler, Changed, that will fire when the text property changes. There is one field, _background, that is the background for the text box. In the Draw method I draw the texture then I draw the string that is the input of the text box. For the font I used the GetFont method of the FontManager like in other places I require a font.

In the HandleInput method there are a ton of if statements. They check to see if a key has been pressed or not. If the key has been pressed it checks to see if either of the shift keys are down. If the shift key is not down the lower case letter, or digit, is added to the _text field inherited from the Control class. If a shift key is down the upper case letter, or symbol, is added to the _text field. In either case the OnChange method is called. If the back space key has been pressed I set the text property to the string minus one character.

The OnChange method invokes the delegate if it is subscribed to. The Update method calls the HandleInput method.

Right now you will have a bunch of errors because I haven't added to methods to Xin. One to check is key has been pressed and the other to check if a key is down. Add these to methods to Xin.

```
        public static bool CheckKeyPressed(Keys key)
        {
            return currentKeyboardState.IsKeyDown(key) && previousKeyboardState.IsKeyUp(key);
        }

        public static bool IsKeyDown(Keys key)
        {
            return currentKeyboardState.IsKeyDown(key);
        }
```

To check for a key press you check if the key is down in the current state but up in the previous state. IsKeyDown just returns the value of IsKeyDown of the KeyboardState. It is just a short cut.

Now we can create the character generator. Right click the GameStates, select Add and then Class. Name this new class NewGameState. Here is the code.

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

```
using ShadowMonsters.Components;
using ShadowMonsters.Controls;

namespace ShadowMonsters.GameStates
{
    public class NewGameState : BaseGameState
    {
        private Texture2D _background;
        private Rectangle _destination = new Rectangle(0, 0, 1280, 720);
        private Rectangle _portraitDestination = new Rectangle(599, 57, 633, 617);
        private LeftRightSelector _portraitSelector;
        private LeftRightSelector _genderSelector;
        private TextBox _nameTextBox;
        private Dictionary<string, Texture2D> _femalePortraits;
        private Dictionary<string, Texture2D> _malePortraits;
        private Button _create;
        private Button _back;

        public NewGameState(Game game) : base(game)
        {
            _femalePortraits = new Dictionary<string, Texture2D>();
            _malePortraits = new Dictionary<string, Texture2D>();
        }

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

        protected override void LoadContent()
        {
            _background = content.Load<Texture2D>(@"chargen-back");

            _genderSelector = new LeftRightSelector(
                content.Load<Texture2D>(@"GUI\g22987"),
                content.Load<Texture2D>(@"GUI\g21245"),
                null)
            {
                Position = new Vector2(207 - 70, 298)
            };
            _genderSelector.SelectionChanged += _genderSelector_SelectionChanged;
            _genderSelector.SetItems(new[] { "Female", "Male" }, 270);

            _portraitSelector = new LeftRightSelector(
                content.Load<Texture2D>(@"GUI\g22987"),
                content.Load<Texture2D>(@"GUI\g21245"),
                null)
            {
                Position = new Vector2(207 - 70, 458)
            };
            _portraitSelector.SelectionChanged += _portraitSelector_SelectionChanged;

            _nameTextBox = new TextBox(
                content.Load<Texture2D>(@"GUI\textbox"))
            {
                Position = new Vector2(207, 138),
                HasFocus = true,
                Enabled = true,
                Color = Color.White
            };
```

```
        _femalePortraits.Add(
            "Healer",
            content.Load<Texture2D>(@"CharacterSprites\healer_f"));
        _femalePortraits.Add(
            "Mage",
            content.Load<Texture2D>(@"CharacterSprites\mage_f"));
        _femalePortraits.Add(
            "Ninja",
            content.Load<Texture2D>(@"CharacterSprites\ninja_f"));
        _femalePortraits.Add(
            "Ranger",
            content.Load<Texture2D>(@"CharacterSprites\ranger_f"));
        _malePortraits.Add(
            "Healer",
            content.Load<Texture2D>(@"CharacterSprites\healer_m"));
        _malePortraits.Add(
            "Mage",
            content.Load<Texture2D>(@"CharacterSprites\mage_m"));
        _malePortraits.Add(
            "Ninja" ,
            content.Load<Texture2D>(@"CharacterSprites\ninja_m"));
        _malePortraits.Add(
            "Ranger",
            content.Load<Texture2D>(@"CharacterSprites\ranger_m"));

        _portraitSelector.SetItems(_femalePortraits.Keys.ToArray(), 270);

        _create = new Button(
            content.Load<Texture2D>(@"GUI\g9202"))
        {
            Text = "Create",
            Position = new Vector2(180, 640)
        };

        _create.Click += _create_Click;

        _back = new Button(
            content.Load<Texture2D>(@"GUI\g9202"))
        {
            Text = "Back",
            Position = new Vector2(350, 640)
        };

        _back.Click += _back_Click;

    }

    private void _back_Click(object sender, EventArgs e)
    {
        manager.PopState();
    }

    private void _create_Click(object sender, EventArgs e)
    {
        Muse.StopSong();
        Game1.Player = new Player(
            GameRef,
            _nameTextBox.Text,
            _genderSelector.SelectedIndex == 0,
```

```
            _portraitSelector.SelectedItem);
        GameRef.GamePlayState.SetUpNewGame();
        manager.PopState();
        manager.PushState(GameRef.GamePlayState);
    }

    private void _genderSelector_SelectionChanged(object sender, EventArgs e)
    {
        if (_genderSelector.SelectedIndex == 0)
        {
            _portraitSelector.SetItems(_femalePortraits.Keys.ToArray(), 270);
        }
        else
        {
            _portraitSelector.SetItems(_malePortraits.Keys.ToArray(), 270);
        }
    }

    private void _portraitSelector_SelectionChanged(object sender, EventArgs e)
    {

    }

    public override void Update(GameTime gameTime)
    {
        _portraitSelector.Update(gameTime);
        _genderSelector.Update(gameTime);
        _nameTextBox.Update(gameTime);
        _create.Update(gameTime);
        _back.Update(gameTime);

        base.Update(gameTime);
    }

    public override void Draw(GameTime gameTime)
    {
        GameRef.SpriteBatch.Begin();

        GameRef.SpriteBatch.Draw(
            _background,
            _destination.Scale(Settings.Scale),
            Color.White);

        if (_genderSelector.SelectedIndex == 0)
        {
            GameRef.SpriteBatch.Draw(
                _femalePortraits[_portraitSelector.SelectedItem],
                _portraitDestination.Scale(Settings.Scale),
                Color.White);
        }
        else
        {
            GameRef.SpriteBatch.Draw(
                _malePortraits[_portraitSelector.SelectedItem],
                _portraitDestination.Scale(Settings.Scale),
                Color.White);
        }

        _genderSelector.Draw(GameRef.SpriteBatch);
        _portraitSelector.Draw(GameRef.SpriteBatch);
```

```
            _nameTextBox.Draw(GameRef.SpriteBatch);
            _create.Draw(GameRef.SpriteBatch);
            _back.Draw(GameRef.SpriteBatch);

            base.Draw(gameTime);

            GameRef.SpriteBatch.End();
        }
    }
}
```

There is a Texture2D field for the background image. There is a destination rectangle for the background image called _destination. There is another destination rectangle that describes where the portrait is drawn. There is a left right selector for choosing the portrait and a left right selector for selecting the gender. There is a text box for the name. There are two dictionaries that hold the female and male portraits. Finally there are two buttons. One to create the character and one to go back to the previous menu. The class does not require any properties to expose values to other classes. In the constructor it initializes the dictionaries.

In the LoadContent method I load the background texture. I create the gender selector passing in the left and right chevrons from the GUI pack. I position it relative to the background image. I then wire the event handler for the SelectionChanged event. I then set the items to Female and Male with a width of 270 pixels. Next I create the portrait selector padding in the chevrons again and positioning it. I then wire the event handler for the SelectionChanged event. I create the text box next passing in the background texture. I will get to content later. I position it, give it focus, enable it and set the color to white. The next thing I do is load the portraits and add them to the portrait dictionaries. For the tutorial I just load the sprite sheets. You would want to have actual portraits for the player to choose from. The next thing that I do is create the create button. I pass in the button texture, set the text to Create and position it at the bottom left of the screen. The next step is to wire the event handler. I do something similar for the back button.

In the Click event handler for the back button I pop the state off of the stack. In the Click event handler for the create button I stop the currently playing song. I then create a new play passing in the game reference, the text in the text box, if the selected index of the gender selector is 0 or not because true is female and 1 is male. I call SetUpNewGame on the game play state to start a new game. I pop the state off the stack and push the game play state on top.

In the event handler for the SelectionChanged event I check if the selected index is 0. If it is I call SetItems of the portrait left right selector passing in the keys collection of the female portraits dictionary as an array. If it is not I call SetItems of the portrait left right selector passing in the keys collection of the male portraits dictionary as an array.

The Update method calls the update methods of the controls. The draw method draws the background image scaling it to fill the screen. If the gender selector has a SelectedIndex of 0 I draw the selected female portrait, scaling it using the Scale property of the settings class. Otherwise I draw the male selector. Next I call the Draw method of the controls.

I update the constructor of the Player class because the key I used in the dictionaries does not include the gender of the sprites. Update the constructor to the following.

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

    if (gender)
        this.textureName = textureName + "_f";
    else
        this.textureName = textureName + "_m";

    sprite = new AnimatedSprite(
        game.Content.Load<Texture2D>(@"CharacterSprites\" + this.textureName),
        Game1.Animations)
    {
        CurrentAnimation = AnimationKey.WalkDown
    };
    Gold = 1000;
    backpack = new Backpack();
}
```

We need to add a field and property to the Game1 class for the new state and create an instance in the constructor. Add the following field and property to the Game1 class and change the constructor to the following.

```
private readonly NewGameState newGameState;

public NewGameState NewGameState => newGameState;

public Game1()
{
    Settings.Load();

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

    graphics.ApplyChanges();


    foreach (var v in graphics.GraphicsDevice.Adapter.SupportedDisplayModes)
    {
        Point p = new Point(v.Width, v.Height);
        string s = v.Width + " by " + v.Height + " pixels";

        if (v.Width >= 1280 && v.Height >= 720)
        {
            Resolutions.Add(s, p);
        }
    }

    Content.RootDirectory = "Content";

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

```
            Components.Add(new Muse(this));
            Muse.SetEffectVolume(Settings.SoundVolume);
            Muse.SetSongVolume(Settings.MusicVolume);

            gamePlayState = new GamePlayState(this);
            conversationState = new ConversationState(this);
            levelUpState = new LevelUpState(this);
            damageState = new DamageState(this);
            battleOverState = new BattleOverState(this);
            battleState = new BattleState(this);
            actionSelectionState = new ActionSelectionState(this);
            shadowMonsterSelectionState = new ShadowMonsterSelectionState(this);
            startBattleState = new StartBattleState(this);
            shopState = new ShopState(this);
            itemSelectionState = new ItemSelectionState(this);
            useItemState = new UseItemState(this);
            mainMenuState = new MainMenuState(this);
            optionState = new OptionState(this);
            newGameState = new NewGameState(this);

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

The last thing to do is trigger the new state. That is done in the MainMenuState. In the Update method you push the new state on the stack if the first menu item is selected. Change the Update method to the follow.

```
        public override void Update(GameTime gameTime)
        {
            frameCount++;

            if (menu.SelectedIndex == -1)
            {
                menu.SetMenuItems(new[] { "New Game", "Continue", "Options", "Exit" });
            }

            if (Xin.CheckKeyReleased(Keys.Escape))
            {
                GameRef.Exit();
            }

            if (Xin.CheckKeyReleased(Keys.Enter) ||
                Xin.CheckKeyReleased(Keys.Space)
                || (menu.MouseOver && Xin.CheckMouseReleased(MouseButtons.Left)) && frameCount
> 5)
            {
                Muse.PlaySoundEffect("menu_click");

                switch (menu.SelectedIndex)
                {
                    case 0:
                        manager.PopState();
                        manager.PushState(GameRef.NewGameState);
                        break;
                    case 1:
                        string path = Environment.GetFolderPath(
```

```
                        Environment.SpecialFolder.ApplicationData);
                    path += "\\ShadowMonsters\\ShadowMonsters.sav";

                    if (!File.Exists(path))
                    {
                        return;
                    }

                    Muse.StopSong();
                    MoveManager.FillMoves();
                    ShadowMonsterManager.FromFile(@".\Content\ShadowMonsters.txt",
content);
                    manager.PopState();
                    manager.PushState(GameRef.GamePlayState);
                    GameRef.GamePlayState.LoadGame();
                    GameRef.GamePlayState.ResetEngine();
                    break;
                case 2:
                    manager.PushState(GameRef.OptionState);
                    break;
                case 3:
                    GameRef.Exit();
                    break;
            }
            frameCount = 0;
        }
        base.Update(gameTime);
    }
```

Now you just need to content that I used. I would recommend downloading the GitHub repository from here https://github.com/Synammon/Shadow-Monsters. I removed the song from the repository because there is a licensing issue so there will be build errors if you try to build. Once you've downloaded the source open the MonoGame Pipeline Tool. Right click the Content node, select Add and then Existing Item. Browse to the content from the source code and add the chargen-back.png file. Right click the GUI folder, select Add and then Existing Item. Navigate to the GUI folder in the source code and add the textbox.png file.

If you build and run now we have a character generator when you select new game from the main menu. I am going to wrap this tutorial up here as we've covered a lot and I don't want to start a new topic at this point. 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!