A Summoner's Tale
Chapter A
All About Character

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

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

I know you are like: What the hell, girl? What's with Chapter A? Okay, I'm a bit of a nerd, a lot like a nerd, and I am numbering this series in hexadecimal. I mean, it is good to know it if you're a developer. It does have its uses. Anyway, my lame sense of humour is out of the way. This tutorial will be on the game. We are going to create a character generator. I wasn't initially going to do one and force the player to play a single character. But, after some introspection, I decided to do one. There are actually three things that I'm going to cover in this tutorial: creating a menu, creating a character generator, and creating the player class.

So, let's get started! Before I get too far, I want to add a class for the player framework. It won't be anything super complex at this point, just a collection of related data. Right-click the SummonersTale folder, select add and then Class. Names this new class Player. The code for that follows next.

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

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

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

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

```
        }

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

        public override void Update(GameTime gameTime)
        {
            if (Sprite != null)
            {
                Sprite.Update(gameTime);
            }

            base.Update(gameTime);
        }

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

This class inherits from DrawableGameComponent, so it will have an Update and Draw method. It can also be added to the list of game components and have its Update and Draw methods called automatically if we want to. I don't see being able to do that, however. So, it is just an option that we have.

The class has one field and three properties. The field is a SpriteBacth that will be used in rendering. The properties at the name, gender, and sprite for the player. They need no explanation from the names.

There is a single constructor for the class. It takes the required Game parameter. It also takes the name, gender, and sprite. The gender is a bool where true is female, and male is false. Next, I retrieve the SpriteBatch from the list of services. I then set the properties to the parameters passed in.

I then implemented the Update method. All it does is check to ensure that there is a sprite to update. If there is a sprite to update, its Update method is called. Similarly, in the Draw method, there is a check that the Sprite property is not null. If so, it calls Begin on the SpriteBatch, the Draw method of the sprite, and then the End method of the SpriteBatch object.

In order to handle multiple resolutions class I needed to make a few changes to the Settings class. I added two constants, one for the base resolution width and one for the height. The other change was to create a scale vector that will be used to transform from the base resolution to the target resolution. Update the Settings class to the following.

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

namespace SummonersTale
{
    public class Settings
    {
        public const int BaseWidth = 1280;
        public const int BaseHeight = 720;

        private static float musicVolume = 0.5f;
        private static float soundVolume = 0.5f;
        private static Point resolution = new(BaseWidth, BaseHeight);

        public static Vector2 Scale
        {
            get { return new((float)resolution.X / BaseWidth, (float)resolution.Y /
BaseHeight); }
        }

        public static float MusicVolume
        {
            get
            {
                return musicVolume;
            }

            set
            {
                musicVolume = MathHelper.Clamp(value, 0, 1f);
            }
        }

        public static float SoundVolume
        {
            get
            {
                return soundVolume;
            }

            set
            {
                soundVolume = MathHelper.Clamp(value, 0, 1f);
            }
        }

        public static Point Resolution
        {
            get { return resolution; }
            set { resolution = value; }
        }

        public static void Save()
```

```
        {
            string path =
Environment.GetFolderPath(Environment.SpecialFolder.ApplicationData);
            path += "/ASummonersTale/";

            if (!Directory.Exists(path))
            {
                Directory.CreateDirectory(path);
            }

            path += "settings.bin";

            using FileStream stream = new(path, FileMode.Create, FileAccess.Write);
            using BinaryWriter writer = new(stream);

            writer.Write(soundVolume);
            writer.Write(musicVolume);
            writer.Write(resolution.X);
            writer.Write(resolution.Y);
            writer.Close();
            stream.Close();
        }

        public static void Load()
        {
            string path =
Environment.GetFolderPath(Environment.SpecialFolder.ApplicationData);
            path += @"/ASummonersTale/settings.bin";

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

            using FileStream stream = new(path, FileMode.Open, FileAccess.Read);
            using BinaryReader reader = new(stream);

            soundVolume = reader.ReadSingle();
            musicVolume = reader.ReadSingle();
            resolution = new(reader.ReadInt32(), reader.ReadInt32());
            reader.Close();
            stream.Close();
        }
    }
}
```

The only thing that you need to remember is that when dividing the target resolution by the base resolution is that you need to cast the target resolution to a float to get floating point numbers, or it won't work right.

Talking about scale, I added an extension method that would scale a rectangle by a scale vector. Right click the SummonersTale project, select Add and then class. Name this new class ExtentionMethods. Here is the code for that class so far.

```
using System;
```

```
using System.Collections.Generic;
using Microsoft.Xna.Framework;
using System.Text;

namespace SummonersTale
{
    public static class ExtentionMethods
    {
        public static Rectangle Scale(this Rectangle r, Vector2 scale)
        {
            Rectangle scaled = new(
                (int)(r.X * scale.X),
                (int)(r.Y * scale.Y),
                (int)(r.Width * scale.X),
                (int)(r.Height * scale.Y));

            return scaled;
        }
    }
}
```

What this does, is given a rectangle, multiply the X value and width by the X scale value and the Y value and height by the y scale value. Effectively, it translates X and Y values to the new position and grows/shrinks the area it covers.

I need to make a change to the Button and RightLeftSelector class. What I need to do is scale the buttons so that they will field the clicks successfully. First, replace the HandleInput method of the Button class as follows.

```
public override void HandleInput()
{
    MouseState mouse = Mouse.GetState();
    Point position = new(mouse.X, mouse.Y);
    Rectangle destination = new(
        (int)(Position.X + Offset.X),
        (int)(Position.Y + Offset.Y),
        _background.Width,
        _background.Height);

    if ((Role == ButtonRole.Accept && Xin.WasKeyReleased(Keys.Enter)) ||
        (Role == ButtonRole.Accept && Xin.WasKeyReleased(Keys.Space)))
    {
        OnClick();
        return;
    }

    if (Role == ButtonRole.Cancel && Xin.WasKeyReleased(Keys.Escape))
    {
        OnClick();
        return;
    }

    if (Xin.WasMouseReleased(MouseButton.Left) && _frames >= 5)
    {
        Rectangle r = destination.Scale(Settings.Scale);
```

```
            if (r.Contains(position))
                OnClick();
        }
    }
```

All the new code does is create a scaled rectangle using the Scale method that was just implemented, passing in the Scale property of the Settings class. I do something similar in the HandleMouseInput method of the RightLeftSelector class. You can replace that code with the following.

```
private void HandleMouseInput()
{
    if (Xin.WasMouseReleased(MouseButton.Left))
    {
        Point mouse = Xin.MouseAsPoint;

        if (_leftSide.Scale(Settings.Scale).Contains(mouse))
        {
            _selectedItem--;
            if (_selectedItem < 0)
                _selectedItem = this.Items.Count - 1;
            OnSelectionChanged();
        }

        if (_rightSide.Scale(Settings.Scale).Contains(mouse))
        {
            _selectedItem++;
            if (_selectedItem >= _items.Count)
                _selectedItem = 0;
            OnSelectionChanged();
        }
    }
}
```

Before I go any further, I need to make a few changes to the BaseGameState. Since it is a small class, I will give you the code for the entire thing. Replace the existing BaseGameState class with the following.

```
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;
using SummonersTale.Forms;
using System;
using System.Collections.Generic;
using System.Text;

namespace SummonersTale.StateManagement
{
    public class BaseGameState : GameState
    {
        #region Field Region

        protected const int TargetWidth = 1280;
        protected const int TargetHeight = 720;

        protected readonly static Random random = new();
        protected readonly SpriteBatch spriteBatch;
```

```
        protected readonly RenderTarget2D renderTarget;

        #endregion

        #region Proptery Region

        protected ControlManager Controls { get; set; }
        protected static Player Player { get; set; }

        public SpriteBatch SpriteBatch
        {
            get { return spriteBatch; }
        }

        #endregion

        #region Constructor Region

        public BaseGameState(Game game)
            : base(game)
        {
            spriteBatch =
(SpriteBatch)Game.Services.GetService(typeof(SpriteBatch));
            renderTarget = new(GraphicsDevice, TargetWidth, TargetHeight);
        }

        #endregion

        protected override void LoadContent()
        {
            Controls = new(content.Load<SpriteFont>(@"Fonts/MainFont"), 100);

            base.LoadContent();
        }
    }
}
```

All that has changed is that I added two new properties. One is for the player object, and the other is a control manager. The player object is static so that it will be shared by all classes inheriting from BaseGameState. I also instantiate the control manager in the LoadContent method because it requires a sprite font. It is important to remember that you need to call base.LoadContent before using it in any child method that makes use of it.

Now, I'm going to add a menu state where the player chooses their options. Right-click the StateManagement folder in the SummonersTale project, select Add and then Class. Name this new class MainMenuState. Here is the code for that class.

```
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework.Media;
using SummonersTale.StateManagement;
using SummonersTale;
using System;
```

```
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Reflection.Metadata;
using System.Text;
using System.Threading.Tasks;
using SummonersTale.Forms;

namespace SummonersTale.StateManagement
{
    public interface IMainMenuState
    {
        GameState GameState { get; }
    }

    public class MainMenuState : BaseGameState, IMainMenuState
    {
        #region Field Region

        private double timer;

        private Button newGameButton;
        private Button oldGameButton;
        private Button optionsButton;
        private Button creditsButton;
        private Button leaveButton;
        private RenderTarget2D renderTarget2D;

        public GameState GameState => this;

        #endregion

        #region Property Region
        #endregion

        #region Constructor Region

        public MainMenuState(Game game) : base(game)
        {
            Game.Services.AddService<IMainMenuState>(this);
        }

        #endregion

        #region Method Region

        protected override void LoadContent()
        {
            base.LoadContent();
            renderTarget2D = new(GraphicsDevice, TargetWidth, TargetHeight);
            Texture2D texture = new(GraphicsDevice, 400, 50);

            Color[] buffer = new Color[400 * 50];

            for (int i = 0; i < 400 * 50; i++)
                buffer[i] = Color.Black;

            texture.SetData(buffer);
```

```
            newGameButton = new(content.Load<Texture2D>(@"GUI/g9202"),
ButtonRole.Menu)
            {
                Text = "New Game",
                Position = new((TargetWidth - 200) / 2, 100),
                Color = Color.White,
                Size = new(200, 50)
            };

            oldGameButton = new(content.Load<Texture2D>(@"GUI/g9202"),
ButtonRole.Menu)
            {
                Text = "Continue",
                Position = new((TargetWidth - 200) / 2, 200),
                Color = Color.White,
                Size = new(200, 50)
            };

            optionsButton = new(content.Load<Texture2D>(@"GUI/g9202"),
ButtonRole.Menu)
            {
                Text = "Options",
                Position = new((TargetWidth - 200) / 2, 300),
                Color = Color.White,
                Size = new(200, 50)
            };

            creditsButton = new(content.Load<Texture2D>(@"GUI/g9202"),
ButtonRole.Menu)
            {
                Text = "Credits",
                Position = new((TargetWidth - 200) / 2, 400),
                Color = Color.White,
                Size = new(200, 50)
            };

            leaveButton = new(content.Load<Texture2D>(@"GUI/g9202"),
ButtonRole.Menu)
            {
                Text = "Leave",
                Position = new((TargetWidth - 200) / 2, 500),
                Color = Color.White,
                Size = new(200, 50)
            };

            newGameButton.Click += NewGameButton_Click;
            oldGameButton.Click += OldGameButton_Click;
            optionsButton.Click += OptionsButton_Click;
            creditsButton.Click += CreditsButton_Click;
            leaveButton.Click += LeaveButton_Click;

            Controls.Add(newGameButton);
            Controls.Add(oldGameButton);
            Controls.Add(optionsButton);
            Controls.Add(creditsButton);
            Controls.Add(leaveButton);
        }
```

```
        private void LeaveButton_Click(object sender, EventArgs e)
        {
            Game.Exit();
        }

        private void CreditsButton_Click(object sender, EventArgs e)
        {
            throw new NotImplementedException();
        }

        private void OptionsButton_Click(object sender, EventArgs e)
        {
            throw new NotImplementedException();
        }

        private void OldGameButton_Click(object sender, EventArgs e)
        {
            throw new NotImplementedException();
        }

        private void NewGameButton_Click(object sender, EventArgs e)
        {
            manager.PushState(Game.Services.GetService<INewGameState>().GameState);
        }

        public override void Update(GameTime gameTime)
        {
            timer += gameTime.ElapsedGameTime.TotalSeconds;

            Controls.Update(gameTime);

            if (timer < 0.25) return;

            if (Xin.WasKeyReleased(Keys.Escape))
            {
                Game.Exit();
            }


            base.Update(gameTime);
        }

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

            GraphicsDevice.SetRenderTarget(renderTarget2D);
            GraphicsDevice.Clear(Color.CornflowerBlue);

            SpriteBatch.Begin(SpriteSortMode.Deferred, BlendState.AlphaBlend,
SamplerState.PointClamp);

            Controls.Draw(SpriteBatch);

            SpriteBatch.End();

            GraphicsDevice.SetRenderTarget(null);
```

```
            SpriteBatch.Begin();

            SpriteBatch.Draw(
                renderTarget2D,
                new Rectangle(0, 0, Settings.Resolution.X, Settings.Resolution.Y),
                Color.White);

            SpriteBatch.End();
        }

        public override void Show()
        {
            timer = 0;

            base.Show();
        }

        #endregion
    }
}
```

There is an interface, like in the other game states. It has a single property that requires returning the current instance. The class inherits from BaseGameState and implements the interface. There is a timer field that makes sure the game isn't closed prematurely. There are five buttons for the different options in the menu. There is one for starting a new game, one for loading a game, going to the options, viewing the credits and exiting the games. There is the read-only property that returns the current instance.

All the constructor does is add this instance as a service to the list of services. I set it to be the interface instead of the class. In the LoadContent method, I create a black texture that I currently don't use it but will be used in the future, though, so I included it now. Next, I start creating the buttons. I pass in a texture from the GUI buttons that I added in the last tutorial. I then set their text, position, size and colour. The size might be a little big, but they work for now. After creating the buttons, I wire the event handlers for their click events. Then, I add them to the control manager. I currently only implement the click event handler for the leave button and new game button. In the event handler of the new game button, I push the instance of the NewGameState, which I will add shortly.

The Update method increases the timer field by the elapsed time since the last update. It then calls the Update method of the control manager to update the buttons and call their HandleInput methods. If the elapsed time is less than a quarter of a second, I exit the method. Here is why I added the time. I check to see if the escape key was pressed, so the player has another option for leaving the game. What I've found is that MonoGame can stutter a little be. A key press or release can carry over a frame or two. So, when I'm handling critical input, I put a short timer on it.

The Draw method is written to handle rendering to multiple resolutions without fuss. I first set the render target to render to our base resolution. I clear the background to cornflower blue. Begin is called per usual. I call the Draw method of the control manager to render the controls. After calling End, I set

the render target to null, resetting rendering back to the window. I call Begin with no parameters. I render to a rectangle equal to the size of the window, scaling the output. Finally, I call End.

There was a dependency on a few states in the main menu state. In this tutorial, I will address the new game. The others I will handle other states. Right-click the StateManagement folder in the SummonersTale project, select Add and then Class. Name this new class NewGameState. This is the code for that state.

```
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework;
using SummonersTale.Forms;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using ShadowMonsters.Controls;
using System.Reflection.Metadata.Ecma335;
using SummonersTale.SpriteClasses;

namespace SummonersTale.StateManagement
{
    public interface INewGameState
    {
        GameState GameState { get; }
    }

    public class NewGameState : BaseGameState, INewGameState
    {
        private Rectangle _portraitDestination = new(599, 57, 633, 617);
        private RightLeftSelector _portraitSelector;
        private RightLeftSelector _genderSelector;
        private Textbox _nameTextBox;
        private readonly Dictionary<string, Texture2D> _femalePortraits;
        private readonly Dictionary<string, Texture2D> _malePortraits;
        private Button _create;
        private Button _back;
        private readonly Dictionary<string, Animation> animations = new();
        private AnimatedSprite _sprite;
        private RenderTarget2D renderTarget2D;

        public GameState GameState => this;

        public NewGameState(Game game) : base(game)
        {
            Game.Services.AddService<INewGameState>(this);
            _femalePortraits = new();
            _malePortraits = new();
        }

        public override void Initialize()
        {
            animations.Clear();
```

```
            Animation animation = new(3, 32, 32, 0, 0) { CurrentFrame = 0,
FramesPerSecond = 5 };
            animations.Add("walkdown", animation);

            animation = new(3, 32, 32, 0, 32) { CurrentFrame = 0, FramesPerSecond =
5 };
            animations.Add("walkleft", animation);

            animation = new(3, 32, 32, 0, 64) { CurrentFrame = 0, FramesPerSecond =
5 };
            animations.Add("walkright", animation);

            animation = new(3, 32, 32, 0, 96) { CurrentFrame = 0, FramesPerSecond =
5 };
            animations.Add("walkup", animation);

            base.Initialize();
        }

        protected override void LoadContent()
        {
            renderTarget2D = new(GraphicsDevice, TargetWidth, TargetHeight);

            Controls = new(content.Load<SpriteFont>(@"Fonts/MainFont"), 100);
            _genderSelector = new(
                content.Load<Texture2D>(@"GUI\g22987"),
                content.Load<Texture2D>(@"GUI\g21245"))
            {
                Position = new Vector2(207 - 70, 298)
            };
            _genderSelector.SelectionChanged += GenderSelector_SelectionChanged;
            _genderSelector.SetItems(new[] { "Female", "Male" }, 270);

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

            _portraitSelector.SelectionChanged += PortraitSelector_SelectionChanged;

            Color[] colourData = new Color[2 * 25];

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

            Texture2D caret = new(GraphicsDevice, 2, 25);
            caret.SetData(colourData);

            _nameTextBox = new(
                content.Load<Texture2D>(@"GUI/textbox"),
                caret)
            {
                Position = new(207, 138),
                HasFocus = true,
```

```
        Enabled = true,
        Color = Color.White,
        Text = "Bethany"
    };

    _femalePortraits.Add(
        "Female 0",
        content.Load<Texture2D>(@"CharacterSprites/femalefighter"));
    _femalePortraits.Add(
        "Female 1",
        content.Load<Texture2D>(@"CharacterSprites/femalepriest"));
    _femalePortraits.Add(
        "Female 2",
        content.Load<Texture2D>(@"CharacterSprites/femalerogue"));
    _femalePortraits.Add(
        "Female 3",
        content.Load<Texture2D>(@"CharacterSprites/femalewizard"));

    _malePortraits.Add(
        "Male 0",
        content.Load<Texture2D>(@"CharacterSprites/malefighter"));
    _malePortraits.Add(
        "Male 1",
        content.Load<Texture2D>(@"CharacterSprites/malepriest"));
    _malePortraits.Add(
        "Male 2",
        content.Load<Texture2D>(@"CharacterSprites/malerogue"));
    _malePortraits.Add(
        "Male 3",
        content.Load<Texture2D>(@"CharacterSprites/malewizard"));


    _portraitSelector.SetItems(_femalePortraits.Keys.ToArray(), 270);
    _sprite = new(_femalePortraits["Female 0"], animations)
    {
        CurrentAnimation = "walkdown",
        IsAnimating = true
    };

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

    _create.Click += Create_Click;

    _back = new(
        content.Load<Texture2D>(@"GUI\g9202"),
        ButtonRole.Cancel)
    {
        Text = "Back",
        Position = new(350, 640),
        Color = Color.White
    };
```

```
            _back.Click += Back_Click;

            Controls.Add(_nameTextBox);
            Controls.Add(_genderSelector);
            Controls.Add(_portraitSelector);
            Controls.Add(_create);
            Controls.Add(_back);
        }

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

        private void Create_Click(object sender, EventArgs e)
        {
            Player = new(
                Game,
                _nameTextBox.Text,
                _genderSelector.SelectedIndex == 0,
                _sprite);

            IGamePlayState gamePlayState =
Game.Services.GetService<IGamePlayState>();

            manager.PopState();
            manager.PushState(gamePlayState.GameState);

            gamePlayState.GameState.Enabled = true;
            gamePlayState.NewGame();
        }

        private void GenderSelector_SelectionChanged(object sender, EventArgs e)
        {
            if (_genderSelector.SelectedIndex == 0)
            {
                _portraitSelector.SetItems(_femalePortraits.Keys.ToArray(), 270);
                _nameTextBox.Text = "Bethany";
                _sprite = new(
                    _femalePortraits[_portraitSelector.SelectedItem],
                    animations)
                {
                    CurrentAnimation = "walkdown",
                    IsAnimating = true
                };
            }
            else
            {
                _portraitSelector.SetItems(_malePortraits.Keys.ToArray(), 270);
                _nameTextBox.Text = "Anthony";
                _sprite = new(
                    _malePortraits[_portraitSelector.SelectedItem],
                    animations)
                {
                    CurrentAnimation = "walkdown",
                    IsAnimating = true
                };
            }
```

```
        }

        private void PortraitSelector_SelectionChanged(object sender, EventArgs e)
        {
            if (_genderSelector.SelectedIndex == 0)
            {
                _sprite = new(
                    _femalePortraits[_portraitSelector.SelectedItem],
                    animations)
                {
                    CurrentAnimation = "walkdown",
                    IsAnimating = true
                };
            }
            else
            {
                _sprite = new(
                    _malePortraits[_portraitSelector.SelectedItem],
                    animations)
                {
                    CurrentAnimation = "walkdown",
                    IsAnimating = true
                };
            }
        }

        public override void Update(GameTime gameTime)
        {
            Controls.Update(gameTime);
            _sprite.Update(gameTime);

            base.Update(gameTime);
        }

        public override void Draw(GameTime gameTime)
        {
            GraphicsDevice.SetRenderTarget(renderTarget2D);
            GraphicsDevice.Clear(Color.CornflowerBlue);

            SpriteBatch.Begin(SpriteSortMode.Deferred, BlendState.AlphaBlend,
SamplerState.PointClamp);

            if (_genderSelector.SelectedIndex == 0)
            {
                SpriteBatch.Draw(
                    _femalePortraits[_portraitSelector.SelectedItem],
                    _portraitDestination,
                    new Rectangle(0, 0, 32, 32),
                    Color.White);
            }
            else
            {
                SpriteBatch.Draw(
                    _malePortraits[_portraitSelector.SelectedItem],
                    _portraitDestination,
                    new Rectangle(0, 0, 32, 32),
                    Color.White);
            }
```

```
            Controls.Draw(SpriteBatch);

            _sprite.Draw(SpriteBatch);

            base.Draw(gameTime);

            SpriteBatch.End();

            GraphicsDevice.SetRenderTarget(null);

            SpriteBatch.Begin();

            SpriteBatch.Draw(
                renderTarget2D,
                new Rectangle(0, 0, Settings.Resolution.X, Settings.Resolution.Y),
                Color.White);

            SpriteBatch.End();

        }
    }
}
```

Sorry for the wall of code. I didn't see a better option to give it to you. Like all of the other game states, there is an interface that has a single property. The interface must return the current instance of the class. I add one for each state rather than using a single one because of the way that I add and retrieve the instances.

So, the class inherits from BaseGameState, because we want to be able to push and pop it. It then implements the interface. There is a rectangle that describes where the portrait would go. It could have just as easily been a picture box. I don't have portraits for the character sprites. So, I just used the sprite, which is at about twenty times magnification. There are right and left selectors to choose the character's gender and sprite. There is a text box for the character's name. There are dictionaries that hold the name and texture of a sprite. There are buttons to create the character and one to return to the main menu state. There is a field that holds the animations for the sprite and one for the sprite. There is just one property that is required by the interface.

There is only one constructor for the class, and that takes a single Game parameter, which is required because game states inherit from DrawableGameComponent. It registers itself as a service. It also initializes the portrait dictionaries. The Initialize method clears the items in the animations dictionary. Then, as we did in tutorial 08, it creates the animations and adds them to the dictionary.

In the LoadContent method, I create the controls for creating a character. The first control is the right and left control for the character's gender. I pass in textures from the GUI items that we added earlier. I set its position, then wire the event handler for the SeletionChanged event. I set the maximum width for rendering. I then do something similar for the portrait, or I should say sprite, selector.

Then, I create a texture for the caret. Following that, I create the text box for entering the name. Next, I create the dictionaries with string keys and Texture2Ds for the values. The keys will be the gender plus a digit. For zero, the texture is the fighter texture, one is the priest, two is the rogue, and finally, three is the wizard. I set the values of the portrait selector to the keys of the dictionary of female portraits. I create a sprite using the first key in the dictionary that is using the walk-down animation, and that is animating.

With the selectors created, I turn my attention to the buttons. They both use the same texture for the button. The create button is the role accept so that if the player hits the enter key, it will fire as if it was clicked. It has the text Create and the position toward the bottom left of the window. I also wire its event handler. The back button is similar. The big difference is that I set the button role to the cancel action. The controls are then added to the control manager.

In the Click event handler for the back button, I pop the create state off of the stack. That is what the intention is. It will do so if it is clicked but not if the escape key is pressed. In that instance, the game will exit completely. That is good enough for now.

In the Click event handler of create button, I create a new instance of the Play class. I pass in the Game property, the text from the text box, if the selected index of the gender selector is 0 or not, and the _sprite field. pop this state off the stack and push the gameplay state onto the stack. I then retrieve the gameplay state using the GetService method. I pop the current state off of the stack and push the instance of the GamePlayState. I set its Enabled property to true and call a new method of the GamePlayState that I will implement shortly.

In the SelectionChanged handler of the gender selector, I see if the selection is 0, which means female. If it is, I change the items of the portrait selector to the female ones. I change the name to Bethany, and the sprite to the selected sprite. I set the sprite's animation to walk down and IsAnimating to true. I do pretty much the same if it is a male character.

The SelectionChanged event handler of the portrait selector is much the same as in the SelectionChanged handler of the gender selector. I see if the gender selection is 0, which means female. If it is, I change the sprite based on the selected item. I set the sprite's animation to walk down and IsAnimating to true. I do pretty much the same if it is a male character.

In the Update method, I call the Update method of the control manager and sprite. I also call base.Update, which would call the Update method of any child components.

You should be familiar with the Draw method now. I set the render target. I clear it to cornflower blue. If the gender selector is 0, I draw the selected sprite's first source rectangle. Again, it is going to be highly distorted because, in my game, I have portraits of all of the actors. If it was not zero, I draw the male one. After drawing the portrait, I draw the control manager and sprite. After the call to base.Draw, which operates the same way as base.Update, just for drawing instead of updating. Then, I call the End

method to stop rendering. The next step is to set the render target back on the screen. Then, draw the render target.

I did make two tweaks to the GamePlayState. The first is that I had hard-coded the resolution. I switched that to using the Settings class. The other is the addition of the NewGame method. All that does, for now, is call LoadContent. Eventually, when we have actual content, it will do more. I added it now rather than later. Update the Draw method of the GamePlayState to the following and add the new method.

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

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

    _tileMap.Draw(gameTime, SpriteBatch, _camera, false);

    spriteBatch.Begin(
        SpriteSortMode.Deferred,
        BlendState.AlphaBlend,
        SamplerState.PointClamp,
        null,
        null,
        null,
        _camera.Transformation);

    sprite.Draw(SpriteBatch);
    SpriteBatch.End();

    GraphicsDevice.SetRenderTarget(null);

    SpriteBatch.Begin(SpriteSortMode.Immediate, BlendState.AlphaBlend,
SamplerState.LinearClamp);

    SpriteBatch.Draw(renderTarget, new Rectangle(new(0, 0), Settings.Resolution),
Color.White);

    SpriteBatch.End();
}

public void NewGame()
{
    LoadContent();
}
```

I also updated the TitleState. I made an update to the Draw method. What I did was introduce a local variable for where the text is being rendered. The reason why is that the text was pixelated, most likely due to rendering on a partial pixel. So, I introduced a local variable and used the NearestInt method of the helper class. Update that method to the following.

```
public override void Draw(GameTime gameTime)
{
```

```
    string message = "Game will begin in " + ((int)_timer).ToString() + " seconds.";
    Vector2 size = _spriteFont.MeasureString(message);

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

    Vector2 location = new((TargetWidth - size.X) / 2, TargetHeight -
(_spriteFont.LineSpacing * 5));
    SpriteBatch.Begin();

    SpriteBatch.DrawString(
        _spriteFont,
        message,
        Helper.NearestInt(location),
        Color.White);

    SpriteBatch.End();

    GraphicsDevice.SetRenderTarget(null);

    SpriteBatch.Begin();

    SpriteBatch.Draw(renderTarget, new Rectangle(new(0,0), new(1920,1080)),
Color.White);

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

You can build and run now. You will bounce to the main menu when the countdown on the title screen is finished. From there, you can go to the new game generator or exit the game. In the new game generator, you can select your character and then begin or go back.

I'm not going to dive any further into this tutorial. I think I've fed you enough for one day. So, that is going to be it for this tutorial. I covered a lot, and I don't want to overload you. I encourage you to keep visiting my blog for the latest news on my tutorials.

Good luck with your game programming adventures.
Cynthia