# Shadow Monsters – MonoGame Tutorial Series
# Chapter 18
# Handling Different Resolutions

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

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

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

The main focus of this tutorial is handling different screen resolutions. To do that I will be adding a couple new game states. I will be adding a menu state and an options state. What I will be doing is scale things based on the selected resolution. The base resolution is the resolution that we've been using all along, 1280 pixels by 720 pixels. It will also be the minimum supported resolution. We will support all resolutions above this.

The first that I'm going to do is add some new sprite font descriptions. Open the MonoGame Pipeline Tool by double clicking it in the Solution Explorer. Right click the Fonts folder in the MonoGame Pipeline Tool and select Add and then New Item. From the dialog that pops up select Sprint Font Description and name it testfont_medium. Repeat the process twice and name the fonts testfont_high and testfont_ultra. Close the MonoGame Pipeline Tool.

In the Solution Explorer right click the Fonts folder under the Content folder and select Add then Existing Item. Browse to the Fonts folder and select the new sprite font descriptions. Replace the contents of the files with the following.

testfont_medium.spritefont

```
<?xml version="1.0" encoding="utf-8"?>
<!--
This file contains an xml description of a font, and will be read by the XNA
Framework Content Pipeline. Follow the comments to customize the appearance
of the font in your game, and to change the characters which are available to draw
with.
-->
<XnaContent xmlns:Graphics="Microsoft.Xna.Framework.Content.Pipeline.Graphics">
  <Asset Type="Graphics:FontDescription">

    <!--
```

```
    Modify this string to change the font that will be imported.
    -->
    <FontName>Kootenay</FontName>

    <!--
    Size is a float value, measured in points. Modify this value to change
    the size of the font.
    -->
    <Size>24</Size>

    <!--
    Spacing is a float value, measured in pixels. Modify this value to change
    the amount of spacing in between characters.
    -->
    <Spacing>0</Spacing>

    <!--
    UseKerning controls the layout of the font. If this value is true, kerning information
    will be used when placing characters.
    -->
    <UseKerning>true</UseKerning>

    <!--
    Style controls the style of the font. Valid entries are "Regular", "Bold", "Italic",
    and "Bold, Italic", and are case sensitive.
    -->
    <Style>Regular</Style>

    <!--
    If you uncomment this line, the default character will be substituted if you draw
    or measure text that contains characters which were not included in the font.
    -->
    <!-- <DefaultCharacter>*</DefaultCharacter> -->

    <!--
    CharacterRegions control what letters are available in the font. Every
    character from Start to End will be built and made available for drawing. The
    default range is from 32, (ASCII space), to 126, ('~'), covering the basic Latin
    character set. The characters are ordered according to the Unicode standard.
    See the documentation for more information.
    -->
    <CharacterRegions>
      <CharacterRegion>
        <Start>&#32;</Start>
        <End>&#126;</End>
      </CharacterRegion>
    </CharacterRegions>
  </Asset>
</XnaContent>
```

## testfont_high.spritefont

```
<?xml version="1.0" encoding="utf-8"?>
<!--
This file contains an xml description of a font, and will be read by the XNA
Framework Content Pipeline. Follow the comments to customize the appearance
of the font in your game, and to change the characters which are available to draw
with.
-->
<XnaContent xmlns:Graphics="Microsoft.Xna.Framework.Content.Pipeline.Graphics">
  <Asset Type="Graphics:FontDescription">
```

```
    <!--
    Modify this string to change the font that will be imported.
    -->
    <FontName>Kootenay</FontName>

    <!--
    Size is a float value, measured in points. Modify this value to change
    the size of the font.
    -->
    <Size>30</Size>

    <!--
    Spacing is a float value, measured in pixels. Modify this value to change
    the amount of spacing in between characters.
    -->
    <Spacing>0</Spacing>

    <!--
    UseKerning controls the layout of the font. If this value is true, kerning information
    will be used when placing characters.
    -->
    <UseKerning>true</UseKerning>

    <!--
    Style controls the style of the font. Valid entries are "Regular", "Bold", "Italic",
    and "Bold, Italic", and are case sensitive.
    -->
    <Style>Regular</Style>

    <!--
    If you uncomment this line, the default character will be substituted if you draw
    or measure text that contains characters which were not included in the font.
    -->
    <!-- <DefaultCharacter>*</DefaultCharacter> -->

    <!--
    CharacterRegions control what letters are available in the font. Every
    character from Start to End will be built and made available for drawing. The
    default range is from 32, (ASCII space), to 126, ('~'), covering the basic Latin
    character set. The characters are ordered according to the Unicode standard.
    See the documentation for more information.
    -->
    <CharacterRegions>
      <CharacterRegion>
        <Start>&#32;</Start>
        <End>&#126;</End>
      </CharacterRegion>
    </CharacterRegions>
  </Asset>
</XnaContent>
```

## testfont_ultra.spritefont

```
<?xml version="1.0" encoding="utf-8"?>
<!--
This file contains an xml description of a font, and will be read by the XNA
Framework Content Pipeline. Follow the comments to customize the appearance
of the font in your game, and to change the characters which are available to draw
with.
-->
```

```
<XnaContent xmlns:Graphics="Microsoft.Xna.Framework.Content.Pipeline.Graphics">
  <Asset Type="Graphics:FontDescription">

    <!--
    Modify this string to change the font that will be imported.
    -->
    <FontName>Kootenay</FontName>

    <!--
    Size is a float value, measured in points. Modify this value to change
    the size of the font.
    -->
    <Size>36</Size>

    <!--
    Spacing is a float value, measured in pixels. Modify this value to change
    the amount of spacing in between characters.
    -->
    <Spacing>0</Spacing>

    <!--
    UseKerning controls the layout of the font. If this value is true, kerning information
    will be used when placing characters.
    -->
    <UseKerning>true</UseKerning>

    <!--
    Style controls the style of the font. Valid entries are "Regular", "Bold", "Italic",
    and "Bold, Italic", and are case sensitive.
    -->
    <Style>Regular</Style>

    <!--
    If you uncomment this line, the default character will be substituted if you draw
    or measure text that contains characters which were not included in the font.
    -->
    <!-- <DefaultCharacter>*</DefaultCharacter> -->

    <!--
    CharacterRegions control what letters are available in the font. Every
    character from Start to End will be built and made available for drawing. The
    default range is from 32, (ASCII space), to 126, ('~'), covering the basic Latin
    character set. The characters are ordered according to the Unicode standard.
    See the documentation for more information.
    -->
    <CharacterRegions>
      <CharacterRegion>
        <Start>&#32;</Start>
        <End>&#126;</End>
      </CharacterRegion>
    </CharacterRegions>
  </Asset>
</XnaContent>
```

I updated the FontManager to use these new fonts. Replace the code in the FontManager class with the following.

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

```
namespace ShadowMonsters
{
    public class FontManager : DrawableGameComponent
    {
        private readonly static Dictionary<string, SpriteFont> _fonts =
            new Dictionary<string, SpriteFont>();
        private static Game gameRef;

        public FontManager(Game game) : base(game)
        {
            gameRef = game;
        }

        protected override void LoadContent()
        {
            _fonts.Add("testfont", Game.Content.Load<SpriteFont>(@"Fonts\testfont"));
            _fonts.Add("testfont_medium",
Game.Content.Load<SpriteFont>(@"Fonts\testfont_medium"));
            _fonts.Add("testfont_high", Game.Content.Load<SpriteFont>(@"Fonts\testfont_high"));
            _fonts.Add("testfont_ultra",
Game.Content.Load<SpriteFont>(@"Fonts\testfont_ultra"));
        }

        public static SpriteFont GetFont(string name)
        {
            if (gameRef.GraphicsDevice.Viewport.Width >= 4000)
            {
                name += "_ultra";
            }
            else if (gameRef.GraphicsDevice.Viewport.Width >= 3000)
            {
                name += "_high";
            }
            else if (gameRef.GraphicsDevice.Viewport.Width >= 1920)
            {
                name += "_medium";
            }

            return _fonts[name];
        }

        public static bool ContainsFont(string name)
        {
            if (gameRef.GraphicsDevice.Viewport.Width >= 4000)
            {
                name += "ultra";
            }
            else if (gameRef.GraphicsDevice.Viewport.Width >= 3000)
            {
                name += "high";
            }
            else if (gameRef.GraphicsDevice.Viewport.Width >= 1920)
            {
                name += "_medium";
            }

            return _fonts.ContainsKey(name);
        }
    }
```

```
}
```

The LoadContent loads the new fonts as testfont_medium, test_fonthigh and testfont_ultra and adds them to the fonts collection. In the GetFont method I check to see if the width of the viewport is greater than 4000 pixels. If it is I append _ultra to the name passed. There is an else if to check if the width is greater than or equal to 3000 pixels. If that is true I append _high. Else if the width is greater than 1920 pixels I append _medium. I do the same thing in the ContainsFont method.

I add a field to the Game1 class that holds the resolutions that are supported by the user's graphics card and monitor. Add the following field to the Game1 class and update the constructor.

```
        public static readonly Dictionary<string, Point> Resolutions =
            new Dictionary<string, Point>();

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

            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);

            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);

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

```
        }
```

The  supported resolutions are loaded into a Dictionary<string, Point> where string will displayed when the user is choosing what resolution to use for the game and the Point is the width and height of the resolution. There is a property of the GraphicsDevice that describes the user's graphics adapter called Adapter. It has a property SupportedDisplayModes that is a collection of the display modes supported by the adapter. In a foreach loop I iterate over that collection. I create a Point using the Width and Height of the display mode. I also create a string using the Width and Height of the display mode. There is a check to see that the width is greater than or equal to 1280 and that the height is greater than or equal to 720. If they are the string and Point are added to the Dictionary.

I added a utility class that holds the settings for the game. It will hold things such as music volume, sound volume and screen resolution. It will be used through out the game when drawing things. Right click the ShadowMonsters project in the Solution Explorer, select Add and then class. Name this new class Settings. Here is the code.

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

namespace ShadowMonsters
{
    public class Settings
    {
        private static float musicVolume = 0.5f;
        private static float soundVolume = 0.5f;
        private static Point resolution = new Point(1280, 720);

        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 Point TileSize
{
    get
    {
        Point p = new Point(64, 64);

        if (resolution.X >= 4000)
        {
            p = new Point(160, 160);
        }
        else if (resolution.X >= 3000)
        {
            p = new Point(128, 128);
        }
        else if (resolution.X >= 1920)
        {
            p = new Point(96, 96);
        }

        return p;
    }
}

public static void Save()
{
    string path = Environment.SpecialFolder.LocalApplicationData.ToString();
    path += @"\Company Name\";

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

    path += "settings.bin";

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

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

public static void Load()
{
    string path = Environment.SpecialFolder.LocalApplicationData.ToString();
    path += @"\Company Name\settings.bin";

    if (!File.Exists(path))
```

```
            {
                Save();
            }

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

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

For now there are three fields. The first musicVolume is a float that should be between 0 and 1. The second soundVolume is a float that should be between 0 and 1. The last is resolution that is a Point that describes the resolution of the screen where the X coordinate is the width and the Y coordinate is the height. There are properties to expose the values of the fields. The set part of the float properties uses the MathHelper.Clamp method to lock the value of the field between 0 and 1. There is a forth property TileSize that returns the size of tiles on the map based on resolution. It will be used when changing resolution and when creating new games and loading new games.

There is  a Save method that saves the settings to the local application data folder. It creates a path to the folder. Here you would replace Company Name with your company's name. It checks to see if the path exists. If it does not it creates it. The name of the settings file is then appended to the path. A FileStream is then created using the path. A BinaryWriter is created next using the stream. It writes the soundVolume, musicVolume, X property of resolution then Y property of resolution.

There is also a Load method that loads the settings back in. It creates the path to the settings file. If it does not exist it calls the Save method to save the settings. It then creates a FileStream for reading using the path. Next it creates a BinaryReader. It reads in the sound volume and music volume. A new Point is created reading in two integers.

I implemented the Settings in the Initialize method of the Game1 class. I call Settings.Load to load the user's settings. I then set the width and height of the window and call the ApplyChanges method to update the window. Update the Initialize method of the Game1 class to the following code.

```
        protected override void Initialize()
        {
            // TODO: Add your initialization logic here

            Settings.Load();

            graphics.PreferredBackBufferWidth = Settings.Resolution.X;
            graphics.PreferredBackBufferHeight = Settings.Resolution.Y;
            graphics.ApplyChanges();

            BuildAnimations();
```

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


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

I updated GameScene.cs's Draw method to use the settings rather than hard coded values. Update the Draw method of GameScene to the following.

```
    public void Draw(SpriteBatch spriteBatch, Texture2D background)
    {
        Color myColor;

        if (textPosition == Vector2.Zero)
        {
            SetText(text);
        }

        if (background != null)
        {
            spriteBatch.Draw(background, Vector2.Zero, Color.White);
        }

        spriteBatch.DrawString(FontManager.GetFont("testfont"),
            text,
            textPosition,
            Color.White);

        Vector2 position = menuPosition;

        Rectangle optionRect = new Rectangle(
            0,
            (int)position.Y,
            Settings.Resolution.X,
            FontManager.GetFont("testfont").LineSpacing);

        isMouseOver = false;

        for (int i = 0; i < options.Count; i++)
        {
            if (optionRect.Contains(Xin.MouseState.Position))
            {
                selectedIndex = i;
                isMouseOver = true;
            }

            if (i == SelectedIndex)
            {
                myColor = HighLightColor;
            }
            else
            {
                myColor = NormalColor;
            }

            spriteBatch.DrawString(FontManager.GetFont("testfont"),
                options[i].OptionText,
```

```
                    position,
                    myColor);

                position.Y += FontManager.GetFont("testfont").LineSpacing + 5;
                optionRect.Y += FontManager.GetFont("testfont").LineSpacing + 5;
            }
        }
```

For the main menu I created a reusable menu component. It displays text over a texture, like a button, for each menu item. Right click the ShadowMonsters project in the Solution Explorer, select Add and then New Folder. Name this new folder Components. Right click the Components folder, select Add and then Class. Name this new class MenuComponent. Here is the code for the class.

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

namespace ShadowMonsters.Components
{
    public class MenuComponent : DrawableGameComponent
    {
        #region Fields

        Game1 gameRef;
        readonly List<string> menuItems = new List<string>();
        int selectedIndex = -1;
        bool mouseOver;

        int width;
        int height;

        Color normalColor = Color.White;
        Color hiliteColor = Color.Red;
        readonly Texture2D texture;

        Vector2 position;

        #endregion Fields

        #region Properties

        public Vector2 Postion
        {
            get { return position; }
            set { position = value; }
        }

        public int Width
        {
            get { return width; }
```

```
}

public int Height
{
    get { return height; }
}

public int SelectedIndex
{
    get { return selectedIndex; }
    set
    {
        selectedIndex = (int)MathHelper.Clamp(
                value,
                0,
                menuItems.Count - 1);
    }
}

public Color NormalColor
{
    get { return normalColor; }
    set { normalColor = value; }
}

public Color HiliteColor
{
    get { return hiliteColor; }
    set { hiliteColor = value; }
}

public bool MouseOver
{
    get { return mouseOver; }
}

#endregion Properties

#region Constructors

public MenuComponent(Game game, Texture2D texture) : base(game)
{
    this.gameRef = (Game1)game;
    this.mouseOver = false;
    this.texture = texture;
}

public MenuComponent(Game game, Texture2D texture, string[] menuItems)
    : this(game, texture)
{
    selectedIndex = 0;

    foreach (string s in menuItems)
    {
        this.menuItems.Add(s);
    }

    MeassureMenu();
}
```

```
        #endregion Constructors

        #region Methods

        public void SetMenuItems(string[] items)
        {
            menuItems.Clear();
            menuItems.AddRange(items);
            MeassureMenu();

            selectedIndex = 0;
        }

        private void MeassureMenu()
        {
            width = texture.Width;
            height = 0;

            foreach (string s in menuItems)
            {
                Vector2 size = FontManager.GetFont("testfont").MeasureString(s);

                if (size.X > width)
                    width = (int)size.X;

                height += texture.Height + 50;
            }

            height -= 50;
        }

        public override void Update(GameTime gameTime)
        {
            Vector2 menuPosition = position;
            Point p = Xin.MouseState.Position;

            Rectangle buttonRect;
            mouseOver = false;

            for (int i = 0; i < menuItems.Count; i++)
            {
                buttonRect = new Rectangle((int)menuPosition.X, (int)menuPosition.Y,
texture.Width, texture.Height);

                if (buttonRect.Contains(p))
                {
                    selectedIndex = i;
                    mouseOver = true;
                }

                menuPosition.Y += texture.Height + 50;
            }

            if (!mouseOver && (Xin.CheckKeyReleased(Keys.Up)))
            {
                selectedIndex--;
                if (selectedIndex < 0)
                {
                    selectedIndex = menuItems.Count - 1;
                }
```

```
            }
            else if (!mouseOver && (Xin.CheckKeyReleased(Keys.Down)))
            {
                selectedIndex++;
                if (selectedIndex > menuItems.Count - 1)
                {
                    selectedIndex = 0;
                }
            }
        }

        public override void Draw(GameTime gameTime)
        {
            Vector2 menuPosition = position;
            Color myColor;
            float myScale;

            for (int i = 0; i < menuItems.Count; i++)
            {
                if (i == SelectedIndex)
                {
                    myColor = HiliteColor;
                    myScale = (2f + (float)Math.Sin(gameTime.TotalGameTime.TotalSeconds * 2)) /
2.5f;
                }
                else
                {
                    myColor = NormalColor;
                    myScale = 1f;
                }

                gameRef.SpriteBatch.Draw(texture, menuPosition, Color.White);

                Vector2 textSize = FontManager.GetFont("testfont").MeasureString(menuItems[i]);

                Vector2 textPosition = menuPosition + new Vector2((int)(texture.Width -
textSize.X) / 2, (int)(texture.Height - textSize.Y) / 2);
                gameRef.SpriteBatch.DrawString(
                    FontManager.GetFont("testfont"),
                    menuItems[i],
                    textPosition,
                    myColor,
                    0f,
                    Vector2.Zero,
                    myScale,
                    SpriteEffects.None,
                    1f);

                menuPosition.Y += texture.Height + 50;
            }
        }

        #endregion Methods

        #region Virtual Methods
        #endregion Virtual Methods

    }
}
```

The class inherits from the DrawableGameComponent class. That is so its Update and Draw methods will be called automatically if it is added to a list of child components.

There are a lot of fields in this class. There is a Game1 field, gameRef, that gives access to a SpriteBatch for drawing. There is a List<string> that is the text for the menu items. The selectedIndex field is what menu item is currently selected. The mouseOver field determines if the mouse is over a button. The width and height fields are the width and height of the menu. The normalColor field is what color to draw text in and hiliteColor is the color to draw the currently selected field in. There is also a texture field that is the texture for the back ground button. Finally position is where the menu is drawn. There are properties to expose the values of the fields except the gameRef field.

There are two constructors for the class. The first takes a Game parameter and a Texture2D parameter. The game parameter is for the Game1 field. The Texture2D is the image for the button. The constructor initializes the fields that correspond to the parameters and sets the mouseOver field to false. The second takes the same parameters as the first plus an array of strings that is the menu items. It calls the first constructor. It sets the selectedIndex field to 0, the first menu item. It then loops over the strings and adds the string for the menu item to the menuItems collection. It then calls MeassureMenu which calculates the height and width of the menu.

SetMenuItems is used to set the menu items, as the name implies. It requires an array of string as a parameter that is the menu items. It removes the existing menu items. The items are then added. The items are then added to the list of menu items. It calls MeassureMenu to measure the menu so it can be positioned. The selectedIndex is set to 0.

MeassureMenu is used to calculate the width and height of the menu. The width is set to the width of the texture for menu items. There is a loop that loops over the menu items. Using the MeasureString method of the SpriteFont class and the testfont I grab the size of the string. If the size of the string is greater than the current width the width is set to the size of the string. The height is set to the Height property of the texture plus a padding of 50. I then remove the padding from the last menu item.

The Update method sets a local variable to the position of the menu. The position of the mouse is saved as point to a local variable. There is a local variable buttonRect that is the rectangle around the button. The mouseOver field is set to false. I then loop over all of the menu items. I create a rectangle using the menu position and the height and width of the texture. If the buttonRect contains the cursor position the selected index is set to the loop variable and mouseOver is set to true. The Y coordinate of the menuPosition field is incremented by the height of the texture plus 50.

If the mouse is not over the menu and the Up key has been released selectedIndex is decremented. If it becomes negative it is set to the number of menu items minus one. Else if the mouse is not over and the Down key has been released selectedIndex is incremented. If the selected index is greater than the number of menu items minus one it is set to zero. The Draw method has a local variable menuPosition that is set to the position of the menu. There are local variables myColor that says what color to draw a menu item in and myScale which scales the selected menu item. It is included for people that have problems seeing colors. It provided a visual cue what it currently selected other than color. The method then

loops over the menu items. If the current loop index is the SelectedIndex property myColor is set to HiliteColor and scale is set to 2 + the sine of the total elapsed seconds in the game times 2. This is then divided by 2.5. Because sine oscillates between -1 and 1 myScale will grow and shrink. If the loop index is not the SelectedIndex property myColor is set to NormalColor and myScale is set to 1. Next it draws the background texture at the current position. I then measure the size of the string. I then center the string inside the texture, without including scaling. I then draw the string. The menu position is then incremented by the texture height plus 50.

For changing resolution we need a we a way for the user to cycle through the supported resolutions. The approach that I'm going to take is a selector with left and right buttons that move the selection left or right one. For the buttons I used a GUI pack by looneybits on https://opengameart.org. You can download the GUI pack using this link: https://opengameart.org/content/gui-buttons-vol1. After you've downloaded the GUI pack extract it to a folder. Open the MonoGame Pipeline Tool. Right click the Content, select Add and then New Folder. Name this new folder GUI. Right click the GUI folder select Add and then Existing Item. Browse to the gui_buttons_vol1\sprites\buttons in the extracted folder. Select all of the .png files and add them. When prompted select the copy option and the all items check box.

I will now add a new control I called LeftRightSelector. Right click the Controls folder in the SolutionExplorer, select Add and then class. Name this new class LeftRightSelector. Here is the code for that class.

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

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

        public event EventHandler SelectionChanged;

        #endregion

        #region Field Region

        private List<string> _items = new List<string>();

        private Texture2D _leftTexture;
        private Texture2D _rightTexture;
        private Texture2D _stopTexture;

        private Color _selectedColor = Color.Red;
        private int _maxItemWidth;
        private int _selectedItem;
        private Rectangle _leftSide = new Rectangle();
```

```
private Rectangle _rightSide = new Rectangle();
private int _yOffset;

#endregion

#region Property Region

public Color SelectedColor
{
    get { return _selectedColor; }
    set { _selectedColor = value; }
}

public int SelectedIndex
{
    get { return _selectedItem; }
    set { _selectedItem = (int)MathHelper.Clamp(value, 0f, _items.Count - 1); }
}

public string SelectedItem
{
    get { return Items[_selectedItem]; }
}

public List<string> Items
{
    get { return _items; }
}

public int MaxItemWidth
{
    get { return _maxItemWidth; }
    set { _maxItemWidth = value; }
}

#endregion

#region Constructor Region

public LeftRightSelector(Texture2D leftArrow, Texture2D rightArrow, Texture2D stop)
{
    _leftTexture = leftArrow;
    _rightTexture = rightArrow;
    _stopTexture = stop;
    TabStop = true;
    Color = Color.White;
}

#endregion

#region Method Region

public void SetItems(string[] items, int maxWidth)
{
    this._items.Clear();

    foreach (string s in items)
        this._items.Add(s);

    _maxItemWidth = maxWidth;
```

```
        }

        protected void OnSelectionChanged()
        {
            if (SelectionChanged != null)
            {
                SelectionChanged(this, null);
            }
        }

        #endregion

        #region Abstract Method Region

        public override void Update(GameTime gameTime)
        {
            HandleMouseInput();
        }

        public override void Draw(SpriteBatch spriteBatch)
        {
            Vector2 drawTo = _position;
            _spriteFont = FontManager.GetFont("testfont");

            _yOffset = (int)((_spriteFont.MeasureString("W").Y - _leftTexture.Height) / 2);
            _leftSide = new Rectangle((int)_position.X, (int)_position.Y + _yOffset,
_leftTexture.Width, _leftTexture.Height);
            //if (selectedItem != 0)
            spriteBatch.Draw(_leftTexture, new Vector2(drawTo.X, drawTo.Y + _yOffset),
Color.White);
            //else
            //    spriteBatch.Draw(stopTexture, drawTo, Color.White);

            drawTo.X += _leftTexture.Width + 5f;

            float itemWidth = _spriteFont.MeasureString(_items[_selectedItem]).X;
            float offset = (_maxItemWidth - itemWidth) / 2;

            drawTo.X += offset;

            if (_hasFocus)
                spriteBatch.DrawString(_spriteFont, _items[_selectedItem], drawTo,
_selectedColor);
            else
                spriteBatch.DrawString(_spriteFont, _items[_selectedItem], drawTo, Color);

            drawTo.X += -1 * offset + _maxItemWidth + 5f;

            _rightSide = new Rectangle((int)drawTo.X, (int)drawTo.Y + _yOffset,
_rightTexture.Width, _rightTexture.Height);
            //if (selectedItem != items.Count - 1)
            spriteBatch.Draw(_rightTexture, new Vector2(drawTo.X, drawTo.Y + _yOffset),
Color.White);
            //else
            //    spriteBatch.Draw(stopTexture, drawTo, Color.White);
        }

        public override void HandleInput()
        {
            if (_items.Count == 0)
```

```
                return;

            if (Xin.CheckKeyReleased(Keys.Left))
            {
                _selectedItem--;
                if (_selectedItem < 0)
                    _selectedItem = this.Items.Count - 1;
                OnSelectionChanged();
            }

            if (Xin.CheckKeyReleased(Keys.Right))
            {
                _selectedItem++;
                if (_selectedItem >= _items.Count)
                    _selectedItem = 0;
                OnSelectionChanged();
            }
        }

        private void HandleMouseInput()
        {
            if (Xin.CheckMouseReleased(MouseButtons.Left))
            {
                Point mouse = Xin.MouseAsPoint;

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

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

        #endregion
    }
}
```

The class inherits from Control. The control has an event handler SelectionChanged that will be triggered when the selection changes. There is a List<string> that holds the items for the control. There are three Texture2D fields. The _leftTexture field is the button for the left side of the control. The _rightTexture field is the button for the right side of the control. The _stopTexture field is for future use. The _selectedColor field is the color to draw the control if it is the current control. The _maxItemWidth field is the width of text area of the control. The _selectedItem field is what item is currently selected. The _leftSide field describes the position of the _leftSideTexture and _rightSide field describes the position of the _rightSideTexture. Finally _yOffset is used in drawing items.

There are properties for the _selectedColor, _selectedIndex, _maxItemWidth and _items

fields. SelectedIndex clamps the selected item between 0 and the number of items minus one. There is also a read only property SelectedItem that returns the selected item.

The constructor takes three Texture2Ds. The first is the left arrow texture, the second is the right arrow texture and the last is the stop texture for future use. The fields are set to the corresponding parameters. The TabStop property of the control is set to true and the Color property is set to white.

The SetItems method takes a string array and a maximum width parameter. It clears the existing item collection. It loops over the items in the array and adds them to the _items collection. The _maxItemWIdth is set to maxWidth.

The OnSelectionChanged method is called when the selection changes. It checks to see if the event has been subscribed to. If it has it invokes the event passing in this and null.

In the override of the Update method I call a method HandleMouseInput that as the name implies handles the mouse.

The Draw method has a local variable drawTo the is used to position items. It's initial value is the position of the control. I then set the _spriteFont field to be the font returned by the FontManager. I measure the W character which is the largest character and then center it based on the height of left texture. I create the _leftSide field using the position of the control. I then draw the left texture. There is some commented out code that is there for future use. I increment the X property of drawTo by 5 pixels. I then measure the item so I can center it horizontally. If the control has focus the text is drawn using _selectedColor or it is drawn in Color. Next drawTo is updated to position the text 5 pixels right of the maximum width. I then create a Rectangle that describes the right texture. There is another bit of commented out code for future use. The right texture is then drawn.

In the override of the HandleInput method I check to see if the item count is zero. If it is I exit out of the method. I then check to see if the Left key has been released. If it has I decrement the selected index. If the selected index is less than zero I set it to be the count property minus one. I then call OnSelectionChanged. Similarly for the Right arrow key I increment the selected item. If it is great than or equal to the Count property it is set to 0. I then call the OnSelectionChanged method.

The HandleMouseInput method checks to see if the left mouse button has been released. If it has it grabs the mouse's position as a Point. If the _leftSide Rectangle contains the position I decrement the selected index. If the selected index is less than zero I set it to be the count property minus one. I then call OnSelectionChanged. If the _rightSide Rectangle contains the position I increment the selected item. If it is great than or equal to the Count property it is set to 0. I then call the OnSelectionChanged method.

I added another control, a button. Right click the Controls folder, select Add then Class. Name this new class Button. This is the new class.

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

```
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace ShadowMonsters.Controls
{
    public class Button : Control
    {
        #region

        public event EventHandler Click;

        #endregion

        #region Field Region

        Texture2D _background;

        #endregion

        #region Property Region
        #endregion

        #region Constructor Region

        public Button(Texture2D background)
        {
            _background = background;
        }

        #endregion

        #region Method Region

        public override void Draw(SpriteBatch spriteBatch)
        {
            spriteBatch.Draw(_background, Position, Color.White);
            _spriteFont = FontManager.GetFont("testfont");
            Vector2 size = _spriteFont.MeasureString(Text);
            Vector2 offset = new Vector2((_background.Width - size.X) / 2, (_background.Height
- size.Y) / 2);
            spriteBatch.DrawString(_spriteFont, Text, Position + offset, Color);
        }

        public override void HandleInput()
        {
            Point position = Xin.MouseAsPoint;
            Rectangle destination = new Rectangle((int)_position.X, (int)_position.Y,
_background.Width, _background.Height);
            if (destination.Contains(position) && Xin.CheckMouseReleased(MouseButtons.Left))
            {
                OnClick();
            }
        }

        private void OnClick()
        {
            Click?.Invoke(this, null);
        }
```

```
        public override void Update(GameTime gameTime)
        {
            HandleInput();
        }

        #endregion
    }
}
```

The class inherits from Control. There is an event handler that will be fired if the button is clicked. There is a Texture2D that is the background image of the button. The constructor requires a Texture2D parameter that is the background for the button. It then sets the field to the value passed in.

The override of the Draw method draws _background field at the Position of the control. I set the _spriteFont field to the font returned by the font manager. I then calculate the size of the string. I the position the text in the center of the button.

In the HandleInput method I get the position of the mouse as a Point. I create a rectangle that describes the position of the button. I check to see if the rectangle contains the point and if the left mouse button has been released. If both are true I call the OnClick method.

In the OnClick method I invoke the delegate if it is not null. In the Update method I call the HandleInput method.

Before I handled the options state I made a few changes to the GamePlayState. I update the Engine field, replaced the hard coded rectangles with rectangles based on settings. I also added a new method that reset the Engine field that was called if the resolution changes. Replace the Editor field, Update, HandlePortals method with the following and add this method.

```
        private Engine engine = new Engine(
            new Rectangle(
                0,
                0,
                Settings.Resolution.X,
                Settings.Resolution.Y),
            Settings.TileSize.X,
            Settings.TileSize.Y);

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

            if ((Xin.CheckKeyReleased(Keys.Space) ||
                Xin.CheckKeyReleased(Keys.Enter)) && frameCount >= 5)
            {
                frameCount = 0;
                StartInteraction();
                HandlePortals();
            }

            if (Xin.CheckKeyReleased(Keys.I))
```

```
    {
        manager.PushState(GameRef.ItemSelectionState);
    }

    if ((Xin.CheckKeyReleased(Keys.B)) && frameCount >= 5)
    {
        frameCount = 0;
        StartBattle();
    }

    if (Xin.CheckKeyReleased(Keys.F1))
    {
        SaveGame();
    }
    if (Xin.CheckKeyReleased(Keys.F2))
    {
        LoadGame();
    }

    Engine.Camera.LockToSprite(
        world.Maps[world.CurrentMapName],
        Game1.Player.Sprite,
        new Rectangle(0, 0, Settings.Resolution.X, Settings.Resolution.Y));
    world.Update(gameTime);
    Game1.Player.Update(gameTime);

    base.Update(gameTime);
}

private void HandlePortals()
{
    foreach (var r in world.Map.PortalLayer.Portals.Keys)
    {
        Portal p = world.Map.PortalLayer.Portals[r];
        Rectangle pRect = new Rectangle(
            (int)Game1.Player.Sprite.Position.X,
            (int)Game1.Player.Sprite.Position.Y,
            Engine.TileWidth,
            Engine.TileHeight);

        if (pRect.Intersects(
            new Rectangle(
                p.SourceTile.X * Engine.TileWidth,
                p.SourceTile.Y * Engine.TileHeight,
                Engine.TileHeight,
                Engine.TileHeight)) ||
                (Xin.CheckMouseReleased(MouseButtons.Left) &&
                    new Rectangle(
                        p.SourceTile.X,
                        p.SourceTile.Y,
                        Engine.TileWidth,
                        Engine.TileHeight).Contains(Xin.MouseAsPoint)))
        {
            world.ChangeMap(p);

            Game1.Player.Position = new Vector2(
                p.DestinationTile.X * Engine.TileWidth,
                p.DestinationTile.Y * Engine.TileHeight);
            Engine.Camera.LockToSprite(
                world.Map,
```

```
                    Game1.Player.Sprite,
                    new Rectangle(0, 0, Settings.Resolution.X, Settings.Resolution.Y));
                Game1.Player.MapName = world.CurrentMapName;
                return;
            }
        }
    }

    public void ResetEngine()
    {
        engine = new Engine(
            new Rectangle(0, 0, Settings.Resolution.X, Settings.Resolution.Y),
            Settings.TileSize.X,
            Settings.TileSize.Y);
    }
```

The new code initializes the engine field using the Resolution property and TileSize property of the Settings class. When calling LockToSprite I create the Rectangle property passing in the resolution in the Settings class.

Now I have everything I need to implement the options state. Right click the GameStates folder in the Solution Explorer, select Add and then Class. Name this new class OptionState. Here is the code.

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

namespace ShadowMonsters.GameStates
{
    public class OptionState : BaseGameState
    {
        #region Field Region

        LeftRightSelector resolutions;
        Button apply;
        Button back;

        #endregion

        #region Property Region
        #endregion

        #region Constructor Region

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

        #endregion

        #region Method Region
```

```
        protected override void LoadContent()
        {
            base.LoadContent();

            resolutions = new LeftRightSelector(
                content.Load<Texture2D>(@"GUI\g22987"),
                content.Load<Texture2D>(@"GUI\g21245"),
                null);

            resolutions.Position = new Vector2(100, 100);

            List<string> items = new List<string>();

            foreach (string s in Game1.Resolutions.Keys)
                items.Add(s);

            resolutions.SetItems(items.ToArray(), 400);

            apply = new Button(content.Load<Texture2D>(@"GUI\g9202"))
            {
                Text = "Apply",
                Position = new Vector2(700, 87)
            };

            apply.Click += Apply_Click;

            back = new Button(content.Load<Texture2D>(@"GUI\g9202"))
            {
                Text = "Back",
                Position = new Vector2(100, 200)
            };

            back.Click += Back_Click;
        }

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

        private void Apply_Click(object sender, EventArgs e)
        {
            GameRef.GraphicsDeviceManager.PreferredBackBufferWidth =
Game1.Resolutions[resolutions.SelectedItem].X;
            GameRef.GraphicsDeviceManager.PreferredBackBufferHeight =
Game1.Resolutions[resolutions.SelectedItem].Y;
            GameRef.GraphicsDeviceManager.ApplyChanges();
            Settings.Resolution = Game1.Resolutions[resolutions.SelectedItem];
        }

        public override void Update(GameTime gameTime)
        {
            resolutions.Update(gameTime);
            apply.Update(gameTime);
            back.Update(gameTime);

            base.Update(gameTime);
        }
```

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

            GameRef.SpriteBatch.Begin();
            resolutions.Draw(GameRef.SpriteBatch);
            apply.Draw(GameRef.SpriteBatch);
            back.Draw(GameRef.SpriteBatch);
            GameRef.SpriteBatch.End();
        }

        #endregion
    }
}
```

The class inherits from BaseGameState. It has a LeftRightSelector field and two Button fields. In the LoadContent method I create the controls. I pass in two textures from the GUI pack that we added earlier and null for the stop texture because we're not implementing it yet. I set the position of the control to 100, 100. There is next a List<string> that will hold the text part of the resolutions. I loop over the keys in the Resolutions collection and add them to the list. I pass in the list to an array and a max width of 400 to the control. Next I create the Apply button passing in one of the textures from the GUI pack and setting the Text property to Apply and the Position to 700, 87. I then wire an event handler for the Click event of the button. I then create the back button the same way and wire the event handler for the Click event.

In the event handler for the Click event of the back button I pop this state off the stack of states and call the ResetEngine method of the GamePlayState. In the event handler for the click event of the apply button I set the Width and Height of the GraphicsDeviceManager to the width and height of the selected resolution and apply the changes. I then set the resolution in the settings to the selected resolution.

In the Update method I call the Update method of the controls. In the Draw method I call Begin on the SpriteBatch object. I then call the Draw method of the controls. Finally I call the End method of the controls.

I will add the main menu state now. Right click the GameStates folder in the Solution Explorer, select Add and then class. Name this new class MainMenuState. Here is the code.

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

namespace ShadowMonsters.GameStates
{
    public class MainMenuState : BaseGameState
    {
```

```
        #region Field Region

        MenuComponent menu;
        private int frameCount;

        #endregion

        #region Property Region
        #endregion

        #region Constructor Region

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

        #endregion

        #region Method Region

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

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

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

            texture.SetData(buffer);

            menu = new MenuComponent(GameRef, texture);
            Components.Add(menu);
        }

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

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

            if (Xin.CheckKeyReleased(Keys.Enter)
                || (menu.MouseOver && Xin.CheckMouseReleased(MouseButtons.Left)) && frameCount
> 5)
            {
                switch (menu.SelectedIndex)
                {
                    case 0:
                        manager.PopState();
                        manager.PushState(GameRef.GamePlayState);
                        GameRef.GamePlayState.SetUpNewGame();
                        break;
                    case 1:
                        string path = Environment.GetFolderPath(
                            Environment.SpecialFolder.ApplicationData);
                        path += "\\ShadowMonsters\\ShadowMonsters.sav";
```

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

                        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);
        }

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

        #endregion
    }
}
```

Like the other game states this class inherits from BaseGameState. There is a MenuComponent field for the menu and a frameCount field that counts the number of frames since the last input.

In the LoadContent method I create a texture for the menu component. I then create the menu component. The menu is then added to the components.

In the Update method I increment the frameCount variable. If the SelectedIndex property of the menu component is -1 I call SetMenuItems passing in an array consisting of New Game, Continue, Options and Exit. I then check if the Enter key or if the mouse is over a menu item and the left mouse button is released and frameCount is greater than 5. There is then a switch on the SelectedIndex property of the menu. If the selected index is zero I pop the menu off the stack and then push the game play state onto the stack and call the SetUpNewGame method of the game play state method to start a new game. If the selected index is one I create a path to the save game file. If the file does not exist I exit the method. Then I fill the moves and load the shadow monsters. I pop the state off the stack and push the game play state on the stack. I then call LoadGame method to continue off where the player left off. If the case is 2 I push the option state on the stack. If the case is 3 I exit the game. I then set frameCount to zero.

In the Draw method I call the Begin method of the sprite batch to start drawing. I then call base.Draw to draw the child components. I then call the End method of the sprite batch to end drawing.

The last thing that I'm going to cover in this tutorial is implementing the changes made. First, I added fields for the new states to the Game1 state. Then I added properties to expose them and a property to expose the GraphicsDeviceManager. I updated the constructor to create the new states and push the main menu state on top of the stack. Add the following fields and properties to the Game1 class and change the constructor.

```
private readonly MainMenuState mainMenuState;
private readonly OptionState optionState;

public MainMenuState MainMenuState => mainMenuState;
public OptionState OptionState => optionState;
public GraphicsDeviceManager Graphics => graphics;

public Game1()
{
    graphics = new GraphicsDeviceManager(this);

    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);

    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);

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

There is still a bit more work to do where handling multiple resolutions. Mostly positioning text and GUI elements dynamically based on the difference between scaling. That is a little more

complicated than what we did with game play. I will plan out how we will tackle that in a future tutorial,
I am going to wrap this tutorial up here. 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!