A Summoner's Tale – MonoGame Tutorial Series

Chapter 2

More Plumbing/Scaffolding

This tutorial series is about creating a Pokemon style game with the MonoGame Framework called A Summoner's Tale. 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 web site: A Summoner's Tale. The source code for each tutorial will be available as well. I will be using Visual Studio 2013 Premium for the series. The code should compile on the 2013 Express version and Visual 2015 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 add a link to my site, http://gameprogrammingadventures.org, and credit to Jamie McMahon.

In the last tutorial I set up the base project with the state manager and a title/intro screen. In this tutorial I will be adding more plumbing/scaffolding to the game as there are some key components that are missing. I also discovered after loading up my project that there was a problem with the fonts that were added. I've fixed this in the project. First, I renamed InterfaceFont.interfacefont to InterfaceFont.spritefont. I also added a new font called GameFont.spritefont using the same process as in the last tutorial.

A quick note on fonts while I'm on the subject. Most fonts are commercial products and cannot be used in your game without paying royalties or buying the font out right. There are a lot of fonts that are available to game developers though. The two sources that I use are http://www.1001fonts.com and http://www.dafont.com. If you filter you will find many different licenses, including 100% free. I'd suggest that if you find a font and use it in your game that you mention to creator somewhere in your credits.

In the last tutorial we got the state manager and added a state and it is rendering. Other than that it doesn't do anything. In order for it to do something it will have to react to user input. I'm going to add a basic version of my input handling component that I wrote for use in XNA 3.0, a long time ago now, called Xin. It attempts to centralize the important code for handling input to try and reduce complexity in the rest of the game. It is pretty common for a developer to do this sort of abstraction process. Find a common task and create a class that wraps that task so that it can be used again in other projects.

First, right click your project and select Add and then New Folder. Name this new folder Components. Now, right click the Components folder that was just added, select Add and then Class. Name this new class Xin. The code for that class follows.

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

namespace Avatars.Components
{
    public enum MouseButtons
    {
        Left,
        Right,
        Center
    }

    public class Xin : GameComponent
    {
        private static KeyboardState currentKeyboardState = Keyboard.GetState();
        private static KeyboardState previousKeyboardState = Keyboard.GetState();

        private static MouseState currentMouseState = Mouse.GetState();
        private static MouseState previousMouseState = Mouse.GetState();

        public static MouseState MouseState
        {
            get { return currentMouseState; }
        }

        public static KeyboardState KeyboardState
        {
            get { return currentKeyboardState; }
        }

        public static KeyboardState PreviousKeyboardState
        {
            get { return previousKeyboardState; }
        }

        public static MouseState PreviousMouseState
        {
            get { return previousMouseState; }
        }

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

        public override void Update(GameTime gameTime)
        {
            Xin.previousKeyboardState = Xin.currentKeyboardState;
            Xin.currentKeyboardState = Keyboard.GetState();

            Xin.previousMouseState = Xin.currentMouseState;
            Xin.currentMouseState = Mouse.GetState();

            base.Update(gameTime);
        }

        public static void FlushInput()
        {
            currentMouseState = previousMouseState;
            currentKeyboardState = previousKeyboardState;
        }
```

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

        public static bool CheckMouseReleased(MouseButtons button)
        {
            switch (button)
            {
                case MouseButtons.Left :
                    return (currentMouseState.LeftButton == ButtonState.Released) &&
(previousMouseState.LeftButton == ButtonState.Pressed);
                case MouseButtons.Right :
                    return (currentMouseState.RightButton == ButtonState.Released) &&
(previousMouseState.RightButton == ButtonState.Pressed);
                case MouseButtons.Center:
                    return (currentMouseState.MiddleButton == ButtonState.Released) &&
(previousMouseState.MiddleButton == ButtonState.Pressed);
            }

            return false;
        }
    }
}
```

You will see that there are a lot of static members in this class. I did that so instead of creating an instance of the class in each other class that it is used in you have a single point for handling input. When you need to do anything input related you just use Xin.member_name.

First, you will notice that there is an enumeration at the namespace level called MouseButtons. I added this because on the one thing that I've always found lacking in XNA and MonoGame because it derives from it is mouse support. What I mean is the keyboard and game pad states can be referenced using the Keys and Buttons enumerations where as for mouse input you have no such option.

This class then derives from GameComponent. I did that so that we can create an instance of Xin and add it to the list of components without creating a member variable in a class and forget about it, for all intents and purposes.

Next there are two variable for KeyboardState and MouseState. These variables hold the current state of the mouse and keyboard and the previous state of the mouse and the keyboard. You need both to check to see when a button or key is first pressed or when it is released. Otherwise you can't tell if a button was down since the last frame or if it was released since the last frame. This will make more sense in a bit.

I also added static readonly properties for both input types and their frame, whether they are the current frame of the game or last frame of the game. These provide raw access to input which is important in some instances.

The constructor is trivial really. It just calls the base constructor that requires a Game parameter that represents the game object the component will be added to. You could in theory require more than one Game object in a game but I've never had to use one.

The methods that follow are the real meet of the class and do most of the work. I will be extending these as time goes on but for the purposes of this tutorial they are sufficient. The first method,

Update, is inherited from the GameComponet class. The way this works is if you create a component and add it to the list of components in a game their Update method is called automatically each frame, if the component is enabled. Inside the update method I first set the variables named previous to current. I then get the current state of the keyboard and mouse. This is how we will detect when a key is pressed or release, but not if it is just down or up. I then call base.Update which would call Update on other related components inside of this class.

The first static method is FlushInput. What this method does is set the current states to their previous states. Since they are now equal and of the methods that would check for new presses or releases will fail.

In CheckKeyReleased checks to see if a key that was down last frame is now up. Let me explain why I did this was as opposed to a key that was up in the previous frame is down in the current frame. What I've found checking it the second way, the key is now down, is that before the call to Xin.FlushInput happens sometimes the new press is caught in the next frame of the game. Checking for a release works around this phenomenon. To do this I use the IsDown and IsUp member methods of the KeyboardState class, which do as you might imagine.

The last, and most interesting, method is the CheckMouseReleased. I say that it is interesting is that compared to checking for keys, checking for mouse buttons is more complex. There are no built in methods of the input classes that check if a mouse button is down. All that there is are named properties that get a ButtonState enumeration that defines if the button is up or down. So, in order to check a button you retrieve the ButtonState and then check its value.

So, in the CheckMouseReleased button I have a switch statement that checks what parameter was passed into the method. Based on what button I have a statement that returns the current state of the button compared with the previous state of the button. If no other response is returned I then return false.

That is all for the Xin class. As you can see there is room for improvement, and you might be thinking of that already, which is good. It is always good to read code, see how it works and then try to implement it or another version of it. It is never a good idea to just copy and paste code without understanding it. That is a speech that I will spare you from.

The next item that I want to add is a menu component that will be used for the main menu and other menus in the game. Right click the Components folder, select Add and then Class. Name this new class MenuComponent. Here is the code for that class.

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

using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;

namespace Avatars.Components
{
    public class MenuComponent
    {
        #region Fields

        SpriteFont spriteFont;
```

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

        int width;
        int height;

        Color normalColor = Color.White;
        Color hiliteColor = Color.Red;

        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(SpriteFont spriteFont, Texture2D texture)
        {
            this.mouseOver = false;
            this.spriteFont = spriteFont;
            this.texture = texture;
        }

        public MenuComponent(SpriteFont spriteFont, Texture2D texture, string[] menuItems)
            : this(spriteFont, 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 = spriteFont.MeasureString(s);

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

                height += texture.Height + 50;
            }

            height -= 50;
        }

        public void Update(GameTime gameTime, PlayerIndex index)
        {
            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 void Draw(GameTime gameTime, SpriteBatch spriteBatch)
        {
            Vector2 menuPosition = position;
            Color myColor;

            for (int i = 0; i < menuItems.Count; i++)
            {
                if (i == SelectedIndex)
                    myColor = HiliteColor;
                else
                    myColor = NormalColor;

                spriteBatch.Draw(texture, menuPosition, Color.White);

                Vector2 textSize = spriteFont.MeasureString(menuItems[i]);

                Vector2 textPosition = menuPosition + new Vector2((int)(texture.Width -
textSize.X) / 2, (int)(texture.Height - textSize.Y) / 2);
                spriteBatch.DrawString(spriteFont,
                    menuItems[i],
                    textPosition,
                    myColor);

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

        #endregion Methods

        #region Virtual Methods
        #endregion Virtual Methods

    }
}
```

Quite a bit of code but it is nothing that is overly complex. First, there are a number of member fields in the class. Since the menu needs to draw text I added a SpriteFont field. There is then a List<string> that will contain the individual menu items to be rendered. The selectedIndex field will return what menu item is currently selected. Next mouseOver is added to allow for mouse support. The width and height fields will be used to control where the menu is rendered. There are two Color fields that control what color menu items are rendered. There is one color for regular menu items and

then a second for highlighted menu items. Texture is a field that will serve as a button background image and position controls where the menu it displayed.

I added some public properties that will expose some of the fields to other classes. You should be able to set the position of the menu items so I exposed the position member variable. Frequently you will need to know the width or height of the menu so I added properties for that as well. I added a property to return and set the selected menu item. You will see that in the set part I use MathHelper.Clamp to make sure the selected item is not outside of the available menu items. There are properties that allow you to get and set the menu colors as well. Finally there is a property that will return the mouseOver member variable.

You will notice that I did not add a property that returns the menu items. The reason is that when the menu items change I want to force a call to a method that will find the height and width of the menu. If I exposed the list it can be modified outside of the class and changes would not be picked up and could lead to a change in the was the menu is rendered.

I added two constructors. The first takes two parameters, the font the menu will be rendered with and the button texture. This one just initializes some of the fields that I added. The second constructor accepts an array of strings as well as the others. It will also call the first constructor to set some of the default values. Inside I set the selectedIndex member variable to 0, the first item, and then in a foreach loop I add each string to the menu items. I then call MeasureMenu to calculate the height and width of the menu.

The first method is called to change the menu items for the menu. It takes a string array that hold the menu items to be displayed. If clears any existing menu items, adds them to the list of menu items, calls the MeasureMenu method to calculate the width and height then finally sets the selectedIndex member variable to 0.

MeasureMenu is where I calculate the width and height of the menu. First, I set width to be the width of the texture. You technically do not want to menu items to be longer than the width of your button but you need to allow for this scenario. I reset the height to 0 as well. In a foreach loop I iterate over all of the menu items. I use the MeasureString member of the SpriteFont class to get the height and width of the string. If the X value is greater than the width than the current width I update that value. I then add the height of the button and some padding to the height member variable. I then subtract the padding as it is added to the last menu item.

In the Update method I handle updating the menu component. I added a local variables that hold the position that the menu is drawn and the mouse as a Point. I then have a Rectangle that will be the bounds of each menu item to determine if the mouse is over them or not. After the variables I set the mouseOver variable to false. This ensures that it is reset at the start of each frame of the game.

Next there is a for loop that loops over the menu items one by one. Inside I calculate the Rectangle that holds the bounds of the button. I use the local variable that is set to be the position of the menu and the height and width of the button texture. I check to see if the mouse is over the current button. If it is I set the selectedIndex member to be that menu and the mouseOver variable to true. I add the padding and the texture height to the current menu item position before moving to the next item.

The if statement that follows handles the user moving the selected menu item using the up and down keys. I do not change the position if the mouse is over a button as that is the button that should have focus. To move the selection up I decrement the selectedIndex member. If it is less than zero I set the menu item to last menu item, which is the Count member of the list of menu items minus one, since it

is zero based. To move down I increment the selectedIndex member. I then check if it is outside the bounds of the list and if it is set it to zero.

The last method is the Draw method that will render the menu items. There are two local variables that hold the menu position and color that the items will be drawn in. You then loop over each of the menu items. I first set the color variable based on what the selected index is. I then draw the button image. I measure the menu item string. I then calculate its position relative to the button texture and center it. I then draw the menu item string. Finally I update the Y coordinate by adding the height of the texture plus the padding.

The last new class that I'm going to add is the start state that displays a menu to the user allowing them to pick options. Right click the GameStates folder, select Add and then Class. Name this new class MainMenuState. Here is the code for that class.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using Avatars.Components;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;

namespace Avatars.GameStates
{
    public interface IMainMenuState : IGameState
    {
    }

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

        Texture2D background;
        SpriteFont spriteFont;
        MenuComponent menuComponent;

        #endregion

        #region Property Region
        #endregion

        #region Constructor Region

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

        #endregion

        #region Method Region

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

        protected override void LoadContent()
        {
            spriteFont = Game.Content.Load<SpriteFont>(@"Fonts\InterfaceFont");
```

```
            background = Game.Content.Load<Texture2D>(@"GameScreens\menuscreen");

            Texture2D texture = Game.Content.Load<Texture2D>(@"Misc\wooden-button");

            string[] menuItems = { "NEW GAME", "CONTINUE", "OPTIONS", "EXIT" };

            menuComponent = new MenuComponent(spriteFont, texture, menuItems);

            Vector2 position = new Vector2();

            position.Y = 90;
            position.X = 1200 - menuComponent.Width;

            menuComponent.Postion = position;

            base.LoadContent();
        }

        public override void Update(GameTime gameTime)
        {
            menuComponent.Update(gameTime);

            if (Xin.CheckKeyReleased(Keys.Space) || Xin.CheckKeyReleased(Keys.Enter) ||
(menuComponent.MouseOver && Xin.CheckMouseReleased(MouseButtons.Left)))
            {
                if (menuComponent.SelectedIndex == 0)
                {
                    Xin.FlushInput();
                }
                else if (menuComponent.SelectedIndex == 1)
                {
                    Xin.FlushInput();
                }
                else if (menuComponent.SelectedIndex == 2)
                {
                    Xin.FlushInput();
                }
                else if (menuComponent.SelectedIndex == 3)
                {
                    Game.Exit();
                }
            }

            base.Update(gameTime);
        }

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

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

            GameRef.SpriteBatch.End();

            base.Draw(gameTime);

            GameRef.SpriteBatch.Begin();

            menuComponent.Draw(gameTime, GameRef.SpriteBatch);

            GameRef.SpriteBatch.End();

        }

        #endregion
    }
```

```
}
```

First, there are using statements to bring the Components namespace in this project as well as a few of the Xna.Framework namespace. There is an empty interface at the start of this class called IMainMenuState. One use for interfaces is to define a contract that other objects can use to interact with this object. An empty interface means that there are no members that will be exposed directly to other objects.

The class itself inherits from BaseGameState so it can be used in the state manager and it implements the ImainMenuSate interface. There are three private member variables: background, spriteFont and menuComponent. The background variable will hold the background image that will be render, spriteFont is for drawing any text and menuComponent will render our menu.

In the constructor I register the class as a service with the framework so it can be retrieved in other classes. This one won't be but I'm being consistent will all states.

In the LoadContent menu I load the font and background. I also load an image for the menu items. Download the Misc content items from this link and extract the file. Back in the project open the content manager by left clicking Content.mgcb and selecting Open. Select the Content node and create a new folder Misc. Right click the Misc folder, select Add and the Existing Item. Browse to the wooden-button.png file and add it. Now, right click the GameScreens folder under the Content folder. If you haven't added the menuscreen.png file to this folder do so now. Once all items are added open the Build menu and select Rebuild.

After loading the content I create a string that holds the menu items that will be displayed and create the menu component. I then position the menu, based on the image that I created for the background. The buttons should sit on top of the chains on the right side of the screen.

In the Update method I first call the Update method of the menu before checking if the space, enter or left mouse button have been released since the last frame. There is then a chain of if-else statements, one for each of the menu items. Inside the if statement I call Xin.FlushInput just to make sure there are no leftovers. The interesting part is that in the last index I call Game.Exit to close the game. It would be better if you put up a pop up asking if the player that they really want to end the game. I will implement that in a future tutorial.

The Draw method is pretty simple. It first renders the background, calls base.Draw and then renders the menu.

Now, I'm going to implement the title screen changing state to the menu state when space, enter or the left mouse button are released. First, update the using statements in the class to make sure all necessary entities are in scope for the class.

```
using System;
using Avatars.Components;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
```

Now, update the Update method to the following. You will get an error message on the state change because we have not yet updated the main game class yet. That is what we will tackle next.

```
public override void Update(GameTime gameTime)
{
    PlayerIndex? index = null;

    elapsed += gameTime.ElapsedGameTime;

    if (Xin.CheckKeyReleased(Keys.Space) || Xin.CheckKeyReleased(Keys.Enter) ||
Xin.CheckMouseReleased(MouseButtons.Left))
    {
        manager.ChangeState((MainMenuState)GameRef.StartMenuState, index);
    }

    base.Update(gameTime);
}
```

There is a nullable variable, index, for game pad support. I'm not including that at the moment but I will be in the future so I need to park this here. I then call the methods off the Xin class to check for the release of the space key, enter key or left mouse button. If any of those have been released I call the ChangeState method to change the state the menu state. You will have an error until we update the Game1 class.

We're in the home stretch now. All that is remaining is updating the Game1 class to handle this new logic. The changes were pretty extensive so I will paste the code for the entire class.

```
using Avatars.Components;
using Avatars.GameStates;
using Avatars.StateManager;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;

namespace Avatars
{
    public class Game1 : Game
    {
        GraphicsDeviceManager graphics;
        SpriteBatch spriteBatch;

        GameStateManager gameStateManager;

        ITitleIntroState titleIntroState;
        IMainMenuState startMenuState;

        static Rectangle screenRectangle;

        public SpriteBatch SpriteBatch
        {
            get { return spriteBatch; }
        }

        public static Rectangle ScreenRectangle
        {
            get { return screenRectangle; }
        }

        public GameStateManager GameStateManager
        {
            get { return gameStateManager; }
        }

        public ITitleIntroState TitleIntroState
        {
```

```
            get { return titleIntroState; }
        }

        public IMainMenuState StartMenuState
        {
            get { return startMenuState; }
        }

        public Game1()
        {
            graphics = new GraphicsDeviceManager(this);
            Content.RootDirectory = "Content";

            screenRectangle = new Rectangle(0, 0, 1280, 720);

            graphics.PreferredBackBufferWidth = ScreenRectangle.Width;
            graphics.PreferredBackBufferHeight = ScreenRectangle.Height;

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

            this.IsMouseVisible = true;

            titleIntroState = new TitleIntroState(this);
            startMenuState = new MainMenuState(this);

            gameStateManager.ChangeState((TitleIntroState)titleIntroState, PlayerIndex.One);
        }

        protected override void Initialize()
        {
            Components.Add(new Xin(this));

            base.Initialize();
        }

        protected override void LoadContent()
        {
            spriteBatch = new SpriteBatch(GraphicsDevice);
        }

        protected override void UnloadContent()
        {
        }

        protected override void Update(GameTime gameTime)
        {
            if (GamePad.GetState(PlayerIndex.One).Buttons.Back == ButtonState.Pressed ||
Keyboard.GetState().IsKeyDown(Keys.Escape))
                Exit();

            base.Update(gameTime);
        }

        protected override void Draw(GameTime gameTime)
        {
            GraphicsDevice.Clear(Color.CornflowerBlue);

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

So, I first removed all of the extra comments that are part of the template to shorten the class as they are not necessary for the project. I also added in a using statement to bring the Components

namespace into scope. I then added a member variable of type IMainMenuState for the start menu for the game. I also added in some properties to expose the member variables to game objects that we will be creating.

In the constructor for the Game1 class I set the IsMouseVisible property to try so that the mouse cursor will be displayed. I also created the new MainMenuState object. Also, in the Initialize method I add an instance of Xin to the game components for the game.

Before building and running the game, first open the Content manager and build the content for the game. Once the content builds successfully then build and run the game. At this point the title screen will appear. Pressing either the space or enter key or clicking the left mouse button will move to the next scene. Once on the menu scene clicking the Exit button will close the game.

I'm going to end the tutorial here as we've covered a lot. The first two tutorials were a little dry in regard to game elements but at this point most of the plumbing/scaffolding is in place. Now we can move on to game elements. Stay tuned for the next tutorial in the series. In that one I will be implementing and new game state and some game play elements.

I wish you the best in your MonoGame Programming Adventures!
Jamie McMahon