A Summoner’s Tale
Chapter 9
Back to the Future, I Mean Editor

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.

This tutorial is going to go back to the editor. First, I am going to implement two controls. The first is a spin box that you can use to choose an item from a list of items. The second is a control that will render a tile map. Depending on how long that takes, I will implement a floating menu that slides in from the right and then slides back out if it is clicked again.

So, let’s get started. Right-click the Forms folder in the SummonersTale folder, and select Add and the Class. Name this new class, RightLeftSelector. Here is the code for that class.

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

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

        public event EventHandler SelectionChanged;

        #endregion

        #region Field Region

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

        private readonly Texture2D _leftTexture;
        private readonly Texture2D _rightTexture;
        private Color _selectedColor = Color.Red;
        private int _maxItemWidth;
        private int _selectedItem;
        private Rectangle _leftSide = new();
        private Rectangle _rightSide = new();
        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); }
        }

        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 RightLeftSelector(Texture2D leftArrow, Texture2D rightArrow)
        {
            _leftTexture = leftArrow;
            _rightTexture = rightArrow;
            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()
        {
            SelectionChanged?.Invoke(this, null);
        }

        #endregion

        #region Abstract Method Region

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

        public override void Draw(SpriteBatch spriteBatch)
        {
            Vector2 drawTo = _position;


            _spriteFont = ControlManager.SpriteFont;

            _yOffset = (int)((_leftTexture.Height - 
_spriteFont.MeasureString("W").Y) / 2);
            _leftSide = new Rectangle(
                (int)_position.X,
                (int)_position.Y,
                _leftTexture.Width,
                _leftTexture.Height);

            //if (selectedItem != 0)
            spriteBatch.Draw(_leftTexture, _leftSide, 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;

            Vector2 off = new(offset, _yOffset);

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

            drawTo.X += _maxItemWidth + 5f;
            _rightSide = new Rectangle((int)drawTo.X, (int)drawTo.Y,
_rightTexture.Width, _rightTexture.Height);
            //if (selectedItem != items.Count - 1)
            spriteBatch.Draw(_rightTexture, _rightSide, Color.White);
            //else
            //    spriteBatch.Draw(stopTexture, drawTo, Color.White);
        }

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

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

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

        private void HandleMouseInput()
        {
            if (Xin.WasMouseReleased(MouseButton.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
    }
}
```

So, a lot going on here. I will break it down into more manageable parts. First, it inherits from the Control class so it can be added to a control manager. There is an event in the class that will be fired when the selected item changes.

There are a number of fields in the class. The first is a list of items in the selector. The next two are textures for buttons to switch the selection left or right. Now, the next one is kind of redundant in that there will only ever be a single item selected. The next field is the maximum width of an item. Following that is the item in the list that is currently selected. After that are rectangles that define the position of the buttons to move the selected item left or right. The Y offset is used in centering items horizontally between the buttons. There properties to expose the selected color, selected index, selected item, the list of items and the max with to be displayed.

The constructor for the class takes two parameters, textures for moving the selection left or right. It sets the textures to the textures passed in. It sets the TabStop property to true on the colour to white.

There is a method that takes an array of strings and the maximum width to be displayed. It is use to position the left and right buttons. It clears the list of items. Then, in a foreach loop, cycles through the strings passed in and adds them as an item. It sets the maximum item width to the value passed in.

The OnSelectionChaned method is called with the selection changes. It fires the off the event handler if it is subscribed to. In the Update method, I call the HandleMouseInput method.

In the Draw method, I set a local variable drawTo to the position of the control. I also set a local variable to the font in the ControlManager. W is, most often, the widest and tallest character in a font. For that reason, when I go calculate the Y offset, I measure the W character. I then take the height of the left texture and subtract half the Y value to figure out where to draw items. Next, I calculate where to draw the left button. I then call the Draw method to draw. I increment drawTo by the width of the left button plus five pixels. I calculate the width of the currently selected item in the list of items. The offset is then the maximum width minus the item width divided by 2. I create an offset vector using the two components. I see why I added the selected colour now. I draw the selected item white if the control does not have focus and red if it does. After drawing the text, I move the drawTo's X value by the maximum width plus five pixels. I calculate where to draw the right button, then draw it.

In the HandleInput method, I check to see if there are any items in the list of items. If there are not, I exit from the method. I checked to see if the left arrow key was pressed next. If it has, I decrement the selected item by one. If it is less than zero, I set it to the end of the list. Which is, of course, the number of items in the list minus one. I then call OnSelectionChanged to fire the event to any subscribers. The check for the right keys works in much the same way. It increments the selected index by one. If that is greater than or equal to the number of items, the selected index is set to one. It then calls OnSelectChanged.

HandleMouseInput works in much the same way. The difference is that it looks for clicks on the left or right buttons. If the left button has been released, and the mouse is over the left button, the selected index is decremented. If it is less than zero, it wraps to the end of the list. The OnSelectionChange method is called. It works in reverse if the right button is clicked.

Next, I created a control that renders the map to a render target. That control will go on the form. The only confusing part will be that we are rendering the map to a different space than the screen. We will need to transform from screen coordinates to map coordinates. Fortunately, it is not that hard.

Right-click the Forms folder in the SummonersTale project, select Add and then Class. Name this new class MapDisplay. Here is the code for that class.

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

namespace SummonersTale.Forms
{
    public class MapDisplay : Control
    {
        private TileMap _tileMap;
        private readonly int _width;
        private readonly int _height;
        private Tile _selectedTile;
        private GameTime _gameTime;

        public Point Tile { get; set; }
        public bool MouseOver { get; set; }

        public int Width { get { return _width; } }
        public int Height { get { return _height; } }

        public MapDisplay(TileMap map, int width, int height)
        {
            Position = new(0, 20);
            Size = new(width * Engine.TileWidth, height * Engine.TileHeight);
            _tileMap = map;

            _width = width;
            _height = height;

            _tileMap = map;
        }

        public void SetMap(TileMap map)
        {
            _tileMap = map;
            _selectedTile = new(0, 0);
            Engine.Camera.Position = new(0, 0);
        }

        public override void Draw(SpriteBatch spriteBatch)
        {
            if (_tileMap != null)
            {
                _tileMap.Draw(new(), spriteBatch, Engine.Camera, true);

                spriteBatch.Begin();
                spriteBatch.DrawString(ControlManager.SpriteFont, "(" + Tile.X + ",
" + Tile.Y + ")", Vector2.Zero, Color.Red);
                spriteBatch.End();
            }
        }

        public override void HandleInput()
        {
            Rectangle r = new(Helper.V2P(Position), Helper.V2P(Size));
            MouseOver = r.Contains(Xin.MouseAsPoint);

            Vector2 position = new()
            {
                X = Xin.MouseAsPoint.X + Engine.Camera.Position.X,
                Y = Xin.MouseAsPoint.Y + Engine.Camera.Position.Y,
            };

            Tile = Engine.VectorToCell(position);

            if (MouseOver)
            {
                Vector2 vector2 = new();

                if (new Rectangle(0, 0, 128, 1080).Contains(Xin.MouseAsPoint))
                {
                    vector2.X = -1;
                }

                if (new Rectangle(0, 0, 1280, 128).Contains(Xin.MouseAsPoint))
                {
                    vector2.Y = -1;
                }

                if (new Rectangle(1280 - 128, 0, 128,
1080).Contains(Xin.MouseAsPoint))
                {
                    vector2.X = 1;
                }

                if (new Rectangle(0, 1080 - 128, 1280,
128).Contains(Xin.MouseAsPoint))
                {
                    vector2.Y = 1;
                }

                if (vector2 != Vector2.Zero)
                    vector2.Normalize();

                Engine.Camera.Position =
                    Engine.Camera.Position + vector2 * 160f *
(float)_gameTime.ElapsedGameTime.TotalSeconds;

                LockCamera();
            }
        }


        private void LockCamera()
        {
            float y = MathHelper.Clamp(Engine.Camera.Position.Y, 0, 
_tileMap.HeightInPixels - _height);
            float x = MathHelper.Clamp(Engine.Camera.Position.X, 0, 
_tileMap.WidthInPixels - _width);

            Engine.Camera.Position = new(x, y);
        }
        public override void Update(GameTime gameTime)
        {
            _gameTime = gameTime;

            if (_tileMap != null)
            {
                _tileMap.Update(gameTime);
                HandleInput();
                Engine.Camera.LockCamera(_tileMap);
            }
        }
    }
}
```

T

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

namespace SummonersTale.Forms
{
    public class MapDisplay : Control
    {
        private TileMap _tileMap;
        private readonly int _width;
        private readonly int _height;
        private Tile _selectedTile;
        private GameTime _gameTime;

        public Point Tile { get; set; }
        public bool MouseOver { get; set; }

        public int Width { get { return _width; } }
        public int Height { get { return _height; } }

        public MapDisplay(TileMap map, int width, int height)
        {
            Position = new(0, 20);
            Size = new(width * Engine.TileWidth, height * Engine.TileHeight);
            _tileMap = map;

            _width = width;
            _height = height;
            _tileMap = map;
        }

        public void SetMap(TileMap map)
        {
            _tileMap = map;
            _selectedTile = new(0, 0);
            Engine.Camera.Position = new(0, 0);
        }

        public override void Draw(SpriteBatch spriteBatch)
        {
            if (_tileMap != null)
            {
                _tileMap.Draw(new(), spriteBatch, Engine.Camera, true);

                spriteBatch.Begin();
                spriteBatch.DrawString(ControlManager.SpriteFont, "(" + Tile.X + ", 
" + Tile.Y + ")", Vector2.Zero, Color.Red);
                spriteBatch.End();
            }
        }

        public override void HandleInput()
        {
            Rectangle r = new(Helper.V2P(Position), Helper.V2P(Size));
            MouseOver = r.Contains(Xin.MouseAsPoint);

            Vector2 position = new()
            {
                X = Xin.MouseAsPoint.X + Engine.Camera.Position.X,
                Y = Xin.MouseAsPoint.Y + Engine.Camera.Position.Y,
            };

            Tile = Engine.VectorToCell(position);

            if (MouseOver)
            {
                Vector2 vector2 = new();

                if (new Rectangle(0, 0, 128, 1080).Contains(Xin.MouseAsPoint))
                {
                    vector2.X = -1;
                }

                if (new Rectangle(0, 0, 1280, 128).Contains(Xin.MouseAsPoint))
                {
                    vector2.Y = -1;
                }

                if (new Rectangle(1280 - 128, 0, 128, 
1080).Contains(Xin.MouseAsPoint))
                {
                    vector2.X = 1;
                }
                if (new Rectangle(0, 1080 - 128, 1280,
128).Contains(Xin.MouseAsPoint))
                {
                    vector2.Y = 1;
                }

                if (vector2 != Vector2.Zero)
                    vector2.Normalize();

                Engine.Camera.Position =
                    Engine.Camera.Position + vector2 * 160f *
(float)_gameTime.ElapsedGameTime.TotalSeconds;

                LockCamera();
            }
        }


        private void LockCamera()
        {
            float y = MathHelper.Clamp(Engine.Camera.Position.Y, 0,
_tileMap.HeightInPixels - _height);
            float x = MathHelper.Clamp(Engine.Camera.Position.X, 0,
_tileMap.WidthInPixels - _width);

            Engine.Camera.Position = new(x, y);
        }
        public override void Update(GameTime gameTime)
        {
            _gameTime = gameTime;

            if (!HasFocus) return;

            if (_tileMap != null)
            {
                _tileMap.Update(gameTime);
                HandleInput();
                Engine.Camera.LockCamera(_tileMap);
            }
        }
    }
}
```

This class inherits from the Control class. It has five fields: _tileMap, _width, _height, _selectedTile and _gameTime. The _tileMap is, of course, a tile map that will be rendered on the control. The _width is the width of the control and the _height of the height of the control. Tile is which tile the mouse is over. Finally, _gameTime is how much time has passed since the last frame of the game.

There are four properties. The first is a Point which is the tile the mouse is over. MouseOver is if the mouse is over the control. The Width and Height of the control.

The constructor takes a TileMap as a parameter and the width and height of the map in tiles. It sets the position to the upper left corner of the control. The Size property is set to the Engine tile width times the

width in tiles. Similarly, the height is set to the height in times the height of a tile. The _width and _height properties are set next. Then, the _tileMap field to the map passed in.
There is a method that sets the _tileMap field to the map passed in. It resets the selected tile to the first tile on the map. It also resets the Camera property of the Engine.
If the _tileMap field is not null, I call the Draw method of the TileMap. I then do a call to the Begin method of the SpriteBatch in order to begin rendering. I then draw what tile the mouse is in. Finally, I call the End method to finish rendering.

In the HandleInput method, I get a rectangle that is the size of the control. I set the MouseOver property to the value of the rectangle containing the mouse as a point. I then calculate what tile the mouse is in. That is the position of the mouse plus the position of the camera. I set the Tile property to a position of the camera as a Vector2 converted to a Point. Next, I check if the mouse is over the control. If it is, I create a new Vector2 that will act like the motion vector in the game. I then constructed a rectangle that describes the left edge of the screen, which is 128 pixels wide. That might be a little much. A width of 64 or 32 might work better. Play around with it and see what works for you. If it is, I set the X property of the vector to -1 to move left. I then constructed a rectangle which describes the top of the screen. If the mouse is in that, I set the Y property of the screen to -1. I do the same for the right edge and bottom edge of the screen, setting the X property to 1 and the Y property to 1. I then normalize the vector if it is not the zero vector. I then set the position of the camera to the desired motion times 160 pixels times the total elapsed time since the last frame in seconds. I then call LockCamera to lock the camera to the map.

The LockCamera method works similarly to the LockCamera method of the TileMap class. I probably could have gotten away with using that. I calculate the Y property first. It clamps the Y property of the camera between zero and the height of the map in pixels minus the height of the control. I do the same for the X property of the camera. I then set the position of the camera. I have to do it this way because it is a property and not a field.

In the Update method, I set the _gameTime field to the gameTime argument of the parameter passed in. After that, I check to see if the control has focus or not. If it does not, I exit the method. I then check to see if the _tileMap property is not null. If that is true, I call its Update method. Because I won't be adding this to the control manager, I call the HandleUpdate method. I then call the LockCamera method of the camera passing in the _tileMap field.

The next step is to add the control to the main form. What I ended up doing was removing the controls that I had placed on the map and adding the map display. I no longer want the status bar, so I set the main form's full-screen property to true. Set the code of the MainForm class to the following.

```
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;
using Psilibrary.TileEngine;
using System;
using System.Collections.Generic;
using System.Text;

namespace SummonersTale.Forms
{
    public class MainForm : Form
    {
        private MapDisplay mapDisplay;
        private RenderTarget2D renderTarget2D;

        public MainForm(Game game, Vector2 position, Point size) : base(game, 
position, size)
        {
            Title = "";
            FullScreen = true;
        }

        public override void Initialize()
        {
            Engine.Reset(new(0, 0, 1280, 1080), 32, 32);
            Engine.ViewportRectangle = new(0, 0, 1280, 1080);

            base.Initialize();
        }

        protected override void LoadContent()
        {
            base.LoadContent();
            renderTarget2D = new(GraphicsDevice, 1280, 1080);

            mapDisplay = new(null, 40, 32)
            {
                HasFocus = false,
                Position = new(0, 0)
            };

            TileSheet sheet = new(content.Load<Texture2D>(@"Tiles/Overworld"), 
"test", new(40, 36, 16, 16));
            TileSet set = new(sheet);

            TileLayer ground = new(100, 100, 0, 0);
            TileLayer edge = new(100, 100, -1, -1);
            TileLayer building = new(100, 100, -1, -1);
            TileLayer decore = new(100, 100, -1, -1);

            TileMap tileMap = new(set, ground, edge, building, decore, "test");
            mapDisplay.SetMap(tileMap);
        }

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

            mapDisplay.Update(gameTime);
        }

        public override void Draw(GameTime gameTime)
        {
            base.Draw(gameTime);
            GraphicsDevice.SetRenderTarget(renderTarget2D);

            mapDisplay.Draw(spriteBatch);

            GraphicsDevice.SetRenderTarget(null);

            spriteBatch.Begin();
            spriteBatch.Draw(renderTarget2D, new Rectangle(0, 0, 1280, 1080), 
Color.White);

            Color[] data = new Color[640 * 1080];

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

            Texture2D background = new(GraphicsDevice, 640, 1080);
            background.SetData(data);

            spriteBatch.Draw(background, new Rectangle(1280, 0, 640, 1080), 
Color.White);
            spriteBatch.End();
        }
    }
}
```

The first change is that I removed the button and the text box. In their place are a RenderTarget2D and a MapDisplay control. In the constructor, I set the Title property to an empty string and, as I mentioned, the FullScreen property to true. In the Initialize method, I reset the engine to render a map that is 1280 pixels wide and 1080 pixels high, with tiles 32 pixels by 32 pixels. Next, I set the ViewportRectangle to be the same. In the LoadContent method, I create a render target that is 1280 pixels wide and 1080 pixels high. I then set the mapDisplay without a map and 40 pixels wide and 32 pixels high. I copied the code for creating a map to this class from the GamePlayState class, minus the random tiles. It creates a TileSheet for creating a map using the tile set that was downloaded in a previous tutorial. I then create a TileSet using the TileSheet. I then create the four layers and fill the ground with the zero tile, which is the grass tile. The other tiles are set to empty. I create the TileMap and then call the SetMap method.

In the Update method, after the call to the base class, I call the update method of the map display. In the Draw method, I SetRenderMethod on the GraphicsDevice passing in our render target. I then call the Draw method of the MapDisplay object to render that. I call SetRenderTarget passing in null to reset the render target back to the screen. I then call SpritBatch.Begin to start rendering to the screen. I fill a rectangle that fits the portion of the screen that is 1280 pixels wide and 1080 pixels high. Because we are not rendering to the full screen, there will be a black zone. That's okay, though, because we want to render some controls there. I then create a Texture2D on the fly to draw over the black zone. I probably should create it earlier on, but it works the way it is. After creating the texture, I draw it to the screen on the right side of the screen.

Hmmm, that didn't take as long as I thought it would. So, as I said, I'm going to add a flyout menu. For that, we're going to need some content. Fortunately, pzUH on https://opengameart.org has a great GUI set that will fit our purposes. You can download them from my Google Drive at this link. Download the file and extract the contents to a folder. Double-click the Content.mgcb link in the Content folder of the

SummonersTale project. Right-click the GUI folder, select Add and then Existing Item. Navigate the folder and select all of the items. When prompted, choose the option to copy to the GUI folder selecting the option to move all files.

No, we are going to need a form for the flyout menu. The first step will be to create the form that will slide in and out of the right side of the screen. Right-click the Forms folder in the SummonersTale project, select Add and then Class. Name this class MenuForm. Here is the code for that form.

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

namespace SummonersTale.Forms
{
    public interface IMenuForm
    {
        GameState GameState { get; }
    }
    public class MenuForm : Form, IMenuForm
    {
        private GraphicsDeviceManager graphicsDeviceManager;

        private Button _menuButton;
        private Button _saveButton;
        private Button _exitButton;

        private float _frames;
        private TimeSpan _timer;
        private bool _show;
        private float _offset;

        public GameState GameState => this;

        public MenuForm(Game game, Vector2 position, Point size) : base(game, 
position, size)
        {
        }

        protected override void LoadContent()
        {

            base.LoadContent();
            graphicsDeviceManager = 
Game.Services.GetService<GraphicsDeviceManager>();

            if (Game.Services.GetService<IMenuForm>() == null)
                Game.Services.AddService(typeof(IMenuForm), this);

            Texture2D background = new(GraphicsDevice, 1, 1);
            Color[] data = new Color[1 * 1];

            for (int i = 0; i < data.Length; i++)
```

```
                data[i] = Color.Transparent;

            background.SetData(data);

            Background.Image = background;

            _menuButton = new(content.Load<Texture2D>("GUI/g21688"),
ButtonRole.Menu)
            {
                Text = "",
                Position = new(graphicsDeviceManager.PreferredBackBufferWidth - 84,
32)
            };

            _menuButton.Click += MenuButton_Click;

            _saveButton = new(content.Load<Texture2D>("GUI/g22337"),
ButtonRole.Menu)
            {
                Text = "",
                Position= new(graphicsDeviceManager.PreferredBackBufferWidth, 112)
            };

            _saveButton.Click += SaveButton_Click;

            _exitButton = new(content.Load<Texture2D>("GUI/g22379"),
ButtonRole.Menu)
            {
                Text = "",
                Position = new(graphicsDeviceManager.PreferredBackBufferWidth, 192)
            };

            _exitButton.Click += ExitButton_Click;
        }

        private void ExitButton_Click(object sender, EventArgs e)
        {
            _show = false;
            _timer = TimeSpan.FromSeconds(0);
        }

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

        private void MenuButton_Click(object sender, EventArgs e)
        {
            if (_frames < 10) return;

            _show = false;
            _timer = TimeSpan.FromSeconds(0);
        }

        public override void Update(GameTime gameTime)
        {
            _frames++;
            if (_frames < 5) return;

            _timer += gameTime.ElapsedGameTime;

            if (_timer.TotalSeconds > .25f && _show)
            {
                _timer = TimeSpan.FromSeconds(.25f);
            }

            if (_show)
            {
                _offset = (float)(84 * (_timer.TotalSeconds / .25f));
            }
            else
            {
                _offset = (float)(84 - 84 * (_timer.TotalSeconds / .25f));
            }

            if (_timer.TotalSeconds >= .25f && !_show)
            {
                _timer = TimeSpan.FromSeconds(.25f);
                manager.PopTopMost();
            }

            if (_timer.TotalSeconds >= 0.25f)
            {
                _menuButton.Update(gameTime);
                _saveButton.Update(gameTime);
                _exitButton.Update(gameTime);
            }
            base.Update(gameTime);
        }

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

            _exitButton.Position = 
new(graphicsDeviceManager.PreferredBackBufferWidth - _offset, 
_exitButton.Position.Y);
            _saveButton.Position = 
new(graphicsDeviceManager.PreferredBackBufferWidth - _offset, 
_saveButton.Position.Y);

            SpriteBatch.Begin();

            _menuButton.Draw(SpriteBatch);
            _exitButton.Draw(SpriteBatch);
            _saveButton.Draw(SpriteBatch);

            SpriteBatch.End();
        }

        public override void Show()
        {
            _frames = 0;
            _timer = TimeSpan.Zero;
            _show = true;
            base.Show();
        }
    }
}
```

This implements an interface, IMenuForm. This is like the interfaces that we defined in the game. It is used to retrieve an instance at run time. It is a single property that returns the object as a GameState. The class inherits from Form, so it has access to the form elements. It also has access to the public and protected members of BaseGameState. There is a GraphicsDeviceManager, so we know the size of the window. There are three buttons. The first toggles the flyout menu, closing it. The save button will just throw an exception if clicked. The exit button also toggles the flyout menu. It would be more appropriate if it closed the editor. I didn't want to do that until I implemented a message box that would ask if the user wanted to close. The _frames field counts how many frames have passed since the menu started to fly in. The _timer field counts the amount of time that has passed since the form started to fly in. The _show field indicates if the form is flying in or flying out. The final field, _offset, dictates where the form is drawn. There is a single property that returns the current instance of the form.

We don't want to do anything in the constructor, so it is empty. It has to be included, though, because of the base class. In the LoadContent method, I retrieve the graphics device manager from the instance that was previously registered. I then register the instance of the form if it is null. I then create a one-by-one pixel texture that is transparent to be used as the background. It is because I just want the form, not all of the fluff that goes with it. I then set the Image property of the background.

After creating and setting the background, I start creating the buttons for the flyout menu. The first button is static and is used to open the menu. It is what is clicked to open the flyout menu. The other two buttons are just there to show that the flyout works. They will have a functionality assigned to them in the future. The menu button is positioned at the far right of the screen. I then wire the event handler of the click event. The other two buttons are positioned off the screen. I also wire their event handlers.

To show that the buttons work, I set the _show field to false and _timer back to zero seconds to have the menu slide out of the screen. I don't do anything in the save button's click event hander right now. I don't handle the menu event handler if ten frames haven't passed. If ten frames have passed, I do the same thing as in the exit button's event handler.

In the Update method, I increment the number of frames that have passed since the form has been shown. If it is less than five, I exit the method. I then increment the timer. If the timer is greater than a quarter of a second and the _show field is true I set it back to a quarter of a second. That is because of the way that I handle sliding in. You will see that shortly. If the _show field is true, I set the _offset field to 84 pixels times the number of seconds times the total seconds divided by a quarter of a second. What this does, is have a percentage of 84 pixels, so over time, it creeps left. I do something similar of _show is false. Then, if _show is set to false, and the timer is greater than or equal to a quarter of a second, I set the timer to a quarter of a second and pop the state off the stack using the PopTopMost method. If the timer is greater than or equal to a quarter of a second, I call the Update method of the buttons.

In the draw method, I set the position of the save and exit buttons to the edge off the screen minus the offset property, having them slide in or slide out. I then call SpriteBatch.Begin to start rendering, render the buttons, then call SpriteBatch.End to finish rendering.

In the Show method, I reset the _frames field to 0. I also set the _timer field to zero seconds and _show to true so the menu will slide in.

So far, so good. All that is left is to have the form show up. I did that in the Editor class and the MainForm class. I will start with the Editor class. Replace it with the following.

```
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Psilibrary.TileEngine;
using SummonersTale;
using SummonersTale.Forms;
using SummonersTale.SpriteClasses;
using SummonersTale.StateManagement;
using System;
using System.Xml.Schema;

namespace SummonersTaleEditor
{
    public class Editor : Game
    {
        private readonly GraphicsDeviceManager _graphics;
        private SpriteBatch _spriteBatch;
        private readonly GameStateManager manager;
        private MainForm _mainForm;
        private MenuForm _menuForm;

        private TileMap _tileMap;

        public Editor()
        {
            _graphics = new GraphicsDeviceManager(this)
            {
                PreferredBackBufferWidth = 1920,
                PreferredBackBufferHeight = 1080,
                IsFullScreen = false,
                SynchronizeWithVerticalRetrace = false
            };

            _graphics.ApplyChanges();

            Content.RootDirectory = "Content";
            IsMouseVisible = true;

            Services.AddService(typeof(GraphicsDeviceManager), _graphics);

            manager = new GameStateManager(this);
            Components.Add(manager);
        }
        protected override void Initialize()
        {
            // TODO: Add your initialization logic here
            Components.Add(new FramesPerSecond(this));
            Components.Add(new Xin(this));

            base.Initialize();
        }

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

            // TODO: use this.Content to load your game content here

            Services.AddService(typeof(SpriteBatch), _spriteBatch);

            _mainForm = new(this, Vector2.Zero, new(1920, 1080))
            {
                FullScreen = true,
                Title = "A Summoner's Tale Editor"
            };

            _menuForm = new(this, new(1920 + 80, 0), new(80, 1080))
            {
                FullScreen = true,
                Title = ""
            };

            manager.ChangeState(_mainForm);

            TileSheet sheet = new(Content.Load<Texture2D>(@"Tiles/Overworld"),
"test", new(40, 36, 16, 16));
            TileSet set = new(sheet);

            TileLayer ground = new(100, 100, 0, 0);
            TileLayer edge = new(100, 100, -1, -1);
            TileLayer building = new(100, 100, -1, -1);
            TileLayer decore = new(100, 100, -1, -1);

            _tileMap = new(set, ground, edge, building, decore, "test");
        }

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

            // TODO: Add your update logic here

            base.Update(gameTime);
        }

        protected override void Draw(GameTime gameTime)
        {
            GraphicsDevice.Clear(Color.CornflowerBlue);
            // TODO: Add your drawing code here

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

What has changed here? Well, I added a MenuForm field to hold the form. Then, in the LoadContent method I create the instance, setting the Text property to the empty screen and the FullScreen property to true so that there will not be a title bar.

That leaves the MainForm. What we need to do is add a button to the form that will cause the form to slide in when it is clicked. Replace the MainForm class with the following code.

```
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;
using Psilibrary.TileEngine;
using System;
using System.Collections.Generic;
using System.Text;

namespace SummonersTale.Forms
{
    public class MainForm : Form
    {
        private MapDisplay _mapDisplay;
        private RenderTarget2D _renderTarget2D;
        private Button _menuButton;
        private IMenuForm _menuForm;

        public MainForm(Game game, Vector2 position, Point size) : base(game,
position, size)
        {
            Title = "";
            FullScreen = true;
        }

        public override void Initialize()
        {
            Engine.Reset(new(0, 0, 1280, 1080), 32, 32);
            Engine.ViewportRectangle = new(0, 0, 1280, 1080);

            base.Initialize();
        }

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

            GraphicsDeviceManager gdm =
Game.Services.GetService<GraphicsDeviceManager>();

            _menuButton = new(content.Load<Texture2D>(@"GUI/g21688"),
ButtonRole.Menu)
            {
                Text = "",
                Position = new(gdm.PreferredBackBufferWidth - 84, 20)
            };

            _menuButton.Click += MenuButton_Click;

            _renderTarget2D = new(GraphicsDevice, 1280, 1080);

            _mapDisplay = new(null, 40, 32)
            {
                HasFocus = false,
                Position = new(0, 0)
            };

            TileSheet sheet = new(content.Load<Texture2D>(@"Tiles/Overworld"),
"test", new(40, 36, 16, 16));
            TileSet set = new(sheet);

            TileLayer ground = new(100, 100, 0, 0);
            TileLayer edge = new(100, 100, -1, -1);
            TileLayer building = new(100, 100, -1, -1);
            TileLayer decore = new(100, 100, -1, -1);

            TileMap tileMap = new(set, ground, edge, building, decore, "test");

            _mapDisplay.SetMap(tileMap);

            Color[] data = new Color[1];
            data[0] = Color.Transparent;

            Texture2D b = new(GraphicsDevice, 1, 1);
            b.SetData(data);
        }

        private void MenuButton_Click(object sender, EventArgs e)
        {
            _menuForm = (IMenuForm)Game.Services.GetService<IMenuForm>();

            manager.PushTopMost(_menuForm.GameState);
            Visible = true;
        }

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

            _mapDisplay.Update(gameTime);
            _menuButton.Update(gameTime);
        }

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

            GraphicsDevice.SetRenderTarget(_renderTarget2D);

            _mapDisplay.Draw(spriteBatch);
            GraphicsDevice.SetRenderTarget(null);

            spriteBatch.Begin();
            spriteBatch.Draw(_renderTarget2D, new Rectangle(0, 0, 1280, 1080), 
Color.White);

            Color[] data = new Color[640 * 1080];

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

            Texture2D background = new(GraphicsDevice, 640, 1080);
            background.SetData(data);

            spriteBatch.Draw(background, new Rectangle(1280, 0, 640, 1080), 
Color.White);

            _menuButton.Position = new(1920 - 84, 32);
            _menuButton.Draw(SpriteBatch);

            spriteBatch.End();
        }
    }
}
```

What has changed here? Well, there is a new Button field that is the button that will trigger the menu flying in. There is also a field for the MenuForm's interface. In the LoadContent method, I retrieve the GraphicsDeviceManager that was added to the services. I then create the menu button positioning it in the upper right-hand corner of the screen. I wire the click event handler of the button.

In the MenuButton_Click event, I grab the instance of the menu form that was added to the list of components of the game. I then use the PushTopMost method to place it on top of the stack. I don't want the main form to update, but I do what it to be visible. For that reason, I set the Visible property to true.

After drawing the map and texture for where the controls for the main form will be rendered, I position the button and call its draw method.

If you build and run now, you will be presented with the editor with a button in the upper right-hand corner of the window. If you click the button, the menu will slide in from the right-hand side of the screen.

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