A Summoner's Tale
Chapter 13
Codex

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.

Before I dive into the main tutorial, I want to add three extension methods. They are all in the SpriteBatch class. They add the ability to draw using Point instead of Vector2. It is just something that comes up frequently. Add these three methods to the ExtensionMethods class in the SummonersTale project.

```
        public static void Draw(this SpriteBatch spriteBatch, Texture2D texture,
Point destination, Rectangle? source, Color color)
        {
            spriteBatch.Draw(texture, new Vector2(destination.X, destination.Y),
source, color);
        }

        public static void Draw(this SpriteBatch spriteBatch, Texture2D texture,
Point destination, Color color)
        {
            spriteBatch.Draw(texture, new Vector2(destination.X, destination.Y),
color);
        }

        public static void DrawString(this SpriteBatch spriteBatch, SpriteFont font,
string text, Point destination, Color tint)
        {
            spriteBatch.DrawString(font, text, new Vector2(destination.X,
destination.Y), tint);
        }
```

The main point of this tutorial is creating shadow monsters. To do that, I added a new project to the solution. In the spirit of building cross-platform games, I added a new Desktop GL project. So, right-click the SummonersTale solution, select Add and then New Project. From the list that comes up, select the Desktop GL option and click Next. Name this new project MonsterEditor then click Create. This new project needs to know about the other projects. So, right-click the dependencies node, and select Add Project Reference. Select the Psilibrary project. Expand the Shared Project node on the right side of the screen, and select SharedProject to reference the projects.

Before I add the forms, I need to add a new control, a check box. We will use the check box to capture the bool properties of objects. Right-click the Controls folder inside SharedProjects, select Add and then Class. Name the new class CheckBox. 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.Text;

namespace SummonersTale.Forms
{
    internal class Checkbox : Control
    {
        public bool Checked { get; set; } = true;
        private Dictionary<bool, Texture2D> textures = new Dictionary<bool,
Texture2D>();
        public int Width { get { return textures[Checked].Width; } }
        public int Height { get { return textures[Checked].Height; } }
        private double timer = 0;

        public Checkbox(Texture2D selected, Texture2D unselected)
        {
            textures.Add(true, selected);
            textures.Add(false, unselected);
        }

        public override void Draw(SpriteBatch spriteBatch)
        {
            Rectangle destination = new(
            (int)Position.X,
            (int)Position.Y,
            (int)Size.X,
            (int)Size.Y);

            spriteBatch.Draw(textures[Checked], destination, Color.White);

            _spriteFont = ControlManager.SpriteFont;

            Vector2 size = _spriteFont.MeasureString(Text);
            Vector2 offset = new((Size.X + 10), ((Size.Y - size.Y) / 2));

            spriteBatch.DrawString(_spriteFont, Text, Helper.NearestInt((Position +
offset)), Color);
        }

        public override void HandleInput()
        {
            if (timer < .25) return;

            Point mouse = Xin.MouseAsPoint;
            Rectangle destination = new(
                (int)Position.X,
                (int)Position.Y,
                (int)Size.X,
                (int)Size.Y);

            _mouseOver = destination.Contains(mouse);

            if (HasFocus && Xin.WasKeyReleased(Keys.Space))
            {
                Checked = !Checked;
                timer = 0;
            }
            else if (_mouseOver && Xin.WasMouseReleased(MouseButton.Left))
            {
                Checked = !Checked;
                timer = 0;
            }
        }

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

There is a public property, Checked, that defines the state of the check box. There is a Dictionary<bool, Texture2D> that holds the textures for checked and unchecked controls. We need to know the height and width, so there are properties to expose those values. We need to keep track of how long has occurred since the check box was last clicked. Without this the selection stutters.

There is a single constructor that takes as parameters the selected and unselected images, in that order. It then adds them to the dictionary of textures for the check box.

The Draw method creates a rectangle that defines the destination of the check box, then renders the appropriate image based on the Checked property. It then gets the font from the control manager and uses it to measure the text of the check box. It then positions the text ten pixels to the right of the box and centers it vertically between the height of the box. It then draws the text.

The HandleInput method checks to see if the timer is less than a quarter of a second. If it is, it exits the method. Next, it grabs the position of the mouse as a point. Following that, it creates a destination rectangle around the box and then checks if the mouse is over the box. If the control has focus and the spacebar was released, Checked is set to not Checked to invert the selection, and the timer is set to zero. Otherwise, it checks to see if the mouse is over and if the left mouse button was released. If that is the case, it does the same as above. The Update method increments the timer by the amount of time that has passed since the last update. It then calls the HandleInput method.

There are some rendering errors happening during the form that displays the controls for creating moves. The easiest solution was to change the way Begin and End are called. Instead of calling Begin and End from the form, I moved the call inside the control manager. First, update the Draw method of the Form class to the following.

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

    if (!Visible) return;

    if (!FullScreen)
    {
        Vector2 size = ControlManager.SpriteFont.MeasureString(Title);
        Vector2 position = new((Bounds.Width - size.X) / 2, 0);
        Label label = new()
        {
            Text = _title,
            Color = Color.White,
            Position = position
        };

        Matrix m = Matrix.Identity;

        SpriteBatch.Begin(SpriteSortMode.BackToFront, BlendState.AlphaBlend,
SamplerState.AnisotropicWrap, null, null, null, m);

        Background.Draw(SpriteBatch);
        TitleBar.Draw(SpriteBatch);

        CloseButton.Draw(SpriteBatch);

        SpriteBatch.End();

        SpriteBatch.Begin();

        label.Position = Helper.NearestInt(position + Position);
        label.Color = Color.White;
        label.Draw(SpriteBatch);

        SpriteBatch.End();

        //m = Matrix.CreateTranslation(new Vector3(0, 20, 0) + new Vector3(Position,
0));

        _controls.Draw(SpriteBatch);
    }
    else
    {
        Background.DestinationRectangle = new(
            0,
            0,
            _graphicsDevice.PreferredBackBufferWidth,
            _graphicsDevice.PreferredBackBufferHeight);
        SpriteBatch.Begin();
        Background.Draw(SpriteBatch);
        SpriteBatch.End();

        _controls.Draw(SpriteBatch);
    }
}
```

While you have the Form class open, I need to make a change to the Background property. I need the set to be protected instead of private. Replace the Background property of the Form class with the following.

```
public PictureBox Background { get; protected set; }
```

Now, for the ControlManager. All I did was when it comes to rendering, is render each control within its own call to begin and end. True, this is inefficient, but without doing it, there was some strange behaviour. Replace the Draw method of the ControlManager with the following.

```
public void Draw(SpriteBatch spriteBatch)
{
    foreach (Control c in this)
    {
        if (c.Visible)
        {
            spriteBatch.Begin(SpriteSortMode.BackToFront, BlendState.AlphaBlend, 
SamplerState.AnisotropicWrap, null, null, null, null);

            c.Draw(spriteBatch);

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

I made a few adjustments to the Textbox class. What I did was set the Size property to the size passed in. I also adjusted the caret so that it would fit inside the text box. Replace the constructor with this new version.

```
public Textbox(GraphicsDevice graphicsDevice, Vector2 size)
    : base()
{

    _text = "";

    Size = size;

    _background = new Texture2D(graphicsDevice, (int)size.X, (int)size.Y);
    _background.Fill(Color.White);

    _border = new Texture2D(graphicsDevice, (int)size.X, (int)size.Y);
    _border.Fill(Color.Black);
    _caret = new Texture2D(graphicsDevice, 2, (int)size.Y - 8);
    _caret.Fill(Color.Black);

    _tint = Color.Black;
    foreach (char c in
"abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTWYYXZ0123456789 -_".ToCharArray())
    {
        validChars.Add(c.ToString());
    }
}
```

I also updated the Draw method. I changed the way the caret is drawn and the draw order for the items. The other thing that I did was not draw the caret if the control does not have focus. Replace the Draw method of the Textbox class with the following version.

```
public override void Draw(SpriteBatch spriteBatch)
{
    Vector2 dimensions = ControlManager.SpriteFont.MeasureString(_text);
    dimensions.Y = 0;

    Rectangle location = new(
        new((int)Position.X, (int)Position.Y),
        new((int)Size.X, (int)Size.Y));

    spriteBatch.Draw(_border, location.Grow(1), null, Color.White, 0f, Vector2.Zero,
SpriteEffects.None, 1f);
    spriteBatch.Draw(_background, location, null, Color.White, 0f, Vector2.Zero,
SpriteEffects.None, 1f);

    spriteBatch.DrawString(
        ControlManager.SpriteFont,
        Text,
        Helper.NearestInt(Position + Vector2.One * 5),
        Color.Black,
        0,
        Vector2.Zero,
        1f,
        SpriteEffects.None,
        0f);
    if (!HasFocus) { return; }
    spriteBatch.Draw(
        _caret,
        Position + dimensions + Vector2.One * 5,
        _tint);
}
```

The last change I made to controls was the RightLeftSelector. What I did was in the Draw method is scale the buttons based on the Size property. Replace the Draw method of the RightLeftSelector with this new version.

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

    _spriteFont = ControlManager.SpriteFont;
    _yOffset = (int)((Size.Y - _spriteFont.MeasureString("W").Y) / 2);
    _leftSide = new Rectangle(
        (int)_position.X,
        (int)_position.Y,
        (int)_size.X,
        (int)_size.Y);

    //if (selectedItem != 0)
    spriteBatch.Draw(_leftTexture, _leftSide, Color.White);
    //else
    //    spriteBatch.Draw(stopTexture, drawTo, Color.White);

    drawTo.X += Size.X + 5f;

    float itemWidth = _spriteFont.MeasureString(_items[_selectedItem]).X;
    float offset = (_maxItemWidth - Size.X) / 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,
        (int)_size.X,
        (int)_size.Y);

    //if (selectedItem != items.Count - 1)
    spriteBatch.Draw(_rightTexture, _rightSide, Color.White);
    //else
    //    spriteBatch.Draw(stopTexture, drawTo, Color.White);
}
```

Okay, those are the last of the changes to controls. Now we can start work on the forms. First, right-click the MonsterEditor project, select Add and then Class. Name this new class MainForm. This is the code for that class.

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

namespace MonsterEditor.Forms
{
    public class MainForm : Form
    {
        private ListBox _movesList;
        private ListBox _monstersList;
        private Button _addMoveButton;
        private Button _editMoveButton;
        private Button _deleteMoveButton;
        private Button _addMonsterButton;
        private Button _editMonsterButton;
        private Button _deleteMonsterButton;

        public MainForm(Game game, Vector2 position, Point size) : base(game, 
position, size)
        {
            Position = new(0, 0);

            Size = new(_graphicsDevice.PreferredBackBufferWidth, 
_graphicsDevice.PreferredBackBufferHeight);
            Bounds = new(0, 0, _graphicsDevice.PreferredBackBufferWidth, 
_graphicsDevice.PreferredBackBufferHeight);
            FullScreen = true;

            Texture2D texture2D = new(GraphicsDevice, 100, 100);
            texture2D.Fill(new Color(240, 240, 240));

            PictureBox bg = new(GraphicsDevice, texture2D, Bounds);
            Background = bg;
        }

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

            Label movesLabel = new()
            {
                Text = "Moves",
                Color = Color.Black
            };

            Controls.Add(movesLabel);

            _movesList = new(GraphicsDevice,
                Game.Content.Load<Texture2D>(@"GUI/DownButton"),
                Game.Content.Load<Texture2D>(@"GUI/UpButton"),
                new(800, 1000))
            {
                Position = new(0,50)
            };

            Controls.Add(_movesList);

            _addMoveButton = new(
                Game.Content.Load<Texture2D>(@"GUI/Button"),
                ButtonRole.Menu)
            {
                Color = Color.Black,
                Text = "Add",
                Position = new(810, 50),
            };
```

```
Controls.Add(_addMoveButton);

_editMoveButton = new(
    Game.Content.Load<Texture2D>(@"GUI/Button"),
    ButtonRole.Menu)
{
    Color = Color.Black,
    Text = "Edit",
    Position = new(810, 100),
};

Controls.Add(_editMoveButton);

_deleteMoveButton = new(
    Game.Content.Load<Texture2D>(@"GUI/Button"),
    ButtonRole.Menu)
{
    Color = Color.Black,
    Text = "Remove",
    Position = new(810, 150),
};

Controls.Add(_deleteMoveButton);

Label monstersLabel = new()
{
    Text = "Monsters",
    Color = Color.Black,
    Position = new(930, 0)
};

Controls.Add(monstersLabel);

_monstersList = new(GraphicsDevice,
    Game.Content.Load<Texture2D>(@"GUI/DownButton"),
    Game.Content.Load<Texture2D>(@"GUI/UpButton"),
    new(800, 1000))
{
    Position = new(930, 50)
};

Controls.Add(_monstersList);

_addMonsterButton = new(
    Game.Content.Load<Texture2D>(@"GUI/Button"),
    ButtonRole.Menu)
{
    Color = Color.Black,
    Text = "Add",
    Position = new(1750, 50),
};

Controls.Add(_addMonsterButton);

_editMonsterButton = new(
    Game.Content.Load<Texture2D>(@"GUI/Button"),
    ButtonRole.Menu)
```

```
            {
                Color = Color.Black,
                Text = "Edit",
                Position = new(1750, 100),
            };

            Controls.Add(_editMonsterButton);

            _deleteMonsterButton = new(
                Game.Content.Load<Texture2D>(@"GUI/Button"),
                ButtonRole.Menu)
            {
                Color = Color.Black,
                Text = "Remove",
                Position = new(1750, 150),
            };

            Controls.Add(_deleteMonsterButton);

            _addMoveButton.Click += AddMoveButton_Click;
            _editMoveButton.Click += EditMoveButton_Click;
            _deleteMoveButton.Click += DeleteMoveButton_Click;

            _addMonsterButton.Click += AddMonsterButton_Click;
            _editMonsterButton.Click += EditMonsterButton_Click;
            _deleteMonsterButton.Click += DeleteMonsterButton_Click;
        }

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

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

        private void AddMoveButton_Click(object sender, EventArgs e)
        {
            MoveForm frm = new(Game, new((1920 - 600) / 2, (1000 - 800) / 2), 
new(600, 800));
            manager.PushState(frm);
            this.Visible = true;
        }

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

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

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

        public override void Update(GameTime gameTime)
        {
            foreach (Control c in Controls)
            {
                c.Update(gameTime);
            }

            base.Update(gameTime);
        }

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

There are a number of fields. There are two list boxes. One will hold the moves the other will hold the shadow monsters. There are buttons to add, edit, and delete monsters and moves. The constructor sets the position to (0, 0). It then sets the size and bounds to fill the screen. The fullscreen property is set to true. It then creates the background for the form. It is a very light grey. It then creates the picture box for the background and sets it as the background.

The LoadContent method creates the controls and sets their properties. I place labels above each of the list boxes to show what type of data they hold. To the right of each of the list boxes are buttons to add, edit, and delete items. I'm not going to go over the creation of the controls other than to say I wire event handlers for each of the buttons.

The only event handler that I've wired so far is the AddMove button. What it does is create a new instance of the form that I will add next. It pushes it on top of the stack and keeps this form visible.

The Update method iterates over all of the controls and calls their update method. It also calls base.Update. The Draw method just calls the base.Draw method to render the form.

I want to be able to use colons in text boxes. Unfortunately, a colon is a shifted semicolon. Semicolons are represented by OemSemicolon in the key enumeration. So, I did make the modifications to the HandleInput method to be able to input semicolons and colons. Replace the HandleInput method of the Textbox class with this new version.

```
public override void HandleInput()
{

    if (!HasFocus)
    {
        return;
    }
    List<Keys> keys = Xin.KeysPressed();

    foreach (Keys key in keys)
    {
        string value = Enum.GetName(typeof(Keys), key);
        if (value == "Back" && _text.Length > 0)
        {
            _text = _text.Substring(0, _text.Length - 1);
            return;
        }
        else if (value == "Back")
        {
            Text = "";
            return;
        }

        if (value.Length == 2 && value.Substring(0,1) == "D")
        {
            value = value.Substring(1);
        }

        if (!Xin.IsKeyDown(Keys.LeftShift) && !Xin.IsKeyDown(Keys.RightShift) &&
!Xin.KeyboardState.CapsLock)
        {
            value = value.ToLower();
        }

        if (validChars.Contains(value))
        {
            if (ControlManager.SpriteFont.MeasureString(_text + value).X < Size.X)
                _text += value;
        }

        if (key == Keys.OemSemicolon && !(Xin.IsKeyDown(Keys.RightShift) &&
Xin.IsKeyDown(Keys.LeftShift)))
        {
            _text += ';';
        }
        else if (key == Keys.OemSemicolon && (Xin.IsKeyDown(Keys.RightShift) ||
Xin.IsKeyDown(Keys.LeftShift)))
            {
            _text += ":";
        }
    }
}
```

What the new code does is check to see if the OemSemicolon key is pressed and if the shift keys are pressed or unpressed. If they are unpressed, I append a semicolon to the text. Otherwise, I append a colon to the text.

That brings us to the move form. Right-click the Forms folder in the MonsterEditor project, select Add and then Form. Name this new form MoveForm. This is the code for that form.

```
using Assimp.Configs;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;
using Psilibrary.ShadowMonsters;
using ShadowMonsters.Controls;
using SummonersTale;
using SummonersTale.Forms;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace MonsterEditor.Forms
{
    public class MoveForm : Form
    {
        public event EventHandler Closing;

        public MoveData Move { get; set; }
        internal Button Okay { get; set; }
        internal Button Cancel { get; set; }
        internal Textbox Name { get; set; }
        internal Label NameLabel { get; set; }

        internal Textbox Elements { get; set; }
        internal Label ElementsLabel { get; set; }

        internal Textbox Level { get; set; }
        internal Label LevelLabel { get; set; }

        internal RightLeftSelector Target { get; set; }
        internal Label TargetLabel { get; set; }

        internal RightLeftSelector TargetAttribute { get; set; }
        internal Label AttributeLabel { get; set; }

        internal Textbox Mana { get; set; }
        internal Label ManaLabel { get; set; }
        internal Textbox Range { get; set; }
        internal Label RangeLabel { get; set; }
        internal RightLeftSelector Status { get; set; }
        internal Label StatusLabel { get; set; }
        internal Textbox Power { get; set; }
        internal Label PowerLabel { get; set; }
        internal Checkbox Hurts { get; set; }
        internal Checkbox IsTemporary { get; set; }

        public MoveForm(Game game, Vector2 position, Point size) : base(game,
position, size)
        {
            FullScreen = false;
            Size = new(600, 580);
            Position = position;

            Texture2D texture2D = new(GraphicsDevice, 100, 100);
            texture2D.Fill(new Color(240, 240, 240));
            PictureBox bg = new(GraphicsDevice, texture2D, Bounds);
            Background = bg;
        }

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

            Okay = new(
                content.Load<Texture2D>(@"GUI/button"),
                ButtonRole.Accept)
            {
                Position = new(this.Bounds.Right - 125, 30),
                Color = Color.Black,
                Text = "OK"
            };

            Okay.Click += Okay_Click;
            Controls.Add(Okay);

            Cancel = new(
                content.Load<Texture2D>(@"GUI/button"),
                ButtonRole.Cancel)
            {
                Position = new(this.Bounds.Right - 125, 80),
                Color = Color.Black,
                Text = "Cancel"
            };

            Cancel.Click += Cancel_Click;
            Controls.Add(Cancel);

            NameLabel = new()
            {
                Position = new(10, 30),
                Color = Color.Black,
                Text = "Name:"
            };

            Controls.Add(NameLabel);

            Name = new(GraphicsDevice, new(300, 30))
            {
                Text = "",
                Position = new(125, 30),
                Color = Color.White,
            };

            Controls.Add(Name);

            ElementsLabel = new()
            {
                Position = new(10, 80),
                Color = Color.Black ,
                Text = "Elements:"
            };

            Controls.Add(ElementsLabel);
```

```
Elements = new(GraphicsDevice, new(300, 30))
{
    Text = "",
    Position = new(125, 80),
    Color = Color.White,
};

Controls.Add(Elements);

LevelLabel = new()
{
    Position = new(10, 130),
    Color = Color.Black,
    Text = "Level:"
};

Controls.Add(LevelLabel);

Level= new(GraphicsDevice, new(300, 30))
{
    Text = "",
    Position = new(125, 130),
    Color = Color.White,
};

Controls.Add(Level);

Target = new(
    content.Load<Texture2D>(@"GUI\g22987"),
    content.Load<Texture2D>(@"GUI\g21245"))
{
    Text = "",
    Position = new(125, 180),
    Color = Color.Black,
    Size = new(32, 32),
};

string[] targets = Enum.GetNames(typeof(TargetType));

Target.SetItems(targets, 200);
Target.SelectedIndex = 1;

Controls.Add(Target);

TargetLabel = new()
{
    Position = new(10, 180),
    Color = Color.Black,
    Text = "Target:"
};

Controls.Add(TargetLabel);

AttributeLabel = new()
{
    Position = new(10, 230),
    Color = Color.Black,
                Text = "Attribute:"
            };

            Controls.Add(AttributeLabel);

            TargetAttribute = new(
                content.Load<Texture2D>(@"GUI\g22987"),
                content.Load<Texture2D>(@"GUI\g21245"))
            {
                Position = new(125, 230),
                MaxItemWidth = 100,
                Size = new(32, 32),
                Color = Color.Black,
            };

            TargetAttribute.SetItems(new string[] { "Health", "Attack", "Defense",
"SpecialAttack", "SpecialDefence", "Speed", "Accuracy" }, 200);
            Controls.Add(TargetAttribute);

            ManaLabel = new()
            {
                Position = new(10, 280),
                Color = Color.Black,
                Text = "Mana:"
            };

            Mana = new(GraphicsDevice, new(200, 30))
            {
                Text = "",
                Position = new(125, 280),
                Color = new Color(255, 255, 255),
                Size = new(300, 30)
            };
            Controls.Add(ManaLabel);
            Controls.Add(Mana);

            RangeLabel = new()
            {
                Position = new(10, 330),
                Color = Color.Black,
                Text = "Range:"
            };

            Range = new(GraphicsDevice, new(200, 30))
            {
                Text = "",
                Position = new(125, 330),
                Color = new Color(2553, 255, 255),
                Size = new(300, 30)
            };

            Controls.Add(RangeLabel);
            Controls.Add(Range);

            PowerLabel = new()
            {
                Position = new(10, 380),
                Color = Color.Black,
                Text = "Power:"
            };

            Power = new(GraphicsDevice, new(300, 30))
            {
                Text = "",
                Position = new(125, 380),
                Color = new Color(2553, 255, 255),
                Size = new(300, 30)
            };

            Controls.Add(PowerLabel);
            Controls.Add(Power);

            Status = new(
                content.Load<Texture2D>(@"GUI\g22987"),
                content.Load<Texture2D>(@"GUI\g21245"))
            {
                Text = "",
                Position = new(125, 430),
                Color = Color.Black,
                Size = new(32, 32),
            };

            Status.SetItems(new string[] { "Normal", "Sleep", "Confused",
"Poisoned", "Paralysis", "Burned", "Frozen" }, 200);
            Controls.Add(Status);

            StatusLabel = new()
            {
                Position = new(10, 430),
                Color = Color.Black,
                Text = "Status:"
            };

            Controls.Add(StatusLabel);

            Hurts = new(content.Load<Texture2D>(@"GUI/selected"),
content.Load<Texture2D>(@"GUI/unselected"))
            {
                Size = new(32, 32),
                Position = new(10, 480),
                Text = "Hurts?",
                Color = Color.Black,
                Enabled = true,
                Visible = true,
            };

            Controls.Add(Hurts);

            IsTemporary = new(content.Load<Texture2D>(@"GUI/selected"),
content.Load<Texture2D>(@"GUI/unselected"))
            {
                Size = new(32, 32),
                Position = new(10, 530),
                Text = "Is Temporary?",
                Color = Color.Black,
                Enabled = true,
                Visible = true,
                Checked = false
            };

            Controls.Add(IsTemporary);
        }

        private void Okay_Click(object sender, EventArgs e)
        {
            bool error = false;

            if (string.IsNullOrEmpty(Name.Text))
            {
                error = true;
                Name.Text = "Please enter a name.";
                Name.Color = Color.Red;
            }

            if (!int.TryParse(Elements.Text, out int elements))
            {
                error = true;
                Elements.Text = "Please select elements.";
                Elements.Color = Color.Red;
            }

            if (!int.TryParse(Level.Text, out int level))
            {
                error = true;
                Level.Text = "Please select level.";
                Level.Color = Color.Red;
            }

            if (!int.TryParse(Power.Text, out int power))
            {
                error = true;
                Power.Text = "Please select power.";
                Power.Color = Color.Red;
            }

            if (!int.TryParse(Mana.Text, out int mana))
            {
                error = true;
                Mana.Text = "Please select mana.";
                Mana.Color = Color.Red;
            }

            string[] range = Range.Text.Split(':');
            if (range.Length < 2)
            {
                error = true;
                Range.Text = "Please enter colon separated.";
            }
            if (!int.TryParse(range[0], out int x))
            {
                error = true;
                Range.Text = "Range must be numeric.";
            }
            int y = 0;

            if (range.Length == 2 && !int.TryParse(range[1], out y))
            {
                error = true;
                Range.Text = "Range must be numeric.";
            }

            if (error)
            {
                return;
            }

            Move = new()
            {
                Name = Name.Text,
                Elements = elements,
                Level = level,
                Target = Enum.Parse<TargetType>(Target.SelectedItem),
                Power = power,
                Mana = new(mana, mana),
                Range = new(x, y),
                Status = Status.SelectedIndex,
                TargetAttribute =
Enum.Parse<TargetAttribute>(TargetAttribute.SelectedItem),
                Hurts = Hurts.Checked,
                IsTemporary = IsTemporary.Checked,
            };

            manager.PopState();
            OnClosing();
        }

        private void Cancel_Click(object sender, EventArgs e)
        {
            Move = null;
            manager.PopState();
            OnClosing();
        }

        private void OnClosing()
        {
            Closing?.Invoke(this, EventArgs.Empty);
        }

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

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

Sorry for the wall of code. I don't know of a better way to present it. It is mostly creating and placing controls on the form. There is an event that will fire when the form is closed. It gives us a chance to create a MoveData object that represents the data that you have entered. There is a property to expose the move.

There are many controls in the class. There are buttons for OK and Cancel. There are labels for the controls, other than the checkboxes, as they have text properties. Most of the properties have text boxes. A few have right-left selectors. Those are the target, target attribute and status. I probably should have chosen a right-left selector for the element. Or another control as the element is a composite element.

The constructor sets the FullScreen property to false. The size is set to (600, 580). I create a texture for the background of the form and fill it with an off-white. I then assign the background of a picture box to the texture, passing the bounds of the form. After creating the picture box, I assign it as the background of the form.

In the LoadContent method, I create all of the control, position them, and, where appropriate, wire their event handlers. I'm not going to go over each individual control. That would be tedious form, and painful for you. For buttons, I create them using the button texture from a previous tutorial. I position on the right edge of the form and just down from the top. I wire event handlers for their Click events. All of the labels are left aligned at ten pixels. The text boxes are all the same dimensions, 300 pixels by 30 pixels. Their left sides are at 125 pixels. The right-left selectors use the buttons form the GUI library that we downloaded earlier in the series. The interesting thing about the is setting their items. Where possible, I used the GetNames method of the Enum class. They're not completely centred, I leave that to you. For the check boxes, I default the Hurts property to true and the IsTemporary property to false. That is all I'm going to say about creating the controls.

In the Okay_Click method, the handler for the click event of the OK button, there is a local variable, error, that determines if there is a validation error. If the Text property of the Name text box is null or empty, I set the error variable to true, set the text property to an error message, and the text colour to red. I probably should create an error provider or display a message box here. I do something similar for the other text boxes. If error is true after all the validations, I exit the method. The only different one is the range. We want a minimum and maximum separated by a colon. So, it first checks to see if the Text property has a colon in it. If not, the error variable is set to true and a message is displayed. Next, it checks to see if the first part is numeric. If not, the same procedure is done with the error variable and the message. For the next one, I need to set the y variable before using TryParse. That is because if there are not two values, using out int y won't be triggered. If there was an error, I exit the method. Otherwise, I create a new MoveData and assign it to the Move property. The only interesting parts are the enumerations. They need to be parsed to strings. I guess Mana and Range are different as well. Mana and Range are Point properties. Mana is the maximum number of times a move can be used and the currently used value. After creating the move data, I pop the state off the stack and call the OnClosing method.

The Cancel_Click method is trivial. It sets the move to null, pops the state off the stack, and calls the OnClosing method. Similarly, the OnClosing method is just as simple. It invokes the event if it is subscribed to. The Update method calls base.Update to have the controls updated, and the Draw method calls base.Draw to have the controls rendered.

Before we go back to the main form, I need to add a manager class that will hold all of our moves and monsters. In the PsiLibrary project, right-click the ShadowMonsters folder, select Add and then Class. Name this new class ShadowMonsterManager. Here is the code for that class.

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

namespace Psilibrary.ShadowMonsters
{
    public class ShadowMonsterManager
    {
        public readonly Dictionary<string, MoveData> Moves = new();
        public readonly Dictionary<string, ShadowMonsterData> Monsters = new();
    }
}
```

Nothing special here. Just two dictionaries, one for moves and one for monsters. We will serialize this class and deserialize it to save and load monsters.

Okay, back to the main form. We need to wire a handler for the close event of the move form and handle the event. To do that, change the AddMoveButton_Click method to the following and add this new method.

```
private void AddMoveButton_Click(object sender, EventArgs e)
{
    MoveForm frm = new(Game, new((1920 - 600) / 2, (1000 - 800) / 2), new(600,
800));
    manager.PushState(frm);
    this.Visible = true;
    frm.Closing += Frm_Closing;
}

private void Frm_Closing(object sender, EventArgs e)
{
    if (sender is MoveForm form && form.Move != null)
    {
        shadowMonsterManager.Moves.Add(form.Move.Name, form.Move);
        _movesList.Items.Add(form.Move.Name);
    }
}
```

In the AddMoveButton_Click method, all I do is wire the event handler for the close event. In the Frm_Closing method, I do something a little more interesting. If the sender parameter is a MoveForm I cast the sender as a MoveForm, and if the Move property of the form is not null, I add the move to the move manager, which I will add in just a moment. I also add the name to the list of moves. It would probably be a good idea to check that the name does not exist before adding it to the dictionary, or it could blow up.

The last piece is to add a field for a ShadowMonsterManager. Add this field with the other field.

```
private readonly ShadowMonsterManager shadowMonsterManager = new();
```

I am going to end this tutorial here. The monster form is going to be double the move form and I don't want to add that now. So, you can build and run now. You can create new moves and add them to the list. I will tackle monsters in the next tutorial, as I'm not going to dive any further in this tutorial. I think I've fed you more than enough for one day. I encourage you to keep visiting my blog for the latest news on my tutorials.

Good luck with your game programming adventures.
Cynthia