Eyes of the Dragon Tutorials 4.0

Part 12

Mechanical Engineering, Well Mechanics

I'm writing these tutorials for the MonoGame 3.8.1 framework using Visual Studio 2022. The tutorials will make more sense if they are read in order. You can find the list of tutorials on the Eyes of the Dragon 4.0 page of my web blog. I will be making the project available on GitHub here. It will be included on the page that links to the tutorials.

This is part twelve of a series of tutorials I plan to write on creating a role-playing game with MonoGame. I've worked on similar tutorials using XNA and the past version of MonoGame. In the process of writing more tutorials, I discovered better ways of doing some things and had to go back to fix things. I'm hoping in this series to avoid making those same mistakes. Also, I am going to make the game cross-platform. In my previous tutorials, they were focused on Windows only. I want to open things up for macOS, Linux, Android, iOS, and any platform MonoGame supports.

My plan for this tutorial was to start on the combat engine. Instead, I will be working on some of the mechanics of the game. I will be implementing the SPECIAL system. It was used in a game I really enjoyed Lionheart: Legacy of the Crusader. It is also used in the Fallout series of games. You can find out more about the system here: http://specialrpg.wikidot.com/d10-special.

To begin, we are going to create an interface that defines the attributes of a character. Right-click the RpgLibrary project, select Add and then New Folder. Name this new folder Characters. Right-click the Characters folder, select Add and then Interface. Name this new interface ICharacter. Here is the code for that interface.

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

namespace RpgLibrary.Characters
{
    public struct AttributePair
    {
        public int Current;
        public int Maximum;

        public AttributePair()
        {
            Current = 10;
            Maximum = 10;
        }
        public AttributePair(int maximum)
        {
            Current = Maximum = maximum;
        }

        public void Adjust(int amount)
        {
            Current += amount;

            if (Current > Maximum)
            {
                Current = Maximum;
            }
        }
    }

    public interface ICharacter
    {
        string Name { get; }

        int Stength { get; set; }
        int Perception { get; set; }
        int Endurance { get; set; }
        int Charisma { get; set; }
        int Intellect { get; set; }
        int Agility { get; set; }
        int Luck { get; set; }

        AttributePair Health { get; set; }
        AttributePair Mana { get; set; }

        int Gold { get; set; }
        int Experience { get; set; }

        bool Enabled { get; set; }
        bool Visible { get; set; }
        Vector2 Position { get; set; }
        Point Tile { get; set; }
        void Update(GameTime gameTime);
        void Draw(SpriteBatch spriteBatch);
    }
}
```

So, I added a struct to the file, AttributePair. It is meant for things like health and mana, where you have a current value and a maximum value. I provided the required default constructor that initializes the values to ten. I also included a constructor that takes the maximum value of the attribute. It also initializes the fields. The last thing that I included in the struct is a method that changes the current value of the pair. It checks to make sure that the adjustment will not place it past the maximum value.

There are a number of properties that a class implementing this interface must implement. The first is a get-only property for the name of the character. When I say character, I mean any game entity. Next, there are sever read and write accessors. One for each of the attributes: Strength, Perception, Endurance, Charisma, Intellect, Agility and Luck. Strength reflects the character's ability to lift heavy objects, wear heavy armour, inflict damage and more. Perception represents how in tune the character is with their environment. Where Strength measures how much, endurance measures how long a character can do physical tasks. Charisma reflects how well a character interacts with other members of the world. As you might imagine, Intellect reflects book smarts and how well the character can perform magic. It also affects the character's mana pool. Agility measures all aspects of dexterity, such as dodging attacks and manipulating mechanical devices. Finally, Luck does not directly influence the world around the character. Instead, it influences checks against other attributes.

There are two attribute pair properties. One for health and another for mana. They are both read and write properties. Health is calculated by multiplying strength and endurance. Mana is calculated by multiplying intellect and perception. That is for a base level one entity. As an entity levels up, they gain greater health and mana.

Next are properties for the gold and experience the character has. There are properties for if the character is enabled and visible. The next two properties are the position of the character in pixels and the position of the character in tiles.

Finally, there are two methods that must be implemented: Update and Draw. They are the same as other entities. They determine if the entity should be updated and if the entity should be rendered.

Now, I'm going to revisit the flow of the game. Instead of going straight from the title screen to the gameplay screen, I'm going to add a character generator. In order to build the character generator, I need to add another control to the library. In the SharedProject, right-click the Controls folder, select Add and then 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 System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace SharedProject.Controls
{
    public enum Direction { Down, Up}

    public class DirectionEventArgs : EventArgs
    {
        public Direction Direction;
    }

    public class RightLeftSelector : Control
    {
        #region Event Region

        public event EventHandler<DirectionEventArgs> 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(Direction direction)
        {
            DirectionEventArgs e = new() { Direction = direction };
            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);

            spriteBatch.Draw(_leftTexture, _leftSide, 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);

            spriteBatch.Draw(_rightTexture, _rightSide, Color.White);
        }
        public override void HandleInput()
        {
            if (_items.Count == 0)
                return;

            if (Xin.WasKeyReleased(Keys.Left) && _selectedItem != 0)
            {
                _selectedItem--;
                OnSelectionChanged(Direction.Down);
            }

            if (Xin.WasKeyReleased(Keys.Right) && _selectedItem != _items.Count - 1)
            {
                _selectedItem++;
                OnSelectionChanged(Direction.Up);
            }
        }

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

                if (_leftSide.Scale(Settings.Scale).Contains(mouse) && _selectedItem != 
0)
                {
                    _selectedItem--;
                    OnSelectionChanged(Direction.Down);
                }

                if (_rightSide.Scale(Settings.Scale).Contains(mouse) && _selectedItem != 
_items.Count - 1)
                {
                    _selectedItem++;
                    OnSelectionChanged(Direction.Up);
                }
            }

            if (Xin.TouchReleased())
            {
                if (_leftSide.Scale(Settings.Scale).Contains(Xin.TouchLocation) && _se-
lectedItem != 0)
                {
                    _selectedItem--;
                    OnSelectionChanged(Direction.Down);
                }

                if (_rightSide.Scale(Settings.Scale).Contains(Xin.TouchLocation) && _se-
lectedItem != _items.Count - 1)
                {
                    _selectedItem++;
                    OnSelectionChanged(Direction.Up);
                }
            }
        }

        #endregion
    }
}
```

First, there is an enum and class inside this file. The enum tells what direction a change occurs, up or down. The class is for the event arguments. It tells the subscriber in what direction the change has occurred. For the direction, moving left is moving down and moving right is moving up, of course. This is a class that I use often when having to choose from a list of items. In this case, we will be choosing from a list of two to nine. Why? I hear you ask. The way character generation is going to work is that the player has twenty-two points to spend on attributes. An attribute has a minimum of two and a maximum of nine. Health will be the product of strength and endurance. Mana will be the product of intellect and perception.

Back to the class. There is an event that will fire whenever the selected index has changed. It has as an event arguments type of DirectionEventArgs. For fields, there is a list of items to select from, textures for the left and right selectors, what colour a selected item is in, the width of the largest item in the list, what item is selected, the destination of the left and right textures and the y-offset, which is used for centering content vertically. There are properties to expose the value of the selected colour, the selected index, the selected item, the items, and the max width.

There is one constructor that takes the texture for the left and right selectors. It sets the appropriate field to the parameter passed in. You want to stop on the selector's TabStop property is set to true. It also initializes Color to white.

The SetItems method takes an array of strings for the items and the maximum width of the selector. It clears the items of the selector. It then loops over the items passed in and adds them to the selector. Finally, it sets the maximum width.

The OnSelectionChanged method is called whenever there is a change in the selection in the selector is changed. It checks to make sure the event is subscribed to before invoking the delegate.

All the Update method does is call a method HandleMouseInput. I guess I could have just placed that code inside of the Update method.

The Draw method draws the control, of course. The first step is to set a local variable to the Position field of the control. Next, I set the SpriteFont property to the SpriteFont of the control manager, just to make it a little shorter. Next, I center the text of the selected item vertically, measuring the W character because, typically, it is the largest character in a font. I calculate where to draw the left selector and then draw it. I increment the X property of the rendering by the width of the left texture plus five pixels for padding. I then calculate the width of the text to be drawn. I center the item horizontally. If the control has focus, I render it using the _selectedColor field. Otherwise, I draw it using the color. I then space the right selector slightly and draw it.

The HandleInput method handles keyboard, mouse and touch input. First, if there are no items in the list, I exit the method. If the left key was pressed, I decrement the selected item if it is greater than zero. If those conditions are true, I decrement the index and I then call the OnSelectionChanged method. For the right key, I check to see if we are on the last item. If we aren't, the selected item is incremented, and the OnSelectionChanged method is called. I do something similar if the left or right selectors are clicked or tapped.

With the selector, we can now build the character creation state. I will be using one similar to the one from A Summoner's Tale tutorial 0A (10). Right-click the GameScreens folder in the SharedProject, select Add and then Class. Name this new class NewGameState. Here is the code for that state. I do apologize for the code wall. There is just a lot going on with seven different attributes.

```
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework;
using System;
using System.Collections.Generic;
using System.Linq;
using SharedProject;
using SharedProject.Controls;
using SharedProject.GamesScreens;
using Microsoft.Xna.Framework.Content;

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

    public class NewGameState : GameState, INewGameState
    {
        private RightLeftSelector _portraitSelector;
        private RightLeftSelector _genderSelector;
        private TextBox _nameTextBox;
        private readonly Dictionary<string, Texture2D> _femalePortraits;
        private readonly Dictionary<string, Texture2D> _malePortraits;
        private Button _create;
        private Button _back;
        private RenderTarget2D renderTarget2D;

        private int _points = 22;

        private Label _pointsLabel;
        private Label _remainingLabel;

        private Label _strengthLabel;
        private RightLeftSelector _strengthSelector;

        private Label _perceptionLabel;
        private RightLeftSelector _perceptionSelector;

        private Label _enduranceLabel;
        private RightLeftSelector _enduranceSelector;

        private Label _charismaLabel;
        private RightLeftSelector _charismaSelector;

        private Label _intellectLabel;
        private RightLeftSelector _intellectSelector;

        private Label _agilityLabel;
        private RightLeftSelector _agilitySelector;
        private Label _luckLabel;
        private RightLeftSelector _luckSelector;

        public GameState GameState => this;

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

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

        protected override void LoadContent()
        {
            SpriteBatch = Game.Services.GetService<SpriteBatch>();
            renderTarget2D = new(GraphicsDevice, Settings.TargetWidth, Settings.Tar-
getHeight);

            ContentManager content = Game.Content;

            string[] items = new[] { "2", "3", "4", "5", "6", "7", "8", "9" };

            controls = new(content.Load<SpriteFont>(@"Fonts/MainFont"), 100);
            _genderSelector = new(
                content.Load<Texture2D>(@"GUI\g22987"),
                content.Load<Texture2D>(@"GUI\g21245"))
            {
                Position = new Vector2(207 - 70, 298)
            };

            _genderSelector.SelectionChanged += GenderSelector_SelectionChanged;
            _genderSelector.SetItems(new[] { "Female", "Male" }, 300);

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

            _portraitSelector.SelectionChanged += PortraitSelector_SelectionChanged;
            _portraitSelector.SetItems(new[] { "Fighter", "Wizard", "Rogue", "Priest" },
300);

            _pointsLabel = new()
            {
                Text = "Points to spend: ",
                Position = new(700, 20),
                Color = Color.White,
            };

            _remainingLabel = new()
            {
                Text = "22",
                Position = new(900, 20),
                Color = Color.Red,
            };

            _strengthLabel = new Label()
            {
                Color = Color.White,
                Text = "Strength",
                Position = new(700, 75)
            };

            _strengthSelector = new RightLeftSelector(
                content.Load<Texture2D>(@"GUI\g22987"),
                content.Load<Texture2D>(@"GUI\g21245"))
            {
                Position = new(900, 75)
            };

            _strengthSelector.SetItems(items, 75);
            _strengthSelector.SelectionChanged += Ability_SelectorChanged;

            _perceptionLabel = new Label()
            {
                Color = Color.White,
                Text = "Perception",
                Position = new(700, 150)
            };

            _perceptionSelector = new RightLeftSelector(
                content.Load<Texture2D>(@"GUI\g22987"),
                content.Load<Texture2D>(@"GUI\g21245"))
            {
                Position = new(900, 150)
            };

            _perceptionSelector.SetItems(items, 75);
            _perceptionSelector.SelectionChanged += Ability_SelectorChanged;

            _enduranceLabel = new Label()
            {
                Color = Color.White,
                Text = "Endurance",
                Position = new(700, 225)
            };

            _enduranceSelector = new RightLeftSelector(
                content.Load<Texture2D>(@"GUI\g22987"),
                content.Load<Texture2D>(@"GUI\g21245"))
            {
                Position = new(900, 225)
            };

            _enduranceSelector.SetItems(items, 75);
            _enduranceSelector.SelectionChanged += Ability_SelectorChanged;

            _charismaLabel = new Label()
            {
                Color = Color.White,
                Text = "Charisma",
                Position = new(700, 300)
            };
```

```
_charismaSelector = new RightLeftSelector(
    content.Load<Texture2D>(@"GUI\g22987"),
    content.Load<Texture2D>(@"GUI\g21245"))
{
    Position = new(900, 300)
};

_charismaSelector.SetItems(items, 75);
_charismaSelector.SelectionChanged += Ability_SelectorChanged;

_intellectLabel = new Label()
{
    Color = Color.White,
    Text = "Intellect",
    Position = new(700, 375)
};

_intellectSelector = new RightLeftSelector(
    content.Load<Texture2D>(@"GUI\g22987"),
    content.Load<Texture2D>(@"GUI\g21245"))
{
    Position = new(900, 375)
};

_intellectSelector.SetItems(items, 75);
_intellectSelector.SelectionChanged += Ability_SelectorChanged;

_agilityLabel = new Label()
{
    Color = Color.White,
    Text = "Agility",
    Position = new(700, 450)
};

_agilitySelector = new RightLeftSelector(
    content.Load<Texture2D>(@"GUI\g22987"),
    content.Load<Texture2D>(@"GUI\g21245"))
{
    Position = new(900, 450)
};

_agilitySelector.SetItems(items, 75);
_agilitySelector.SelectionChanged += Ability_SelectorChanged;

_luckLabel = new Label()
{
    Color = Color.White,
    Text = "Luck",
    Position = new(700, 525)
};

_luckSelector = new RightLeftSelector(
    content.Load<Texture2D>(@"GUI\g22987"),
    content.Load<Texture2D>(@"GUI\g21245"))
{
    Position = new(900, 525)
};

_luckSelector.SetItems(items, 75);
_luckSelector.SelectionChanged += Ability_SelectorChanged;

Texture2D background = new(GraphicsDevice, 100, 25);
Texture2D caret = new(GraphicsDevice, 2, 25);
Texture2D border = new(GraphicsDevice, 100, 25);

caret.Fill(Color.Black);
border.Fill(Color.Black);
background.Fill(Color.White);

_nameTextBox = new(background, caret, border)
{
    Position = new(207, 138),
    HasFocus = true,
    Enabled = true,
    Color = Color.Black,
    Text = "Bethany",
    Size = new(100, 25)
};

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

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


_portraitSelector.SetItems(_femalePortraits.Keys.ToArray(), 270);
_create = new(
    content.Load<Texture2D>(@"GUI\g9202"),
    ButtonRole.Accept)
{
    Text = "Create",
    Position = new(180, 640)
};

_create.Click += Create_Click;
_back = new(
    content.Load<Texture2D>(@"GUI\g9202"),
    ButtonRole.Cancel)
```

```
        {
            Text = "Back",
            Position = new(350, 640),
            Color = Color.White
        };

        _back.Click += Back_Click;

        ControlManager.Add(_pointsLabel);
        ControlManager.Add(_remainingLabel);
        ControlManager.Add(_nameTextBox);
        ControlManager.Add(_genderSelector);
        ControlManager.Add(_portraitSelector);
        ControlManager.Add(_create);
        ControlManager.Add(_back);
        ControlManager.Add(_strengthLabel);
        ControlManager.Add(_strengthSelector);
        ControlManager.Add(_perceptionLabel);
        ControlManager.Add(_perceptionSelector);
        ControlManager.Add(_enduranceLabel);
        ControlManager.Add(_enduranceSelector);
        ControlManager.Add(_charismaLabel);
        ControlManager.Add(_charismaSelector);
        ControlManager.Add(_intellectLabel);
        ControlManager.Add(_intellectSelector);
        ControlManager.Add(_agilityLabel);
        ControlManager.Add(_agilitySelector);
        ControlManager.Add(_luckLabel);
        ControlManager.Add(_luckSelector);
    }

    private void Ability_SelectorChanged(object sender, DirectionEventArgs e)
    {
        if (e.Direction == Direction.Up)
        {
            if (_points > 0)
            {
                _points--;
            }
            else
            {
                ((RightLeftSelector)sender).SelectedIndex--;
            }
        }
        else
        {
            if (_points < 22)
            {
                _points++;
            }
            else
            {
                ((RightLeftSelector)sender).SelectedIndex++;
            }
        }

        _remainingLabel.Text = _points.ToString();
    }

    private void Back_Click(object sender, EventArgs e)
```

```
        {
        }

        private void Create_Click(object sender, EventArgs e)
        {
            //Player = new(
            //    Game,
            //    _nameTextBox.Text,
            //    _genderSelector.SelectedIndex == 0,
            //    null);
            if (_points > 0) return;

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

            StateManager.PopState();
            StateManager.PushState(gamePlayState.Tag);

            gamePlayState.Tag.Enabled = true;
            //gamePlayState.NewGame();
        }

        private void GenderSelector_SelectionChanged(object sender, EventArgs e)
        {
        }

        private void PortraitSelector_SelectionChanged(object sender, EventArgs e)
        {
            if (_genderSelector.SelectedIndex == 0)
            {
            }
            else
            {
            }
        }

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

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

            SpriteBatch.Begin(SpriteSortMode.Deferred, BlendState.AlphaBlend, Sampler-
State.PointClamp);

            ControlManager.Draw(SpriteBatch);

            base.Draw(gameTime);

            SpriteBatch.End();

            GraphicsDevice.SetRenderTarget(null);

            SpriteBatch.Begin();

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

            SpriteBatch.End();

        }
    }
}
```

Wow, that's a lot of code. Fortunately, it is a lot of duplicate code. Not that it is doing the exact same thing, it is that it is very similar. As with all game states, there is an interface in this state that the state will implement. So, the class implements the interface but also inherits from GameState.

The first two fields are RightLeftSelectors which are used to select the player's portrait, which is not implemented at this point, and the player's gender. The next field is for the player's name. After that, are dictionaries that will hold the portraits of the character. The next fields are for creating the character and the other for going back. Next is the RenderTarget. Typically, I don't like magic numbers but this one slipped my radar. It is the number of points the player has to spend on attributes. As I've mentioned on my blog, an attribute has a minimum of two and a maximum of nine. In the SPECIAL system that I'm borrowing from, the player has 36 points to spend. However, attributes start at zero. I'm starting them at two and because there are sever attributes, I subtracted fourteen. There are also labels stating the text that there are points available and a label for the amount.

Following that, there are seven sets of fields. One for each of the attributes. There is a label that describes the attribute and a right-left selector to choose the value.

There is a single property that returns the instance of the NewGameState. The constructor registers the state as a service. It also initializes the portrait dictionaries. The Initialize method is currently unused but may be in the future, so it is included.

The LoadContent method retrieves the SpriteBatch that was registered in the game. It then creates the render target that we will be using to render our scene. I also assign the content manager to a shorter variable name. Next, is an array of strings with values between two and nine. They will populate the right-left selectors. Next, I create the ControlManager.

After creating the control manager, it is time to start creating controls. First up, is the right-left selector for gender. I wire an event handler for future use. I then set the text and maximum width. It is the same process for the portrait selector. It has as values the four classes that we will be implementing.

After creating those, I create the labels for the description of the number of points left to spend and the actual value left. The only interesting thing is that I set the colour of the points remaining to red.

Now I create the labels for the attributes and their selectors. The labels are trivial. The only thing is they are positioned to the left of the selectors. The selectors are straightforward as well. They are positioned to the right of the labels. I use the same buttons as for the gender and portrait, or class, selector. I then call SetItems on the selector to set its items. I pass in the array from earlier and I pass in seventy-five for the max width. That was discovered through trial and error. I follow the process a total of seven times, one for each attribute. I also wire them all to the same event handler that will control the delta of the points left.

Now, I create the text box for the player's name. I create the textures for the background, the caret and the border. I then fill them using the Fill extension method. Now I create the text box and set its Size property. I then load the portraits, otherwise known as sprites. I then set the items for the portrait selector. Now I create the buttons for creating the character and going back to the previous screen. I also wire their event handlers. The last step is to add the controls to the control manager.

The Ability_SelectorChanged method handles the player trying to change an attribute. It takes as parameters the selector that triggered the handler and a DirectionEventArgs that is the direction the player selected. The Up direction is if the right side has been selected. If it has and there are points available, it decrements points. Otherwise, it decrements the selector that triggered the event, negating the choice. Similarly, if there are fewer than twenty-two points, I increment points. Otherwise, I increment the selector, negating the choice.

Currently, the back event does nothing. I will handle that in a future tutorial, along with a few other enhancements that I have in mind. Fortunately, they do not involve giving you the code for the entire class all at once.

In the click event handler of the Create button, there is a bit of commented-out code that would create the player, if there was a player. It is another enhancement for a future tutorial. If there are points left, I exit the method. Next, I get the gameplay state from the services. I pop the current state off of the stack. I then push the gameplay state on top of the stack. I set it to enabled, and there is more code that is commented out that will start a new game when we get there.

The event handlers for the gender selector and portrait selector currently do nothing. Again, future enhancements. Similarly, the Update method just calls the update method of the base class. The drawing code should be familiar by now. Set the render target, clear the buffer, render the scene, reset the buffer, and finally draw the render target.

The next piece of the puzzle is implementing the new state in the Desktop class. I noticed that the SummonersTale.StateManagement namespace crept into that class. I'm not editing it at this point, so it can stay. Replace the Desktop class with the following.

```
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using SharedProject;
using SharedProject.GameScreens;
using SharedProject.GamesScreens;
using SharedProject.StateManagement;
using SummonersTale.StateManagement;

namespace EyesOfTheDragon
{
    public class Desktop : Game
    {
        private readonly GraphicsDeviceManager _graphics;
        private SpriteBatch _spriteBatch;
        public GameStateManager GameStateManager { get; private set; }
        public ITitleState TitleState { get; private set; }
        public IStartMenuState StartMenuState { get; private set; }
        public INewGameState NewGameState { get; private set; }
        public IGamePlayState GamePlayState { get; private set; }
        public IConversationState ConversationState { get; private set; }

        public IConversationManager ConversationManager { get; private set; }
        public Player Player { get; private set; }

        public Desktop()
        {
            _graphics = new GraphicsDeviceManager(this);

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

            _graphics.PreferredBackBufferWidth = Settings.BaseWidth;
            _graphics.PreferredBackBufferHeight = Settings.BaseHeight;
            _graphics.ApplyChanges();

            Components.Add(new Xin(this));

            GameStateManager = new GameStateManager(this);
            Components.Add(GameStateManager);
            Services.AddService(typeof(GameStateManager), GameStateManager);
        }

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

            Settings.TargetHeight = _graphics.PreferredBackBufferHeight;
            Settings.TargetWidth = _graphics.PreferredBackBufferWidth;

            base.Initialize();
        }

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

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

            ConversationManager = new ConversationManager(this);
            Components.Add((ConversationManager)ConversationManager);
            ConversationManager.LoadConversations(this);

            TitleState = new TitleState(this);
            StartMenuState = new StartMenuState(this);
            NewGameState = new NewGameState(this);
            GamePlayState = new GamePlayState(this);
            ConversationState = new ConversationState(this);

            GameStateManager.PushState((TitleState)TitleState);
        }
        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);
        }
    }
}
```

No big surprises here. Just add a property for the state. Initialize the property in the LoadContent method. The next step is to move from the title screen to the new game state. Replace the title state with the following code.

```
using System;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.Graphics;
using SharedProject.Controls;
using SharedProject.GamesScreens;
using SummonersTale.StateManagement;

namespace SharedProject.GameScreens
{
    public interface ITitleState
    {
        GameState Tag { get; }
    }

    public class TitleState : GameState, ITitleState
    {
        #region Field region

        Texture2D backgroundImage;
        RenderTarget2D renderTarget;

        readonly GameState tag;

        public GameState Tag
        {
            get { return tag; }
        }

        #endregion

        #region Constructor region
        public TitleState(Game game)
            : base(game)
        {
            Game.Services.AddService<ITitleState>(this);
            SpriteBatch = Game.Services.GetService<SpriteBatch>();
        }

        #endregion

        #region XNA Method region

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

            renderTarget = new(GraphicsDevice, Settings.BaseWidth, Settings.BaseHeight);

            ContentManager Content = Game.Content;
            backgroundImage = Content.Load<Texture2D>(@"Backgrounds\titlescreen");

            Label startLabel = new()
            {
                Position = new Vector2(350, 600),
                Text = "Tap to begin",
                Color = Color.White,
                TabStop = true,
                HasFocus = true
            };

            startLabel.Selected += StartLabel_Selected; ;

            ControlManager.Add(startLabel);
        }

        private void StartLabel_Selected(object sender, EventArgs e)
        {
            GameState state = Game.Services.GetService<INewGameState>().GameState;
            StateManager.PushState(state);
        }

        public override void Update(GameTime gameTime)
        {
            if (Xin.WasMouseReleased(MouseButtons.Left) || Xin.TouchReleased() ||
Xin.WasKeyPressed())
            {
                StartLabel_Selected(this, null);
            }
            base.Update(gameTime);
        }

        public override void Draw(GameTime gameTime)
        {
            GraphicsDevice.SetRenderTarget(renderTarget);
            renderTarget.GraphicsDevice.Clear(Color.Black);

            SpriteBatch.Begin();

            SpriteBatch.Draw(
                backgroundImage,
                Settings.BaseRectangle,
                Color.White);

            base.Draw(gameTime);

            SpriteBatch.End();

            GraphicsDevice.SetRenderTarget(null);

            SpriteBatch.Begin();

            SpriteBatch.Draw(renderTarget, Settings.TargetRectangle, Color.White);

            SpriteBatch.End();
        }

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

            LoadContent();
        }

        #endregion
    }
}
```

As with the Desktop class, the namespace from Summoner's Tale crept in. What I do is grab the instance of the new game state and push it on top of the stack.

Now, as a result of my solution imploding the other day, I had to make changes to the Activity1 class and the Android class. What I had to do is rename the Android class to Game and update the Activity1 class. Right-click the Android class and select Rename. Change the name to Game. Now, replace the Activity1 class with the following version.

```
using Android.App;
using Android.Content.PM;
using Android.OS;
using Android.Views;
using Microsoft.Xna.Framework;

namespace EyesOfTheDragonAndroid
{
    [Activity(
        Label = "@string/app_name",
        MainLauncher = true,
        Icon = "@drawable/icon",
        AlwaysRetainTaskState = true,
        LaunchMode = LaunchMode.SingleInstance,
        ScreenOrientation = ScreenOrientation.Landscape,
        ConfigurationChanges = ConfigChanges.Orientation | ConfigChanges.Keyboard | Con-
figChanges.KeyboardHidden | ConfigChanges.ScreenSize
    )]
    public class Activity1 : AndroidGameActivity
    {
        private Game _game;
        private View _view;
        protected override void OnCreate(Bundle bundle)
        {
            base.OnCreate(bundle);

            _game = new Game();
            _view = _game.Services.GetService(typeof(View)) as View;
            SetContentView(_view);
            HideSystemUI();
            _game.Run();
        }

        private void HideSystemUI()
        {
            // Apparently for Android OS Kitkat and higher, you can set a full screen
mode. Why this isn't on by default, or some kind
            // of simple switch, is beyond me.
            // Got this from the following forum post: http://community.mo-
nogame.net/t/blocking-the-menu-bar-from-appearing/1021/2
            if (Build.VERSION.SdkInt >= BuildVersionCodes.Kitkat)
            {
                View decorView = Window.DecorView;
                var uiOptions = (int)decorView.SystemUiVisibility;
                var newUiOptions = (int)uiOptions;

                newUiOptions |= (int)SystemUiFlags.LowProfile;
                newUiOptions |= (int)SystemUiFlags.Fullscreen;
                newUiOptions |= (int)SystemUiFlags.HideNavigation;
                newUiOptions |= (int)SystemUiFlags.ImmersiveSticky;

                decorView.SystemUiVisibility = (StatusBarVisibility)newUiOptions;

                this.Immersive = true;
            }
        }
    }
}
```

The last change I am going to make in this tutorial is to the Settings class. All I did was change the initialization of the Resolution property. Instead of using Target, I use Base. This is only until I implement saving and loading settings. Replace the Settings class with this version.

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

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

        public static Rectangle BaseRectangle { get { return new(0, 0, BaseWidth, Base-
Height); } }
        public static Rectangle TargetRectangle { get { return new(0, 0, TargetWidth,
TargetHeight); } }
        public static Point Resolution { get; set; } = new(BaseWidth, BaseHeight);
        public static int TargetWidth { get; set; } = BaseWidth;
        public static int TargetHeight { get; set; } = BaseHeight;

        public static Vector2 Scale
        {
            get { return new((float)TargetWidth / BaseWidth, (float)TargetHeight / Base-
Height);}
        }


    }
}
```

If you build and run now, you can run through the process of creating a character, for the most part. I'm going to park this tutorial here. There is more that I could pack in, but I will save it for the following tutorials. I don't want you to have too much to digest at once. I encourage you to visit the news page of my site, https://cynthiamcmahon.ca/blog/, for the latest news on my tutorials. Also, I'm thinking of reviving my newsletter so you will be informed of new stuff rather than having to keep looking for new things.

Good luck with your game programming adventures!
Cynthia