Eyes of the Dragon Tutorials
Part 53
Levelling Up

I'm writing these tutorials for the MonoGame 3.8 framework using Visual Studio 2019. 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 page of my web blog. I will be making each version of the project available on GitHub here. It will be included on the page that links to the tutorials.

The main focus for this tutorial will be levelling up the player's character. Also, it will cover display the character's stats because we will need that in order to add points to the character's stats. So, let's get started.

The first thing that I am going to do is add a new game screen for displaying the player's stats. Right click the GameScreens folder in the EyesOfTheDragon project, select Add and then Class. Name this new class StatsScreen. Here is the code.

```
using MGRpgLibrary;
using MGRpgLibrary.Controls;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using System;
using System.Collections.Generic;
using System.Text;

namespace EyesOfTheDragon.GameScreens
{
    public class StatsScreen : BaseGameState
    {
        protected Texture2D _background;
        protected Label _str;
        protected Label _dex;
        protected Label _cun;
        protected Label _mag;
        protected Label _wil;
        protected Label _con;
        protected Label _lvl;
        protected Label _xp;
        protected Label _next;
        protected Label _hp;
        protected Label _res;
        protected Label _resName;

        public StatsScreen(Game game, GameStateManager manager) : base(game, manager)
        {
        }

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

            Color[] data = new Color[500 * 500];
```

```
for (int i = 0; i < data.Length; i++)
{
    data[i] = Color.Black;
}

_background = new Texture2D(GraphicsDevice, 500, 500);
_background.SetData(data);

int yOffset = (720 - 500) / 2;
int xOffset = (1280 - 500) / 2 + 20;

Label label = new Label()
{
    Position = new Vector2(xOffset, 40 + yOffset),
    Text = "Strength: ",
    Color = Color.White,
    TabStop = false
};

ControlManager.Add(label);

_str = new Label()
{
    Position = new Vector2(xOffset + 160, 40 + yOffset),
    Text = $"{GamePlayScreen.Player.Character.Entity.Strength}",
    Color = Color.White,
    TabStop = false
};

ControlManager.Add(_str);

label = new Label()
{
    Position = new Vector2(xOffset, 80 + yOffset),
    Text = "Dexterity: ",
    Color = Color.White,
    TabStop = false
};

ControlManager.Add(label);

_dex = new Label()
{
    Position = new Vector2(xOffset + 160, 80 + yOffset),
    Text = $"{GamePlayScreen.Player.Character.Entity.Dexterity}",
    Color = Color.White,
    TabStop = false
};

ControlManager.Add(_dex);

label = new Label()
{
    Position = new Vector2(xOffset, 120 + yOffset),
    Text = "Cunning: ",
    Color = Color.White,
    TabStop = false
};
```

```
ControlManager.Add(label);

_cun = new Label()
{
    Position = new Vector2(xOffset + 160, 120 + yOffset),
    Text = $"{GamePlayScreen.Player.Character.Entity.Cunning}",
    Color = Color.White,
    TabStop = false
};

ControlManager.Add(_cun);

label = new Label()
{
    Position = new Vector2(xOffset, 160 + yOffset),
    Text = "Magic: ",
    Color = Color.White,
    TabStop = false
};

ControlManager.Add(label);

_mag = new Label()
{
    Position = new Vector2(xOffset + 160, 160 + yOffset),
    Text = $"{GamePlayScreen.Player.Character.Entity.Magic}",
    Color = Color.White,
    TabStop = false
};

ControlManager.Add(_mag);

label = new Label()
{
    Position = new Vector2(xOffset, 200 + yOffset),
    Text = "Will Power: ",
    Color = Color.White,
    TabStop = false
};

ControlManager.Add(label);

_wil = new Label()
{
    Position = new Vector2(xOffset + 160, 200 + yOffset),
    Text = $"{GamePlayScreen.Player.Character.Entity.Willpower}",
    Color = Color.White,
    TabStop = false
};

ControlManager.Add(_wil);
label = new Label()
{
    Position = new Vector2(xOffset, 240 + yOffset),
    Text = "Constitution: ",
    Color = Color.White,
    TabStop = false
};

ControlManager.Add(label);
```

```
_con = new Label()
{
    Position = new Vector2(xOffset + 160, 240 + yOffset),
    Text = $"{GamePlayScreen.Player.Character.Entity.Constitution}",
    Color = Color.White,
    TabStop = false
};

ControlManager.Add(_con);

label = new Label()
{
    Position = new Vector2(xOffset + 250, yOffset + 40),
    Text = "Level:",
    Color = Color.White,
    TabStop = false
};

ControlManager.Add(label);

_lvl = new Label
{
    Position = new Vector2(xOffset + 250 + 80, yOffset + 40),
    Text = $"{GamePlayScreen.Player.Character.Entity.Level}",
    Color = Color.Yellow,
    TabStop = false
};

ControlManager.Add(_lvl);

label = new Label()
{
    Position = new Vector2(xOffset + 250, yOffset + 80),
    Text = "XP:",
    Color = Color.White,
    TabStop = false
};

ControlManager.Add(label);

_xp = new Label
{
    Position = new Vector2(xOffset + 250 + 80, yOffset + 80),
    Text = $"{GamePlayScreen.Player.Character.Entity.Experience}",
    Color = Color.Yellow,
    TabStop = false
};

ControlManager.Add(_xp);

label = new Label()
{
    Position = new Vector2(xOffset + 250, yOffset + 120),
    Text = "HP:",
    Color = Color.White,
    TabStop = false
};

ControlManager.Add(label);
```

```
            _hp = new Label
            {
                Position = new Vector2(xOffset + 250 + 80, yOffset + 120),
                Text = $"{GamePlayScreen.Player.Character.Entity.Health.CurrentValue} / 
{GamePlayScreen.Player.Character.Entity.Health.MaximumValue}",
                Color = Color.Yellow,
                TabStop = false
            };

            ControlManager.Add(_hp);

            _resName = new Label
            {
                Position = new Vector2(xOffset + 250, yOffset + 160),
                Color = Color.White,
                TabStop = false
            };

            _resName.Text = GamePlayScreen.Player.Character.Entity.Mana.MaximumValue > 0 ? 
"MP:" : "SP:";

            ControlManager.Add(_resName);

            _res = new Label
            {
                Position = new Vector2(xOffset + 250 + 80, yOffset + 160),
                Color = Color.Yellow,
                TabStop = false
            };

            _res.Color = GamePlayScreen.Player.Character.Entity.Mana.MaximumValue > 0 ? 
Color.Yellow : Color.Blue;

            _res.Text = GamePlayScreen.Player.Character.Entity.Mana.MaximumValue > 0 ?
                $"{GamePlayScreen.Player.Character.Entity.Mana.CurrentValue} / " +
                $"{GamePlayScreen.Player.Character.Entity.Mana.MaximumValue}" :
                $"{GamePlayScreen.Player.Character.Entity.Stamina.CurrentValue} / " +
                $"{GamePlayScreen.Player.Character.Entity.Stamina.MaximumValue}";

            ControlManager.Add(_res);
            LinkLabel close = new LinkLabel()
            {
                Position = new Vector2(390 + (500 - 
FontManager.GetFont("testfont").MeasureString("Close").X) / 2,
                    (720 - 500) / 2 + 500 - FontManager.GetFont("testfont").LineSpacing - 10),
                Color = Color.White,
                TabStop = true,
                HasFocus = true,
                Text = "Close"
            };

            ControlManager.Add(close);

            close.Selected += Close_Selected;
        }

        private void Close_Selected(object sender, EventArgs e)
        {
            OnCloseSelected();
        }

        protected virtual void OnCloseSelected()
        {
            StateManager.PopState();
        }

        public override void Update(GameTime gameTime)
        {
            ControlManager.Update(gameTime, PlayerIndex.One);

            base.Update(gameTime);
        }

        public override void Draw(GameTime gameTime)
        {
            base.Draw(gameTime);
            _str.Text = $"{GamePlayScreen.Player.Character.Entity.Strength}";
            _dex.Text = $"{GamePlayScreen.Player.Character.Entity.Dexterity}";
            _cun.Text = $"{GamePlayScreen.Player.Character.Entity.Cunning}";
            _mag.Text = $"{GamePlayScreen.Player.Character.Entity.Magic}";
            _wil.Text = $"{GamePlayScreen.Player.Character.Entity.Willpower}";
            _con.Text = $"{GamePlayScreen.Player.Character.Entity.Constitution}";
            _lvl.Text = $"{GamePlayScreen.Player.Character.Entity.Level}";
            _xp.Text = $"{GamePlayScreen.Player.Character.Entity.Experience}";
            _hp.Text = $"{GamePlayScreen.Player.Character.Entity.Health.CurrentValue} / 
{GamePlayScreen.Player.Character.Entity.Health.MaximumValue}";
            _resName.Text = GamePlayScreen.Player.Character.Entity.Mana.MaximumValue > 0 ? 
"MP:" : "SP:";
            _res.Color = GamePlayScreen.Player.Character.Entity.Mana.MaximumValue > 0 ? 
Color.Yellow : Color.Blue;
            _res.Text = GamePlayScreen.Player.Character.Entity.Mana.MaximumValue > 0 ?
                $"{GamePlayScreen.Player.Character.Entity.Mana.CurrentValue} / " +
                $"{GamePlayScreen.Player.Character.Entity.Mana.MaximumValue}" :
                $"{GamePlayScreen.Player.Character.Entity.Stamina.CurrentValue} / " +
                $"{GamePlayScreen.Player.Character.Entity.Stamina.MaximumValue}";

            GameRef.SpriteBatch.Begin();

            GameRef.SpriteBatch.Draw(_background, new Rectangle((1280 - 500) / 2, (720 - 500) / 
2, 500, 500), Color.White);
            ControlManager.Draw(GameRef.SpriteBatch);

            GameRef.SpriteBatch.End();
        }
    }
}
```

So, it's a lot of code. It is very repetitious, though. First, there are some using statements to bring classes in the MGRpgLibrary and MonoGame libraries into scope. The class inherits from BaseGameState so it can be used in the StateManager.

There is a Texture2D field that is the background for the screen. The screen is meant to be a popup and not fill the entire screen. I will get to it in a minute but I will mention I create the background on the fly rather than create one in Photoshop or some other program. I don't want you to have to download something and add it to the project.

The rest of the fields are Labels. Since their value can change between views I had to create a Label for each value that I want to display. I chose Strength, Dexterity, Cunning, Magic, Will Power, Constitution, Level, Experience, Health, Mana or Stamina, and the text for the resource being displayed.

In the LoadContent method I create a texture on the fly. The first step is to create an array of Color objects that describe the texture's color data. I create one 500 pixels by 500 pixels and fill it with black. I then create the texture and set the color data. I next create some local variables that are the offset of the X and Y coordinates of the background. I had to do this because I center the texture and I need to position the labels according to the offset instead of the origin.

Next I start creating the labels. For the text for each attribute, other than mana or stimina, the values are fixed so I just use a local variable. I set its Position, Text, Color and TabStop properties. The labels are then added to the ControlManager. For the values, I do the same. However, the text is set to the value of the attribute being displayed. I use interpolation when setting the text.

The interesting cases are mana and stamina. I check the Maximum value for the mana of the character. If it is greater than zero I set the Text property of the label to Mana, otherwise I set it to Stamina. Similarly, for the value I check the Maximum value of the mana of the character and compare it to zero. If it is greater than 0 I display the text in yellow, blue otherwise. The values are also set by comparing the maximum mana to zero.

After creating the labels for the attributes I create a LinkLabel that is used to close the screen. I just position it at the bottom of the texture, set its Text property to Close, TabStop property to try and HasFocus to true. I then wire an event handler for the Selected event. In that event handler I pop the state off of the stack.

What the Update method does is call the Update method of the control manager passing in the required parameters. Next it calls the Update method of the base class.

The Draw method is where the magic takes place. I first call base.Update to render the base class because I want this state on top of everything else. Next, I set the Text properties for the attributes the same way I did in the LoadContent method. Finally I draw the background texture and control manager, in that order so the control manager is on top of the background texture.

Let's trigger the new screen. That will done in the Update method of the GamePlayScreen. I will trigger it when the C key has been released. First, I need to create an instance of StatsScreen in the Game1 class. Add the following property to the Game1 class and update the constructor.

```
public StatsScreen StatsScreen { get; private set; }

public Game1()
{
    _graphics = new GraphicsDeviceManager(this);
    ScreenRectangle = new Rectangle(
        0,
        0,
        ScreenWidth,
        ScreenHeight);
    IsMouseVisible = true;

    Content.RootDirectory = "Content";

    Components.Add(new InputHandler(this));

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

    _ = new TextureManager();

    TitleScreen = new TitleScreen(this, _gameStateManager);
    StartMenuScreen = new StartMenuScreen(this, _gameStateManager);
    GamePlayScreen = new GamePlayScreen(this, _gameStateManager);
    CharacterGeneratorScreen = new CharacterGeneratorScreen(this, _gameStateManager);
    SkillScreen = new SkillScreen(this, _gameStateManager);
    LoadGameScreen = new LoadGameScreen(this, _gameStateManager);
    ConversationScreen = new ConversationScreen(this, _gameStateManager);
    ShopScreen = new ShopState(this, _gameStateManager);
    InventoryScreen = new InventoryScreen(this, _gameStateManager);
    CombatScreen = new CombatScreen(this, _gameStateManager);
    GameOverScreen = new GameOverScreen(this, _gameStateManager);
    LootScreen = new LootScreen(this, _gameStateManager);
    StatsScreen = new StatsScreen(this, _gameStateManager);

    _gameStateManager.ChangeState(TitleScreen);

    IsFixedTimeStep = false;
    _graphics.SynchronizeWithVerticalRetrace = false;
}
```

Now, we can push the instance onto the stack from the GamePlayState state. Change the Update method of the GamePlayState class to the following.

```
public override void Update(GameTime gameTime)
{
    world.Update(gameTime);
    player.Update(gameTime);
    player.Camera.LockToSprite(player.Sprite);

    if (InputHandler.KeyReleased(Keys.Space) ||
        InputHandler.ButtonReleased(Buttons.A, PlayerIndex.One))
    {
        foreach (ILayer layer in World.Levels[World.CurrentLevel].Map.Layers)
        {
            if (layer is CharacterLayer)
            {
                foreach (Character c in ((CharacterLayer)layer).Characters.Values)
                {
                    float distance = Vector2.Distance(
                        player.Sprite.Center,
                        c.Sprite.Center);

                    if (distance < Character.SpeakingRadius && c is NonPlayerCharacter)
                    {
                        NonPlayerCharacter npc = (NonPlayerCharacter)c;

                        if (npc.HasConversation)
                        {
                            StateManager.PushState(GameRef.ConversationScreen);

                            GameRef.ConversationScreen.SetConversation(
                                player,
                                npc,
                                npc.CurrentConversation);

                            GameRef.ConversationScreen.StartConversation();
                        }
                    }
                    else if (distance < Character.SpeakingRadius && c is Merchant)
                    {
                        StateManager.PushState(GameRef.ShopScreen);
                        GameRef.ShopScreen.SetMerchant(c as Merchant);
                    }
                }
            }
        }
    }

    MobLayer mobLayer = World.Levels[World.CurrentLevel].Map.Layers.Find(x => x is MobLayer) as 
MobLayer;

    foreach (var mob in mobLayer.Mobs.Where(kv => kv.Value.Entity.Health.CurrentValue <= 
0).ToList())
    {
        mobLayer.Mobs.Remove(mob.Key);
    }

    foreach (Rectangle r in mobLayer.Mobs.Keys)
    {
        float distance = Vector2.Distance(mobLayer.Mobs[r].Sprite.Center, 
player.Sprite.Center);

        if (distance < Mob.AttackRadius)
        {
            GameRef.CombatScreen.SetMob(mobLayer.Mobs[r]);

            StateManager.PushState(GameRef.CombatScreen);
            Visible = true;
        }
    }

    if (InputHandler.KeyReleased(Keys.I))
    {
        StateManager.PushState(GameRef.InventoryScreen);
    }

    if (InputHandler.KeyReleased(Keys.C))
    {
        StateManager.PushState(GameRef.StatsScreen);
        Visible = true;
    }

    base.Update(gameTime);
}
```

The one thing I did after pushing the instance of StatsScreen onto the stack is set the Visible property of the GamePlayState to true so it will render underneath the StatsScreen.

So, I found a bug while testing. It turns out I missed an instance of the mob dying on the CombatScreen. Since I missed it, the character will never gain experience. Change the Update method of the CombatScreen to the following.

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

    if (_queueTimer > 2)
    {
        if (GameState._descriptions.Count > 0)
        {
            GameState._descriptions.Dequeue();
        }

        _queueTimer = 0;
    }

    if (!_displayActionTexture)
    {
        _scene.Update(gameTime, PlayerIndex.One);
    }

    if (GamePlayScreen.Player.Character.Entity.Health.CurrentValue <= 0)
    {
        StateManager.ChangeState(GameRef.GameOverScreen);
    }

    if (InputHandler.KeyReleased(Microsoft.Xna.Framework.Input.Keys.Space) && !
_displayActionTexture)
    {
        switch (_scene.SelectedIndex)
        {
            case 0:
                _queueTimer = 0;
                if (GamePlayScreen.Player.Character.Entity.Dexterity >= _mob.Entity.Dexterity)
                {
                    _descriptions.Enqueue($"You attack the {_mob.Entity.EntityClass}.");
                    _mob.Attack(GamePlayScreen.Player.Character.Entity);

                    if (_mob.Entity.Health.CurrentValue <= 0)
                    {
                        StateManager.PopState();
                        StateManager.PushState(GameRef.LootScreen);
                        GamePlayScreen.Player.Character.Entity.AddExperience(_mob.XPValue);
                        GameRef.LootScreen.Gold = _mob.GoldDrop;

                        foreach (var i in _mob.Drops)
                        {
                            GameRef.LootScreen.Items.Add(i);
                        }

                        return;
                    }

                    _descriptions.Enqueue($"The {_mob.Entity.EntityClass} attacks you.");
                    _mob.DoAttack(GamePlayScreen.Player.Character.Entity);
                }
                else
                {
                        _descriptions.Enqueue($"The {_mob.Entity.EntityClass} attacks you.");
                        _mob.DoAttack(GamePlayScreen.Player.Character.Entity);
                        _descriptions.Enqueue($"You attack the {_mob.Entity.EntityClass}.");
                        _mob.Attack(GamePlayScreen.Player.Character.Entity);

                        if (_mob.Entity.Health.CurrentValue <= 0)
                        {
                            StateManager.PopState();
                            StateManager.PushState(GameRef.LootScreen);

                            GamePlayScreen.Player.Character.Entity.AddExperience(_mob.XPValue);
                            foreach (var i in _mob.Drops)
                            {
                                GameRef.LootScreen.Items.Add(i);
                            }

                            return;
                    }
                }
                break;
            case 1:
                _displayActionTexture = true;

                _actions.Clear();
                Entity entity = GamePlayScreen.Player.Character.Entity;

                if (entity.Mana.MaximumValue > 0)
                {
                    foreach (SpellData s in DataManager.SpellData.SpellData.Values)
                    {
                        foreach (string c in s.AllowedClasses)
                        {
                            if (c == entity.EntityClass && entity.Level >= s.LevelRequirement)
                            {
                                _actions.Add(s.Name);
                            }
                        }
                    }
                }
                else
                {
                    foreach (TalentData s in DataManager.TalentData.TalentData.Values)
                    {
                        foreach (string c in s.AllowedClasses)
                        {
                            if (c == entity.EntityClass && entity.Level >= s.LevelRequirement)
                            {
                                _actions.Add(s.Name);
                            }
                        }
                    }
                }
                return;
            case 2:
                Vector2 center = GamePlayScreen.Player.Sprite.Center;
                Vector2 mobCenter = _mob.Sprite.Center;
                _action = 0;
                if (center.Y < mobCenter.Y)
                {
                    GamePlayScreen.Player.Sprite.Position.Y -= 8;
                }
                else if (center.Y > mobCenter.Y)
                {
                    GamePlayScreen.Player.Sprite.Position.Y += 8;
                }
                else if (center.X < mobCenter.X)
                {
                    GamePlayScreen.Player.Sprite.Position.X -= 8;
                }
                else
                {
                    GamePlayScreen.Player.Sprite.Position.X += 8;
                }

                StateManager.PopState();
                break;
        }
    }

    if (_displayActionTexture)
    {
        if (InputHandler.KeyReleased(Microsoft.Xna.Framework.Input.Keys.Escape))
        {
            _displayActionTexture = false;
            return;
        }
        else if (InputHandler.KeyReleased(Microsoft.Xna.Framework.Input.Keys.Space))
        {
            DoAction();
        }
        else if (InputHandler.KeyReleased(Microsoft.Xna.Framework.Input.Keys.Down))
        {
            _action++;

            if (_action >= _actions.Count)
            {
                _action = 0;
            }
        }
        else if (InputHandler.KeyReleased(Microsoft.Xna.Framework.Input.Keys.Up))
        {
            _action--;

            if (_action < 0)
            {
                _action = _actions.Count - 1;
            }
        }
    }

    base.Update(gameTime);
}
```

Now, if the player hits before the mob and the mob dies the character will gain experience. The way I am going to handle levelling up is have a table with experience values. If the player's experience is greater than or equal to the next level they will level up. I will push a level up screen onto the stack where they can assign three points to their attributes. I will not be handling choosing a new skill in this tutorial, though. I am still formulating how I want to handle that.

This will be done in a few steps. First, I added an array of integers to the Mechanics class to hold the experience required to level up at that level. Change the Mechanics class to the following.

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

namespace RpgLibrary
{
    public enum DieType { D4 = 4, D6 = 6, D8 = 8, D10 = 10, D12 = 12, D20 = 20, D100 = 100 }

    public static class Mechanics
    {
        #region Field Region

        public static readonly int[] Experiences = new[] {
            0,
            1000,
            2000,
            4000,
            8000,
            12000,
            16000,
            32000,
            64000,
            100000,
            200000,
            400000,
            600000,
            800000,
            1000000,
            1200000,
            1600000,
            2000000,
            4000000,
            10000000,
            20000000
        };
        static Random random = new Random();

        public static int GetModifier(int value)
        {
            if (value <= 3)
            {
                return -4;
            }

            if (value <= 5)
            {
                return -2;
            }

            if (value <= 8)
            {
                return -1;
            }

            if (value > 8 && value <= 12)
            {
                return 0;
            }

            if (value <= 14)
            {
                return 1;
            }

            if (value <= 16)
            {
                return 2;
            }

            if (value <= 18)
            {
                return 3;
            }

            if (value <= 20)
            {
                return 4;
            }

            return 5;
        }

        #endregion

        #region Property Region

        public static Random Random => random;

        #endregion

        #region Constructor Region
        #endregion

        #region Method Region

        public static int RollDie(DieType die)
        {
            return random.Next(0, (int)die) + 1;
        }

        public static int GetAttributeByString(Entity entity, string attribute)
        {
            int value = 0;

            switch (attribute.ToLower())
            {
                case "strength":
                    value = entity.Strength;
                    break;
                case "dexterity":
                    value = entity.Dexterity;
                    break;
```

```
                case "cunning":
                    value = entity.Cunning;
                    break;
                case "willpower":
                    value = entity.Willpower;
                    break;
                case "magic":
                    value = entity.Magic;
                    break;
                case "constitution":
                    value = entity.Constitution;
                    break;
            }

            return value;
        }

        #endregion

        #region Virtual Method Region
        #endregion
    }
}
```

I picked some arbitrary values for levelling up. At higher levels they are probably too high but they are good enough for now. Before I get to that I need to update the Entity class. I need to add a way to adjust the stats. Since they are private set accessors I cannot do it outside of the class. The same is true for levels. The approach I took to resolve this was add a method that takes as parameters the attribute to be modified and the amount and a method to increase the level. Add the following methods to the Entity class.

```
public void LevelUp()
{
    Level++;
}

public void AdjustAttribut(string attribute, int amount)
{
    switch (attribute)
    {
        case "Strength":
            Strength += amount;
            break;
        case "Dexterity":
            Dexterity += amount;
            break;
        case "Cunning":
            Cunning += amount;
            break;
        case "Magic":
            Magic += amount;
            break;
        case "Will Power":
            Willpower += amount;
            break;
        case "Constitution":
            Constitution += amount;
            break;
    }
}
```

The LevelUp method just increments the Level property. In the AdjustAttribute method there is just a switch on the attribute passed in. If it matches it increments the attribute by the amount.
The next step is to add a state that will handle levelling up. Right click the GameScreens folder in the EyesOfTheDragon project, select Add and then Class. Name this new class LevelScreen. Here is the code for that class.

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

namespace EyesOfTheDragon.GameScreens
{
    public class LevelScreen : StatsScreen
    {
        private float _points;
        private LinkLabel _strBtn;
        private LinkLabel _dexBtn;
        private LinkLabel _cunBtn;
        private LinkLabel _magBtn;
        private LinkLabel _wilBtn;
        private LinkLabel _conBtn;

        public LevelScreen(Game game, GameStateManager manager) : base(game, manager)
        {
        }

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

            _strBtn = new LinkLabel
            {
                Text = "+",
                Position = _str.Position + new Vector2(50, 0),
                TabStop = true,
            };

            ControlManager.Add(_strBtn);


            _dexBtn = new LinkLabel
            {
                Text = "+",
                Position = _dex.Position + new Vector2(50, 0),
                TabStop = true,
            };

            ControlManager.Add(_dexBtn);

            _cunBtn = new LinkLabel
            {
                Text = "+",
                Position = _cun.Position + new Vector2(50, 0),
                TabStop = true,
            };

            ControlManager.Add(_cunBtn);

            _magBtn = new LinkLabel
            {
                Text = "+",
                Position = _mag.Position + new Vector2(50, 0),
                TabStop = true,
            };

            ControlManager.Add(_magBtn);

            _wilBtn = new LinkLabel
            {
                Text = "+",
                Position = _wil.Position + new Vector2(50, 0),
                TabStop = true,
            };

            ControlManager.Add(_wilBtn);

            _conBtn = new LinkLabel
            {
                Text = "+",
                Position = _con.Position + new Vector2(50, 0),
                TabStop = true,
            };

            ControlManager.Add(_conBtn);

            _strBtn.Selected += StrBtn_Selected;
            _dexBtn.Selected += DexBtn_Selected;
            _cunBtn.Selected += CunBtn_Selected;
            _magBtn.Selected += MagBtn_Selected;
            _wilBtn.Selected += WilBtn_Selected;
            _conBtn.Selected += ConBtn_Selected;
        }

        private void StrBtn_Selected(object sender, EventArgs e)
        {
            if (_points > 0)
            {
                GamePlayScreen.Player.Character.Entity.AdjustAttribut("Strength", 1);
                _points--;
            }
        }

        private void DexBtn_Selected(object sender, EventArgs e)
        {
            if (_points > 0)
            {
                GamePlayScreen.Player.Character.Entity.AdjustAttribut("Dexterity", 1);
                _points--;
            }
        }

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

```
        {
            if (_points > 0)
            {
                GamePlayScreen.Player.Character.Entity.AdjustAttribut("Cunning", 1);
                _points--;
            }
        }

        private void MagBtn_Selected(object sender, EventArgs e)
        {
            if (_points > 0)
            {
                GamePlayScreen.Player.Character.Entity.AdjustAttribut("Magic", 1);
                _points--;
            }
        }

        private void WilBtn_Selected(object sender, EventArgs e)
        {
            if (_points > 0)
            {
                GamePlayScreen.Player.Character.Entity.AdjustAttribut("Will Power", 1);
                _points--;
            }
        }

        private void ConBtn_Selected(object sender, EventArgs e)
        {
            if (_points > 0)
            {
                GamePlayScreen.Player.Character.Entity.AdjustAttribut("Constitution", 1);
                _points--;
            }
        }

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

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

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

            _points = 3;
        }

        protected override void OnCloseSelected()
        {
            if (_points == 0)
            {
                base.OnCloseSelected();
            }
        }
    }
```

```
}
```

This class has using statements for some of the MGRpgLibrary classes and MonoGame classes. Since is shares the layout of the StatsScreen it inherits from StatsScreen instead of BaseGameState. For fields it has an integer, _points, which is the amount of points remaining to be spent on attributes. It also has link label fields for each of the attributes. They will display a + sign indicating that if it is tapped a point will be added to that attribute.

The LoadContent method first creates the LinkLabels setting the Text property, Position to the right of the attribute and TabStop property to true. The control is then added to the control manager. After creating the LinkLabels I wire event handlers for their Selected property.

The event handlers check to see if there are any points remaining to be spent. If there are, it calls the AdjustAttribute method on the Entity that we defined earlier. It then decrements the number of points available to be spent.

There are overrides of the Update and Draw methods that currently do nothing but may be helpful in the future so I included them now. I also included an override of the Show method that sets the _points field to 3. The last override is of the OnCloseSelected method. It will call the base method only if there are no points remaining.

The next step will be to create an instance of the LevelScreen in the Game1 class. Add the following property to the Game1 class and initialize it in the constructor.

```csharp
public LevelScreen LevelScreen { get; private set; }

public Game1()
{
    _graphics = new GraphicsDeviceManager(this);
    ScreenRectangle = new Rectangle(
        0,
        0,
        ScreenWidth,
        ScreenHeight);
    IsMouseVisible = true;

    Content.RootDirectory = "Content";

    Components.Add(new InputHandler(this));

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

    _ = new TextureManager();

    TitleScreen = new TitleScreen(this, _gameStateManager);
    StartMenuScreen = new StartMenuScreen(this, _gameStateManager);
    GamePlayScreen = new GamePlayScreen(this, _gameStateManager);
    CharacterGeneratorScreen = new CharacterGeneratorScreen(this, _gameStateManager);
    SkillScreen = new SkillScreen(this, _gameStateManager);
    LoadGameScreen = new LoadGameScreen(this, _gameStateManager);
    ConversationScreen = new ConversationScreen(this, _gameStateManager);
    ShopScreen = new ShopState(this, _gameStateManager);
    InventoryScreen = new InventoryScreen(this, _gameStateManager);
    CombatScreen = new CombatScreen(this, _gameStateManager);
    GameOverScreen = new GameOverScreen(this, _gameStateManager);
    LootScreen = new LootScreen(this, _gameStateManager);
    StatsScreen = new StatsScreen(this, _gameStateManager);
    LevelScreen = new LevelScreen(this, _gameStateManager);

    _gameStateManager.ChangeState(TitleScreen);

    IsFixedTimeStep = false;
    _graphics.SynchronizeWithVerticalRetrace = false;
}
```

The last thing to do is display the LevelScreen when the player can level up. I did that in the Update method of the GamePlayScreen. Change that method to the following.

```
public override void Update(GameTime gameTime)
{
    world.Update(gameTime);
    player.Update(gameTime);
    player.Camera.LockToSprite(player.Sprite);

    if (InputHandler.KeyReleased(Keys.Space) ||
        InputHandler.ButtonReleased(Buttons.A, PlayerIndex.One))
    {
        foreach (ILayer layer in World.Levels[World.CurrentLevel].Map.Layers)
        {
            if (layer is CharacterLayer)
            {
                foreach (Character c in ((CharacterLayer)layer).Characters.Values)
                {
                    float distance = Vector2.Distance(
                        player.Sprite.Center,
                        c.Sprite.Center);

                    if (distance < Character.SpeakingRadius && c is NonPlayerCharacter)
                    {
                        NonPlayerCharacter npc = (NonPlayerCharacter)c;

                        if (npc.HasConversation)
                        {
                            StateManager.PushState(GameRef.ConversationScreen);

                            GameRef.ConversationScreen.SetConversation(
                                player,
                                npc,
                                npc.CurrentConversation);

                            GameRef.ConversationScreen.StartConversation();
                        }
                    }
                    else if (distance < Character.SpeakingRadius && c is Merchant)
                    {
                        StateManager.PushState(GameRef.ShopScreen);
                        GameRef.ShopScreen.SetMerchant(c as Merchant);
                    }
                }
            }
        }
    }

    MobLayer mobLayer = World.Levels[World.CurrentLevel].Map.Layers.Find(x => x is MobLayer) as
MobLayer;

    foreach (var mob in mobLayer.Mobs.Where(kv => kv.Value.Entity.Health.CurrentValue <=
0).ToList())
    {
        mobLayer.Mobs.Remove(mob.Key);
    }

    foreach (Rectangle r in mobLayer.Mobs.Keys)
    {
        float distance = Vector2.Distance(mobLayer.Mobs[r].Sprite.Center,
player.Sprite.Center);

        if (distance < Mob.AttackRadius)
        {
            GameRef.CombatScreen.SetMob(mobLayer.Mobs[r]);

            StateManager.PushState(GameRef.CombatScreen);
            Visible = true;
        }
    }

    if (InputHandler.KeyReleased(Keys.I))
    {
        StateManager.PushState(GameRef.InventoryScreen);
    }

    if (InputHandler.KeyReleased(Keys.C))
    {
        StateManager.PushState(GameRef.StatsScreen);
        Visible = true;
    }

    if (Player.Character.Entity.Level < Mechanics.Experiences.Length)
    {
        if (Player.Character.Entity.Experience >=
Mechanics.Experiences[Player.Character.Entity.Level])
        {
            Player.Character.Entity.LevelUp();
            StateManager.PushState(GameRef.LevelScreen);
            Visible = true;
        }
    }

    base.Update(gameTime);
}
```

What I do is check to see if the player's level is less than the length of the Experiences array that I added to the Mechanics class. If it is, I check to see if the player's experience is greater than or equal to the experience at the player's current level in the array. If it is, I call the LevelUp method of the Entity, push the LevelScreen on the stack and set the Visible property of the GamePlayScreen to true.

If you build and run now you can view the character's stats and if you defeat enough bandits you can level up the player.

So, that is going to be it for this tutorial. I accomplished what I intended and I don't want to venture further in this tutorial. So, please continue to visit my blog, https://cynthiamcmahon.ca/blog/, for the latest news on my tutorials and other goodness.

Good luck with your Game Programming Adventures!

Cynthia