A Summoner's Tale – MonoGame Tutorial Series

Chapter 13

Leveling Up

This tutorial series is about creating a Pokemon style game with the MonoGame Framework called A Summoner's Tale. The tutorials will make more sense if you read them in order as each tutorial builds on the previous tutorials. You can find the list of tutorials on my web site: A Summoner's Tale. The source code for each tutorial will be available as well. I will be using Visual Studio 2013 Premium for the series. The code should compile on the 2013 Express version and Visual Studio 2015 versions as well.

I want to mention though that the series is released as Creative Commons 3.0 Attribution. It means that you are free to use any of the code or graphics in your own game, even for commercial use, with attribution. Just add a link to my site, http://gameprogrammingadventures.org, and credit to Jamie McMahon.

What I am going to tackle first in this tutorial is to add the ability to level up avatars after a battle. There are two ways that you could handle an avatar levelling up. One ways is that you could increase the avatar's status automatically in code or you can give the player the ability to assign points to an avatar's stats. I personally like giving the player choices so I went with that route.

First, you will want to download the content that I used for this tutorial. You can download the content using this link A Summoner's Tale Content 13. First, let's add the background for the level up state. In the solution explorer expand the Content folder and then the GameScreens folder. Right click the GameScreens folder, select Add and then Existing Item. Add the levelup-menu.png to this folder. Now open the Content.mgcb by double clicking on it. Right click on the GameScreens folder select Add and then Existing Item. Add the levelup-menu.png that we just added to that folder. Hit <ctrl>+s to save the changes and then under the Build menu select Build to build the content.

The next thing that I want to do is add a new state for levelling up and avatar. Right click the GameStates folder, select Add and then Class. Name this class LevelUpState. Here is the code for that state.

```
using Avatars.AvatarComponents;
using Avatars.Components;
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 Avatars.GameStates
```

```
{
    public interface ILevelUpState
    {
        void SetAvatar(Avatar playerAvatar);
    }

    public class LevelUpState : BaseGameState, ILevelUpState
    {
        #region Field Region

        private Rectangle destination;
        private int points;
        private int selected;
        private SpriteFont font;
        private Avatar player;
        private Dictionary<string, int> attributes = new Dictionary<string, int>();
        private Dictionary<string, int> assignedTo = new Dictionary<string, int>();
        private Texture2D levelUpBackground;

        #endregion

        #region Property Region
        #endregion

        #region Constructor Region

        public LevelUpState(Game game)
            : base(game)
        {
            attributes.Add("Attack", 0);
            attributes.Add("Defense", 0);
            attributes.Add("Speed", 0);
            attributes.Add("Health", 0);
            attributes.Add("Done", 0);

            foreach (string s in attributes.Keys)
                assignedTo.Add(s, 0);
        }

        #endregion

        #region Method Region

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

        protected override void LoadContent()
        {
            levelUpBackground = GameRef.Content.Load<Texture2D>(
                @"GameScreens\levelup-menu");

            font = GameRef.Content.Load<SpriteFont>(@"Fonts\scenefont");

            destination = new Rectangle(
                (Game1.ScreenRectangle.Width - levelUpBackground.Width) / 2,
                (Game1.ScreenRectangle.Height - levelUpBackground.Height) / 2,
                levelUpBackground.Width,
                levelUpBackground.Height);

            base.LoadContent();
        }

        public override void Update(GameTime gameTime)
        {
```

```
            PlayerIndex index = PlayerIndex.One;
            int i = 0;
            string attribute = "";

            if (Xin.CheckKeyReleased(Keys.Down))
            {
                selected++;

                if (selected >= attributes.Count)
                    selected = attributes.Count - 1;
            }
            else if (Xin.CheckKeyReleased(Keys.Up))
            {
                selected--;

                if (selected < 0)
                    selected = 0;
            }

            if (Xin.CheckKeyReleased(Keys.Space) || Xin.CheckKeyReleased(Keys.Enter))
            {
                if (selected == 4 && points == 0)
                {
                    foreach (string s in assignedTo.Keys)
                    {
                        player.AssignPoint(s, assignedTo[s]);
                    }

                    manager.PopState();
                    manager.PopState();
                    manager.PopState();
                    return;
                }
            }

            int increment = 1;

            if (Xin.CheckKeyReleased(Keys.Right) && points > 0)
            {
                foreach (string s in assignedTo.Keys)
                {
                    if (s == "Done")
                        return;

                    if (i == selected)
                    {
                        attribute = s;
                        break;
                    }

                    i++;
                }


                if (attribute == "Health")
                    increment *= 5;

                points--;
                assignedTo[attribute] += increment;

                if (points == 0)
                    selected = 4;
            }
            else if (Xin.CheckKeyReleased(Keys.Left) && points <= 3)
            {
                foreach (string s in assignedTo.Keys)
```

```
            {
                if (s == "Done")
                    return;

                if (i == selected)
                {
                    attribute = s;
                    break;
                }

                i++;
            }

            if (assignedTo[attribute] != attributes[attribute])
            {
                if (attribute == "Health")
                    increment *= 5;

                points++;
                assignedTo[attribute] -= increment;
            }
        }

        base.Update(gameTime);
    }

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


        GameRef.SpriteBatch.Begin();
        GameRef.SpriteBatch.Draw(levelUpBackground, destination, Color.White);

        Vector2 textPosition = new Vector2(destination.X + 5, destination.Y + 5);

        GameRef.SpriteBatch.DrawString(font, player.Name, textPosition, Color.Black);
        textPosition.Y += font.LineSpacing * 2;

        int i = 0;

        foreach (string s in attributes.Keys)
        {
            Color tint = Color.Black;

            if (i == selected)
                tint = Color.Red;

            if (s != "Done")
            {
                GameRef.SpriteBatch.DrawString(font, s + ":", textPosition, tint);
                textPosition.X += 125;

                GameRef.SpriteBatch.DrawString(font, attributes[s].ToString(),
textPosition, tint);
                textPosition.X += 40;

                GameRef.SpriteBatch.DrawString(font, assignedTo[s].ToString(),
textPosition, tint);
                textPosition.X = destination.X + 5;

                textPosition.Y += font.LineSpacing;
            }
            else
            {
                GameRef.SpriteBatch.DrawString(font, "Done", textPosition, tint);
```

```
                    textPosition.Y += font.LineSpacing * 2;
                }
                i++;
            }

            GameRef.SpriteBatch.DrawString(font, points.ToString() + " point left.",
textPosition, Color.Black);
            GameRef.SpriteBatch.End();
        }

        public void SetAvatar(Avatar playerAvatar)
        {
            player = playerAvatar;

            attributes["Attack"] = player.BaseAttack;
            attributes["Defense"] = player.BaseDefense;
            attributes["Speed"] = player.BaseSpeed;
            attributes["Health"] = player.BaseHealth;

            assignedTo["Attack"] = player.BaseAttack;
            assignedTo["Defense"] = player.BaseDefense;
            assignedTo["Speed"] = player.BaseSpeed;
            assignedTo["Health"] = player.BaseHealth;

            points = 3;
            selected = 0;
        }

        #endregion
    }
}
```

First, there is an interface that I added for the state like the other game states. It has a single method in it, SetAvatar, that is used to pass the avatar that is to be levelled up to the state. The class then inherits from the BaseGameState abstract class so it can be used by the state manager and implements the interface that was just defined above.

For field in the class there is a Rectangle that will hold the destination of the level up state on the screen. Next there is a integer that holds the number of points that are available to be assigned to the avatar. There is also a field selected that holds the current option selected in the game state for assigning points. Since the state renders text there is a SpriteFont field for drawing text. There is also an Avatar type field that will hold the avatar that we will be updating. There are then two dictionaries that hold the attributes that the avatar currently has, attributes, and the assigned point, assignedTo. The last field that I included in this class is a field to hold the image for the level up state.

I added a single constructor to this game state. What it does is create the two dictionaries and initializes their values to 0. I also included a Done attribute that will be displayed with the other attributes that can be selected when all points have been assigned to.

In the LoadContent method I first load the image for the level up state into its field. I then load the font into its field as well. Next up I center the level up background image on the screen. Finally I call the LoadContent method on the base class to load any base content.

The way the level up state was designed to work is that there is the list of attributes and a Done option. When an attribute is selected if the player presses the left key the selected attribute will have an assigned point taken away and if the right key was selected a point will be added. Once all of the attribute points have been assigned they can choose the Done option to assign the points.

So, in the Update method I handle this logic. First, there are local variables to hold what the selected index and item are. I then check to see if the Down key was pressed. If it was pressed I increment the selected field of this class. I then check to see if that value is greater than or equal to the number of elements in the dictionary. If it is I reset selected back to the last element in the list. I do the same thing for the Up key but in reverse. I make sure that the value is never below 0.

After checking for up and down I check if the Space and Enter keys were pressed. If one of them was pressed I then check if the selected item is the last item, Done, and that there are no remaining points to be assigned. If those conditions are true I call the AssignPoint method on the player avatar to assign any points to that attribute. I then remove the states that were added to the stack.

There is then a local variable called increment. This value holds how many points are assigned based on the selected attribute. All attributes but health add 1 point to the selected attribute. The health attributes add 5 points to the avatar's health.

Now, I check to see if the Right key has been released and that there are free points to spend. If there are I loop through all of the keys in the assignedTo dictionary. If the selected option is Done then I exit the method. I then compare the variable i with the field selected. If they are the same I set attribute to be the current key. I then check if attribute is Health and if it is I multiply the value by 5. I then subtract 1 from the points that are available and increment that key in the dictionary. Finally, if there are no points left to assigned I set the selected attribute to the Done option.

I do the same thing in the case where I'm checking if the Left key was released if there are assigned points that can be moved. Next is the same loop that checks to see what attribute is currently selected and if it is Done exit the method. There is then an if statement that validates that the player can remove a point from that category.

The Draw method is pretty straight forward. First, draw the image for the background of the state. Next, create a local variable that determines where text will be positioned. Next I write the name of the avatar and then increment the position the next value will be written to. There is then a local variable i that is used for indexing items. In a foreach loop I go over the keys in the dictionary. I have a tint colour of Black that determines what colour to draw text in. If the current option is the selected option I change the tint colour to Red.

If the option is not the Done option I draw the attribute in the tint colour. I update the X value of the position to draw the individual parts and then reset it back to its default value and increment the Y spacing attribute. I then increase the i local variable and go onto the next iteration of the loop. If it is Done I draw Done and then increase the Y position so there is space between Done and the last bit of text to be drawn. The last bit of text to be draw is how many points are left to be assigned.

Finally there is a SetAvatar method that sets the Avatar being used in the LevelUpState. What it does is set the Avatar field player to be the avatar passed in. It then sets the values of the two dictionaries based off of that avatar. Finally it resets the points and selected fields.

Now that we have a LevelUpState class we need to add it to the game. Open the Game1.cs file. Where the other game states are add the following field.

```
        ILevelUpState levelUpState;
```

With the other game state properties add this property to expose the field to other classes.

```
        public ILevelUpState LevelUpState
        {
            get { return levelUpState; }
        }
```

Update the constructor to initialize the LevelUpState that we just created.

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

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

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

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

            this.IsMouseVisible = true;

            titleIntroState = new TitleIntroState(this);
            startMenuState = new MainMenuState(this);
            gamePlayState = new GamePlayState(this);
            conversationState = new ConversationState(this);
            battleState = new BattleState(this);
            battleOverState = new BattleOverState(this);
            damageState = new DamageState(this);
            levelUpState = new LevelUpState(this);

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

            characterManager = CharacterManager.Instance;
        }
```

The next step will be to call this state when a battle is over to see if the avatar needs to be levelled up. That will be done in the BattleOverState in the Update method. Update that method to the following.

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

            if (Xin.CheckKeyReleased(Keys.Space) || Xin.CheckKeyReleased(Keys.Enter))
            {
                if (levelUp)
                {
                    manager.PushState((LevelUpState)GameRef.LevelUpState,
PlayerIndexInControl);
                    GameRef.LevelUpState.SetAvatar(player);

                    this.Visible = true;
                }
                else
                {
                    manager.PopState();
```

```
                manager.PopState();
            }
        }

        base.Update(gameTime);
    }
```

All that the new code does is if the field levelUp is true is push the level up state on top of the stack of game states and then sets the avatar that levelled up to the current avatar in use. What I've found is the Wind avatar almost always wins the battle against the Fire avatar in my testing, which is part of the reason you need to do a lot of testing of your game play elements. You might think that something works fine but in reality it is broken and the player will not be able to get past a certain point. In order to trigger an avatar level up I modified the CheckLevelUp method of the Avatar class. Update that method to the following code.

```
    public bool CheckLevelUp()
    {
        bool leveled = false;

        if (experience >= 50 * (1 + (long)Math.Pow((level - 1), 2.5)))
        {
            leveled = true;
            level++;
        }

        return leveled;
    }
```

All that this does is lower the amount of experience required to reach level 2 so fighting and losing will still cause the player's avatar to level up. I also want to update the AssignPoint method of the Avatar class because I'm controlling the way points are assigned in the LevelUpState instead of the Avatar class. Update that method to the following.

```
    public void AssignPoint(string s, int p)
    {
        switch (s)
        {
            case "Attack":
                attack += p;
                break;
            case "Defense":
                defense += p;
                break;
            case "Speed":
                speed += p;
                break;
            case "Health":
                health += p;
                break;
        }
    }
```

All that changes here is that when I update the health attribute it just uses the value passed in instead of 5 times the value passed in.

There is one minor issue still. You can continuously battle the other characters and train your avatar infinitely. There would be diminishing returns because as you level up when you battle you will gain less and less experience. It would be better if once you battled them you could not battle them again, like in Pokemon. It raises an issue with my story line. If you are fighting with summoned beings, how are you going to handle random encounters? I will leave the latter for another tutorial but I will cover limiting the number of times you can battle an NPC.

In order to do that I added a new property to the ICharacter interface, Battled. Update that interface to the following.

```
public interface ICharacter
{
    string Name { get; }
    bool Battled { get; set; }
    AnimatedSprite Sprite { get; }
    Avatar BattleAvatar { get; }
    Avatar GiveAvatar { get; }
    string Conversation { get; }
    void SetConversation(string newConversation);
    void Update(GameTime gameTime);
    void Draw(GameTime gameTime, SpriteBatch spriteBatch);
}
```

The next step will be to update the Character class to include this update. In this case I'm just going to use an auto-property rather than having a field and work with the field using the property. Add the following line with the other properties in the Character class.

```
        public bool Battled { get; set; }
```

The last step will be that in the GamePlayState class when checking if the player triggered a battle check if the player has battled that character previously and if they have not go to the battle state. If they go to the battle state you need to update that property. Rather than paste the entire Update method I'm only pasting the if statement that checks if the B key was released.

```
            if (Xin.CheckKeyReleased(Keys.B))
            {
                foreach (string s in map.Characters.Keys)
                {
                    ICharacter c = CharacterManager.Instance.GetCharacter(s);
                    float distance = Vector2.Distance(player.Sprite.Center, c.Sprite.Center);

                    if (Math.Abs(distance) < 72f && !c.Battled)
                    {
                        GameRef.BattleState.SetAvatars(player.CurrentAvatar, c.BattleAvatar);
                        manager.PushState(
                            (BattleState)GameRef.BattleState,
                             PlayerIndexInControl);
                        c.Battled = true;
                    }
                }
            }
```

I'm going to end the tutorial here as I like to keep the tutorials to a reasonable length so there is not a lot of new code to digest at once. Please stay tuned for the next tutorial in this series. If you don't want to have to keep visiting the site to check for new tutorials you can sign up for my newsletter on

the site and get a weekly status update of all the news from Game Programming Adventures. You can also follow my tutorials on Twitter at https://twitter.com/GPAAdmi77640534.

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