A Summoner's Tale – MonoGame Tutorial Series

Chapter 10

Creating Avatars

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.

This tutorial will cover creating avatars and the moves that they can learn. The way that I implemented avatars in the demo that I created was as a CSV file. That data is read when the game loads and the avatars are then created from each line in the CSV file. This is what a sample avatar looked like in my demo.

```
Fire,Fire,100,1,12,8,10,50,Tackle:1,Block:1,Flare:2,None:100,None:100,None:100
```

The first value is the avatar's name, really creative wasn't I. The next value is the avatars element. The next value was how much it would cost to buy the avatar from a character. The next value is the avatar's level. The next four elements are the attack, defense, speed and health of the avatar. The next six parameters are the moves that the avatar knows or can learn. They consist of two values. The name of the move and the level at which it unlocks. I will implement this a little differently in the tutorial. I'm going to make the move list dynamic but still follow the same rule.

Before I implement this I want to add in a manager class to manage moves and avatars. Right click the AvatarComponents folder, select Add and then Class. Name this new class MoveManager. Here is the code.

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

namespace Avatars.AvatarComponents
{
    public static class MoveManager
    {
        #region Field Region
```

```
        private static Dictionary<string, IMove> allMoves = new Dictionary<string, IMove>();
        private static Random random = new Random();

        #endregion

        #region Property Region

        public static Random Random
        {
            get { return random; }
        }

        #endregion

        #region Constructor Region
        #endregion

        #region Method Region

        public static void FillMoves()
        {
            //AddMove(new Tackle());
            //AddMove(new Block());
            //AddMove(new Haste());
            //AddMove(new Bless());
            //AddMove(new Curse());
            //AddMove(new Heal());
            //AddMove(new Flare());
            //AddMove(new Shock());
            //AddMove(new Gust());
            //AddMove(new Frostbite());
            //AddMove(new Shade());
            //AddMove(new Burst());
            //AddMove(new RockThrow());
        }

        public static IMove GetMove(string name)
        {
            if (allMoves.ContainsKey(name))
                return (IMove)allMoves[name].Clone();

            return null;
        }

        public static void AddMove(IMove move)
        {
            if (!allMoves.ContainsKey(move.Name))
                allMoves.Add(move.Name, move);
        }

        #endregion
    }
}
```

This is like most of the other managers that we've implemented through out the game so far. It consists of a static member variable to hold all of the objects that are being managed, in this case IMove. I included a Random member variable that will be shared by all of the moves as well. This just helps to make sure that the random number generation is consistent across all objects. You can also specify a seed here so that the same number series is generated each time that you run the game. It would be a good idea to do that when in debug mode by not when in release mode. Another difference in this class is that I don't exposes the moves with a property. I force the use of the get and add methods to get and add moves to the manager. I included the FillMoves function that I created with all of the moves commented out, because they haven't been implemented yet and will

cause a compile time error since they could not be found.

GetMove is used to retrieve a move from the manager. Rather than just returning that object I return a clone of the object. If you don't anywhere that you modify that object it will affect all other variables that point to that object. This is because classes are reference types. That means that the variable does not contain the data, it points to a location in memory that contains the data.

The AddMove method works in reverse. It takes a move and checks to see if it has been added. If it has not yet been added it is added to the move collection.

The next thing that I want to add is a method to the Avatar class that will take a string in the format above and return an Avatar object. Add the following method to the bottom of the Avatar class.

```
public static Avatar FromString(string description, ContentManager content)
{
    Avatar avatar = new Avatar();
    string[] parts = description.Split(',');

    avatar.name = parts[0];
    avatar.texture = content.Load<Texture2D>(@"AvatarImages\" + parts[0]);
    avatar.element = (AvatarElement)Enum.Parse(typeof(AvatarElement), parts[1]);
    avatar.costToBuy = int.Parse(parts[2]);
    avatar.level = int.Parse(parts[3]);
    avatar.attack = int.Parse(parts[4]);
    avatar.defense = int.Parse(parts[5]);
    avatar.speed = int.Parse(parts[6]);
    avatar.health = int.Parse(parts[7]);
    avatar.currentHealth = avatar.health;

    avatar.knownMoves = new Dictionary<string, IMove>();

    for (int i = 8; i < parts.Length; i++)
    {
        string[] moveParts = parts[i].Split(':');

        if (moveParts[0] != "None")
        {
            IMove move = MoveManager.GetMove(moveParts[0]);
            move.UnlockedAt = int.Parse(moveParts[1]);

            if (move.UnlockedAt <= avatar.Level)
                move.Unlock();

            avatar.knownMoves.Add(move.Name, move);
        }
    }

    return avatar;
}
```

The method accepts two parameters. It accepts a string that describes the avatar and a ContentManager to load the avatar's image. It first creates a new Avatar object using the private constructor that was added to the class. I then call the Split method on the description to break it up into its individual parts.

The next series of lines assign the value of the Avatar object using the parts. The texture is assigned by loading the image using the content manager that was passed in. Up until I assign the currentHealth member variable I use the Parse method of the individual items to get the associated

values. For the element I had to specify the type of the enumeration that holds the elements and the value. The currentHealth member is assigned the value of the health member.

The next step is to create the Dictionary that will hold the moves that the avatar knows. Next up is a loop that will loop through the remaining parts of the string and get the moves for the avatar. The first step is to split the string into parts base on the colon. If the first part of the string is None then skip it. I then use the GetMove method of the MoveManager to get the move. I then use the second part to determine what level the move unlocks at. If the avatar's level is greater than or equal to that I unlock the move. Finally the move is added to the dictionary of moves for the avatar. The avatar is then returned to the calling method.

As you saw from the MoveManager I had created several moves for the demo that I did. Since moves implement the IMove interface most of the code is identical. The differences are in the constructors and the Clone methods. I'm going to implement two moves in this tutorial. For the other moves I suggest that you download the source code for them.

What I am going to implement are a basic attack move, Tackle, and a basic block move, Block. Let's start with Tackle. Right click the AvatarComponents folder, select Add and then Class. Name this new class Tackle. Here is the code for that class.

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

namespace Avatars.AvatarComponents
{
    public class Tackle : IMove
    {
        #region Field Region

        private string name;
        private Target target;
        private MoveType moveType;
        private MoveElement moveElement;
        private Status status;
        private bool unlocked;
        private int unlockedAt;
        private int duration;
        private int attack;
        private int defense;
        private int speed;
        private int health;

        #endregion

        #region Property Region

        public string Name
        {
            get { return name; }
        }

        public Target Target
        {
            get { return target; }
        }

        public MoveType MoveType
        {
```

```
            get { return moveType; }
        }

        public MoveElement MoveElement
        {
            get { return moveElement; }
        }

        public Status Status
        {
            get { return status; }
        }

        public int UnlockedAt
        {
            get { return unlockedAt; }
            set { unlockedAt = value; }
        }

        public bool Unlocked
        {
            get { return unlocked; }
        }

        public int Duration
        {
            get { return duration; }
            set { duration = value; }
        }

        public int Attack
        {
            get { return attack; }
        }

        public int Defense
        {
            get { return defense; }
        }

        public int Speed
        {
            get { return speed; }
        }

        public int Health
        {
            get { return health; }
        }

        #endregion

        #region Constructor region

        public Tackle()
        {
            name = "Tackle";
            target = Target.Enemy;
            moveType = MoveType.Attack;
            moveElement = MoveElement.None;
            status = Status.Normal;
            duration = 1;
            unlocked = false;
            attack = MoveManager.Random.Next(0, 0);
            defense = MoveManager.Random.Next(0, 0);
            speed = MoveManager.Random.Next(0, 0);
```

```
            health = MoveManager.Random.Next(10, 15);
        }

        #endregion

        #region Method Region

        public void Unlock()
        {
            unlocked = true;
        }

        public object Clone()
        {
            Tackle tackle = new Tackle();
            tackle.unlocked = this.unlocked;
            return tackle;
        }

        #endregion
    }
}
```

There are member variables to expose each of the properties that must be implemented for the IMove interface. I will run over what they are briefly. Name is the name that will be displayed when the avatar uses this move. Target is what the move targets. MoveType is what kind of move it is. MoveElement is the element associated with the move. Status determines if there is a status change for the move. Unlocked is if the move is unlocked or not. Duration is how long the move lasts. Moves that have a duration of 1 take place that round. Moves with a duration greater than 1 last that many rouds. Attack is applied to the target's attack attribute. Defense, Speed and Health are applied to their respective attributes.

The constructors set the properties for the move. For tackle, the name is set to Tackle, the target is an enemy, the type is attack, the element is none, the status is normal, the duration is 1, as explained earlier that means it is applied immediately and initially the move is locked. Next I up I assign attack, defense, speed and health random values. In this case an attack damages an opponent so I generate a number between 10 and 14 because the upper limit is exclusive for this method of the Random class. The Clone method creates a new Tackle object and assigns it's unlocked value to the object returned.

Next up I'm going to add in the Block class. This move is a buff for the avatar and increase's its defense score for a few rounds. Right click the AvatarComponents folder, select Add and then Class. Name this new class Block. Here is the code for that class.

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

namespace Avatars.AvatarComponents
{
    public class Block : IMove
    {
        #region Field Region

        private string name;
        private Target target;
        private MoveType moveType;
        private MoveElement moveElement;
```

```
        private Status status;
        private bool unlocked;
        private int unlockedAt;
        private int duration;
        private int attack;
        private int defense;
        private int speed;
        private int health;

        #endregion

        #region Property Region

        public string Name
        {
            get { return name; }
        }

        public Target Target
        {
            get { return target; }
        }

        public MoveType MoveType
        {
            get { return moveType; }
        }

        public MoveElement MoveElement
        {
            get { return moveElement; }
        }

        public Status Status
        {
            get { return status; }
        }

        public int UnlockedAt
        {
            get { return unlockedAt; }
            set { unlockedAt = value; }
        }

        public bool Unlocked
        {
            get { return unlocked; }
        }

        public int Duration
        {
            get { return duration; }
            set { duration = value; }
        }

        public int Attack
        {
            get { return attack; }
        }

        public int Defense
        {
            get { return defense; }
        }

        public int Speed
```

```
        {
            get { return speed; }
        }

        public int Health
        {
            get { return health; }
        }

        #endregion

        #region Constructor Region

        public Block()
        {
            name = "Block";
            target = Target.Self;
            moveType = MoveType.Buff;
            moveElement = MoveElement.None;
            status = Status.Normal;
            unlocked = false;
            duration = 5;
            attack = MoveManager.Random.Next(0, 0);
            defense = MoveManager.Random.Next(2, 6);
            speed = MoveManager.Random.Next(0, 0);
            health = MoveManager.Random.Next(0, 0);
        }

        #endregion

        #region Method Region

        public void Unlock()
        {
            unlocked = true;
        }

        public object Clone()
        {
            Block block = new Block();
            block.unlocked = this.unlocked;
            return block;
        }

        #endregion
    }
}
```

Until you get to the constructor the code is identical to the Tackle move. Once you hit the constructor just a few of the properties change. The Name property changes to Block, the target to self, the type to buff, the duration to 5 and the random numbers generated. In this case Health is 0 and defense is between 2 and 5, because 6 is excluded from that range. The next difference is not until the Clone method where I created a Block object to return rather than a Tackle object.

At the time that I implemented this I was on a time restriction that I needed to finish the game for submission to the challenge I was building for. In hind sight this could have been better designed, similar to how I created avatars using strings. It would be idea to modify this so that you have perhaps one class that all moves share rather than a class for each move. I will leave that as an exercise, unless I get requests to demonstrate how you could do that.

So now I'm going to add some avatars to the game. I apologize for the lack of imagination when it

comes to their names in advance. The first step will be to right click the Avatars project, select Add and then Folder. Name this new folder Data. Right click this new Data folder, select Add and then New Item. From the list of templates choose Textfile and name it Avatars.csv. Next, select the Avatars.csv file in the solution. In the properties window for the file change the Copy to Output Directory property to Copy always. This will ensure that the file is copied to the output directory for testing. Copy and paste the following lines into the Avatars.csv file.

```
Dark,Dark,100,1,9,12,10,50,Tackle:1,Block:1
Earth,Earth,100,1,10,10,9,60,Tackle:1,Block:1
Fire,Fire,100,1,12,8,10,50,Tackle:1,Block:1
Light,Light,100,1,12,9,10,50,Tackle:1,Block:1
Water,Water,100,1,9,12,10,50,Tackle:1,Block:1
Wind,Wind,100,1,10,10,12,50,Tackle:1,Block:1
```

All of the avatars start out basically the same. Each of them costs 100 gold to purchase and their levels start at 1.The next few properties are their attack, defense, speed and health. I tried to make each of them slightly different than the others. For example, in my description I said that earth avatars are strong but slow. For that reason I reduces their speed to 9 but increased their health to 60. I would recommend in your own games that you play test the different avatars to make sure that you don't end up with a "broken" avatar that your players will always use because they can't be beaten by other avatars.

The next step will be load these into the game. I did that by adding in a class to manage the avatars in the game. To this class I added a FromFile method that read the file line by line and created the avatars from each line. Let's add that to the game now. Right click the AvatarComponents folder, select Add and then Class. Name this new class AvatarManager. Here is the code for that class.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.IO;
using Microsoft.Xna.Framework.Content;

namespace Avatars.AvatarComponents
{
    public static class AvatarManager
    {
        #region Field Region

        private static Dictionary<string, Avatar> avatarList = new Dictionary<string,
Avatar>();

        #endregion

        #region Property Region

        public static Dictionary<string, Avatar> AvatarList
        {
            get { return avatarList; }
        }

        #endregion

        #region Constructor Region

        #endregion

        #region Method Region
```

```
        public static void AddAvatar(string name, Avatar avatar)
        {
            if (!avatarList.ContainsKey(name))
                avatarList.Add(name, avatar);
        }

        public static Avatar GetAvatar(string name)
        {
            if (avatarList.ContainsKey(name))
                return (Avatar)avatarList[name].Clone();

            return null;
        }

        public static void FromFile(string fileName, ContentManager content)
        {
            using (Stream stream = new FileStream(fileName, FileMode.Open, FileAccess.Read))
            {
                try
                {
                    using (TextReader reader = new StreamReader(stream))
                    {
                        try
                        {
                            string lineIn = "";

                            do
                            {
                                lineIn = reader.ReadLine();
                                if (lineIn != null)
                                {
                                    Avatar avatar = Avatar.FromString(lineIn, content);
                                    if (!avatarList.ContainsKey(avatar.Name))
                                        avatarList.Add(avatar.Name, avatar);
                                }
                            } while (lineIn != null);
                        }
                        catch
                        {
                        }
                        finally
                        {
                            if (reader != null)
                                reader.Close();
                        }
                    }
                }
                catch
                {
                }
                finally
                {
                    if (stream != null)
                        stream.Close();
                }
            }
        }

        #endregion
    }
}
```

This manager is similar to all other manager classes. It maintains a dictionary for the objects to be managed and has a mechanism to add and retrieve objects. What is different about this one is that it has a FromFile method that will open the file, read the avatars from the file and add them to the

dictionary. The FromFile method accepts the name of the file to be read and ContentManager to load the images for the avatars.

In order to read from a file you need a Stream object. In this case I created a FileStream as my content was stored on disk. There are other types of streams that you can create in C, such as a NetworkStream that could be used to pull the file from a server if you were to implement your game on Android or iOS. This would prevent cheating where players modify your content locally. Once the stream is open I create a TextReader using the stream to read the text file. I will be loading each line into a variable, lineIn. Next up is a do-while loop that loops for as long as there is data in the file. Each pass through I try to read the next line of the file using ReadLine. If that is not null then data was read in. I then create an Avatar object using the FromString method that I showed earlier in the tutorial. Then if there is not already an avatar with that name in the collection I call the Add method to add the avatar to the collection.

There is one last piece of the puzzle missing. There are no images for the for the avatars so when you try to read them in the game will crash. For my demo I grabbed the six images from the YuGiOh card game. They will be good enough for this tutorial as well. You can find my images at this link.

Once you've downloaded and extracted the files open the MonoGame pipeline again. Select the Content folder then click the New Folder icon in the tool bar. Name this new folder AvatarImages. Now select the AvatarImages folder then click the Add Existing Item button in the ribbon. Add the images, dark.PNG, earth.PNG, fire.PNG, light.PNG, water.PNG and wind.PNG. Once the images have been added save the project, click Build from the menu and select Build to build the content project then close the window.

Now we can load the avatars into the game. First, in the MoveManager update the FillMoves folder to create the Tackle and Block moves. If you added the other moves you can create them as well. Now, open the Game1 class and update the LoadContent method to the following.

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

    AvatarComponents.MoveManager.FillMoves();
    AvatarComponents.AvatarManager.FromFile(@".\Data\avatars.csv", Content);
}
```

What this does is create the moves for the avatars and then loads the avatars from the file that we created. At this point you will be able to build and run the game with out problems.

I'm going to end the tutorial here as it is a lot to digest in one sitting. 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