A Summoner's Tale – MonoGame Tutorial Series

Chapter 12

Battling Avatars Continued

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.

In this tutorial I'm going to pick up where I left off in the last tutorial in battling avatars. In this tutorial I will add in the next steps to actually battle the two avatars, such as selecting moves and having them applied to the target. I won't be covering the player and opponent having multiple avatars and will focus and just a single avatar. I will write a separate tutorial on how to add that functionality.

In the last tutorial I had added a game state called BattleState. This scene will display the available moves and allow the player to choose the move that they want. The options for this will need to be set each time the SetAvatars is called. Update that method to the following.

```
public void SetAvatars(Avatar player, Avatar enemy)
{
    this.player = player;
    this.enemy = enemy;

    player.StartCombat();
    enemy.StartCombat();

    List<SceneOption> moves = new List<SceneOption>();

    if (combatScene == null)
        LoadContent();

    foreach (string s in player.KnownMoves.Keys)
    {
        SceneOption option = new SceneOption(s, s, new SceneAction());
        moves.Add(option);
    }

    combatScene.Options = moves;
}
```

The change from last time is I built a List<SceneOption> that holds the moves for the avatar. I then check if the combatScene member variable is null and if it is I call LoadContent to create it. I had to do that because since I did not add the game component to the list of components in the game the LoadContent method is not automatically called.

In a foreach loop I loop over all of the keys in the KnownMoves dictionary. Inside I create a new scene options and add it to the list of moves. Before exiting the method I assign the scene options to the list that I just created.

I have updated the LoadContent method to handle the problem where it is not called automatically. All I did was have an if statement to make sure the combatScene is null before trying to load. Update the LoadContent method to the following.

```
protected override void LoadContent()
{
    if (combatScene == null)
    {
        combatBackground = GameRef.Content.Load<Texture2D>(@"Scenes\scenebackground");

        avatarFont = GameRef.Content.Load<SpriteFont>(@"Fonts\scenefont");
        avatarBorder = GameRef.Content.Load<Texture2D>(@"Misc\avatarborder");
        avatarHealth = GameRef.Content.Load<Texture2D>(@"Misc\avatarhealth");

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

        combatScene = new GameScene(GameRef, "", new List<SceneOption>());
    }

    base.LoadContent();
}
```

Next would be to wire the scene to perform the action the player selects and apply it to the enemy. First, I want to add in two game states. One state will be displayed when the battle is over. The other will be displayed while the moves of both avatars are being resolved for the current turn.

First, I am going to add the battle over state. Right click the GameScenes folder, select Add and then Class. Name the class BattleOverState. Here is the code.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Avatars.AvatarComponents;
using Avatars.GameStates;
using Avatars.Components;

namespace Avatars.GameStates
{
    public interface IBattleOverState
    {
        void SetAvatars(Avatar player, Avatar enemy);
    }

    public class BattleOverState : BaseGameState, IBattleOverState
    {
        #region Field Region
        private Avatar player;
        private Avatar enemy;
        private Texture2D combatBackground;
        private Rectangle playerRect;
        private Rectangle enemyRect;
        private Rectangle playerBorderRect;
        private Rectangle enemyBorderRect;
        private Rectangle playerMiniRect;
        private Rectangle enemyMiniRect;
        private Rectangle playerHealthRect;
        private Rectangle enemyHealthRect;
        private Rectangle healthSourceRect;
        private Vector2 playerName;
        private Vector2 enemyName;
        private float playerHealth;
        private float enemyHealth;
        private Texture2D avatarBorder;
        private Texture2D avatarHealth;
        private SpriteFont avatarFont;
        private SpriteFont font;
        private string[] battleState;
        private Vector2 battlePosition;
        private bool levelUp;

        #endregion

        #region Property Region
        #endregion

        #region Constructor Region

        public BattleOverState(Game game)
            : base(game)
        {
            battleState = new string[3];

            battleState[0] = "The battle was won!";
            battleState[1] = " gained ";
            battleState[2] = "Continue";

            battlePosition = new Vector2(25, 475);

            playerRect = new Rectangle(10, 90, 300, 300);
            enemyRect = new Rectangle(Game1.ScreenRectangle.Width - 310, 10, 300, 300);

            playerBorderRect = new Rectangle(10, 10, 300, 75);
            enemyBorderRect = new Rectangle(Game1.ScreenRectangle.Width - 310, 320, 300,
75);

            healthSourceRect = new Rectangle(10, 50, 290, 20);
            playerHealthRect = new Rectangle(playerBorderRect.X + 12, playerBorderRect.Y +
52, 286, 16);
            enemyHealthRect = new Rectangle(enemyBorderRect.X + 12, enemyBorderRect.Y + 52,
286, 16);

            playerMiniRect = new Rectangle(playerBorderRect.X + 11, playerBorderRect.Y + 11,
28, 28);
            enemyMiniRect = new Rectangle(enemyBorderRect.X + 11, enemyBorderRect.Y + 11,
28, 28);

            playerName = new Vector2(playerBorderRect.X + 55, playerBorderRect.Y + 5);
            enemyName = new Vector2(enemyBorderRect.X + 55, enemyBorderRect.Y + 5);
        }

        #endregion
        #region Method Region

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

        protected override void LoadContent()
        {
            combatBackground = GameRef.Content.Load<Texture2D>(@"Scenes\scenebackground");

            avatarFont = GameRef.Content.Load<SpriteFont>(@"Fonts\GameFont");
            avatarBorder = GameRef.Content.Load<Texture2D>(@"Misc\avatarborder");
            avatarHealth = GameRef.Content.Load<Texture2D>(@"Misc\avatarhealth");

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

            base.LoadContent();
        }

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

            if (Xin.CheckKeyReleased(Keys.Space) || Xin.CheckKeyReleased(Keys.Enter))
            {
                if (levelUp)
                {
                    this.Visible = true;
                }
                else
                {
                    manager.PopState();
                    manager.PopState();
                }
            }

            base.Update(gameTime);
        }

        public override void Draw(GameTime gameTime)
        {
            Vector2 position = battlePosition;

            base.Draw(gameTime);

            GameRef.SpriteBatch.Begin();

            GameRef.SpriteBatch.Draw(combatBackground, Vector2.Zero, Color.White);

            for (int i = 0; i < 2; i++)
            {
                GameRef.SpriteBatch.DrawString(font, battleState[i], position, Color.Black);
                position.Y += avatarFont.LineSpacing;
            }

            GameRef.SpriteBatch.DrawString(font, battleState[2], position, Color.Red);

            GameRef.SpriteBatch.Draw(player.Texture, playerRect, Color.White);
            GameRef.SpriteBatch.Draw(enemy.Texture, enemyRect, Color.White);

            GameRef.SpriteBatch.Draw(avatarBorder, playerBorderRect, Color.White);

            playerHealth = (float)player.CurrentHealth / (float)player.GetHealth();
            MathHelper.Clamp(playerHealth, 0f, 1f);
            playerHealthRect.Width = (int)(playerHealth * 286);
            GameRef.SpriteBatch.Draw(avatarHealth, playerHealthRect, healthSourceRect,
Color.White);

            GameRef.SpriteBatch.Draw(avatarBorder, enemyBorderRect, Color.White);

            enemyHealth = (float)enemy.CurrentHealth / (float)enemy.GetHealth();
            MathHelper.Clamp(enemyHealth, 0f, 1f);
            enemyHealthRect.Width = (int)(enemyHealth * 286);

            GameRef.SpriteBatch.Draw(avatarHealth, enemyHealthRect, healthSourceRect,
Color.White);
            GameRef.SpriteBatch.DrawString(avatarFont, player.Name, playerName,
Color.White);
            GameRef.SpriteBatch.DrawString(avatarFont, enemy.Name, enemyName, Color.White);

            GameRef.SpriteBatch.Draw(player.Texture, playerMiniRect, Color.White);
            GameRef.SpriteBatch.Draw(enemy.Texture, enemyMiniRect, Color.White);

            GameRef.SpriteBatch.End();
        }

        public void SetAvatars(Avatar player, Avatar enemy)
        {
            levelUp = false;

            long expGained = 0;

            this.player = player;
            this.enemy = enemy;

            if (player.Alive)
            {
                expGained = player.WinBattle(enemy);
                battleState[0] = player.Name + " has won the battle!";
                battleState[1] = player.Name + " has gained " + expGained + " experience";

                if (player.CheckLevelUp())
                {
                    battleState[1] += " and gained a level!";

                    foreach (string s in player.KnownMoves.Keys)
                    {
                        if (player.KnownMoves[s].Unlocked == false && player.Level >=
player.KnownMoves[s].UnlockedAt)
                        {
                            player.KnownMoves[s].Unlock();
                            battleState[1] += " " + s + " was unlocked!";
                        }
                    }

                    levelUp = true;
                }
                else
                {
                    battleState[1] += ".";
                }
            }
            else
            {
                expGained = player.LoseBattle(enemy);

                battleState[0] = player.Name + " has lost the battle.";
                battleState[1] = player.Name + " has gained " + expGained + " experience";

                if (player.CheckLevelUp())
                {
                    battleState[1] += " and gained a level!";

                    foreach (string s in player.KnownMoves.Keys)
                    {
                        if (player.KnownMoves[s].Unlocked == false && player.Level >=
player.KnownMoves[s].UnlockedAt)
                        {
                            player.KnownMoves[s].Unlock();
                            battleState[1] += " " + s + " was unlocked!";
                        }
                    }

                    levelUp = true;
                }
                else
                {
                    battleState[1] += ".";
                }
            }
        }

        #endregion
    }
}
```

I included an interface for this class, IBattleOverState, that includes one method that must be implemented SetAvatars. This method is used just like the one in IBattleState for setting the battle avatars.

Like the BattleState this class has a lot of member variables for drawing and positioning graphical elements. There are member variables for both avatars as well. I included an array of strings called battleState. These strings are used to build and display the results of the battle to the player. I also included a member variable for positioning this element on the screen. The last member variable is a bool that measures if the avatar has leveled up or not.

The constructor just initializes member variables. I also create an array to hold some of the predefined strings and assign them values.

This time I didn't wrap loading the content into an if statement. The reason why is in the previous game state I was initializing values right after calling SetAvatar. In this case the load does not need to be done automatically and is deferred until after all creation has finished.

In the Update method I have the PlayerIndex variable that you have seen through out all of the game states so far. Next is an if statement where I check if the space key or enter key have been released. There is then an if statement that checks if the levelUp member variable is true or not. If it is I just set the Visible property of the game state to true. Later on in the tutorial I will be adding a level up state. If it was false I call PopState twice. The reason will be that multiple states will need to be popped of the stack to get back to the game play state.

Like in the other states the Draw method just positions all of the elements. What is different is that I also position the text displaying the battle state. First, I loop over all of the elements for the state and draw them at the desired position. I then update the Y position by the LineSpacing property for the font to move onto the next line.

In the SetAvatars method I determine what needs to be rendered by the scene. I check the Alive property of the player's avatar to determine if that avatar won the battle. If it did I call the WinBattle method passing in the enemy after to calculate how much experience the avatar gained for defeating this avatar. I then update the battle strings with these values. I next call the CheckLevelUp method on the player avatar to check if the avatar has levelled up or not. If it has I update the string being displayed. I then loop through all of the known moves and unlock any moves that unlock at the new level. I also append text to the display that the move was unlocked. Finally I set the levelUp member variable to true. If it did not level up I just append a period.

If the player's avatar lost I call the LostBattle method to assign it the experience it learned during the battle. I then constructor the strings that will display that the battle was lost. I also go over the same process of checking if the avatar levelled up or not.

Next up I want to add in the damage state. This state will resolve the selected moves for the avatars. Right click the GameStates folder, select Add and then Class. Name this new class DamageState. Here is the code for that class.

```
using Avatars.AvatarComponents;
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 Avatars.GameStates
{
    public enum CurrentTurn
    {
        Players, Enemies
    }

    public interface IDamageState
    {
        void SetAvatars(Avatar player, Avatar enemy);
        void SetMoves(IMove playerMove, IMove enemyMove);
        void Start();
    }

    public class DamageState : BaseGameState, IDamageState
    {
        #region Field Region

        private CurrentTurn turn;
        private Texture2D combatBackground;
        private SpriteFont avatarFont;
        private SpriteFont font;
        private Rectangle playerRect;
        private Rectangle enemyRect;
        private TimeSpan cTimer;
        private TimeSpan dTimer;
        private Avatar player;
        private Avatar enemy;
        private IMove playerMove;
        private IMove enemyMove;
        private bool first;
        private Rectangle playerBorderRect;
        private Rectangle enemyBorderRect;
        private Rectangle playerMiniRect;
        private Rectangle enemyMiniRect;
        private Rectangle playerHealthRect;
        private Rectangle enemyHealthRect;
        private Rectangle healthSourceRect;
        private float playerHealth;
        private float enemyHealth;
        private Texture2D avatarBorder;
        private Texture2D avatarHealth;
        private Vector2 playerName;
        private Vector2 enemyName;

        #endregion

        #region Property Region
        #endregion

        #region Constructor Region

        public DamageState(Game game)
            : base(game)
        {
            playerRect = new Rectangle(10, 90, 300, 300);
            enemyRect = new Rectangle(Game1.ScreenRectangle.Width - 310, 10, 300, 300);

            playerBorderRect = new Rectangle(10, 10, 300, 75);
            enemyBorderRect = new Rectangle(Game1.ScreenRectangle.Width - 310, 320, 300, 
75);

            healthSourceRect = new Rectangle(10, 50, 290, 20);
            playerHealthRect = new Rectangle(playerBorderRect.X + 12, playerBorderRect.Y + 
52, 286, 16);
            enemyHealthRect = new Rectangle(enemyBorderRect.X + 12, enemyBorderRect.Y + 52, 
286, 16);

            playerMiniRect = new Rectangle(playerBorderRect.X + 11, playerBorderRect.Y + 11, 
28, 28);
            enemyMiniRect = new Rectangle(enemyBorderRect.X + 11, enemyBorderRect.Y + 11, 
28, 28);

            playerName = new Vector2(playerBorderRect.X + 55, playerBorderRect.Y + 5);
            enemyName = new Vector2(enemyBorderRect.X + 55, enemyBorderRect.Y + 5);
        }

        #endregion

        #region Method Region

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

        protected override void LoadContent()
        {
            combatBackground = GameRef.Content.Load<Texture2D>(@"Scenes\scenebackground");

            avatarBorder = GameRef.Content.Load<Texture2D>(@"Misc\avatarborder");
            avatarHealth = GameRef.Content.Load<Texture2D>(@"Misc\avatarhealth");

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

            base.LoadContent();
        }

        public override void Update(GameTime gameTime)
        {
            PlayerIndex index;
            if ((cTimer > TimeSpan.FromSeconds(4) || !enemy.Alive || !player.Alive) &&
dTimer > TimeSpan.FromSeconds(3))
            {
                if (!enemy.Alive || !player.Alive)
                {
                    manager.PopState();
                    manager.PushState((BattleOverState)GameRef.BattleOverState,
PlayerIndex.One);
                    GameRef.BattleOverState.SetAvatars(player, enemy);
                }
                else
                {
                    manager.PopState();
                }
            }
            else if (cTimer > TimeSpan.FromSeconds(2) && first && enemy.Alive &&
player.Alive)
            {
                first = false;
                dTimer = TimeSpan.Zero;
                if (turn == CurrentTurn.Players)
                {
                    turn = CurrentTurn.Enemies;
                    enemy.ResoleveMove(enemyMove, player);
                }
                else
                {
                    turn = CurrentTurn.Players;
                    player.ResoleveMove(playerMove, enemy);
                }
            }
            else if (cTimer == TimeSpan.Zero)
            {
                dTimer = TimeSpan.Zero;
                if (turn == CurrentTurn.Players)
                {
                    player.ResoleveMove(playerMove, enemy);
                }
                else
                {
                    enemy.ResoleveMove(enemyMove, player);
                }
            }

            cTimer += gameTime.ElapsedGameTime;
            dTimer += gameTime.ElapsedGameTime;

            base.Update(gameTime);
        }

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

            GameRef.SpriteBatch.Begin();

            GameRef.SpriteBatch.Draw(combatBackground, Vector2.Zero, Color.White);

            GameRef.SpriteBatch.Draw(player.Texture, playerRect, Color.White);
            GameRef.SpriteBatch.Draw(enemy.Texture, enemyRect, Color.White);

            Vector2 location = new Vector2(25, 475);

            if (turn == CurrentTurn.Players)
            {
                GameRef.SpriteBatch.DrawString(font, player.Name + " uses " +
playerMove.Name + ".", location, Color.Black);

                if (playerMove.Target == Target.Enemy && playerMove.MoveType ==
MoveType.Attack)
                {
                    location.Y += avatarFont.LineSpacing;

                    if (Avatar.GetMoveModifier(playerMove.MoveElement, enemy.Element) < 1f)
                    {
                        GameRef.SpriteBatch.DrawString(font, "It is not very effective.",
location, Color.Black);
                    }
                    else if (Avatar.GetMoveModifier(playerMove.MoveElement, enemy.Element) >
1f)
                    {
                        GameRef.SpriteBatch.DrawString(font, "It is super effective.",
location, Color.Black);
                    }
                }
            }
            else
            {
                GameRef.SpriteBatch.DrawString(font, "Enemy " + enemy.Name + " uses " +
enemyMove.Name + ".", location, Color.Black);

                if (enemyMove.Target == Target.Enemy && playerMove.MoveType ==
MoveType.Attack)
                {
                    location.Y += avatarFont.LineSpacing;

                    if (Avatar.GetMoveModifier(enemyMove.MoveElement, player.Element) < 1f)
                    {
                        GameRef.SpriteBatch.DrawString(font, "It is not very effective.",
location, Color.Black);
                    }
                    else if (Avatar.GetMoveModifier(enemyMove.MoveElement, player.Element) >
1f)
                    {
                        GameRef.SpriteBatch.DrawString(font, "It is super effective.",
location, Color.Black);
                    }
                }
            }

            GameRef.SpriteBatch.Draw(avatarBorder, playerBorderRect, Color.White);
            GameRef.SpriteBatch.Draw(player.Texture, playerRect, Color.White);
            GameRef.SpriteBatch.Draw(enemy.Texture, enemyRect, Color.White);

            GameRef.SpriteBatch.Draw(avatarBorder, playerBorderRect, Color.White);

            playerHealth = (float)player.CurrentHealth / (float)player.GetHealth();
            MathHelper.Clamp(playerHealth, 0f, 1f);
            playerHealthRect.Width = (int)(playerHealth * 286);

            GameRef.SpriteBatch.Draw(avatarHealth, playerHealthRect, healthSourceRect,
Color.White);

            GameRef.SpriteBatch.Draw(avatarBorder, enemyBorderRect, Color.White);

            enemyHealth = (float)enemy.CurrentHealth / (float)enemy.GetHealth();
            MathHelper.Clamp(enemyHealth, 0f, 1f);
            enemyHealthRect.Width = (int)(enemyHealth * 286);

            GameRef.SpriteBatch.Draw(avatarHealth, enemyHealthRect, healthSourceRect,
Color.White);
```

```
            GameRef.SpriteBatch.DrawString(avatarFont, player.Name, playerName,
Color.White);
            GameRef.SpriteBatch.DrawString(avatarFont, enemy.Name, enemyName, Color.White);

            GameRef.SpriteBatch.Draw(player.Texture, playerMiniRect, Color.White);
            GameRef.SpriteBatch.Draw(enemy.Texture, enemyMiniRect, Color.White);

            GameRef.SpriteBatch.End();
        }

        public void SetAvatars(Avatar player, Avatar enemy)
        {
            this.player = player;
            this.enemy = enemy;

            if (player.GetSpeed() >= enemy.GetSpeed())
            {
                turn = CurrentTurn.Players;
            }
            else
            {
                turn = CurrentTurn.Enemies;
            }
        }

        public void SetMoves(IMove playerMove, IMove enemyMove)
        {
            this.playerMove = playerMove;
            this.enemyMove = enemyMove;
        }

        public void Start()
        {
            cTimer = TimeSpan.Zero;
            dTimer = TimeSpan.Zero;
            first = true;
        }

        #endregion
    }
}
```

There is first an enumeration CurrentTurn with values Player and Enemy. These values determine who's turn it is during this round of combat. There is then an interface IDamageState that defines the public members that can be called. They are SetAvatars which is the same as the other two methods, SetMoves which sets what moves each avatar is going to attempt to use and Start which starts the timers for this state.

Just like the other states there are a lot of member variables for positioning and drawing the visual elements. In hind sight it probably would have been better to make those variables protected in the BattleState class and inherit these two classes from that class instead of duplicating these variables in these other classes. There is also a member variable names turn that holds whose turn it is. The next new members are cTimer and dTimer. These are used to measure how much time has passed after a certain point. There are also IMove members that hold what move the avatars are going to use. Finally, first holds who goes first, the player or the enemy.

This constructor works like the other ones in that it just positions elements at certain points on the screen. The LoadContent method just loads the content for displaying the avatars and their properties.

The Update method is where the magic happens, so to speak. There is an if statement that checks a number of conditions. It checks if any of the following are true and if the time passed in dTimer is greater than 3. Those conditions are cTimer is greater than 4 seconds, the enemy avatar is not alive or the player avatar is not alive. Inside that if I check to see if either the player avatar or enemy avatar are not alive. If this is true I pop the current state off the stack, the damage state, and push the BattleOver state onto the stack. I then call the SetAvatars method passing in the player and enemy avatars. If they are both still alive I just pop this state off this stack which returns control back to the battle state.

There is then an if that check that cTimer is greater than 2 seconds, the first member variable is true and both avatars are in good health. In this case first means that if this is the first move resolved or the second move resolved. In this case I want to switch and resolve the second move. I set first to false. I then reset the duration of the dTimer memeber variable. I check the turn variable to see who's turn it is. If it is player I switch that member variable to be the enemies turn and ResolveMove to resolve the enemies move. Same in reverse for the else step.

The else if checks to see if cTimer is zero. That means that this is the first time that the Update method has been called. In that case I reset dTimer to 0 and call the ResolveMove method on whatever avatar's turn it is.

The last thing that I do is increment the two timer variables by adding in the ElapsedGameTime property of the gameTime parameter that is passed to Update. You will want to play with the duration of the timers to have them match what you are expecting. You could also play an animation when each avatar attacks. I will do just that in a future tutorial.

In the Draw method I draw the scene. Much of it will be familiar from the other two states. What is new is that I draw the out come of the last move resolved. First, I display what move the avatar has used. I then move to the next line and display if it wasn't effective or if it was super effective, based on the element of the move used and the element of the defending avatar.

The SetAvatars method assigns the member variables for the avatars to the values passed in. I then check if the player's speed is greater than or equal to the opponent's speed. If it is the first turn is the player's turn otherwise the enemy resolves its move first.

SetMoves just sets the moves to the values that are passed in. Start resets the two timers to zero and resets that first member variable to true so that both moves will be resolved.

The last thing to do is to is to have the Update method of the BattleState class drive the choices for the combat. Change the code of the Update method in BattleState to the following.

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

    if (Xin.CheckKeyReleased(Keys.P))
        manager.PopState();

    combatScene.Update(gameTime, index.Value);

    if (Xin.CheckKeyReleased(Keys.Space) || Xin.CheckKeyReleased(Keys.Enter))
    {
        manager.PushState((DamageState)GameRef.DamageState, index);
        GameRef.DamageState.SetAvatars(player, enemy);
        IMove enemyMove = null;

        do
        {
            int move = random.Next(0, enemy.KnownMoves.Count);
            int i = 0;

            foreach (string s in enemy.KnownMoves.Keys)
            {
                if (move == i)
                    enemyMove = (IMove)enemy.KnownMoves[s].Clone();
                i++;
            }

        } while (!enemyMove.Unlocked);


GameRef.DamageState.SetMoves((IMove)player.KnownMoves[combatScene.OptionText].Clone(),
enemyMove);
        GameRef.DamageState.Start();

        player.Update(gameTime);
        enemy.Update(gameTime);
    }

    Visible = true;

    base.Update(gameTime);
}
```

So, what is new that there is a check to see if the space bar or enter key have been released, meaning that the player has made their selection. If they have I push the damage state onto the stack of states. I then call the SetAvatars method passing in the two avatars. Next is a local variable of type IMove that will hold the move the enemy avatar will use. Follow that is a do while loop that generates a random number in the range of all known moves for the avatar. In a foreach loop I then iterate over the keys in the KnownMoves collection. If the selected move matches an variable that increments during each iteration of the loop I set enemyMove to be a clone of that move. I then check to see if that moves is unlocked or not. If it is not I repeat the process.

The next step is that I call the SetMoves method on the damage state to set the moves they are going to be applied. I also call the Start method to reset the timers for the state. Finally I call the Update method of the player and enemy avatars. This allows any effects that have a duration to count down and remove themselves. I also assign the Visible property of the state to true. This keeps this state visible while I draw the damage state.

You can now build and run the game. If you move the player next to one of the two avatars and press the B key you will be able to start a battle with the character's avatar and play through the battle.

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