A Summoner's Tale – MonoGame Tutorial Series

Chapter 11

Battling 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.

So, we have the player, characters and avatars. The next step will be battling avatars. The first step will be to add to the player class the avatars that the player has captured/learned. I say captured as well as learned because I will be including a second player component that works more like Pokemon than my demo.

To get started open the Player class in the PlayerComponents folder. I updated this class to make the private member variables protected member variables. I also added in a field and accessor methods. Update that class to the following.

```
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.TileEngine;
using Avatars.AvatarComponents;

namespace Avatars.PlayerComponents
{
    public class Player : DrawableGameComponent
    {
        #region Field Region

        protected Game1 gameRef;
        protected string name;
        protected bool gender;
        protected string mapName;
        protected Point tile;
        protected AnimatedSprite sprite;
        protected Texture2D texture;
```

```
        protected float speed = 180f;

        protected Vector2 position;
        protected Dictionary<string, Avatar> avatars = new Dictionary<string, Avatar>();
        private string currentAvatar;

        #endregion

        #region Property Region

        public Vector2 Position
        {
            get { return sprite.Position; }
            set { sprite.Position = value; }
        }

        public AnimatedSprite Sprite
        {
            get { return sprite; }
        }

        public float Speed
        {
            get { return speed; }
            set { speed = value; }
        }

        public Avatar CurrentAvatar
        {
            get { return avatars[currentAvatar]; }
        }

        #endregion

        #region Constructor Region

        private Player(Game game)
            : base(game)
        {
        }

        public Player(Game game, string name, bool gender, Texture2D texture)
            : base(game)
        {
            gameRef = (Game1)game;
            this.name = name;
            this.gender = gender;

            this.texture = texture;
            this.sprite = new AnimatedSprite(texture, gameRef.PlayerAnimations);
            this.sprite.CurrentAnimation = AnimationKey.WalkDown;
        }

        #endregion

        #region Method Region

        public virtual void AddAvatar(string avatarName, Avatar avatar)
        {
            if (!avatars.ContainsKey(avatarName))
                avatars.Add(avatarName, avatar);
        }

        public virtual Avatar GetAvatar(string avatarName)
        {
            if (avatars.ContainsKey(avatarName))
```

```
                return avatars[avatarName];

            return null;
        }

        public virtual void SetAvatar(string avatarName)
        {
            if (avatars.ContainsKey(avatarName))
                currentAvatar = avatarName;
            else
                throw new IndexOutOfRangeException();
        }

        public void SavePlayer()
        {
        }

        public static Player Load(Game game)
        {
            Player player = new Player(game);

            return player;
        }

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

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

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

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

            sprite.Draw(gameTime, gameRef.SpriteBatch);
        }

        #endregion
    }
}
```

The new field that I added is a Dictionary<string, Avatar> that holds all the avatars that the player owns. I also added a field currentAvatar that holds which is the avatar that will be used when combat first starts. I also added a virtual property that returns the avatar with that name. I then added a virtual method AddAvatar that checks if an avatar with that name already exist and if it doesn't adds it to the collection of avatars. The GetAvatar virtual method checks to see if an avatar with that name exists in the collection and if it does it returns it. Otherwise it returns null. I also added a SetAvatar method that will set the currentAvatar member to the value passed in if the collection of avatars contains the key passed in. If it does not exist in the collection I thrown an exception.

Why did I make some of the members virtual? I'm taking a slightly different approach with have different types of player components. Instead of using interfaces I'm using inheritance. This is really helpful to learn of you don't know it so that is why I included it in this tutorial. Any of the virtual

members can be overridden in the class that inherits from the player with the same name and parameters but behave differently.

Another note, why do I throw exceptions instead of letting the game crash and handle the exception there? The first is throwing a specific exception gives me an idea of what the problem is and how to stop. The second reason is that eventually I start catching the exceptions and handle them. This helps prevent the player from injecting values into the game to cheat as well as find logic errors.

The next thing that I want to add is a second player class, called PPlayer. This class deals with avatars more like the player in Pokemon because they are limited to the number of avatars they have at one time. Right click the PlayerComponents folder in your solution, select Add and then Class. Name the new class PPlayer. 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.PlayerComponents
{
    public class PPlayer : Player
    {
        public const int MaxAvatars = 6;

        private List<Avatar> battleAvatars = new List<Avatar>();
        private int currentAvatar;

        public override Avatar CurrentAvatar
        {
            get { return battleAvatars[currentAvatar]; }
        }

        public PPlayer(Game game, string name, bool gender, Texture2D texture)
            : base(game, name, gender, texture)
        {
        }

        public void SetCurrentAvatar(int index)
        {
            if (index < 0 || index > MaxAvatars)
                throw new IndexOutOfRangeException();

            currentAvatar = index;
        }

        public Avatar GetBattleAvatar(int index)
        {
            if (index < 0 || index > MaxAvatars)
                throw new IndexOutOfRangeException();

            return battleAvatars[index];
        }

        public void AddBattleAvatar(Avatar avatar)
        {
            if (battleAvatars.Count >= MaxAvatars - 1)
                throw new OverflowException();
```

```
            battleAvatars.Add(avatar);
        }

        public void RemoveBattleAvatar(int index)
        {
            if (index >= battleAvatars.Count)
                throw new IndexOutOfRangeException();

            battleAvatars.RemoveAt(index);
        }
    }
}
```

There is a constant in this class that describes the maximum number of avatars in a party and it is set to 6. This could be set to just about any value but to be consistent with other classes I went with 6. was pretty good for demonstration purposes. Next up is a List<Avatar> which will hold the avatars for the party and an integer variable that holds the currently selected avatar that will be used when a battle starts. There is then a property where I override the behavior of the CurrentAvatar property. Instead of using the string value in the other class I use the integer value in this class.

In order to inherit from a class you need to call one of its constructors. The parent class has a private constructor and a public constructor. Since the values need to be set I added in the parameters to the list of arguments for the constructor to the signature for the constructor and the call the parent using base.

After that there are some methods for manipulating the battle avatars. SetCurrentAvatar is used to assign which avatar is the default for starting combat. It checks to see if the value passed in is out of bounds. If it is it throws an exception, otherwise it sets the value. GetBattleAvatar is used to get the actual avatar object. It also checks to make sure the value is in range and if it is throw an exceptions. Otherwise return the avatar at that particular index.

AddBattleAvatar is used to add an avatar to the player's battle avatar list. It checks to make sure that there is room for the avatar. Since the index is zero based if the count is greater than or equal to the maximum number of avatars minus one there is no room for it so throw an exception. Otherwise it is find to return the avatar. The RemoveBattleAvatar method checks that the index is with in the range of allowed indexes and if it is throws an exception. Otherwise it removes the avatar at the specified index.

I want to make an update to the AvatarManager method FromFile before moving onto the next step. What I want to do is change it so that the dictionary key is always in lower case rather than mixed case. To do that I used the method .ToLowerInvariant(). You could probably get away with just ToLower though. I included the Invariant part because I deal with localization on a daily basis and you may want to update your game to support multiple languages. Here is the update for that method.

```
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.ToLowerInvariant()))
                                avatarList.Add(avatar.Name.ToLowerInvariant(), avatar);
                        }
                    } while (lineIn != null);
                }
                catch
                {
                }
                finally
                {
                    if (reader != null)
                        reader.Close();
                }
            }
        }
        catch
        {
        }
        finally
        {
            if (stream != null)
                stream.Close();
        }
    }
}
```

I'm now going to update the Character so that when a string is passed to the FromString method it will used the 5th value in the string to assign the current battleAvatar. Here is the update.

```
public static Character FromString(Game game, string characterString)
{
    if (gameRef == null)
        gameRef = (Game1)game;

    if (characterAnimations.Count == 0)
        BuildAnimations();

    Character character = new Character();
    string[] parts = characterString.Split(',');

    character.name = parts[0];
    Texture2D texture = game.Content.Load<Texture2D>(@"CharacterSprites\" + parts[1]);
    character.sprite = new AnimatedSprite(texture, gameRef.PlayerAnimations);

    AnimationKey key = AnimationKey.WalkDown;
    Enum.TryParse<AnimationKey>(parts[2], true, out key);

    character.sprite.CurrentAnimation = key;

    character.conversation = parts[3];

    character.battleAvatar = AvatarManager.GetAvatar(parts[4].ToLowerInvariant());

    return character;
}
```

The change is the second last line of code. I call the GetAvatar method of the AvatarManager class passing in the 5th part of the string in lower case. I'm now going to make a similar change to the PCharacter class. It will add upto six avatars to the character's list of avatars. Update that method as follows.

```
public static PCharacter FromString(Game game, string characterString)
{
    if (gameRef == null)
        gameRef = (Game1)game;

    if (characterAnimations.Count == 0)
        BuildAnimations();

    PCharacter character = new PCharacter();
    string[] parts = characterString.Split(',');

    character.name = parts[0];
    Texture2D texture = game.Content.Load<Texture2D>(@"CharacterSprites\" + parts[1]);
    character.sprite = new AnimatedSprite(texture, gameRef.PlayerAnimations);

    AnimationKey key = AnimationKey.WalkDown;
    Enum.TryParse<AnimationKey>(parts[2], true, out key);

    character.sprite.CurrentAnimation = key;

    character.conversation = parts[3];

    for (int i = 4; i < 10 && i < parts.Length; i++)
        character.avatars[i - 4] = AvatarManager.GetAvatar(parts[i].ToLowerInvariant());

    return character;
}
```

What the code does is similar to what I did when building avatars. For avatars I would read their moves until there were no moves left in the string.

Next step is to assign the player and the characters avatars. I will do that in the SetUpNewGame method of the GamePlayState class. What I did was move creating the player form LoadContent to SetUpNewGame. Please update those two methods to the following.

```
protected override void LoadContent()
{
}

public void SetUpNewGame()
{
    Texture2D spriteSheet = content.Load<Texture2D>(@"PlayerSprites\maleplayer");

    player = new Player(GameRef, "Wesley", false, spriteSheet);
    player.AddAvatar("fire", AvatarManager.GetAvatar("fire"));
    player.SetAvatar("fire");

    Texture2D tiles = GameRef.Content.Load<Texture2D>(@"Tiles\tileset1");
    TileSet set = new TileSet(8, 8, 32, 32);
    set.Texture = tiles;

    TileLayer background = new TileLayer(200, 200);
    TileLayer edge = new TileLayer(200, 200);
    TileLayer building = new TileLayer(200, 200);
    TileLayer decor = new TileLayer(200, 200);
```

```
    map = new TileMap(set, background, edge, building, decor, "test-map");

    map.FillEdges();
    map.FillBuilding();
    map.FillDecoration();

    ConversationManager.CreateConversations(GameRef);

    ICharacter teacherOne = Character.FromString(GameRef,
"Lance,teacherone,WalkDown,teacherone,water");
    ICharacter teacherTwo = PCharacter.FromString(GameRef,
"Marissa,teachertwo,WalkDown,tearchertwo,wind,earth");

    teacherOne.SetConversation("LanceHello");
    teacherTwo.SetConversation("MarissaHello");

    GameRef.CharacterManager.AddCharacter("teacherone", teacherOne);
    GameRef.CharacterManager.AddCharacter("teachertwo", teacherTwo);

    map.Characters.Add("teacherone", new Point(0, 4));
    map.Characters.Add("teachertwo", new Point(4, 0));

    camera = new Camera();
}
```

The LoadContent now does nothing and is there in case it is need in the future. I moved creating the player to the top of the method. After creating the player I add a "fire" avatar to their avatar collection and set their current avatar to "fire". For the first character that is a Character I added another value to the string, water, that will set their avatar to be a "water" avatar. For the other character, that is a PCharacter, I added a "wind" and "earth" avatar to the character's collection.

The last thing that I'm going to add is the base battle state and show how to switch to it from the game play state. For that you will need two graphics. One is a border that holds the health bar and the actual health bar. You can download those images using this link. Once they are ready open the MonoGame content builder. Select the Misc folder and then click the Add Existing Item and browse to the avatarborder.png file and add id to the project. Repeat the process for the avatarhealth.png. Now save and build the project.

Next, right click the GameStates folder, select Add and then Class Name this new class BattleState. Here is the code for that class.

```
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.ConversationComponents;
using Avatars.Components;

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

    public class BattleState : BaseGameState, IBattleState
```

```
    {
        #region Field Region

        private Avatar player;
        private Avatar enemy;
        private GameScene combatScene;
        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 font;
        private SpriteFont avatarFont;

        #endregion

        #region Property Region
        #endregion

        #region Constructor Region

        public BattleState(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");

            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();
        }

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

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

            GameRef.SpriteBatch.Begin();

            combatScene.Draw(gameTime, GameRef.SpriteBatch, combatBackground, font);

            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;

            player.StartCombat();
            enemy.StartCombat();
        }
```

```
        public void StartBattle()
        {
            player.StartCombat();
            enemy.StartCombat();
            playerHealth = 1f;
            enemyHealth = 1f;
        }

        #endregion
    }
}
```

First off, I included an interface for the state IBattleState. This is the contract that is provided to other states to interface with this state. The members that in includes are SetAvatars and StartBattle. The first is used to place the avatars on the screen where as the second is used to start a battle between the two avatars.

Most of the member variables in this class are for position elements and holding the image for the elements. There are also an Avatar field for the player and the enemy. There is also a GameScene that will be used by the player to select what move they want the avatar to use that round.

All the constructor does is position items based on the screen rectangle. The elements that are being places are the images for the avatars, the position of their bar that holds the health bar and the actual health bar.

In the LoadContent method I load in the background, two fonts that were already added to the project and the two textures that were just added. I also create a basic combat scene using the conversation components. This will be used to display options to the player and other messages during combat. Currently the Update screen just calls the base update method in order to update the components.

The Draw method first starts drawing the sprite batch. Since it is at the back the scene is drawn first. After that I draw the player's avatar and then the enemy's avatar. Then I draw the borders and the avatar's current health. To draw the current health I get the percentage of the avatar's health is left. I then clamp it between 0 and 1. To determine the width out I multiple the fraction of the remain health by the width of the texture. After drawing the bars I write the name of the avatar into the bar. Finally I then draw the avatar images.

The SetAvatars method is used to set the avatars. After assigning the local member variables to the corresponding parameters I call StartCombat to start combat between the two. In the StartBattle method I also call the StartCombat method and set two member variables to 1f.

The last thing that I'm going to cover is moving from the game play state to this new state. First, we need to update the Game1 class to create a base battle state that will be reused for all battles in the game. The change was I add a member variable for the state, a property to expose it and create it in the constructor. Here is the updated Game1 class.

```
using Avatars.CharacterComponents;
using Avatars.Components;
using Avatars.GameStates;
using Avatars.StateManager;
using Avatars.TileEngine;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;
```

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

namespace Avatars
{
    public class Game1 : Game
    {
        GraphicsDeviceManager graphics;
        SpriteBatch spriteBatch;
        Dictionary<AnimationKey, Animation> playerAnimations = new Dictionary<AnimationKey,
Animation>();

        GameStateManager gameStateManager;
        CharacterManager characterManager;

        ITitleIntroState titleIntroState;
        IMainMenuState startMenuState;
        IGamePlayState gamePlayState;
        IConversationState conversationState;
        IBattleState battleState;

        static Rectangle screenRectangle;

        public SpriteBatch SpriteBatch
        {
            get { return spriteBatch; }
        }

        public static Rectangle ScreenRectangle
        {
            get { return screenRectangle; }
        }

        public ITitleIntroState TitleIntroState
        {
            get { return titleIntroState; }
        }

        public IMainMenuState StartMenuState
        {
            get { return startMenuState; }
        }

        public IGamePlayState GamePlayState
        {
            get { return gamePlayState; }
        }

        public IBattleState BattleState
        {
            get { return battleState; }
        }

        public Dictionary<AnimationKey, Animation> PlayerAnimations
        {
            get { return playerAnimations; }
        }

        public CharacterManager CharacterManager
        {
            get { return characterManager; }
        }

        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);

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

            characterManager = CharacterManager.Instance;
        }

        protected override void Initialize()
        {
            Components.Add(new Xin(this));

            Animation animation = new Animation(3, 64, 64, 0, 0);
            playerAnimations.Add(AnimationKey.WalkDown, animation);

            animation = new Animation(3, 64, 64, 0, 64);
            playerAnimations.Add(AnimationKey.WalkLeft, animation);

            animation = new Animation(3, 64, 64, 0, 128);
            playerAnimations.Add(AnimationKey.WalkRight, animation);

            animation = new Animation(3, 64, 64, 0, 192);
            playerAnimations.Add(AnimationKey.WalkUp, animation);


            base.Initialize();
        }

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

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

        protected override void UnloadContent()
        {
        }

        protected override void Update(GameTime gameTime)
        {
            if (GamePad.GetState(PlayerIndex.One).Buttons.Back == ButtonState.Pressed ||
Keyboard.GetState().IsKeyDown(Keys.Escape))
                Exit();

            base.Update(gameTime);
        }

        protected override void Draw(GameTime gameTime)
        {
```

```
            GraphicsDevice.Clear(Color.CornflowerBlue);

            base.Draw(gameTime);
        }
    }
}
```

Next up I'm going to update the Update method in the GamePlayScreen to switch states to the battle state if the player is close to the character and presses B. Change that method to the following.

```
public override void Update(GameTime gameTime)
{
    Vector2 motion = Vector2.Zero;
    int cp = 8;

    if (Xin.KeyboardState.IsKeyDown(Keys.W) && Xin.KeyboardState.IsKeyDown(Keys.A))
    {
        motion.X = -1;
        motion.Y = -1;
        player.Sprite.CurrentAnimation = AnimationKey.WalkLeft;
    }
    else if (Xin.KeyboardState.IsKeyDown(Keys.W) && Xin.KeyboardState.IsKeyDown(Keys.D))
    {
        motion.X = 1;
        motion.Y = -1;
        player.Sprite.CurrentAnimation = AnimationKey.WalkRight;
    }
    else if (Xin.KeyboardState.IsKeyDown(Keys.S) && Xin.KeyboardState.IsKeyDown(Keys.A))
    {
        motion.X = -1;
        motion.Y = 1;
        player.Sprite.CurrentAnimation = AnimationKey.WalkLeft;
    }
    else if (Xin.KeyboardState.IsKeyDown(Keys.S) && Xin.KeyboardState.IsKeyDown(Keys.D))
    {
        motion.X = 1;
        motion.Y = 1;
        player.Sprite.CurrentAnimation = AnimationKey.WalkRight;
    }
    else if (Xin.KeyboardState.IsKeyDown(Keys.W))
    {
        motion.Y = -1;
        player.Sprite.CurrentAnimation = AnimationKey.WalkUp;
    }
    else if (Xin.KeyboardState.IsKeyDown(Keys.S))
    {
        motion.Y = 1;
        player.Sprite.CurrentAnimation = AnimationKey.WalkDown;
    }
    else if (Xin.KeyboardState.IsKeyDown(Keys.A))
    {
        motion.X = -1;
        player.Sprite.CurrentAnimation = AnimationKey.WalkLeft;
    }
    else if (Xin.KeyboardState.IsKeyDown(Keys.D))
    {
        motion.X = 1;
        player.Sprite.CurrentAnimation = AnimationKey.WalkRight;
    }

    if (motion != Vector2.Zero)
    {
        motion.Normalize();
        motion *= (player.Speed * (float)gameTime.ElapsedGameTime.TotalSeconds);
```

```
        Rectangle pRect = new Rectangle(
            (int)player.Sprite.Position.X + (int)motion.X + cp,
            (int)player.Sprite.Position.Y + (int)motion.Y + cp,
            Engine.TileWidth - cp,
            Engine.TileHeight - cp);

        foreach (string s in map.Characters.Keys)
        {
            ICharacter c = GameRef.CharacterManager.GetCharacter(s);
            Rectangle r = new Rectangle(
                (int)map.Characters[s].X * Engine.TileWidth + cp,
                (int)map.Characters[s].Y * Engine.TileHeight + cp,
                Engine.TileWidth - cp,
                Engine.TileHeight - cp);

            if (pRect.Intersects(r))
            {
                motion = Vector2.Zero;
                break;
            }
        }

        Vector2 newPosition = player.Sprite.Position + motion;

        player.Sprite.Position = newPosition;
        player.Sprite.IsAnimating = true;
        player.Sprite.LockToMap(new Point(map.WidthInPixels, map.HeightInPixels));
    }
    else
    {
        player.Sprite.IsAnimating = false;
    }

    camera.LockToSprite(map, player.Sprite, Game1.ScreenRectangle);
    player.Sprite.Update(gameTime);

    if (Xin.CheckKeyReleased(Keys.Space) || Xin.CheckKeyReleased(Keys.Enter))
    {
        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)
            {
                IConversationState conversationState =
(IConversationState)GameRef.Services.GetService(typeof(IConversationState));
                manager.PushState(
                    (ConversationState)conversationState,
                    PlayerIndexInControl);

                conversationState.SetConversation(player, c);
                conversationState.StartConversation();
            }
        }
    }

    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)
```

```
            {
                GameRef.BattleState.SetAvatars(player.CurrentAvatar, c.BattleAvatar);
                manager.PushState(
                    (BattleState)GameRef.BattleState,
                    PlayerIndexInControl);
            }
        }
    }
    base.Update(gameTime);
}
```

What this new code does is check to see if the B key was released. Then like when I checked for starting a conversation with a character I cycle through all of the characters. I get the character and get its distance to the player. If the distance is less than 72 I call the SetAvatars method of the battle state. I then push the battle state onto the stack.

You can now build and run the game. If you move the player close to either of the characters that were added you can press the B key and the game will switch to the battle state. Once you're in the battle state you're stuck there though. For now I'm going to add a quick escape from the battle stated. Change the Update method of the BattleState class to the following.

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

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

    base.Update(gameTime);
}
```

That is going to wrap up this tutorial. In the next tutorial I will get into the two avatars battling each other. I will also be working on preparing the game editor that I used for the demo I made so that you can play around with the framework up until now and see how all the pieces tie together.

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