A Summoner's Tale – MonoGame Tutorial Series

Chapter 9

Conversations 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 will be continuing on with having conversations with the characters in the game. First I will add the assets for conversations. Conversations require text, an object to indicate an item is selected and they require a background. You can download the items I used from this link.

After you have downloaded and extracted the content open the MonoGame content builder. Select the Fonts folder and then Add Existing Item button in the tool bar. Browse for the scenefont.spritefont and add it to that to the projection. Now select the Content node and select the New Folder button on the tool bar and name this folder Scenes. Select the Scenes folder and then click the Add Existing Item button in the tool bar. Browse for the scenebackground.png file and add it to the folder. Now, select the Misc folder, and from the toolbar select Add Exiting item. Navigate to the selected.png file and add it to that folder. From the toolbar hit the Save button and then the Build button to build the content. Now close the content builder.

Before we go much further I want to fix a bug that I found in the GameScene class that I created. In the constructors for the class not all of the fields were being assigned correctly. Update those constructors as follows.

```
private GameScene()
{
    NormalColor = Color.Blue;
    HighLightColor = Color.Red;
}

public GameScene(string text, List<SceneOption> options)
    : this()
{
    this.text = text;
    this.options = options;
    textPosition = Vector2.Zero;
}

public GameScene(Game game, string text, List<SceneOption> options)
    : this(text, options)
{
    this.game = game;
}
```

I also want to make an addition to the AnimatedSprite class. What I want to do is add a calculated property that will return the center of the sprite on the screen. This will be used to tell if two sprites are close together. All it does is take the Position of the sprite and add half the height and width to the position to get its center. Add the following property to the AnimatedSprite class.

```
public Vector2 Center
{
    get { return Position + new Vector2(Width / 2, Height / 2); }
}
```

The next thing is to add a state to handle conversations. Right click the GameStates folder, select Add and then Class. Name this new class ConversationState. The initial code for that class follows next.

```
using Avatars.CharacterComponents;
using Avatars.ConversationComponents;
using Avatars.PlayerComponents;
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 interface IConversationState
    {
        void SetConversation(Player player, ICharacter character);
        void StartConversation();
    }

    public class ConversationState : BaseGameState, IConversationState
    {
        #region Field Region

        private Conversation conversation;
        private SpriteFont font;
        private Texture2D background;
        private Player player;
        private ICharacter speaker;

        #endregion

        #region Property Region
        #endregion

        #region Constructor Region

        public ConversationState(Game game)
            : base(game)
        {
            game.Services.AddService(typeof(IConversationState), this);
        }
        #endregion

        #region Method Region

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

        protected override void LoadContent()
        {
            font = GameRef.Content.Load<SpriteFont>(@"Fonts\scenefont");
            background = GameRef.Content.Load<Texture2D>(@"Scenes\scenebackground");
            base.LoadContent();
        }

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

            GameRef.SpriteBatch.Begin();
            conversation.Draw(gameTime, GameRef.SpriteBatch);
            GameRef.SpriteBatch.End();
        }

        public void SetConversation(Player player, ICharacter character)
        {
            this.player = player;
            speaker = character;

            if (ConversationManager.ConversationList.ContainsKey(character.Conversation))
                this.conversation =
ConversationManager.ConversationList[character.Conversation];
            else
                manager.PopState();
        }

        public void StartConversation()
        {
            conversation.StartConversation();
        }

        #endregion
    }
}
```

There are first a number of using statements to bring classes from other namespaces into scope for this class rather than need to explicitly reference them. I then added in an interface IconversationState. This interface will be implemented in the class and the class will register the interface as the service. The methods that must be implemented are SetConversation and StartConversation. SetConversation requires two parameters, the Player object and an ICharacter. Since I implemented Character and PCharacter from this interface either can be passed to this method. This method will initialize the conversation. The other method StartConversation will begin the conversation between the player and the character.

The class inherits from BaseGameState so that it can be used with the state manager and it implements the interface. There are a few private member variables. They hold the Conversation that is currently in progress, the SpriteFont to draw text with, the Texture2D for the conversation scene, the Player object and an ICharacter ther represents the character the player is talking with. I did not include any properties in this class to expose member variable. If you need to expose a member variable it would be best to include it in the interface, like the member variables, and implement them as part of the interface.

There is another reason why that this is important that I have not discussed yet. Suppose that you have created a library from your game code so that you can reuse in other games. Now, suppose you need to extend an object in the library, like adding a new ICharacter type. Rather than adding the new class to the library you can create a new class in the game and have it implement ICharacter. That means that anywhere in the library that you have used ICharacter as a type you can pass the new type from the game and you will not need to recompile the library to use this new object. In essence you are making a library that you can create plugins for. You could also publish this library for others to use in their games.

The constructor registers this instance of the object as a service that can be retrieved and consumed as a service in another class. This goes with my last comment. The class can be included in a library and then consumed in another project. The user will only know about what you've published in the interface. They can also extend it by implementing the interface in their own class(es).

In the LoadContent method I load in the font and the background into their respective member variables. The Draw method just calls the Draw method of the current conversation being displayed. SetConversation sets the player member to the value passed in and speaker to the value passed in. It then checks to see if the character has a conversation associated with them. If they do I set the conversation member variable to that conversation. If they don't have a conversation associated with them I pop the state off the stack and return control back to the calling class. The StartConversation method just calls the StartConversation method of the conversation. It would probably be a good idea to make sure that it is not null before doing this but I will leave that as an exercise.

One method is missing from this class, the Update method. That is because it is the brains of this class and it will need to be implemented in stages. First, add these two using statements with the others to bring some classes into scope in this class.

```
using Avatars.Components;
using Microsoft.Xna.Framework.Input;
```

I added those because we will be using the input manager to test if the player has selected a scene option. Avatars.Components brings the input handler, Xin, into scope and the other brings in items like the Keys enumeration into scope.

Now, add this Update method to the class between LoadContent and Draw.

```
public override void Update(GameTime gameTime)
{
    if (Xin.CheckKeyReleased(Keys.Space) || Xin.CheckKeyReleased(Keys.Enter))
    {
        switch (conversation.CurrentScene.OptionAction.Action)
        {
            case ActionType.Buy :
                break;
            case ActionType.Change :
                speaker.SetConversation(conversation.CurrentScene.OptionScene);
                manager.PopState();
                break;
            case ActionType.End :
                manager.PopState();
                break;
            case ActionType.GiveItems :
                break;
            case ActionType.GiveKey :
                break;
            case ActionType.Quest :
                break;
            case ActionType.Sell :
                break;
            case ActionType.Talk :
                conversation.ChangeScene(conversation.CurrentScene.OptionScene);
                break;
        }
    }
    conversation.Update(gameTime);
    base.Update(gameTime);
}
```

This current implementation checks to see if the space or enter keys have been released. If they have been released there is a switch statement on the currently selected scene option. I then included a case for each value in the enumeration for ActionType. For this tutorial I added code to handle three different actions: Change, End and Talk.

What Change does is changes the conversation for the speaker to an entirely new conversation. To do that I call the SetConversation method on the speaker passing in the OptionScene. The next step is to pop the state off the stack. You would use this when the player is speaking to a character and they choose an option that would end the conversation but the next time they talk to this character you want a different conversation.

End is a trivial case. All it does is pop the conversation state off the stack. As you've guessed this is used when the player stops talking to the character.

Change is a trivial case as well. What it does is call the ChangeScene method passing in the OptionScene of the current scene. This will be used when you want to move the conversation to another branch such as a continue option or answering yes or no to a question from the character.

The last thing to do is to call the Update method on the conversation. If you don't the player will not be able to change their response during the conversation.

Now, let's add this to the game. I'm going to post the code for the entire Game1 class, just to make it easier to follow the changes that are being made. What I have done though is first add a new field of type IConversationState. The next change was to create an instance of the new ConversationState class and assign it to that field. I didn't include a property to expose it because I will be demonstrating how to retrieve it as a service. Here is the code for the new 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;

        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 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);
            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);

        }

        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);
        }
    }
}
```

The next thing I want to do is to create a conversation for each of the characters that I added to the demo. To do that I added a new method to the ConversationManager class called CreateConversations. Add this method to that class.

```
public static void CreateConversations(Game gameRef)
{
    Texture2D sceneTexture = gameRef.Content.Load<Texture2D>(@"Scenes\scenebackground");
    SpriteFont sceneFont = gameRef.Content.Load<SpriteFont>(@"Fonts\scenefont");

    Conversation c = new Conversation("MarissaHello", "Hello", sceneTexture, sceneFont);
    c.BackgroundName = "scenebackground";
    c.FontName = "scenefont";

    List<SceneOption> options = new List<SceneOption>();
    SceneOption option = new SceneOption(
        "Good bye.",
        "",
        new SceneAction() { Action = ActionType.End, Parameter = "none" });
    options.Add(option);

    GameScene scene = new GameScene(
        gameRef,
        "Hello, my name is Marissa. I'm still learning about summoning avatars.",
        options);

    c.AddScene("Hello", scene);

    ConversationList.Add("MarissaHello", c);

    c = new Conversation("LanceHello", "Hello", sceneTexture, sceneFont);
    c.BackgroundName = "scenebackground";
    c.FontName = "scenefont";

    options = new List<SceneOption>();
    option = new SceneOption(
        "Yes",
        "ILikeFire",
        new SceneAction() { Action = ActionType.Talk, Parameter = "none" });
    options.Add(option);

    option = new SceneOption(
        "No",
        "IDislikeFire",
        new SceneAction() { Action = ActionType.Talk, Parameter = "none" });
    options.Add(option);

    scene = new GameScene(
        gameRef,
        "Fire avatars are my favorites. Do you like fire type avatars too?",
        options);

    c.AddScene("Hello", scene);

    options = new List<SceneOption>()
    option = new SceneOption(
                    "Good bye.",
                    "",
                    new SceneAction() { Action = ActionType.End, Parameter = "none" });
    options.Add(option);

    scene = new GameScene(
        gameRef,
        "That's cool. I wouldn't want to hug one though.",
        options);

    c.AddScene("ILikeFire", scene);

    options = new List<SceneOption>()
    option = new SceneOption(
                    "Good bye.",
                    "",
                    new SceneAction() { Action = ActionType.End, Parameter = "none" });
    options.Add(option);

    scene = new GameScene(
        gameRef,
        "Each to their own I guess.",
        options);

    c.AddScene("IDislikeFire", scene);

    conversationList.Add("LanceHello", c);
}
```

First thing of note is that this takes a Game type parameter. It is used to give us access to the content manager for importing the background for the scene and the font for the scene. That is exactly what I did at the start of this method.

The first conversation that I'm going to implement is the simplest type. It just one scene and the scene has one option. I first created a Conversation object with the parameters: "MarissaHello", "Hello", sceneTexture and sceneFont. The first is the name of the conversation, followed by the name of the first scene and the texture and font.

Next is a List<SceneOption> to hold the options for the scene I'm adding to the conversation. I then create a SceneOption object with the text Good bye, no scene to transition to and for the SceneAction is has ActionType.End and none for the parameter. If you recall from above this will just terminate the conversation and pop that state off the stack. This option is then added to the list.

Since a Conversation is made up of GameScene object I create one with the text to be displayed and the List<SceneOption> that I already created. The scene is then added to the conversation and the conversation is listed to the conversation list.

The other conversation constructed similarly. The difference is that the first scene has two options associated with it, a Yes option and a No option. These options use ActionType.Talk and change to the different branches, ILikeFire or IdislikeFire. Those two scene options display some text based on the player's choice.

The next thing to do for the demo is to create the conversations and then attach them to the characters. I did that in the SetUpNewGame in the GamePlayState class. Update that method as follows.

```
public void SetUpNewGame()
{
    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");
    ICharacter teacherTwo = PCharacter.FromString(GameRef,
"Marissa,teachertwo,WalkDown,tearchertwo");
```

```
    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();
}
```

All I did was call the static CreateConversation method on the ConversationManager class. I also called the SetConversation method on the characters that I created and passed in the name of each conversation, LanceHello and MarissaHello.

The last update to have conversations working in the demo is to modify the Update method of the GamePlayState class to check if the space bar or enter key have been pressed and if they have check to see if the player is close to a character. If they are close to a character start a conversation with that character. Update that method as follows.

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

What I did is add an if statement to check if either the space bar or enter key have been released since the last frame. I then iterate over all of the keys of the Characters property of the current map. I then grab the character with that name from the CharacterManager. I then use Vector2.Distance to get how far apart the two centers are. If the absolute value is less than 72 I get the IConversationState that was registered as a service. I then call the SetConversation method passing in the player object and the character object. Finally I start the conversation.

Now, I'm going to explain why I find the distance and compare it to 72. What I'm doing here is a type of collision detection called bounding circles rather than bounding boxes. What that means is I construct a circle around the objects and check to see if they collide by calculating the distance between their centers. I pad the circle so that the player just has to be close, not necessarily colliding with the character. I also like to call this type of collision detection proximity collision detection rather.

I'm going to end the tutorial here as it is a lot to digest in one sitting. In the next tutorial I will cover updating the game to allow for conversations with other characters in the game. 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