A Summoner's Tale – MonoGame Tutorial Series

Chapter 15

Saving Game State

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 Cynthia McMahon.

When I asked on the blog for tutorial suggestions one of the comments asked how one would go about saving game state. I will cover adding that feature to the game in this tutorial. The first question that needs to be answered is what exactly needs to be save? The main objects will be the player and their avatars. Other objects would be conversation states, quest states and other events.

First, you must ask yourself the question, "What needs to be saved?". There are a variety of approaches that could be taken. Only save changed data relevant to the current game or use a shotgun approach and save the entire world. This is the easier of the two to implement so I've decided to go with that approach. Let's get started.

First, I want to update the ICharacter interface to add a Save method that any class that implements the interface must implement. Find the ICharacter interface and update the code to the following.

```
using Avatars.AvatarComponents;
using Avatars.ConversationComponents;
using Avatars.TileEngine;
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;
using System.IO;

namespace Avatars.CharacterComponents
{
    public interface ICharacter
    {
        string Name { get; }
```

```
        bool Battled { get; set; }
        AnimatedSprite Sprite { get; }
        Avatar BattleAvatar { get; }
        Avatar GiveAvatar { get; }
        string Conversation { get; }
        void SetConversation(string newConversation);
        void Update(GameTime gameTime);
        void Draw(GameTime gameTime, SpriteBatch spriteBatch);
        bool Save(BinaryWriter writer);
    }
}
```

The Save method will require that a BinaryWriter object be passed into it. This writer will be used to actually save the character to disk. Why did I decide on using a BinaryWriter instead of a TextWriter or saving as XML? The reason is the other two formats are easy to read and manipulate. A binary file is harder for the player to decode and manipulate. Ideally you'd want to also encrypt the files as well but I will leave that as an exercise.

Next, I want to add the same method to the base Avatar class, but not implement it quite yet. Open the Avatar class and add the following method. Make sure to include the following using statement at the top of the file as well.

```
using System.IO;

        public bool Save(BinaryWriter writer)
         {
            return true;
        }
```

I will implement this method shortly. The last place to add a Save method will be the World class. Again, I'm just adding the stub for now and the rest will be filled out later. Here is the code, including the required using statement.

```
using System.IO;

        public bool Save(BinaryWriter writer)
         {
            return true;
        }
```

The last class that I want to add a Save method to is the Player class. Add the same method and using statements as before, repeated here.

```
using System.IO;

        public bool Save(BinaryWriter writer)
         {
            return true;
        }
```

The method stub needs to be added to the TileMap class as well. Add the method stub and using statement to TileMap.

```
using System.IO;
```

```
        public bool Save(BinaryWriter writer)
        {
            return true;
        }
```

The next step will be to trigger the save process. I did that by checking if the F1 key has been released and if it has trigger the save event for the world. Modify the Update method as follows. Make sure to add a using statement for System.IO at the beginning of the file.

```
using System.IO;
        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 world.CurrentMap.Characters.Keys)
                {
                    ICharacter c = GameRef.CharacterManager.GetCharacter(s);
                    Rectangle r = new Rectangle(
                        (int)world.CurrentMap.Characters[s].X * Engine.TileWidth + cp,
                        (int)world.CurrentMap.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(world.CurrentMap.WidthInPixels,
world.CurrentMap.HeightInPixels));
            }
            else
            {
                player.Sprite.IsAnimating = false;
            }

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

            if (Xin.CheckKeyReleased(Keys.Space) || Xin.CheckKeyReleased(Keys.Enter))
            {
                foreach (string s in world.CurrentMap.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();
                    }
                }

                foreach (Rectangle r in world.CurrentMap.PortalLayer.Portals.Keys)
                {
                    Portal p = world.CurrentMap.PortalLayer.Portals[r];
```

```
                    float distance = Vector2.Distance(
                        player.Sprite.Center,
                        new Vector2(
                            r.X * Engine.TileWidth + Engine.TileWidth / 2,
                            r.Y * Engine.TileHeight + Engine.TileHeight / 2));

                    if (Math.Abs(distance) < 64f)
                    {
                        world.ChangeMap(p.DestinationLevel, new Rectangle(p.DestinationTile.X,
p.DestinationTile.Y, 32, 32));

                        player.Position = new Vector2(
                            p.DestinationTile.X * Engine.TileWidth,
                            p.DestinationTile.Y * Engine.TileHeight);
                        camera.LockToSprite(world.CurrentMap, player.Sprite, Game1.ScreenRectangle);

                        return;
                    }
                }
            }

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

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

            if (Xin.CheckKeyReleased(Keys.F1))
            {
                FileStream stream = new FileStream("avatars.sav", FileMode.Create,
FileAccess.Write);
                BinaryWriter writer = new BinaryWriter(stream);
                world.Save(writer);
                player.Save(writer);
                writer.Close();
                stream.Close();
            }
            base.Update(gameTime);
        }
```

First, there is of course the check to see if the F1 key has been released this frame. If it has I create a FileStream to write the file to disk. The location of this file will be inside the debug folder, for now. It will create a new file even if there is an existing file and open it with write permissions. I then create the BinaryWriter passing in the FileStream. Next step is to call the Save method on the World object and the Save method of the Player object passing in the BinaryWriter object. I then close both the writer and the stream.

The first Save method that I will implement is on the World class. Modify that class to the following

code.

```
public bool Save(BinaryWriter writer)
{
    writer.Write(currentMapName);

    foreach (string s in maps.Keys)
        maps[s].Save(writer);

    return true;
}
```

What is happening here is first I'm writing the name of the current map the player is in. Next I loop through all of the TileMaps and call their save method. It is still just returning true but eventually we will add in some error checking to make sure nothing unusual happens and that we can recover from it.

Now with the World done we'll tackle the TileMap class. Update the Save method of the TileMap class to the following.

```
public bool Save(BinaryWriter writer)
{
    foreach (string s in characters.Keys)
    {
        ICharacter c = CharacterManager.Instance.GetCharacter(s);
        c.Save(writer);
    }

    return true;
}
```

In this case what we do is iterate over the list of characters on the map. Then we use the CharacterManager object to get the character. Finally we call the Save method on the Character. I had to make two tweaks here. First, I added a new field called textureName that will as the name implies store the name of the texture for this character. Second, I set the value of the variable in the FromString method. Here is the updated code for the field and methods.

```
private string textureName;

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];
    character.textureName = parts[1];
    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;
}


public bool Save(BinaryWriter writer)
{
    StringBuilder b = new StringBuilder();
    b.Append(name);
    b.Append(",");
    b.Append(textureName);
    b.Append(",");
    b.Append(sprite.CurrentAnimation);

    writer.Write(b.ToString());

    if (givingAvatar != null)
        givingAvatar.Save(writer);

    if (battleAvatar != null)
        battleAvatar.Save(writer);

    return true;
}
```

In the FromString method I assign the new field, characterTexture name to the name of the texture for the character. In the Save method I create a StringBuilder object and append the fields of the class in the order that are required by the FromString method, other than the Avatars. I then call the Write method of the BinaryWriter to write the character to disk. Here is where you'd want to encrypt the string in some way. The string will show up as text in the file even though it's a binary file. After calling the Write method I call the Save method of the two Avatar objects in this class. There are null checks to make sure there is something to be written.

I'm now going to do something similar in the PCharacter class. I added the same field and updated the FromString method. Add this field, update the FromString method and update the Save method to the following. I also write the Avatar objects separately from the character being written.

```
private string textureName;

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];
    character.textureName = parts[1];
    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.currentAvatar = int.Parse(parts[4]);

    for (int i = 5; i < 11 && i < parts.Length; i++)
        character.avatars[i - 5] = AvatarManager.GetAvatar(parts[i].ToLowerInvariant());

    return character;
}

public bool Save(BinaryWriter writer)
{
    StringBuilder b = new StringBuilder();

    b.Append(name);
    b.Append(",");
    b.Append(textureName);
    b.Append(",");
    b.Append(sprite.CurrentAnimation);
    b.Append(",");
    b.Append(conversation);
    b.Append(",");
    b.Append(currentAvatar);

    writer.Write(b.ToString()

    foreach (Avatar a in avatars)
    {
         if (a != null)
            a.Save(writer);
    }

    return true;
}
```

Very similar to the Character class. The differences are that I updated to the FromString method to shift the fields one part of the string. I also updated the loop where we iterate over the list of avatars to use the new index. Similarly, I use a string builder to create an object that can be written to disk. Once the player stats have been written the next step is to write out the avatars. To do that I look over all of the avatars in the array and call their Save methods passing in the writer so they can be saved to disk. Again, there is a null check to make sure that there is something to be written to disk. I still only return true because we haven't implemented that part yet.

Now it is time to implement the Save method of the Avatar class. Find that method and update it to the following.

```
public bool Save(BinaryWriter writer)
{
    StringBuilder b = new StringBuilder();

    b.Append(name);
    b.Append(",");
    b.Append(element);
    b.Append(",");
    b.Append(experience);
    b.Append(",");
    b.Append(costToBuy);
    b.Append(",");
    b.Append(level);
    b.Append(",");
```

```
    b.Append(attack);
    b.Append(",");
    b.Append(defense);
    b.Append(",");
    b.Append(speed);
    b.Append(",");
    b.Append(health);
    b.Append(",");
    b.Append(currentHealth);


    foreach (string s in knownMoves.Keys)
    {
        b.Append(",");
        b.Append(s);
    }

    writer.Write(b);

    return true;
}
```

Just like in the other methods there is StringBuilder that I use to build the string to written to disk. I then append the fields of the class one by one with a comma afterwards. Instead of writing a comma after currentHealth I go straight to a loop. In the loop I append the comma and then the key for the move. Finally, I write the StringBuilder to disk.

I'm going to end this tutorial here. In the next tutorial I will reverse the process and load a game from disk. Keep checking back on the blog for news on that tutorial. I hope to have it up in the next week or so.

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!
Cynthia McMahon