A Summoner’s Tale
Chapter C
Movement Revisited

Welcome to my Summoner’s Tale tutorial series on creating a Pokémon-inspired game with MonoGame. I'm writing these tutorials for the MonoGame 3.8.1 framework using Visual Studio 2022. The code should work on previous versions of MonoGame and Visual Studio. I plan on creating the editors on macOS and Windows. I’m unfamiliar with Linux, so a few projects may not be done for that platform.

The tutorials will make more sense if they are read in order. You can find the list of tutorials on my web blog, A Summoner’s Tale page. In addition to the PDFs, I will make the code for each tutorial available on GitHub here: https://github.com/Synammon/summoners-tale. It will also be included on the page that links to the tutorials.

In this tutorial, we are going to go deeper into cross-platform games. The objective is to be able to move the sprite the same way on all platforms. One of the most difficult aspects of a role-playing game is moving the player. On a desktop, it is easy. You can read the keyboard and the mouse or even a gamepad. If a keyboard is attached to an Android device, you could use that. Just don’t expect the player to have one. I haven’t found many keyboard options so far as iOS is concerned. I know it is possible with iPadOS, but not sure about phones.

So, how are we going to move the player? Well, we are going to create a virtual D-pad and display it on all devices. We could hide it and only display it on mobile devices, but that makes more work as we need to duplicate code for Android and iOS. The solution that I’m going with is to display the D-pad on all platforms.

Since we are talking about movement, I am implementing tile-based movement in cardinal directions only. So, no diagonal movement. To get started, let’s add the D-Pad. What I did was add four buttons and position them in the bottom left corner of the screen. Thankfully, the GUI controls that we chose has buttons appropriate for them.

Before I get to the gameplay state, I want to make a few changes to the button class. They are necessary for the buttons on the gameplay state. I need to add an event that will fire if the mouse or the touch screen is being pressed on a button. I want to make use of the Size property. The last change is that I want to default a button’s Text property to the empty string, to prevent the null reference exceptions if you don’t set the text explicitly. Update the Button class to the following code.

```
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input.Touch;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework;
using SummonersTale.Forms;
using System;
using System.Collections.Generic;
using System.Text;
using static System.Net.Mime.MediaTypeNames;

namespace SummonersTale
{
    public enum ButtonRole { Accept, Cancel, Menu }

    public class Button : Control
    {
        #region

        public event EventHandler Click;
        public event EventHandler Down;

        #endregion
        #region Field Region

        private readonly Texture2D _background;
        float _frames;

        public ButtonRole Role { get; set; }
        public int Width { get { return _background.Width; } }
        public int Height { get { return _background.Height; } }

        #endregion

        #region Property Region
        #endregion

        #region Constructor Region

        public Button(Texture2D background, ButtonRole role)
        {
            Role = role;
            _background = background;
            Size = new(background.Width, background.Height);
            Text = "";
        }

        #endregion

        #region Method Region

        public override void Draw(SpriteBatch spriteBatch)
        {
            Rectangle destination = new(
            (int)Position.X,
            (int)Position.Y,
            (int)Size.X,
            (int)Size.Y);

            spriteBatch.Draw(_background, destination, Color.White);

            _spriteFont = ControlManager.SpriteFont;

            Vector2 size = _spriteFont.MeasureString(Text);
            Vector2 offset = new((Size.X - size.X) / 2, ((Size.Y - size.Y) / 2));
            spriteBatch.DrawString(_spriteFont, Text, Helper.NearestInt((Position +
offset)), Color);
        }

        public override void HandleInput()
        {
            MouseState mouse = Mouse.GetState();
            Point position = new(mouse.X, mouse.Y);
            Rectangle destination = new(
                (int)(Position.X + Offset.X),
                (int)(Position.Y + Offset.Y),
                _background.Width,
                _background.Height);

            if ((Role == ButtonRole.Accept && Xin.WasKeyReleased(Keys.Enter)) ||
                (Role == ButtonRole.Accept && Xin.WasKeyReleased(Keys.Space)))
            {
                OnClick();
                return;
            }

            if (Role == ButtonRole.Cancel && Xin.WasKeyReleased(Keys.Escape))
            {
                OnClick();
                return;
            }

            if (Xin.WasMouseReleased(MouseButton.Left) && _frames >= 5)
            {
                Rectangle r = destination.Scale(Settings.Scale);

                if (r.Contains(position))
                {
                    OnClick();
                    return;
                }
            }

            if (Xin.TouchReleased() && _frames >= 5)
            {
                Rectangle rectangle= destination.Scale(Settings.Scale);

                if (rectangle.Contains(Xin.TouchReleasedAt))
                {
                    OnClick();
                    return;
                }
            }

            if (Xin.IsMouseDown(MouseButton.Left))
            {
                Rectangle rectangle = destination.Scale(Settings.Scale);

                if (rectangle.Contains(Xin.MouseAsPoint))
                {
                    OnDown();
                    return;
                }
            }

            if (Xin.TouchLocation != new Vector2(-1, -1))
            {
                Rectangle rectangle = destination.Scale(Settings.Scale);

                if (rectangle.Contains(Xin.TouchLocation))
                {
                    OnDown();
                    return;
                }
            }
        }

        private void OnDown()
        {
            Down?.Invoke(this, null);
        }

        private void OnClick()
        {
            Click?.Invoke(this, null);
        }

        public override void Update(GameTime gameTime)
        {
            _frames++;
            HandleInput();
        }

        public void Show()
        {
            _frames = 0;
        }

        #endregion
    }
}
```

The first change is that I added a new event, Down, that will be triggered if the button is being pressed. Not clicked, pressed. In the constructor, I set the Size property of the button to the height and width of the texture for the button. I then set the Text property to the empty string. If the Size property is overridden, the button will be drawn at that size, which is important for our buttons on the screen. That is because the button textures are fairly large, and we don't want to render them at their full size.

You can see in the Draw method I now use the Size property when constructing the destination rectangle. I now use the Size property of the button to center the text inside it as well. Similarly, in the HandleInput method, I use the Size property to create the destination rectangle. The method flows pretty much the same. It first checks for clicks, or taps, on the button. Once those checks are done, I check to see if the left mouse button is down. If it is, and scaled rectangle contains the mouse position as a point I call a new method, OnDown, and exit the method. I check to see if the screen is currently being touched, using the TouchLocation property (which I changed). If it is being touched, I create the scaled rectangle and see if the touch location is inside the rectangle. If it is, I call the OnDown method.

The OnDown method is similar to the OnClick method. It checks to see if the OnDown method is subscribed to. If it is, it invokes the handlers. I just changed the TouchLocation property in the Xin class to default to (-1, -1) instead of (0, 0). I didn't want to risk missing a touch that was precisely at (0, 0). Replace the TouchLocation property of the Xin class to the following.

```
public static Vector2 TouchLocation
{
    get
    {
        Vector2 result = Vector2.Zero - Vector2.One;

        if (touchLocations.Count > 0)
        {
            if (touchLocations[0].State == TouchLocationState.Pressed ||
                touchLocations[0].State == TouchLocationState.Moved)
            {
                result = touchLocations[0].Position;
            }
        }

        return result;
    }
}
```

So, the GamePlayState morphed dramatically. No, not polymorphism again. It actually changed dramatically to use the buttons and tile-based movement. The changes are such that I am going to give you the code for the entire class. Replace the GamePlayState with the following code. Sorry in advance for the wall of code.

```
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;
using Psilibrary.TileEngine;
using SummonersTale.SpriteClasses;
using System;
using System.Collections.Generic;
using System.Reflection.Metadata;
using System.Text;

namespace SummonersTale.StateManagement
{
    public interface IGamePlayState
    {
        GameState GameState { get; }
        void NewGame();
    }

    public class GamePlayState : BaseGameState, IGamePlayState
    {
        private TileMap _tileMap;
        private Camera _camera;
        private AnimatedSprite sprite;
        private Button upButton, downButton, leftButton, rightButton;
        private bool inMotion = false;
        private Rectangle collision = new();
        private float speed;
        private Vector2 motion;

        public GameState GameState => this;

        public GamePlayState(Game game) : base(game)
        {
            Game.Services.AddService((IGamePlayState)this);
        }

        public override void Initialize()
        {
            Engine.Reset(new(0, 0, 1280, 720), 32, 32);
            _camera = new();
            motion = new();
            speed = 96;

            base.Initialize();
        }

        protected override void LoadContent()
        {
            TileSheet sheet = new(content.Load<Texture2D>(@"Tiles/Overworld"),
"test", new(40, 36, 16, 16));
            TileSet set = new(sheet);

            TileLayer ground = new(100, 100, 0, 0);
            TileLayer edge = new(100, 100, -1, -1);
            TileLayer building = new(100, 100, -1, -1);
            TileLayer decore = new(100, 100, -1, -1);

            for (int i = 0; i < 1000; i++)
            {
                edge.SetTile(random.Next(0, 100), random.Next(0, 100), 0,
random.Next(0, 64));
            }

            _tileMap = new(set, ground, edge, building, decore, "test");

            Texture2D texture =
content.Load<Texture2D>(@"CharacterSprites/femalepriest");

            Dictionary<string, Animation> animations = new();

            Animation animation = new(3, 32, 32, 0, 0) { CurrentFrame = 0,
FramesPerSecond = 8 };
            animations.Add("walkdown", animation);

            animation = new(3, 32, 32, 0, 32) { CurrentFrame = 0, FramesPerSecond =
8 };
            animations.Add("walkleft", animation);

            animation = new(3, 32, 32, 0, 64) { CurrentFrame = 0, FramesPerSecond =
8 };
```

```
            animations.Add("walkright", animation);

            animation = new(3, 32, 32, 0, 96) { CurrentFrame = 0, FramesPerSecond =
8 };
            animations.Add("walkup", animation);

            sprite = new(texture, animations)
            {
                CurrentAnimation = "walkdown",
                IsActive = true,
                IsAnimating = true,
            };

            base.LoadContent();

            rightButton = new(content.Load<Texture2D>("GUI/g21245"),
ButtonRole.Menu)
            {
                Position = new(80, Settings.BaseHeight - 80),
                Size = new(32, 32),
                Text = "",
                Color = Color.White,
            };

            rightButton.Down += RightButton_Down;

            Controls.Add(rightButton);

            upButton = new(content.Load<Texture2D>("GUI/g21263"), ButtonRole.Menu)
            {
                Position = new(48, Settings.BaseHeight - 48 - 64),
                Size= new(32, 32),
                Text= "",
                Color= Color.White,
            };

            upButton.Down += UpButton_Down;

            Controls.Add(upButton);

            downButton = new(content.Load<Texture2D>("GUI/g21272"), ButtonRole.Menu)
            {
                Position= new(48, Settings.BaseHeight - 48),
                Size = new(32, 32),
                Text = "",
                Color = Color.White,
            };

            downButton.Down += DownButton_Down;

            Controls.Add(downButton);

            leftButton = new(content.Load<Texture2D>("GUI/g22987"), ButtonRole.Menu)
            {
                Position = new(16, Settings.BaseHeight - 80),
                Size = new(32, 32),
                Text = "",
                Color = Color.White,
            };

            leftButton.Down += LeftButton_Down;

            Controls.Add(leftButton);
        }

        private void LeftButton_Down(object sender, EventArgs e)
        {
            if (!inMotion)
            {
                MoveLeft();
            }
        }

        private void MoveLeft()
        {
            motion = new(-1, 0);
            inMotion = true;
            sprite.CurrentAnimation = "walkleft";
            collision = new(
                (sprite.Tile.X - 2) * Engine.TileWidth,
                sprite.Tile.Y * Engine.TileHeight,
                Engine.TileWidth,
                Engine.TileHeight);
        }

        private void RightButton_Down(object sender, EventArgs e)
        {
            if (!inMotion)
            {
                MoveRight();
            }
        }

        private void MoveRight()
        {
            motion = new(1, 0);
            inMotion = true;
            sprite.CurrentAnimation = "walkright";
            collision = new(
                (sprite.Tile.X + 2) * Engine.TileWidth,
                sprite.Tile.Y * Engine.TileHeight,
                Engine.TileWidth,
                Engine.TileHeight);
        }

        private void DownButton_Down(object sender, EventArgs e)
        {
            if (!inMotion)
            {
                MoveDown();
            }
        }

        private void MoveDown()
        {
            motion = new(0, 1);
```

```
            Point newTile = sprite.Tile + new Point(0, 2);
            inMotion = true;
            sprite.CurrentAnimation = "walkdown";
            collision = new(
                newTile.X * Engine.TileWidth,
                newTile.Y * Engine.TileHeight,
                Engine.TileWidth,
                Engine.TileHeight);
        }

        private void UpButton_Down(object sender, EventArgs e)
        {
            if (!inMotion)
            {
                MoveUp();
            }
        }

        private void MoveUp()
        {
            motion = new(0, -1);
            inMotion = true;
            sprite.CurrentAnimation = "walkup";
            collision = new(
                sprite.Tile.X * Engine.TileWidth,
                (sprite.Tile.Y - 2) * Engine.TileHeight,
                Engine.TileWidth,
                Engine.TileHeight);
        }

        public override void Update(GameTime gameTime)
        {
            Controls.Update(gameTime);

            if (Xin.IsKeyDown(Microsoft.Xna.Framework.Input.Keys.A) && !inMotion)
            {
                MoveLeft();
            }
            else if (Xin.IsKeyDown(Microsoft.Xna.Framework.Input.Keys.D) &&
!inMotion)
            {
                MoveRight();
            }

            if (Xin.IsKeyDown(Microsoft.Xna.Framework.Input.Keys.W) && !inMotion)
            {
                MoveUp();
            }
            else if (Xin.IsKeyDown(Microsoft.Xna.Framework.Input.Keys.S) &&
!inMotion)
            {
                MoveDown();
            }

            if (motion != Vector2.Zero)
            {
                motion.Normalize();
            }
            else
            {
                inMotion = false;
                return;
            }

            if (!sprite.LockToMap(new(99 * Engine.TileWidth, 99 * 
Engine.TileHeight), ref motion))
            {
                inMotion = false;
                return;
            }

            Vector2 newPosition = sprite.Position + motion * speed * 
(float)gameTime.ElapsedGameTime.TotalSeconds;

            Rectangle nextPotition = new Rectangle(
                (int)newPosition.X,
                (int)newPosition.Y,
                Engine.TileWidth,
                Engine.TileHeight);

            if (nextPotition.Intersects(collision))
            {
                inMotion = false;
                motion = Vector2.Zero;
                sprite.Position = new((int)sprite.Position.X, 
(int)sprite.Position.Y);
                return;
            }
            sprite.Position = newPosition;
            sprite.Tile = Engine.VectorToCell(newPosition);

            _camera.LockToSprite(sprite, _tileMap);

            sprite.Update(gameTime);

            base.Update(gameTime);
        }

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

            GraphicsDevice.SetRenderTarget(renderTarget);
            GraphicsDevice.Clear(Color.CornflowerBlue);

            _tileMap.Draw(gameTime, SpriteBatch, _camera, false);

            spriteBatch.Begin(
                SpriteSortMode.Deferred,
                BlendState.AlphaBlend,
                SamplerState.PointClamp,
                null,
                null,
                null,
                _camera.Transformation);
            sprite.Draw(SpriteBatch);
            SpriteBatch.End();

            spriteBatch.Begin();
            Controls.Draw(SpriteBatch);
            spriteBatch.End();

            GraphicsDevice.SetRenderTarget(null);

            SpriteBatch.Begin(SpriteSortMode.Immediate, BlendState.AlphaBlend, 
SamplerState.LinearClamp);

            SpriteBatch.Draw(renderTarget, new Rectangle(new(0, 0), 
Settings.Resolution), Color.White);

            SpriteBatch.End();
        }

        public void NewGame()
        {
            LoadContent();
        }
    }
}
```

There are several new fields. First, there are four Button fields, one for each direction. Next, there is a field that tells if the sprite is in motion or not. Following that is a rectangle that the player's sprite will collide with once it has reached the end of the next tile. Finally, there is a new field, speed, that is the speed of the sprites in pixels per second.

In the Initialize method, I set the base values of some of the fields as before. This time, I set the speed of the sprite to 96.

In the LoadContent method, after creating the map, that sprite and the call to base.LoadContent, I create the buttons. I pass in the appropriate texture from the GUI textures that we added a few tutorials back. You could choose the direction arrows rather than the pointers. The choice is yours. I position them with a hole in the middle. As part of initializing them, I set their size to (32, 32). The only other thing of interest is that I wire the event handlers for the Down event. Also, I add them to the control manager so their Update and Draw methods will be called when I call the Update and Draw method of the control manager. It is cleaner to do it just once.

So, the event handlers are pretty straightforward. They check to see if the sprite is in motion already. If it is not, they call a move method in the appropriate direction. That method will also be called if the key for that direction is down in the Update method.

I will dissect the MoveLeft method and leave the others up to you. They are pretty much the same, the only difference is direction. In the MoveLeft method, I set the motion to (-1, 0), which is moving left across the screen. I set the inMotion property to true so that no further movement will take place until the sprite has stopped moving. I set the animation for the sprite to the walk-left animation. Here is an exciting piece. To test when to stop moving, I create a rectangle that is two tiles to the left of the current tile the sprite is in. So, the sprite will walk through the first tile and then stop when it reaches the outer edges of the collision rectangle. I've seen people calculating velocities, scaling those, calculating timing, and other methods for doing this. This method is much cleaner.

The Update method calls the Update method of the control manager. Then, in the checks, if a key is down, it also checks to make sure the sprite is not in motion. If the key is down and the sprite is not in motion, I call the appropriate Move method. Now, if the motion vector is the zero vector, I set inMotion to false and exit the method. I then calculate the new position of the sprite if it can move in that direction. I then create a rectangle based on its new position. If it collides with the collision rectangle, I set inMotion to false, the motion vector to the zero vector, and, most importantly, cast the position to the nearest integer. This doesn't matter so much on desktop, but it is vital on iOS and Android. If you don't cast them to integers, the floating point builds up, and eventually, you won't be able to move down or right. I then calculate the tile the player's sprite is in using the VectorToCell method.

The Draw method is pretty much the same. The difference is that after drawing the map and the sprite. I call Begin to start rendering. I then render the control manager to draw the buttons. Finally, I call End to finish rendering.

So, we've gone mobile. Things are working basically the same on all platforms. I'm not going to dive any further into this tutorial. I think I've fed you more than enough for one day. So, that is going to be it for this tutorial. I covered a lot, and I don't want to overload you. I encourage you to keep visiting my blog for the latest news on my tutorials.

Good luck with your game programming adventures.
Cynthia