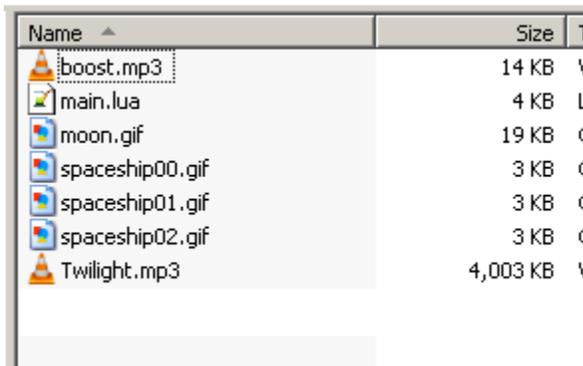


Adding Some Pizzaz

Extract the files I've sent you into Proj 05. This should be nearly identical to what you've written so far in Proj04. This way we can stay in sync.

I added a line that prints out the current FPS of the program. I've also added some comments to the code.



Name	Size
boost.mp3	14 KB
main.lua	4 KB
moon.gif	19 KB
spaceship00.gif	3 KB
spaceship01.gif	3 KB
spaceship02.gif	3 KB
Twilight.mp3	4,003 KB

Anytime you want to leave a comment for yourself or a future programmer you can do this by typing `--` (dash dash) ahead of your message.

```
-- This is a comment for humans to read
```

You can also place this at the end of a line to clarify something. Everything after the `--` is for human eyes only and Love will ignore it.

```
width = love.graphics.getWidth() -- get the Love window's width
```

Sound Objects

We've already used Love's graphics component. It knows how to load and draw images, draw lines, circles and many other things.

Love also has an audio component. It knows how to load sounds and music and how to play them.

```
ambientMusic = love.audio.newSource("twilight.mp3", "stream")
```

Here we are telling Love to create a variable called `ambientMusic` that knows how to play `twilight.mp3` which is a song. `love.audio.newSource` is similar to `love.graphics.newImage`. Instead of pictures, it uses sound files. It can play `.mp3` files, `.ogg` files, and a few others.

You'll notice that at the end of the `newSource` command there's the word "stream". When you stream something, you grab it in small chunks just as you need it. The other method is called "static". If you use static then the entire sound is loaded into memory.

You can think of streaming like eating supper. You stream the food to your mouth from the plate bit by bit. The advantage is that you don't have to put your whole supper in your mouth at once. The disadvantage is that to eat you have to pick up a fork or spoon, fill it, and lift the food to your mouth. So, to start eating may involve a short delay. But once you get started it can be continuous.

You can think of static as a hard candy that you pop into your mouth all at once. The whole thing is there. To eat you just suck. It's very quick to start and stop but it has to fit in your mouth.

A song that's a few minutes long can become huge when loaded into memory. Also, no one will notice if there's a short sub-second delay when you start playing the song. Because of this we usually stream songs.

If we need a sound that we will start and stop frequently and that has to match some on-screen event closely, we will choose static. Our spaceship's engine would be a good example.

So let's go ahead and add some lines to `love.load()` in our program to create these sounds.

```
5     --play some background music using streaming
6     ambientMusic = love.audio.newSource("twilight.mp3","stream")
7     ambientMusic:setLooping(true)           --play again when done
8     ambientMusic:setVolume(.25)           --lower the volume. 1 is max, 0 is off
9     love.audio.play(ambientMusic)         --start playing
```

Let's look carefully at these extra instructions.

```
ambientMusic:setVolume(.25)   VS   love.audio:setVolume(.25)
```

The `:` notation is something new we haven't seen before.

The command **`love.audio.setVolume(.25)`** is telling Love to play ALL sounds at a certain volume. The value 1 is full volume and 0 is off. So 0.25 would play at a quarter of the full volume.

On the other hand **`ambientMusic:setVolume(.25)`** is telling only that sound to play more quietly. Some types of objects have properties that can be set using the `:` notation. So when you see this, it's the programmer giving that specific object some special instructions.

So, `:setLooping` tells the sound to start over when it's done, and `:setVolume` tells the sound to play more softly.

Finally, we tell Love to play the music with `love.audio.play(ambientMusic)`.

Type the lines in and give it a try. The song builds slowly for a few seconds so give it a chance

Roaring Rockets

```
41     spaceship.sound = love.audio.newSource("boost.mp3", "static")
42     spaceship.sound:setVolume(.5)
43     spaceship.sound:setLooping(true)
```

Here we're loading a short roaring sound. Again, we'll set the volume lower as it's a bit loud and we'll tell it to loop when it's done.

When we press the 'w' key to fire the engine we want to start playing this sound. When we let go of the 'w' key the sound should stop.

```
if love.keyboard.isDown("w") then
    --use trigonometry to boost spaceship in direction it is pointing
    spaceship.velX = spaceship.velX + math.sin(spaceship.direction) * spaceship.acceleration * dt
    spaceship.velY = spaceship.velY + math.cos(spaceship.direction) * -spaceship.acceleration * dt

    love.audio.play(spaceship.engineSound)

    --alternate the boost images while boosting to create simple animation
    if spaceship.useImage == "boost1" then
        spaceship.useImage = "boost2"
    else
        spaceship.useImage = "boost1"
    end
else
    love.audio.stop(spaceship.engineSound)

    --use coasting image if ship is drifting
    spaceship.useImage = "coasting"
end
```

Here's a first try at adding an engine sound **BUT** there's small problem.

```
if love.keyboard.isDown("w") then
```

If the user is holding down the 'w' key then Love will execute the instructions that follow until it encounters the **else** keyword. That part is good since Love will encounter

```
love.audio.play(spaceship.engineSound)
```

and will start playing the engine sound as desired. The problem arises the next time Love calls `love.update()`. Since the user is still holding down the 'w' key, Love will try to start playing the engine sound again. So the sound will 'stutter' and never play properly.

The solution is to check that the sound is not already playing before we try to start it. Similarly, when the user lets go of the 'w' key we have to make sure the sound is still playing before trying to stop it.

The sound object has a property called `isStopped()` which is true if the sound is not playing. So `spaceship.engineSound.isStopped()` is true if it's not playing, and false if it is playing.

```

    if spaceship.engineSound:isStopped() then
        love.audio.play(spaceship.engineSound)
    end

```

and

```

    if not spaceship.engineSound:isStopped() then
        love.audio.stop(spaceship.engineSound)
    end

```

Here you're seeing the instruction **not** for the first time. Just like in English, the keyword **not** reverses the truth of the next statement. So

```

spaceship.engineSound:isStopped()    -- true if not playing
not spaceship.engineSound:isStopped() -- true if playing

```

Adding these tests makes sure we only start the sound if it's not playing, and only stop the sound if it is playing.

Here's the complete modified section of the code.

```

if love.keyboard.isDown("w") then
    --use trigonometry to boost spaceship in direction it is pointing
    spaceship.velX = spaceship.velX + math.sin(spaceship.direction) * spaceship.acceleration * dt
    spaceship.velY = spaceship.velY + math.cos(spaceship.direction) * -spaceship.acceleration * dt

    --start playing the spaceship sound if it's not playing yet
    if spaceship.engineSound:isStopped() then
        love.audio.play(spaceship.engineSound)
    end

    --alternate the boost images while boosting to create simple animation
    if spaceship.useImage == "boost1" then
        spaceship.useImage = "boost2"
    else
        spaceship.useImage = "boost1"
    end
end
else
    --stop the spaceship sound if it's playing
    if not spaceship.engineSound:isStopped() then
        love.audio.stop(spaceship.engineSound)
    end

    --use coasting image if ship is drifting
    spaceship.useImage = "coasting"
end

```

With this addition you should have music and sound when you fire your rocket engine.

That's it for now. Next time we'll investigate functions and how they can simplify our code in preparation for adding more goodies like a satellite and asteroids.