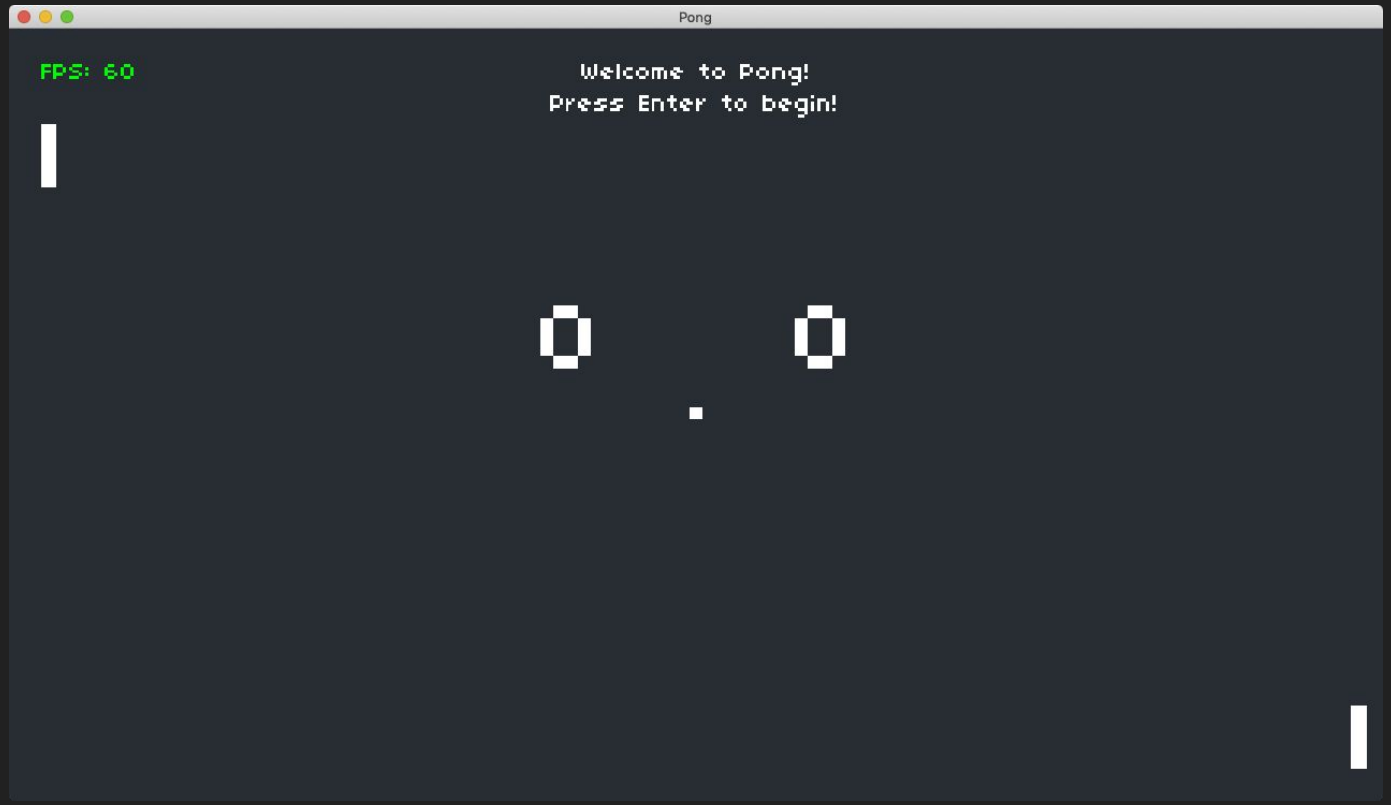
This is CS50.
Games Edition


Pong
FPS: 60
Welcome to Pong!
Press Enter to begin!
0
0

Pong Topics

- Lua
- LÖVE
- Drawing Shapes
- Drawing Text
- DeltaTime and Velocity
- Game State
- Basic OOP (Object-Oriented Programming)
- Box Collision (Hitboxes)
- Sound Effects (with bfxr)


Super Mario 50

Mario Topics

- Spritesheets and Raster Graphics
- Tile Maps
- Gravity
- Animation
- Procedural Generation

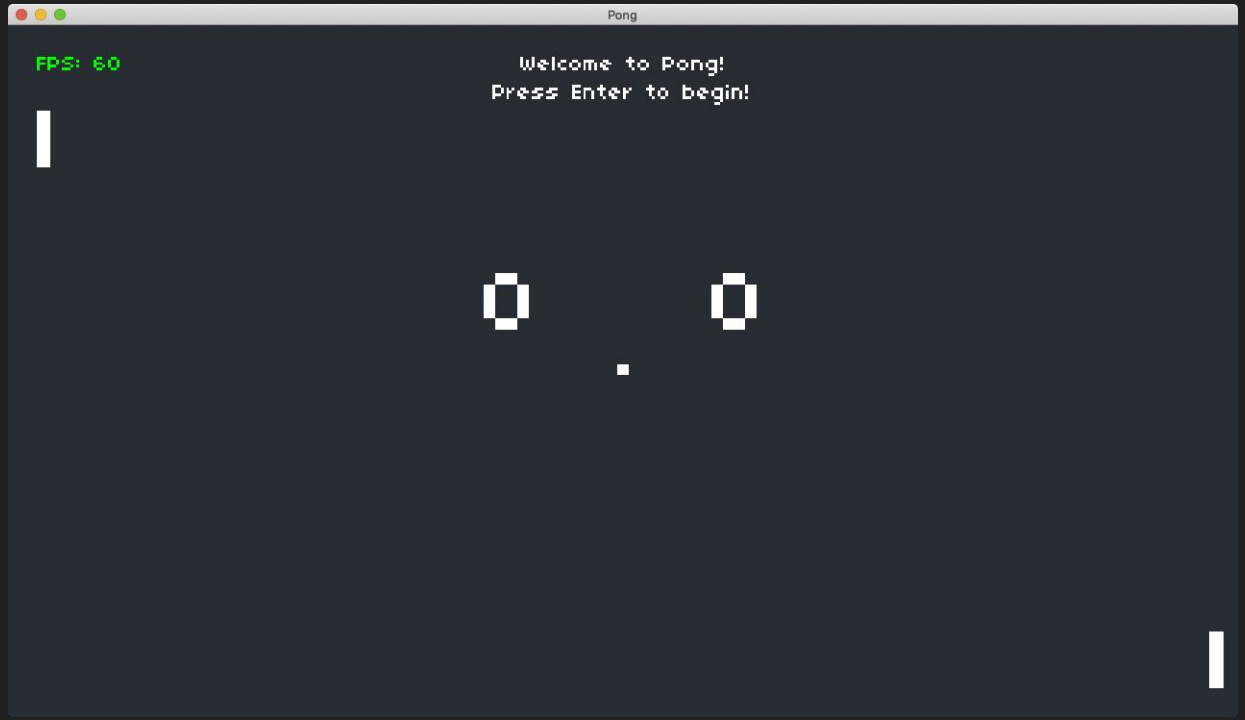
Topics

- Lua
- LÖVE
- Drawing Shapes
- Drawing Text
- DeltaTime and Velocity
- Game State
- Basic OOP (Object-Oriented Programming)
- Box Collision (Hitboxes)
- Sound Effects (with bfxr)

This is CS50. Games Edition

Pong

Our Goal



What is Lua?

- Portuguese for “moon”; invented in 1993 in Brazil
- Flexible, lightweight scripting language focused around “tables”
- Intended for embedded use in larger applications
- Very popular in the video game industry
- Similar(ish) to JavaScript
- Excellent for storing data as well as code (data-driven design)

What is LöVE?

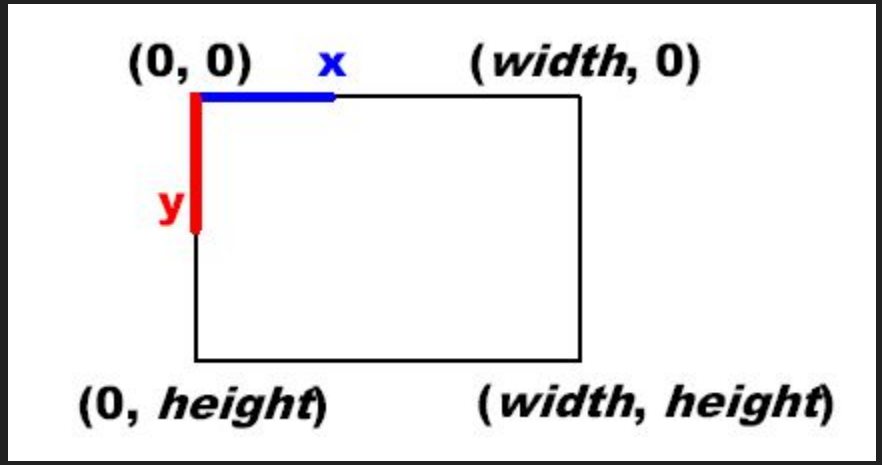
- Fast 2D game development framework written in C++
- Uses Lua as its scripting language
- Contains modules for graphics, keyboard input, math, audio, windowing, physics, and much more
- Completely free and portable to all major desktops and Android/iOS
- Great for prototyping!

Installing LöVE

https://love2d.org/#download

https://love2d.org/wiki/Getting_Started

2D Coordinate System

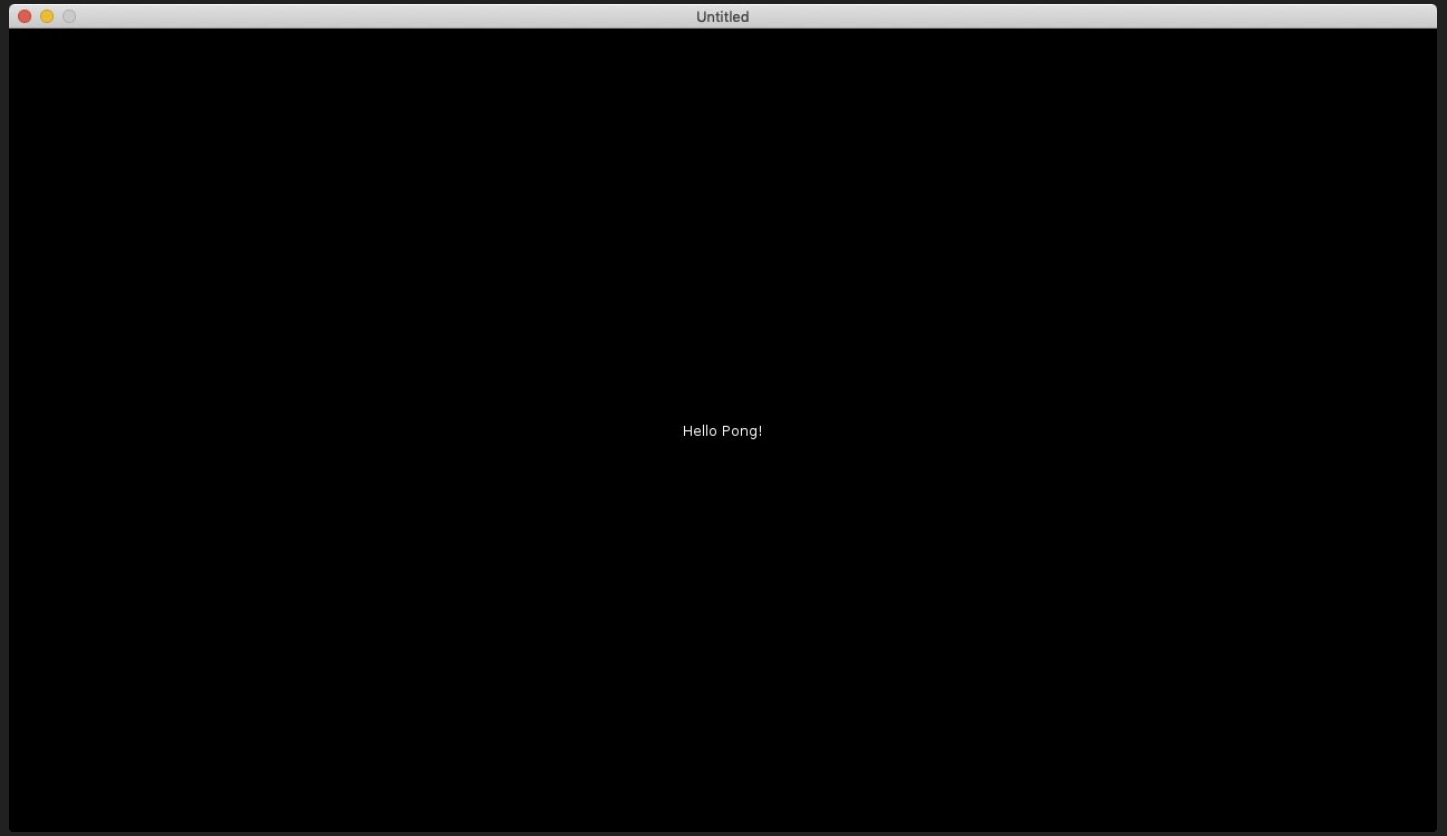


http://rbwhitaker.wdfiles.com/local--files/monogame-introduction-to-2d-graphics/2DCoordinateSystem.png

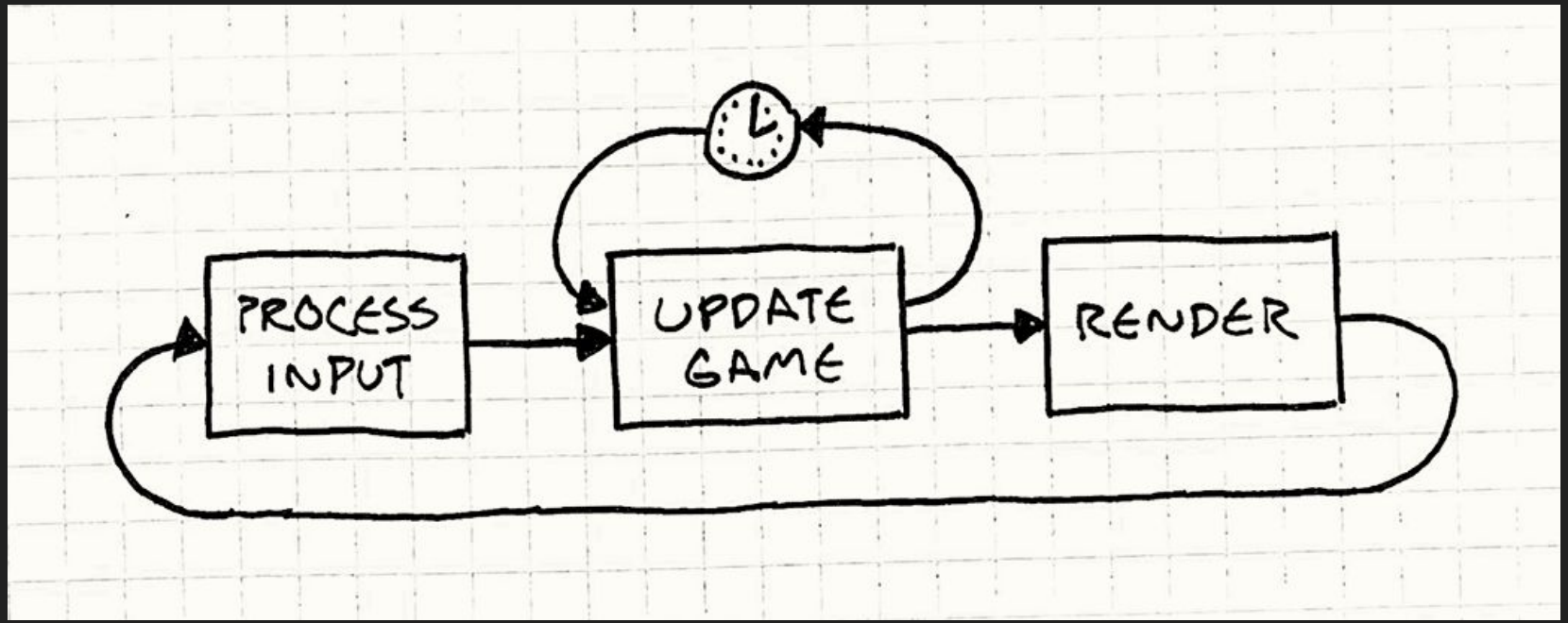
System where objects have an X and Y coordinate (X, Y) and are drawn accordingly; (0, 0) would be the top-left of our system, with positive directions moving down and to the right and negative values moving up and to the left

pong0

"The Day-0 Update"


Untitled
Hello Pong!

What is a game loop?



Credit: Robert Nystrom

http://gameprogrammingpatterns.com/game-loop.html

pong0: Important Functions, p.1

- `love.load()`
  - Used for initializing our game state at the very beginning of program execution.
- `love.update(dt)`
  - Called each frame by LÖVE; `dt` will be the elapsed time in seconds since the last frame, and we can use this to scale any changes in our game for even behavior across frame rates.
- `love.draw()`
  - Called each frame by LÖVE after update for drawing things to the screen once they've changed.

LÖVE2D expects these functions to be implemented in `main.lua` and calls them internally; if we don't define them, it will still function, but our game will be fundamentally incomplete, at least if `update` or `draw` are missing!

pong0: Important Functions, p.2

- `love.graphics.printf(text, x, y, [width], [align])`
  - Versatile print function that can align text left, right, or center on the screen.

- `love.window.setMode(width, height, params)`
  - Used to initialize the window's dimensions and to set parameters like vsync (vertical sync), whether we're fullscreen or not, and whether the window is resizable after startup. Won't be using past this example in favor of the push virtual resolution library, which has its own method like this, but useful to know if encountered in other code.

pong1

"The Low-Res Update"


Untitled
Hello Pong!

pong1: Important Functions

- `love.graphics.setDefaultFilter(min, mag)`

-Sets the texture scaling filter when minimizing and magnifying textures and fonts; default is bilinear, which causes blurriness, and for our use cases we will typically want nearest-neighbor filtering (`'nearest'`), which results in perfect pixel upscaling and downscaling, simulating a retro feel.

- `love.keypressed(key)`

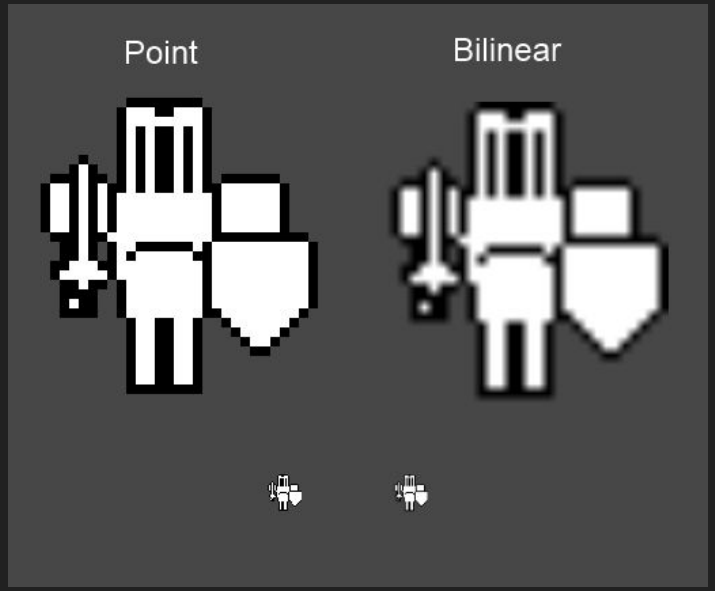
-A LÖVE callback function that executes whenever we press a key, assuming we've implemented this in our `main.lua`, in the same vein as `love.load(), love.update(dt), and love.draw()`.

- `love.event.quit()`

-Simple function that terminates the application.

Texture Filtering

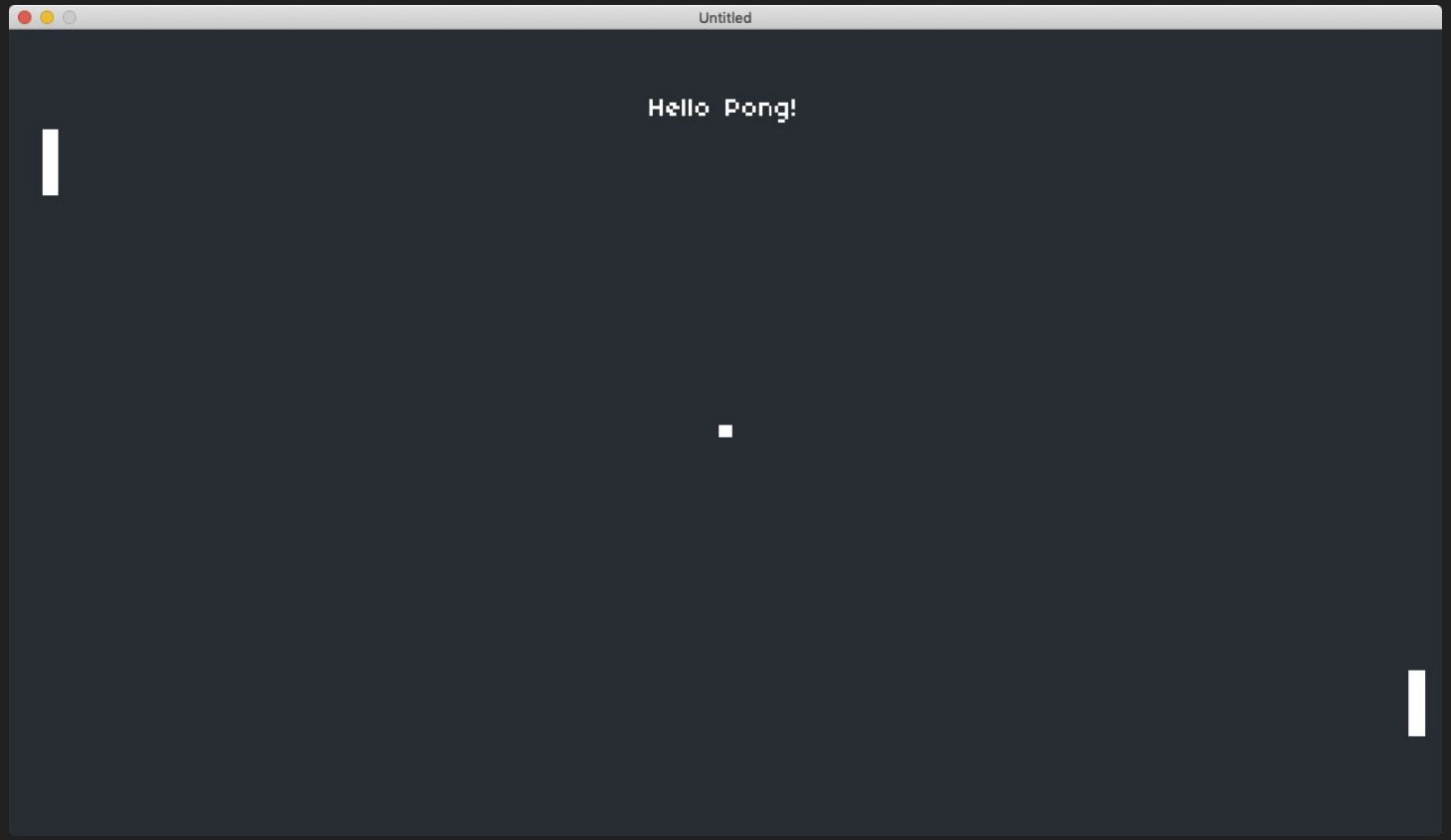
- “Point” == “Nearest Neighbor” (`'nearest'` in our source code)
- Bilinear/trilinear/anisotropic filtering cause blurriness in 2D, as seen below!



https://answers.unity.com/questions/602721/sprite-filtering-pointbilinear.html

pong2

“The Rectangle Update”


Untitled
Hello Pong!

pong2: Important Functions, p.1

- `love.graphics.newFont(path, size)`
  - Loads a font file into memory at a specific `path`, setting it to a specific `size`, and storing it in an object we can use to globally change the currently active font that LÖVE is using to render text (functioning like a state machine).

- `love.graphics.setFont(font)`
  - Sets LÖVE's currently active font (of which there can only be one at a time) to a passed-in `font` object that we can create using `love.graphics.newFont`.

- `love.graphics.clear(r, g, b, a)`
  - Wipes the entire screen with a color defined by an RGBA set, each component of which being from 0.0-1.0 (which maps to 0-255, for 8 bits per pixel component).

pong2: Important Functions, p.2

- `love.graphics.rectangle(mode, x, y, width, height)`
  - Draws a rectangle onto the screen using whichever our active color is (`love.graphics.setColor`, which we don't need to use in this particular project since most everything is white, the default LÖVE color). `mode` can be set to `'fill'` or `'line'`, which result in a filled or outlined rectangle, respectively, and the other four parameters are its position and size dimensions. This is the cornerstone drawing function of the entirety of our Pong implementation!

pong3

“The Paddle Update”

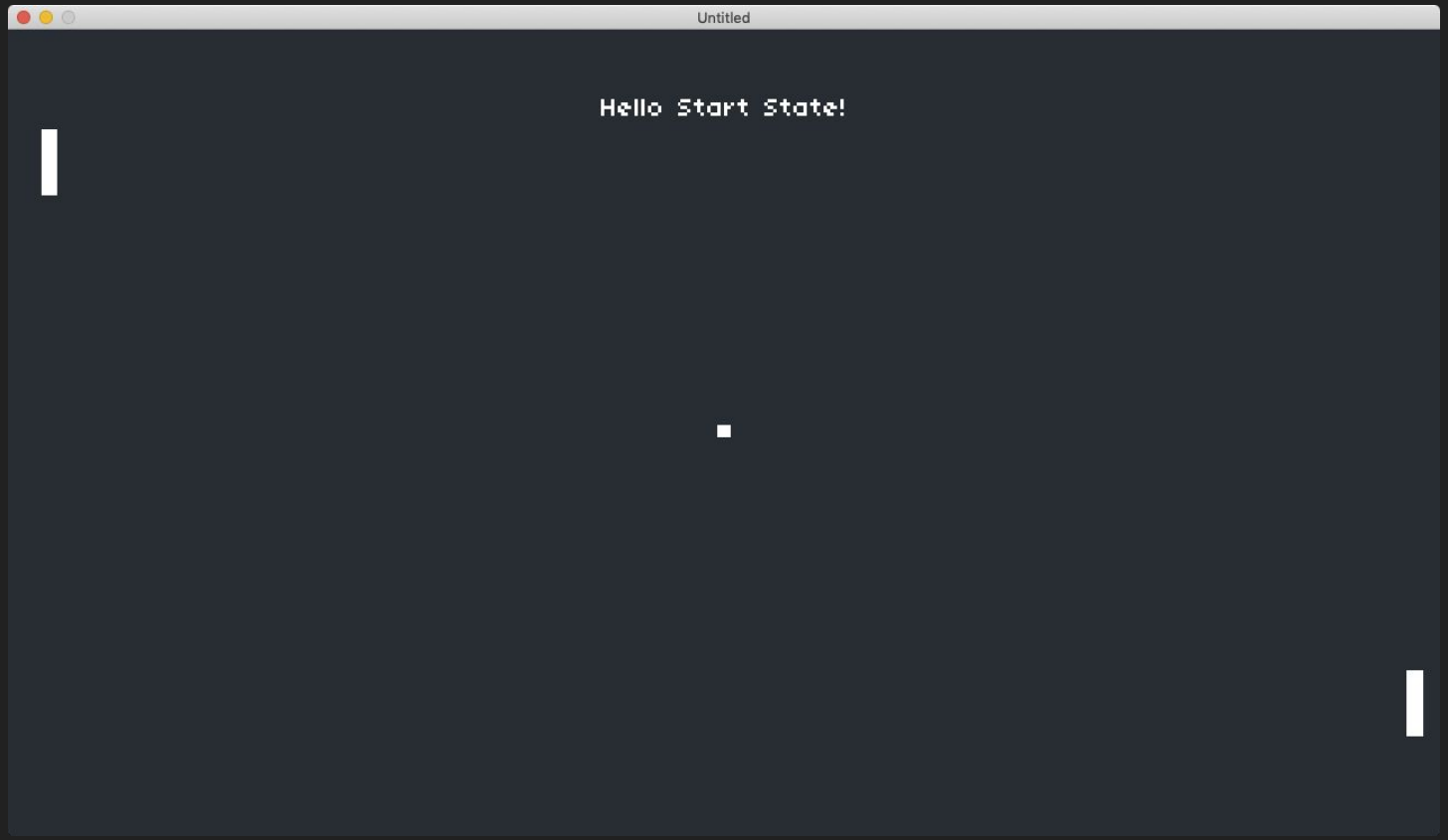

Untitled
Hello Pong!
0
0

pong3: Important Functions

- `love.keyboard.isDown(key)`
  - Returns true or false depending on whether the specified key is currently held down; differs from `love.keypressed(key)` in that this can be called arbitrarily and will continuously return true if the key is pressed down, where `love.keypressed(key)` will only fire its code once every time the key is initially pressed down. However, since we want to be able to move our paddles up and down by holding down the appropriate keys, we need a function to test for longer periods of input, hence the use of `love.keyboard.isDown(key)`!

pong4

“The Ball Update”


Untitled
Hello Start State!

pong4: Important Functions, p.1

- `math.randomseed(num)`
  - ”Seeds” the random number generator used by Lua (`math.random`) with some value such that its randomness is dependent on that supplied value, allowing us to pass in different numbers each playthrough to guarantee non-consistency across different program executions (or uniformity if we want consistent behavior for testing).

- `os.time()`
  - Lua function that returns, in seconds, the time since 00:00:00 UTC, January 1, 1970, also known as Unix epoch time (https://en.wikipedia.org/wiki/Unix_time).

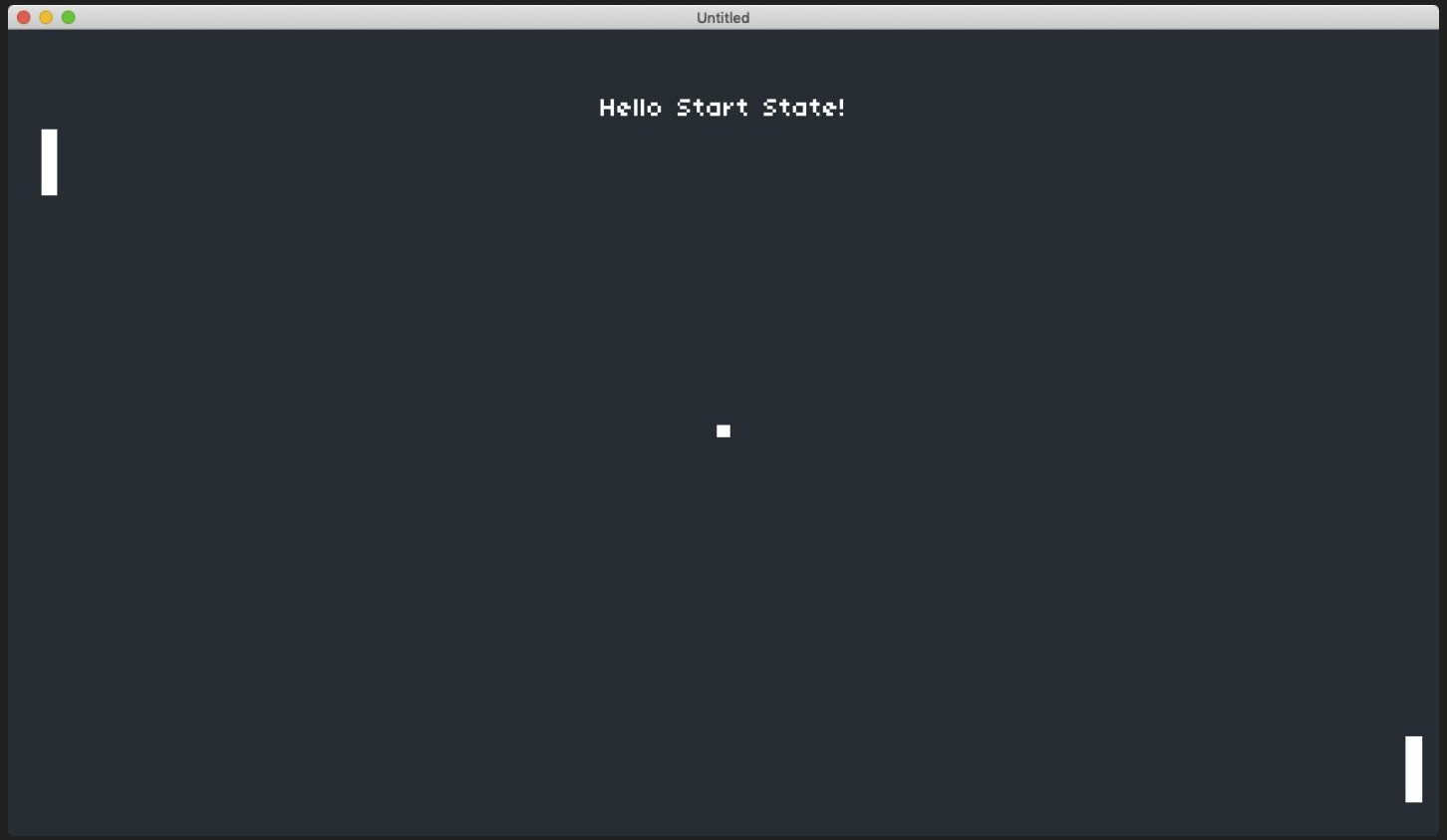
- `math.random(min, max)`
  - Returns a random number, dependent on the seeded random number generator, between `min` and `max`, inclusive.

pong4: Important Functions, p.1

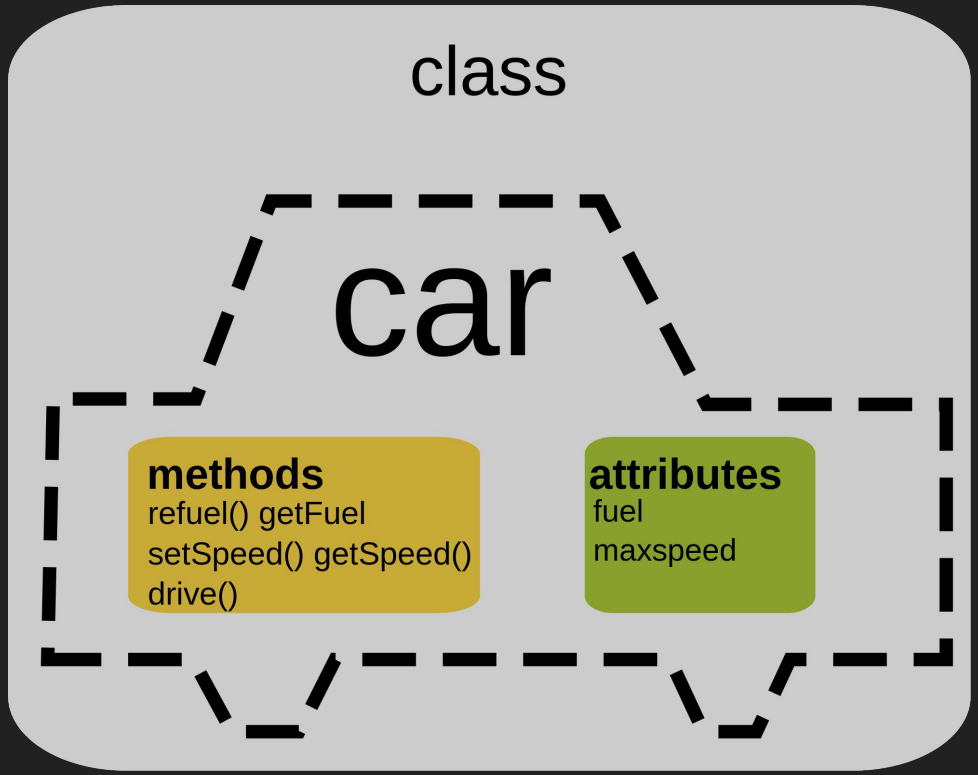
- `math.min(num1, num2)`
  - Returns the lesser of the two numbers passed in.
- `math.max(num1, num2)`
  - Returns the greater of the two numbers passed in.

pong5

"The Class Update"


Untitled
Hello Start State!

What is a class? p.1



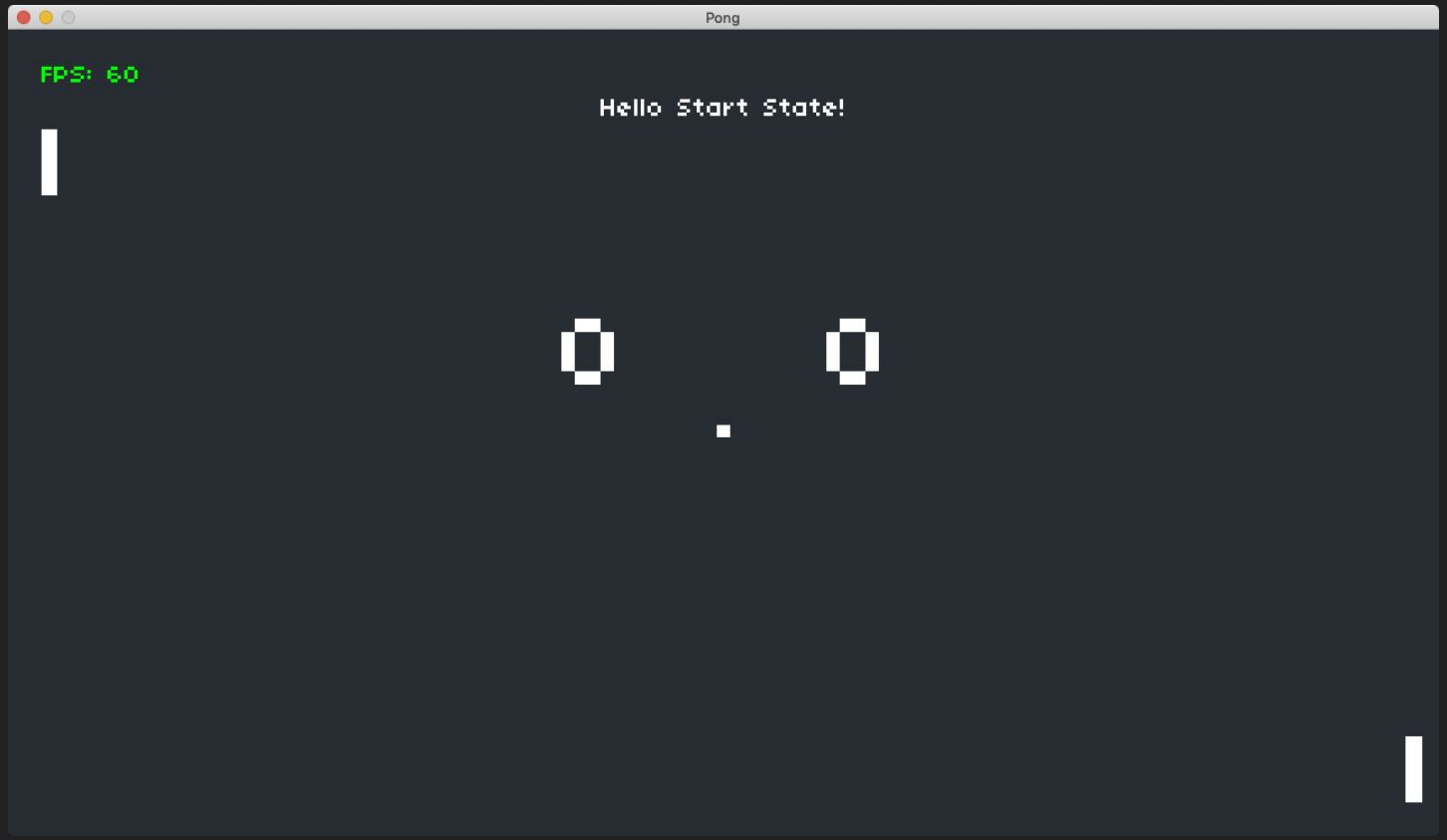
https://commons.wikimedia.org/wiki/File:CPT-OOP-objects_and_classes_-_attmeth.svg

What is a class? p.2

- Blueprints for creating bundles of data and code that are related.
- A “Car” class can have attributes that describe its brand, model, color, miles, and anything else descriptive; these are also known as “fields”.
- A “Car” class can also have “methods” that define its behavior, such as “accelerate”, “turn”, “honk”, and more, which take the form of functions.
- Objects are instantiated from these class blueprints, and it’s these concrete objects that are the physical “cars” you see on the road, as opposed to the blueprints that may exist in the factory.
- Our Paddles and Ball are perfect simple use cases for taking some of our code and bundling it together into classes and objects.

pong6

"The FPS Update"

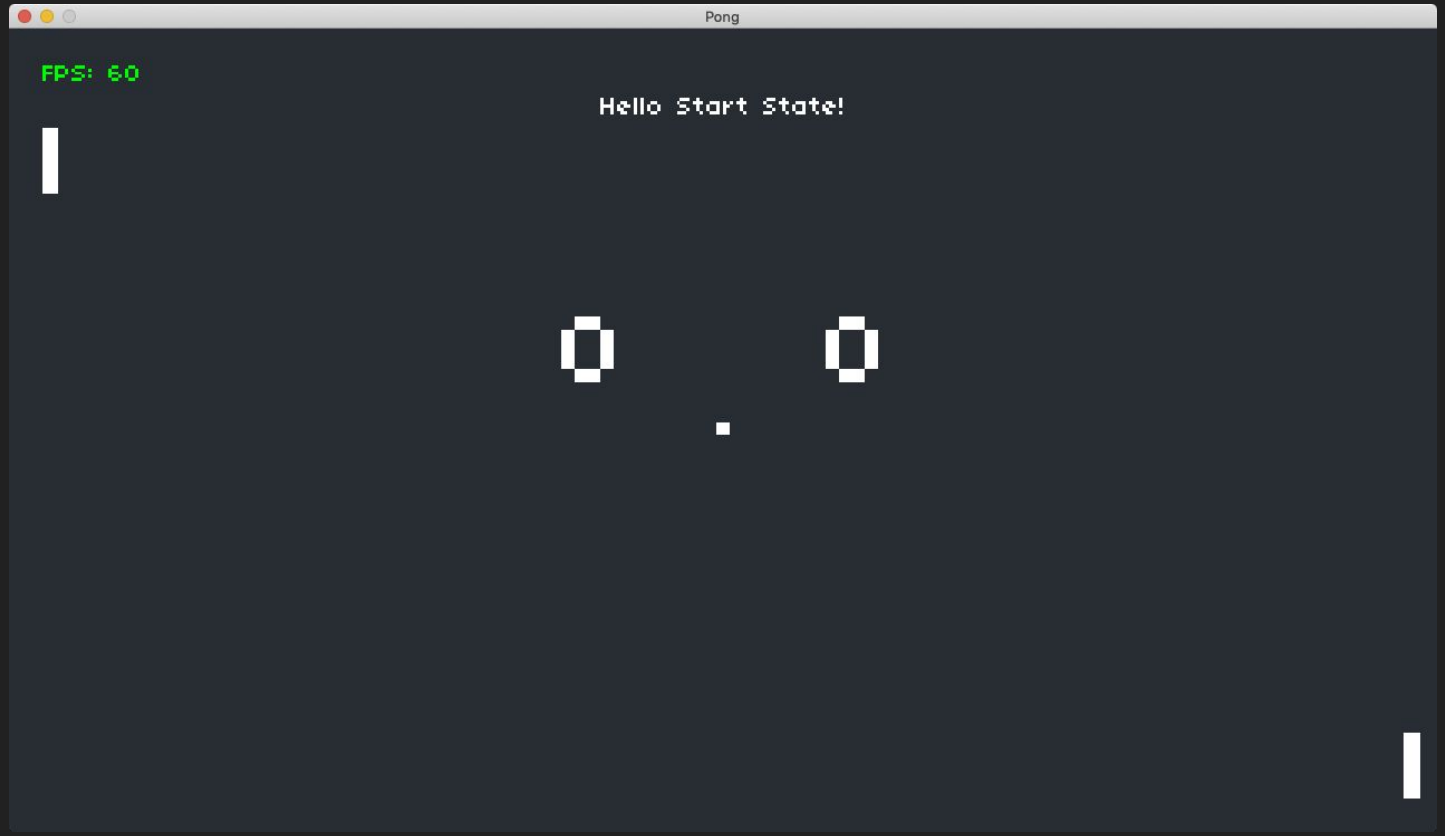

Pong
FPS: 60
Hello Start State!
0
0

pong6: Important Functions, p.1

- `love.window.setTitle(title)`
  - Simply sets the title of our application window, adding a slight level of polish.
- `love.timer.getFPS()`
  - Returns the current FPS of our application, making it easy to monitor when printed.

pong7

“The Collision Update”

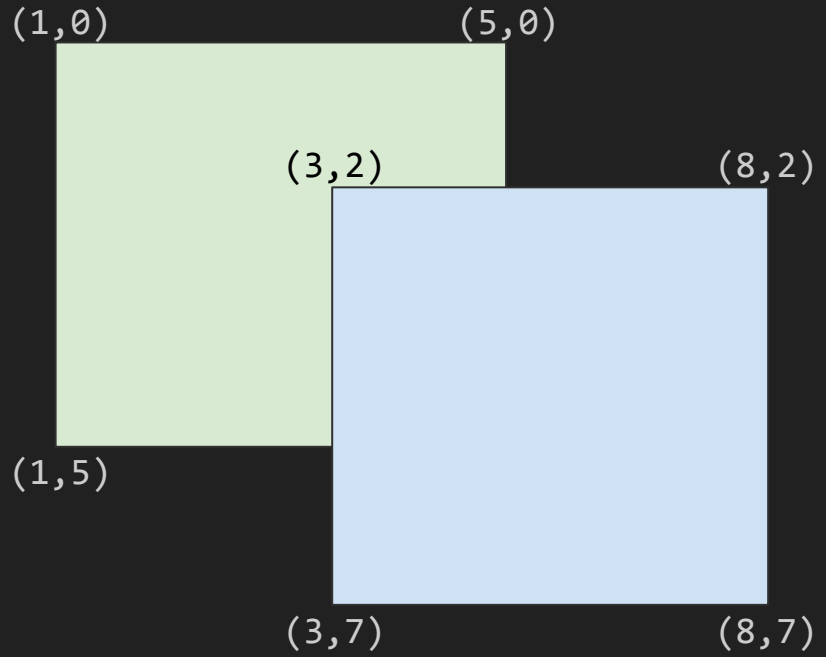

Pong
FPS: 60
Hello Start State!
0
0

AABB Collision Detection, p.1

- Relies on all colliding entities to have “axis-aligned bounding boxes”, which simply means their collision boxes contain no rotation in our world space, which allows us to use a simple math formula to test for collision:

```
if rect1.x is not > rect2.x + rect2.width and
   rect1.x + rect1.width is not < rect2.x and
   rect1.y is not > rect2.y + rect2.height and
   rect1.y + rect1.height is not < rect2.y:
      collision is true
else
   collision is false
```

AABB Collision Detection, p.2



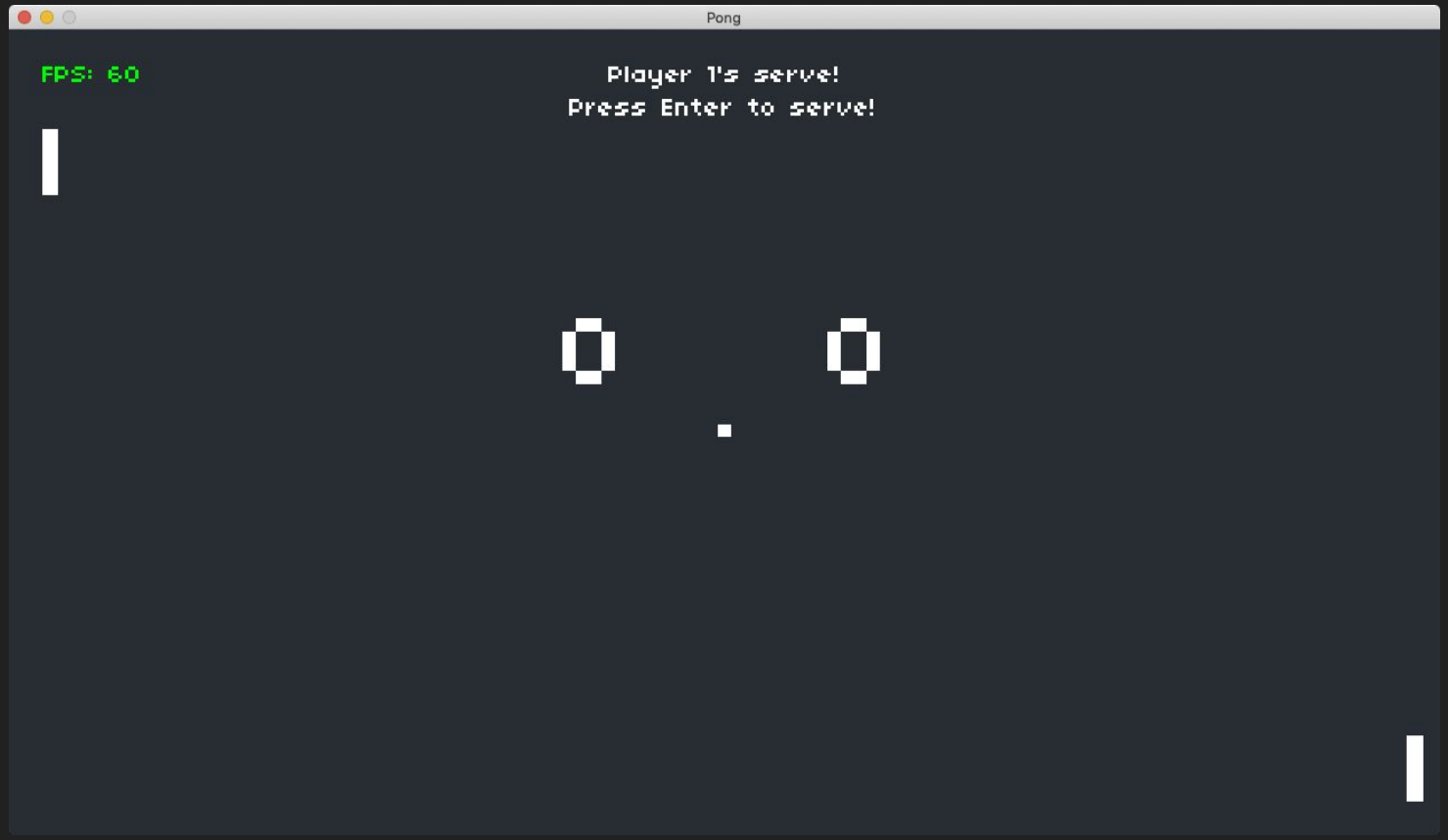
pong8

"The Score Update"

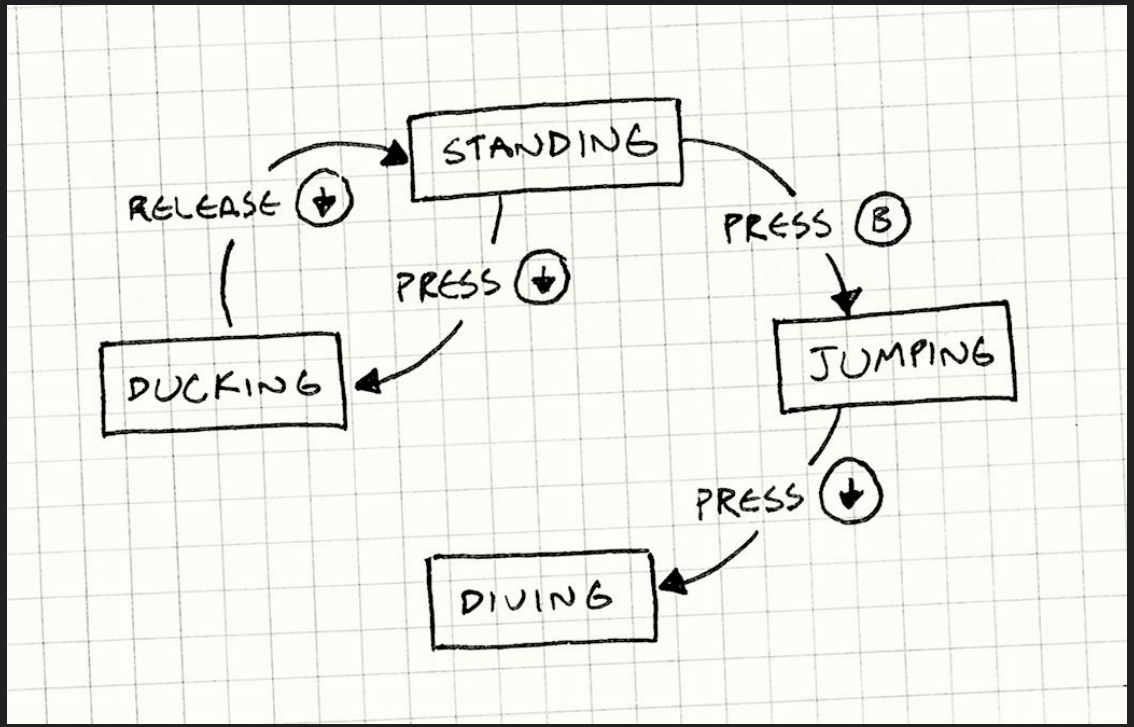

Pong
FPS: 60
1
0

pong9

"The Serve Update"


Pong
FPS: 60
Player 1's serve!
Press Enter to serve!
0
0

State Machine



Credit: Robert Nystrom

http://gameprogrammingpatterns.com/state.html

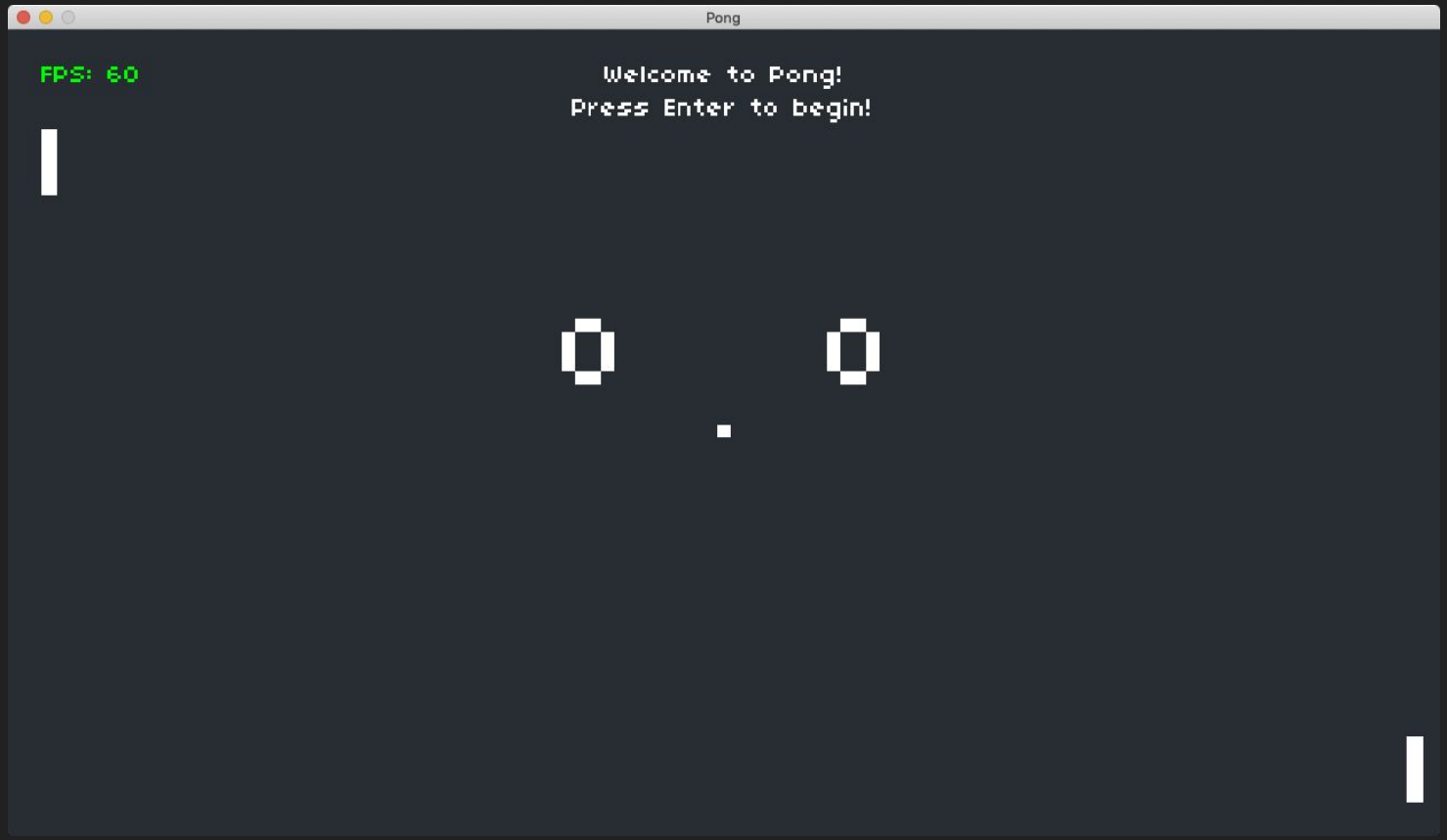
pong10

"The Victory Update"


Pong
FPS: 60
Player 1 wins!
Press Enter to restart!
10
9

pong11

"The Audio Update"


Pong
FPS: 60
Welcome to Pong!
Press Enter to begin!
0
0

pong11: Important Functions

- `love.audio.newSource(path, [type])`
  - Creates a LÖVE Audio object that we can play back at any point in our program. Can also be given a “type” of “stream” or “static”; streamed assets will be streamed from disk as needed, whereas static assets will be preserved in memory. For larger sound effects and music tracks, streaming is more memory-effective; in our examples, audio assets are static, since they’re so small that they won’t take up much memory at all.

bfxr, p.1

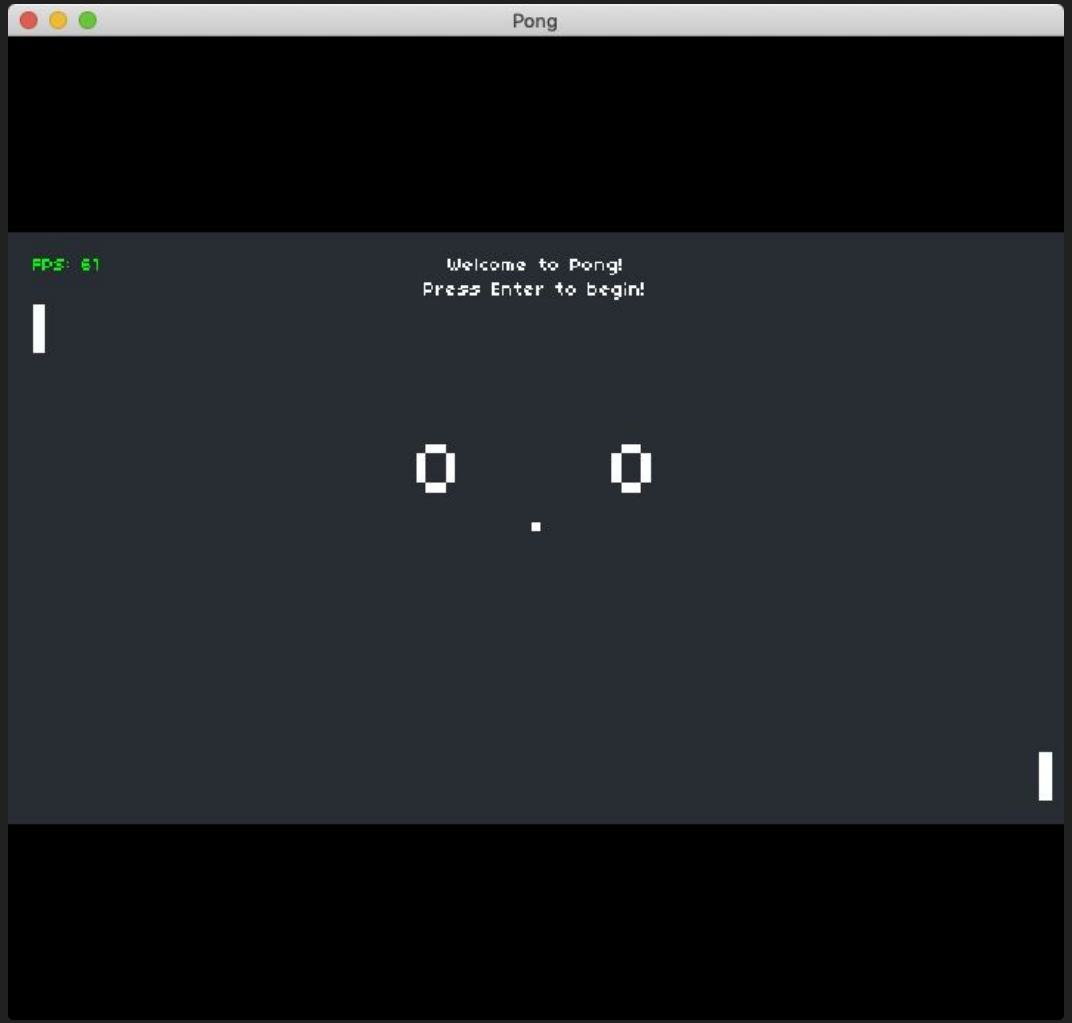

Bfxr
Pickup/Coin
Laser/Shoot
Explosion
Powerup
Hit/Hurt
Jump
Blip/Select
Randomize
Mutation
Create New Sound
Apply Synth
Revert Synth
Duplicate Synth
Jump
Pickup_Coin
Laser_Shoot
Explosion
Randomize
Randomize2
Randomize3
Randomize4
Randomize5
Randomize6
Synth
Mixer
Triangle
Sin
Square
Saw
Breaker
Tan
Whistle
White
Pink
Attack Time
Sustain Time
Punch
Decay Time
Compression
Frequency
Frequency Cutoff
Frequency Slide
Delta Slide
Vibrato Depth
Vibrato Speed
Harmonics
Harmonics Falloff
Pitch Jump Repeat Speed
Pitch Jump Amount 1
Pitch Jump Onset 1
Pitch Jump Amount 2
Pitch Jump Onset 2
Square Duty
Duty Sweep
Repeat Speed
Flanger Offset
Flanger Sweep
Low-pass Filter Cutoff
Play On Change
Play
Master Volume
Export Wav
Load from Disk
Save to Disk
Copy
Paste
Copy Link
About

bfxr, p.2

- Simple sound-generating program, freely available on major OSes.
- Used to generate all sound effects for our Pong example and most examples going forward.
- https://www.bfxr.net/

pong12

"The Resize Update"


Pong
FPS: 61
Welcome to Pong!
Press Enter to begin!
0
0

pong12: Important Functions

- `love.resize(width, height)`
  - Called by LÖVE every time we resize the application; logic should go in here if anything in the game (like a UI) is sized dynamically based on the window size. `push:resize()` needs to be called here for our use case so that it can dynamically rescale its internal canvas to fit our new window dimensions.

pong13*

"The AI Update"

Pong Assignment

- Implement a basic AI that tracks the ball as Player 2.
- Feel free to make it as easy or difficult as you wish!
- Make sure the tracking movement is smooth and not instantaneous.

This is CS50.