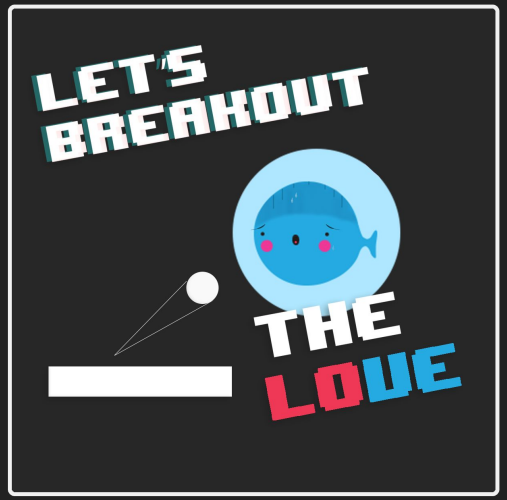


WHO AM I?

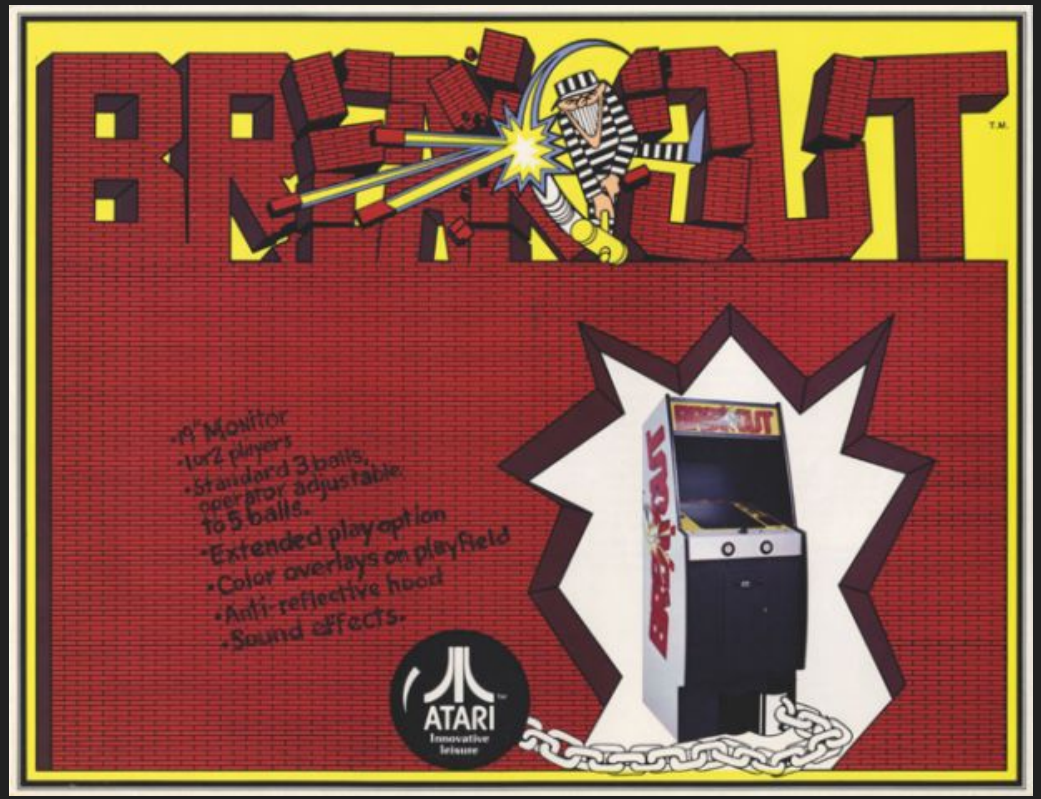
I’m **Gabriele Pappalardo**, 20 years old

I am a:

- **CS Student** at Unipi
- **UX Designer** at **ntop**
- an aspiring **Game Developer**
- but most importantly…

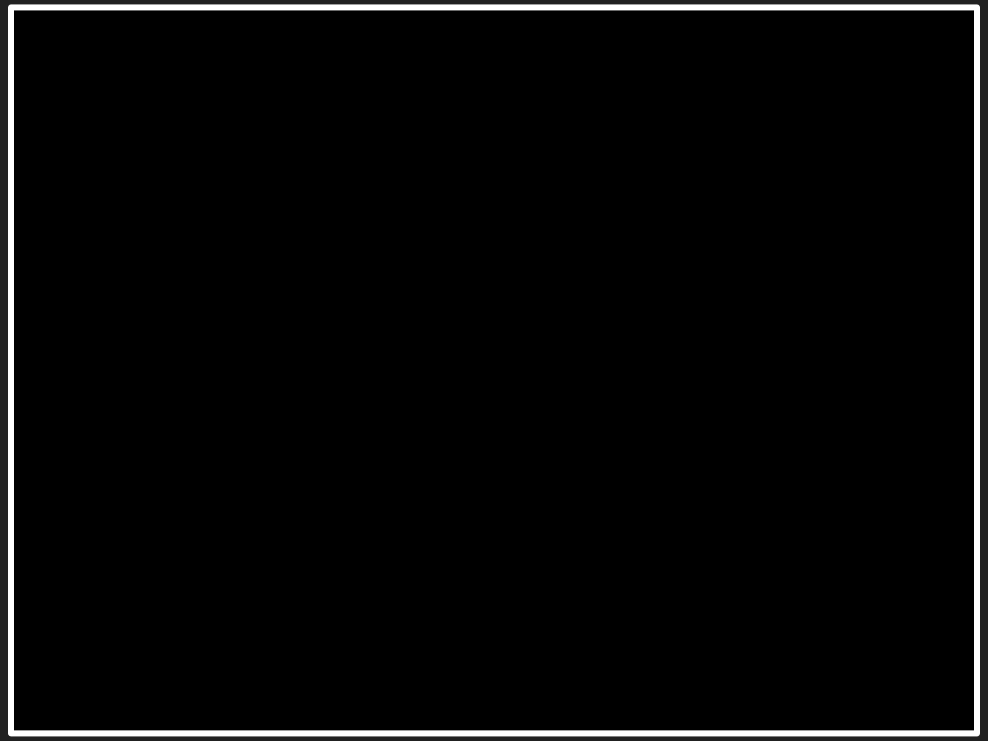
- …a **videogamer**!

WHAT WE'RE GONNA CREATE?



BREAKOUT (ARCADE)

FINAL (EXPECTED) RESULT



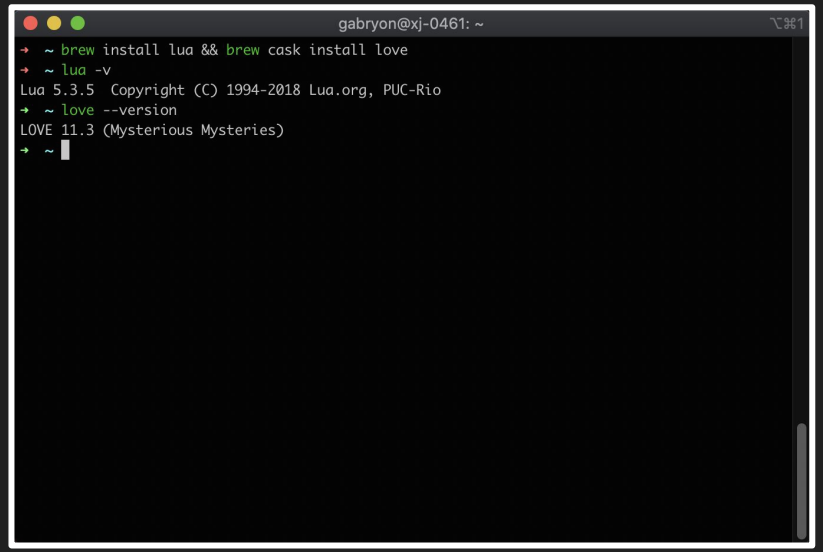
INSTALLING THE TOOLS (1)

To create our clone we need:

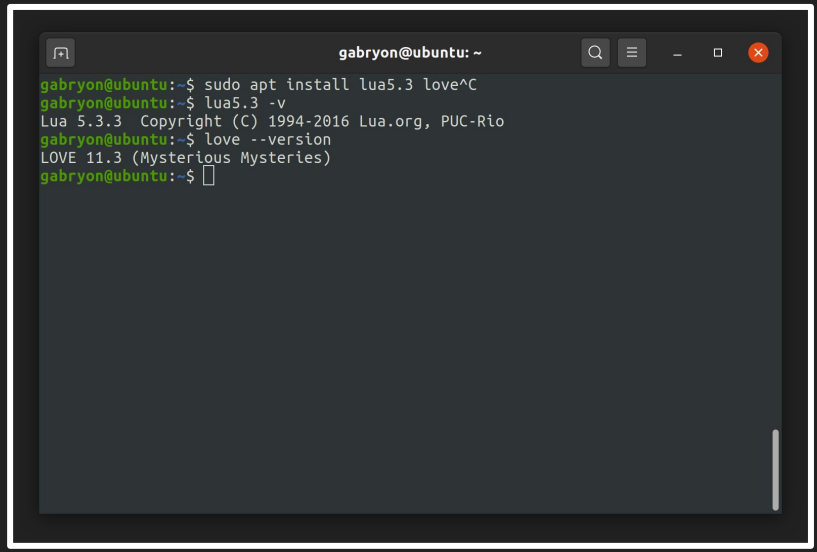
- **Lua:** scripting language (version 5.3 it's fine)
- **LÖVE2D**: the game framework (version 11.3^)
- and, of course, a **will of fire**!

I'm gonna use **Mac OS Catalina** through this session, but you can also use **Linux** (any distro you like) or **Windows** (7, 8(?), 10).

INSTALLING THE TOOLS (2)



MacOS (10.15.4)



Ubuntu 20.04 (Linux)

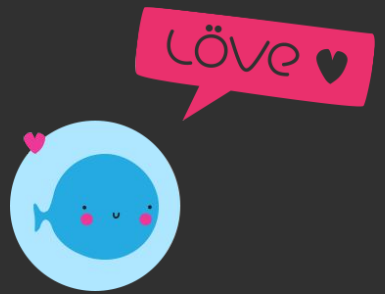
THE LUA LANGUAGE

*"**Lua** is a **powerful**, efficient, lightweight, embeddable scripting language. It supports procedural programming, object-oriented programming, functional programming, data-driven programming, and data description."*
- **Lua Website**

THE LÖVE2D FRAMEWORK

"LÖVE2D is a framework for making 2D games in the Lua programming language. LÖVE is totally free, and can be used in anything from friendly open-source hobby projects, to evil, closed-source commercial ones"

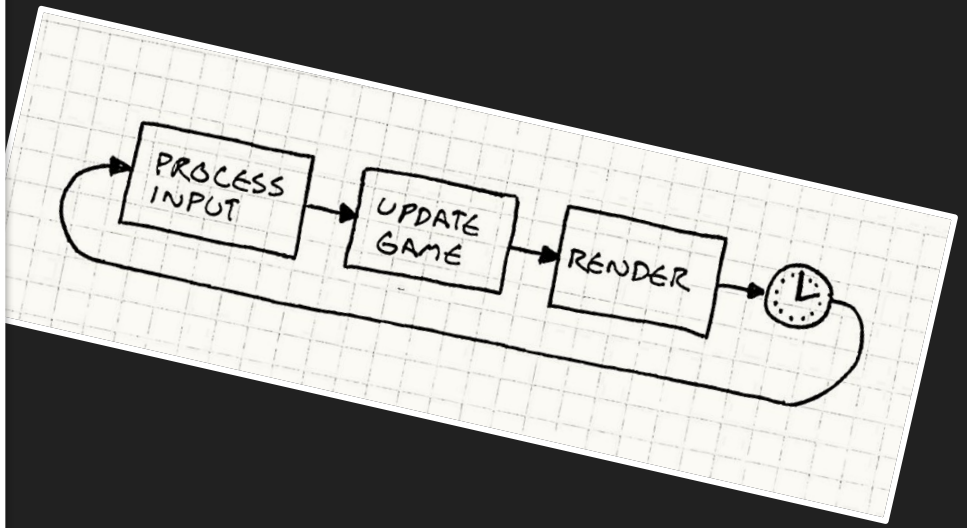
For the LÖVE *2D documentation follow this link:*
https://love2d.org/wiki/Main_Page



THE GAME LOOP

A **game loop** is a *loop* runs continuously during the game, it consists of 3 steps:

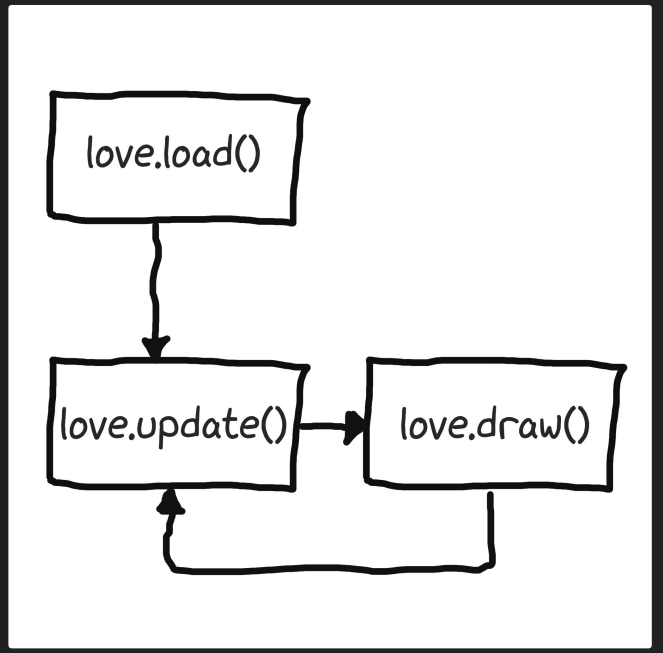
1. **get inputs** from the user;
2. **update** the game;
3. **render** the game;



THE MAIN LÖVE FUNCTIONS...

In this talk we are gonna use these functions (mainly):

- love.**load**( )
- love.**update**( )
- love.**draw**( )



...AND THE OTHER (NOT LESS) IMPORTANT (1)

Interface for game window (**Window** Module):

- love.**window**.setTitle()
- love.**window**.setMode()

Interface for game graphics (**Graphics** Module):

- love.**graphics**.setColor(r, g, b, a)
- love.**graphics**.rectangle('fill', x, y, w, h)

Interface for math (**Math** Module):

- love.**math**.random(min, max)
- ~~love.**math**.setRandomSeed(seed),~~

  https://love2d.org/wiki/love.math.random

...AND THE OTHER (NOT LESS) IMPORTANT (2)

Interface for game audio (**Audio** Module):

- love.**audio**.play(soundToPlay)
- love.**audio**.newSource(path, type)

Interface for keyboard inputs (**Keyboard** Module):

- love.**keyboard**.isDown(key)

LET’S START!

PART 1: THE PADDLE

LEFT & RIGHT MOVEMENT

In our game we need our paddle to move **left** or **right**. The paddle movements is really simple and it needs simple basis of **Linear Algebra** (2D) and **Physics** (always in 1D/2D).

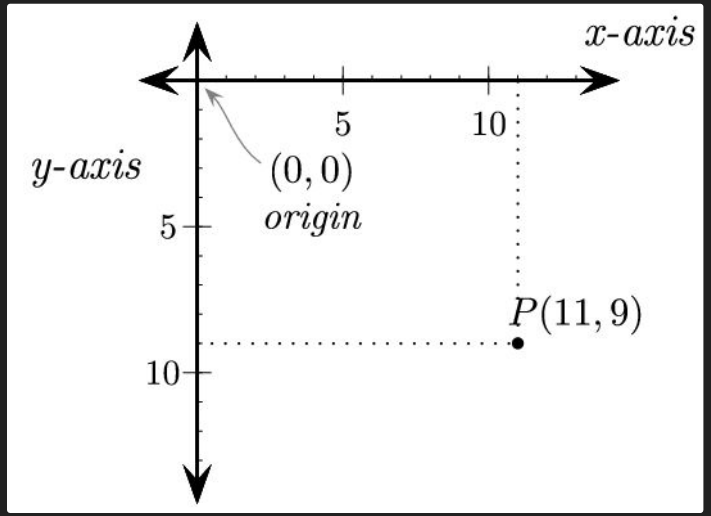
1D PHYSICS

Move the paddle (and other objects) is very simple, we use the “**Linear Motion**”.

In our game the paddle moves at constant speed only in **one** dimension: the x-axis.

We need this simple equation:

$$x(t) = v_x(\Delta t) + x(t-1)$$



PART 2: THE BALL

2D PHYSICS

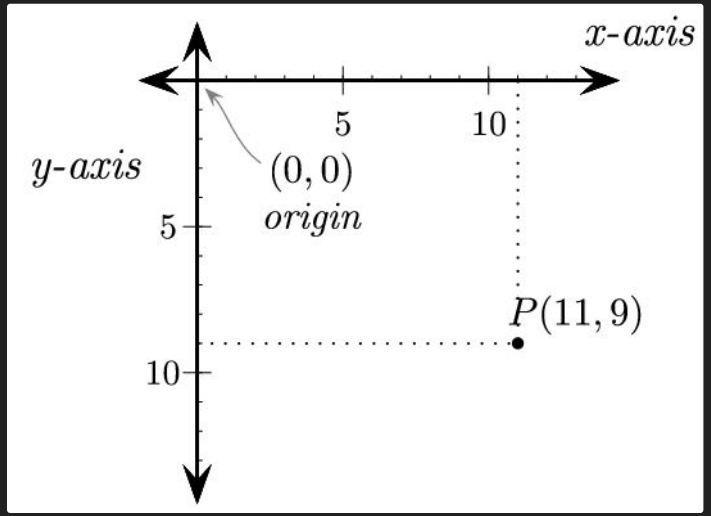
Moving the ball works the same as moving the paddle (more or less).

The ball moves around in the screen: up and down, left and right. So, we need to use also the y-axis this time.

We need these 2 equations:

$$x(t) = v_x(\Delta t) + x(t-1)$$

$$y(t) = v_y(\Delta t) + y(t-1)$$



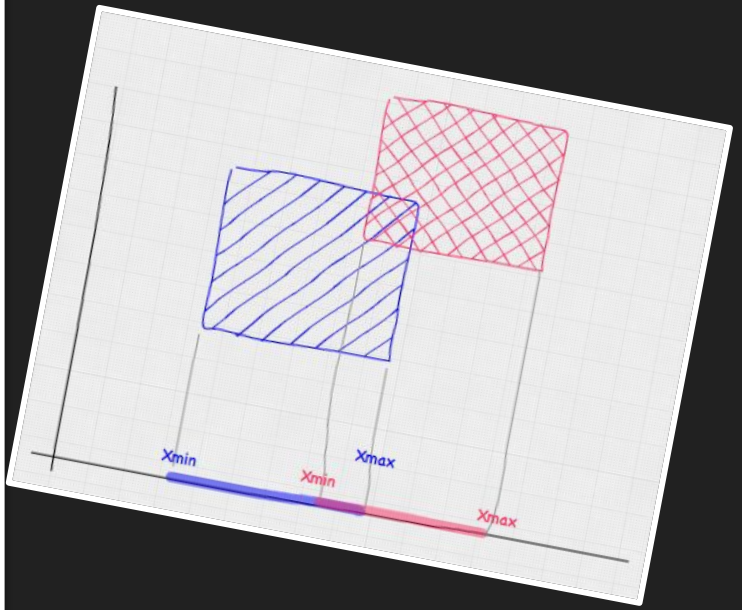
COLLISION TIME!

AABB: Axis-Aligned Bounding Box

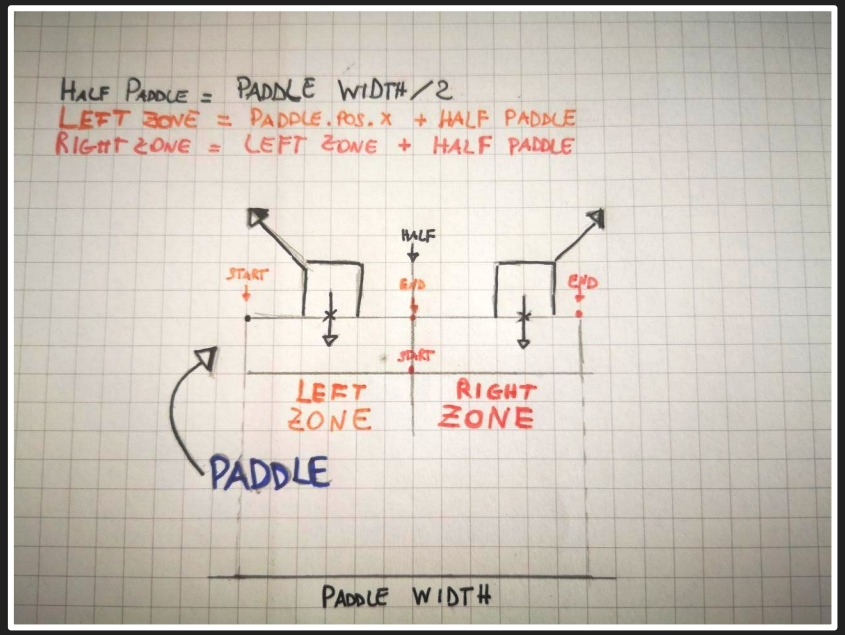
“**AABB** stands for axis-aligned bounding box, a rectangular collision shape aligned to the base axes of the scene, which in 2D aligns to the x and y axis.

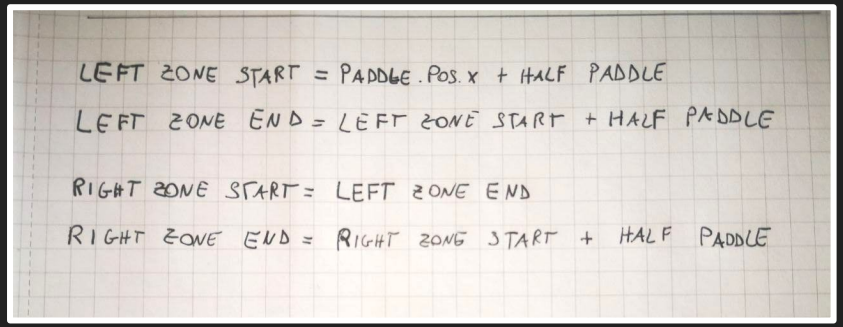
Being axis-aligned means the rectangular box has no rotation and its edges are parallel to the base axes of the scene (e.g. left and right edge are parallel to the y axis).

The fact that these boxes are always aligned to the axes of the scene makes calculations easier. ”



BALL && PLAYER COLLISION





PART 3: THE BLOCKS

THANK YOU!

Made with ❤️ by @gabryon99

RESOURCES

- thehistoryofhowweplay.wordpress.com/2018/12/29/a-breakout-story/
- love2d.org/wiki
- gameprogrammingpatterns.com/
- learnopengl.com/In-Practice/2D-Game/Collisions/Collision-detection
- developer.mozilla.org/en-US/docs/Games/Techniques/2D_collision_detection