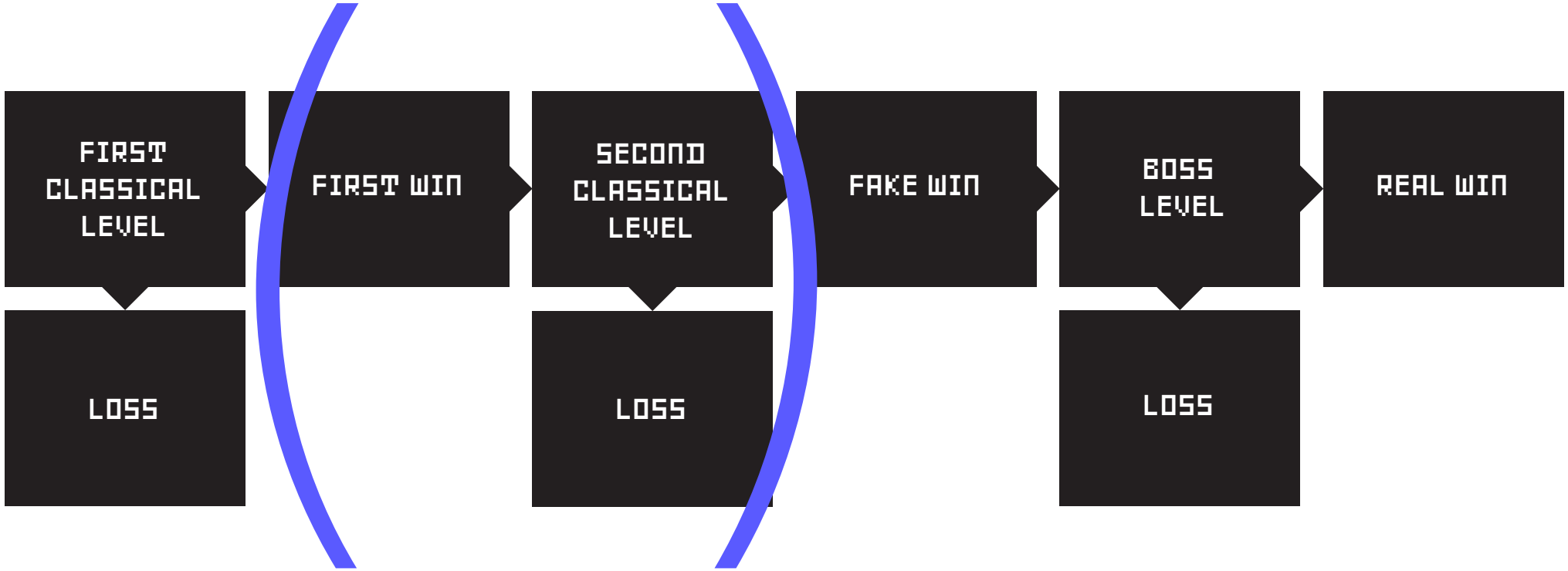
PAC-MAN PROJECT

DEFENSE

Arthur AMORIM
Luc JIANG
Liam RAMPON

FLOW OF THE GAME



CODE STRUCTURE

MVC pattern and object oriented not ECS system
because too complicated and not too many entities

► For a classic stage

- Entities (Pac-Man and ghosts)
- Stage containing Squares

► For the boss level

- One base “Entity” class (move() and render()) + heirs

PROGRAM FLOW

Same flow for every level:

- Call objects **constructors**
- **while (!quit)**
  - **move** entities
  - **collide** entities
  - **render** stage

Separate steps ► we can act on each of them independently

MOVING IN THE MAZE

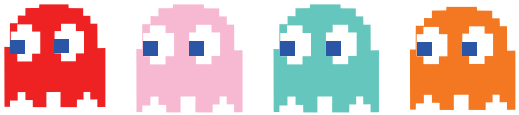
► **Staying in the maze**

Matrix of squares ► an entity moves between two squares' centers and the path is recalculated once on a new center

► **Moving the right way**

Path calculated through **pathfinding** function (function pointer easily changed depending on the entity and its state)

MOVING IN THE MAZE



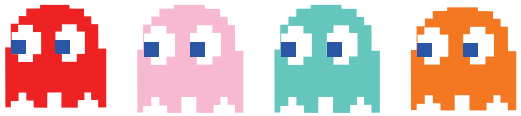
► **Pros:**

- wall detection is easy,
- entity collision detection is fast,
- pathfinding is pretty fast.

► **Cons:**

- collision might fail,
- a teleporter is much harder to code,
- movements can only be vertical or horizontal.

MOVING WITH THE BOSS ...



► **All objects:**

"Classic" entities with variable position and speed vectors.
Every frame: position += time_delta*speed.

► **Pros:**

- more intuitive,
- complex movement are easier (rotation while translating).

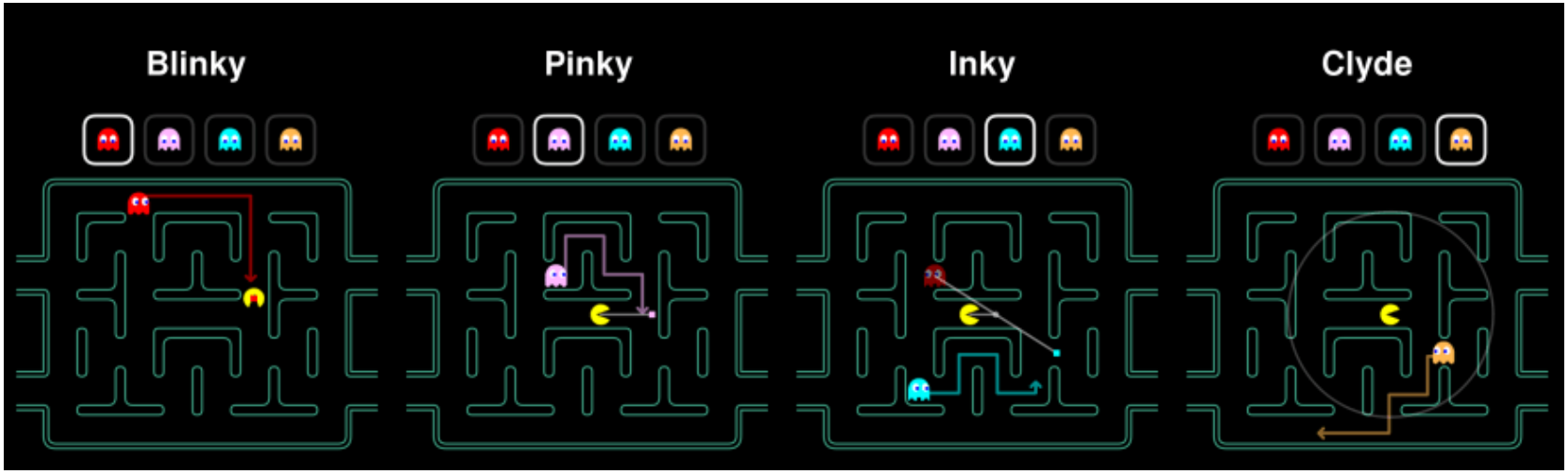
► **Cons:**

- AABB collision detection (a bit harder to code),
- bugs sometimes when encountering wall.

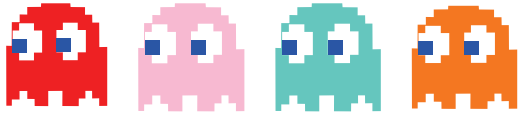
PATH FINDING IN THE MAZE

► **For ghosts:**

Same as in the original Namco game,
Among **available squares**, choose the one closest to a specific **target square**.



PATH FINDING IN THE MAZE



► **For dead ghosts:**

- **BFS** (concurrent **Dijkstra**) to spawn,
- A* unnecessary (small grid ≠ graph and labyrinth).

► **For Pac-Man:**

- If no intersection, then go straight;
- If intersection, then go in direction of last key input;
- If trying to go through a wall, then stop.

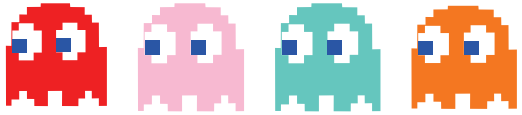
PATH FINDING IN THE BOSS STAGE

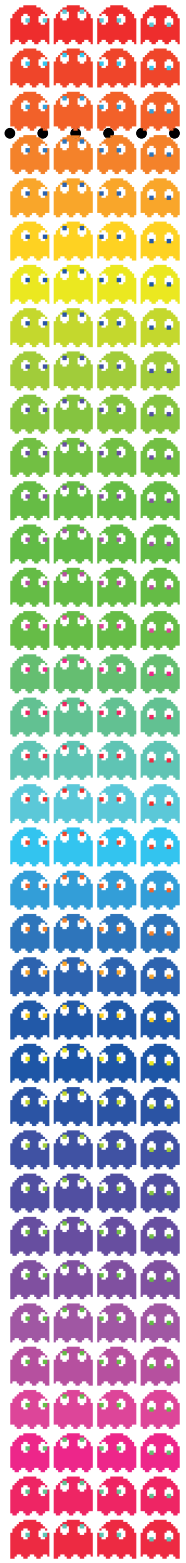
► **Most objects:**

Find **vector** from position to target point, **normalize** it, multiply by the entity's **speed**.

- boss does it only when rumbling;
- pong ball once, at spawn, then only makes -1*speed(x/y) when wall or racket encountered;
- pong racket targets same x as it has and y of the ball, then goes back to mid-height;
- bullets spawn in a circle around the boss and target the boss spawn position;

GRAPHICS AND ANIMATION

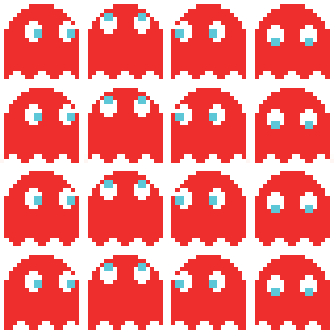


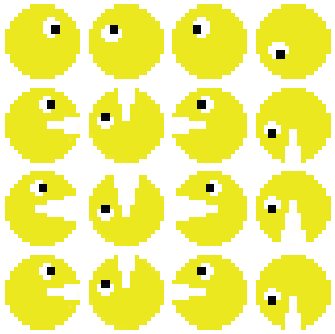


► **For entities:**

- Use of **sprites**,
- Horizontal is direction, vertical is time,
- Easy to rotate, flip, change directions,
- Rather simple to create new sprites (mostly 20*20 px).

► Much better than geometrical rendering

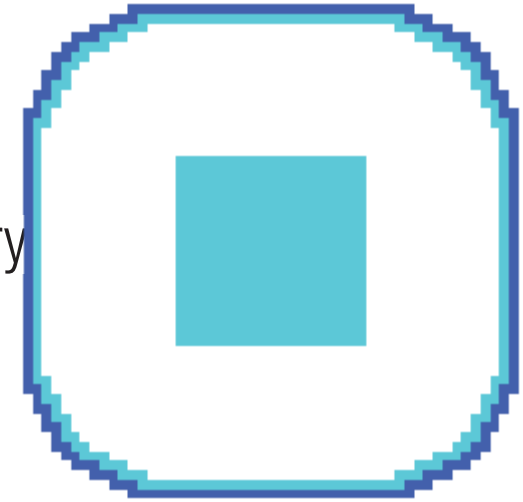




YOU WON

... ?

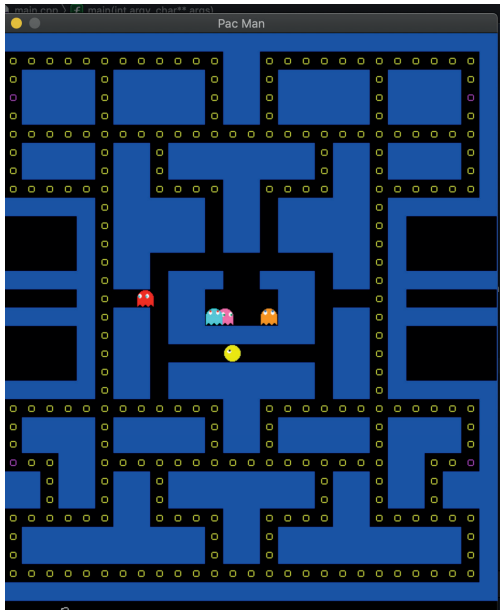
GRAPHICS AND ANIMATION

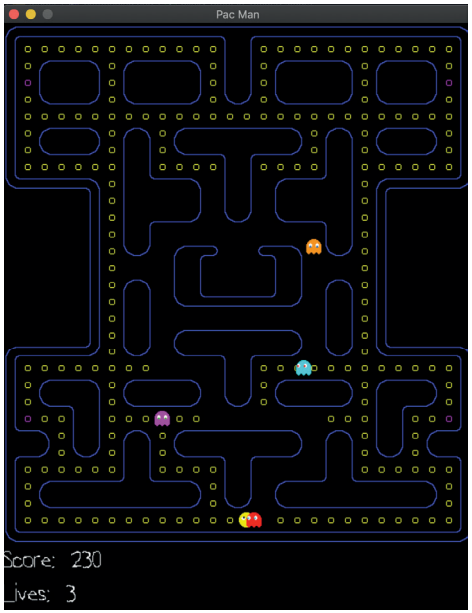


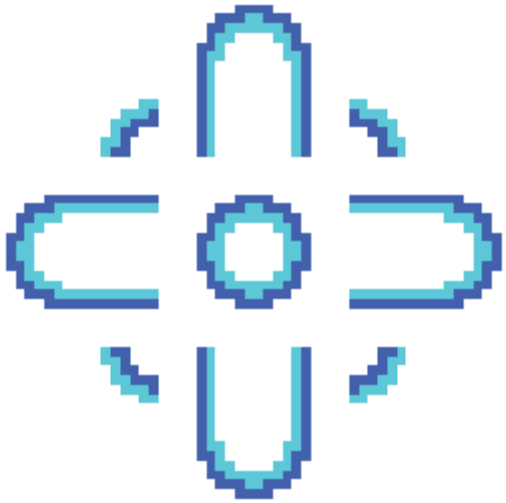
► **For the walls**

- A **generalized method** that would work with any layout,
- >> Several bits of wall pasted one over the other with a binary function to select what shape

► Much better than geometrical rendering







REMARKS

► **SDL Wrapping:**

SDL is a pretty tough library to learn how to use properly so we relied on a few «wrapped» programs found on some tutorials :

- **LTexture** loads a picture and renders it at a given point on the screen.
- **LTimer** is a timer...
- **FPSCapper** restricts FPS (because VSync isn't enabled on all devices). Otherwise on Mac, it would run at 1200 fps.

► **An unforeseen difficulty:**

It turns out that it's actually much **harder to distibute** our game than if it was made with Python Scripts for instance.