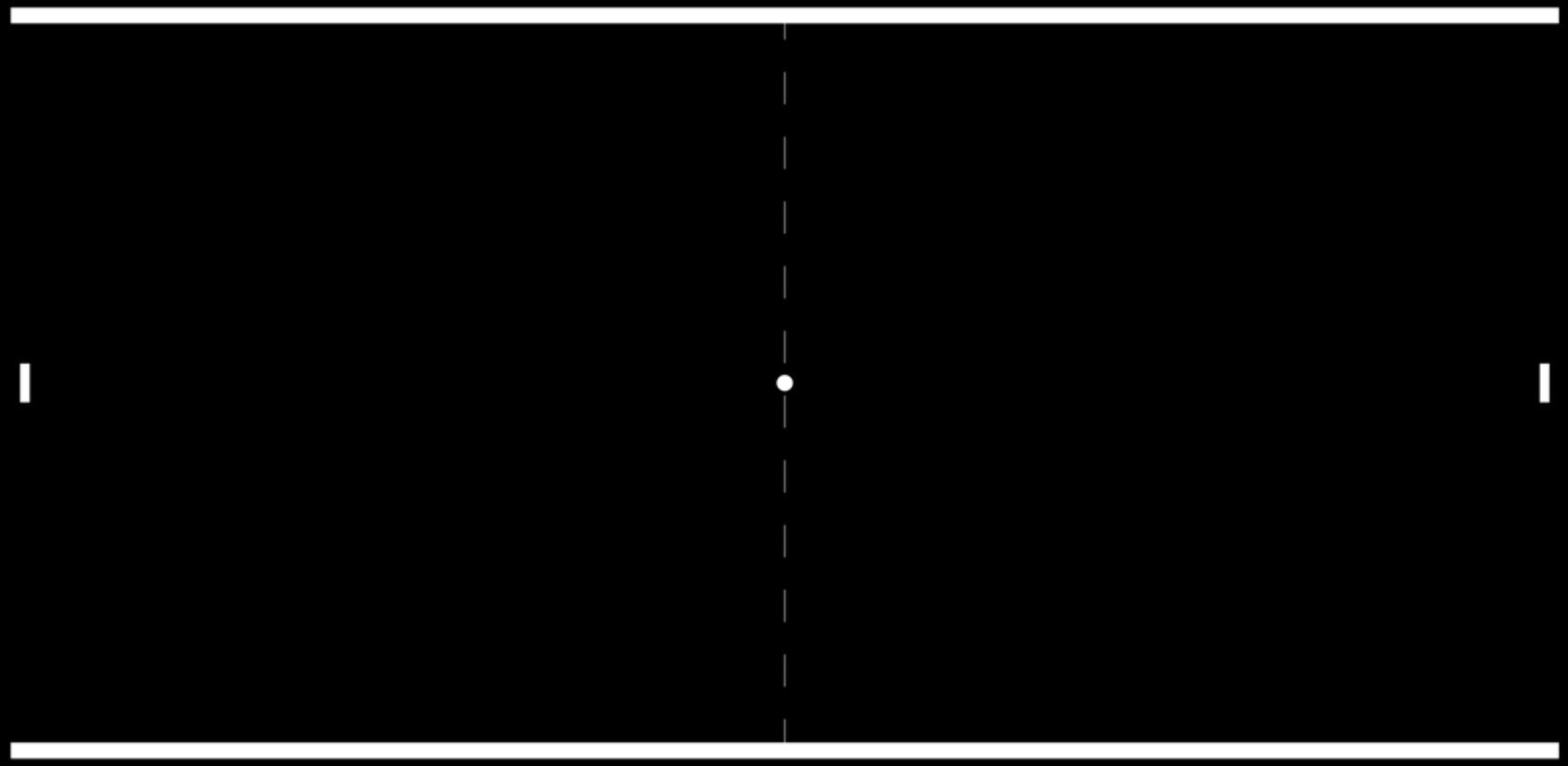
PONG

SESSION 2

PROJECT SO FAR

From the last session you should have a static game window, as below:



WHAT ARE WE GOiNG TO DO TODAY?

- Animate the canvas
- Implementing collision detection
- Game logic & end game scenarios

SESSION MATERIALS

FORK OR DOWNLOAD THE GITHUB REPO:
HTTPS://GITHUB.COM/INFPALS/IP2023-BIG-PROJECT-1-UPDATED-TEMPLATE

ATTENDANCE FORM

Please fill in to let us know you came and that we should keep planning similar events.

EVENT LISTENER

- Event listeners call a function when an action occurs on the event target.

e.g. **canvas.addEventListener("click", (e) => {...})** will call the inner lambda function when the canvas is clicked. The parameter e is the event being passed to the function so you can extract the coordinates of a mouse click (e.clientX and e.clientY) or value of key press (e.key), etc.

The (e) => {...} function is also known as an arrow function in JS. Learn more here:
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Functions/Arrow_functions

UPDATE LOOP

Currently all of the classes draw but never update. We will need to create update functions.

Each update method will take a dT parameter.

We will start by animating the ball:

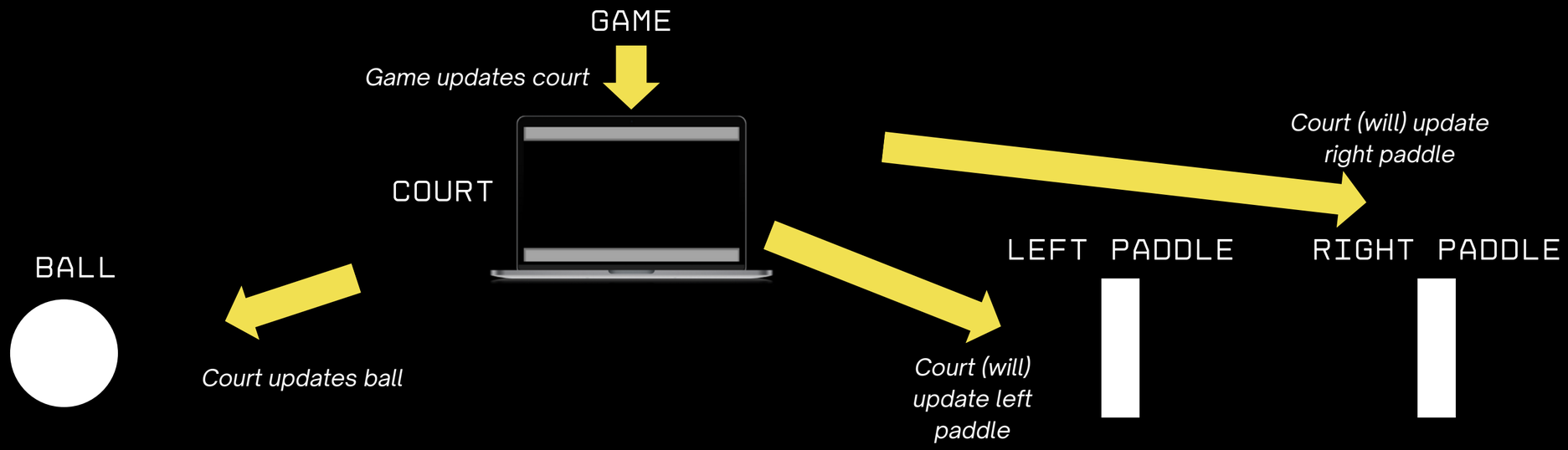
- First the ball needs a velocity vector (e.g. {x:-100, y:100}) and an acceleration (this could be a vector, but a magnitude is fine, e.g. 2), add these as variables in the constructor.
- Now create the update method for ball
  - The x and y coordinates of the ball can be updated to be products of velocity, acceleration and dT
    - *this.y += this.velocity.y * this.acceleration * dT*
    - *this.x += this.velocity.x * this.acceleration * dT*

UPDATE LOOP (CONT.)

The ball.update(dT) method is never being called.
To fix this:

- Create an update method in court and call update on the ball
- Then update the court from the *setInterval* game loop where we wrote the //update comment.

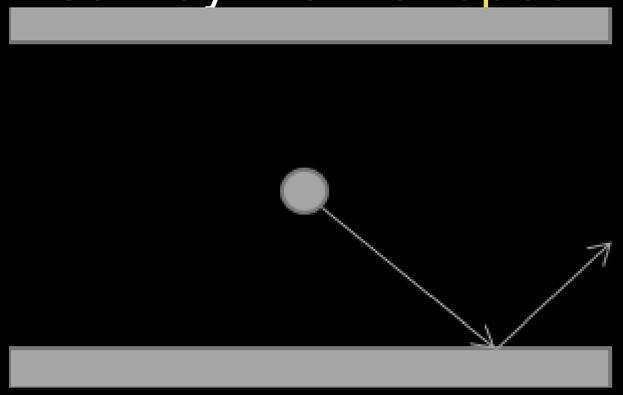
The following diagram shows the hierarchy of method calls:



BOUNDS

If successful, your ball should fly off the screen (depending on the initial speed).
We need bounds!

- In the court class add a getter method (*see below*) to return the pixel bounds for upper, lower, left and right.
  - Now a specific bound can be referenced by the direction, e.g. *this.court.bounds.lower*
- Now use the bounds of the court to reverse the y velocity if the ball hits the upper or lower borders.
- Take care to include the radius in your comparison statements, as the y position is the centre of the circle.
- You may wish to reposition the ball if it hits the bounds to prevent drifting.



```
get bounds () {
    return {
        upper: SETTINGS.courtMarginY + SETTINGS.wallSize,
        lower: this.canvas.height - (SETTINGS.courtMarginY + SETTINGS.wallSize),
        left: 0,
        right: this.canvas.width
    }
}
```

MOVING THE PADDLE

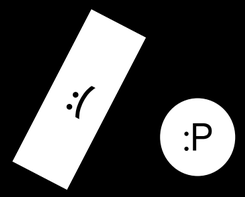
WHY?
This abstraction allows you to change the way you control the paddle, keys or mouse, and apply it to other paddles if you wanted to make this 2 player
Learn more in INF1B

- You have an empty class PaddleController
  - This class will have a constructor passing and storing a paddle object.
- Add a mousemove event listener to the constructor.
  - Inside set the paddle y coordinate to the mouse y position.
  - Make sure to catch the scenario where the mouse is out of bounds (preventing the paddle from going off-canvas), the court bounds upper and lower may help here.
- Instantiate the PaddleController in the court constructor and test out your moving paddle!

Hint: The PaddleController constructor should have this outline

```
this.paddle = paddle
this.canvas = this.paddle.court.canvas
this.canvas.addEventListener('mousemove', (e) => {
// Your logic here
 }
```

COLLISION DETECTION



- Currently the ball will go through the paddle

- We have provided a Rectangle class, the key method is overlaps.

- Create a collisionBox getter (remember the keyword is *get*) in the Paddle class which returns a rectangle object giving the position, width and height of the paddle.
- Do the same for the Ball, be careful with the parameters as the x, y are in the centre of the ball.
- Now add if-else cases in the ball update method, checking for overlaps and reversing the x component of velocity.
- You may also wish to reposition the x coordinate of the ball to the paddle face to prevent it from drifting along the paddle
    - e.g. for the left paddle it would be:
        - *this.x = this.court.leftPaddle.collisionBox.right + this.radius*

SCOREBOARD

- In order to display and keep score, we have provided an empty Scoreboard class.
  - It will have variables for the score of each player and a round counter.
  - It will also need a draw method
    - Use the ctx.fillText(text, x, y) method to write the scores.
      - You will use this method 3 times, once for each score and once for the round number, see format above.
      - Use ctx.font to change the font (it is provided as SETTINGS.smallFont)
- Then, instantiate a scoreboard in the court and draw the scoreboard in the court.draw method.

- In Court, create a method scorePoint(player) that adds a point to the score of the passed player number, then respawns the ball and increases round number.

- Now, in ball.update, add if-else statements to catch when the ball goes out of left or right bounds of the court, use this to score the relevant points.

BALL SERVE

- The ball is fired in the same direction every time, not very fair or hard to predict.

- Create a function in Court called spawnBall() that will randomly assign a velocity and reset the position of the ball.
  - Math.random() is a function that will return a number between 0 and 1, so if you want 50% psuedo-randomness, you could use this where Math.random() > 0.5
- Also, create a reset() method in Paddle, resetting the position.
- Now we can create a Court method, e.g. startMatch(), that resets the ball, the paddles, and the scoreboard.

ENDGAME

- Your game should be somewhat playable (even if the AI doesn't play yet).
- However, there is no endgame scenario, so it goes on forever.

- In Scoreboard create a getter called winner that returns the player number if a player has reached the win score, otherwise return 0.
  - Hint: Use the *winScore* in SETTINGS.
- Then, in order to stop the game, create a property of Court called isMatchRunning and set it to false.
- At the top of court.update(dT), if isMatchRunning is false, then return nothing (i.e. return by itself), this breaks out of the update loop.

- Finally, edit scorePoint to check for a winner (use the getter), if there is a winner then set isMatchRunning to false. Otherwise, increase round and spawn ball as before.

GAME START

- You may have noticed your game now doesn’t play :(
- We need to trigger setting isMatchRunning to true.
- In the Game constructor, create a Rectangle object, positioning it in the middle of the canvas, call this startButton.
- Now, add an event listener to the canvas and check if the start button contains the mouse click (x, y).
- If it does, start the match from court!
    - In court.startMatch() set isMatchRunning to true

NEXT TIME...

You should now have a working (one-sided) game.

Next time we will cover:

- Implementing the AI
- Design tweaks to make for a better experience.