

Prompt 1

In p5.js, Implement a game that is a bit like Pong. The player uses the keyboard to move a bat up and down to try to hit a ball. If the player hits the ball, the bat grows by 5 pixels. If the player misses the ball, the bat shrinks by 2 pixels and the ball's speed increases by one pixel.

Gemini response 1

To make the ball appear from the right side of the screen, you just need to change its starting horizontal position. The ball's horizontal position is determined by the `ballX` variable.

In your code, find the `resetBall()` function and change the line that sets the initial `ballX` value.

Original Code

JavaScript

```
None
function resetBall() {
  ballX = width / 2; // Ball starts in the center
  ballY = height / 2;
  // ... rest of the code
}
```

Modified Code

Change `ballX = width / 2;` to `ballX = width;`

JavaScript

```
None
function resetBall() {
  ballX = width; // Ball now starts from the far right edge
  ballY = height / 2;
  ballXSpeed = 4;
  ballYSpeed = 4;

  // Randomize initial direction of the ball
  if (random(1) > 0.5) {
```

```
    ballXSpeed *= -1;
}
if (random(1) > 0.5) {
    ballYSpeed *= -1;
}
}
```

With this single change, the ball will now always reset to the right edge of the canvas, moving towards the player's paddle.

Prompt 2

The ball seems to appear from the centre of the screen. Make it appear from the right instead.

Response 2

That's an excellent observation. The ball isn't disappearing; it's being reset to the center of the screen every time it hits the bat. To make it bounce back realistically, you need to remove the `resetBall()` call from the `hitBall()` function.

Here's how to modify your code to achieve the desired bouncing effect:

Before

This is the original `hitBall()` function. Notice the `resetBall()` call at the end, which is causing the ball to teleport back to the center of the screen.

JavaScript

```
None
function hitBall() {
    // Reverse ball's horizontal direction and speed it up slightly
    ballXSpeed *= -1.05;

    // Grow the paddle
    paddleHeight += 5;
```

```
// Reset ball position (This is the line to remove)
resetBall();
}
```

After

By simply removing the `resetBall()` line, the ball will now reverse its direction when it collides with the paddle, creating a proper bounce. The other logic, which increases the speed and grows the paddle, will remain intact.

JavaScript

```
None
function hitBall() {
    // Reverse ball's horizontal direction and speed it up slightly
    ballXSpeed *= -1.05;

    // Grow the paddle
    paddleHeight += 5;
}
```

Make this small change, and the ball will bounce off the paddle just like in a classic game of Pong!

Prompt 3

The ball disappears when it hits the bat. Make it bounce back instead.

Response 3

That's an excellent observation. The ball isn't disappearing; it's being reset to the center of the screen every time it hits the bat. To make it bounce back realistically, you need to remove the `resetBall()` call from the `hitBall()` function.

Here's how to modify your code to achieve the desired bouncing effect:

Before

This is the original `hitBall()` function. Notice the `resetBall()` call at the end, which is causing the

ball to teleport back to the center of the screen.

JavaScript

```
None

function hitBall() {
    // Reverse ball's horizontal direction and speed it up slightly
    ballXSpeed *= -1.05;

    // Grow the paddle
    paddleHeight += 5;

    // Reset ball position (This is the line to remove)
    resetBall();
}
```

After

By simply removing the `resetBall()` line, the ball will now reverse its direction when it collides with the paddle, creating a proper bounce. The other logic, which increases the speed and grows the paddle, will remain intact.

JavaScript

```
None

function hitBall() {
    // Reverse ball's horizontal direction and speed it up slightly
    ballXSpeed *= -1.05;

    // Grow the paddle
    paddleHeight += 5;
}
```

Make this small change, and the ball will bounce off the paddle just like in a classic game of Pong!

Prompt 4

You've found an important detail in the original collision logic! You're right, the current code only checks if the ball hits the right side of the paddle. To treat the bat as a solid object, the collision detection needs to be more robust, checking for the ball's position relative to all four sides of the bat.

To fix this, you need to adjust the collision check in your `draw()` function. Here's the updated

collision logic that treats the paddle as a solid rectangle.

Revised Collision Logic

The new code uses a more standard **rectangle collision detection** method. It checks if the ball's X and Y coordinates overlap with the paddle's X and Y coordinates simultaneously.

JavaScript

```
None

// Check for collision with the paddle
if (
    ballX + ballDiameter / 2 >= paddleX && // Check if the ball's
    right edge is past the paddle's left edge
    ballX - ballDiameter / 2 <= paddleX + paddleWidth && // Check
    if the ball's left edge is past the paddle's right edge
    ballY + ballDiameter / 2 >= paddleY && // Check if the ball's
    bottom edge is past the paddle's top edge
    ballY - ballDiameter / 2 <= paddleY + paddleHeight // Check
    if the ball's top edge is past the paddle's bottom edge
) {
    hitBall();
}
```

By replacing the old `if` statement with this one, the game will now correctly detect a collision regardless of which side of the paddle the ball hits. The ball will bounce back appropriately, and the game mechanics will work as intended.