The ball is bouncing off the walls freely and you can watch it indefinitely, but currently there's no interactivity. It's not a game if you cannot control it! So let's add some user interaction: a controllable paddle.

**Defining a paddle to hit the ball**

So, we need a paddle to hit the ball — let's define a few variables for that. Add the following variables near the top of your code, beside your other variables:

var paddleHeight = 10;

var paddleWidth = 75;

var paddleX = (canvas.width-paddleWidth)/2;

Here we're defining the height and width of the paddle, and its starting point on the x axis, for use in calculations further on down the code. Let's create a function that will draw the paddle on the screen. Add the following just below your drawBall() function:

function drawPaddle() {

ctx.beginPath();

ctx.rect(paddleX, canvas.height-paddleHeight, paddleWidth, paddleHeight);

ctx.fillStyle = "#0095DD";

ctx.fill();

ctx.closePath();

}

**Allowing the user to control the paddle**

We can draw the paddle wherever we want, but it should respond to the user's actions — it is time to implement some keyboard controls. We will need:

* Two variables for storing information on whether the left or right control button is pressed.
* Two event listeners for keydown and keyup events — we want to run some code to handle the paddle movement when the buttons are pressed.
* Two functions handling the keydown and keyup events  the code that will be run when the buttons are pressed.
* The ability to move the paddle left and right

Pressed buttons can be defined and initialized with boolean variables, like so. Add these lines somewhere near the rest of your variables:

var rightPressed = false;

var leftPressed = false;

The default value for both is false because at the beginning the control buttons are not pressed. To listen for key presses, we will set up two event listeners. Add the following lines just above the setInterval() line at the bottom of your JavaScript:

document.addEventListener("keydown", keyDownHandler, false);

document.addEventListener("keyup", keyUpHandler, false);

When the keydown event is fired on any of the keys on your keyboard (when they are pressed), the keyDownHandler() function will be executed. The same pattern is true for the second listener: keyup events will fire the keyUpHandler() function (when the keys stop being pressed). Add these to your code now, below the addEventListener() lines:

function keyDownHandler(e) {

if(e.keyCode == 39) {

rightPressed = true;

}

else if(e.keyCode == 37) {

leftPressed = true;

}

}

function keyUpHandler(e) {

if(e.keyCode == 39) {

rightPressed = false;

}

else if(e.keyCode == 37) {

leftPressed = false;

}

}

When we press a key down, this information is stored in a variable. The relevant variable in each case is set to true. When the key is released, the variable is set back to false.

Both functions take an event as a parameter, represented by the e variable. From that you can get useful information: the keyCode holds the information about the key that was pressed. For example key code 37 is the left cursor key and 39 is the right cursor. If the left cursor is pressed, then the leftPressed variable is set to true, and when it is released the leftPressed variable is set to false. The same pattern follows with the right cursor and the rightPressed variable.

The paddle moving logic

We now have the variables for storing the info about the pressed keys, event listeners and relevant functions set up. Now we'll get onto the actual code to use all that and move the paddle on the screen. Inside the draw() function, we will check if the left or right cursor keys are pressed when each frame is rendered. Our code might look like this:

if(rightPressed) {

paddleX += 7;

}

else if(leftPressed) {

paddleX -= 7;

}

If the left cursor is pressed, the paddle will move 7 pixels to the left, and if the right cursor is pressed, the paddle will move 7 pixels to the right. This currently works ok, but the paddle disappears off the edge of the canvas if we hold either key for too long. We could improve that and move the paddle only within the boundaries of the Canvas by changing the code as follows:

if(rightPressed && paddleX < canvas.width-paddleWidth) {

paddleX += 7;

}

else if(leftPressed && paddleX > 0) {

paddleX -= 7;

}

The paddleX position we're using will move between 0 on the left side of the Canvas andcanvas.width-paddleWidth on the right hand side, which will work exactly as we want it.

Add the above code block into the draw() function at the bottom, just above the closing curly brace.

The only thing left to do now is call the drawPaddle() function from within the draw() function, to actually print it on the screen. Add the following line inside your draw() function, just below the line that calls drawBall():

drawPaddle();

**Compare your code**

Here's the working code for you to compare yours against:

**Exercise**: make the paddle move faster or slower, or change its size.

**Next steps**

Now we have something resembling a game; the only trouble now is that you can just continue hitting ball with the paddle forever. This will all change in the fifth chapter, [Game over](https://developer.mozilla.org/en-US/docs/Games/Workflows/Breakout_game_from_scratch/Game_over), when we start to add in an endgame state for our game