Steps to creating the game with the bouncing ball inside a rectangle:

**1. create the canvas and draw on it;**

-create the index.html;

- body contains canvas and script elements;

- in style.css create some styling for the canvas;

- in .js grab the reference to the canvas by using its ID;

- create the drawingContext var to store the 2D rendering context

- all the instructions are between the [beginPath()](https://developer.mozilla.org/en-US/docs/Web/API/CanvasRenderingContext2D/beginPath" \o "beginPath()) and [closePath()](https://developer.mozilla.org/en-US/docs/Web/API/CanvasRenderingContext2D/closePath" \o "closePath()) methods.

**2. move the ball;**

- define a drawing loop – which will run infinitely/setInterval() – function draw;

- define the starting point (var x and y);

- use the arc() method to move the ball;

- make the ball appear it’s moving, use small values for x and y – create dx and dy;

- it appears the ball is leaving a trail behind it;

- use the method clearRect() to clear the canvas before each frame

**3. bounce the ball off the walls;**

- define the variable ballRadius to make calculations easier;

- update the line that draws the ball inside the drawBall() function;

- check, on every frame, whether the ball is touching the top edge of the Canvas — if yes, we'll reverse the ball movement so it will start to move in the opposite direction and stay within the visible boundaries;

if(y + dy < 0) {

dy = -dy;

}

- If the y value of the ball position is lower than zero, change the direction of the movement on the y axis by setting it equal to itself, reversed.

if(y + dy > canvas.height) {

dy = -dy;

}

- merge those two statements into one to save on code verbosity / needs more words, or longer words, than is necessary to adequately express the intent of the code:

if(y + dy > canvas.height || y + dy < 0) {

dy = -dy;

}

-repeat the statements for x instead of y;

!!!!- ball disappearing into the wall;

- because we’re calculating collision point of the wall and the center of the ball instead of its circumference.

- update code to this:

y + dy < 0

- and this:

x + dx > canvas.width-ballRadius

**4. build the bricks;**

- define the bricks variables;

- create a loop to go through rows and columns and create the bricks; it will create objects for each x and y position to paint each brick on the screen;

- create a function drawBricks() to loop through all the bricks in the array and draw them on screen; it loops through row and columns to set the x and y position for each brick;

!!!!!-all bricks appear in the same place – coordinates set to (0, 0);

- create brickX and brickY variables to calculate position for each brick for each loop iteration;

- each brickX = brickWidth + brickPadding, multiplied by the column number, c, plus the brickOffsetLeft; the logic for the brickY is identical - it uses the values for row number, r, brickHeight, and brickOffsetTop;

- call the function drawBricks() inside the draw() function;

**5. collision with the bricks;**

- create collision detection function – it will loop through all the bricks and compare brick’s position to the ball’s coordinates as each frame is drawn;

- define the b variable for storing the brick object in every loop of the collision detection;

- if the center of the ball is inside the coordinates of one of our bricks, we'll change the direction of the ball;

* x position of the ball is greater than the x position of the brick.
* x position of the ball is less than the x position of the brick plus its width.
* y position of the ball is greater than the y position of the brick.
* y position of the ball is less than the y position of the brick plus its height.

- make the bricks disappear after they are hit;

- add an extra parameter to indicate whether we want to paint each brick on the screen or not;

- where we initialized the bricks, we add status property to each brick object;

- check the value of each brick’s status in the drawBricks() function before drawing it;

- if status is 1, then draw it, but if it's 0, then it was hit by the ball and we don't want it on the screen anymore;

- track and update the status in the collision detection function;

- if the brick is active (its status is 1) we will check whether the collision happens; if a collision happens we'll set the status of the brick to 0;

- call the collisionDetection() function in the draw() function;

**6. track score & win**

- define a variable to record the score;

- a function to create and update the score display;

- add line to the collisionDetection() function to increment the value of the score;

- call the drawscore() function, inside the draw() function;

- display a winning message when all bricks have been destroyed, inside the collisionDetection() function;

- document.location.reload() function reloads the page and starts the game again once the alert button is clicked.

**7.** **Paddle and keyboard controls**

- define variables for paddle;

-set the height and width of the paddle;

- how to control the paddle:

-variables to store the data;

- event listeners – for when buttons are pressed;

- functions handling events when the buttons are pressed;

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.

-move the paddle left and right;

- paddle moving logic (only within the canvas surface);

**8. Game over**

- instead of allowing the ball to hit all four walls, hitting the bottom one will end the game;

- add if else la if-ul cu winning;

- collision detection between ball and paddle;

**9. add lives to player**

-