Paddle Game Questions - Jameson Loewen

1. Provide a written response to your video that:
   - identifies the programming language;
   - identifies the purpose of your program; and
   - explains what the video illustrates.

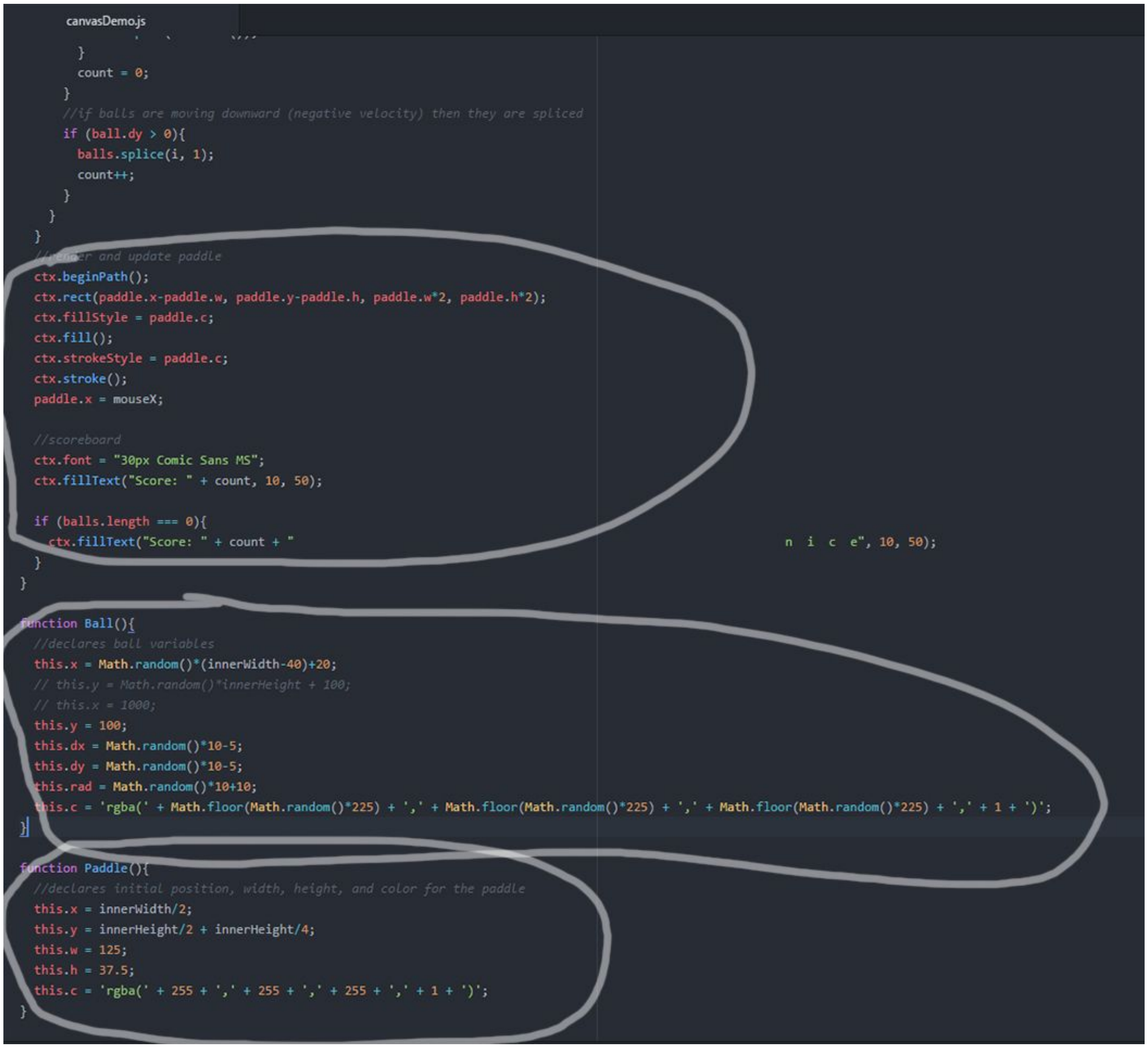
   The language used was javascript. The purpose was to create a paddle game that increases in difficulty over time. The goal is to "catch" all the balls and if any hit the bottom of the paddle the game resets with more balls, making it harder and harder to win. The video illustrates the way the balls interact with the paddle and how the game starts and ends.

2. Describe the incremental and iterative development process of your program, focusing on two distinct points in that process. Describe the difficulties and/or opportunities you encountered and how they were resolved or incorporated. In your description clearly, indicate whether the development described was collaborative or independent. At least one of these points must refer to independent program development.

   An iterative and incremental process I used when developing the program was when I took the code I had designed last year and re-analyzed it for this project. I took a base design (increment 1) and made a new program that used some of the code from the previous project. This new increment underwent the same process as the previous project, but this time I was able to build off of the knowledge of a previous version and use it to better my program. During the iteration stage, I had to face the problem that was it was written by me with one year less experience. My code wasn't clean and there were really no comments. This was solved by cleaning up and commenting while going through every line of code and trying to make sense of it. Another problem I faced during this process was trying to get my old variables to align with the new ones. I constantly over-wrote old variables which lead to several errors. As I went through the code commenting, I also noted what the variables were called so it was less of an issue. This project was a collaborative effort with my partner and I working on the same pieces of code at a time, planning out how to fix errors and make the program seamless.

3. Capture and paste the program code segment that implements an algorithm (marked with an oval) that is fundamental for your program to achieve its intended purpose. Your code segment must include an algorithm that integrates other algorithms and integrates mathematical and/or logical concepts. Describe how each algorithm within your selected

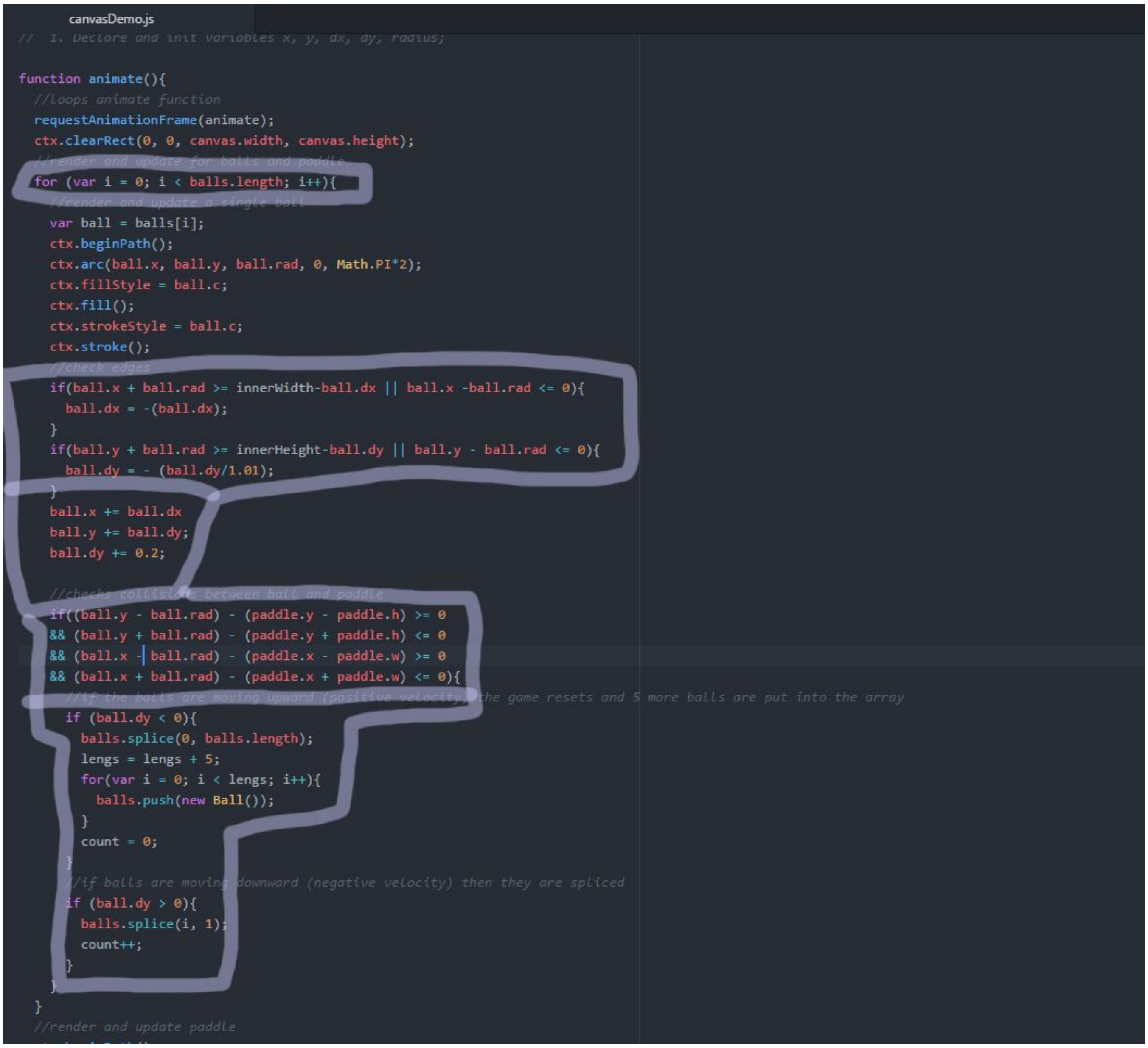
algorithm functions independently, as well as in combination with others, to form a new algorithm that helps to achieve the intended purpose of the program. *(Approximately 200 words)*


canvasDemo.js

```
      }
      count = 0;
    }
    //if balls are moving downward (negative velocity) then they are spliced
    if (ball.dy > 0){
      balls.splice(i, 1);
      count++;
    }
  }
}
//render and update paddle
ctx.beginPath();
ctx.rect(paddle.x-paddle.w, paddle.y-paddle.h, paddle.w*2, paddle.h*2);
ctx.fillStyle = paddle.c;
ctx.fill();
ctx.strokeStyle = paddle.c;
ctx.stroke();
paddle.x = mouseX;

//scoreboard
ctx.font = "30px Comic Sans MS";
ctx.fillText("Score: " + count, 10, 50);

if (balls.length === 0){
  ctx.fillText("Score: " + count + "                                    n  i  c  e", 10, 50);
}
}

function Ball(){
  //declares ball variables
  this.x = Math.random()*(innerWidth-40)+20;
  // this.y = Math.random()*innerHeight + 100;
  // this.x = 1000;
  this.y = 100;
  this.dx = Math.random()*10-5;
  this.dy = Math.random()*10-5;
  this.rad = Math.random()*10+10;
  this.c = 'rgba(' + Math.floor(Math.random()*225) + ',' + Math.floor(Math.random()*225) + ',' + Math.floor(Math.random()*225) + ',' + 1 + ')';
}

function Paddle(){
  //declares initial position, width, height, and color for the paddle
  this.x = innerWidth/2;
  this.y = innerHeight/2 + innerHeight/4;
  this.w = 125;
  this.h = 37.5;
  this.c = 'rgba(' + 255 + ',' + 255 + ',' + 255 + ',' + 1 + ')';
}
```

The “Paddle” and “Ball” functions take in no variables from other algorithms, only making their own inside of their body. However, as shown in the uppermost algorithm, both are used in the rest of the program to render and update other objects in the program. This algorithm uses other algorithms while it is running. For example, the top algorithm uses .stroke() and .fill(), two algorithms provided by the javascript language itself. We do not need to know what is inside fill or stroke, but rather their function of coloring or changing the appearance of an object. The top algorithm also uses the context class, or ctx, meaning all of the function calls following it are other abstractions pulled from the ctx class, which we again, as coders, do not need to see to know they do. Finally, the top algorithm pulls variables from the bottom two algorithms so that they can be rendered properly.

4. Capture and paste the program code segment that contains an abstraction you developed (marked with a rectangle in section 3 below). Your abstraction should integrate mathematical and logical concepts. Explain how your abstraction helped manage the complexity of your program.

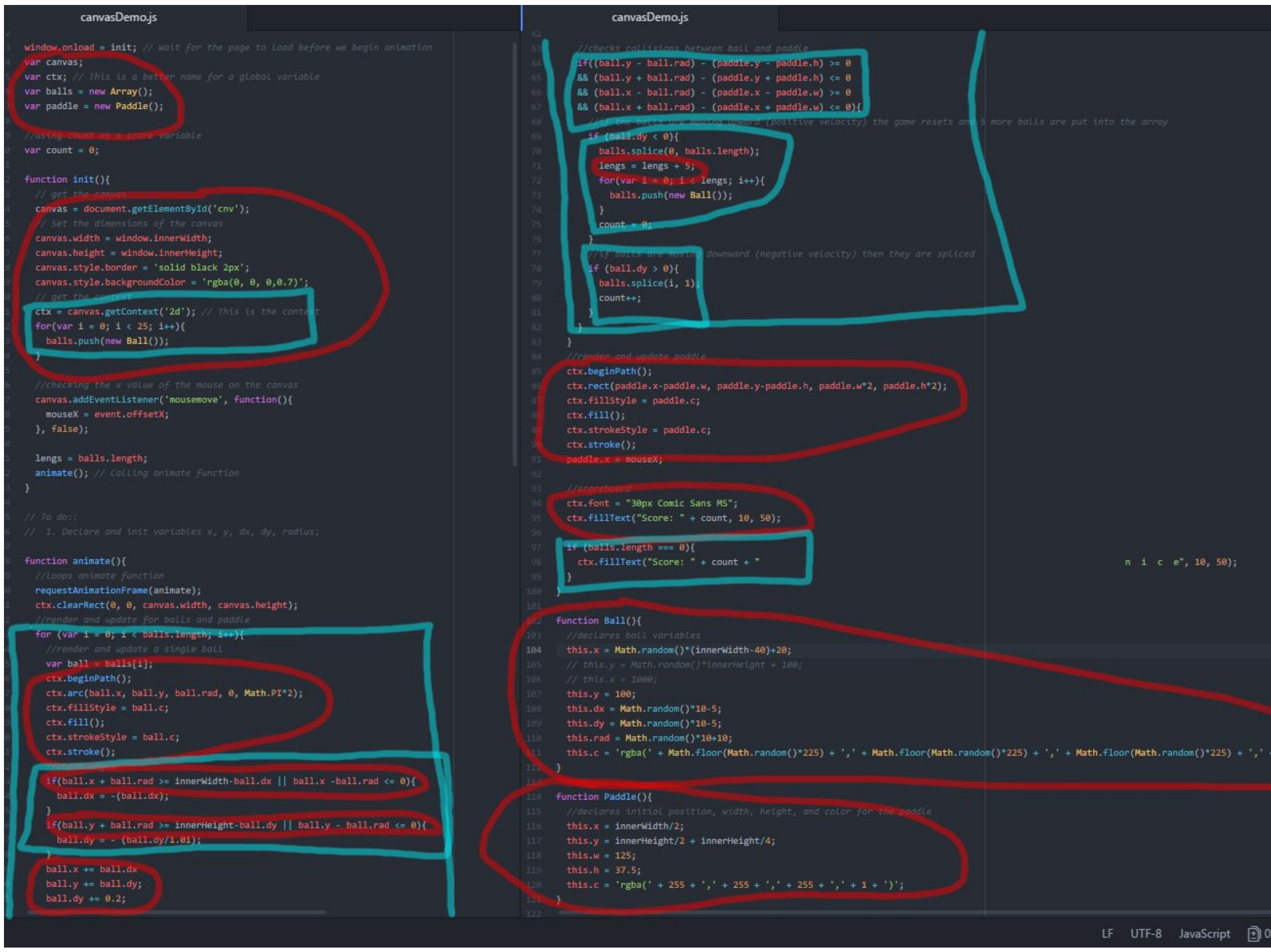


canvasDemo.js

```
//  1. Declare and init variables x, y, dx, dy, radius;

function animate(){
  //loops animate function
  requestAnimationFrame(animate);
  ctx.clearRect(0, 0, canvas.width, canvas.height);
  //render and update for balls and paddle
  for (var i = 0; i < balls.length; i++){
    //render and update a single ball
    var ball = balls[i];
    ctx.beginPath();
    ctx.arc(ball.x, ball.y, ball.rad, 0, Math.PI*2);
    ctx.fillStyle = ball.c;
    ctx.fill();
    ctx.strokeStyle = ball.c;
    ctx.stroke();
    //check edges
    if(ball.x + ball.rad >= innerWidth-ball.dx || ball.x -ball.rad <= 0){
      ball.dx = -(ball.dx);
    }
    if(ball.y + ball.rad >= innerHeight-ball.dy || ball.y - ball.rad <= 0){
      ball.dy = - (ball.dy/1.01);
    }
    ball.x += ball.dx
    ball.y += ball.dy;
    ball.dy += 0.2;

    //checks collisions between ball and paddle
    if((ball.y - ball.rad) - (paddle.y - paddle.h) >= 0
    && (ball.y + ball.rad) - (paddle.y + paddle.h) <= 0
    && (ball.x - ball.rad) - (paddle.x - paddle.w) >= 0
    && (ball.x + ball.rad) - (paddle.x + paddle.w) <= 0){
      //if the balls are moving upward (positive velocity) the game resets and 5 more balls are put into the array
      if (ball.dy < 0){
        balls.splice(0, balls.length);
        lengs = lengs + 5;
        for(var i = 0; i < lengs; i++){
          balls.push(new Ball());
        }
        count = 0;
      }
      //if balls are moving downward (negative velocity) then they are spliced
      if (ball.dy > 0){
        balls.splice(i, 1);
        count++;
      }
    }
  }
  //render and update paddle
```

We have the ability, using abstraction, to create multiple “Ball” objects that all share the same values. This allows us to check all of the balls positions in relation to both the walls of the screen and the paddle with only a few lines of code. Abstraction allowed me to add or take away balls with the change of one number rather than having to worry about adding or subtracting whole chunks of code that might have slowed down the program.

5. Capture and paste your entire program code in this section.
    - Mark with an oval the segment of program code that implements the algorithm and integrates mathematical and /or logical concepts.
    - Mark with a rectangle the segment of program code that represents an abstraction you developed.
    - Include comments or citations for program code that has been written by someone else.



canvasDemo.js

```
window.onload = init; // Wait for the page to load before we begin animation
var canvas;
var ctx; // This is a better name for a global variable
var balls = new Array();
var paddle = new Paddle();

//using count as a score variable
var count = 0;

function init(){
  // get the canvas
  canvas = document.getElementById('cnv');
  // Set the dimensions of the canvas
  canvas.width = window.innerWidth;
  canvas.height = window.innerHeight;
  canvas.style.border = 'solid black 2px';
  canvas.style.backgroundColor = 'rgba(0, 0, 0,0.7)';
  // get the context
  ctx = canvas.getContext('2d'); // This is the context
  for(var i = 0; i < 25; i++){
    balls.push(new Ball());
  }

  //checking the x value of the mouse on the canvas
  canvas.addEventListener('mousemove', function(){
    mouseX = event.offsetX;
  }, false);

  lengs = balls.length;
  animate(); // Calling animate function
}

// To do::
//  1. Declare and init variables x, y, dx, dy, radius;

function animate(){
  //loops animate function
  requestAnimationFrame(animate);
  ctx.clearRect(0, 0, canvas.width, canvas.height);
  //render and update for balls and paddle
  for (var i = 0; i < balls.length; i++){
    //render and update a single ball
    var ball = balls[i];
    ctx.beginPath();
    ctx.arc(ball.x, ball.y, ball.rad, 0, Math.PI*2);
    ctx.fillStyle = ball.c;
    ctx.fill();
    ctx.strokeStyle = ball.c;
    ctx.stroke();
    //check [illegible]
    if(ball.x + ball.rad >= innerWidth-ball.dx || ball.x -ball.rad <= 0){
      ball.dx = -(ball.dx);
    }
    if(ball.y + ball.rad >= innerHeight-ball.dy || ball.y - ball.rad <= 0){
      ball.dy = - (ball.dy/1.01);
    }
    ball.x += ball.dx
    ball.y += ball.dy;
    ball.dy += 0.2;
```

canvasDemo.js

```

    //checks collisions between ball and paddle
    if((ball.y - ball.rad) - (paddle.y - paddle.h) >= 0
    && (ball.y + ball.rad) - (paddle.y + paddle.h) <= 0
    && (ball.x - ball.rad) - (paddle.x - paddle.w) >= 0
    && (ball.x + ball.rad) - (paddle.x + paddle.w) <= 0){
      //if the balls are moving upward (positive velocity) the game resets and 5 more balls are put into the array
      if (ball.dy < 0){
        balls.splice(0, balls.length);
        lengs = lengs + 5;
        for(var i = 0; i < lengs; i++){
          balls.push(new Ball());
        }
        count = 0;
      }
      //if balls are moving downward (negative velocity) then they are spliced
      if (ball.dy > 0){
        balls.splice(i, 1);
        count++;
      }
    }
  }
  //render and update paddle
  ctx.beginPath();
  ctx.rect(paddle.x-paddle.w, paddle.y-paddle.h, paddle.w*2, paddle.h*2);
  ctx.fillStyle = paddle.c;
  ctx.fill();
  ctx.strokeStyle = paddle.c;
  ctx.stroke();
  paddle.x = mouseX;

  //scoreboard
  ctx.font = "30px Comic Sans MS";
  ctx.fillText("Score: " + count, 10, 50);

  if (balls.length === 0){
    ctx.fillText("Score: " + count + "                                            n  i  c  e", 10, 50);
  }
}

function Ball(){
  //declares ball variables
  this.x = Math.random()*(innerWidth-40)+20;
  // this.y = Math.random()*innerHeight + 100;
  // this.x = 1000;
  this.y = 100;
  this.dx = Math.random()*10-5;
  this.dy = Math.random()*10-5;
  this.rad = Math.random()*10+10;
  this.c = 'rgba(' + Math.floor(Math.random()*225) + ',' + Math.floor(Math.random()*225) + ',' + Math.floor(Math.random()*225) + ','
}

function Paddle(){
  //declares initial position, width, height, and color for the paddle
  this.x = innerWidth/2;
  this.y = innerHeight/2 + innerHeight/4;
  this.w = 125;
  this.h = 37.5;
  this.c = 'rgba(' + 255 + ',' + 255 + ',' + 255 + ',' + 1 + ')';
}

```

LF UTF-8 JavaScript