**Week 1 Directions**

For the next three weeks, you will be creating a breakout game. Breakout is an arcade game in which a layer of bricks lines the top third of the screen. The goal is to destroy them all by repeatedly bouncing a ball off a paddle into them.

Here is a free online version of the game to get a feel of what the finished product should look like: [https://elgoog.im/breakout/Links to an external site.](https://elgoog.im/breakout/). This is based on when Google had an easter egg where the browser automatically pulled up this game when you searched "Atari Breakout" images.

This week, you will be modifying the existing code.

Module 13: Breakout Game Part 1

First, under your COSC1350 folder, create a new folder named m13, and save the attached files. Open the m13 folder in Visual Studio Code and open the three files. Look over the existing code, and familiarize yourself with what is happening. If you need help understanding something, reach out via Discord, office hours, or direct email. This is a big project, and each student is expected to spend some time talking to the instructor. These instructions alone are not enough to successfully code this game; it will take creativity, intuition, research, and, above all, questions.

**Make the Webpage Look More Appealing**

Before getting too far into the hard stuff, it would be nice if what you're working on isn't a complete eyesore.

1. Label the breakout game with a header. Perhaps something like an h1 tag with the content 'Breakout Game'.

2. Center the header and canvas elements on the page.

3. Style the rest of the page to your liking.

At a minimum, add a colored background and a border around the breakout game. A cool font would be nice

**Get the Ball Moving**

4. Uncomment the lines of code in the draw function to make the ball's position change.

5. Bounce the ball off the wall by comparing xPos and the boundaries of the game, and negating xMoveDist.

hint: if(xPos > canvas.width - ballRadius) xMoveDist = -xMoveDist;

6. Do the same for the height of the canvas element and the yMoveDist.

After this step, the ball should be infinitely bouncing off the walls

**Add the Paddle**

7. Create a new function that draws a 100x15px rectangle at the bottom center of the canvas. Feel free to use ctx.fillStyle to color your paddle and ball differently!

8. Add keydown and keyup event listeners to the DOM, and create two new listener functions for each event. Create two boolean variables called moveLeft and moveRight that are true when their respective key is down, and false when it is up.

9. Change the paddle's drawn x position to be a variable instead of a hard-coded value. Name this variable xPaddle, and in the draw function check if moveLeft or moveRight is true, and adjust the xPaddle by 3 in the appropriate direction.

Remember that the draw function is constantly being called, which is how simple if statements are enough to check for changes in the game state. You should now have a paddle that can move left and right. Make sure you don't allow for the paddle to move off the screen.

Before submitting the assignment, double-check that you have updated the name and date at the top of the source code files, and have replaced my comments with your own. When finished, upload your m13 folder to your GitHub repository. Once you are ready for me to grade, submit your URL here.

**Attachments:**

* [breakout.html](#breakouthtml)
* [breakout.css](#breakoutcss)
* [breakout.js](#breakoutjs)

breakout.html

<!DOCTYPE html>

<!--

Student Name:

File Name: breakout.html

Date:

-->

<html lang="en-US">

<head>

<meta charset="utf-8" />

<title>Guessing Game</title>

<link href="breakout.css" rel="stylesheet" type="text/css">

</head>

<body>

<canvas id="myCanvas" width="600" height="400"></canvas>

<script src="breakout.js"></script>

</body>

</html>

breakout.css

/\*

Author:

Date:

File Name: breakout.css

\*/

/\* CSS Reset \*/

body, header, nav, main, footer, img, h1, h3, ul {

margin: 0;

padding: 0;

border: 0;

}

/\*

Remember, since the canvas element has the id=myCanvas, selecting it

for styling can be as easy as:

#myCanvas {

border-style: solid;

}

Of course, since there will only be one canvas element on the page, you could also just use

canvas {

border-style: solid;

}

Tip: I like to style my homework assignments to match my text editor's theme.

\*/

canvas {

border-style: solid;

}

breakout.js

/\* ..:: B R E A K O U T G A M E ::..

\*

\* breakout.js

\* Author:

\* Date:

\* Project for COSC 1350

\*

\*/

// get the canvas element from the DOM.

const canvas = document.getElementById("myCanvas");

/\* create a "2d rendering context".

\* I suggest looking up and reading about the canvas.getContext function.

\* You don't have to understand everything about canvas rendering contexts,

\* but it help you get to know what the ctx object can and can't draw.

\*/

const ctx = canvas.getContext("2d");

//drawing a ball requires the x position, y position, and radius

let ballRadius = 15, xPos = canvas.width / 2, yPos = canvas.height / 2;

//xy move distance. These values are used to move the ball around.

let xMoveDist = 3, yMoveDist = 3;

//function that draws the ball on the canvas

ballRender=()=>{

ctx.beginPath();

//arc creates circular arc starting at 0, ending at 2pi (360 degrees)

ctx.arc(xPos, yPos, ballRadius, 0, Math.PI \* 2);

//fill in the circular path with default color

ctx.fill();

ctx.closePath();

}

/\*

\* draw() can be thought of as our main function.

\* We execute draw every few milliseconds to give our

\* canvas the appearance of being animated. Notice how in the draw function

\* the first thing done is ctx.clearRect(), which clears the whole canvas

\* before drawing the next frame of animation.

\*

\* Right now, it only calls ballRender() over and over again.

\* Changing the xPos and yPos will cause the ball to be drawn somewhere else

\* next time the function is called.

\*/

draw=()=> {

ctx.clearRect(0, 0, canvas.width, canvas.height);

ballRender();

//uncomment when you're ready to send the ball flying!

// xPos += xMoveDist;

// yPos += yMoveDist;

};

/\*

\* setInterval(func, delay)

\* this built-in global JavaScript function executes 'func' function every

\* 'delay' milliseconds, and returns an interval ID. We won't really use intervalID

\* so don't worry to much about that for now.

\*

\* Try playing around with the refreshRate value.

\*/

const refreshRate = 40;

const intervalID = setInterval(draw, refreshRate);

**Week 2 Directions**

For the next three weeks, you will be creating a breakout game. Breakout is an arcade game in which a layer of bricks lines the top third of the screen. The goal is to destroy them all by repeatedly bouncing a ball off a paddle into them.

Here is a free online version of the game to get a feel of what the finished product should look like: [https://elgoog.im/breakout/Links to an external site.](https://elgoog.im/breakout/). This is based on when Google had an easter egg where the browser automatically pulled up this game when you searched "Atari Breakout" images.

Module 13: Breakout Game Part 2

Now it's time to line the top of the game with bricks. It is easiest to manage the bricks by defining them as objects with an x position, y position, and hit status. Then, you can use an array to hold your brick objects and iterate through that array to draw them on the canvas element. After doing that, we will be ready to add collision detection to the game.

**Define the Bricks**

1. Define values for the bricks. For this version of breakout, we will have 4 rows and 6 columns of 90x25px bricks. Allow for a 40px space between the ceiling of the canvas element and the top of the brick row.

To nicely align the bricks, they should be padded 5px away from the walls, and 10px away from each other. Therefore, your values should be brickRows=4, brickColumns=6, brickWidth=90, brickHeight=25, brickPadding=10, brickTopOffset=40, brickLeftOffset=5.

2. Define the array of brick objects.

This is easiest done with two for loops: one to iterate through the rows, and another nested loop to iterate through the columns. Then, you can use which row or column your brick is at, and accordingly calculate and store the xy position.

3. Draw the bricks on the canvas element.

Simply create a function that iterates through your array of bricks, and draws them as rectangles with their corresponding values that should already be defined. Remember to add this function to the draw function.

4. Add collision detection to the bricks.

To do this, you will need to compare the ball's position with the position of each brick. Again, use a for loop to iterate through the bricks. If the brick hasn't been hit, check if the ball's x position is greater than the brick's x position AND less than the brick's x position plus the brick's width, AND the same for the ball and brick's y positions, then update the brick's hit status. Also, negate the ball's yMoveDist just like if it hit the ceiling of the game.

5. Go back and edit the brick drawing function to check whether the hit status of the brick indicates to draw the brick or not.

Notice that the ball only bounces off of the brick if it hasn't been hit yet. Once the brick is hit, it should be invisible and not cause the ball to bounce off it anymore. The bricks should now be falling victim to the entropy of the ball

6. Add collision detection to the paddle by modifying the bottom wall of the game's logic in the draw function.

Instead of checking if the ball is at the height of the canvas, check if it is at the height of the canvas minus the height of the paddle and if its x coordinates are within the paddle's x coordinates, much like how you did it for the bricks. If the ball does not hit the x coordinates of the paddle, then use clearInterval to stop the calling of the draw function. Add an alert or some text to the DOM to let the user know that it is game over

Before submitting the assignment, double-check that you have everything working. When finished, upload your m13 folder to your GitHub repository. Once you are ready for me to grade, submit your URL here.

**Week 3 Directions**

Your Breakout Game should be almost complete! This week, implement the following:

1. Add the appropriate JavaScript code to keep score.
2. Add a reset button to the DOM that calls a function that resets the game.

In addition, add any finishing touches, such as additional styling, user instructions, and/or code comments to your Breakout Game. Lastly, TEST, TEST, TEST! The program MUST be fully functional, with no errors. Note: no credit will be given for non-functional submissions (code that doesn't work).