The idea of this game is you have thirty (30) seconds to whack the mole as many times as you can and each time you get it, your score count rises. The complete repository can be seen <a href=”https://github.com/thedumebi/whack-a-mole”>here</a>. We have four files, html, css, js and the mole image itself. The html is pretty simple, we have 9 divs where our mole can show up in any of them.

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Whack a mole</title>

    <link rel="stylesheet" href="styles.css">

</head>

<body>

    <h1>Whack a mole</h1>

    <h2>Your Score: <span id="score">0</span></h2>

    <h2 id="time-left">30</h2>

    <button id="start">start</button>

    <div class="grid">

        <div class="square" id="1"></div>

        <div class="square" id="2"></div>

        <div class="square" id="3"></div>

        <div class="square" id="4"></div>

        <div class="square" id="5"></div>

        <div class="square" id="6"></div>

        <div class="square" id="7"></div>

        <div class="square" id="8"></div>

        <div class="square" id="9"></div>

    </div>

</body>

<script src="app.js" charset="UTF-8"></script>

</html>

Our CSS file is also quite simple, we give the divs a width and height so they can contain the image then we also define a mole class that would add the mole image. We are going to add this class to our div when we randomly select a square.

.square {

  width: 200px;

  height: 200px;

  border: solid black;

  border-radius: 100%;

  background-color: white;

}

.grid {

  display: inline-flex;

  flex-flow: row wrap;

  justify-content: center;

  align-content: center;

  width: 618px;

  height: 618px;

}

.mole {

  background-image: url(mole.jpg);

  background-size: cover;

}

The magic happens in our app.js file. To start with, we get all the things required from our html by using <code>querySelector()</code>.

const square = document.querySelectorAll(".square");

const mole = document.querySelectorAll(".mole");

const timeLeft = document.querySelector("#time-left");

const button = document.querySelector("#start");

let score = document.querySelector("#score");

let result = 0;

let currentTime = timeLeft.textContent;

Next, we define a function that helps us pick a random square and when it does, we add the mole class defined in our CSS. Because this is going to be running at a time interval of thirty (30) seconds, we first of all remove the class mole from all our squares then after picking a random square, we add it. After getting the random square, we assign the id (HTML) of it to hitPosition

function randomSquare() {

  square.forEach((className) => {

    className.classList.remove("mole");

  });

  let randomSquare = square[Math.floor(Math.random() \* 9)];

  randomSquare.classList.add("mole");

  // assign the id of randomSquare to hitPosition

  hitPosition = randomSquare.id;

}

Next, we add an event listener to each square so that the result updates if the square you click is the right square (you might want to add an else statement for negative marking if you hit the wrong square 😉).

square.forEach((square) => {

  square.addEventListener("click", () => {

    if (square.id === hitPosition) {

      result = result + 1;

      score.textContent = result;

    }

  });

});

Next, we define a countdown function that would call our <code>randomSquare()</code> function and then subtract our <code>currentTime</code>. When our time reaches zero (0), we alert the player that the game is over and tell them their score.

function countDown() {

  randomSquare();

  currentTime--;

  timeLeft.textContent = currentTime;

  if (currentTime === 0) {

    document.querySelector(".mole").classList.remove("mole");

    alert(`GAME OVER! Your final score is ${result}`);

  }

}

The next thing to do is to define our play function. At the beginning of the game, we haven’t started so we would assign a variable start which would be false then once we start, it becomes true. In our play function, we are going to set our result to zero so anytime we play, it resets back to zero then we are going to set an interval that would call our <code>countdown()</code> function and then if the <code>currentTime</code> is zero (0), we clear the interval and reset some things.

let start = false;

function play() {

    result = 0

    score.textContent = result;

  this.timerId = setInterval(() => {

    countDown();

    if (currentTime === 0) {

      clearInterval(this.timerId);

      timeLeft.textContent = 30;

      currentTime = timeLeft.textContent;

      start = false

    }

  }, 1000);

}

The only thing left is to add a button event listener so that when we click the button, the play function gets called if the game hasn’t started yet.

button.addEventListener("click", () => {

    console.log(start);

  if (!start) {

    play();

  }

  start = true;

});