1. Whack-A-Mole Game

Index.html

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Talha - Whack-a-mole Game</title>
    <link rel="stylesheet" href="style.css" />
    <link rel="icon" href="https://i.ibb.co/M6KTWnf/pic.jpg" />

    <!--
👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇-
->
    <!-- Also uploaded the demo of this code in a gif :
https://c.tenor.com/x8v1oNUOmg4AAAAd/tenor.gif-->
    <!--
👆👆👆👆👆👆👆👆👆👆👆👆👆👆👆👆👆👆👆👆👆👆👆👆👆👆👆👆👆👆👆👆👆👆👆👆👆👆👆👆👆👆👆👆👆👆-
->

    <!--
👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇-
->
    <!-- More html-css-js Games Calculators Games Cards Elements Projects on
https://www.github.com/he-is-talha -->
    <!--
👆👆👆👆👆👆👆👆👆👆👆👆👆👆👆👆👆👆👆👆👆👆👆👆👆👆👆👆👆👆👆👆👆👆👆👆👆👆👆👆👆👆👆👆👆👆-
->
  </head>
  <body>
    <h1>Whack-a-mole!</h1>
    <h2>Score: <span class="score">0</span></h2>
    <button id="start" onClick="startGame()">Start</button>
    <div class="game">
      <div class="hole hole1">
        <div class="mole"></div>
      </div>
      <div class="hole hole2">
        <div class="mole"></div>
      </div>
      <div class="hole hole3">
        <div class="mole"></div>
      </div>
      <div class="hole hole4">
        <div class="mole"></div>
      </div>
      <div class="hole hole5">
        <div class="mole"></div>
      </div>
      <div class="hole hole6">
        <div class="mole"></div>
      </div>
    </div>
    <script src="script.js"></script>
  </body>
</html>
```

style.css

```
@import url("https://fonts.googleapis.com/css?family=Nunito");

*,
*:before,
*:after {
  -webkit-box-sizing: inherit;
  -moz-box-sizing: inherit;
  box-sizing: inherit;
}

html {
  -moz-box-sizing: border-box;
  -webkit-box-sizing: border-box;
  box-sizing: border-box;
  font-size: 10px;
}

body {
  padding: 0;
  margin: 0;
  font-family: "Nunito", sans-serif;
  background: #fff9e2;
  text-align: center;
}

h1 {
  font-size: 4.5rem;
  line-height: 1;
  margin: 2rem 0 0 0;
  color: #ff7660;
}

h2 {
  font-size: 3rem;
  color: #3b1010;
  margin: 2rem;
}

.score {
  background: #ffe5cf;
  padding: 0 3rem;
  line-height: 1;
  -webkit-border-radius: 1rem;
  -moz-border-radius: 1rem;
  border-radius: 1rem;
  color: #3b1010;
}

.game {
  width: 600px;
  height: 400px;
  display: -webkit-box;
  display: -webkit-flex;
  display: flex;
  flex-wrap: wrap;
  margin: 0 auto;
}

.hole {
  flex: 1 0 33.33%;
```

```
  overflow: hidden;
  position: relative;
}

.hole:after {
  display: block;
  background: url("https://s3-us-west-
2.amazonaws.com/s.cdpn.io/1159990/dirt.svg")
    bottom center no-repeat;
  background-size: contain;
  content: "";
  width: 100%;
  height: 70px;
  position: absolute;
  z-index: 2;
  bottom: -30px;
}

.mole {
  background: url("https://s3-us-west-
2.amazonaws.com/s.cdpn.io/1159990/mole.svg")
    bottom center no-repeat;
  background-size: 60%;
  position: absolute;
  top: 100%;
  width: 100%;
  height: 100%;
  transition: all 0.4s;
}

.hole.up .mole {
  top: 0;
}

#start {
  font-family: "Nunito", sans-serif;
  display: inline-block;
  text-decoration: none;
  border: 0;
  background: #3b1010;
  color: #fff;
  font-size: 2rem;
  padding: 1rem 2rem;
  cursor: pointer;
  margin: 1rem;
}

#start:hover {
  opacity: 0.8;
}
```

main.js

```
const holes = document.querySelectorAll(".hole");
const scoreBoard = document.querySelector(".score");
const moles = document.querySelectorAll(".mole");
const button = document.querySelector("#start");
let lastHole;
let timeUp = false;
let score = 0;

function randomTime(min, max) {
  return Math.round(Math.random() * (max - min) + min);
```

```
}

function randomHole(holes) {
  const idx = Math.floor(Math.random() * holes.length);
  const hole = holes[idx];

  if (hole === lastHole) {
    console.log("Same one");
    return randomHole(holes);
  }

  lastHole = hole;
  return hole;
}

function peep() {
  const time = randomTime(200, 1000);
  const hole = randomHole(holes);
  hole.classList.add("up");
  setTimeout(() => {
    hole.classList.remove("up");
    if (!timeUp) peep();
  }, time);
}

function startGame() {
  scoreBoard.textContent = 0;
  timeUp = false;
  score = 0;
  button.style.visibility = "hidden";
  peep();
  setTimeout(() => {
    timeUp = true;
    button.innerHTML = "Try again?";
    button.style.visibility = "visible";
  }, 10000);
}

function bonk(e) {
  if (!e.isTrusted) return;
  score++;
  this.classList.remove("up");
  scoreBoard.textContent = score;
}

moles.forEach((mole) => mole.addEventListener("click", bonk));
```

2. Connect-Four-Game

index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
      <meta charset="UTF-8">
      <meta name="viewport" content="width=device-width, initial-scale=1.0">
      <title>Talha - Connect Four</title>

      <!--
👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇-
->
      <!-- Also uploaded the demo of this code in a gif :
https://c.tenor.com/x8v1oNUOmg4AAAAd/tenor.gif-->
      <!--
👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇-
->

      <!--
👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇-
->
      <!-- More html-css-js Games Calculators Games Cards Elements Projects
on https://www.github.com/he-is-talha -->
      <!--
👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇-
->

      <link rel="icon" href="https://i.ibb.co/M6KTWnf/pic.jpg">

      <!-- Google Fonts -->
      <link rel="preconnect" href="https://fonts.gstatic.com">
      <link
href="https://fonts.googleapis.com/css2?family=Poppins:wght@500&display=swap"
rel="stylesheet">

      <!-- CSS -->
      <link rel="stylesheet" href="style.css">

</head>
<body>

      <div id="main-container">

            <div id="player">

                  <h1 id="player-type">Player - 1</h1>

            </div>

            <div id="grid">

                  <div class="row">

                        <div class="col">

                              <button class="btn btn-1"></button>

                        </div>

                        <div class="col">
```

```
                <button class="btn btn-2"></button>
            </div>
            <div class="col">
                <button class="btn btn-3"></button>
            </div>
            <div class="col">
                <button class="btn btn-4"></button>
            </div>
            <div class="col">
                <button class="btn btn-5"></button>
            </div>
            <div class="col">
                <button class="btn btn-6"></button>
            </div>
            <div class="col">
                <button class="btn btn-7"></button>
            </div>
        </div>
        <div class="row">
            <div class="col">
                <button class="btn btn-8"></button>
            </div>
            <div class="col">
                <button class="btn btn-9"></button>
            </div>
            <div class="col">
                <button class="btn btn-10"></button>
            </div>
            <div class="col">
                <button class="btn btn-11"></button>
            </div>
```

```
            <div class="col">
                  <button class="btn btn-12"></button>
            </div>
            <div class="col">
                  <button class="btn btn-13"></button>
            </div>
            <div class="col">
                  <button class="btn btn-14"></button>
            </div>
      </div>
      <div class="row">
            <div class="col">
                  <button class="btn btn-15"></button>
            </div>
            <div class="col">
                  <button class="btn btn-16"></button>
            </div>
            <div class="col">
                  <button class="btn btn-17"></button>
            </div>
            <div class="col">
                  <button class="btn btn-18"></button>
            </div>
            <div class="col">
                  <button class="btn btn-19"></button>
            </div>
            <div class="col">
                  <button class="btn btn-20"></button>
            </div>
            <div class="col">
                  <button class="btn btn-21"></button>
            </div>
```

```
</div>
<div class="row">
      <div class="col">
            <button class="btn btn-22"></button>
      </div>
      <div class="col">
            <button class="btn btn-23"></button>
      </div>
      <div class="col">
            <button class="btn btn-24"></button>
      </div>
      <div class="col">
            <button class="btn btn-25"></button>
      </div>
      <div class="col">
            <button class="btn btn-26"></button>
      </div>
      <div class="col">
            <button class="btn btn-27"></button>
      </div>
      <div class="col">
            <button class="btn btn-28"></button>
      </div>
</div>
<div class="row">
      <div class="col">
            <button class="btn btn-29"></button>
      </div>
      <div class="col">
            <button class="btn btn-30"></button>
      </div>
```

```
            <div class="col">
                  <button class="btn btn-31"></button>
            </div>
            <div class="col">
                  <button class="btn btn-32"></button>
            </div>
            <div class="col">
                  <button class="btn btn-33"></button>
            </div>
            <div class="col">
                  <button class="btn btn-34"></button>
            </div>
            <div class="col">
                  <button class="btn btn-35"></button>
            </div>
      </div>
      <div class="row">
            <div class="col">
                  <button class="btn btn-36"></button>
            </div>
            <div class="col">
                  <button class="btn btn-37"></button>
            </div>
            <div class="col">
                  <button class="btn btn-38"></button>
            </div>
            <div class="col">
                  <button class="btn btn-39"></button>
            </div>
            <div class="col">
                  <button class="btn btn-40"></button>
            </div>
```

```
                        <div class="col">
                              <button class="btn btn-41"></button>
                        </div>
                        <div class="col">
                              <button class="btn btn-42"></button>
                        </div>
                  </div>
            </div>
            <button type="button" id="reset-btn">Play Again</button>
      </div>
      <script src="script.js"></script>
</body>
</html>
```

style.css

```
body {
      background-color: #e9e7fd;
}

/* Main Container */

#main-container {

      align-items: center;
      display: flex;
      flex-direction: column;
      justify-content: center;
      min-height: 100vh;

}

/* Player Details */

#player {

      background-color: #d5deff;
      border: 8px solid #4f3ff0;
      border-radius: 10px;
      margin-top: 50px;
      padding: 20px;
      width: 550px;

}

#player-type {

      color: #4f3ff0;
      font-family: "Poppins";
      letter-spacing: 5px;
      text-align: center;
```

```
      text-transform: uppercase;

}

/* Grid */

#grid {

      background-color: #4f3ff0;
      border: 3.5px solid #d5deff;
      border-radius: 8px;
      box-shadow: 2px 3px 7px grey;
      margin-top: 50px;
      max-width: 600px;
      padding: 3px;

}

/* Grid Row */

.row {

      display: flex;

}

/* Grid Column */

.col {

      align-items: center;
      background-color: #d5deff;
      border: 1px solid  #4f3ff0;
      border-radius: 5px;
      display: flex;
      justify-content: center;
      height: 75px;
      margin: 5px;
      width:  75px;

}

/* Buttons */

.btn {

      background-color: transparent;
      border: none;
      color: transparent;
      height: 100%;
      padding: 0;
      width: 100%;

}

#reset-btn {

      background-color: transparent;
      border: 2px solid #4f3ff0;
      border-radius: 5px;
      color: #4f3ff0;
      font-family: "Poppins";
      font-size: 1.5rem;
```

```
      margin: 50px 0;
      padding: 10px 40px;
      text-transform: uppercase;
      transition: 0.7s;

}

#reset-btn:hover {

      background-color: #4f3ff0;
      color: #d5deff;
      cursor: pointer;
      transition: 0.7s;

}

/* Player - 1 Buttons */

.btn-player-1 {

      background-color: #34c471;
      border: 2px solid #34c471;
      border-radius: 50%;
      color: red;
      height:  50px;
      width: 50px;

}

/* Player - 2 Buttons */

.btn-player-2 {

      background-color: #df3670;
      border: 2px solid #df3670;
      border-radius: 50%;
      color: red;
      height:  50px;
      width: 50px;

}

/* Media Queries */

@media (max-width: 800px) {
      #grid {
            width: 500px;
      }

      .col {
            height: 62px;
            margin: 4px;
            width: 62px;
      }

      #player {
            width: 450px;
      }

      #reset-btn {
            font-size: 1.2rem;
      }
```

```
      .btn-player-1 {
            height: 40px;
            width: 40px;
      }

      .btn-player-2 {
            height: 40px;
            width: 40px;
      }
}

@media (max-width: 550px) {
      #grid {
            width: 400px;
      }

      .col {
            height: 50px;
            margin: 3px;
            width: 50px;
      }

      #player {
            width: 350px;
      }

      #reset-btn {
            font-size: 1rem;
      }

      .btn-player-1 {
            height: 30px;
            width: 30px;
      }

      .btn-player-2 {
            height: 30px;
            width: 30px;
      }
}

@media (max-width:  450px) {
      #grid {
            width: 90%;
      }

      .col {
            height: 40px;
            margin: 2px;
      }

      #player {
            align-items: center;
            display: flex;
            border-width: 5px;
            justify-content: center;
            height: 30px;
            width: 78%;
      }

      #player-type {
            font-size: 1.2rem;
      }
```

```
      #reset-btn {
            font-size: 0.8rem;
      }

      .btn-player-1 {
            height: 20px;
            width: 20px;
      }

      .btn-player-2 {
            height: 20px;
            width: 20px;
      }
}
```

main.js

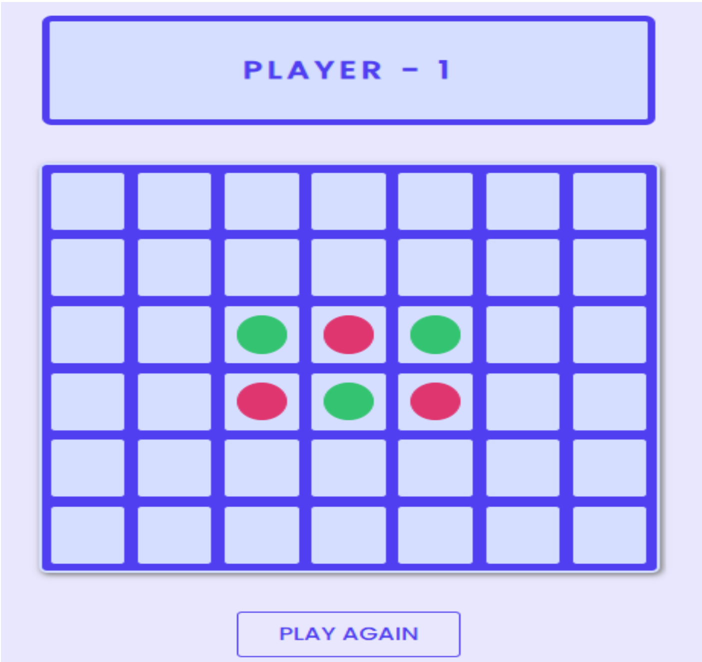
```
// DOM Variables

var buttons = document.getElementsByClassName("btn");
var reset = document.getElementById("reset-btn");
var playerType = document.getElementById("player-type");

// Game Flow Variables

var playerNumber = 1; // Initially player - 1 gets to start his/her turn

var filledGrid = []; // Player board

var filledCells = 0; // No. of cells that has been filled

for(var i = 0; i < 6; i++) {

      var arr = [-1 , -1 , -1 , -1 , -1 , -1 , -1]; // Board is initialised
with -1
      filledGrid.push(arr);

}


// Event Listener for Buttons

reset.addEventListener("click" , function() {

      resetBoard();

});

for(var i = 0; i < buttons.length; i++) {

      // Handing the Event when button was clicked

      buttons[i].addEventListener("click" , function() {

            // Make move and disable the button to avoid furthur clicking it
again

            var buttonNo = this.classList[1];
            makeMove(this , buttonNo.slice(4));

      });
```

```
}


// Function to Make Move on the passed button and disable it
function makeMove(button , buttonNo) {

     var row = buttonNo % 7 === 0 ? Math.floor(buttonNo / 7) - 1 :
Math.floor(buttonNo / 7);
     var col = buttonNo % 7 === 0 ? 6: (buttonNo % 7) - 1;

     if(playerNumber === 1) {

          button.classList.add("btn-player-1");


          filledGrid[row][col] = 1;
          filledCells++;


          if(playerWon(row , col , 1) === true) {
               setTimeout(function() {
                    alert("Game Over: Green Wins");
                    resetBoard();
               } , 200);
          }

          // Update the player
          playerNumber = 2;
          playerType.textContent = "Player - 2";

     } else {

          button.classList.add("btn-player-2");


          filledGrid[row][col] = 2;
          filledCells++;

          if(playerWon(row , col , 2) === true) {
               setTimeout(function() {
                    alert("Game Over : Red Wins");
                    resetBoard();
               } , 200);
          }

          // Update the player
          playerNumber = 1;
          playerType.textContent = "Player - 1";

     }

     // If all the cells has been filled

     if(filledCells === 42) {
          setTimeout(function() {
               alert("Game Draw");
               resetBoard();
          } , 200);
          return;
     }

     // Disable the button is the move is made
     setTimeout(function () {
```

```
            button.disabled = true;
      },10);

}

function playerWon(row , col , player) {

      var count = 0;

      // Check for columns

      for(var i = 0; i < 7; i++) {
            if(filledGrid[row][i] === player) {
                  count++;
                  if(count === 4) return true;
            } else {
                  count = 0;
            }

      }

      count = 0;

      // Check for Rows

      for(var i = 0; i < 6; i++) {
            if(filledGrid[i][col] === player) {
                  count++;
                  if(count === 4) return true;
            } else {
                  count = 0;
            }
      }


      count = 0;

      // Check for primary diagonal

      if(row >= col) {

            var i = row - col;
            var j = 0;

            for(; i <= 5; i++ , j++) {
                  if(filledGrid[i][j] === player) {
                        count++;
                        if(count == 4) return true;
                  } else {
                        count = 0;
                  }
            }
      } else {

            var i = 0;
            var j = col - row;

            for(; j <= 6; i++ , j++) {
                  if(filledGrid[i][j] === player) {
                        count++;
                        if(count == 4) return true;
                  } else {
                        count = 0;
```

```
                }
            }

        }

        count = 0;

        // Check for secondary diagonal

        if(row + col <= 5) {

            var i = row + col;
            var j = 0;

            for(; i >= 0 && j <= row + col; i-- , j++) {
                if(filledGrid[i][j] === player) {
                    count++;
                    if(count == 4) return true;
                } else {
                    count = 0;
                }
            }

        } else {

            var i = 5;
            var j = row + col - 5;

            for(; j <= 6; j++ , i--) {
                if(filledGrid[i][j] === player) {
                    count++;
                    if(count == 4) return true;
                } else {
                    count = 0;
                }
            }

        }
        return false;

}

// Function to reset the Board completely
function resetBoard() {

        // Remove all the disabled buttons and the styles

        for(var i = 0; i < buttons.length; i++) {
            buttons[i].disabled = false;
            buttons[i].classList.remove("btn-player-1");
            buttons[i].classList.remove("btn-player-2");
        }


        // Player Number is changed to 1

        playerNumber = 1;
        playerType.textContent = "Player - 1";


        // Filled Cells is changed to 0

        filledCells = 0;
```

```
        // Filling the Board with -1

        for(var i = 0; i < 6; i++) {
                for(var j = 0; j < 7; j++) {
                        filledGrid[i][j] = -1;
                }
        }

}
```



3. Crossy-Road-Game

index.html

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <title>Talha - Crossy Road Game Clone</title>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width" />
    <link rel="stylesheet" href="style.css" />
    <link rel="icon" href="https://i.ibb.co/M6KTWnf/pic.jpg" />

    <!--
👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇-
->
    <!-- Also uploaded the demo of this code in a gif :
https://c.tenor.com/x8v1oNUOmg4AAAAd/tenor.gif-->
    <!--
👆👆👆👆👆👆👆👆👆👆👆👆👆👆👆👆👆👆👆👆👆👆👆👆👆👆👆👆👆👆👆👆👆👆👆👆👆👆👆👆👆👆👆-
->


    <!--
👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇👇-
->
    <!-- More html-css-js Games Calculators Games Cards Elements Projects on
https://www.github.com/he-is-talha -->
    <!--
👆👆👆👆👆👆👆👆👆👆👆👆👆👆👆👆👆👆👆👆👆👆👆👆👆👆👆👆👆👆👆👆👆👆👆👆👆👆👆👆👆👆👆-
->
```

```
  </head>
  <body>
    <div id="counter">0</div>

    <div id="controlls">
      <div>
        <button id="forward">
          <svg width="30" height="30" viewBox="0 0 10 10">
            <g transform="rotate(0, 5,5)">
              <path d="M5,4 L7,6 L3,6 L5,4" />
            </g>
          </svg>
        </button>
        <button id="left">
          <svg width="30" height="30" viewBox="0 0 10 10">
            <g transform="rotate(-90, 5,5)">
              <path d="M5,4 L7,6 L3,6 L5,4" />
            </g>
          </svg>
        </button>
        <button id="backward">
          <svg width="30" height="30" viewBox="0 0 10 10">
            <g transform="rotate(180, 5,5)">
              <path d="M5,4 L7,6 L3,6 L5,4" />
            </g>
          </svg>
        </button>
        <button id="right">
          <svg width="30" height="30" viewBox="0 0 10 10">
            <g transform="rotate(90, 5,5)">
              <path d="M5,4 L7,6 L3,6 L5,4" />
            </g>
          </svg>
        </button>
      </div>
    </div>

    <div id="end">
      <button id="retry">Retry</button>
    </div>
    <script
src="https://cdnjs.cloudflare.com/ajax/libs/three.js/99/three.min.js"></scrip
t>
    <script src="script.js"></script>
  </body>
</html>
```

style.css

```
@import url("https://fonts.googleapis.com/css?family=Press+Start+2P");

body {
  margin: 0;
  font-family: "Press Start 2P", cursive;
  font-size: 2em;
  color: white;
}
button {
  outline: none;
  cursor: pointer;
  border: none;
  box-shadow: 3px 5px 0px 0px rgba(0, 0, 0, 0.75);
}
```

```
#counter {
  position: absolute;
  top: 20px;
  right: 20px;
}
#end {
  position: absolute;
  min-width: 100%;
  min-height: 100%;
  display: flex;
  align-items: center;
  justify-content: center;
  visibility: hidden;
}
#end button {
  background-color: red;
  padding: 20px 50px 20px 50px;
  font-family: inherit;
  font-size: inherit;
}
#controlls {
  position: absolute;
  min-width: 100%;
  min-height: 100%;
  display: flex;
  align-items: flex-end;
  justify-content: center;
}
#controlls div {
  display: grid;
  grid-template-columns: 50px 50px 50px;
  grid-template-rows: auto auto;
  grid-column-gap: 10px;
  grid-row-gap: 10px;
  margin-bottom: 20px;
}
#controlls button {
  width: 100%;
  background-color: white;
  border: 1px solid lightgray;
}
#controlls button:first-of-type {
  grid-column: 1/-1;
}
```

main.js

```
const counterDOM = document.getElementById("counter");
const endDOM = document.getElementById("end");

const scene = new THREE.Scene();

const distance = 500;
const camera = new THREE.OrthographicCamera(
  window.innerWidth / -2,
  window.innerWidth / 2,
  window.innerHeight / 2,
  window.innerHeight / -2,
  0.1,
  10000
);

camera.rotation.x = (50 * Math.PI) / 180;
```

```
camera.rotation.y = (20 * Math.PI) / 180;
camera.rotation.z = (10 * Math.PI) / 180;

const initialCameraPositionY = -Math.tan(camera.rotation.x) * distance;
const initialCameraPositionX =
  Math.tan(camera.rotation.y) *
  Math.sqrt(distance ** 2 + initialCameraPositionY ** 2);
camera.position.y = initialCameraPositionY;
camera.position.x = initialCameraPositionX;
camera.position.z = distance;

const zoom = 2;

const chickenSize = 15;

const positionWidth = 42;
const columns = 17;
const boardWidth = positionWidth * columns;

const stepTime = 200; // Miliseconds it takes for the chicken to take a step
forward, backward, left or right

let lanes;
let currentLane;
let currentColumn;

let previousTimestamp;
let startMoving;
let moves;
let stepStartTimestamp;

const carFrontTexture = new Texture(40, 80, [{ x: 0, y: 10, w: 30, h: 60 }]);
const carBackTexture = new Texture(40, 80, [{ x: 10, y: 10, w: 30, h: 60 }]);
const carRightSideTexture = new Texture(110, 40, [
  { x: 10, y: 0, w: 50, h: 30 },
  { x: 70, y: 0, w: 30, h: 30 },
]);
const carLeftSideTexture = new Texture(110, 40, [
  { x: 10, y: 10, w: 50, h: 30 },
  { x: 70, y: 10, w: 30, h: 30 },
]);

const truckFrontTexture = new Texture(30, 30, [{ x: 15, y: 0, w: 10, h: 30
}]);
const truckRightSideTexture = new Texture(25, 30, [
  { x: 0, y: 15, w: 10, h: 10 },
]);
const truckLeftSideTexture = new Texture(25, 30, [
  { x: 0, y: 5, w: 10, h: 10 },
]);

const generateLanes = () =>
  [-9, -8, -7, -6, -5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
    .map((index) => {
      const lane = new Lane(index);
      lane.mesh.position.y = index * positionWidth * zoom;
      scene.add(lane.mesh);
      return lane;
    })
    .filter((lane) => lane.index >= 0);

const addLane = () => {
  const index = lanes.length;
```

```
  const lane = new Lane(index);
  lane.mesh.position.y = index * positionWidth * zoom;
  scene.add(lane.mesh);
  lanes.push(lane);
};

const chicken = new Chicken();
scene.add(chicken);

hemiLight = new THREE.HemisphereLight(0xffffff, 0xffffff, 0.6);
scene.add(hemiLight);

const initialDirLightPositionX = -100;
const initialDirLightPositionY = -100;
dirLight = new THREE.DirectionalLight(0xffffff, 0.6);
dirLight.position.set(initialDirLightPositionX, initialDirLightPositionY,
200);
dirLight.castShadow = true;
dirLight.target = chicken;
scene.add(dirLight);

dirLight.shadow.mapSize.width = 2048;
dirLight.shadow.mapSize.height = 2048;
var d = 500;
dirLight.shadow.camera.left = -d;
dirLight.shadow.camera.right = d;
dirLight.shadow.camera.top = d;
dirLight.shadow.camera.bottom = -d;

// var helper = new THREE.CameraHelper( dirLight.shadow.camera );
// var helper = new THREE.CameraHelper( camera );
// scene.add(helper)

backLight = new THREE.DirectionalLight(0x000000, 0.4);
backLight.position.set(200, 200, 50);
backLight.castShadow = true;
scene.add(backLight);

const laneTypes = ["car", "truck", "forest"];
const laneSpeeds = [2, 2.5, 3];
const vechicleColors = [0xa52523, 0xbdb638, 0x78b14b];
const threeHeights = [20, 45, 60];

const initaliseValues = () => {
  lanes = generateLanes();

  currentLane = 0;
  currentColumn = Math.floor(columns / 2);

  previousTimestamp = null;

  startMoving = false;
  moves = [];
  stepStartTimestamp;

  chicken.position.x = 0;
  chicken.position.y = 0;

  camera.position.y = initialCameraPositionY;
  camera.position.x = initialCameraPositionX;

  dirLight.position.x = initialDirLightPositionX;
  dirLight.position.y = initialDirLightPositionY;
```

```
};

initaliseValues();

const renderer = new THREE.WebGLRenderer({
  alpha: true,
  antialias: true,
});
renderer.shadowMap.enabled = true;
renderer.shadowMap.type = THREE.PCFSoftShadowMap;
renderer.setSize(window.innerWidth, window.innerHeight);
document.body.appendChild(renderer.domElement);

function Texture(width, height, rects) {
  const canvas = document.createElement("canvas");
  canvas.width = width;
  canvas.height = height;
  const context = canvas.getContext("2d");
  context.fillStyle = "#ffffff";
  context.fillRect(0, 0, width, height);
  context.fillStyle = "rgba(0,0,0,0.6)";
  rects.forEach((rect) => {
    context.fillRect(rect.x, rect.y, rect.w, rect.h);
  });
  return new THREE.CanvasTexture(canvas);
}

function Wheel() {
  const wheel = new THREE.Mesh(
    new THREE.BoxBufferGeometry(12 * zoom, 33 * zoom, 12 * zoom),
    new THREE.MeshLambertMaterial({ color: 0x333333, flatShading: true })
  );
  wheel.position.z = 6 * zoom;
  return wheel;
}

function Car() {
  const car = new THREE.Group();
  const color =
    vechicleColors[Math.floor(Math.random() * vechicleColors.length)];

  const main = new THREE.Mesh(
    new THREE.BoxBufferGeometry(60 * zoom, 30 * zoom, 15 * zoom),
    new THREE.MeshPhongMaterial({ color, flatShading: true })
  );
  main.position.z = 12 * zoom;
  main.castShadow = true;
  main.receiveShadow = true;
  car.add(main);

  const cabin = new THREE.Mesh(
    new THREE.BoxBufferGeometry(33 * zoom, 24 * zoom, 12 * zoom),
    [
      new THREE.MeshPhongMaterial({
        color: 0xcccccc,
        flatShading: true,
        map: carBackTexture,
      }),
      new THREE.MeshPhongMaterial({
        color: 0xcccccc,
        flatShading: true,
        map: carFrontTexture,
      }),
```

```
      new THREE.MeshPhongMaterial({
        color: 0xcccccc,
        flatShading: true,
        map: carRightSideTexture,
      }),
      new THREE.MeshPhongMaterial({
        color: 0xcccccc,
        flatShading: true,
        map: carLeftSideTexture,
      }),
      new THREE.MeshPhongMaterial({ color: 0xcccccc, flatShading: true }), //
top
      new THREE.MeshPhongMaterial({ color: 0xcccccc, flatShading: true }), //
bottom
    ]
  );
  cabin.position.x = 6 * zoom;
  cabin.position.z = 25.5 * zoom;
  cabin.castShadow = true;
  cabin.receiveShadow = true;
  car.add(cabin);

  const frontWheel = new Wheel();
  frontWheel.position.x = -18 * zoom;
  car.add(frontWheel);

  const backWheel = new Wheel();
  backWheel.position.x = 18 * zoom;
  car.add(backWheel);

  car.castShadow = true;
  car.receiveShadow = false;

  return car;
}

function Truck() {
  const truck = new THREE.Group();
  const color =
    vechicleColors[Math.floor(Math.random() * vechicleColors.length)];

  const base = new THREE.Mesh(
    new THREE.BoxBufferGeometry(100 * zoom, 25 * zoom, 5 * zoom),
    new THREE.MeshLambertMaterial({ color: 0xb4c6fc, flatShading: true })
  );
  base.position.z = 10 * zoom;
  truck.add(base);

  const cargo = new THREE.Mesh(
    new THREE.BoxBufferGeometry(75 * zoom, 35 * zoom, 40 * zoom),
    new THREE.MeshPhongMaterial({ color: 0xb4c6fc, flatShading: true })
  );
  cargo.position.x = 15 * zoom;
  cargo.position.z = 30 * zoom;
  cargo.castShadow = true;
  cargo.receiveShadow = true;
  truck.add(cargo);

  const cabin = new THREE.Mesh(
    new THREE.BoxBufferGeometry(25 * zoom, 30 * zoom, 30 * zoom),
    [
      new THREE.MeshPhongMaterial({ color, flatShading: true }), // back
      new THREE.MeshPhongMaterial({
```

```
        color,
        flatShading: true,
        map: truckFrontTexture,
      }),
      new THREE.MeshPhongMaterial({
        color,
        flatShading: true,
        map: truckRightSideTexture,
      }),
      new THREE.MeshPhongMaterial({
        color,
        flatShading: true,
        map: truckLeftSideTexture,
      }),
      new THREE.MeshPhongMaterial({ color, flatShading: true }), // top
      new THREE.MeshPhongMaterial({ color, flatShading: true }), // bottom
    ]
  );
  cabin.position.x = -40 * zoom;
  cabin.position.z = 20 * zoom;
  cabin.castShadow = true;
  cabin.receiveShadow = true;
  truck.add(cabin);

  const frontWheel = new Wheel();
  frontWheel.position.x = -38 * zoom;
  truck.add(frontWheel);

  const middleWheel = new Wheel();
  middleWheel.position.x = -10 * zoom;
  truck.add(middleWheel);

  const backWheel = new Wheel();
  backWheel.position.x = 30 * zoom;
  truck.add(backWheel);

  return truck;
}

function Three() {
  const three = new THREE.Group();

  const trunk = new THREE.Mesh(
    new THREE.BoxBufferGeometry(15 * zoom, 15 * zoom, 20 * zoom),
    new THREE.MeshPhongMaterial({ color: 0x4d2926, flatShading: true })
  );
  trunk.position.z = 10 * zoom;
  trunk.castShadow = true;
  trunk.receiveShadow = true;
  three.add(trunk);

  height = threeHeights[Math.floor(Math.random() * threeHeights.length)];

  const crown = new THREE.Mesh(
    new THREE.BoxBufferGeometry(30 * zoom, 30 * zoom, height * zoom),
    new THREE.MeshLambertMaterial({ color: 0x7aa21d, flatShading: true })
  );
  crown.position.z = (height / 2 + 20) * zoom;
  crown.castShadow = true;
  crown.receiveShadow = false;
  three.add(crown);

  return three;
```

```
}

function Chicken() {
  const chicken = new THREE.Group();

  const body = new THREE.Mesh(
    new THREE.BoxBufferGeometry(
      chickenSize * zoom,
      chickenSize * zoom,
      20 * zoom
    ),
    new THREE.MeshPhongMaterial({ color: 0xffffff, flatShading: true })
  );
  body.position.z = 10 * zoom;
  body.castShadow = true;
  body.receiveShadow = true;
  chicken.add(body);

  const rowel = new THREE.Mesh(
    new THREE.BoxBufferGeometry(2 * zoom, 4 * zoom, 2 * zoom),
    new THREE.MeshLambertMaterial({ color: 0xf0619a, flatShading: true })
  );
  rowel.position.z = 21 * zoom;
  rowel.castShadow = true;
  rowel.receiveShadow = false;
  chicken.add(rowel);

  return chicken;
}

function Road() {
  const road = new THREE.Group();

  const createSection = (color) =>
    new THREE.Mesh(
      new THREE.PlaneBufferGeometry(boardWidth * zoom, positionWidth * zoom),
      new THREE.MeshPhongMaterial({ color })
    );

  const middle = createSection(0x454a59);
  middle.receiveShadow = true;
  road.add(middle);

  const left = createSection(0x393d49);
  left.position.x = -boardWidth * zoom;
  road.add(left);

  const right = createSection(0x393d49);
  right.position.x = boardWidth * zoom;
  road.add(right);

  return road;
}

function Grass() {
  const grass = new THREE.Group();

  const createSection = (color) =>
    new THREE.Mesh(
      new THREE.BoxBufferGeometry(
        boardWidth * zoom,
        positionWidth * zoom,
        3 * zoom
```

```
      ),
      new THREE.MeshPhongMaterial({ color })
    );

  const middle = createSection(0xbaf455);
  middle.receiveShadow = true;
  grass.add(middle);

  const left = createSection(0x99c846);
  left.position.x = -boardWidth * zoom;
  grass.add(left);

  const right = createSection(0x99c846);
  right.position.x = boardWidth * zoom;
  grass.add(right);

  grass.position.z = 1.5 * zoom;
  return grass;
}

function Lane(index) {
  this.index = index;
  this.type =
    index <= 0
      ? "field"
      : laneTypes[Math.floor(Math.random() * laneTypes.length)];

  switch (this.type) {
    case "field": {
      this.type = "field";
      this.mesh = new Grass();
      break;
    }
    case "forest": {
      this.mesh = new Grass();

      this.occupiedPositions = new Set();
      this.threes = [1, 2, 3, 4].map(() => {
        const three = new Three();
        let position;
        do {
          position = Math.floor(Math.random() * columns);
        } while (this.occupiedPositions.has(position));
        this.occupiedPositions.add(position);
        three.position.x =
          (position * positionWidth + positionWidth / 2) * zoom -
          (boardWidth * zoom) / 2;
        this.mesh.add(three);
        return three;
      });
      break;
    }
    case "car": {
      this.mesh = new Road();
      this.direction = Math.random() >= 0.5;

      const occupiedPositions = new Set();
      this.vechicles = [1, 2, 3].map(() => {
        const vechicle = new Car();
        let position;
        do {
          position = Math.floor((Math.random() * columns) / 2);
        } while (occupiedPositions.has(position));
```

```
        occupiedPositions.add(position);
        vechicle.position.x =
          (position * positionWidth * 2 + positionWidth / 2) * zoom -
          (boardWidth * zoom) / 2;
        if (!this.direction) vechicle.rotation.z = Math.PI;
        this.mesh.add(vechicle);
        return vechicle;
      });

      this.speed = laneSpeeds[Math.floor(Math.random() * laneSpeeds.length)];
      break;
    }
    case "truck": {
      this.mesh = new Road();
      this.direction = Math.random() >= 0.5;

      const occupiedPositions = new Set();
      this.vechicles = [1, 2].map(() => {
        const vechicle = new Truck();
        let position;
        do {
          position = Math.floor((Math.random() * columns) / 3);
        } while (occupiedPositions.has(position));
        occupiedPositions.add(position);
        vechicle.position.x =
          (position * positionWidth * 3 + positionWidth / 2) * zoom -
          (boardWidth * zoom) / 2;
        if (!this.direction) vechicle.rotation.z = Math.PI;
        this.mesh.add(vechicle);
        return vechicle;
      });

      this.speed = laneSpeeds[Math.floor(Math.random() * laneSpeeds.length)];
      break;
    }
  }
}

document.querySelector("#retry").addEventListener("click", () => {
  lanes.forEach((lane) => scene.remove(lane.mesh));
  initaliseValues();
  endDOM.style.visibility = "hidden";
});

document
  .getElementById("forward")
  .addEventListener("click", () => move("forward"));

document
  .getElementById("backward")
  .addEventListener("click", () => move("backward"));

document.getElementById("left").addEventListener("click", () =>
move("left"));

document.getElementById("right").addEventListener("click", () =>
move("right"));

window.addEventListener("keydown", (event) => {
  if (event.keyCode == "38") {
    // up arrow
    move("forward");
  } else if (event.keyCode == "40") {
```

```
    // down arrow
    move("backward");
  } else if (event.keyCode == "37") {
    // left arrow
    move("left");
  } else if (event.keyCode == "39") {
    // right arrow
    move("right");
  }
});

function move(direction) {
  const finalPositions = moves.reduce(
    (position, move) => {
      if (move === "forward")
        return { lane: position.lane + 1, column: position.column };
      if (move === "backward")
        return { lane: position.lane - 1, column: position.column };
      if (move === "left")
        return { lane: position.lane, column: position.column - 1 };
      if (move === "right")
        return { lane: position.lane, column: position.column + 1 };
    },
    { lane: currentLane, column: currentColumn }
  );

  if (direction === "forward") {
    if (
      lanes[finalPositions.lane + 1].type === "forest" &&
      lanes[finalPositions.lane + 1].occupiedPositions.has(
        finalPositions.column
      )
    )
      return;
    if (!stepStartTimestamp) startMoving = true;
    addLane();
  } else if (direction === "backward") {
    if (finalPositions.lane === 0) return;
    if (
      lanes[finalPositions.lane - 1].type === "forest" &&
      lanes[finalPositions.lane - 1].occupiedPositions.has(
        finalPositions.column
      )
    )
      return;
    if (!stepStartTimestamp) startMoving = true;
  } else if (direction === "left") {
    if (finalPositions.column === 0) return;
    if (
      lanes[finalPositions.lane].type === "forest" &&
      lanes[finalPositions.lane].occupiedPositions.has(
        finalPositions.column - 1
      )
    )
      return;
    if (!stepStartTimestamp) startMoving = true;
  } else if (direction === "right") {
    if (finalPositions.column === columns - 1) return;
    if (
      lanes[finalPositions.lane].type === "forest" &&
      lanes[finalPositions.lane].occupiedPositions.has(
        finalPositions.column + 1
      )
```

```
    )
      return;
    if (!stepStartTimestamp) startMoving = true;
  }
  moves.push(direction);
}

function animate(timestamp) {
  requestAnimationFrame(animate);

  if (!previousTimestamp) previousTimestamp = timestamp;
  const delta = timestamp - previousTimestamp;
  previousTimestamp = timestamp;

  // Animate cars and trucks moving on the lane
  lanes.forEach((lane) => {
    if (lane.type === "car" || lane.type === "truck") {
      const aBitBeforeTheBeginingOfLane =
        (-boardWidth * zoom) / 2 - positionWidth * 2 * zoom;
      const aBitAfterTheEndOFLane =
        (boardWidth * zoom) / 2 + positionWidth * 2 * zoom;
      lane.vechicles.forEach((vechicle) => {
        if (lane.direction) {
          vechicle.position.x =
            vechicle.position.x < aBitBeforeTheBeginingOfLane
              ? aBitAfterTheEndOFLane
              : (vechicle.position.x -= (lane.speed / 16) * delta);
        } else {
          vechicle.position.x =
            vechicle.position.x > aBitAfterTheEndOFLane
              ? aBitBeforeTheBeginingOfLane
              : (vechicle.position.x += (lane.speed / 16) * delta);
        }
      });
    }
  });

  if (startMoving) {
    stepStartTimestamp = timestamp;
    startMoving = false;
  }

  if (stepStartTimestamp) {
    const moveDeltaTime = timestamp - stepStartTimestamp;
    const moveDeltaDistance =
      Math.min(moveDeltaTime / stepTime, 1) * positionWidth * zoom;
    const jumpDeltaDistance =
      Math.sin(Math.min(moveDeltaTime / stepTime, 1) * Math.PI) * 8 * zoom;
    switch (moves[0]) {
      case "forward": {
        const positionY =
          currentLane * positionWidth * zoom + moveDeltaDistance;
        camera.position.y = initialCameraPositionY + positionY;
        dirLight.position.y = initialDirLightPositionY + positionY;
        chicken.position.y = positionY; // initial chicken position is 0

        chicken.position.z = jumpDeltaDistance;
        break;
      }
      case "backward": {
        positionY = currentLane * positionWidth * zoom - moveDeltaDistance;
        camera.position.y = initialCameraPositionY + positionY;
        dirLight.position.y = initialDirLightPositionY + positionY;
```

```
        chicken.position.y = positionY;

        chicken.position.z = jumpDeltaDistance;
        break;
      }
      case "left": {
        const positionX =
          (currentColumn * positionWidth + positionWidth / 2) * zoom -
          (boardWidth * zoom) / 2 -
          moveDeltaDistance;
        camera.position.x = initialCameraPositionX + positionX;
        dirLight.position.x = initialDirLightPositionX + positionX;
        chicken.position.x = positionX; // initial chicken position is 0
        chicken.position.z = jumpDeltaDistance;
        break;
      }
      case "right": {
        const positionX =
          (currentColumn * positionWidth + positionWidth / 2) * zoom -
          (boardWidth * zoom) / 2 +
          moveDeltaDistance;
        camera.position.x = initialCameraPositionX + positionX;
        dirLight.position.x = initialDirLightPositionX + positionX;
        chicken.position.x = positionX;

        chicken.position.z = jumpDeltaDistance;
        break;
      }
    }
    // Once a step has ended
    if (moveDeltaTime > stepTime) {
      switch (moves[0]) {
        case "forward": {
          currentLane++;
          counterDOM.innerHTML = currentLane;
          break;
        }
        case "backward": {
          currentLane--;
          counterDOM.innerHTML = currentLane;
          break;
        }
        case "left": {
          currentColumn--;
          break;
        }
        case "right": {
          currentColumn++;
          break;
        }
      }
      moves.shift();
      // If more steps are to be taken then restart counter otherwise stop
stepping
      stepStartTimestamp = moves.length === 0 ? null : timestamp;
    }
  }

  // Hit test
  if (
    lanes[currentLane].type === "car" ||
    lanes[currentLane].type === "truck"
  ) {
```

```
    const chickenMinX = chicken.position.x - (chickenSize * zoom) / 2;
    const chickenMaxX = chicken.position.x + (chickenSize * zoom) / 2;
    const vechicleLength = { car: 60, truck: 105 }[lanes[currentLane].type];
    lanes[currentLane].vechicles.forEach((vechicle) => {
      const carMinX = vechicle.position.x - (vechicleLength * zoom) / 2;
      const carMaxX = vechicle.position.x + (vechicleLength * zoom) / 2;
      if (chickenMaxX > carMinX && chickenMinX < carMaxX) {
        endDOM.style.visibility = "visible";
      }
    });
  }
  renderer.render(scene, camera);
}

requestAnimationFrame(animate);
```