<!DOCTYPE *html*>

<html *lang*="en">

<head>

    <meta *charset*="UTF-8">

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

    <title>AS2</title>

    <link *rel*="stylesheet" *href*="style.css">

    <link *rel*="shortcut icon" *href*="images/icon.png" *type*="image/x-icon">

    <script *src*="script.js"  *defer*></script>

    <script *src*="maze.js" ></script>

    <script *src*="leader.js" ></script>

    <script *src*="enemy.js" *defer*></script>

</head>

<body>

    <header>

        <h1>Snack Man!</h1>

    </header>

    <div *class*="menuDiv">

        <div *class*="menu">

            <img *src*="images/icon.png" *alt*="logo">

            <h1 >Start</h1>

        </div>

    </div>

    <div *class*="leaderboard">

        <h1>LeaderBoard:</h1>

        <ol>

        </ol>

    </div>

    <main>

    </main>

    <div *class*="lives">

        <h1>Lives:</h1>

        <ul>

        </ul>

    </div>

    <div *class*="score">

        <h1>Score:</h1>

        <p>0</p>

    </div>

    <div *class*="controls">

        <button *id*="lbttn">&#9664;</button> *<!-- LEFT -->*

        <button *id*="ubttn">&#9650;</button> *<!-- UP -->*

        <button *id*="rbttn">&#9654;</button> *<!-- RIGHT -->*

        <button *id*="dbttn">&#9660;</button> *<!-- DOWN -->*

    </div>

</body>

</html>

*/\* snackman css  \*/*

@import url('https://fonts.googleapis.com/css2?family=Press+Start+2P&display=swap');

\* {

    margin: 0;

    padding: 0;

    color: white;

    font-family: 'Press Start 2P';

}

body {

    background-color: black;

    height: 100vh;

    display: grid;

    grid-template-columns: 1fr 80vh 1fr;

    grid-template-rows: auto;

    grid-template-areas:

        "header header header"

        "lboard main score"

        "lboard main controls";

    align-items: center;

    justify-items: center;

}

header {

    grid-area: header;

    text-align: center;

}

*.menuDiv* {

    position: absolute;

    display: flex;

    height: 100vh;

    align-items: center;

    justify-content: center;

}

*.menuDiv* div {

    background-color: #ccc;

    cursor: pointer;

    border-radius: 20px;

    box-shadow: 4px 4px 4px #000;

    padding: 20px 30px;

    display: flex;

    height: 32px;

    z-index: 999;

}

*.menu* h1 {

    text-shadow: 2px 2px 2px #000;

    font-size: 2em;

    text-decoration: none;

    margin-left: 10px;

}

*/\* game over css  \*/*

*.endDiv* {

    position: absolute;

    display: none;

    height: 100vh;

    align-items: center;

    justify-content: center;

}

*.endDiv* div {

    background-color: #ccc;

    z-index: 100;

    cursor: pointer;

    border-radius: 20px;

    box-shadow: 4px 4px 4px #000;

    padding: 20px 30px;

    display: flex;

    height: 32px;

}

*.endDiv* h1 {

    text-shadow: 2px 2px 2px #000;

    font-size: 2em;

    text-decoration: none;

    margin-left: 10px;

}

header h1 {

    font-size: 4em;

}

main {

    grid-area: main;

    display: grid;

    width: 80vh;

    height: 80vh;

    align-items: center;

    justify-items: center;

}

*/\* Lives styling \*/*

*.lives* {

    position: absolute;

    top: 0;

    right: 0;

    height: auto;

    display: flex;

    align-items: center;

    margin: 0.5em;

}

*.lives* li {

    list-style: none;

    background-color: yellow;

    width: 2em;

    height: 2em;

    border-radius: 100%;

    margin: 5px;

}

*.lives* ul {

    display: flex;

}

*.score* {

    grid-area: score;

}

*.leaderboard* {

    grid-area: lboard;

}

*.leaderboard* ol {

    margin-left: 2.5em;

    margin-top: 1em;

    text-align: left;

}

*.leaderboard* ol li {

    font-size: 0.85em;

    padding: 1em 0;

}

div {

    height: 100%;

    text-align: center;

}

div h1 {

    font-size: 1.25em;

    text-decoration: underline;

}

div p {

    margin: 0.5em;

    font-size: 2em;

}

*#player* {

    background-color: yellow;

    border-radius: 100%;

    position: relative;

    width: 85%;

    height: 85%;

}

*.enemy* {

    background-color: green;

    border-radius: 100%;

    height: 90%;

    width: 90%;

    position: relative;

    z-index: 998;

}

*.point* {

    border-radius: 100%;

    background-color: white;

}

*.extraLife*{

    background-color: #f54263;

}

*.wall* {

    background-color: blue;

    width: 100%;

    height: 100%;

}

*/\* Styles for the player direction \*/*

*.up*, *.down*, *.left*, *.right* {

    width: 100%;

    height: 100%;

    background-color: black;

    animation: mouth 1.25s infinite steps(1, end);

    position: absolute;

    top: 0;

    left: 0;

}

*.up* { clip-path: polygon(0 0, 100% 0, 50% 50%); }

*.down* { clip-path: polygon(0 100%, 100% 100%, 50% 50%); }

*.left* { clip-path: polygon(0 0, 0 100%, 50% 50%); }

*.right* { clip-path: polygon(100% 0, 100% 100%, 50% 50%); }

@keyframes mouth {

    0%, 100% { opacity: 1; }

    50% { opacity: 0; }

}

*/\* Directional button styling \*/*

*.controls* {

    grid-area: controls;

    display: grid;

    grid-template-columns: repeat(3, 1fr);

    grid-template-rows: repeat(3, 1fr);

    grid-template-areas:

        ". upArrow ."

        "leftArrow . rightArrow"

        ". downArrow .";

    width: 25vh;

    height: 25vh;

}

button {

    background-color: rgb(10, 10, 10);

}

button*:nth-child*(1) { grid-area: leftArrow; }

button*:nth-child*(2) { grid-area: upArrow; }

button*:nth-child*(3) { grid-area: rightArrow; }

button*:nth-child*(4) { grid-area: downArrow; }

@keyframes deathAnimation {

    0% {

        transform: scale(1);

    }

    50% {

        transform: scale(0.5);

    }

    100% {

        transform: scale(0);

    }

}

@keyframes hitAnimation {

    0%, 100% {

        background-color: yellow;

        transform: scale(1);

    }

    50% {

        background-color: red;

        transform: scale(0.75);

    }

}

*.dead* {

    animation: deathAnimation 1.5s ease-in-out forwards;

}

*.hit* {

    animation: hitAnimation 1.5s ease-in-out;

}

*//this is the main js file that is responisble for most of the functionality of the game*

*// global variables*

*//movement key checks*

let upPressed = false;

let downPressed = false;

let leftPressed = false;

let rightPressed = false;

*//used for the score*

let num =0 ;

let score =0;

*// selects the main tag*

const main = document.querySelector('main');

*//lives global variables*

const liveIcon = document.querySelector('.lives ul ');

let aliveCheck =true ;

let canRemoveLife = true;

let lastCalled = 0;

const delay = 1500;

let counter = 0;

let hearts = 3;

let block;

*//creates  the  random maze*

randomMaze(height, width, maze);

MazeAssignment();

*// always displays the leaderBoard*

leaderboardAlways();

function lives(){

*// add the 3 lives at the start of the game*

    let lives = 3;

    for (let i =0; i<lives; i++){

        let life = document.createElement('li');

        liveIcon.append(life);

    }

   }

lives();

*// the function that starts the game on start button*

function play(event){

*// checks if a player collects a point*

    function pointCheck(){

        const points = document.querySelectorAll('.point');

        const postion = player.getBoundingClientRect()

        const pointOpacity = 0.1; *//sets the opacity of the point when collected*

        for (let i =0; i< points.length; i++) {

            let pointPostion = points[i].getBoundingClientRect()

            if (postion.right > pointPostion.left && postion.left < pointPostion.right && postion.bottom > pointPostion.top && postion.top < pointPostion.bottom){

                if (points[i].classList.contains('extraLife') && points[i].style.opacity !=pointOpacity ){

                    addLife();

            }

                if (points[i].style.opacity !=pointOpacity){

                    points[i].style.opacity = pointOpacity;

                    scoreupdate(points.length);

                }

            }

        }

    }

*// function to update the score called when a point is collected*

    function scoreupdate(points){

        let totalPoints= points;

        num+=1;

        const pElement =document.querySelector('.score p')

        score = Number(pElement.innerText);

        score ++;

        pElement.innerText = score

*// when a player gets all the points on the level  go to a next level*

        if(num == totalPoints){

            enemyMoveEnable = false;

            setTimeout(() => {

                clearMaze();

*// allows the player to move once the level is clear*

                play(event);

            },2000);

        }

     }

*// will display gameover message*

    function gameOver(){

        clearInterval(Maininterval);

        enemyMoveEnable = false;

        let gameOver = document.querySelector('.menu h1');

        let menu = document.querySelector('.menu');

        menu.style.display = 'flex';

        leaderBoard()

        gameOver.textContent = 'GAMEOVER!  RESTART?'

        gameOver.addEventListener('click',function(event){

*// reloads the screen*

            event.stopPropagation();

            location.reload();

        });

    }

*// checks if an enemy hits the player*

    function enemyCheck(){

        const enemys = document.querySelectorAll('.enemy');

*// gets the players postion*

        let playerPostion = player.getBoundingClientRect();

        for (let enemy of enemys) {

            let enemyPostion = enemy.getBoundingClientRect();

            if (playerPostion.left < enemyPostion.right && playerPostion.right > enemyPostion.left && playerPostion.bottom > enemyPostion.top && playerPostion.top < enemyPostion.bottom){

                if(aliveCheck && canRemoveLife){

                    canRemoveLife = false;

                    removeLife();

*// makes its so the player is invincable for 5 seconds*

                    setTimeout(() =>{

                        canRemoveLife = true;

                        aliveCheck = true;

                    },5000)

                }

            }

        }

    }

*//    removes a life on hit*

   function removeLife(){

*// checks to see if the player is dead*

        counter ++;

*// adds the dead animation when the player loses all the lives*

        if (counter == hearts ){

            aliveCheck = false;

            player.classList.add('dead')

            liveIcon.lastElementChild.remove();

            setTimeout(() => {

*// adds game over message when dead*

                gameOver();

            }, 1500);

        }

*// if collision it adds the hit class  and removes a life*

        else{

            player.classList.add('hit')

            if (liveIcon.lastElementChild) {

                liveIcon.lastElementChild.remove();

            }

*// makes it so the player cant move for 1.5 seconds after getting hit*

            aliveCheck =false;

            setTimeout(() => {

                aliveCheck = true;

                player.classList.remove('hit')

            }, 1500);

    }

   }

*//    player viarables*

    const player = document.querySelector('#player');

    const playerMouth = player.querySelector('.mouth');

    let playerTop = 0;

    let playerLeft = 0;

*// calls the powerup function*

    powerups();

    let Maininterval = setInterval(function() {

        let postion = player.getBoundingClientRect();

        enemyCheck();

        pointCheck();

*//movement controls*

*//down*

        if((downPressed ) )  {

*// finds the next postion*

                let new\_bottom = postion.bottom + 1;

                let btml =document.elementFromPoint(postion.left, new\_bottom);

                let btmr =document.elementFromPoint(postion.right, new\_bottom);

*// only moves if the next square isnt a wall*

                if(btml && btmr &&(btml.classList.contains('wall') == false && btmr.classList.contains('wall') == false) && aliveCheck){

                    playerTop++;

                    player.style.top = playerTop + 'px';

                }

                else if(btml& btmr &(btml.classList.contains('wall') == true && btmr.classList.contains('wall') == true)){

                    downPressed =false;

                }

                if (playerMouth){

                    playerMouth.classList = 'down';

                }

            }

*//up*

        else if((upPressed  ) ) {

            let newTop =postion.top -1;

            let topL = document.elementFromPoint(postion.left, newTop);

            let topR = document.elementFromPoint(postion.right, newTop);

            if(topL &&  topR &&(topL.classList.contains('wall') == false && topR.classList.contains('wall') == false) && aliveCheck) {

                playerTop--;

                player.style.top = playerTop + 'px';

            }

            else if(topL && topR &&(topL.classList.contains('wall') == true && topR.classList.contains('wall') == true)){

                upPressed =false;

            }

            if (playerMouth){

                playerMouth.classList = 'up';

            }

        }

*//left*

        else if((leftPressed ) ) {

            let newLeft = postion.left -1;

            let leftT = document.elementFromPoint(newLeft,postion.top )

            let leftB = document.elementFromPoint(newLeft,postion.bottom)

            if(leftT && leftB &&(leftT.classList.contains('wall') == false && leftB.classList.contains('wall') == false )&& aliveCheck){

                playerLeft--;

                player.style.left = playerLeft + 'px';

            }

            else if(leftT && leftB &&(leftT.classList.contains('wall') == true && leftB.classList.contains('wall') == true)){

                leftPressed =false;

            }

            if (playerMouth){

                playerMouth.classList = 'left';

            }

        }

*//right*

        else if((rightPressed ) ) {

            let newRight = postion.right +1;

            let rightT = document.elementFromPoint(newRight , postion.top )

            let rightB = document.elementFromPoint(newRight,postion.bottom )

            if(rightT &&rightB &&(rightT.classList.contains('wall') == false && rightB.classList.contains('wall') == false )&& aliveCheck){

                playerLeft++;

                player.style.left = playerLeft + 'px';

                }

            else if(rightT&rightB&(rightT.classList.contains('wall') == true && rightB.classList.contains('wall') == true)){

                    rightPressed =false;

                }

            if (playerMouth){

                    playerMouth.classList = 'right';

                }

            }

    }, 10);

*// start button*

        let start = document.querySelector('.menu')

*// if the start div is clicked or if a child of the start div is clicked it will dissapear*

        if (start.contains(event.target)) {

            start.style.display = 'none';

*//arrow buttons functions*

*// makes the enemies start moving once the game has started*

            enemyMove();

*//down arrow*

            function downArrow() {

                downPressed =true;

                rightPressed=false;

                upPressed=false;

                leftPressed=false;

            }

*// upArrow*

            function upArrow() {

                upPressed =true;

                downPressed=false;

                rightPressed=false;

                leftPressed=false;

            }

*// left arrow*

            function leftArrow() {

                leftPressed =true;

                downPressed=false;

                upPressed=false;

                rightPressed=false;

            }

*// right arrow*

            function rightArrow() {

                rightPressed =true;

                downPressed=false;

                upPressed=false;

                leftPressed=false;

            }

*//Player movement*

            function keyUp(event) {

*// only runs if the player starts the game*

                if (event.key === 'ArrowUp') {

                    upPressed = false;

                } else if (event.key === 'ArrowDown') {

                    downPressed = false;

                } else if (event.key === 'ArrowLeft') {

                    leftPressed = false;

                } else if (event.key === 'ArrowRight') {

                    rightPressed = false;

                }

            }

            function keyDown(event) {

                if (event.key === 'ArrowUp') {

                    upPressed = true;

                } else if (event.key === 'ArrowDown') {

                    downPressed = true;

                } else if (event.key === 'ArrowLeft') {

                    leftPressed = true;

                } else if (event.key === 'ArrowRight') {

                    rightPressed = true;

                }

            }

*// interval that update movement and does collision detection*

*// events listeners*

            document.addEventListener('keydown', keyDown);

            document.addEventListener('keyup', keyUp);

            document.getElementById('dbttn').addEventListener('click', downArrow);

            document.getElementById('ubttn').addEventListener('click', upArrow);

            document.getElementById('lbttn').addEventListener('click', leftArrow);

            document.getElementById('rbttn').addEventListener('click', rightArrow);

        }

    }

let start = document.querySelector('.menu')

start.addEventListener('click', play);

*//this js file is responsible for the maze logic*

*// maze gloabl variables*

let walls = 6;

let enemyCount = 1;

let height =8;

let width = 8;

let maze = [];

let powerupEnable =1;

let x, y;

*// generates the new maze*

function clearMaze() {

    let blockContents = document.querySelectorAll('.block');

    blockContents.forEach((block) => {

        block.remove();

    });

    maze=[];

    clearEnemyMovement();

*//makes the level more dificult*

    mazeDifficulty();

*// creates the maze after deleting it*

    randomMaze(height,width,maze)

    widthIncrease(width);

    MazeAssignment();

*// allows the enemy to move*

    enemyMoveEnable=true;

    enemyMove();

*// resets the score*

    num =0;

*//powerups*

    powerups();

    powerupEnable =1;

}

*// makes the enemies stop moving  when level is completed*

function clearEnemyMovement(){

    clearInterval(enemyMovementInterval)

    clearInterval(enemySecondInterval)

}

*// assigns the maze array with its class*

const MazeAssignment =() =>{

    for (let y of maze) {

        for (let x of y) {

            block = document.createElement('div');

            block.classList.add('block');

            switch (x) {

                case 1:

                    block.classList.add('wall');

                    break;

                case 2:

                    block.id = 'player';

                    let mouth = document.createElement('div');

                    mouth.classList.add('mouth');

                    block.appendChild(mouth);

                    break;

                case 3:

                    block.classList.add('enemy');

                    break;

                default:

                    block.classList.add('point');

                    block.style.height = '1vh';

                    block.style.width = '1vh';

            }

            main.appendChild(block);

        }

    }

}

*// generates the random maze*

function randomMaze(height, width,maze)  {

*// intialise the array with all points*

    for (let i = 0; i < height; i++) {

        maze.push([])

        for (let j = 0; j < width; j++) {

            maze[i].push(0)

        }

    }

*// places the walls on the outside*

    for (let i = 0; i < height; i++) {

        maze[i][0] = 1

        maze[i][width - 1] = 1

    }

    for (let i =0; i< width; i++) {

        maze[0][i] = 1

        maze[height - 1][i] = 1

    }

*// player postion*

    const numOfRows = maze.length;

    const rowLength =maze[0].length;

    maze[1][1]=2;

*// enemy postion*

    for (let i =0; i < enemyCount; i++) {

        const mazearray = Math.floor(Math.random() \* numOfRows)

        const rowArray = Math.floor(Math.random() \* rowLength)

        if (maze[mazearray][rowArray] === 0) {

            maze[mazearray][rowArray] = 3

        } else {

            i--

        }

    }

*// walls inside maze not border  postion*

    for (let i =0; i < walls; i++) {

        const mazearray = Math.floor(Math.random() \* numOfRows)

        const rowArray = Math.floor(Math.random() \* rowLength)

        if (maze[mazearray][rowArray] === 0  &&

*// limits where the walls can spawn to prevent an unplayable game*

            !(mazearray+1 < numOfRows && rowArray+1 < rowLength && maze[mazearray+1][rowArray+1] == 1) &&

            !(mazearray-1 >= 0 && rowArray+1 < rowLength && maze[mazearray-1][rowArray+1] == 1) &&

            !(mazearray+1 < numOfRows && rowArray-1 >= 0 && maze[mazearray+1][rowArray-1] == 1) &&

            !(mazearray-1 >= 0 && rowArray-1 >= 0 && maze[mazearray-1][rowArray-1] == 1)) {

            maze[mazearray][rowArray] = 1

        } else {

            i--

        }

    }

    return maze

}

*// makes the maze more difficult*

const mazeDifficulty = () =>{

    enemyCount  ++;

    height ++;

    widthIncrease(++width);

    walls +=3;

}

*// changes the width depending on what the width variable is set to*

*// increase the width*

function widthIncrease(width){

    let mainWidth = document.querySelector('main');

    mainWidth.style.gridTemplateColumns = `repeat(${width}, 1fr)`;

}

*// makes sure the width increase loads in*

document.addEventListener('DOMContentLoaded', (event) => {

    widthIncrease(width);

});

function powerups(){

    for (let i = 0; i < height; i++) {

        for (let j = 0; j < width; j++) {

            if (maze[i][j] === 0 && powerupEnable === 1) {

                    let randomTime = Math.floor(Math.random() \* (1000 - 3000 + 1)) + 20000;

                    powerupEnable = 0;

*//need to find a way to get the point to be added  when the class is being added to non point*

                    setTimeout(() => {

                        let points = document.querySelectorAll('.point');

                        pointSelection(points.length);

*// adds the extra life class when collected*

                            points[x].classList.add('extraLife')

                            console.log(points[x])

                            powerupEnable=2;

                        }, randomTime);

                }

            }

    }

}

*//gets a random point on the maze*

function pointSelection(pointsArr){

*// selects a random point*

    x = Math.floor(Math.random() \* pointsArr);

};

*// adds the life*

function addLife (){

    let extraLife = document.createElement('li');

    liveIcon.appendChild(extraLife);

    hearts++;

}

*//this js file is responsible for the enemy logic*

*// enemy global variables*

let enemyMoveEnable =true;

let enemyMovementInterval;

let enemySecondInterval;

*//getting alot of errors when enemy is moving after clear maybe define that function in enemy move*

function enemyMove(){

        let enemies = document.querySelectorAll('.enemy');

        enemyMovementInterval= setInterval(function enemyMovement(){

            enemies.forEach(function enemyMovements(enemy){

            let enemyTop = parseInt(enemy.style.top) || 0;

            let enemyLeft = parseInt(enemy.style.left) || 0;

            let randnum = Math.floor(Math.random() \* 4 + 1);

            enemySecondInterval = enemy.interval;

            if (enemySecondInterval){

                clearInterval(enemySecondInterval);

            }

            enemy.interval = setInterval(function(){

*// gets the enemy postion*

                    let enemyPostion = enemy.getBoundingClientRect();

*// gets a randnum and that choses direction*

                    if (randnum === 1 && enemyMoveEnable){

*//bottom*

                        let newBottom = enemyPostion.bottom + 1;

                        let btmL = document.elementFromPoint(enemyPostion.left, newBottom);

                        let btmR = document.elementFromPoint(enemyPostion.right , newBottom);

                        if (btmL && btmR&&(btmL.classList.contains('wall') == false && btmR.classList.contains('wall') == false)) {

                            enemyTop++;

                            enemy.style.top = enemyTop + 'px';

                        }

                    }

                    else if(randnum === 2 && enemyMoveEnable){

*//top*

                        let newTop = enemyPostion.top - 1;

                        let topL = document.elementFromPoint(enemyPostion.left, newTop);

                        let topR = document.elementFromPoint(enemyPostion.right , newTop);

                        if (topL&& topR &&(topL.classList.contains('wall') == false && topR.classList.contains('wall') == false)) {

                            enemyTop--;

                            enemy.style.top = enemyTop + 'px';

                        }

                    }

                    else if(randnum === 3 && enemyMoveEnable){

*//left*

                        let newLeft = enemyPostion.left - 1;

                        let leftT = document.elementFromPoint(newLeft, enemyPostion.top);

                        let leftB = document.elementFromPoint(newLeft, enemyPostion.bottom);

                        if (leftT&& leftB&&(leftT.classList.contains('wall') == false && leftB.classList.contains('wall') == false) ){

                            enemyLeft--;

                            enemy.style.left = enemyLeft + 'px';

                        }

                    }

                    else if(randnum === 4 && enemyMoveEnable){

*//right*

                        let newRight = enemyPostion.right + 1;

                        let rightT = document.elementFromPoint(newRight, enemyPostion.top);

                        let rightB = document.elementFromPoint(newRight, enemyPostion.bottom);

                        if (rightT && rightB&&(rightT.classList.contains('wall') == false && rightB.classList.contains('wall') == false)){

                            enemyLeft++;

                            enemy.style.left = enemyLeft + 'px';

                        }

                    }

            },10);

            });

        },1000);

}

*//this js file is responsible for the leaderborad logic*

*// this function creats an object with a userame and score to organise localStorage with the scores*

function leaderBoard() {

*// gets the user to enter a name for the leaderBoard*

    let username = prompt('Enter your name to go on to the leaderboard');

    if(username == null || username.trim == '') {

        leaderBoard();

        return;

    }

*// checks if the username is already in the localStorage*

    username = username.trim();

    if (sameUsername(username)==false) {

        leaderBoard();

        return;

    }

*// checks if the username isnt empty*

    if (username != '' ) {

        let points = {};

*// Get all scores from localStorage*

        for (let i = 0; i < localStorage.length; i++) {

            let key = localStorage.key(i);

            let value = localStorage.getItem(key);

            let lastDot = value.lastIndexOf('.');

            let score = parseInt(value.substring(lastDot + 1));

            points[key] = score;

        }

*// Add the current user's score*

        points[username] = score;

*// Convert the points object to an array of objects*

        let scores = Object.keys(points).map(key => ({ username: key, score: points[key] }));

*// Sort the scores in descending order*

        scores.sort((a, b) => b.score - a.score);

*// If there are more than 5 scores, remove the ones with the lowest scores*

        for (let i = 0; i < scores.length; i++) {

            localStorage.setItem(scores[i].username, scores[i].username + '......' + scores[i].score);

        }

    }

    else{

        leaderBoard();

    }

    }

*// this functions displays the score at the start of each round*

    function leaderboardAlways() {

        const scoresElement = document.querySelector('.leaderboard ol');

*// Get all scores from localStorage*

        let scoresList = [];

        for (let i = 0; i < localStorage.length; i++) {

            let key = localStorage.key(i);

            let value = localStorage.getItem(key);

            let lastDot = value.lastIndexOf('.');

            let score = parseInt(value.substring(lastDot + 1));

            scoresList.push({ username: key, score: score });

        }

*// Sort the scores in descending order*

        scoresList.sort((a, b) => b.score - a.score);

        if (scoresList.length > 5){

            scoresList.length = 5

        }

*// Display the scores*

        for (let i = 0; i < scoresList.length; i++) {

            let listItem = document.createElement('li');

            listItem.textContent = scoresList[i].username + '......' + scoresList[i].score;

            scoresElement.appendChild(listItem);

        }

    }

*//function for checking if username is the same*

    function sameUsername(username){

        for (let i = 0; i < localStorage.length; i++) {

            console.log(localStorage.length )

            let key = localStorage.key(i);

            console.log(localStorage.key(i));

            if (key === username){

                return false;

            }

        }

    }