```
// weapons

class Weapon {
    constructor(name, damage, className, image) {
        this.name = name;
        this.damage = damage;
        this.className = className;
        this.image = image;
    }
}

var quarter = new Weapon("quarter note", 10, "weapon-1", "quarter.png");
var eighth = new Weapon("eighth note", 5, "weapon-2", "eighth.png");
var half = new Weapon("half note", 20, "weapon-3", "half.png");
var whole = new Weapon("whole note", 40, "weapon-4", "whole.png");
var keyblack = new Weapon("keyblack", 15, "block", "black.png");


// players
const player1 = {
    position: {
        // x: 0,
        //y: 0
    },
    //health: 100,
    currentWeapon: quarter,
    isDefending: false
};

const player2 = {
    position: {
        //x: 0,
        //y: 0
    },
    //health: 100,
    currentWeapon: quarter,
    isDefending: false
};

/*========================================== Build The Game
========================================== */
// Generate random numbers
const generateRandomNum = () => Math.floor(Math.random() * 10);

// Generate grid with blocks
for (let i = 0; i < 10; i++) {
    for (let j = 0; j < 10; j++) {
        $('.grid-container').append('<div class="grid-item" data-y=' + i + ' data-x=' + j +
        '></div>');

        //$('#boardContainer').append('<div class="canvas"</div>');
    }
```

```
}

// Iterates different elements to display them on the board
function generate(func, times) {
    for (let i = 0; i < Number(times); i++) {
        func();
    }
}
// This functions helps blocks, weapons and players find an available aquare in the board
function placeElements(className) {
    const random_x = generateRandomNum();
    const random_y = generateRandomNum();
    $('.grid-item').each(function () {
        const element = $(this);
        if (this.dataset['x'] == random_x && this.dataset['y'] == random_y) {
            if (!(this.classList.contains("unavailable"))) {
                element.addClass(className);
                element.addClass("unavailable");
                // updates the position values to the player objects
                if (className === "player-1") {
                    player1.position.x = this.dataset['x'];
                    player1.position.y = this.dataset['y'];
                } else if (className === "player-2") {
                    player2.position.x = this.dataset['x'];
                    player2.position.y = this.dataset['y'];
                } else if (className === "weapon-1" ||
                    className === "weapon-2" ||
                    className === "weapon-3" ||
                    className === "weapon-4") {
                    element.addClass("weapon");
                }
                if (playerEncounter()) {
                    console.log("Early Encounter");
                    playerReset(className);
                    placeElements(className);
                }
            } else {
                // Function calls itself recursively to find an open space
                placeElements(className);
            }
        }
    });
}


function generateGame() {
    //reset();
    // Anonymous functions
    generate(function () {
        placeElements("block");
    }, 20);
    generate(function () {
```

```
        placeElements("weapon-1")
    }, 1);
    generate(function () {
        placeElements("weapon-2");
    }, 1);
    generate(function () {
        placeElements("weapon-3");
    }, 1);
    generate(function () {
        placeElements("weapon-4");
    }, 1);
    generate(function () {
        placeElements("keyblack");
    }, 8);
    generate(function () {
        placeElements("player-1");
    }, 1);
    generate(function () {
        placeElements("player-2");
    }, 1);
    movePlayer(player1);
    movePlayer(player2);
    pathHighlight();
    weaponDisplay(player1);
    weaponDisplay(player2);

}


function playerOneHighlight() {
    document.getElementById('playerOneTurn').style.color = "green";
    document.getElementById('playerTwoTurn').style.color = "white";
}

function playerTwoHighlight() {
    document.getElementById('playerOneTurn').style.color = "white";
    document.getElementById('playerTwoTurn').style.color = "green";

}


/*================================== Player Movements
================================== */
// Variable to check movements
let playerTurn = true;
//document.getElementById('playerOne').style.color = "green";

function pathHighlight() {
    //if (!fightMode) {
    if (playerTurn) {
        possiblePath(player1);
        $('.player-1').click(function () {
```

```
            $('.possible').css("background-color", "rgba(0, 255, 0, 0.6)");
        });


        playerOneHighlight();
    } else {
        possiblePath(player2);
        $('.player-2').click(function () {
            $('.possible').css("background-color", "rgba(0, 255, 0, 0.6)");
        });
        playerTwoHighlight();
    }

}

function movePlayer(player) {
    //fightMode = false;

    $('.grid-item').click(function () {
        pathHighlight();
        const element = $(this);
        const block = this;
        // Make sure is within distance
        if (element.hasClass("possible")) {

            if (player === player2) {
                playerTwoHighlight();

                $('.player-2').click(function () {
                    $('.possible').css("background-color", "rgba(0, 255, 0, 0.6)");
                });

                document.getElementById("rest").style.display = "none";
                if (!playerTurn) {
                    playerOneHighlight();

                    weaponChecker(block, player);
                    handleWeapon(element, player);
                    playerReset("player-2");
                    element.addClass("player-2");
                    handleFight();
                    playerTurn = !playerTurn;
                }
            }


            if (player === player1) {
                playerOneHighlight();
                $('.player-1').click(function () {
                    $('.possible').css("background-color", "rgba(0, 255, 0, 0.6)");
                });

```

```
            document.getElementById("rest2").style.display = "none";
            if (playerTurn) {

                weaponChecker(block, player);
                handleWeapon(element, player);
                playerReset("player-1");
                element.addClass("player-1");
                handleFight();
                playerTurn = !playerTurn;

            }
        }

    }

});
}

function getPlayerPosition() {
    $('.grid-item').each(function () {
        const element = $(this);
        // I take the coordinates of the
        if (element.hasClass("player-1")) {
            player1.position.x = this.dataset['x'];
            player1.position.y = this.dataset['y'];
        }
        if (element.hasClass("player-2")) {
            player2.position.x = this.dataset['x'];
            player2.position.y = this.dataset['y'];
        }
    });
}

/*================================== Handle Weapons
================================== */
function playerReset(player) {
    $('.grid-item').each(function () {
        const element = $(this);
        element.removeClass(player);
        element.removeClass("possible");
        $('#aKey').attr("disabled", false);
        $('#dKey').attr("disabled", false);
        $('#left').attr("disabled", false);
        $('#right').attr("disabled", false);

    });
}

function squareOccupied(element) {
    return (
        element.hasClass("block") ||
        element.hasClass("player-1") ||
```

```
        element.hasClass("player-2")
    );
}

function possiblePath(player) {
    $('.grid-item').each(function () {
        const element = $(this);
        const block = this;
        if (isInDistance(player, block) && !squareOccupied(element)) {
            element.addClass("possible");
        }
    });
    $('.grid-item').each(function () {
        const element = $(this);
        const block = this;
        // value larger with larger X
        if (isInDistance(player, block) && (block.dataset['x'] > player.position.x)) {


            //removing inDistance allows for diagonal movement
            //if((block.dataset['x'] > player.position.x)){
            if (squareOccupied(element)) {
                occupiedObject = this;
                $('.possible').each(function () {
                    const element = $(this);
                    const block = this;

                    if (block.dataset['x'] > occupiedObject.dataset['x']) {
                        // console.log("block x is greater than player x");
                        element.removeClass("possible");
                    }
                });
            }
        }
        // value with lower X
        if (isInDistance(player, block) && (block.dataset['x'] < player.position.x)) {

            if (squareOccupied(element)) {
                occupiedObject = this;
                $('.possible').each(function () {
                    const element = $(this);
                    const block = this;

                    if (block.dataset['x'] < occupiedObject.dataset['x']) {
                        //console.log("block x is less than player x");
                        element.removeClass("possible");
                    }
                });
            }
        }
        // value with higher y
        if (isInDistance(player, block) && (block.dataset['y'] > player.position.y)) {
```

```
309
310             if (squareOccupied(element)) {
311                 occupiedObject = this;
312                 $('.possible').each(function () {
313                     const element = $(this);
314                     const block = this;
315
316                     if (block.dataset['y'] > occupiedObject.dataset['y']) {
317                         //console.log("block y is greater than player y");
318                         element.removeClass("possible");
319                     }
320                 });
321             }
322         }
323         // value with lower y
324         if (isInDistance(player, block) && (block.dataset['y'] < player.position.y)) {
325
326             if (squareOccupied(element)) {
327                 occupiedObject = this;
328                 $('.possible').each(function () {
329                     const element = $(this);
330                     const block = this;
331
332                     if (block.dataset['y'] < occupiedObject.dataset['y']) {
333                         //  console.log("block y is less than player y");
334                         //alert("up");
335                         element.removeClass("possible");
336                     }
337                 })
338             }
339         }
340     })
341
342
343     $('.grid-item').each(function () {
344         $('.possible').css("background-color", "");
345         //document.getElementById('playerTwo').style.color = "white";
346     });
347
348
349 }
350 // I need to reference all the elements that are "inDistance"
351 // To determine if they have block or player class, if they do
352 // movement is not possible
353 function isInDistance(player, block) {
354     const firstCondition = (Math.abs(block.dataset['x'] - player.position.x) < 4) &&
355         (block.dataset['y'] === player.position.y);
356     const secondCondition = (Math.abs(block.dataset['y'] - player.position.y) < 4) &&
357         (block.dataset['x'] === player.position.x);
358     return (firstCondition || secondCondition);
359     // $('.grid-item').each(pathHighlight);
360 }
```

```
361
362 // Check if there are any weapons in the path
363 function weaponChecker(block, player) {
364     checkSmallerX(block, player);
365     checkSmallerY(block, player);
366     checkLargerX(block, player);
367     checkLargerY(block, player);
368 }
369
370 // Check if there is a weapon left of the player
371 function checkSmallerX(block, player) {
372     if (block.dataset['x'] < player.position.x) {
373         $('.possible').each(function () {
374             const element = $(this);
375             const innerBlock = this;
376             if ((innerBlock.dataset['x'] < player.position.x) &&
377                 (innerBlock.dataset['y'] == player.position.y) &&
378                 innerBlock.dataset['x'] > block.dataset['x']) {
379                 if (element.hasClass("weapon") ||
380
381                     element.hasClass("keyblack")) {
382                     weaponChange(element, player);;
383                 }
384             }
385         })
386     }
387 }
388
389 // Check if there is a weapon right of the player
390 function checkLargerX(block, player) {
391     if (block.dataset['x'] > player.position.x) {
392         $('.possible').each(function () {
393             const element = $(this);
394             const innerBlock = this;
395             if ((innerBlock.dataset['x'] > player.position.x) &&
396                 (innerBlock.dataset['y'] == player.position.y) &&
397                 (innerBlock.dataset['x'] < block.dataset['x'])) {
398                 if (element.hasClass("weapon") ||
399
400                     element.hasClass("keyblack")) {
401                     weaponChange(element, player);
402                 }
403             }
404         })
405     }
406 }
407
408 // Check if there is a weapon under the player
409 function checkSmallerY(block, player) {
410     if (block.dataset['y'] < player.position.y) {
411         $('.possible').each(function () {
412             const element = $(this);
```

```
            const innerBlock = this;
            if ((innerBlock.dataset['y'] < player.position.y) &&
                (innerBlock.dataset['x'] == player.position.x) &&
                innerBlock.dataset['y'] > block.dataset['y']) {
                if (element.hasClass("weapon") ||

                    element.hasClass("keyblack")) {
                    weaponChange(element, player);
                }
            }
        })
    }
}

// Check if there is a weapon over the player
function checkLargerY(block, player) {
    if (block.dataset['y'] > player.position.y) {
        $('.possible').each(function () {
            const element = $(this);
            const innerBlock = this;
            if ((innerBlock.dataset['y'] > player.position.y) &&
                (innerBlock.dataset['x'] == player.position.x) &&
                innerBlock.dataset['y'] < block.dataset['y']) {
                if (element.hasClass("weapon") ||

                    element.hasClass("keyblack")) {
                    weaponChange(element, player);
                }
            }
        })
    }
}

var eighthNote = document.getElementById("eighth");
var quarterNote = document.getElementById("quarter");
var halfNote = document.getElementById("half");
var wholeNote = document.getElementById("whole");
var eNote = document.getElementById("enote");
var qNote = document.getElementById("qnote");
var hNote = document.getElementById("hnote");
var wNote = document.getElementById("wnote");


// If there is a weapon, handle it by removing the class and replacing it
// with the current one of the player
function weaponChange(element, player) {
    let playerWeapon = player.currentWeapon;
    let damage = this.damage;
    if (element.hasClass("weapon-1")) {
        element.removeClass("weapon-1");
        element.addClass(playerWeapon.className);
        player.currentWeapon = quarter;
```

```
        weaponDisplay(player);
        quarter.damage = 10;

    } else if (element.hasClass("weapon-2")) {
        element.removeClass("weapon-2");
        element.addClass(playerWeapon.className);
        player.currentWeapon = eighth;
        weaponDisplay(player);
        eighth.damage = 5;

    } else if (element.hasClass("weapon-3")) {
        element.removeClass("weapon-3");
        element.addClass(playerWeapon.className);
        player.currentWeapon = half;
        weaponDisplay(player);
        half.damage = 20;

    } else if (element.hasClass("weapon-4")) {
        element.removeClass("weapon-4");
        element.addClass(playerWeapon.className);
        player.currentWeapon = whole;
        weaponDisplay(player);
        whole.damage = 40;

    } else if (element.hasClass("keyblack")) {
        //keyblack.damage = 15;
        if (player === player1) {
            document.getElementById("rest").style.display = "block"
            document.getElementById("myProgress").value -= 15;
            progressMeter();
            meterHealth();
        } else if (player === player2) {
            document.getElementById("rest2").style.display = "block"
            document.getElementById("myPro").value -= 15;
            proMeter();
            meterHealth();
        }
        element.removeClass("keyblack");

    }

}


function weaponDisplay(player) {
    if (player === player1) {

        if (player.currentWeapon == eighth) {
            eighthNote.style.display = "block";
        } else {
            eighthNote.style.display = "none";
        }
```

```
517
518         if (player.currentWeapon == quarter) {
519             quarterNote.style.display = "block";
520         } else {
521             quarterNote.style.display = "none";
522         }
523         if (player.currentWeapon == half) {
524             halfNote.style.display = "block";
525         } else {
526             halfNote.style.display = "none";
527         }
528         if (player.currentWeapon == whole) {
529             wholeNote.style.display = "block";
530         } else {
531             wholeNote.style.display = "none";
532         }
533
534
535     }
536     if (player === player2) {
537         if (player.currentWeapon == eighth) {
538             eNote.style.display = "block";
539         } else {
540             eNote.style.display = "none";
541         }
542
543         if (player.currentWeapon == quarter) {
544             qNote.style.display = "block";
545         } else {
546             qNote.style.display = "none";
547         }
548         if (player.currentWeapon == half) {
549             hNote.style.display = "block";
550         } else {
551             hNote.style.display = "none";
552         }
553         if (player.currentWeapon == whole) {
554             wNote.style.display = "block";
555         } else {
556             wNote.style.display = "none";
557         }
558
559     }
560 }
561
562 /*============================== Player Encounters  and fighting
    ============================== */
563 //let fightMode = false;
564
565 function handleWeapon(element, player) {
566     weaponChange(element, player);
567 }
```

```
// What happens when players encounter each other
function playerEncounter() {
    getPlayerPosition();
    const xPosition = Math.abs(Number(player1.position.x) - Number(player2.position.x));
    const yPosition = Math.abs(Number(player2.position.y) - Number(player1.position.y));
    return (((xPosition == 0) && (yPosition == 1)) ||
        ((yPosition == 0) && (xPosition == 1))
    )

}


// Logic to take care of the turns
function handleFight() {
    if (playerEncounter()) {
        //fightMode = true;
        //movePlayer(player1) = false;
        //movePlayer(player2) = false;

        $("#toggleButton").css("color", "green");
        $("#WASDmodal").css("display", "block");
        $("#arrowKeysmodal").css("display", "block");
        $("#block").css("display", "none");
        $("#block2").css("display", "none");

        const oneAttacks = document.getElementById('aKey');
        const twoAttacks = document.getElementById('left');
        const oneDefends = document.getElementById('dKey');
        const twoDefends = document.getElementById('right');

        if (!playerTurn) {

            //}
            $('#aKey').attr("disabled", false).css("backgroundColor", "rgba(255, 19, 24,
            0.55)");
            $('#dKey').attr("disabled", false).css("backgroundColor", "rgba(255, 19, 24,
            0.55)");
            $('#left').css("color", "");
            $('#right').css("color", "");
            //$('#left').attr("disabled", true);
            //$('#right').attr("disabled", true);
        }
        oneAttacks.onclick = function () {
            playerReset(player1);
            //$('#aKey').on('click', function(){
            meterHealth();
            proMeter();
            proBar.value -= player1.currentWeapon.damage;
            $('#aKey').attr("disabled", true).css("backgroundColor", "");
            $('#dKey').attr("disabled", true).css("backgroundColor", "");

            playerTwoHighlight();
```

```
618         $('#left').css("backgroundColor", "rgba(0, 118, 255, 0.57)");
619         $('#right').css("backgroundColor", "rgba(0, 118, 255, 0.57)");
620     }
621
622     //$('#aKey').attr("disabled", false);
623     oneDefends.onclick = function () {
624         playerReset(player1);
625         //$('#dKey').on('click', function(){
626         //defendFunc();
627         meterHealth();
628         //player1.isDefending = true;
629         progressMeter();
630
631         backBar.value -= player2.currentWeapon.damage / 2;
632         $('#dKey').attr("disabled", true).css("backgroundColor", "");
633         $('#aKey').attr("disabled", true).css("backgroundColor", "");
634
635         playerTwoHighlight();
636         $('#left').css("backgroundColor", "rgba(0, 118, 255, 0.57)");
637         $('#right').css("backgroundColor", "rgba(0, 118, 255, 0.57)");
638         //}
639     }
640     //$('#dKey').attr("disabled", false);
641     if (playerTurn) {
642         //if(movePlayer(player2)){
643         //fightMode = false;
644
645         //}
646         $('#left').attr("disabled", false).css("backgroundColor", "rgba(0, 118, 255, 0.57)"
            );
647         $('#right').attr("disabled", false).css("backgroundColor", "rgba(0, 118, 255,
            0.57)");
648         $('#aKey').css("color", "");
649         $('#dKey').css("color", "");
650         //$('#aKey').attr("disabled", true);
651         //$('#dKey').attr("disabled", true);
652     }
653     twoAttacks.onclick = function () {
654         playerReset(player2);
655         //$('#left').on('click', function(){
656         meterHealth();
657         progressMeter();
658         backBar.value -= player2.currentWeapon.damage;
659         $('#left').attr("disabled", true).css("backgroundColor", "");
660         $('#right').attr("disabled", true).css("backgroundColor", "");
661
662         playerOneHighlight();
663         $('#aKey').css("backgroundColor", "rgba(255, 19, 24, 0.55)");
664         $('#dKey').css("backgroundColor", "rgba(255, 19, 24, 0.55)");
665     }
666     //$('#left').attr("disabled", false);
667     twoDefends.onclick = function () {
```

```
            playerReset(player2);
            //$('#right').on('click', function(){
            //defendFunc();
            meterHealth();
            //player2.isDefending = true;
            proMeter();
            proBar.value -= player1.currentWeapon.damage / 2;
            $('#right').attr("disabled", true).css("backgroundColor", "");
            $('#left').attr("disabled", true).css("backgroundColor", "");

            playerOneHighlight();
            $('#aKey').css("backgroundColor", "rgba(255, 19, 24, 0.55)");
            $('#dKey').css("backgroundColor", "rgba(255, 19, 24, 0.55)");
        }
        //getPlayerPosition();


        //$('#right').attr("disabled", false);
    }
    //*/
    //playerReset(player1);
    //playerReset(player2);
    else if (!playerEncounter()) {
        $("#toggleButton").css("color", "");
        $("#WASDmodal").css("display", "none");
        $("#arrowKeysmodal").css("display", "none");

        $("#block").css("display", "block");
        $("#block2").css("display", "block");
        //fightMode = false;

        $('#right').attr("disabled", true);
        $('#left').attr("disabled", true);
        $('#dKey').attr("disabled", true);
        $('#aKey').attr("disabled", true);
    }

}


// Logic to take care of the attacks
function meterHealth() {

    handleFight();

    if (backBar.value <= 0) {
        backBar.value = 0;

        window.open('beethoven.html', "_self");
        //beetAudio();


```

```
720     }
721     //if (player2.health <= 0 ) {
722     if (proBar.value <= 0) {
723         proBar.value = 0;
724 
725         window.open('mozart.html', "_self");
726         //ma.play();
727     }
728 }
```