JS\Create PT\Ultra Crush Siblings\index.html

```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">
  <link rel="stylesheet" href="masterCSS.css">
  <script defer type="module" src="./index.js"></script>
  <title>Ultra Crush Siblings</title>
</head>
<body>
  <canvas id="myCanvas"></canvas>
</body>
</html>
```

index.js

```
import { sleep, getRandInt } from "./utility.js";
import * as fighters from "./fighters.js"
import {detectionBox} from "./cpuLogic.js"

/** @type {HTMLCanvasElement} */
const c = myCanvas;

// Get the dimensions of the canvas
const cWidth = c.offsetWidth;
const cHeight = c.offsetHeight;
// Update this to match the CSS width and height of the canvas
c.width = cWidth;
c.height = cHeight;
const ctx = c.getContext("2d");
// Control how often the game updates so there won't be an issue on high refresh rate
monitors
const tps = 60

var curkeys = []
var newkeys = [];

// Sprite sheet of an archer
// Source: https://astrobob.itch.io/arcane-archer
// Accessed: 3/2/23
const PURPLE_ARROW_PATH = "purple_arrow/spritesheet.png"

// Sprite sheet of an warrior
// Source: https://astrobob.itch.io/arcane-archer
// Accessed: 3/3/23
const WARRIOR_PATH = "warrior/spritesheet.png"

var fighterArr = [new fighters.purple_arrow(PURPLE_ARROW_PATH, true), new
fighters.warrior(WARRIOR_PATH, true)]
var fighterArrCpu = [new fighters.purple_arrow(PURPLE_ARROW_PATH, false), new
fighters.warrior(WARRIOR_PATH, false)]

var currFighter = 0
var currCpu = 0

export var player = new fighters.purple_arrow(PURPLE_ARROW_PATH, true)
export var cpu = new fighters.warrior(WARRIOR_PATH, false)

var pLives = 3;
var cLives = 3;

var state = "title"

const bg = new Image()
// Image of a platformer stage
// Source: https://ansimuz.itch.io/bulkhead-walls-environment\
// Accessed: 3/24/23
bg.src = "./assets/bulkhead-wallsx3.png"
```

```
var titleOptionNum = 0
var titleOptionArr = [{
    text: "start",
    state: "char_select",
    arrowOffset: -120,
    height: -45,
},
{
    text: "instructions",
    state: "instructions",
    arrowOffset: -220,
    height: +65,
}]


/**
 * Function that draws text on the canvas.
 *
 * @param {String} text
 * @param {Number} x
 * @param {Number} y
 * @param {Number} size
 * @param {String} color
 * @param {String} font
 * @param {String} xAlign
 * @param {String} yAlign
 */
function drawText(text, x=cWidth/2, y=cHeight/2-160, size=100, color="white",
font="ssbu", xAlign="center", yAlign="middle") {
    ctx.fillStyle = color
    ctx.textAlign = xAlign
    ctx.textBaseline = yAlign
    ctx.font = `${size}px ${font}`
    text = formatText(text)
    ctx.fillText(text, x, y)
}

/**
 * A function that formats the passed through string.
 *
 * @param {String} text
 * @returns formatted string
 */
function formatText(text) {
    let outputString = ""
    for (let i = 0; i < text.length; i++) {
        if(text[i] == " ") {
            outputString += "  ";
            continue;
        }
        outputString += text[i] + "";
    }
    return outputString;
}
```

```
/**
 * Function that dictates movement of the player character based on user input given
through currkeys array and newkeys array
 */
function checkMovement(){
    if(player.attacking) return
    if(curkeys[65] && !player.atacking) player.moveX(-1)
    else if(curkeys[68] && !player.atacking) player.moveX(1)
    else player.applyFriction()
    if(newkeys[32]) player.jump()
    else if (curkeys[74]) player.shield()
    else if(newkeys[75]) player.ability1()
    else if (newkeys[76]) player.ability2()
    if (!curkeys[74]) player.shielding = false
}

/**
 * Function that updates the game state
 */
function updateGameScreen() {
    checkMovement()
    player.update()
    // cpuLogic.cpuLevel1()
    cpu.update()
    detectionBox.update()
    if (!player.alive && player.lives-1 > 0) {
        fighterArr = [new fighters.purple_arrow(PURPLE_ARROW_PATH, true), new
fighters.warrior(WARRIOR_PATH, true)]
        player = fighterArr[currFighter];
        pLives -= 1;
        player.lives = pLives;
    }
    else if (!player.alive && player.lives =/= 0) {
        player.lives = 0
        state = "game_over"
    }

    if (!cpu.alive && cpu.lives-1 > 0) {
        fighterArrCpu = [new fighters.purple_arrow(PURPLE_ARROW_PATH, false), new
fighters.warrior(WARRIOR_PATH, false)]
        cpu = fighterArrCpu[currCpu]
        cLives -= 1;
        cpu.lives = cLives;
    }
    else if (!cpu.alive && cpu.lives =/= 0) {
        cpu.lives = 0
        state = "game_over"
    }
}

/**
 * Updates the instruction screen
 */
function updateInstructionScreen(){
```

```
    //Return to home screen when ENTER or ESC is pressed (ESC is intuitive so even
though its not in the instructions I made it an option)
    if(!newkeys[13] && !newkeys[27]) return
    state = "title";
}

/**
 * Function that updates the title screen
*/
function updateTitleScreen() {
    //Handles choosing an option on title screen
    if(newkeys[40]) {
        titleOptionNum++
        if(titleOptionNum > titleOptionArr.length-1) titleOptionNum = 0;
    }
    else if(newkeys[38]) {
        titleOptionNum--
        if(titleOptionNum < 0) titleOptionNum = titleOptionArr.length-1;
    }
    else if(newkeys[13]) {
        //Delay so you don't choose an option right when the state is changed
        sleep(20).then(() =>{
            state = titleOptionArr[titleOptionNum].state
        })
    }
}

/**
 * Updates the character select screen
*/
function updateCharacterSelectScreen(){
    //Updates the position and animations of characters
    for (let i = 0; i < fighterArr.length; i++) {
        fighterArr[i].updateSelectMode();
        fighterArr[i].x = cWidth*i/5+90;
        fighterArr[i].y = cHeight-fighterArr[i].height*1.4;
    }
    //Return to title
    if(newkeys[27]) state = "title";
    // Choose character
    else if(newkeys[37] && currFighter > 0) currFighter--;
    else if(newkeys[39] && currFighter < fighterArr.length-1) currFighter++;
    else if(newkeys[13]) {
        player = fighterArr[currFighter];
        player.x = 20
        player.y = 0;
        currCpu  = getRandInt(0, fighterArrCpu.length);
        cpu = fighterArrCpu[currCpu]
        cpu.x = cWidth-player.width-20
        cpu.y = 0;
        cpu.direction = -1
        state = "game"
    }
}
```

```
/**
 * Updates the gameover screen
 */
function updateGameOverScreen(){
    //Return to home screen when ENTER or ESC is pressed (ESC is intuitive so even
though its not in the instructions I made it an option)
    if(!newkeys[13]) return
    pLives = 3;
    cLives = 3;
    fighterArr = [new fighters.purple_arrow(PURPLE_ARROW_PATH, true), new
fighters.warrior(WARRIOR_PATH, true)]
    fighterArrCpu = [new fighters.purple_arrow(PURPLE_ARROW_PATH, false), new
fighters.warrior(WARRIOR_PATH, false)]
    sleep(20).then(() => {
        state = "title";
    })
}

/**
 * Function that draws the game state
*/
function drawGameScreen(){
    ctx.drawImage(bg, 0, 0, cWidth, cHeight)

    player.draw(ctx);
    cpu.draw(ctx);

    let radius = 15

    drawText("Player:", 70, cHeight-30, 20)

    // Stock indicators
    if(player.lives > 0) {
        ctx.beginPath();
        ctx.arc(150, cHeight-30, radius, 0, 2 * Math.PI, false);
        ctx.fillStyle = "rgba(0, 0, 255, .7)"
        ctx.fill();
        ctx.lineWidth = 3;
        ctx.strokeStyle = 'rgba(0, 0, 180, .7)';
        ctx.stroke();
    }

    if(player.lives > 1) {
        ctx.beginPath();
        ctx.arc(200, cHeight-30, radius, 0, 2 * Math.PI, false);
        ctx.fillStyle = "rgba(0, 0, 255, .7)"
        ctx.fill();
        ctx.lineWidth = 3;
        ctx.strokeStyle = 'rgba(0, 0, 180, .7)';
        ctx.stroke();
    }

    if(player.lives > 2) {
        ctx.beginPath();
        ctx.arc(250, cHeight-30, radius, 0, 2 * Math.PI, false);
        ctx.fillStyle = "rgba(0, 0, 255, .7)"
```

```
        ctx.fill();
        ctx.lineWidth = 3;
        ctx.strokeStyle = 'rgba(0, 0, 180, .7)';
        ctx.stroke();
    }

    drawText(":Cputer", cWidth-70, cHeight-30, 20)

    // Stock indicators
    if(cpu.lives > 0) {
        ctx.beginPath();
        ctx.arc(cWidth-150, cHeight-30, radius, 0, 2 * Math.PI, false);
        ctx.fillStyle = "rgba(255, 0, 0, .7)"
        ctx.fill();
        ctx.lineWidth = 3;
        ctx.strokeStyle = 'rgba(180, 0, 0, .7)';
        ctx.stroke();
    }

    if(cpu.lives > 1) {
        ctx.beginPath();
        ctx.arc(cWidth-200, cHeight-30, radius, 0, 2 * Math.PI, false);
        ctx.fillStyle = "rgba(255, 0, 0, .7)"
        ctx.fill();
        ctx.lineWidth = 3;
        ctx.strokeStyle = 'rgba(180, 0, 0, .7)';
        ctx.stroke();
    }

    if(cpu.lives > 2) {
        ctx.beginPath();
        ctx.arc(cWidth-250, cHeight-30, radius, 0, 2 * Math.PI, false);
        ctx.fillStyle = "rgba(255, 0, 0, .7)"
        ctx.fill();
        ctx.lineWidth = 3;
        ctx.strokeStyle = 'rgba(180, 0, 0, .7)';
        ctx.stroke();
    }

    if(!cpu.alive && cpu.lives-1 < 0 ) {
        drawText("yOu win")
    }

    if(!player.alive && player.lives-1 < 0) {
        drawText("yOu lOse")
    }

}

/**
 * Draws the title screen
*/
function drawTitleScreen() {
    ctx.drawImage(bg, 0, 0, cWidth, cHeight);
```

```
    drawText('Ultra Crush', cWidth/2, cHeight/2-250, 60)
    drawText('Siblings', cWidth/2, cHeight/2-150, 60)
    drawText('>', cWidth/2+titleOptionArr[titleOptionNum].arrowOffset,
cHeight/2+titleOptionArr[titleOptionNum].height, 70)
    for (let i = 0; i < titleOptionArr.length; i++) {
        drawText(titleOptionArr[i].text, cWidth/2, cHeight/2+titleOptionArr[i].height,
45)
    }
    drawText('Use arrow keys to move and enter to select', cWidth/2,
cHeight/2+290, 17)
}

/**
 * Draws the instructions screen
*/
function drawInstructionScreen(){
    ctx.drawImage(bg, 0, 0, cWidth, cHeight);
    // Make the background less visible so the instructions have easier readability
    ctx.fillStyle = 'rgba(0, 0, 0, .5)'
    ctx.fillRect(0, 0, cWidth, cHeight)

    // Draw instructions
    drawText('Ultra Crush Siblings', cWidth/2, cHeight/2-200, 70)
    drawText('Attack the opposing fighter and attempt to knock them off the map.',
cWidth/2, cHeight/2-75, 23)
    drawText('Use WASD to move, to use abilities 1 and 2 press K and L.', cWidth/2,
cHeight/2-20, 23)
    drawText('Press SPACE to jump, press it again to double jump.', cWidth/2,
cHeight/2+35, 23)
    drawText('Shield with J to prevent knockback and damage.', cWidth/2, cHeight/2+90,
23)
    drawText('Whoever gets knocked off 3 times loses.', cWidth/2, cHeight/2+145, 23)
    drawText('Good Luck Have Fun.', cWidth/2, cHeight/2+200, 23)
    drawText('Press Enter to go back', cWidth/2, cHeight/2+290, 17)
}

function drawCharacterSelectScreen(){
    let offsetW = 0
    ctx.drawImage(bg, 0, 0, cWidth, cHeight)
    ctx.fillStyle = '#2E514F';
    ctx.fillRect(0, cHeight*53.4/64, cWidth, cHeight)

    //Draw all characters
    for (let i = 0; i < fighterArr.length; i++) {
        fighterArr[i].draw(ctx)
    }
    if (currFighter == 1) offsetW = 10

    drawText('v', fighterArr[currFighter].x + offsetW +
fighterArr[currFighter].width/2, cHeight*52/64, 40, "white", "Arial")
    drawText('Press Enter to choose a character (Press ESC to go back)',
cWidth/2, cHeight/2+293, 15)
}

function drawGameOver() {
    let color = "white"
    drawGameScreen()
```

```
    ctx.fillStyle = 'rgba(0, 0, 0, .3)'
    ctx.fillRect(0, 0, cWidth, cHeight)

    if(player.lives <= 0) {
        drawText("yOu lOse", cWidth/2, cHeight/2-160, 100, "red")
    }
    if(cpu.lives <= 0) {
        drawText("yOu win", cWidth/2, cHeight/2-160, 100, "blue")
    }

    drawText("Press ENTER to return", cWidth/2, cHeight*2/3, 50)
    drawText("to title screen", cWidth/2, cHeight*2/3+60, 50)
}

/**
 * Draws the correct scene depending on the state
 */
function drawFrame() {
    // Clear the canvas each frame
    ctx.clearRect(0, 0, cWidth, cHeight);
    if(state == "title") drawTitleScreen()
    else if (state == "instructions") drawInstructionScreen()
    else if(state == "char_select") drawCharacterSelectScreen()
    else if(state == "game") drawGameScreen()
    else if(state == "game_over") drawGameOver()
    requestAnimationFrame(drawFrame);
}

/**
 * Updates the game depending on the current state
*/
function update() {
    if(state == "title") updateTitleScreen()
    else if (state == "instructions") updateInstructionScreen()
    else if(state == "char_select") updateCharacterSelectScreen()
    else if(state == "game") updateGameScreen()
    else if(state == "game_over") updateGameOverScreen()


    for (let i = 0; i < newkeys.length; i++) {
        newkeys[i] = false;
    }
    setTimeout(() => {
        requestAnimationFrame(update);
    }, 1000 / tps);
}

/**
 * Function that initializes the game
 */
function init() {
    // This segment of code allows me to collect keyboard inputs from the user
    // Source: Teacher
    // Accessed on 2/9/23
```

```
    window.addEventListener('keydown',
    /**
     * Function that takes the event and updates the keys being pressed down
     * @param {Event} e
     */
    function(e){ if(!curkeys[e.keyCode]){
        curkeys[e.keyCode] = true;
        newkeys[e.keyCode] = true;}})
        window.addEventListener('keyup',
        /**
         * Function that takes the event and updates the keys being pressed down
        * @param {Event} e
        */
        function(e){ curkeys[e.keyCode] = false;
            if (e.keyCode == 65 || e.keyCode == 68) {
                player.velocity[0] = 0;
            }
        })


    //Imaging stuff so sprites scale smoothly and maintain their original look
    ctx.imageSmoothingEnabled = false;
    ctx.webkitImageSmoothingEnabled = false;


    // Start the loop for the canvas
    update();
    drawFrame();
}

init();
```

JS\Create PT\Ultra Crush Siblings\cpuLogic.js

```
import {player, cpu} from "./index.js"
import {Timer, getRandInt} from "./utility.js"
import {ground} from "./fighters.js"

// Variables relating cpu logic
var cpuReactionTime = 250
var cpuReactionTimer = new Timer(cpuReactionTime)
var cpuDirection = 0
var cpuBoxOffset = 250

// Detection box for the CPU
export var detectionBox = {
    x: 0,
    y: 0,
    width: cpuBoxOffset,
    height: cpuBoxOffset,
    update: function() {
        this.x = (cpu.x+cpu.width/2)-cpuBoxOffset/2;
        this.y = (cpu.y+cpu.height/2)-cpuBoxOffset/2;
    },
}

/**
 *
 * Function that checks if object 1 and object 2 intersect then returns true if they
do and false if they don't
 *
 * @param {*} obj1
 * @param {*} obj2
 * @returns true or false
 */
function intersects(obj1, obj2 = detectionBox) {
    if (obj1.x < obj2.x + obj2.width && obj1.x + obj1.width > obj2.x && obj1.y <
obj2.y + obj2.height && obj1.y + obj1.height > obj2.y) {
        return true;
    }
    else {
        return false;
    }
}



/**
 * Function that controls the cpu actions
 */
export function cpuLogic() {
    // Make sure the cpu is in a state where it can move and that the player isn't
dead or off the map
    if(!cpu.alive || cpu.hurt || cpu.attacking || player.y+player.height >
myCanvas.offsetHeight-ground || !player.alive) return;
    // Allows the cpu to have a more human reaction time because it becomes insanely
difficult if it reacts in 1/60 of a second
```

```
    if(cpuReactionTimer.interval != 500) {
        cpuReactionTimer.interval = 500;
        cpuReactionTimer.accum = 0;
    }
    // Move towards the player
    if(cpuReactionTimer.isReady()){
        if(player.x < cpu.x){
            cpuDirection = -1
        }
        else if(player.x > cpu.x){
            cpuDirection = 1
        }
        else{
            cpuDirection = 1
        }
        // Check if the player is in range then attacks
        if(intersects(player)){
            let x = getRandInt(0, 4)
            if(x == 1) cpu.ability1()
            else cpu.ability1()
        }
    }

    // Head back towards the map if the cpu is off the map
    if(cpu.x < -cpu.width*2){
        cpuDirection = 1
    }
    else if(cpu.x + cpu.width > myCanvas.offsetWidth+cpu.width*2){
        cpuDirection = -1
    }
    // Jump to get back on the map
    if(cpu.y+cpu.height>myCanvas.offsetHeight-ground+1) cpu.jump()
    // The movements that were set above
    cpu.moveX(cpuDirection)
    cpu.moveX(cpuDirection)
}
```

JS\Create PT\Ultra Crush Siblings\utility.js

```
/**
 * Function that sleeps the inputted ms, you can also toggle a log that says how long
it sleeps and when it finishes
 *
 * @param {Number} ms
 * @param {Boolean} lg
 * @returns resolved promise
 */
export async function sleep(ms, lg=false){
    if(ms ⩽ 0){
        return console.error("Invalid sleep time: " + ms);
    }
    if (lg) console.log("Sleeping " + ms + "ms...")

    return new Promise(resolve ⇒ setTimeout(resolve, ms)).then(() ⇒ {
        if (lg) console.log("done")
    })
}

/**
 * Function that generates a random intger from
 * Source: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference
/Global_Objects/Math/random
 * Accessed: 3/26/23
 *
 * @param {Number} min
 * @param {Number} max
 * @returns the random number
 */
export function getRandInt(min, max) {
    min = Math.ceil(min);
    max = Math.floor(max);
    return Math.floor(Math.random() * (max - min) + min); // The maximum is exclusive
and the minimum is inclusive
}

// Class that checks if the passed through interval has elapsed
// Source: Teacher
// From: utilityClassesAndFunctions.js
// Accessed: 3/28/23
export class Timer{
    constructor(timeIntervalMS, enabled = true){
        this.oldTime = new Date();

        this.interval = timeIntervalMS;

        this.accum = 0;

        this.enabled = enabled;
    }

    /**
```

```
     * Source: Teacher
     * Accessed: 3/28/23
     * From: utilityClassesAndFunctions.js
     * @returns if the timer is ready
     */
    isReady(){
        var curTime = new Date();
        var delta = curTime - this.oldTime;

        this.accum += delta;
        this.oldTime = curTime;
        if (this.accum > this.interval && this.enabled){
            this.accum = 0;
            return true;
        }
        else{
            return false;
        }
    }
}
```

JS\Create PT\Ultra Crush Siblings\fighters.js

```
import {sleep} from "./utility.js"
import {player, cpu} from "./index.js"
import * as cpuLogic from "./cpuLogic.js"

/**
 *
 * @param {*} enemy An object representing an enemy
 * @param {*} range The range of the attack of the object of the attack such as a
projectile
 * @returns
 */
function collides(range, enemy) {
    if (range.x < enemy.x + enemy.width && range.x + range.width > enemy.x && range.y
< enemy.y + enemy.height && range.y + range.height > enemy.y) {
        return true;
    }
    return false;
}


/** @type {HTMLCanvasElement} */
const c = myCanvas;

const cWidth = c.offsetWidth;
const cHeight = c.offsetHeight;

// Path for fighter assets
const path = "./assets/fighters/";

const gravity = 0.4;
export const ground = 120;

// Stagger frame for animation to have a consistent and normalize the speed
const staggerFrame = 5;


const SHIELD = new Image()
// Custom asset made by me in pixlr
// Created On: 3/15/23
SHIELD.src = "./assets/shield.png"


class projectile {
    /**
     * Constructs the projectile object
     *
     * @param {HTMLImageElement} img
     * @param {Number} x
     * @param {Number} y
     * @param {Number} sx
     * @param {Number} direction
```

```
     */
    constructor(img, isPlayer, x=-800, y=500, sx=80, direction=1) {
        this.img = img;
        this.isPlayer = isPlayer
        this.x = x;
        this.y = y;
        this.width = this.img.width
        this.height = this.img.height
        this.speedX = sx;
        this.direction= direction;
    }

    /**
     * Draws the object
     *
     * @param {CanvasRenderingContext2D} ctx
     */
    draw(ctx){
        //No need to draw if it's off the screen
        if(this.x + this.img.width < -200 || this.x > cWidth - this.img.width + 200)
return;

        if(this.direction == -1) {
            //This all essentially flips the image

            //Translates to the image position
            ctx.translate(this.x,this.y);

            // scaleX by -1; flip horizontally
            ctx.scale(-1,1);

            // draw the img
            // no need for x,y since we've already translated
            ctx.drawImage(this.img, 0, 0, this.img.width, this.img.height,
-this.img.width, 0, this.img.width, this.img.height);

            // reset transformations to default
            ctx.setTransform(1,0,0,1,0,0);
        }
        else {
            ctx.drawImage(this.img, this.x, this.y, this.img.width, this.img.height);
        }
    }

    /**
     * Updates the object
     */
    update(){
        if(this.x + this.img.width < 0 || this.x > cWidth-this.img.width + 200)
return;
        this.x += this.speedX * this.direction
        // Checks if shot by player
        if(this.isPlayer) {
            // Checks for collision then applies damage if collided
            if(collides(cpu, this) && !cpu.shielding) {
                this.x = -800
```

```
                cpu.damage += 25
                cpu.velocity[0] += 2*(1+cpu.damage*5/100)*this.direction
                cpu.frameNum = 1
                cpu.hurt = true
                sleep(1000).then(() => {
                    cpu.hurt = false
                })
            }
        }
        // Since it wasn't shot by the player
        else {
            // Checks for collision then applies damage if collided
            if(collides(player, this) && !player.shielding) {
                this.x = -800
                player.damage += 25
                player.velocity[0] += 2*(1+player.damage*5/100)*this.direction
                player.frameNum = 1
                player.hurt = true
                sleep(1000).then(() => {
                    player.hurt = false
                })
            }
        }
    }
}


// General fighter class the other fighters will inherit
export class fighter {
    /**
     * Contructs the fighter object
     *
     * @param {*} src This can be an array or a string depending on the sprite sheet
given
     * @param {Number} x
     * @param {Number} y
     * @param {Number} width
     * @param {Number} height
     */
    constructor(src, isPlayer=false, spW=64, spH=64, x = cWidth/2, y = 0, width = 15,
height = 8.5) {
        this.isPlayer = isPlayer;

        this.scale = 1.5

        this.attackRange = {
            x: this.x,
            y: this.y,
            width: 20,
            height: this.height,
        }

        this.img = new Image();
        if(Array.isArray(src)){
            this.srcArray = src;
```

```
            this.img.src = path + src[0]
        }
        else this.img.src = path + src;

        this.offsetHeight = 0;
        this.offsetWidth = 20;

        this.shieldOffsetX = 0;
        this.offsetWidthY = 0;

        this.x = x;
        this.y = y;

        this.velocity = [0, 0] // [vx, vy]
        this.direction = 1;
        this.maxSpeed = 10
        this.jumpForce = 15
        this.grounded = true;
        this.hasDoubleJump = true

        this.lives = 3

        this.width = cWidth/width;
        this.height = cHeight/height;

        this.spriteWidth = spW;
        this.spriteHeight = spH;

        this.damage = 0;

        this.maxFrameNum = 1
        this.frameNum = 0
        this.gameFrame = 0

        this.animation = 8-1

        this.alive = true;
        this.hurt = false;
        this.moving = false;
        this.attacking = false;
        this.falling = false;

        this.shielding = false;


    }

    /**
     * Updates the fighter
     */
    update() {
        if(!this.isPlayer) {
            this.applyFriction()
            cpuLogic.cpuLogic()
        }
```

```
        this.applyGravity()
        this.gameFrame++

        this.checkState()

        this.animate()

        this.x += this.velocity[0];
        this.y += this.velocity[1];
        if(this.y == cHeight - ground - this.height) this.grounded = true
        else this.grounded = false

        if (this.direction == 1) {
            this.attackRange.x = this.x + this.width
        }
        else {
            this.attackRange.x = this.x - this.attackRange.width
        }
        this.attackRange.y = this.y

    }

    /**
     * Special update function for character select
     */
    updateSelectMode() {
        this.gameFrame++

        this.animate()
    }

    /**
     * Draw the fighter on the canvas
     *
     * @param {CanvasRenderingContext2D} ctx
     */
    drawHitboxes(ctx) {
        ctx.fillStyle = "black";
        ctx.fillRect(this.x, this.y, this.width, this.height);
        ctx.fillStyle = "red";
        ctx.fillRect(this.attackRange.x, this.attackRange.y, this.attackRange.width,
this.attackRange.height);
    }

    /**
     * Draw the fighter on the canvas
     *
     * @param {CanvasRenderingContext2D} ctx
     */
    draw(ctx) {
        if(this.direction == -1) {
            //This all essentially flips the image

            //Translates to the images position
            ctx.translate(this.x,this.y);
```

```
            // scaleX by -1; this "trick" flips horizontally
            ctx.scale(-1,1);

            // draw the img
            // no need for x,y since we've already translated
            ctx.drawImage(this.img, (this.frameNum-1)*this.spriteWidth,
(this.animation-1)*this.spriteHeight, this.spriteWidth, this.spriteHeight,
-this.spriteWidth-this.offsetWidth, this.offsetHeight, this.spriteWidth*this.scale,
this.spriteHeight*this.scale);

            // always clean up -- reset transformations to default
            ctx.setTransform(1,0,0,1,0,0);
        }
        else {
            ctx.drawImage(this.img, (this.frameNum-1)*this.spriteWidth,
(this.animation-1)*this.spriteHeight, this.spriteWidth, this.spriteHeight, this.x-
this.offsetWidth, this.y+this.offsetHeight, this.spriteWidth*this.scale,
this.spriteHeight*this.scale);
        }
        if(this.shielding){
            ctx.drawImage(SHIELD, this.x-this.shieldOffsetX, this.y-
this.shieldOffsetY, this.width, this.height);
        }
    }

    /**
     * Applies friction to the fighter
     */
    applyFriction() {
        if(this.velocity[0] !== 0) {
            if(this.velocity[0] > 0) {
                this.velocity[0] -= gravity
            }
            else if (this.velocity[0] < 0) {
                this.velocity[0] += gravity
            }
            if(this.velocity[0] > -1 && this.velocity[0] < 1 && this.velocity[0] !==
0) {
                this.velocity[0] = 0
            }

        }
    }

    /**
     * Applies gravity to the fighter
     */
    applyGravity(){
        if( this.y < cHeight-ground - this.height || this.y > cHeight - ground)
this.velocity[1] += gravity;
        else {
            if(this.x + this.width < 0 || this.x > cWidth) {
                this.velocity[1] += gravity;
                return
            }
            if(this.velocity[1] > 0){
                this.velocity[1] = 0;
```

```
                this.y = cHeight - ground-this.height;
                this.hasDoubleJump = true;
            }
        }
    }

    /**
     * Activates the shield for the fighter
     */
    shield() {
        if(!this.grounded || this.attacking) return
        this.velocity[0] = 0
        this.shielding = true
    }

    /**
     * Function to control the fighters movement
     *
     * @param {Number} direction 1, -1 or 0
     */
    moveX(direction = 0) {
        if(this.hurt) return

        if (direction ≠ 0){
            this.direction = direction;
        }

        this.velocity[0] += direction * 1;

        // Don't want movement limits to interfere with damage knockback
        if(this.velocity[0] > this.maxSpeed+5) return;
        else if(this.velocity[0] < -this.maxSpeed-5) return;

        else if(this.velocity[0] > this.maxSpeed) this.velocity[0] = this.maxSpeed;
        else if(this.velocity[0] < -this.maxSpeed) this.velocity[0] = -this.maxSpeed ;


    }

    /**
     * Function to make the fighter jump
     */
    jump() {
        if(!this.grounded && !this.hasDoubleJump || this.shielding || this.attacking)
return
        if(this.hasDoubleJump && !this.grounded) this.hasDoubleJump = false
        this.velocity[1] = -this.jumpForce;
    }
}

export class purple_arrow extends fighter {
    /**
     *
     * @param {String} src
     * @param {Boolean} isPlayer
```

```
     * @param {Number} x
     * @param {Number} y
     * @param {Number} width
     * @param {Number} height
     * @param {Number} spW Sprite Width
     * @param {Number} spH Sprite Height
     */
    constructor(src, isPlayer=false, spW=64, spH=64, x = cWidth/2, y = 0, width = 16,
height = 9) {
        super(src, isPlayer, spW, spH, x, y, width, height)
        this.scale = 1.5

        this.arrow_img = new Image()
        // Image of an arrow
        // Source: https://astrobob.itch.io/arcane-archer
        // Accessed on 3/24/23
        this.arrow_img.src = "./assets/fighters/purple_arrow/projectile.png"

        this.offsetHeight = -5;
        this.offsetWidth = 20;


        this.maxSpeed = 15


        this.maxFrameNum = 7
        this.frameNum = 0
        this.gameFrame = 0

        this.animation = 0

        this.arrow = new projectile(this.arrow_img, this.isPlayer)
    }


    /**
     * Draw the fighter on the canvas
     *
     * @param {CanvasRenderingContext2D} ctx
     */
    draw(ctx) {
        if((this.direction == -1 && this.animation!= 5) || (this.direction == 1 &&
this.animation == 5)) {
            //This all essentially flips the image

            //Translates to the image position
            ctx.translate(this.x,this.y);

            // scaleX by -1; flip horizontally
            ctx.scale(-1,1);

            // draw the img
            // no need for x,y since we've already translated
```

```
            ctx.drawImage(this.img, (this.frameNum-1)*this.spriteWidth,
(this.animation-1)*this.spriteHeight, this.spriteWidth, this.spriteHeight,
-this.spriteWidth-this.offsetWidth, this.offsetHeight, this.spriteWidth*this.scale,
this.spriteHeight*this.scale);

            // always clean up -- reset transformations to default
            ctx.setTransform(1,0,0,1,0,0);
        }
        else if ((this.direction == 1 && this.animation≠ 5) || (this.direction == -1
&& this.animation == 5)) {
            ctx.drawImage(this.img, (this.frameNum-1)*this.spriteWidth,
(this.animation-1)*this.spriteHeight, this.spriteWidth, this.spriteHeight, this.x-
this.offsetWidth, this.y+this.offsetHeight, this.spriteWidth*this.scale,
this.spriteHeight*this.scale);
        }
        if(this.shielding){
            ctx.drawImage(SHIELD, this.x-this.offsetWidth/4, this.y-this.offsetHeight,
this.width, this.height);
        }
        this.arrow.draw(ctx)
    }

    /**
     * Updates the fighter
     */
    update() {
        this.applyGravity()
        this.gameFrame++

        if(!this.isPlayer) {
            this.applyFriction()
            cpuLogic.cpuLogic()
        }


        this.checkState()

        this.animate()

        this.x += this.velocity[0];
        this.y += this.velocity[1];
        if(this.y == cHeight - ground - this.height) this.grounded = true
        else this.grounded = false

        this.arrow.update()
    }

    /**
     * Activates the fighter's first ability
     */
    ability1() {
        if(this.attacking || this.hurt) return
        this.velocity[0] = 0
        this.attacking = "ability1"
        this.frameNum = 1
        sleep(900).then(() => {
            this.attacking = false
```

```
            if(this.hurt) return
            this.arrow = new projectile(this.arrow_img, this.isPlayer, this.x,
this.y+this.height/2, 60, this.direction)
        })
    }

    /**
     * Activates the fighter's second ability
     */
    ability2() {
        if(this.attacking || this.hurt) return
        this.velocity[0] = this.velocity[0] * 1.5
        this.attacking = "ability2"
        this.frameNum = 1
        this.maxSpeed = this.maxSpeed*1.5
        sleep(1000).then(() => {
            this.attacking = false
            if(this.velocity[0] < 0) this.velocity[0] = -1
            else if(this.velocity[0] > 0) this.velocity[0] = 1
            this.maxSpeed = this.maxSpeed/1.5
        })

    }

    /**
     * Checks for a few states and responds correctly
     */
    checkState() {
        for (let i = 0; i < 4; i++) {
            if(this.attacking == "ability2" && Math.abs(this.velocity[0]) <
this.maxSpeed) this.moveX(this.direction)
            if(Math.abs(this.velocity[0]) > this.maxSpeed) {
                if(this.velocity[0] < -this.maxSpeed) this.velocity[0] =
-this.maxSpeed
                else if(this.velocity[0] > this.maxSpeed) this.velocity[0] =
this.maxSpeed
            }
        }
        if(this.x < -500 || this.x+this.width > cWidth + 500 || this.y > cWidth + 500
|| this.y < -500) this.alive = false
    }

    // Code to manage current animation and animation frame.
    // Author: Me
    // Source: characters.js
    // Accessed on 2/16/23
    animate(){
         //Staggers the frames so the animations don't play too fast
         if(this.gameFrame % staggerFrame != 0) return;
         //Animates next frame if there is another frame otherwise start over from
first frame
         if(this.frameNum < this.maxFrameNum) this.frameNum++;
         else{
             //Checks if the character lost because then there is no need to update
animations
             if(!this.alive) return;
             this.frameNum = 1
```

```
        }
        //Eveything below handles a majority of the animation logic

        //Checks if the conditions are met runs the animation then returns otherwise
        //Attacked Animations
       if(this.hurt){
            this.maxFrameNum = 2;
            this.animation = 8;
            if(this.frameNum > this.maxFrameNum) this.frameNum = 1;
            return
        }

        if(this.attacking) {
           if (this.attacking == "ability1") {
               this.animation = 4
               this.maxFrameNum = 7
           }
           else if (this.attacking == "ability2") {
               this.animation = 3
               this.maxFrameNum = 7
               this.velocity[0] = this.velocity[0] * 1.5
           }
            return;
        }

        else if(!this.grounded){
           let vy = this.velocity[1]
           if(vy <= 0 && this.animation != 7) {
               this.animation = 7
               this.maxFrameNum = 4
           }
           else if(vy > 0 && this.animation!= 5) {
               this.animation = 5
               this.maxFrameNum = 2
               if(this.frameNum > this.maxFrameNum) this.frameNum = 1
           }
           if(this.frameNum > this.maxFrameNum) this.frameNum = 1
           return
        }

        else if(this.velocity[0] != 0){
            this.animation = 1;
            this.maxFrameNum = 8;
            if(this.frameNum > this.maxFrameNum) this.frameNum = 1
            return
           }

           else{
            this.animation = 6
            this.maxFrameNum = 4;
            if(this.frameNum > this.maxFrameNum) this.frameNum = 1
            return;
        }
    }
}
```

```
export class warrior extends fighter {
    /**
     * Constructs the warrior class  which extends the fighter class
     *
     * @param {String} src
     * @param {Boolean} isPlayer
     * @param {Number} x
     * @param {Number} y
     * @param {Number} width
     * @param {Number} height
     * @param {Number} spW Sprite Width
     * @param {Number} spH Sprite Height
     */
    constructor(src, isPlayer=false, spW=80, spH=80, x = cWidth/2, y = 0, width = 16,
height = 9) {
        super(src, isPlayer, spW, spH, x, y, width, height)
        this.scale = 2

        this.attackRange = {
            x: this.x,
            y: this.y,
            width: 80,
            height: this.height,
        }

        this.offsetWidth = 40;
        this.offsetHeight = -57;

        this.shieldOffsetX = 0;
        this.shieldOffsetY = -5;

        this.maxSpeed = 10
        this.jumpForce = 10


        this.maxFrameNum = 7
        this.frameNum = 0
        this.gameFrame = 0

        this.animation = 0
    }

    /**
     * Activates the first ability of the fighter
     */
    ability1() {
        if(this.attacking || this.hurt) return
        this.velocity[0] = 0
        this.attacking = "ability1"
        this.frameNum = 1
        sleep(1500).then(() => {
            this.attacking = false
            if(this.hurt) return
            if(this.isPlayer) {
```

```
            // Checks for collision then applies damage if collided
            if (collides(cpu, this.attackRange) && !cpu.shielding) {
                cpu.hurt = true
                cpu.damage += 80
                cpu.velocity[0] += 2*(1+cpu.damage*5/100)*this.direction
                cpu.frameNum = 1
                sleep(1000).then(() => {
                    cpu.hurt = false
                })
            }
        }

        else {
            // Checks for collision then applies damage if collided
            if (collides(player, this.attackRange) && !player.shielding) {
                player.hurt = true
                player.damage += 80
                player.velocity[0] += 2*(1+player.damage*5/100)*this.direction
                player.frameNum = 1
                sleep(1000).then(() => {
                    player.hurt = false

                })
            }
        }

    })
}

/**
 * Activates the second ability of the fighter
 */
ability2() {
    if(this.attacking || this.hurt) return
    this.velocity[0] = this.velocity[0] * 1.5
    this.attacking = "ability2"
    this.frameNum = 1
    this.maxSpeed = 20
    this.moveX(this.direction)
    sleep(500).then(() => {
        this.attacking = "ability1"
        this.frameNum = 10
        this.maxSpeed = 10
        sleep(150).then(
            () => {
                this.attacking = false
                if(this.hurt) return
                // Checks for collision then applies damage if collided
                if (collides(cpu, this.attackRange) && !cpu.shielding) {
                    cpu.damage += 60
                    cpu.velocity[0] += 2*(1+cpu.damage*5/100)*this.direction
                    cpu.frameNum = 1
                    cpu.hurt = true
                    sleep(1000).then(() => {
                        cpu.hurt = false
```

```
                    })
                }
                // Checks for collision then applies damage if collided
                else if (collides(player, this.attackRange) && !player.shielding)
{
                    player.hurt = true
                    player.damage += 80
                    player.velocity[0] += 2*(1+player.damage*5/100)*this.direction
                    player.frameNum = 1
                    sleep(1000).then(() => {
                        player.hurt = false

                    })
                }

                this.moveX(this.direction)
            })
        })

    }

    /**
     * Checks for a few states and responds correctly
     */
    checkState() {
        for (let i = 0; i < 4; i++) {
            if(this.attacking == "ability2" && Math.abs(this.velocity[0]) <
this.maxSpeed) this.moveX(this.direction)
            if(Math.abs(this.velocity[0]) > this.maxSpeed) {
                if(this.velocity[0] < -this.maxSpeed) this.velocity[0] =
-this.maxSpeed
                else if(this.velocity[0] > this.maxSpeed) this.velocity[0] =
this.maxSpeed
            }
        }
        if(this.x < -500 || this.x+this.width > cWidth + 500 || this.y > cWidth + 500
|| this.y < -500) this.alive = false
    }

    // Code to manage current animation and animation frame.
    // Author: Me
    // Source: characters.js
    // Accessed on 2/16/23
    animate(){
         //Staggers the frames so the animations don't play too fast
         if(this.gameFrame % staggerFrame != 0) return;
         //Animates next frame if there is another frame otherwise start over from
first frame
         if(this.frameNum < this.maxFrameNum) this.frameNum++;
         else{
             //Checks if the character lost because then there is no need to update
animations
             if(!this.alive) return;
             this.frameNum = 1
         }
         //Eveything below handles a majority of the animation logic
```

```
         //Checks if the conditions are met runs the animation then returns when
selecting the animation
        if(this.hurt){
            this.maxFrameNum = 5;
            this.animation = 4;
            if (this.frameNum > this.maxFrameNum) this.frameNum = 1
            return
        }

        if(this.attacking) {
            if (this.attacking == "ability1") {
                this.animation = 3
                this.maxFrameNum = 12
            }
            else if (this.attacking == "ability2") {
                this.animation = 3
                this.maxFrameNum = 7
                this.velocity[0] = this.velocity[0] * 1.5
            }
            if(this.frameNum == this.totalFrames) {
                if(this.currentAttack == 1) {
                    util.sleep(400).then(() => {
                        this.canAttack1 = true;
                    })
                }
                else if(this.currentAttack == 2) {
                    util.sleep(1700).then(() => {
                        this.canAttack2 = true;
                    })
                }
                    this.attacking = false;
                    this.currentAttack = 0;
             }
             return;
        }

        else if(!this.grounded){
            let vx = this.velocity[0]
            this.animation = 2
            this.maxFrameNum = 6
            if (vx == 0) {
                this.animation = 1
                this.maxFrameNum = 9
            }
            if(this.frameNum > this.maxFrameNum) this.frameNum = 1
            return
        }

        else if(this.velocity[0] != 0){
             this.animation = 2;
             this.maxFrameNum = 6;
             if(this.frameNum > this.maxFrameNum) this.frameNum = 1
             return
            }
```

```
            else{
             this.animation = 1
             this.maxFrameNum = 9;
             if(this.frameNum > this.maxFrameNum) this.frameNum = 1
             return;
            }
        }
    }
```

JS\Create PT\Ultra Crush Siblings\masterCSS.css

```css
/* Font that looks nice */
/* From: https://fontmeme.com/fonts/super-smash-font/ */
/* Accessed on 3/10/23 */
@font-face {
    font-family: "ssbu";
    src: url('./assets/fonts/ssbu.ttf')
}
html,
body {
    margin: 0;
    padding: 0;
    height: 100%
}



body{
    background: #1a1e24;
    color: whitesmoke
    font-family: Noto Sans;
    width: 100vw;
    height: 100vh;
    margin: 0;
    display: flex;
    flex-wrap: wrap;
    justify-content: center;
    align-items: center;
    text-align: center;
    gap: 0px;
}

canvas{
    display: flex;
    background: whitesmoke
    border-radius: 10px;
    /* border: .25em solid #8b8b8b; */
    scale: 1;
}

#myCanvas{
    /* Relative as possible so it works on many devices */
    /* width: 46.875vw; */
    /* height: 61.538461538vh; */
    width:1200px;
    height:600px;

    scale:1.1
}

/*Original Dimensions: 1200x600*/
```