```
// moving through space

const movingInstructions2D = ["move quickly", "move carefully", "move steadily"];
const breathingSounds = ["/sound/breathe-in/breatheIn1.mp3",
"/sound/breathe-in/breatheIn2.mp3", "/sound/breathe-in/breatheIn3.mp3",
"/sound/breathe-in/breatheIn4.mp3", "/sound/breathe-out/breatheOut1.mp3",
"/sound/breathe-out/breatheOut2.mp3", "/sound/breathe-out/breatheOut3.mp3",
"/sound/breathe-out/breatheOut4.mp3"]

const objects = [
 {object: document.querySelector('#endGame'), sound: "/sound/bells/bell8.mp3",
prompt: " "},
 {object: document.querySelector('#obj1'), sound: "/sound/bells/bell1.mp3", prompt:
"navigate to center"},
 {object: document.querySelector('#obj2'), sound: "/sound/bells/bell2.mp3", prompt:
"glitch briefly"},
 {object: document.querySelector('#obj3'), sound: "/sound/bells/bell3.mp3", prompt:
"drift slightly"},
 {object: document.querySelector('#obj4'), sound: "/sound/bells/bell4.mp3", prompt:
"simulate calm"},
 {object: document.querySelector('#obj5'), sound: "/sound/bells/bell5.mp3", prompt:
"make a loop"},
 {object: document.querySelector('#obj6'), sound: "/sound/bells/bell6.mp3", prompt:
"invert logic"},
 {object: document.querySelector('#obj7'), sound: "/sound/bells/bell7.mp3", prompt:
"reroute direction"},
 {object: document.querySelector('#obj8'), sound: "/sound/bells/bell1.mp3", prompt:
"observe pattern"},
 {object: document.querySelector('#obj9'), sound: "/sound/bells/bell2.mp3", prompt:
"compress distance"},
 {object: document.querySelector('#obj10'), sound: "/sound/bells/bell7.mp3",
prompt: "float like data"},
]

function characterWalking(keyPressed) {
        const character = document.querySelector('.character');
        //const objects = document.querySelectorAll('.obj');
        const screenWidth = window.innerWidth;
        const screenHeight = window.innerHeight;

        track1.src = getRandom(breathingSounds);
        track1.play();

        // Helper functions for vw <-> px
```

```
function pxToVw(px) { return (px / window.innerWidth) * 100; }
function vwToPx(vw) { return (vw / 100) * window.innerWidth; }
function pxToVh(px) { return (px / window.innerHeight) * 100; }
function vhToPx(vh) { return (vh / 100) * window.innerHeight; }

// Store character position in vw/vh
if (character.dataset.vw === undefined || character.dataset.vh ===
undefined) {
  // Initialize from current px position
  const rect = character.getBoundingClientRect();
  character.dataset.vw = pxToVw(rect.left);
  character.dataset.vh = pxToVh(rect.top);
}
let currentVw = parseFloat(character.dataset.vw);
let currentVh = parseFloat(character.dataset.vh);

// Movement step in vw/vh
const stepVw = pxToVw(30); // 10px step, converted to vw
const stepVh = pxToVh(30); // 10px step, converted to vh

// Movement
if (keyPressed.key === "ArrowLeft" && currentVw > 0) {
  currentVw -= stepVw;
  if (currentVw < 0) currentVw = 0;
} else if (keyPressed.key === "ArrowRight" && currentVw < 100 -
pxToVw(character.clientWidth)) {
  currentVw += stepVw;
  if (currentVw > 100 - pxToVw(character.clientWidth)) currentVw = 100 -
pxToVw(character.clientWidth);
} else if (keyPressed.key === "ArrowUp" && currentVh > 0) {
  currentVh -= stepVh;
  if (currentVh < 0) currentVh = 0;
} else if (keyPressed.key === "ArrowDown" && currentVh < 100 -
pxToVh(character.clientHeight)) {
  currentVh += stepVh;
  if (currentVh > 100 - pxToVh(character.clientHeight)) currentVh = 100 -
pxToVh(character.clientHeight);
}

character.dataset.vw = currentVw;
character.dataset.vh = currentVh;
character.style.left = `${currentVw}vw`;
character.style.top = `${currentVh}vh`;

objHit = false;
```

```
 currentHitObj = null;
 let endGameOverlap = false;

 objects.forEach(obj => {
   const objRect = obj.object.getBoundingClientRect();
   const charRect = character.getBoundingClientRect();

   const margin = 40; // wiggle room in pixels
   let isCollision =
     charRect.left < objRect.right + margin &&
     charRect.right > objRect.left - margin &&
     charRect.top < objRect.bottom + margin &&
     charRect.bottom > objRect.top - margin;

   const charRectDebug = character.getBoundingClientRect();
   const objRectDebug = objRect;
   //console.log('character:', charRectDebug);
   //console.log(obj.object.id + ' :', objRectDebug);
   //console.log('isCollision:', isCollision);

   if (obj.object.id == "endGame") {
     const endGameDiv = document.getElementById('endGame');
     const endGameRect = endGameDiv.getBoundingClientRect();
     console.log('endGameDiv:', endGameRect);
     // Extra checks
     if (charRectDebug.width === 0 || charRectDebug.height === 0)
console.warn('Character has zero size!');
     if (endGameRect.width === 0 || endGameRect.height === 0)
console.warn('endGameDiv has zero size!');
     if (window.getComputedStyle(character).display === 'none')
console.warn('Character is display:none!');
     if (window.getComputedStyle(endGameDiv).display === 'none')
console.warn('endGameDiv is display:none!');
     if (window.getComputedStyle(character).visibility === 'hidden')
console.warn('Character is visibility:hidden!');
     if (window.getComputedStyle(endGameDiv).visibility === 'hidden')
console.warn('endGameDiv is visibility:hidden!');
     if (isCollision) {
       endGameOverlap = true;
     }
   }

   if (isCollision) {
     objHit = true;
     currentHitObj = obj; // <- Store the hit object
```

```
    } else {
      //obj.object.style.backgroundColor = "transparent";
    }
  });

  // Only check for endGame center alignment if it is the object hit
  function isCenterOverlap(divA, divB, maxDistance = 30) {
    const rectA = divA.getBoundingClientRect();
    const rectB = divB.getBoundingClientRect();
    const centerAx = rectA.left + rectA.width/2;
    const centerAy = rectA.top + rectA.height/2;
    const centerBx = rectB.left + rectB.width/2;
    const centerBy = rectB.top + rectB.height/2;
    const dx = centerAx - centerBx;
    const dy = centerAy - centerBy;
    const distance = Math.sqrt(dx*dx + dy*dy);
    return distance < maxDistance;
  }
  if (endGameOverlap) {
     console.log("endGame overlap detected");
    const endGameDiv = document.getElementById('endGame');
    if (isCenterOverlap(character, endGameDiv, 30)) {
      console.log("Character and endGame are centered within 30px!");
      // Expand the endGame div to become a full-screen square
      if (endGameDiv) {
        // Get current position and size
        const rect = endGameDiv.getBoundingClientRect();
        // Set to fixed at current position and size
        endGameDiv.style.position = 'fixed';
        endGameDiv.style.top = rect.top + 'px';
        endGameDiv.style.left = rect.left + 'px';
        endGameDiv.style.width = rect.width + 'px';
        endGameDiv.style.height = rect.height + 'px';
        endGameDiv.style.zIndex = '9999';
        endGameDiv.style.transition = 'all 0.8s
cubic-bezier(0.77,0,0.175,1)';
        // Animate to full screen square and border-radius 0
        setTimeout(() => {
          endGameDiv.style.top = '0';
          endGameDiv.style.left = '0';
          endGameDiv.style.width = '100vw';
          endGameDiv.style.height = '100vw';
          endGameDiv.style.borderRadius = '0';
        }, 10);
        // Black hole text logic (after expansion)
```

```
      setTimeout(() => {
        habbohotel.style.display = "none";
        systemBackground.style.display = "block";
        fadeAudioOut(pianoMusic, 10000);
        fadeAudioIn(music, 10000);
        systemCheckTime1();
        setTimeout(()=>{
          glitchDivBackgroundFlicker(systemBackground, 2000);
            }, 13000)
            setTimeout(()=> {
              fadeOutElement(systemBackground, "0.1s", 100);
              setTimeout(()=>{
                systemBackground.style.display = "none";
              }, 1000);
              addVideoToDesktop();
            }, 15000);
            setTimeout(()=>{
              endOfPiece = true;
              fadeAudioIn(pianoMusic, 20000);
              pauseAllYouTubePlayersAndFadePopups(5000);
            }, 80000)
            setTimeout(()=>{
              fadeOutElement(desktopBackground, "5s", 5000);
              setTimeout(()=>{
                desktopBackground.remove();
              }, 5000);
              decideDrawingCanvasVisibility(true);
              const drawingBackground = document.querySelector('.drawing');
              fadeInElement(drawingBackground, "2s", 2000);
              fadeAudioOut(music, 10000);
            }, 95000)

            setTimeout(()=>{
              const drawingBackground = document.querySelector('.drawing');
              fadeOutElement(drawingBackground, "5s", 5000);
              fadeAudioOut(pianoMusic, 5000);
              setTimeout(()=>{
                location.reload();
              }, 10000);
            }, 120000);
      }, 900); // after expansion animation
    }
  }
}
// Maintain persistent random instruction until another object is hit
```

```
 if (!window._lastObjHitState) window._lastObjHitState = false;
 if (!window._currentMovingInstruction) window._currentMovingInstruction =
getRandom(movingInstructions2D);

 if (objHit) {
   console.log("OBJ HIT:", currentHitObj);
   characterText.textContent = currentHitObj.prompt;
   characterText.style.color = "black";
   window._lastObjHitState = true;
 } else {
   console.log("OBJ NOT HIT");
   if (window._lastObjHitState) {
     // Only pick a new random instruction when transitioning from hit to
not hit
     window._currentMovingInstruction = getRandom(movingInstructions2D);
   }
   characterText.style.color = "black";
   characterText.textContent = window._currentMovingInstruction;
   window._lastObjHitState = false;
 }
}
```