**Project Documentation: “Eras” – An Interactive Human Evolution Experience**

**1. Project Overview**

“Eras” is a **visual interactive experience** created in p5.js that allows a user to explore different stages of human history through interaction. The main concept revolves around **scene transitions**: the user navigates a human character on a “Begin” page, interacts with blocks to reveal portals, and enters different eras such as Prehistoric, Ancient, Medieval, Industrial, and Modern.

The project combines **sprite-based animations, physics-based jumping, collision detection**, and **dynamic scene management**. The key idea is to **teach users about human evolution** interactively while creating a game-like experience inspired by Mario-style mechanics.

**2. Inspiration**

The inspiration came from **classic platformer games** like Super Mario, where the player interacts with objects (blocks) to change the environment. The goal was to **merge educational content with interactive gameplay**, allowing users to explore historical eras, learn key information, and experience smooth scene transitions using **animation and portals**.

**3. Features**

* A human character with **idle and walking animations**, scalable relative to the canvas.
* **Jump mechanics** with collision detection against blocks.
* **Interactive blocks** that spawn **portals** when hit.
* **Green portals** take the player to a specific era.
* **Red portals** return the player to the Begin page.
* **Era pages** with background images, era names, and descriptive boxes.
* **Dynamic scaling** to fit any screen size using windowWidth and windowHeight.
* **Single-interaction blocks**: only one block is active at a time to prevent multiple portal activations.

**4. Problems Faced and Solutions**

| **Problem** | **Solution** |
| --- | --- |
| Human sprite appeared very squished | Adjusted width/height based on **sprite sheet frame ratio** and canvas size. |
| Blocks were too low and jump height was inconsistent | Increased jump height and repositioned blocks higher above baseline. |
| Portal appeared above blocks or disappeared during motion | Fixed portal location to **bottom-right of canvas** and made portal persistent until used. |
| Multiple blocks could be hit simultaneously | Added logic so **only one block is interactable at a time**; toggling a block locks others. |
| Scene transitions didn’t trigger on portal collision | Created **precise rectangle-ellipse collision detection** between human bounds and portal. |
| Text overflowed the era description box | Used textLeading and text() with width/height constraints to ensure **text fits inside the box**. |
| Backgrounds did not cover full canvas | Used imageMode(CORNER) and scaled backgrounds to width x height. |
| Human could leave canvas | Added constrain() for X-axis to **prevent leaving the left side on Begin page**, and smooth transitions to right. |

**5. Class Structure**

**5.1 Human Class**

Handles the player character: **position, animation, movement, jump physics**.

class Human {

constructor(x, y, idleSheet, walkSheet, frameCounts) { ... }

move(dir) { ... } // Move left/right

update() { ... } // Animate sprite

show() { ... } // Draw sprite on canvas

getBounds() { ... } // Return human bounding box for collision

}

**Explanation:**

* x, y represent the human’s **bottom-center position**.
* Maintains **facing direction** for left/right movement.
* Animates using frameIndex and frame counters for **idle and walk**.
* getBounds() provides **accurate rectangle for collision detection**.

**5.2 Block Class**

Represents a block that can be **hit to spawn a portal**.

class Block {

constructor(x, y, w, h, eraIndex) { ... }

show() { ... } // Draw block

togglePortal(human) { ... } // Detect interaction & toggle portal

}

**Explanation:**

* hit tracks whether the block’s portal is active.
* togglePortal() uses **tight collision detection**: only toggles when human is underneath and not falling past.
* Other blocks are locked when one block’s portal is active.

**5.3 Portal Class**

Represents the **green/return portal**.

class Portal {

constructor(x, y, w, h, col, labelText) { ... }

show() { ... } // Draw portal with optional label

}

**Explanation:**

* x, y define bottom-right position.
* ellipse() used to represent the portal visually.
* Label displays either "To [Era]" or "Back to Beginning".
* Portal collision is **detected using rectangle-ellipse math** to ensure accuracy.

**5.4 Scene Management**

The state variable handles the current scene:

* "begin" → Begin page with blocks and portals.
* "era" → Specific era page with background, description box, human, and return portal.

Transitions occur when **human collides with a portal**, using:

if (checkPortalCollision(human, beginPortal)) {

state = "era";

currentEra = blocks.find(b => b.hit).eraIndex;

...

}

**6. Use Cases**

* **Educational:** Users can explore human history in an interactive way.
* **Gaming Experience:** Combines platformer mechanics with educational content.
* **Web Integration:** Can run in a browser; responsive to different screen sizes.
* **Interactive Learning:** Blocks and portals provide gamified learning triggers.

**7. Lessons Learned**

* **Collision mechanics**: Detecting interaction between rectangles (human) and ellipses (portal) required precise math.
* **Sprite scaling**: Maintaining aspect ratio is crucial to avoid squished or stretched sprites.
* **Modular coding**: Breaking into Human, Block, Portal classes made debugging easier.
* **Scene management**: Proper state handling ensures smooth transitions.
* **Text layout**: Using text boxes and constraints ensures readability.
* **Game feel**: Small details like jump height, portal position, and block locking affect usability.

**8. Conclusion**

This project demonstrates how **interactive web experiences** can be created using p5.js, combining **animation, physics, collision, and scene management**. It integrates educational content with a **game-like interface**, showing the value of modular design, debugging complex interactions, and learning from iterative development.

**Code Reference – “Eras” Project**

**1. Human Class**

Handles human movement, animations, and collisions.

class Human {

constructor(x, y, idleSheet, walkSheet, frameCounts) {

this.x = x; // bottom-center X

this.y = y; // baseline Y

this.idleSheet = idleSheet; // idle animation sprite sheet

this.walkSheet = walkSheet; // walk animation sprite sheet

this.frameCounts = frameCounts; // {idle:7, walk:10}

this.frameIndex = 0;

this.frameCounter = 0;

this.currentAnim = "idle"; // "idle" or "walk"

this.facing = 1; // 1 = right, -1 = left

// Preserve aspect ratio based on sprite sheet

let singleFrameWidth = idleSheet.width / frameCounts.idle;

let singleFrameHeight = idleSheet.height;

let desiredHeight = height \* 0.25;

this.height = desiredHeight;

this.width = desiredHeight \* (singleFrameWidth / singleFrameHeight);

}

move(dir) {

if (dir !== 0) {

this.currentAnim = "walk";

this.facing = dir;

this.x += dir \* 10; // human speed

} else {

this.currentAnim = "idle";

}

}

update() {

this.frameCounter++;

if (this.frameCounter % 5 === 0) {

this.frameCounter = 0;

this.frameIndex++;

let count = this.currentAnim === "idle" ? this.frameCounts.idle : this.frameCounts.walk;

if (this.frameIndex >= count) this.frameIndex = 0;

}

}

show() {

let sheet = this.currentAnim === "idle" ? this.idleSheet : this.walkSheet;

let frameCount = this.currentAnim === "idle" ? this.frameCounts.idle : this.frameCounts.walk;

let frameW = sheet.width / frameCount;

let sx = this.frameIndex \* frameW;

push();

translate(this.x, this.y - this.height / 2);

scale(this.facing, 1);

imageMode(CENTER);

image(sheet, 0, 0, this.width, this.height, sx, 0, frameW, sheet.height);

pop();

}

getBounds() {

return {

x1: this.x - this.width / 2,

y1: this.y - this.height,

x2: this.x + this.width / 2,

y2: this.y

};

}

}

**Key Points:**

* Animation frames are extracted from sprite sheets.
* Aspect ratio is preserved to avoid squishing.
* getBounds() is used for **collision detection** with blocks and portals.

**2. Block Class**

Represents a block that can be **hit to spawn a portal**.

class Block {

constructor(x, y, w, h, eraIndex) {

this.x = x;

this.y = y;

this.w = w;

this.h = h;

this.hit = false; // portal active?

this.readyToToggle = true; // prevents repeated toggling while in contact

this.eraIndex = eraIndex; // which era this block leads to

}

show() {

fill(this.hit ? "grey" : "yellow");

rect(this.x, this.y, this.w, this.h);

}

togglePortal(human) {

let hb = human.getBounds();

let horizontalCollision = hb.x2 > this.x && hb.x1 < this.x + this.w;

let verticalCollision = hb.y1 < this.y + this.h && hb.y2 > this.y;

if (horizontalCollision && verticalCollision && this.readyToToggle) {

this.hit = !this.hit; // toggle portal

this.readyToToggle = false; // prevent multiple activations

return true;

}

if (!verticalCollision) this.readyToToggle = true;

return false;

}

}

**Key Points:**

* togglePortal() ensures portal activates **only when human collides correctly**.
* Other blocks can be locked while one portal is active.

**3. Portal Class**

Represents portals (green to era, red to Begin page).

class Portal {

constructor(x, y, w, h, col, labelText = null) {

this.baseX = x;

this.baseY = y;

this.w = w;

this.h = h;

this.col = col;

this.labelText = labelText;

}

show() {

fill(this.col);

noStroke();

ellipse(this.baseX + this.w / 2, this.baseY, this.w, this.h); // oval portal

if (this.labelText) {

fill(255);

textSize(18);

textAlign(CENTER, BOTTOM);

text(this.labelText, this.baseX + this.w / 2, this.baseY - this.h / 2 - 5);

}

}

}

**Key Points:**

* Label helps the player know **which era they are going to**.
* Oval shape visually separates portals from blocks.

**4. Main Sketch Logic**

let human, blocks = [], currentPortal = null;

let state = "begin"; // "begin" or "era"

let currentEra = 0;

let eraHumanBaseline;

function preload() {

idleSheet = loadImage("Idle.png");

walkSheet = loadImage("Walk.png");

// Load era backgrounds

eraBGs = [

loadImage("prehistoric.png"),

loadImage("ancient.png"),

loadImage("medieval.png"),

loadImage("industrial.png"),

loadImage("modern.png")

];

}

function setup() {

createCanvas(windowWidth, windowHeight);

eraHumanBaseline = height - height\*0.12;

human = new Human(width\*0.05, eraHumanBaseline, idleSheet, walkSheet, {idle:7, walk:10});

// Create 5 blocks horizontally centered

let blockW = width \* 0.05;

let blockH = height \* 0.05;

let startX = width \* 0.2;

let gap = width \* 0.12;

for (let i = 0; i < 5; i++){

blocks.push(new Block(startX + i\*gap, eraHumanBaseline - height\*0.15, blockW, blockH, i));

}

}

function draw() {

background(100, 150, 200);

if (state === "begin") drawBegin();

else drawEra();

}

function drawBegin(){

// Draw ground

fill(100,50,20);

rect(0, eraHumanBaseline, width, height\*0.12);

// Draw blocks

blocks.forEach(b => b.show());

// Check block interaction

blocks.forEach(b => {

if (b.togglePortal(human)){

currentPortal = new Portal(width - 150, eraHumanBaseline - 20, 80, 40, "green", "To " + ["Prehistoric","Ancient","Medieval","Industrial","Modern"][b.eraIndex]);

blocks.forEach(other => { if(other !== b) other.hit = false; }); // lock other blocks

}

});

// Draw portal if exists

if (currentPortal) currentPortal.show();

// Human movement

let dir = 0;

if (keyIsDown(RIGHT\_ARROW)) dir = 1;

if (keyIsDown(LEFT\_ARROW)) dir = -1;

human.move(dir);

human.update();

human.show();

// Portal collision → enter era

if (currentPortal){

let hb = human.getBounds();

let px = currentPortal.baseX + currentPortal.w/2;

let py = currentPortal.baseY;

if (dist((hb.x1+hb.x2)/2,(hb.y1+hb.y2)/2, px, py) < currentPortal.w/2){

state = "era";

currentEra = blocks.find(b => b.hit).eraIndex;

human.x = width\*0.1; human.y = eraHumanBaseline;

}

}

// Prevent going off left

if (human.x < 0) human.x = 0;

}

function drawEra(){

// Draw era background

imageMode(CORNER);

image(eraBGs[currentEra], 0, 0, width, height);

// Draw ground

fill(100,50,20);

rect(0, eraHumanBaseline, width, height\*0.12);

// Draw human

let dir = 0;

if (keyIsDown(RIGHT\_ARROW)) dir = 1;

if (keyIsDown(LEFT\_ARROW)) dir = -1;

human.move(dir);

human.update();

human.show();

// Draw red return portal

let returnPortal = new Portal(width - 150, eraHumanBaseline - 20, 80, 40, "red", "Back to Beginning");

returnPortal.show();

// Collision to go back

let hb = human.getBounds();

let px = returnPortal.baseX + returnPortal.w/2;

let py = returnPortal.baseY;

if (dist((hb.x1+hb.x2)/2,(hb.y1+hb.y2)/2, px, py) < returnPortal.w/2){

state = "begin";

currentPortal = null;

blocks.forEach(b => b.hit = false);

human.x = width\*0.05; human.y = eraHumanBaseline;

}

}

**5. Summary**

* Each **class is modular**, allowing easy adjustments.
* **State management** (begin vs era) controls scene transitions.
* **Collision detection** ensures smooth interaction with blocks and portals.
* **Sprite scaling** maintains natural human proportions.
* **Portals** visually guide the player and control navigation.