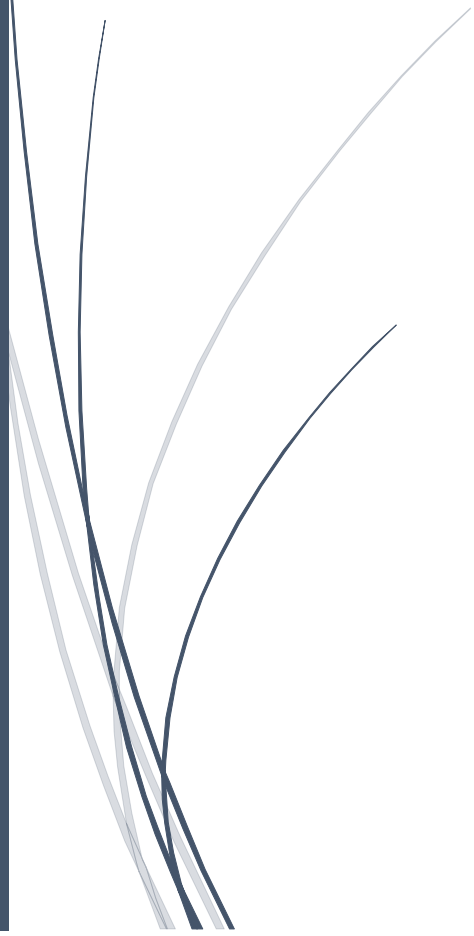
19-04-2023

IDS-forår 23

Studie nr. 68932

Tegn: 9889



Sebastian Antonio Lira
RUC

Navn: Sebastian Antonio Lira	Kursus: IDS	Dato: 19-04-2023
Studie: Roskilde Universitet	Stud nr. 68932	Underskrift:

Indhold

Navn: Sebastian Antonio Lira	Kursus: IDS	Dato: 19-04-2023
Studie: Roskilde Universitet	Stud nr. 68932	Underskrift:

Paragraph

The concept behind this Pong game with face-tracking controls is to create an interactive and engaging gaming experience that combines a classic arcade game with modern face-tracking technology. By leveraging the capabilities of the **p5.js** library for rendering and user interaction, and the **clmtrackr.js** library for face detection, the game offers a unique and intuitive control scheme that challenges players in a new and exciting way. The game aims to tackle the monotony of traditional control methods by providing a more immersive and dynamic experience, which adapts to the player's real-time movements. The idea behind the integration of face tracking not only adds an innovative element to the gameplay but also encourages physical activity and attentiveness, creating a more active and enjoyable gaming experience.

Program Description

The program is an interactive Pong game that integrates face tracking technology to control the player's paddle. It is designed using the **p5.js** library for graphics rendering and user interaction, along with the clmtrackr.js library for face tracking. The game comprises a video feed where the player's face is detected in real-time, and the paddle's position is adjusted based on the detected face's vertical position. The primary objective of the game is to bounce the ball between two paddles, with the computer-controlled paddle on the right side, and the player's face-controlled paddle on the left side.

Detailed Code Overview

The code is organized into the following sections to facilitate better understanding and functionality:

1. Variable declarations: This section consists of variables for face tracking, video input, pong game elements (paddles and their properties), and the ball (including its properties such as radius and speed).

```
// Variables for the face tracking model and video input
let tracker;
let video;
let isModelReady = false;

// Variables for the pong game
let paddleHeight = 100;
let paddleWidth = 20;
let paddleSpeed = 5;
let leftPaddle;
let rightPaddle;

// Variables for the ball
let ball;
let ballRadius = 12;
let ballSpeed = 5;

```

2. The **setup()** function: This function initializes essential game components such as the canvas, video elements, face tracking model, paddles, and the ball.

```
function setup() {
  // Create the canvas and video elements
  const canvas = createCanvas(640, 480);
  canvas.parent("sketch");

  video = createCapture(VIDEO);
  video.size(width, height);
  video.hide();

  // Start the face tracking model
  tracker = new clm.tracker();
  tracker.init();
  tracker.start(video.elt);
  isModelReady = true;
```

3. The **draw()** function: This function serves as the primary loop of the program, handling the video feed, face detection, paddle and ball updates, as well as collision checks between the ball and paddles.

```
function draw() {
  background(220);

  // Draw the video feed on the canvas
  image(video, 0, 0, width, height);

  // If the model is ready, detect faces in the video input
  if (isModelReady) {
    const positions = tracker.getCurrentPosition();
    if (positions !== false) {
      // Update the position of the player's paddle based on the face
position
      const face = positions[62];
      if (face) {
        const paddleY = map(
          face[1],
          0,
          height,
          paddleHeight / 2,
          height - paddleHeight / 2
        );
        leftPaddle.pos.y = paddleY;
```

4. The Paddle class: This class represents the paddles used in the game and contains relevant methods for their functionality.

```
// Paddle class
class Paddle {
  constructor(x, y, width, height) {
    this.pos = createVector(x, y);
    this.width = width;
    this.height = height;
  }

  update() {
    this.pos.y = constrain(
      this.pos.y,
      this.height / 2,
      height - this.height / 2
    );
  }
```

5. The Ball class: This class represents the ball used in the game and contains methods for its movement, rendering, and collision detection.

```
// Ball class
class Ball {
  constructor(x, y, radius, speed) {
    this.pos = createVector(x, y);
    this.vel = p5.Vector.random2D();
    this.vel.setMag(speed);
    this.radius = radius;
  }

  update() {
    this.pos.add(this.vel);
    if (this.pos.y < this.radius || this.pos.y > height - this.radi
      this.vel.y *= -1;
    }

    if (this.pos.x < 0 || this.pos.x > width) {
      this.pos.x = width / 2;
      this.pos.y = height / 2;
      this.vel = p5.Vector.random2D();
      this.vel.setMag(ballSpeed);
    }
  }
```

In-depth Data Structures

The primary data structures employed in this program are as follows:

1. Vectors: The p5.Vector class is utilized to store and manipulate 2D vectors effectively. Vectors are essential for representing the position and velocity of game elements such as paddles and the ball.

2. Arrays: Arrays are employed to store the detected facial feature points provided by the face tracking model. This information is crucial for updating the position of the player's paddle based on the vertical position of their face.

Comprehensive Design

The program's design adheres to object-oriented programming principles with the creation of the Paddle and Ball classes. This design approach ensures better code organization and simplifies the manipulation of game elements.

The **setup()** function is integral to initializing game elements and the face tracking model. It creates the canvas and video elements and attaches them to the **DOM** using the **parent()** method. The face tracking model is initialized using the **clm.tracker()** class and started with the video element as input.

The **draw()** function serves as the central loop of the game, managing the rendering of the video feed, face detection, and game element updates. The face tracking model detects faces in the video input, and the player's paddle position is updated accordingly. The paddles and the ball are updated, drawn on the canvas, and the ball's collisions with the paddles are checked within this loop.

The Paddle class has a constructor for initializing its position, width, and height, as well as two methods: **update()** and **show().** The **update()** method ensures the paddle's vertical position remains within the canvas boundaries, while the show() method renders the paddle on the canvas.

The Ball class has a constructor for initializing its position, velocity, and radius, and three methods: **update(), show(),** and **checkPaddleCollision(paddle**). The **update()** method modifies the ball's position based on its velocity, handles wall collisions, and resets the ball's position and velocity when it goes out of bounds. The **show()** method is responsible for rendering the ball on the canvas. The `**checkPaddleCollision(paddle)**` method is essential for detecting collisions between the ball and a given paddle, updating the ball's velocity upon a collision, and adding a slight effect based on the difference between the ball's position and the paddle's position. This effect ensures that the game remains dynamic and challenging.

What does not work?

The right paddle: The computer-controlled right paddle currently follows a simple algorithm to move vertically, attempting to align itself with the ball's position. This basic approach might not provide a sufficiently engaging and challenging experience for the player. To address this issue, an adaptive AI algorithm could be implemented, allowing the computer-controlled paddle to adjust its difficulty based on the player's performance. Additionally, incorporating different difficulty levels and strategies would make the gameplay more dynamic and enjoyable for players with varying skill levels.

Face-tracking optimization: The current face-tracking implementation, using the clmtrackr.js library, may not be optimized enough, leading to inaccuracies or inconsistencies in paddle control. To improve the face-tracking performance, I consider experimenting with different face-tracking libraries, such as Face-API.js or TensorFlow.js with a pre-trained face-detection model. Fine-tuning the face-tracking model, adjusting parameters, or integrating more robust face-detection algorithms could also enhance the overall accuracy and responsiveness of the face-tracking controls.

Ball getting vertically stuck: In certain cases, the ball may get stuck in a vertical movement pattern, making it difficult for the player to interact with the ball and limiting the gameplay dynamics. To resolve this issue, i would consider adding a mechanism to detect and prevent the ball from getting vertically stuck. One approach is to introduce a small random horizontal velocity component whenever the ball collides with a paddle or a boundary. This variation in velocity would ensure that the ball maintains a non-vertical trajectory, keeping the gameplay engaging and challenging.

By addressing these issues and incorporating the necessary improvements, the face-tracking Pong game will offer a more polished and satisfying gaming experience. Focusing on optimizing the face-tracking controls, enhancing the computer-controlled paddle's AI, and preventing the ball from getting vertically stuck will contribute to a more immersive, dynamic, and enjoyable game.

Enhancements and Future Work

The program, as it currently stands, offers a solid foundation for the Pong game with face-tracking controls. However, there are potential enhancements and features that can be added to improve the overall experience and complexity of the game:

1. Adaptive Computer Opponent: The computer-controlled paddle could use an AI algorithm to adapt its difficulty level based on the player's performance. This would ensure that the game remains engaging for players of different skill levels.
2. Scoring System: The addition of a scoring system would provide a clear objective for the player and a sense of accomplishment when they score points. The scores could be displayed on the canvas for easy access during gameplay.
3. Power-ups: Incorporating power-ups into the game would add an extra layer of depth and excitement. Power-ups could temporarily enhance a paddle's size, speed, or even add special abilities like freezing the opponent's paddle for a short duration.
4. Multiple Balls: Introducing multiple balls into the game would increase the challenge and require players to track and react to multiple objects simultaneously.
5. Customizable Game Settings: Allowing players to customize game settings, such as ball speed, paddle size, and game duration, would enable them to tailor the game to their preferences and skill level.
6. Multiplayer Mode: Implementing a multiplayer mode would enable two players to compete against each other using face tracking to control their respective paddles, further enhancing the game's interactivity and competitiveness.

Conclusion

This program demonstrates an engaging and interactive Pong game implementation that uses face tracking as a control input. By leveraging the **p5.js** library for rendering and user interaction, alongside the clmtrackr.js library for face tracking, the game features a unique control scheme that sets it apart from traditional Pong games. The program follows object-oriented programming principles with the creation of the Paddle and Ball classes, resulting in better code organization and more straightforward manipulation of game elements. With the addition of enhancements and new features, the game has the potential to become an even more immersive and enjoyable experience for players.

Link to code:

https://editor.p5js.org/Seblira99/collections/Iv6Jc6qok

Sources:

Jeff Thompson(2020) Face Tracking with P5JS: Part 1

https://www.youtube.com/watch?v=kCHpFe4T7_A

Audun Mathias Øygard Clmtrackr.js library

https://github.com/auduno/clmtrackr/releases/tag/v1.1.2