Faculty of Computer Science and Engineering Skopje

Documentation for Flappy Bird Multiplayer

Advanced Web Design

Mentor:
Boban Joksimoski

Author:
Ivan Ristovski 211059

Table of Contents

Introduction

This is document is a complete overview of my Flappy Bird Multiplayer app, developed as a project for the Advanced Web Design course at my university. The game is a web-based multiplayer adaptation of the classic Flappy Bird game, designed to allow multiple players to join and compete in real-time. The project focuses on implementing advanced web design techniques, integrating real-time data synchronization with Firebase, and creating an engaging and interactive user experience.

Technologies Used

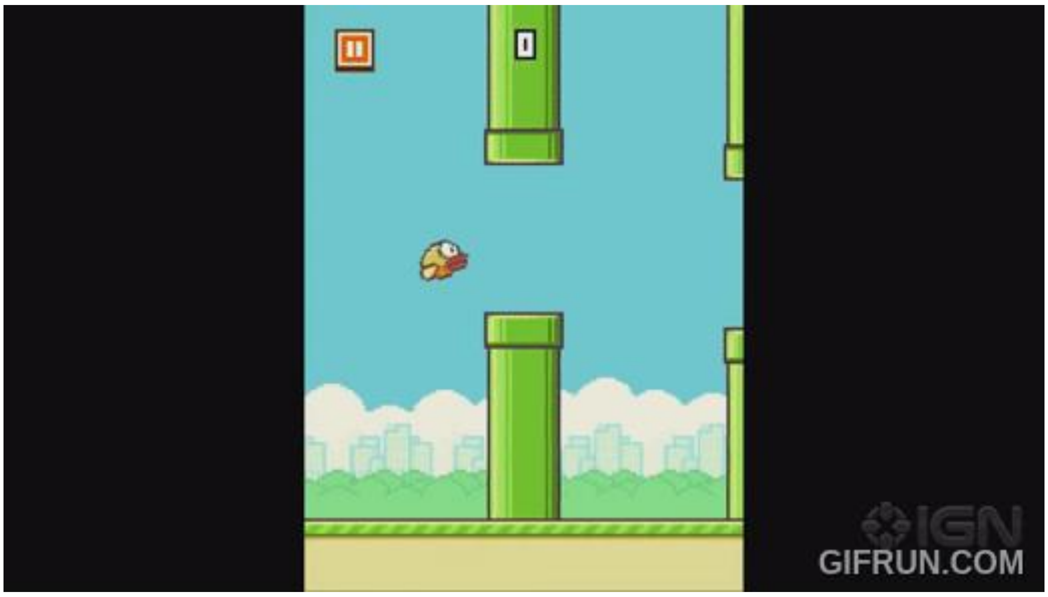
- **JavaScript:** The core programming language used to develop the game logic, handle player input, manage game state, and interact with the Firebase database. JavaScript enables dynamic, real-time updates and interactions for a smooth multiplayer experience
- **HTML5:** Provides the structural foundation of the web application, including the layout of the game interface and controls. The HTML5 <canvas> element serves as a container for drawing and manipulating the game graphics using JavaScript, allowing for dynamic rendering and game interaction within the browser.
- **CSS3:** Used for styling the game interface, including layout, fonts, colors, etc. Helps to make the project create an engaging and visually appealing experience for the players
- **Firebase:** A cloud-based platform that provides real-time database services to synchronize game data across all players in real-time. Firebase handles the backend aspects of user authentication, data storage, and event-driven updates, enabling the multiplayer functionality of the game.

Figure 1 HTML5,CSS3,JavaScript and Firebase's logos

Setup

More explanation

Flappy Bird was a popular and addictive mobile game released in 2013 for iOS and Android devices. The game quickly became a viral sensation due to its simple yet challenging mechanics, where players control a small bird and navigate it through a series of pipes by tapping the screen to stay airborne. The objective was to fly between gaps in the pipes without hitting them, accumulating points for each set of pipes successfully passed. Sadly, the game was so addictive that it had to be taken off from iOS and Google Play. But fortunate for me, that means that I can safely reuse some of its art without problems. Here is a quick recap of how it looked.



GIF 1 Flappy bird (courtesy – IGN)

Inspired by this classic, my **Flappy Bird Multiplayer** game retains the core mechanics of the original game but introduces a competitive multiplayer twist. In this version, multiple players can join the game simultaneously and compete to see who can survive the longest. The game is designed to sync all players' actions and game elements in real-time, ensuring a fair and competitive environment.

Finally, time for the code breakdown.

Code breakdown

Game Setup: HTML, CSS and Firebase Initialization

Firstly, we will take a look at the HTML file (index.html). Apart from the standard code that is inside every HTML file, we have:

```
 <link rel="stylesheet" href="flappy.css">
    <link rel="preconnect" href="https://fonts.googleapis.com">
    <link rel="preconnect" href="https://fonts.gstatic.com" crossorigin>
    <link
href="https://fonts.googleapis.com/css2?family=Press+Start+2P&display=swap"
rel="stylesheet">
```

This is used for importing a custom font ("Press Start 2P") from Google Fonts.

The **<link rel="preconnect"** lines optimize loading by establishing early connections to the font servers, improving performance.

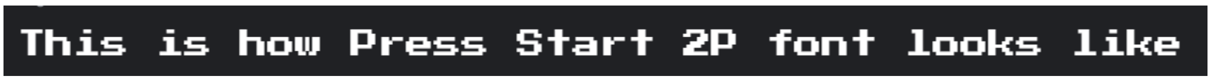

This is how Press Start 2P font looks like

Figure 2 Press Start 2P font (https://fonts.google.com/specimen/Press+Start+2P)

```
<canvas id="board"></canvas>
    <div class="game-container"></div>
```

The **<canvas>** element with the ID board is for rendering the game's graphics, such as the bird, pipes, and the background. The <div class="game-container"> is used for showing an image of the player's bird and its name.

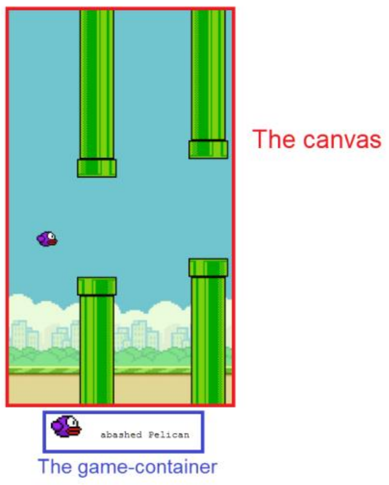


Figure 3 The canvas and the game-container element

Next up, we have the Firebase Scripts

```
<script src="https://www.gstatic.com/firebasejs/8.10.1/firebase-
app.js"></script>
    <script src="https://www.gstatic.com/firebasejs/8.10.1/firebase-
auth.js"></script>
    <script src="https://www.gstatic.com/firebasejs/8.10.1/firebase-
database.js"></script>
```
These scripts are for integrating Firebase into the game. The **firebase-app.js** initializes the Firebase app, **firebase-auth.js** handles user authentication, and **firebase-database.js** enables real-time database functionality to manage game state and synchronize data between players.

```
const firebaseConfig = {
          apiKey: "AIzaSyD1FDPY9IUbNoqTw_ubK5_plWTB48rMg9k",
          authDomain: "multiplayer-flappy-demo.firebaseapp.com",
          databaseURL: "https://multiplayer-flappy-demo-default-rtdb.europe-
west1.firebasedatabase.app",
          projectId: "multiplayer-flappy-demo",
          storageBucket: "multiplayer-flappy-demo.appspot.com",
          messagingSenderId: "106848050761",
          appId: "1:106848050761:web:dbd3c905a3c7b411a96998"
        };
        firebase.initializeApp(firebaseConfig);
    </script>
```

This block is for configuring and initializing Firebase using a specific configuration object. The **firebase.initializeApp(firebaseConfig)** function call sets up the Firebase app with the provided API keys and URLs, allowing the game to interact with Firebase services.

Now, the CSS3 file (flappy.css)

```
body{
    font-family: "Press Start 2P", system-ui;
    text-align: center;
}
#board {
    background-image: url("assets/flappy-background.png");
}
canvas {
    border: 2px solid black;
}
.Character_img{
    width: 50px;
    padding-right: 20px;
    padding-top: 10px;
}
```

This is pretty self-explanatory, so we will use a few words to explain it. At the **body**, the font which we explained before, is added. We also center the game and the picture and text below it. At the **#board**, which is an id that the canvas uses, we put the famously known background that flappy bird used back in the day. For the canvas we add a border which will make the whole experience look nicer. Next up, we have the **Character_img** class, where we also stylize the players' birds' images bin the **game-container**.

JavaScript

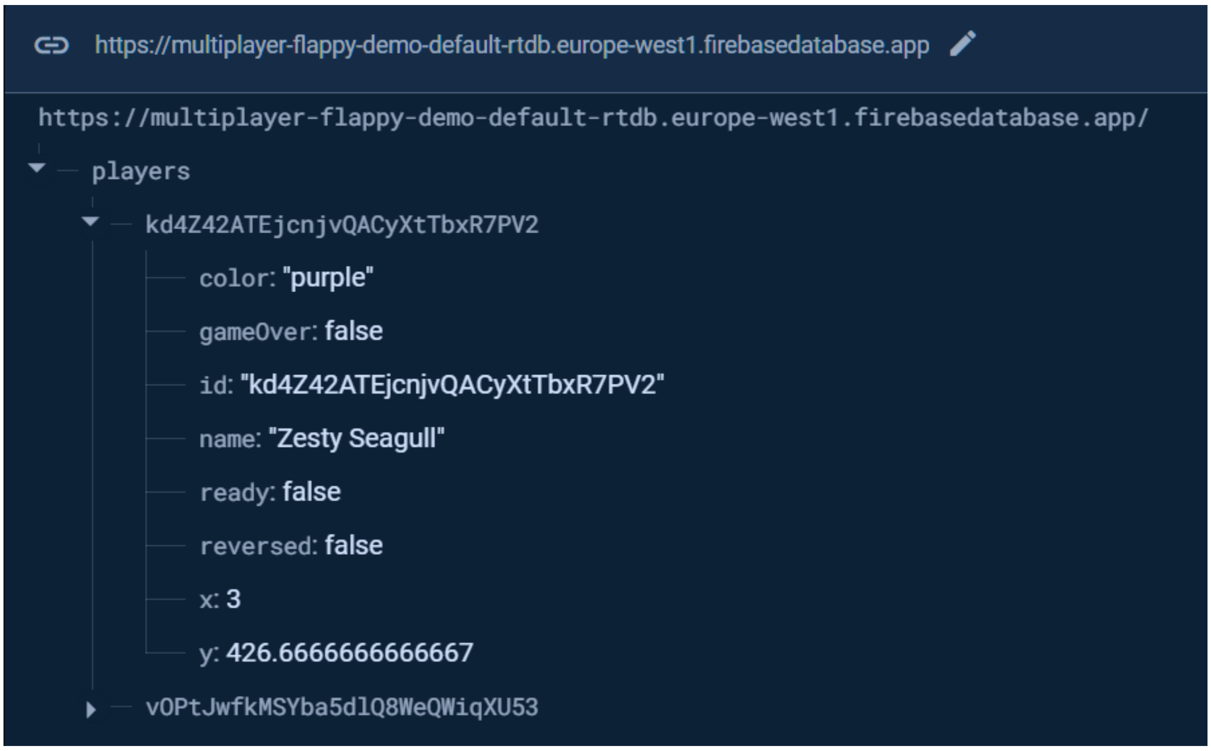
window.onload

In my JavaScript code, I start by defining a window.onload function. This function is crucial because it ensures that all the webpage's content, including images, scripts, and other assets, are fully loaded before any game logic is executed. Inside this function, I initialize variables related to the game and set up the core elements necessary for the game to run smoothly. Everything that we cover here is in this function. Note: I will go through the code chronologically, line by line, so, please keep in mind that some parts might appear a bit messy. This is because the code evolved over time as I encountered different challenges and added new features.

Global variables and constants

- Firebase based variables

```
let playerId; //unique for each player in the game session
let playerRef; //var used to store a reference to the Firebase, also unique
let playerElements = {}; //object that stores the DOM elements for a player
let players = {}; //all the current players' data in the game session
```



```
https://multiplayer-flappy-demo-default-rtdb.europe-west1.firebasedatabase.app

https://multiplayer-flappy-demo-default-rtdb.europe-west1.firebasedatabase.app/
  players
    kd4Z42ATEjcnjvQACyXtTbxR7PV2
      color: "purple"
      gameOver: false
      id: "kd4Z42ATEjcnjvQACyXtTbxR7PV2"
      name: "Zesty Seagull"
      ready: false
      reversed: false
      x: 3
      y: 426.6666666666667
    vOPtJwfkMSYba5dlQ8WeQWiqXU53
```

Figure 4 Firebase - Realtime Database, how the players' objects look

- Constants, objects, flags and other variables

```
• const playerColors = ["black", "blue", "brown", "cyan", "gray",
  "green", "peach", "pink", "purple", "red", "white", "yellow"];
```

```
const reversedPlayerColors = ["black", "blue", "brown", "cyan", "gray",
"green"]; // used for loading a reversed pipe array, explained further
const birdImages = {
        "black": "assets/flappy-bird-black.png",
        "blue": "assets/flappy-bird-blue.png",
        "brown": "assets/flappy-bird-brown.png",
        "cyan": "assets/flappy-bird-cyan.png",
        "gray": "assets/flappy-bird-gray.png",
        "green": "assets/flappy-bird-green.png",
        "peach": "assets/flappy-bird-peach.png",
        "pink": "assets/flappy-bird-pink.png",
        "purple": "assets/flappy-bird-purple.png",
        "red": "assets/flappy-bird-red.png",
        "white": "assets/flappy-bird-white.png",
        "yellow": "assets/flappy-bird-yellow.png",
};
let board;
let boardHeight = 640; //the background image has 640px height
let boardWidth = 360; // that’s why i have chosen these values
let context;

let birdWidth = 34;
let birdHeight = 24;
let birdX = boardWidth / 8; //bird’s x position
let birdY = boardHeight / 2;// y position, used for centering the bird
let birdImg;
let bird = {
        x: birdX,
        y: birdY,
        width: birdWidth,
        height: birdHeight
    };
let pipeArray = [];
let pipeWidth = 64;
let pipeHeight = 512;
let pipeX = boardWidth; //makes the pipes outside the right edge
let pipeY = 0;

const predefinedPipeYValues = [
        -390, -150, -330, -240, -270,
        -120, -420, -360, -180, -300,
        -450, -150, -330, -270, -420,
        -360, -120, -390, -240, -120
    ]; // -450 pixels is the upmost possible pipe, and -128 downmost

    const reversedPipeYValues = [
        -120, -240, -390, -120, -360,
        -420, -270, -330, -150, -450,
•          -300, -180, -360, -420, -120,
•          -270, -240, -330, -150, -390
•      ]; // I use 2 pipe presets, will explain why further
•  let useReversed = false;
•      let currentPipeIndex = 0;
•
•      let topPipeImg;
•      let bottomPipeImg;
•
•      let velocityX = -2;
•      let velocityY = 0;
•      let gravity = 0.4;
•      let jumped = false;
•
•      let gameOver = false;
•      let score = 0;
•      let gameStarted = false;
•      let waitingForPlayers=false;
•
•      const gameContainer = document.querySelector(".game-container");
•      const playerYpositions = [(boardHeight / 1.5),
•  (boardHeight / 2),(boardHeight / 2.5),(boardHeight / 3)]
•  // 4 predefined positions for the players' birds
```

Functions

```
function randomFromArray(array) {
        return array[Math.floor(Math.random() * array.length)];
    } // returns a random element from the array

function getName() {
        const adjective = randomFromArray([
            "Knowledgeable", "Coordinated", "Defective", "Ready", "Grumpy",
            "hysterical", "Bored", "Hateful", "Longing", "Laughable",
            "Grotesque", "Jumpy", "Noxious", "Abashed", "Whimsical",
            "Lavish", "Zesty", "Quiet", "Obscene", "Unkempt"
        ]);
        const birdName = randomFromArray([
            "Sparrow", "Eagle", "Parrot", "Penguin", "Hummingbird",
            "Flamingo", "Owl", "Peacock", "Falcon", "Robin", "Swan",
            "Pelican", "Woodpecker", "Kingfisher", "Canary", "Crow",
            "Dove", "Heron", "Toucan", "Seagull"
        ]);
        return `${adjective} ${birdName}`;
    } // 20 adjectives, 20 types of birds, every player gets a unique name

function drawAllBirds() {
        if (!context) return; // if context is not defined, return
        Object.keys(players).forEach(id => {
            const player = players[id];
            const img = new Image();
            img.src = birdImages[player.color];
            context.drawImage(img, player.x, player.y, birdWidth, birdHeight);
        });
    } //gets an array of all player IDs from the players object, and for each
one, it retrieves the player object corresponding to the player’s id, creates
a new image and draws the bird on its place, corresponding to the player color

    function initGame() {
        const allPlayersRef = firebase.database().ref(`players`);
       //creates a reference to the players node in the Firebase database
        allPlayersRef.on("value", (snapshot) => {
            players = snapshot.val() || {}; // retrieves the current state of
all players
            drawAllBirds(); // calls a function that draws every player’s bird

            checkIfAllPlayersReady(); // checks if all players are ready

            if (players[playerId]) {
                birdImg.src = birdImages[players[playerId].color];
            } // if the player’s data exists, it updates the bird’s image

        }); //Sets up an event listener for the value event on the
allPlayersRef, and triggers when there is a change to the entire players node

        // Handle new players joining
        allPlayersRef.on("child_added", (snapshot) => {
            const addedPlayer = snapshot.val();
            addPlayerElement(addedPlayer);
        }); // whenever a new child(player) is added to the players node, it
calls addPlayerElement function

        // Handle players leaving
        allPlayersRef.on("child_removed", (snapshot) => {
            const removedPlayer = snapshot.val();
            removePlayerElement(removedPlayer.id);
        }); // whenever a child(player) is removed from players node, it calls
removePlayerElement function






// Handle player updates (e.g., movement, score)
        allPlayersRef.on("child_changed", (snapshot) => {
            const changedPlayer = snapshot.val();
            updatePlayerElement(changedPlayer.id, changedPlayer);
        });
    } // whenever a child(player) is updated, it calls the updatePlayerElement
Function

function checkForColorAndSetReversed(colorsToCheck) {
        const hasColor = Object.values(players).some(player =>
colorsToCheck.includes(player.color)); //boolean value, if the player’s bird’s
color is included in the colorsToCheck array

        if (hasColor) {
            useReversed = true; // Set the flag to use the reversed array
        } else {
            useReversed = false; // Otherwise, use the normal array
        }
    } // we are setting up the useReversed for another function which will be
explained after.

function addPlayerElement(player) {
        const characterElement = document.createElement("div");
        characterElement.classList.add("Character");

        characterElement.innerHTML = `
        <div class="Character_sprite grid-cell"></div>
        <div class="Character_name-container">
            <img src="" class="Character_img"/>
            <span class="Character_name"></span>
        </div>
    `;

    playerElements[player.id] = characterElement;
    characterElement.querySelector(".Character_name").innerText = player.name;
    const imgElement = characterElement.querySelector(".Character_img");
    imgElement.src = "assets/flappy-bird-" + `${player.color}` + ".png";
    gameContainer.appendChild(characterElement);
    } //This function adds the player’s bird image and the name under the
canvas, in the game-container
 function updatePlayerElement(playerId, playerData) {
        const characterElement = playerElements[playerId];
        if (characterElement) {
            characterElement.querySelector(".Character_name").innerText =
playerData.name;
        }
    } //this updates the player’s name

 function removePlayerElement(playerId) {
        const characterElement = playerElements[playerId];
        if (characterElement) {
            gameContainer.removeChild(characterElement);
            delete playerElements[playerId];
        }
    } //removes the element from the game-container, removes the image and
name of the player

function setupFlappyBirdGame(color) {
        board = document.getElementById("board");
        board.height = boardHeight;
        board.width = boardWidth;
        context = board.getContext("2d");
       // initializes the game board
        birdImg = new Image();
        birdImg.src = birdImages[color] || "assets/flappy-bird-yellow.png"; //
Default to yellow if color not found
        birdImg.onload = function() {
            context.drawImage(birdImg, bird.x, bird.y, bird.width,
bird.height); //loads the bird’s image
        };
         // Wait until background and bird images are loaded
        document.addEventListener("keydown", startGame);
        document.addEventListener("keydown", moveBird);
        document.addEventListener("keyup", function(e) {
            if (e.code == "Space" || e.code == "ArrowUp" || e.code == "KeyX")
{
                jumped = false; // Resets the flag when the key is released
            }
        });
    }
 function checkIfAllPlayersReady() {
        const allPlayersReady = Object.values(players).every(player =>
player.ready);
        checkForColorAndSetReversed(reversedPlayerColors);
        if (allPlayersReady && !gameStarted) {
            startGameForAllPlayers();
        }
    } // if all players are ready, it starts the game
     function startGame(e){
        if (e.code === "ArrowUp" && !gameStarted) {
            playerRef.update({ready : true});
            checkIfAllPlayersReady();
        }
    }// if the key pressed was arrow up and the game hasn’t started, makes the
player ready, and calls to check if others are too

function startGameForAllPlayers() {

            gameStarted = true;
            topPipeImg = new Image();
            topPipeImg.src = "assets/pipe1.png";

            bottomPipeImg = new Image();
            bottomPipeImg.src = "assets/pipe2.png";

            setInterval(placePipes, 1500); // every 1.5 seconds
            requestAnimationFrame(update);
        } //creates 2 image objects, for the top pipe and the bottom pipe, and
sets up an interval to call placePipes every 1.5 seconds, also calls the
update function before every repaint.

function update() {
        if (!context) return;

        context.clearRect(0, 0, board.width, board.height);
      // clearing the entire board to avoid drawing over the previous frame
        // Draw pipes first
        for (let i = 0; i < pipeArray.length; i++) {
            let pipe = pipeArray[i];
            pipe.x += velocityX;
            context.drawImage(pipe.img, pipe.x, pipe.y, pipe.width,
pipe.height);
        } // iterating through all the pipes, moves them to the left to create
the scrolling effect and draws them

        drawAllBirds();
        // Bird physics for the current player
        if(!players[playerId].gameOver){
            velocityY += gravity; //applies the gravity effect
            bird.y = Math.max(bird.y + velocityY, 0); // Limit the bird.y to
the top of the canvas
            context.drawImage(birdImg, bird.x, bird.y, bird.width,
bird.height);

            // Check for collisions
            for (let i = 0; i < pipeArray.length; i++) {
                let pipe = pipeArray[i];
                if (detectCollision(bird, pipe)) {
                    playerRef.update({ gameOver: true });
                    break;
                }
            } //if the player hits the pipe, it updates the playerRef to
gameOver, meaning that he won’t play anymore for the round


            // If the bird falls
            if (bird.y > board.height) {
                playerRef.update({ gameOver: true });
            }
        }
         //clear pipes that have moved out of view
        while (pipeArray.length > 0 && pipeArray[0].x < -pipeWidth) {
            pipeArray.shift(); //removes first element from the array
        }
          // Check how many players are still alive
        let alivePlayers = Object.values(players).filter(player =>
!player.gameOver);

        if (players[playerId].gameOver) {
            if (alivePlayers.length === 1) {
                // If only one player is alive, show the game over message on
this player's screen
                const winnerName = alivePlayers[0].name;
                context.fillStyle = "white";
                context.font = "12px 'Press Start 2P', sans-serif";
                context.fillText(`GAME OVER,`, 5, 90);
                context.fillText(`${winnerName} WINS`, 5, 120);
                context.fillText("RESTART THE GAME", 5, 150);
            context.fillText("TO PLAY AGAIN", 5, 180);
            }
        } else if (alivePlayers.length === 1 && alivePlayers[0].id ===
playerId) {
            // If this is the last player alive, show "YOU WON" when they die
            context.fillStyle = "white";
            context.font = "12px 'Press Start 2P', sans-serif";
            context.fillText("YOU WON", 5, 90);
            context.fillText("RESTART THE GAME", 5, 120);
            context.fillText("TO PLAY AGAIN", 5, 150);
        }

        // Update the player's position in Firebase
        playerRef.update({
            x: bird.x,
            y: bird.y
        });

        requestAnimationFrame(update); //ensures a loop
    }




function placePipes() {
        if (gameOver || !gameStarted) {
            return;
        }
        //PipeY and reversedPipeY - vertical starting positions of the pipes
        let pipeY = predefinedPipeYValues[currentPipeIndex];
        let reversedPipeY = reversedPipeYValues[currentPipeIndex]
        let openingSpace = board.height / 4;
        const allPlayersReversed = Object.values(players).every(player =>
player.reversed); //checks if all players are reversed


        if(useReversed){ //if they are reversed
            let topPipe = {
                img: topPipeImg,
                x: pipeX,
                y: reversedPipeY,
                width: pipeWidth,
                height: pipeHeight,
                passed: false
            }; //creates the object

            pipeArray.push(topPipe); //adds it in the array

            let bottomPipe = {
                img: bottomPipeImg,
                x: pipeX,
                y: reversedPipeY + pipeHeight + openingSpace,
                width: pipeWidth,
                height: pipeHeight,
                passed: false
            };//creates the object
            pipeArray.push(bottomPipe); //adds it in the array
            currentPipeIndex = (currentPipeIndex + 1) %
predefinedPipeYValues.length; // points to the next predefined pipe position
        }
        else{
            let topPipe = {
                img: topPipeImg,
                x: pipeX,
                y: pipeY,
                width: pipeWidth,
                height: pipeHeight,
                passed: false
            };

            pipeArray.push(topPipe);

            let bottomPipe = {
                img: bottomPipeImg,
                x: pipeX,
                y: pipeY + pipeHeight + openingSpace,
                width: pipeWidth,
                height: pipeHeight,
                passed: false
            };
            pipeArray.push(bottomPipe);
            currentPipeIndex = (currentPipeIndex + 1) %
predefinedPipeYValues.length;
    }
}
function resetGameState() {
    bird.y = birdY;
    velocityY = 0; // Reset velocity
    gameOver = false;
    pipeArray = [];
    useReversed = !useReversed;

    // Clear and reinitialize the canvas
    context.clearRect(0, 0, board.width, board.height);

    // Reinitialize game elements
    setupFlappyBirdGame(players[playerId].color);
    setInterval(placePipes, 1500);
    requestAnimationFrame(update);

    playerRef.update({
        x: birdX,
        y: birdY,
        gameOver: false
    });
}
// this doesn’t work, so every time the game is over, the players will need to
refresh the browser tab to play again, I will let this stay so I can update it
in the future if I want to.
unction moveBird(e) {
        if (players[playerId].gameOver) return;

      if (e.code == "Space" || e.code == "ArrowUp" || e.code == "KeyX") {

            velocityY = -6;
            jumped = true;
            if (gameOver && gameStarted) {
                resetGameState();
            } // doesn’t work
        }
    }
 function detectCollision(a, b) {
        return a.x < b.x + b.width &&
               a.x + a.width > b.x &&
               a.y < b.y + b.height &&
               a.y + a.height > b.y;
    } //a function that detects collision

// Initialize Firebase and start the game
    firebase.auth().onAuthStateChanged((user) => {
        if (user) {
            playerId = user.uid;
            playerRef = firebase.database().ref(`players/${playerId}`);
             // setting up a player in the Firebase database
            const name = getName();
            const color = randomFromArray(playerColors)

            // Determine the number of existing players
            const existingPlayers = Object.keys(players).length;

                // Assign a y position from the array based on the player's
index
            const assignedY = playerYpositions[existingPlayers %
playerYpositions.length];

            playerRef.set({
                id: playerId,
                name,
                color,
                x: 3,  // Fixed x position for all players
                y: assignedY, // Assign the calculated y position
                gameOver: false,
                ready: false,
                reversed: false
            }); //writing the player's data to the Realtime Db

            playerRef.onDisconnect().remove(); //removes the player's data if
the player disconnects

            initGame();

            // Set up the Flappy Bird game once Firebase data is loaded
            firebase.database().ref(`players`).once('value').then(snapshot =>
{
                players = snapshot.val() || {};
                if (players[playerId]) {
                    setupFlappyBirdGame(players[playerId].color);
                }
            });
        } else {
            // Handle user not signed in
            console.error("User not signed in");
        }
    });

    firebase.auth().signInAnonymously().catch((error) => {
        console.error("Authentication failed:", error.code, error.message);
    });
};
```

Conclusion

As we wrap up our development journey with the Multiplayer Flappy Bird game, I take pride in the achievements and insights gained throughout this project. My goal was to create an engaging and interactive multiplayer experience that challenges players while including real-time updates and dynamic gameplay.

I am thrilled with the progress made and the innovations implemented, from seamless player integration to real-time Firebase synchronization.

Thank you for following my development process. I invite you to call your friends and dive into the Multiplayer Flappy Bird game, test your skills, and enjoy competitive spirit of the game.