A
Mini-Project Report on

Flappy Bird Game

Submitted in partial fulfillment of the requirements
for the degree of
BACHELOR OF ENGINEERING
IN
Computer Science & Engineering
Artificial Intelligence & Machine Learning

by

Varun Raut (22106106)
Pratik Redekar (22106029)
Sarang Sawant (22106080)
Pranet Pednekar (22106098)

Under the guidance of

Prof. Monali Korde

Department of Computer Science & Engineering
(Artificial Intelligence & Machine Learning)
A. P. Shah Institute of Technology
G. B. Road, Kasarvadavali, Thane (W)-400615
University Of Mumbai
2023-2024

A. P. SHAH INSTITUTE OF TECHNOLOGY

CERTIFICATE

This is to certify that the project entitled “**Flappy Bird Game**” is a bonafide work of Varun Raut (22106106), Pratik Redekar (22106029), Sarang Sawant (22106080), Pranet Pednekar (22106098) submitted to the University of Mumbai in partial fulfillment of the requirement for the award of **Bachelor of Engineering in Computer Science & Engineering (Artificial Intelligence & Machine Learning).**

_________________	_________________
Prof. Monali Korde	Dr. Jaya Gupta
Mini Project Guide	Head of Department

A. P. SHAH INSTITUTE OF TECHNOLOGY

Project Report Approval

This Mini project report entitled "**Flappy Bird Game***"* by **Varun Raut, Pratik Redekar, Sarang Sawant** and **Pranet Pednekar** is approved for the degree of ***Bachelor of Engineering*** in ***Computer Science &Engineering***, (AIML) ***2023-24***.

External Examiner: ________________

Internal Examiner: ________________

Place: APSIT, Thane
Date:

Declaration

We declare that this written submission represents my ideas in my own words and where others' ideas or words have been included, I have adequately cited and referenced the original sources. I also declare that I have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in my submission. I understand that any violation of the above will be cause for disciplinary action by the Institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

Varun Raut	Pratik Redekar	Sarang Sawant	Pranet Pednekar
(22106106)	(22106029)	(22106080)	(22106098)

ABSTRACT

Players of the Flappy Bird game must guide a bird through a network of pipes by touching the screen to make the bird to flap its wings. This is a well-known 2D mobile game. The goal is to avoid obstacles while accumulating the greatest score possible. In order to give customers a fun and engaging gaming experience, this project tries to reproduce the well-known Flappy Bird game utilizing HTML5, CSS and JavaScript. It entails developing user-friendly controls, putting physics-based game mechanics into use. Through this project, we investigate the fundamentals of game development and demonstrate how to create a mobile game that can be played on a variety of devices, giving us new perspectives on game design, programming, and user interaction.

Index

CHAPTER 1
INTRODUCTION

1. INTRODUCTION

The Flappy Bird game project using HTML5, CSS, and JavaScript aims to recreate the addictive and challenging gameplay of the original mobile game, providing an enjoyable experience for web users. It demonstrates the power of web technologies in building engaging and interactive games that can be accessed across different platforms with ease. The project’s framework, HTML5, specifies the layout of the game’s webpage. Along with the necessary markup for user interface elements, it offers the canvas element for generating graphics and text. The game's visual styling and formatting is performed using CSS. It makes sure that the game has distinctive fonts, colors, and layout designs to make it aesthetically engaging. The project’s main component, JavaScript, is in charge of the game’s functionality and interactivity. It manages the game’s physics, user input, collision detection, and logic. The gameplay and game over screens are both managed by JavaScript. Our project brings this addictive gaming experience to the web, allowing users to play right in their browsers without the need for any additional installations.

Some of the key features of this project:

- The main gameplay elements of the original Flappy Bird are accurately recreated in this game, in which users manipulate the bird’s altitude by clicking or tapping while guiding it through a series of pipes with variable gaps.
- The project records the player’s score and updates it after successfully traversing through a set of pipes.

CHAPTER 2
LITERATURE SURVEY

2. LITERATURE SURVEY

2.1-HISTORY

Flappy Bird was first made available for mobile platforms in 2013. Dong Nguyen created the game. Its straightforward but addictive gameplay helped it become extremely popular. Following Flappy Bird's commercial success, a lot of developers started making HTML, CSS, and JavaScript versions of the game in the year 2013-14. These imitations tried to reproduce the same gameplay experience online. From 2014 onwards some developers made Flappy Bird available as open-source code in JavaScript, HTML, and CSS. For those trying to comprehend the rules of the game and its coding, these projects served as invaluable resources. A number of online tutorials and educational websites have released how-to manuals on how to make a Flappy Bird game using web technologies. These tutorials frequently take viewers step-by-step through the game development process, making it simple for newcomers to understand. Furthermore, many JavaScript game development frameworks and libraries, like Phaser, have grown in popularity over time. These resources and tools make it easier to develop games and let you make Flappy Bird-like titles. Projects related to Flappy Bird continue to receive contributions from the web development community. These contributions might boost the gameplay, graphics, and responsiveness on mobile devices. To create games with a more immersive user experience, developers have combined responsive design, web audio, and touch screen compatibility with Flappy Bird-style features. The development and evolution of HTML, CSS, and JavaScript-based web-based Flappy Bird games can be followed through online forums, tutorials, GitHub repositories, and web development communities. It has become a valuable resource for those interested in web game development as developers and educators have shared their experiences and knowledge in creating and improving these types of games over the years.

2.2-LITERATURE REVIEW

The use of web technologies like HTML5, CSS, and JavaScript to develop browser-based games is becoming more and more common, according to a number of sources in the game development literature. Independent game developers frequently choose these technologies because of their cross-platform compatibility and simplicity of use. Flappy Bird is frequently used as a classic example of a straightforward yet intensely addictive mobile game. Its simple mechanics are well-known to both researchers and developers, making it a great option for projects and educational settings. The topic of HTML5 canvas is frequently brought up when talking about game development. It is suitable for making 2D games like Flappy Bird because it enables dynamic rendering of graphics and animations within the web browser. JavaScript's use in the creation of video games is well-documented. Literature emphasizes its application in implementing user interaction, physics, collision detection, and game logic. For effective game development, game developers use JavaScript frameworks and libraries. Principles of responsive web design are essential when creating browser-based games because they allow for cross-platform compatibility. Several studies look into the psychology of game design and gamification strategies, both of which are crucial for developing captivating and compulsively playable gameplay experiences like Flappy Bird. In order to increase player engagement, games like Flappy Bird must incorporate game over screens, score tracking, and leaderboard systems. The literature survey reveals that recreating the Flappy Bird game project using HTML, CSS, and JavaScript aligns with established practices in game development. It showcases how web technologies have become a viable platform for building engaging and accessible games, and it emphasizes the importance of game design principles, user experience, and responsive design in creating successful browser-based games.

CHAPTER 3

PROBLEM STATEMENT

3. PROBLEM STATEMENT

The problem at hand is to develop a web-based Flappy Bird game using HTML, CSS, and JavaScript. The primary goal is to recreate the addictive and engaging gameplay experience of the original Flappy Bird mobile game for web users. Therefore the problem statement revolves around capturing the essence of the original while addressing the technical challenges of web development, cross-platform compatibility, user interface design, game logic, and accessibility. The successful completion of this project will result in an engaging and enjoyable gaming experience accessible to users across different web platforms.

CHAPTER 4

EXPERIMENTAL SETUP

4.1- HARDWARE SETUP

1. Computer: Any computer or device capable of running a modern web browser will suffice. This can be a desktop, laptop, tablet, or smartphone.

2. Input Devices: We will need a keyboard and mouse for controlling the game.

3. Display: Any screen capable of displaying a web page is suitable.

4.2- SOFTWARE SETUP

Web browser:

Testing and debugging: To check how your game appears to users, we can open the HTML file in a web browser. Real-time testing and debugging depend on this. Developer tools, which include functions like a console for debugging JavaScript, analyzing components, tracking network activity, and more, are also included with browsers.

Real-time updates: We can refresh the browser to see the updates right away as we make changes to the program in the code editor. We are able to perfect the game's appearance and functionality through this iterative approach. With responsive design, we can test how the game responds to multiple screen resolutions and sizes to make sure it is mobile-friendly and versatile.

Code Editor:

Writing Code: To organize the game, we will write HTML. To implement game logic, we will write JavaScript. To style the game pieces, we will write CSS. The availability of tools in code editors, such as syntax highlighting and auto-completion, makes it simpler to produce error-free code.

Managing Project Files: Using a code editor, we may create, arrange, and manage our project files. The majority of code editors let us open numerous files at once and offer a project view, making it simple to transition between various project components.

Debugging: To find and solve bugs in our code, code editors frequently provide integrated debugging tools or let us incorporate other debugging tools.

CHAPTER 5

PROPOSED SYSTEM AND IMPLEMENTATION

5.1- BLOCK DIAGRAM OF THE PROPOSED SYSTEM

Working of Program

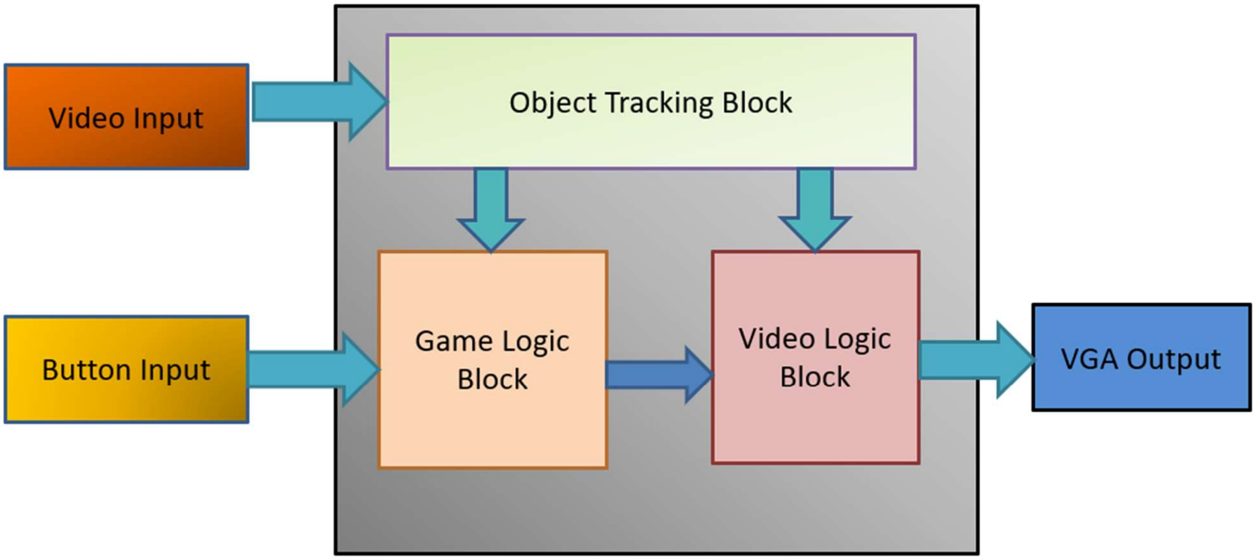


Figure: Block diagram of the program for Flappy bird game.

5.2- DESCRIPTION OF THE BLOCK DIAGRAM

Video Input Block:
This block represents the video input from an external source or a webcam feed.

Button Input Block:
This block represents the input from buttons or other physical controls. Players can use physical buttons or controls to interact with the game.

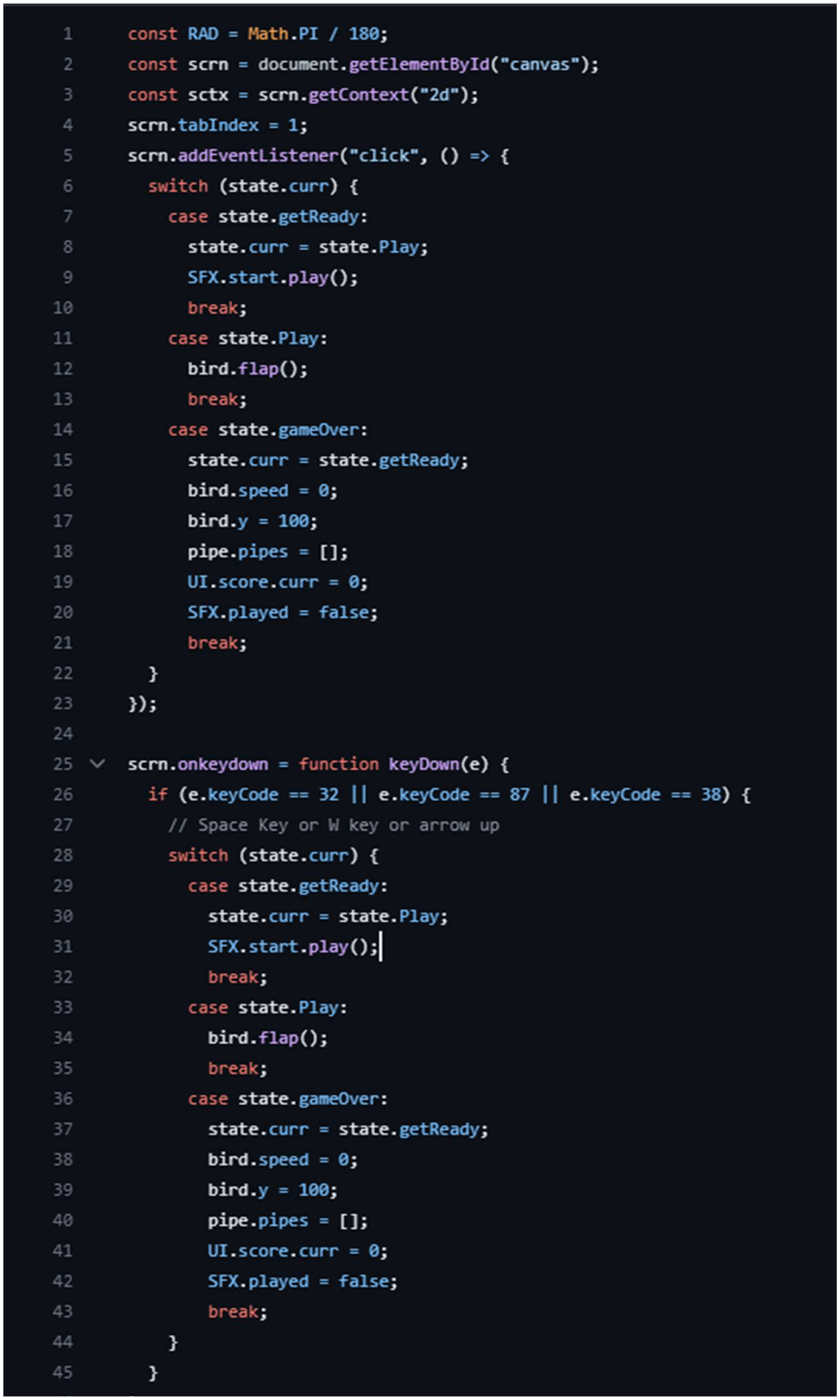
Object Tracking Block:
Object tracking is the process of identifying and following objects in a video stream.

Game Logic Block:
This block contains the core logic of the game, including the bird's movement, pipe generation, collision detection, scoring, and game over conditions. The Flappy Bird game logic determines how the game operates. It includes gravity for the bird, pipe generation, collision checks, and updating the game's score.

Video Logic Block:
Video logic manages the interaction between video input, object tracking, and the game. The video logic block processes the video input, and translates these inputs into game commands.

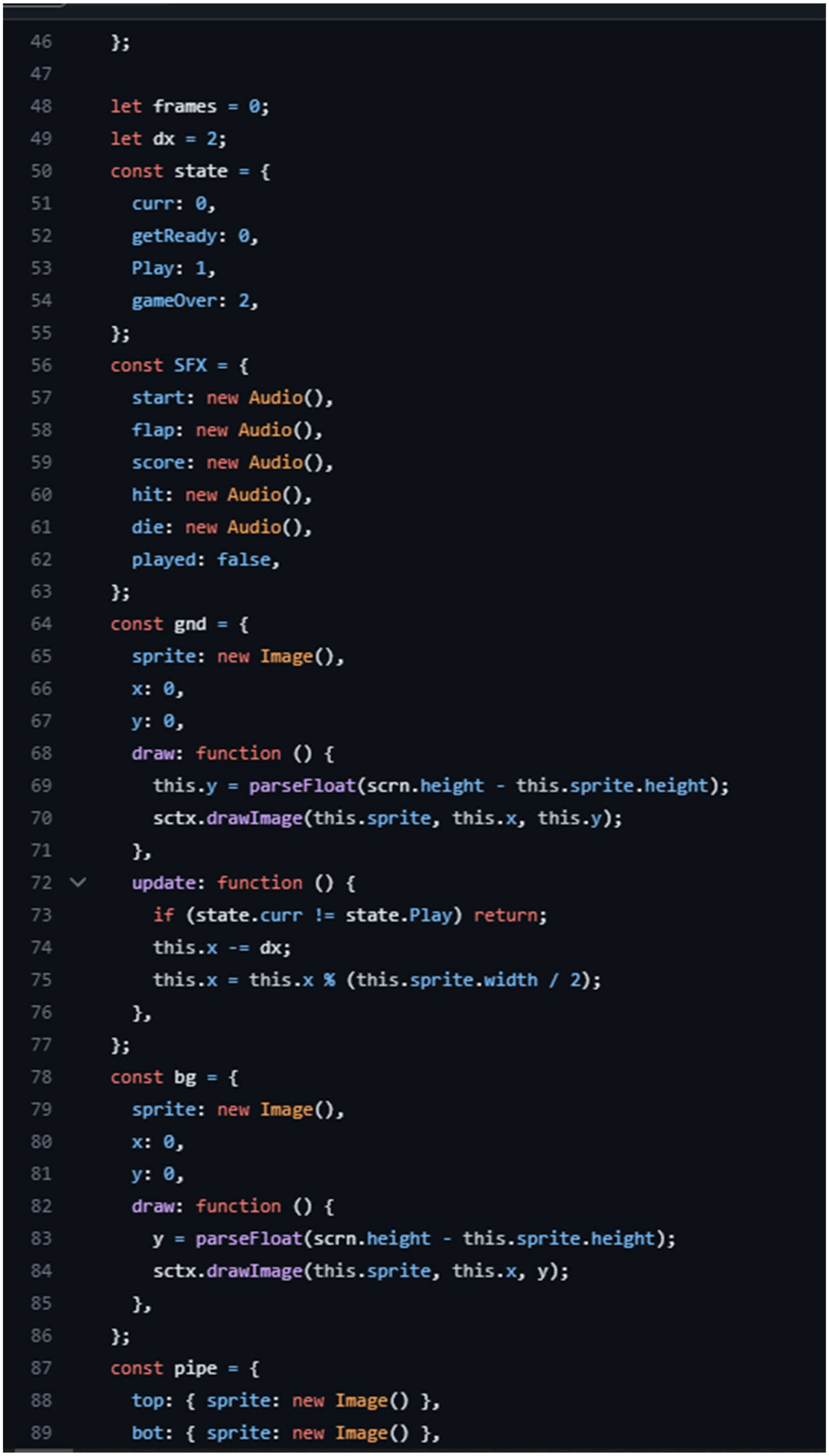
VGA Output Block:
The VGA output block represents the video output from the game, which can be displayed on a VGA monitor or screen. You can display the game's visuals, including the bird, pipes, and background, on a VGA monitor or compatible screen to provide a larger and more immersive gaming experience.

5.3- IMPLEMENTATION



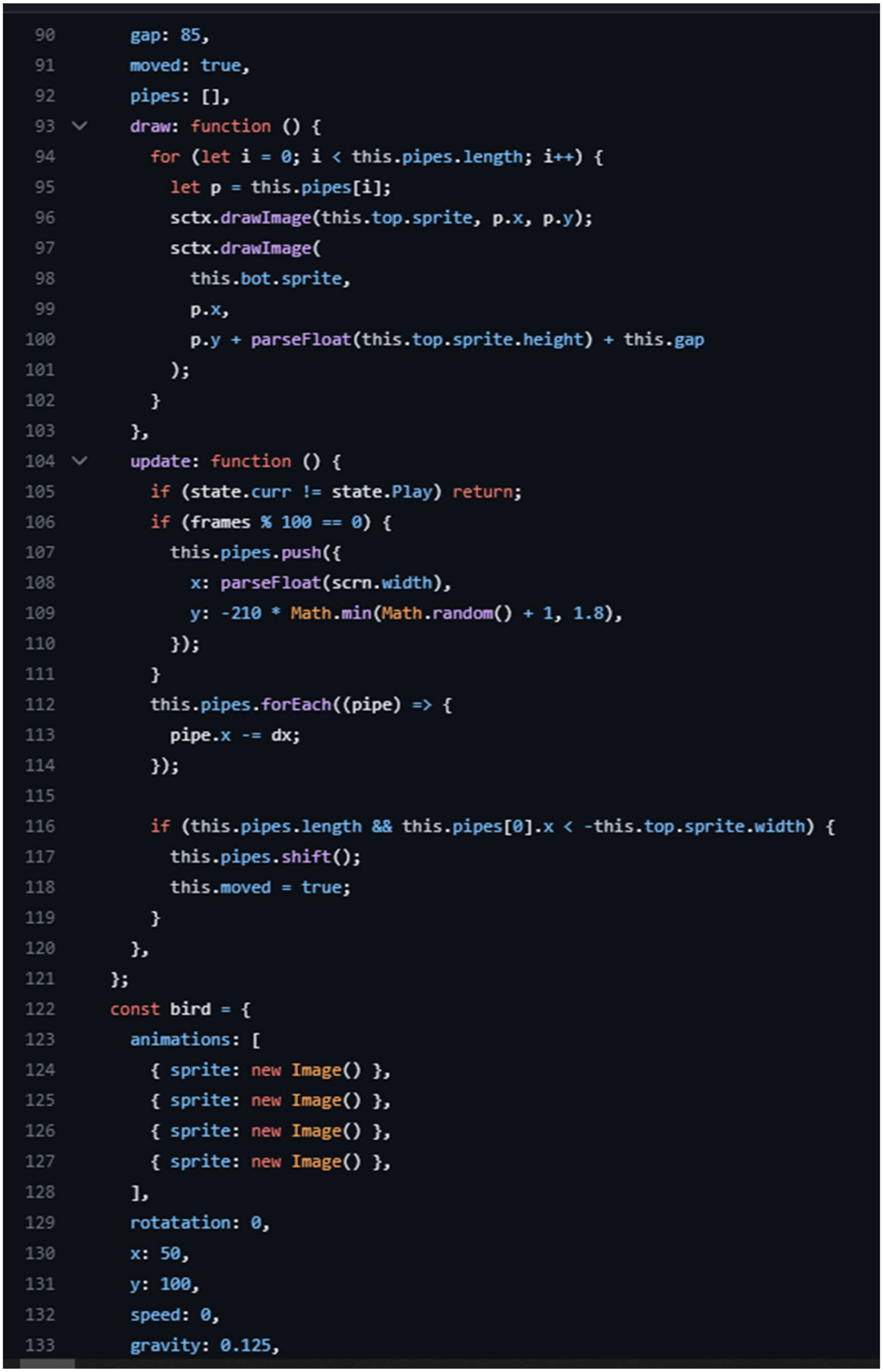
```
const RAD = Math.PI / 180;
const scrn = document.getElementById("canvas");
const sctx = scrn.getContext("2d");
scrn.tabIndex = 1;
scrn.addEventListener("click", () => {
  switch (state.curr) {
    case state.getReady:
      state.curr = state.Play;
      SFX.start.play();
      break;
    case state.Play:
      bird.flap();
      break;
    case state.gameOver:
      state.curr = state.getReady;
      bird.speed = 0;
      bird.y = 100;
      pipe.pipes = [];
      UI.score.curr = 0;
      SFX.played = false;
      break;
  }
});

scrn.onkeydown = function keyDown(e) {
  if (e.keyCode == 32 || e.keyCode == 87 || e.keyCode == 38) {
    // Space Key or W key or arrow up
    switch (state.curr) {
      case state.getReady:
        state.curr = state.Play;
        SFX.start.play();
        break;
      case state.Play:
        bird.flap();
        break;
      case state.gameOver:
        state.curr = state.getReady;
        bird.speed = 0;
        bird.y = 100;
        pipe.pipes = [];
        UI.score.curr = 0;
        SFX.played = false;
        break;
    }
  }
```



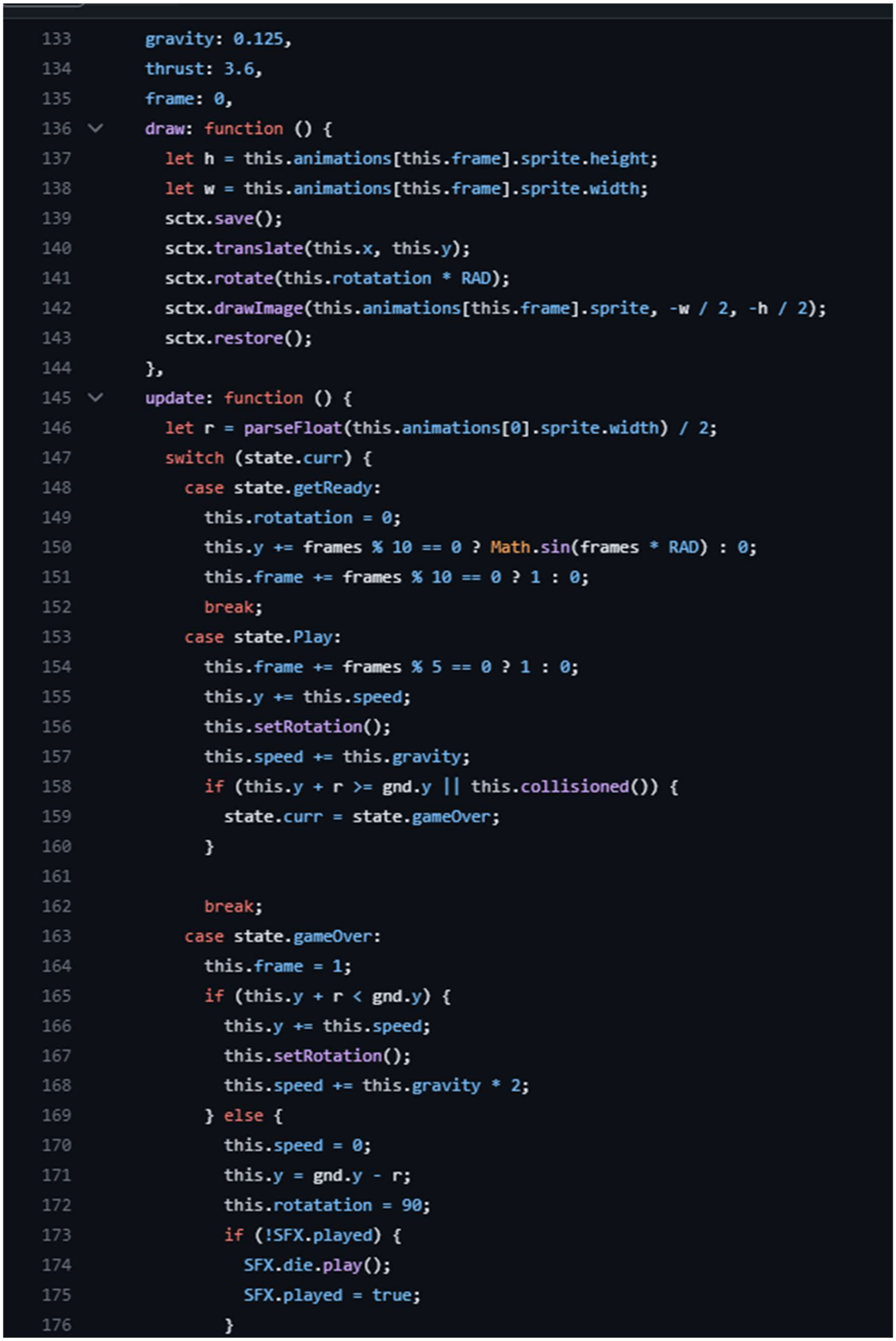
```
};

let frames = 0;
let dx = 2;
const state = {
  curr: 0,
  getReady: 0,
  Play: 1,
  gameOver: 2,
};
const SFX = {
  start: new Audio(),
  flap: new Audio(),
  score: new Audio(),
  hit: new Audio(),
  die: new Audio(),
  played: false,
};
const gnd = {
  sprite: new Image(),
  x: 0,
  y: 0,
  draw: function () {
    this.y = parseFloat(scrn.height - this.sprite.height);
    sctx.drawImage(this.sprite, this.x, this.y);
  },
  update: function () {
    if (state.curr != state.Play) return;
    this.x -= dx;
    this.x = this.x % (this.sprite.width / 2);
  },
};
const bg = {
  sprite: new Image(),
  x: 0,
  y: 0,
  draw: function () {
    y = parseFloat(scrn.height - this.sprite.height);
    sctx.drawImage(this.sprite, this.x, y);
  },
};
const pipe = {
  top: { sprite: new Image() },
  bot: { sprite: new Image() },
```



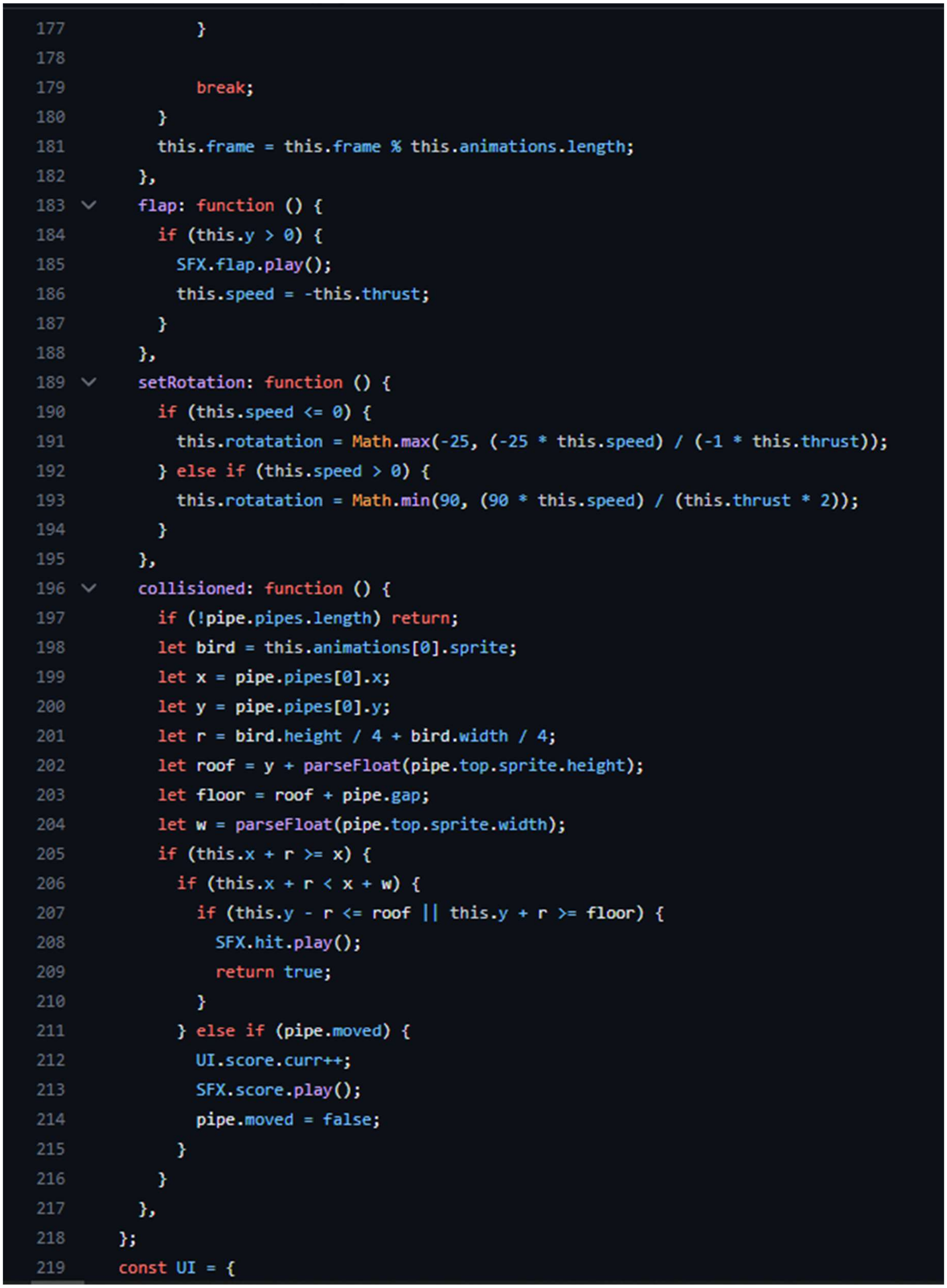
```
  gap: 85,
  moved: true,
  pipes: [],
  draw: function () {
    for (let i = 0; i < this.pipes.length; i++) {
      let p = this.pipes[i];
      sctx.drawImage(this.top.sprite, p.x, p.y);
      sctx.drawImage(
        this.bot.sprite,
        p.x,
        p.y + parseFloat(this.top.sprite.height) + this.gap
      );
    }
  },
  update: function () {
    if (state.curr != state.Play) return;
    if (frames % 100 == 0) {
      this.pipes.push({
        x: parseFloat(scrn.width),
        y: -210 * Math.min(Math.random() + 1, 1.8),
      });
    }
    this.pipes.forEach((pipe) => {
      pipe.x -= dx;
    });

    if (this.pipes.length && this.pipes[0].x < -this.top.sprite.width) {
      this.pipes.shift();
      this.moved = true;
    }
  },
};
const bird = {
  animations: [
    { sprite: new Image() },
    { sprite: new Image() },
    { sprite: new Image() },
    { sprite: new Image() },
  ],
  rotatation: 0,
  x: 50,
  y: 100,
  speed: 0,
  gravity: 0.125,
```



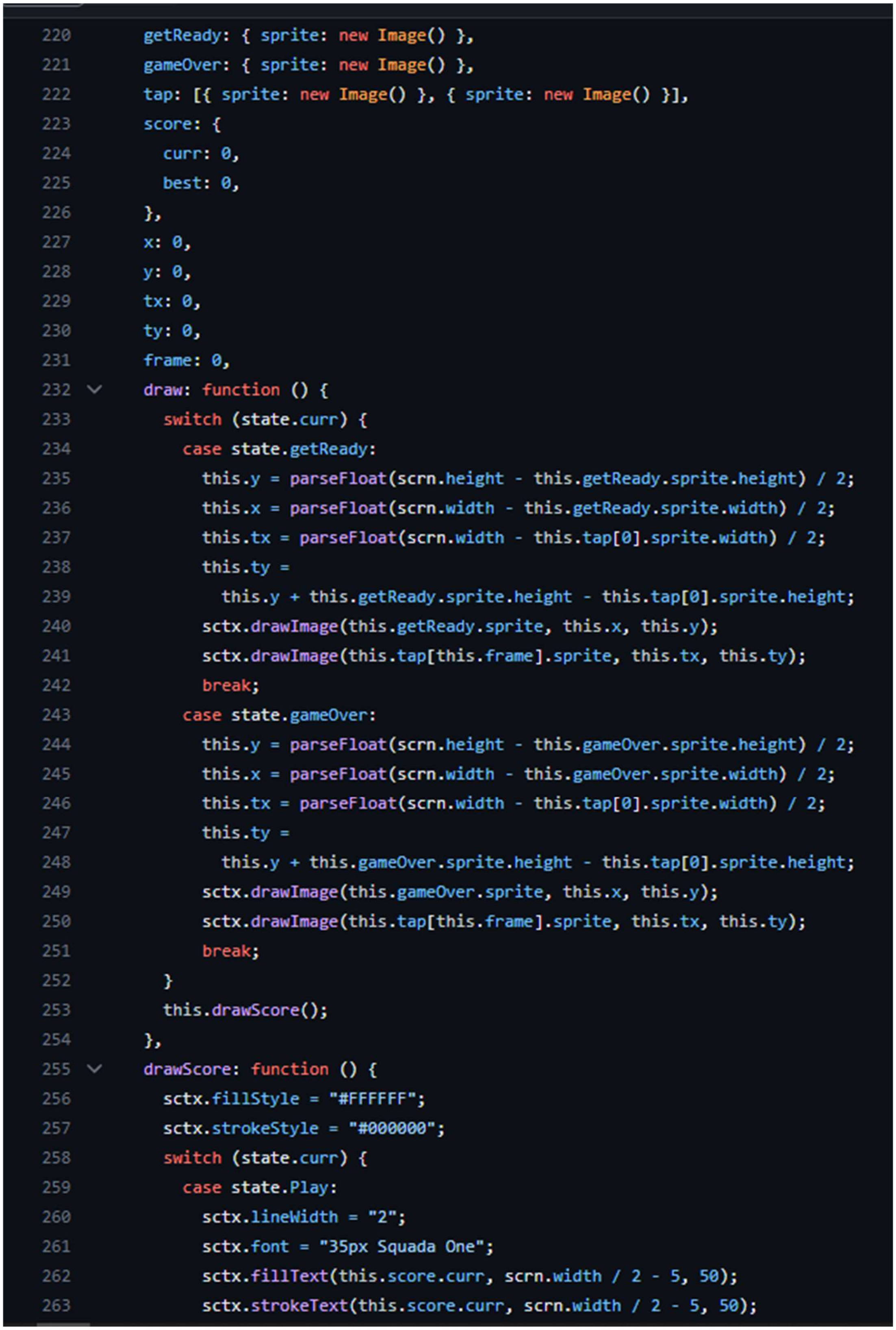
```
    gravity: 0.125,
    thrust: 3.6,
    frame: 0,
    draw: function () {
      let h = this.animations[this.frame].sprite.height;
      let w = this.animations[this.frame].sprite.width;
      sctx.save();
      sctx.translate(this.x, this.y);
      sctx.rotate(this.rotatation * RAD);
      sctx.drawImage(this.animations[this.frame].sprite, -w / 2, -h / 2);
      sctx.restore();
    },
    update: function () {
      let r = parseFloat(this.animations[0].sprite.width) / 2;
      switch (state.curr) {
        case state.getReady:
          this.rotatation = 0;
          this.y += frames % 10 == 0 ? Math.sin(frames * RAD) : 0;
          this.frame += frames % 10 == 0 ? 1 : 0;
          break;
        case state.Play:
          this.frame += frames % 5 == 0 ? 1 : 0;
          this.y += this.speed;
          this.setRotation();
          this.speed += this.gravity;
          if (this.y + r >= gnd.y || this.collisioned()) {
            state.curr = state.gameOver;
          }

          break;
        case state.gameOver:
          this.frame = 1;
          if (this.y + r < gnd.y) {
            this.y += this.speed;
            this.setRotation();
            this.speed += this.gravity * 2;
          } else {
            this.speed = 0;
            this.y = gnd.y - r;
            this.rotatation = 90;
            if (!SFX.played) {
              SFX.die.play();
              SFX.played = true;
            }
```

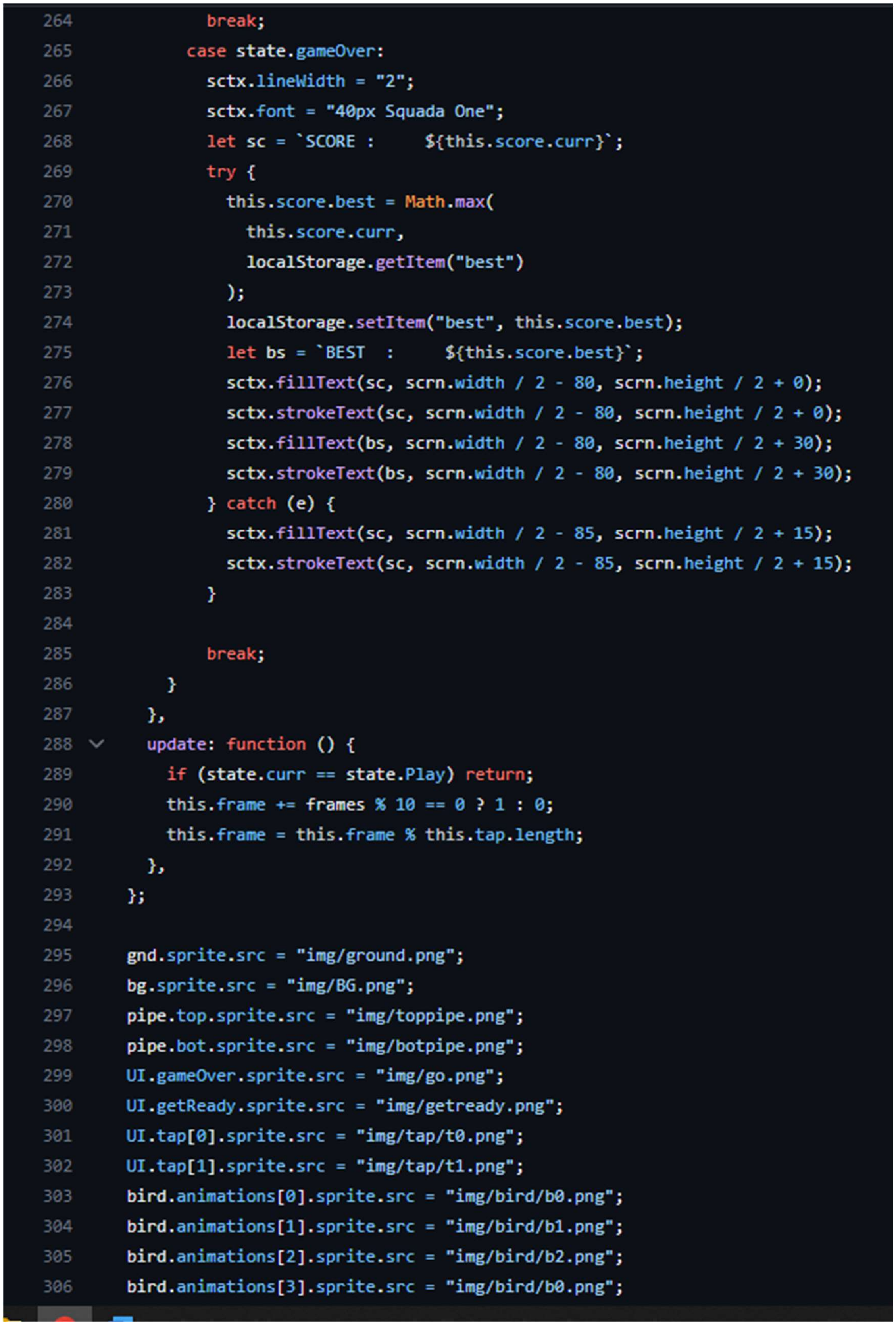


```
        }

        break;
    }
    this.frame = this.frame % this.animations.length;
  },
  flap: function () {
    if (this.y > 0) {
      SFX.flap.play();
      this.speed = -this.thrust;
    }
  },
  setRotation: function () {
    if (this.speed <= 0) {
      this.rotatation = Math.max(-25, (-25 * this.speed) / (-1 * this.thrust));
    } else if (this.speed > 0) {
      this.rotatation = Math.min(90, (90 * this.speed) / (this.thrust * 2));
    }
  },
  collisioned: function () {
    if (!pipe.pipes.length) return;
    let bird = this.animations[0].sprite;
    let x = pipe.pipes[0].x;
    let y = pipe.pipes[0].y;
    let r = bird.height / 4 + bird.width / 4;
    let roof = y + parseFloat(pipe.top.sprite.height);
    let floor = roof + pipe.gap;
    let w = parseFloat(pipe.top.sprite.width);
    if (this.x + r >= x) {
      if (this.x + r < x + w) {
        if (this.y - r <= roof || this.y + r >= floor) {
          SFX.hit.play();
          return true;
        }
      } else if (pipe.moved) {
        UI.score.curr++;
        SFX.score.play();
        pipe.moved = false;
      }
    }
  },
};
const UI = {
```



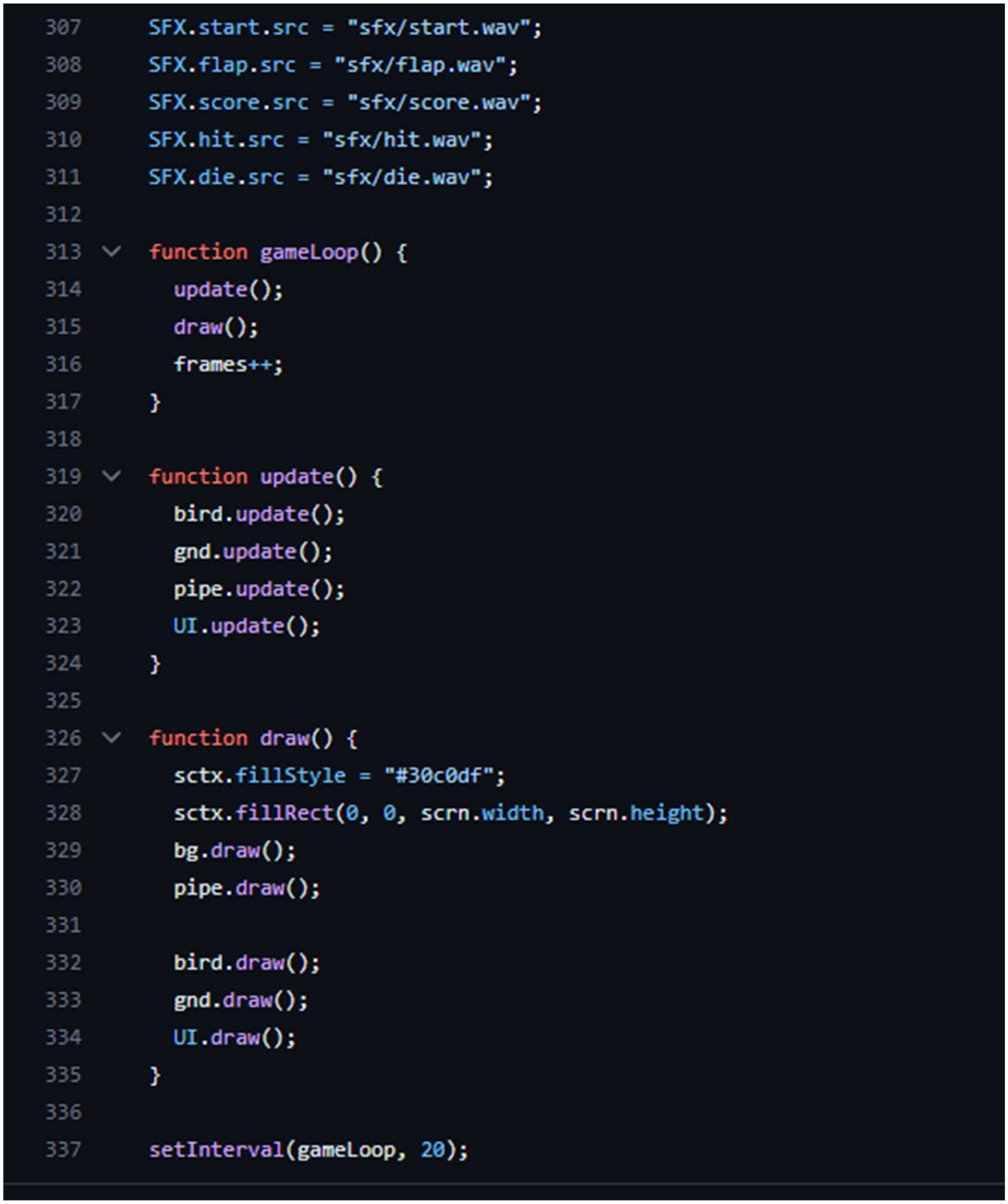
```
  getReady: { sprite: new Image() },
  gameOver: { sprite: new Image() },
  tap: [{ sprite: new Image() }, { sprite: new Image() }],
  score: {
    curr: 0,
    best: 0,
  },
  x: 0,
  y: 0,
  tx: 0,
  ty: 0,
  frame: 0,
  draw: function () {
    switch (state.curr) {
      case state.getReady:
        this.y = parseFloat(scrn.height - this.getReady.sprite.height) / 2;
        this.x = parseFloat(scrn.width - this.getReady.sprite.width) / 2;
        this.tx = parseFloat(scrn.width - this.tap[0].sprite.width) / 2;
        this.ty =
          this.y + this.getReady.sprite.height - this.tap[0].sprite.height;
        sctx.drawImage(this.getReady.sprite, this.x, this.y);
        sctx.drawImage(this.tap[this.frame].sprite, this.tx, this.ty);
        break;
      case state.gameOver:
        this.y = parseFloat(scrn.height - this.gameOver.sprite.height) / 2;
        this.x = parseFloat(scrn.width - this.gameOver.sprite.width) / 2;
        this.tx = parseFloat(scrn.width - this.tap[0].sprite.width) / 2;
        this.ty =
          this.y + this.gameOver.sprite.height - this.tap[0].sprite.height;
        sctx.drawImage(this.gameOver.sprite, this.x, this.y);
        sctx.drawImage(this.tap[this.frame].sprite, this.tx, this.ty);
        break;
    }
    this.drawScore();
  },
  drawScore: function () {
    sctx.fillStyle = "#FFFFFF";
    sctx.strokeStyle = "#000000";
    switch (state.curr) {
      case state.Play:
        sctx.lineWidth = "2";
        sctx.font = "35px Squada One";
        sctx.fillText(this.score.curr, scrn.width / 2 - 5, 50);
        sctx.strokeText(this.score.curr, scrn.width / 2 - 5, 50);
```



```
        break;
      case state.gameOver:
        sctx.lineWidth = "2";
        sctx.font = "40px Squada One";
        let sc = `SCORE :     ${this.score.curr}`;
        try {
          this.score.best = Math.max(
            this.score.curr,
            localStorage.getItem("best")
          );
          localStorage.setItem("best", this.score.best);
          let bs = `BEST  :     ${this.score.best}`;
          sctx.fillText(sc, scrn.width / 2 - 80, scrn.height / 2 + 0);
          sctx.strokeText(sc, scrn.width / 2 - 80, scrn.height / 2 + 0);
          sctx.fillText(bs, scrn.width / 2 - 80, scrn.height / 2 + 30);
          sctx.strokeText(bs, scrn.width / 2 - 80, scrn.height / 2 + 30);
        } catch (e) {
          sctx.fillText(sc, scrn.width / 2 - 85, scrn.height / 2 + 15);
          sctx.strokeText(sc, scrn.width / 2 - 85, scrn.height / 2 + 15);
        }

        break;
    }
  },
  update: function () {
    if (state.curr == state.Play) return;
    this.frame += frames % 10 == 0 ? 1 : 0;
    this.frame = this.frame % this.tap.length;
  },
};

gnd.sprite.src = "img/ground.png";
bg.sprite.src = "img/BG.png";
pipe.top.sprite.src = "img/toppipe.png";
pipe.bot.sprite.src = "img/botpipe.png";
UI.gameOver.sprite.src = "img/go.png";
UI.getReady.sprite.src = "img/getready.png";
UI.tap[0].sprite.src = "img/tap/t0.png";
UI.tap[1].sprite.src = "img/tap/t1.png";
bird.animations[0].sprite.src = "img/bird/b0.png";
bird.animations[1].sprite.src = "img/bird/b1.png";
bird.animations[2].sprite.src = "img/bird/b2.png";
bird.animations[3].sprite.src = "img/bird/b0.png";
```



```
SFX.start.src = "sfx/start.wav";
SFX.flap.src = "sfx/flap.wav";
SFX.score.src = "sfx/score.wav";
SFX.hit.src = "sfx/hit.wav";
SFX.die.src = "sfx/die.wav";

function gameLoop() {
  update();
  draw();
  frames++;
}

function update() {
  bird.update();
  gnd.update();
  pipe.update();
  UI.update();
}

function draw() {
  sctx.fillStyle = "#30c0df";
  sctx.fillRect(0, 0, scrn.width, scrn.height);
  bg.draw();
  pipe.draw();

  bird.draw();
  gnd.draw();
  UI.draw();
}

setInterval(gameLoop, 20);
```

CHAPTER 6
CONCLUSION

6- CONCLUSION

The Flappy Bird project is an educational and engaging exercise in web development and game design, leveraging HTML, JavaScript, and CSS to create a simplified version of the popular Flappy Bird game. This project is a fantastic way to apply our web development skills and delve into the world of game development. It offers a tangible example of how HTML, JavaScript, and CSS can be used to create a simple yet engaging game. This experience equips us with valuable insights and practical knowledge, which can be applied to more complex game projects and other web development endeavors.

REFERENCES

REFERENCES

[1] Sung K, Pavleas J, Arnez F, Pace J, Sung K, Pavleas J, Arnez F, Pace J. Introducing 2D Game Engine Development with JavaScript. Build Your Own 2D Game Engine and Create Great Web Games: Using HTML5, JavaScript, and WebGL. 2015:1-3.

[2] Bunyan, Karl. Build an HTML5 Game: A Developer's Guide with CSS and JavaScript. No Starch Press, 2015.

[3] Burchard, Evan. The Web Game Developer's Cookbook: Using JavaScript and HTML5 to Develop Games. Addison-Wesley, 2013.