PAC-MAN

Revolutionary Arcade Game

Pritam Gurung –

Word Count: 4223

Summary

The Pacman X project was created to showcase foundational web development skills using JavaScript and HTML. The primary objective was to recreate the classic Pacman game with modern web technologies, delivering an interactive browser-based experience.

Developed using knowledge from my foundation year supplemented with extensive research and self-learning, the project utilised JavaScript for game logic, animations, and user interaction. Key functionalities like movement, collision detection, and scoring were coded to ensure smooth gameplay. The Pacman and Ghost classes capture core mechanics, defining behaviours such as movement patterns collision detection, and state changes.

A significant feature of this project is the implementation of Dijkstra's algorithm to enhance AI of the ghosts. This algorithm calculates the shortest path for the ghosts, making their movement more strategic and challenging for players.

HTML was used to structure the game interface, incorporating the canvas element to render the visuals. The game environment, including the maze and object, was possible through JavaScript. Sound effects were added to add nostalgia to the game, and the performance optimisation technique ensured smooth game operation.

In summary, Pacman X demonstrates the potential of JavaScript and HTML in creating rich interactive web-based games. It pays homage to a classic arcade showcasing modern web technology capabilities, making it smooth and enjoyable to play in a browser.

Table of Contents

A list of Tables and Illustration

Tables

Illustration

1. Introduction

1.1 History of Pacman

Pacman was developed by Namco and first released in Japan on May 22, 1980, and quickly became a cultural phenomenon. The game's designer, Toru Iwatani, aimed to create a game that would appeal to a broad audience, including women and families. Unlike the predominant shooting games of the era, Pacman introduced non-violent, maze-based gameplay mechanics that were both innovative and engaging.

The game features a yellow, pie-shaped character named Pacman who navigates a maze eating pellets while being pursued by four ghosts: Blinky, Pinky, Inky, and Clyde the objective is to clear the maze of all pellets while avoiding the ghosts, which become more aggressive as the game progresses. Power pellets, located at the corners of the maze, temporally turn the ghost blue, allowing Pacman to eat them for bonus points.

1.2 Significance in the Gaming Industry

Pacman revolutionised the arcade game industry and has had a lasting impact on video game design and culture. Its success was unprecedented, leading to a surge in arcade game popularity and contributing to the golden age of arcade video games. Pacman's appeal was diverse: it featured simple yet addictive gameplay, colourful graphics, and a memorable cast of characters. Additionally, the game stood out for its non-violent nature, making it accessible to a wider audience.

1.3 Aims and Objectives

The primary aim of this project is to develop a fully functional and engaging Pacman game that captures the essence of the original while incorporating modern programming techniques. The specific objectives of the project are:

1. Develop Pacman's Movement and Controls:
- Implement smooth and responsive control for Pacman using keyboard input.
- Ensure accurate collision detection with maze walls and pellets.
2. Implement Ghost AI:
- Develop unique behaviours for each ghost to mimic their original patterns.
- Use Dijkstra's algorithm to enhance ghost pathfinding, making their movements more strategic and challenging for players.
3. Design and Implement Power Pellets:
- Include power pellets that allow Pacman to temporarily eat the ghosts.
- Implement the visual and gameplay effects associated with power pellets such as turning ghosts blue and the corresponding score increase.

4. Create the Iconic Map:
- Design the game map to closely resemble the original Pacman maze.
- Ensure all elements, such as pellets, power pellets, and ghost starting positions are accurately placed.
5. Animate Pacman and Ghosts:
- Develop smooth animations for Pacman's movement and eating actions.
- Animate ghost movements and state changes 9e.g, turning blue when power pellets are consumed)
6. Enhance Game Quality and Experience:
- Incorporate the iconic "waka waka" sound when Pacman eats pellets to enhance auditory feedback.
- Optimize game performance to ensure smooth gameplay without lag or glitches.
- Try including other background sounds to enhance the gameplay experience.

1.4 Project Motivation

The choice to recreate Pacman for this project is driven by several factors. First, I had the desire to recreate simple arcade games to broaden my horizons and deepen my knowledge and practical experience in computer science, particularly within the realm of game development. Recreating simple arcade games such as Pacman, allows me to diversify my skills and gain a better understanding of the game mechanics and design principles. This experience will not only enhance my current capabilities but also significantly enrich my future academic studies and career progression in computer science. Second, the development of a Pacman game encompasses various fundamental aspects of programming and game design, including artificial intelligence for ghost behaviours, collision detection, and user interface design. This project provides an opportunity to apply my knowledge that I have gained from my foundation year at Sussex practically and engagingly, while also offering a challenging yet manageable scope for a student project.

By developing Pacman as my project, this falls under the category of several computing concepts such as:

- Web Development: Utilising HTML for the structure of the game, integrating it with other web technologies.
- Game Design: Creating and managing game mechanics, including user input for Pacman's movement, collision detection, and game state transition.
- Artificial Intelligence: Implementing ghost behaviours using Dijkstra's algorithm to create a strategic gameplay experience.

1.5 Educational and Practical Benefits

Recreating Pacman allows for a comprehensive exploration of game development and by working on this project, I will gain hands-on experience in

- Web Development: Using HTML to structure and present the game
- Programming: Writing efficient and clean codes to handle various game functionalities.
- Software Design: Structuring the game's architecture to be modular and maintainable.
- Problem-Solving: Tackling challenges related to game logic such as ghost AI and collision detection.
- Project Management: Planning and executing a project from inception completion, including documenting the process and presenting the final product.

This project not only reinforces the technical skills I have learnt during my foundation year, but allows me to demonstrate critical thinking, creativity and time management.

1.6 Overview of the Project

This project aims to develop a fully functional Pacman game, demonstrating the application of computing skills in game development. The game will include key features such as

- Pacman's movement and control.
- Ghost AI behaviour with Dijkstra algorithm.
- Power-ups and scoring system as well as displaying Pacman lives.
- An iconic maze layout with pellets and power pellets
- Pacman animation movement and the "waka waka" sound effect.

The project will follow a structured development process, including requirements analysis, design, implementation, and testing. The final deliverable will be a playable Pacman game, accompanied by a detailed report documenting the development process and the outcome of my finished game.

Background Research

2.1 Introduction to Background Research

This section of the report explores essential technologies, techniques and methodologies required to develop a fully functional and engaging Pacman game we know and love. It covers the development frameworks, sprite and animation techniques, artificial intelligence, web development integration, sound design, and performance optimization.

2.2 Game Development Frameworks

To develop Pacman, HTML and JavaScript were used. HTML provides the structure and canvas element for rendering the game, while JavaScript handles the game logic, animations, and interactions. These programming languages were chosen for their support in making web-based game development possible and its ease of use.

The project set-up involves creating an HTML file that links to the necessary JavaScript files which are game.js, pacman.js, and ghost.js and it includes the canvas elements where the game is rendered which can be shown in Figure 1.

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta http-equiv="X-UA-compatible" content="IE=edge" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Pacman</title>
  </head>

  <body
    style="
      margin: 0px;
      background-color: black;
      overflow-y: hidden;
      overflow-x: hidden;
    "
  >
    <audio id="eat-sound" src="sounds/waka.wav"></audio>

    <canvas id="canvas" width="500" height="500"></canvas>
    <div style="display: none">
      <img id="animations" src="assets/animations.gif" alt="" />
      <img id="ghosts" src="assets/ghost.png" alt="" />
    </div>
    <script src="ghost.js"></script>
    <script src="pacman.js"></script>
    <script src="game.js"></script>
  </body>
</html>
```

Figure 1: HTML setup for Pacman game.

2.3 Sprite and Animation Techniques

Creating and animating sprites is crucial for the visual appeal of the game as it's one of the core aspects of Pacman that makes it truly alive and iconic. The sprites for Pacman and the ghosts were designed to replicate the classic arcade game's look and feel. The sprites were created as individual frames and then animated to provide smooth motion.

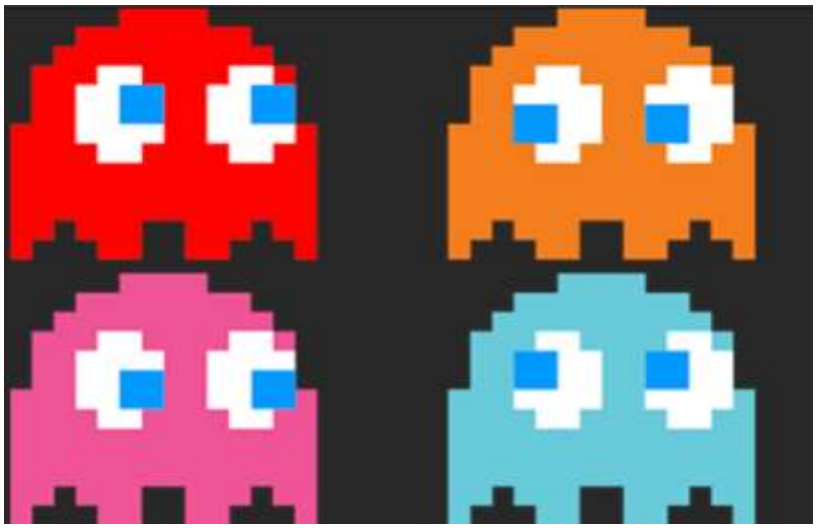


Figure 2: Sprite design for the ghosts in Pacman featuring Blinky (red), Pinky (pink), Inky (cyan) and Clyde (orange).

Pacman's animations is created using a sequence of frames, a technique also employed by Walt Disney in their cartoon animations. This method stimulates Pacman's mouth opening and closing as he moves, resulting in a smooth and visually appealing animation of his movement.

Figure 3: Animation frames for Pacman's movement.

The Pacman class in Pacman.js handles the movement and animation of the Pacman character. The following code snippet shows how the animation frames are cycled to create the movement effects.

```
changeAnimation() {
  this.currentFrame =
    this.currentFrame == this.frameCount ? 1 : this.currentFrame + 1;
}
```

```
class Pacman {
  constructor(x, y, width, height, speed) {
    this.x = x;
    this.y = y;
    this.width = width;
    this.height = height;
    this.speed = speed;
    this.direction = DIRECTION_RIGHT;
    this.nextDirection = this.direction;
    this.currentFrame = 1;
    this.frameCount = 7;

    setInterval(() => {
      this.changeAnimation();
    }, 100);
  }
```

```
draw() {
  canvasContext.save();
  canvasContext.translate(
    this.x + oneBlockSize / 2,
    this.y + oneBlockSize / 2
  );
  canvasContext.rotate((this.direction * 90 * Math.PI) / 180);

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

  canvasContext.drawImage(
    pacmanFrames,
    (this.currentFrame - 1) * oneBlockSize,
    0,
    oneBlockSize,
    oneBlockSize,
    this.x,
    this.y,
    this.width,
    this.height
  );

  canvasContext.restore();
}
```

Figure 4: Code snippets for Pacman Animation

2.4 Artificial Intelligence in Games

Artificial Intelligence (AI) is a critical component in creating a fun strategic and engaging Pacman Game. The classic Pacman game features sophisticated AI behaviour for the ghosts, including chase, and scatter, and hopefully implement frightened modes. Each ghost follows a specific algorithm to create a unique gameplay experience.

To enhance the ghost AI, I implemented Dijkstra's algorithm for pathfinding within the maze. Dijkstra's algorithm is a graph search algorithm that finds the shortest path between nodes, which in this case, are the position within the maze. This approach ensures that the ghosts can strategically navigate the maze to chase Pacman shown in Figure 5.

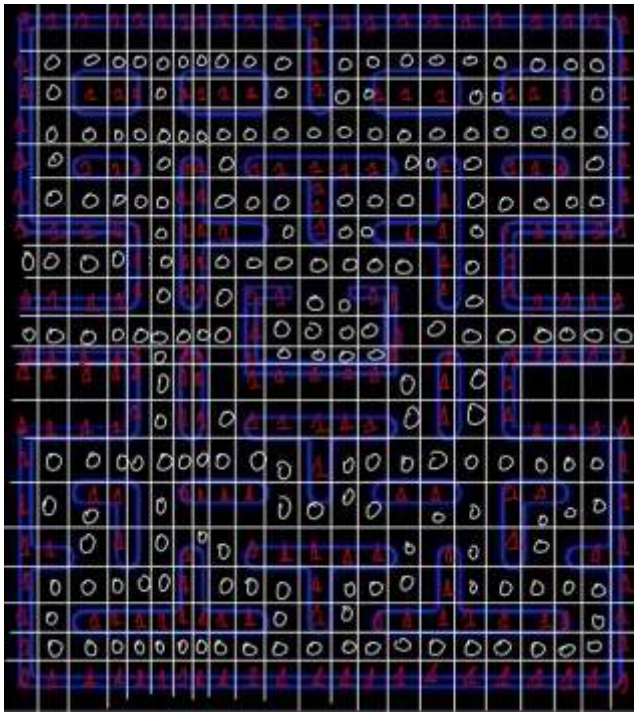


Figure 5: Maze layout with nodes for implementing Dijkstra's algorithm.

2.5 Flowchart of Ghost AI behaviour

To further clarify the ghost AI behaviour, the following flowchart illustrates the decision-making process used by the ghosts to chase Pacman.

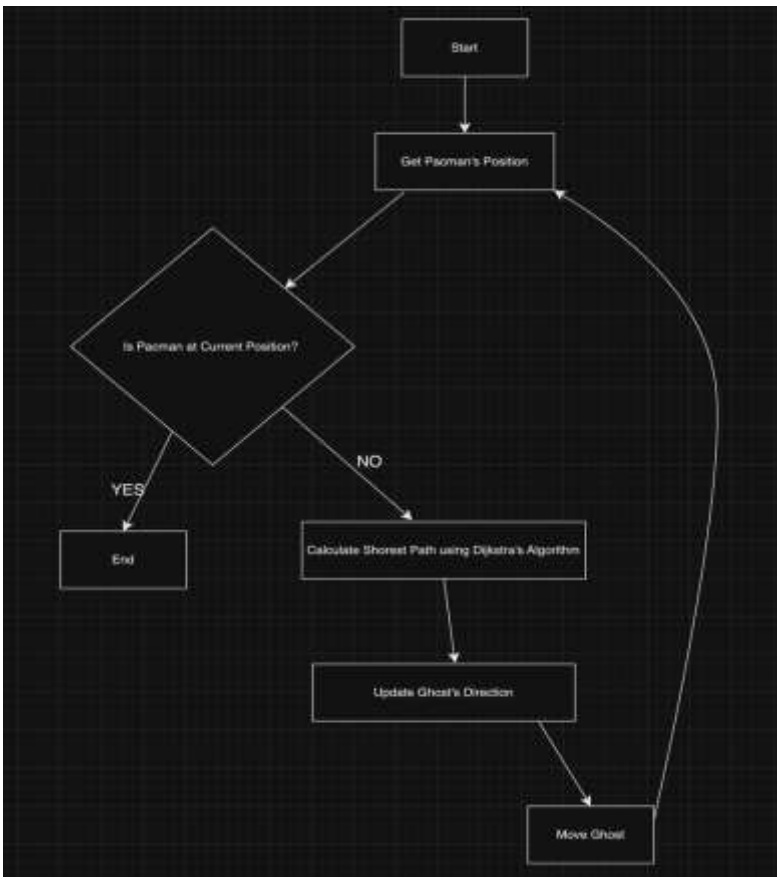


Figure 6: Flowchart of Ghost AI behaviour.

The Ghost class in ghost.js includes methods for pathfinding and movement which utilises Dijkstra's algorithm shown in Figure 7.

```
class Ghost {
  constructor(
    x,
    y,
    width,
    height,
    speed,
    imageX,
    imageY,
    imageWidth,
    imageHeight,
    range
  ) {
    this.x = x;
    this.y = y;
    this.width = width;
    this.height = height;
    this.speed = speed;
    this.direction = DIRECTION_RIGHT;
    this.imageX = imageX;
    this.imageY = imageY;
    this.imageHeight = imageHeight;
    this.imageWidth = imageWidth;
    this.range = range;
    this.randomTargetIndex = parseInt(
      Math.random() * randomTargetsForGhosts.length
    );
```

```
calculateNewDirection(map, destX, destY) {
  let mp = [];
  for (let i = 0; i < map.length; i++) {
    mp[i] = map[i].slice();
  }

  let queue = [
    {
      x: this.getMapX(),
      y: this.getMapY(),
      moves: [],
    },
  ];

  while (queue.length > 0) {
    let poped = queue.shift();
    if (poped.x == destX && poped.y == destY) {
      return poped.moves[0];
    } else {
      mp[poped.y][poped.x] = 1;
      let neighborList = this.addNeighbors(poped, mp);
      for (let i = 0; i < neighborList.length; i++) {
        queue.push(neighborList[i]);
      }
    }
  }

  return DIRECTION_UP;
}
```

Figure 7: Code snippets for Ghost AI behaviour using Dijkstra's algorithm.

2.6 Web Development for Game Integration

HTML and CSS were used to structure and style the game interface, providing a visually appealing and responsive design. JavaScript handles the game logic, animations, and interactions. The HTML file includes the necessary canvas elements for rendering the game and links to the JavaScript files where the game logic is implemented.

2.7 Sound Design and Implementation

Sound effects play a crucial role in enhancing the gaming experience. The iconic "waka waka" sound was integrated into the game to provide auditory feedback and increase players immersion in the game The eat method in pacman.js demonstrates how the sound effect is triggered when Pacman eats a pellet shown in Figure 8.

```
eat() {
  const eatSound = document.getElementById("eat-sound");
  for (let i = 0; i < map.length; i++) {
    for (let j = 0; j < map[0].length; j++) {
      if (map[i][j] == 2 && this.getMapX() == j && this.getMapY() == i) {
        map[i][j] = 3;
        score++;
        eatSound.play();
      }
    }
  }
}
```

Figure 8: Code snippet for playing sound effects when Pacman eats a pellet.

2.8 Optimising Game for Performance

Optimising game performance ensures a smooth and responsive experience for players. Strategies include efficient rendering techniques, minimising loads, and optimising asset sizes. Regular testing and debugging are essential to identifying and fixing issues that may affect performance, ensuring the game runs efficiently and it's operating as expected.

The game.js file sets up the game loop and includes a method for rendering and updating the game state shown in Figure 9.

```
let gameLoop = () => {
  draw();
  update();
};

let update = () => {
  pacman.moveProcess();
  pacman.eat();
  for (let i = 0; i < ghosts.length; i++) {
    ghosts[i].moveProcess();
  }

  if (pacman.checkGhostCollision()) {
    console.log("hit");
    restartGame();
  }
  if (score >= foodCount) {
    drawWin();
    clearInterval(gameInterval);
  }
};
```

Figure 9: Code snippet for the game loop and state updates.

3. Requirements Analysis and Specification

3.1 Introduction

This section outlines the functional and non-functional requirements for the Pacman project. Its purpose is to clearly define the expected functionalities, performance standards, and design constraints of the game, ensuring it aligns with user needs and technical specifications.

3.2 Functional Requirements

Functional requirements point out what the system should do. It describes the interaction between the system and its users and other systems.

3.3 Pacman Movement

Requirements: Pacman must move in four directions (up, down, left, right) based on user input (arrow keys or WASD)

Implementation: Pacman continues moving in the last direction pressed until another direction is chosen. This can be shown in Figure 1 of what makes the movement possible.

```
  setTimeout(() => {
    if (k == 37 || k == 65) {
      // left
      pacman.nextDirection = DIRECTION_LEFT;
    } else if (k == 38 || k == 87) {
      // up
      pacman.nextDirection = DIRECTION_UP;
    } else if (k == 39 || k == 68) {
      // right
      pacman.nextDirection = DIRECTION_RIGHT;
    } else if (k == 40 || k == 83) {
      // bottom
      pacman.nextDirection = DIRECTION_BOTTOM;
    }
  }, 1);
});
```

Figure 1: Code snippet for Pacman movement controls.

3.4 Ghost AI

Requirement: Ghosts must follow the AI algorithm to chase Pacman using Dijkstra's algorithm for pathfinding.

Implementation: Each ghost has a unique behaviour pattern (e.g., random movement, targeted chasing). This can be shown in Figure 2.

```
class Ghost {
  changeDirectionIfPossible() {
    let tempDirection = this.direction;
    this.direction = this.calculateNewDirection(
      map,
      parseInt(this.target.x / oneBlockSize),
      parseInt(this.target.y / oneBlockSize)
    );
    if (typeof this.direction == "undefined") {
      this.direction = tempDirection;
      return;
    }
    this.moveForwards();
    if (this.checkCollision()) {
      this.moveBackwards();
      this.direction = tempDirection;
    } else {
      this.moveBackwards();
    }
  }
}
```

Figure 2: Code snippet for Ghost AI direction change.

3.5 Collision Detection

Requirements: The game must detect collisions between Pacman and walls, pellets, and ghosts.

Implementation: On collision with a ghost, Pacman should lose a life. On collision with a pellet, Pacman should eat the pellet and increase the score. Shown in Figure 3.

```
eat() {
  const eatSound = document.getElementById("eat-sound");
  for (let i = 0; i < map.length; i++) {
    for (let j = 0; j < map[0].length; j++) {
      if (map[i][j] == 2 && this.getMapX() == j && this.getMapY() == i) {
        map[i][j] = 3;
        score++;
        eatSound.play(); // Play sound effect
      }
    }
  }
}
```

Figure 3: Code snippet for Pacman eating pellets and playing sound effects.

3.6 Scoring and Lives

Requirement: The game must maintain a score based on the number of pellets eaten. Pacman should have a set number of lives, which decreases upon collision with a ghost.

Implementation: The score and lives are displayed on the game screen shown in Figure 4.

```
let lives = 3;
```

```
let score = 0;
```

```
let drawLives = () => {
  canvasContext.font = "20px Emulogic";
  canvasContext.fillStyle = "white";
  canvasContext.fillText("Lives: ", 220, oneBlockSize * (map.length + 1) + 10);
  for (let i = 0; i < lives; i++) {
    canvasContext.drawImage(
      pacmanFrames,
      2 * oneBlockSize,
      0,
      oneBlockSize,
      oneBlockSize,
      350 + i * oneBlockSize,
      oneBlockSize * map.length + 10,
      oneBlockSize,
      oneBlockSize
    );
  }
};
```

Figure 4: Code snippet for drawing scores and lives.

3.7 Game Over and Win Conditions:

Requirement: The game must display a “Game Over” message when Pacman runs of out of lives. The game should display a “You Win” message when all pellets are eaten.

Implementation: Game states are managed and appropriate messages are displayed on game conditions which can be shown in Figure 5 below.

```
let gameOver = () => {
  drawGameOver();
  clearInterval(gameInterval);
};

let drawGameOver = () => {
  canvasContext.font = "20px Emulogic";
  canvasContext.fillStyle = "white";
  canvasContext.fillText("Game Over!", 150, 200);
};

let drawWin = () => {
  canvasContext.font = "20px Emulogic";
  canvasContext.fillStyle = "white";
  canvasContext.fillText("YOU WIN!", 0, 200);
};
```

Figure 5: Code snippet for Game Over and Win messages.

3.8 Power Pellets and Ghost Vulnerability

Requirement: Pacman should be able to consume power pellets to turn ghosts blue, making them vulnerable and causing them to run away from Pacman.

Implementation: This feature was attempted but not fully implemented due to inadequate research and inexperience of being able to implement these unique features.

Challenges and Solutions: Integrating the power pellet logic with the existing ghost AI proved to be significantly challenging. Despite multiple attempts, the ghosts did not consistently turn blue and become vulnerable as expected. Additionally, contact with power pellets often resulted in crashes and a dark blank screen whilst trying to find the cause behind it, proving unsuccessful. The code for this logic can be found in Figure 6.

```
class PowerPellet {
  constructor(x, y) {
    this.x = x;
    this.y = y;
    this.consumed = false;
  }

  consume() {
    this.consumed = true;
  }
}

class Ghost {
  constructor(x, y, color) {
    this.x = x;
    this.y = y;
    this.color = color;
    this.vulnerable = false;
  }

  turnBlue() {
    this.color = "blue";
    this.vulnerable = true;
  }

  resetColor(originalColor) {
    this.color = originalColor;
    this.vulnerable = false;
  }
}
```

Figure 6: Code snippet for power pellet consumption logic and ghost colour change.

3.9 Non – Functional Requirements

Non-functional requirements define the quality attributes, system performance, and user experience.

Performance

Requirements: The game should run at a minimum of 30 frames per second (FPS). The game should be responsive to user inputs with minimal lag.

Implementation: The game loop is set to run at 30 fps. This can be shown in Fiure 7.

```
let fps = 30;
```

```
let gameInterval = setInterval(gameLoop, 1000 / fps);
```

Figure 7: Code snippet for setting the game FPS.

Usability

Requirement: The user interface should be intuitive and easy to navigate around. Controls should be easy to understand and use as it only involve clicks and using WASD or arrow keys.

Compatibility

Requirement: The game should be compatible with modern web browsers (Chrome, Firefox, Safari.) The game should adjust to different screen sizes and resolutions.

3.10 Sound Effects

Requirement: The game should include sound effects for actions such as eating pellets and losing a life to enhance the gaming experience.

Implementation: Sound effects are triggered based on game events. This can be shown in Figure 8.

```
const playSound = (sound) => {
  if (userInteracted) {
    sound.play().catch((error) => {
      console.error("Failed to play sound:", error);
    });
  }
};
```

Figure 8: Code snippet for playing sound effects.

3.11 Use Case Diagram

The following use case diagram to illustrates the interaction between the user (Pacman) and the system, including actions like moving, eating pellets, and interacting with ghosts which can be shown in Figure 9.

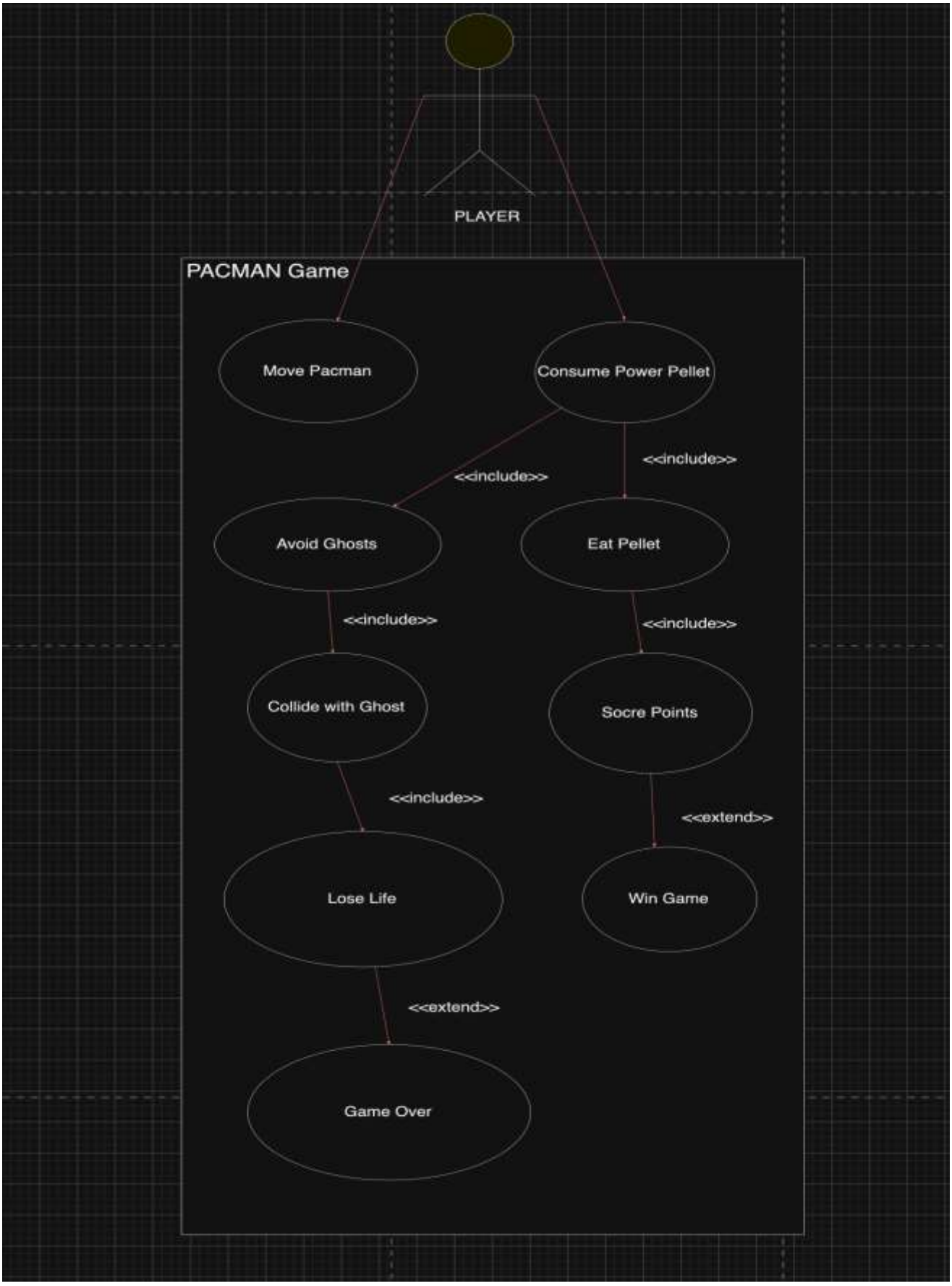


Figure 9: Use Case Diagram of Pacman Game Interactions.

4. Design

4.1 Introduction

The design section of this report explores the architecture and detailed design elements of the Pacman game. It seeks to clarify the system's structure, its components, and their interactions, ensuring a seamless and functional implementation to ensure game runs well.

4.2 System Architecture

The Pacman game is designed with a modular approach, where different components interact to form the complete system. The primary component includes the game engine, the player-controlled Pacman, ghosts with AI behaviour, and the user interface.

Components:

- Game Engine: Manages the game loop, rendering, and state transitions.
- Pacman: Represents the player character with movements and collision detection.
- Ghost: AI-controlled enemies that navigate the maze and interact with Pacman.
- User Interface: Displays the game, score, and other information.
- Maze/Grid: The game map where Pacman and ghosts navigate.
- Sound Manager: Handles game sounds like the "waka waka" sound.

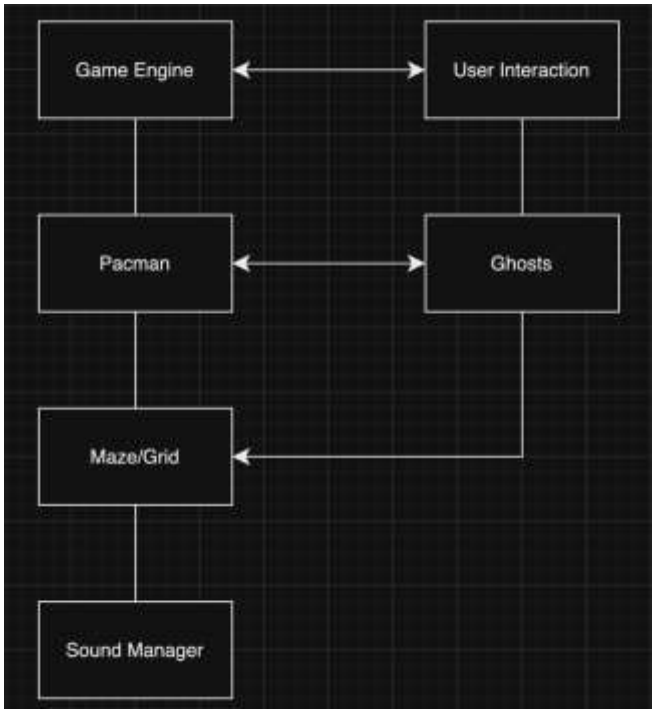


Figure 1: High-level system architecture of the Pacman game.

4.3 Detailed Design

The detailed design section delves into the specific components and their interactions. This includes class diagrams, sequence diagrams, and an explanation of key.

The Pacman class is responsible for the player character's behaviour, including movement, collision detection, and interaction with other game elements.

4.3.1 Pacman Class Diagrams

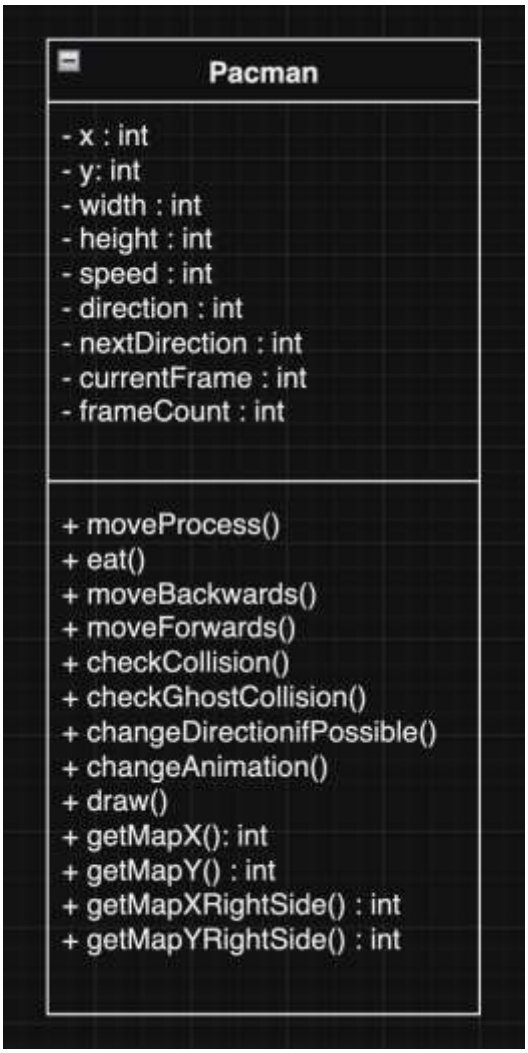


Figure 2: Class diagram for the Pacman character.

4.3.2 Ghost Class Diagram

The Ghost class handles the behaviour of the AI-controlled enemies, including movement, collision detection, and interaction with Pacman.

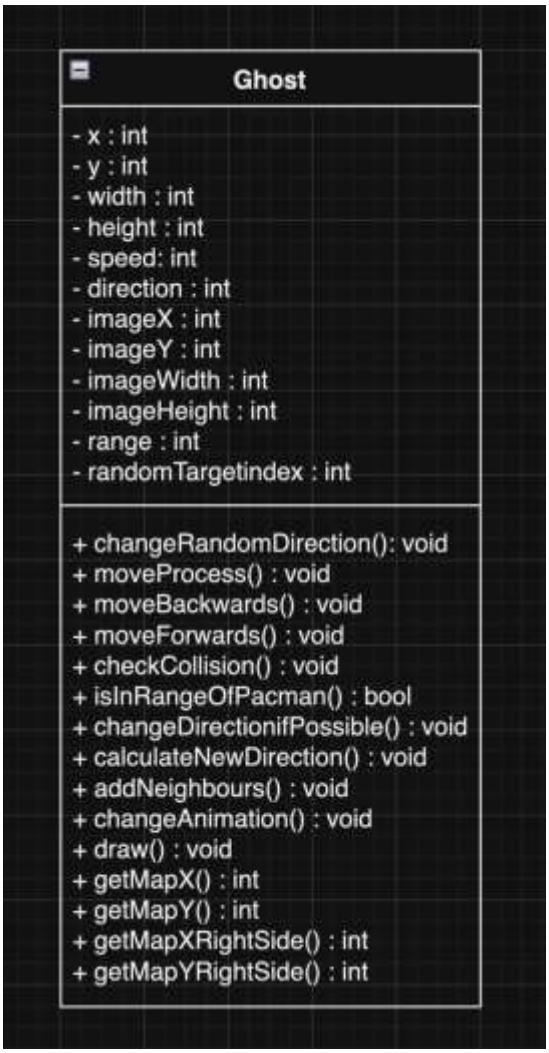


Figure 3: Class diagram for the Ghost character.

4.3.3 Sequence Diagrams

Pacman Movement Sequence Diagram

- Shows the sequence of interaction when the player moves Pacman
- The player presses a key to move Pacman, the Game engine updates Pacman's direction, and Pacman moves on the Maze/Grid. The Collision Detector checks for any collisions, and the Game Engine updates and renders the new game state.

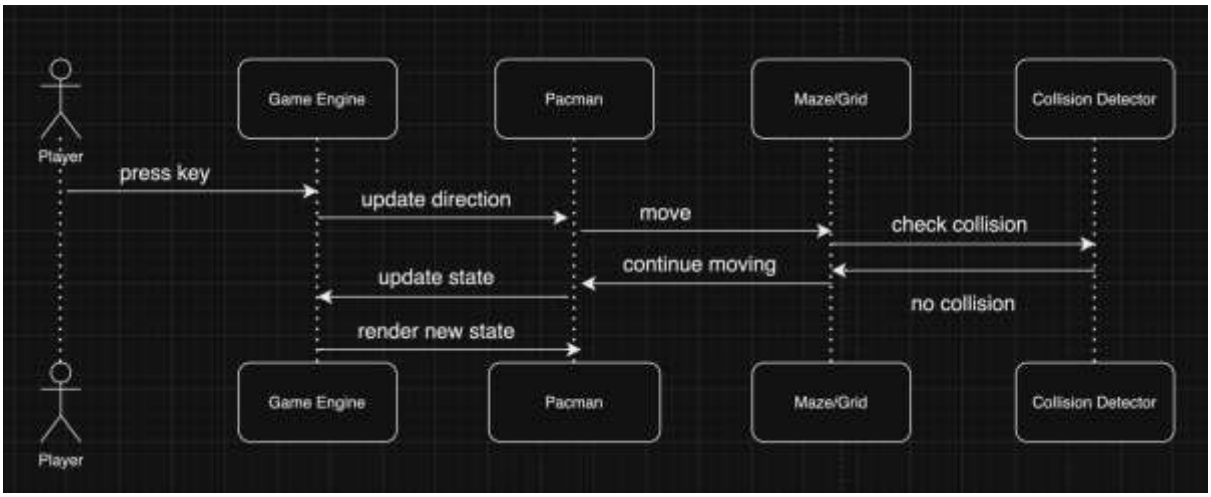


Figure 4: Sequence diagram for Pacman movement.

Ghost AI Behaviour Sequence Diagram

- Illustrates the sequence of operation for ghost AI behaviour using Dijkstra's algorithm
- The Game Engine updates the ghost's target to Pacman's Position. The Ghost requests the shortest path, and the Pathfinding Algorithm uses Dijkstra's algorithm to calculate and return this path. The Ghost updates its direction and moves towards Pacman. The Maze/Grid updates the ghost's position, and the Collision Detector checks for collisions with walls or Pacman.

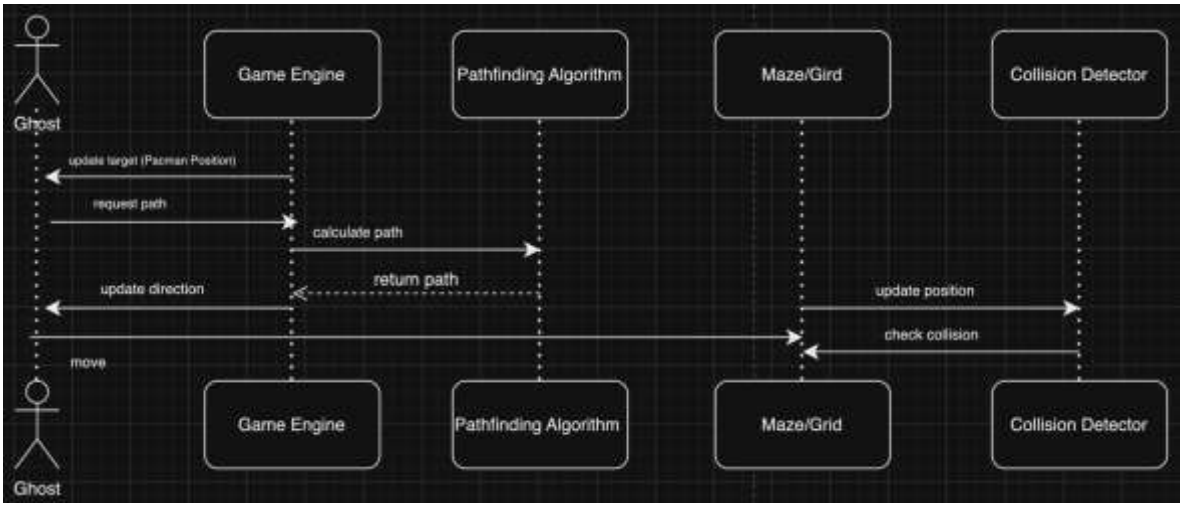


Figure 5: Sequence diagram for Ghost AI behaviour.

4.4 User interface Design

The user interface design focuses on the visual representation of the game. It includes the game canvas, score display, and lives shown.

Wireframes

Game Screen Layout

- Displays the maze, Pacman, ghosts, score, and lives.

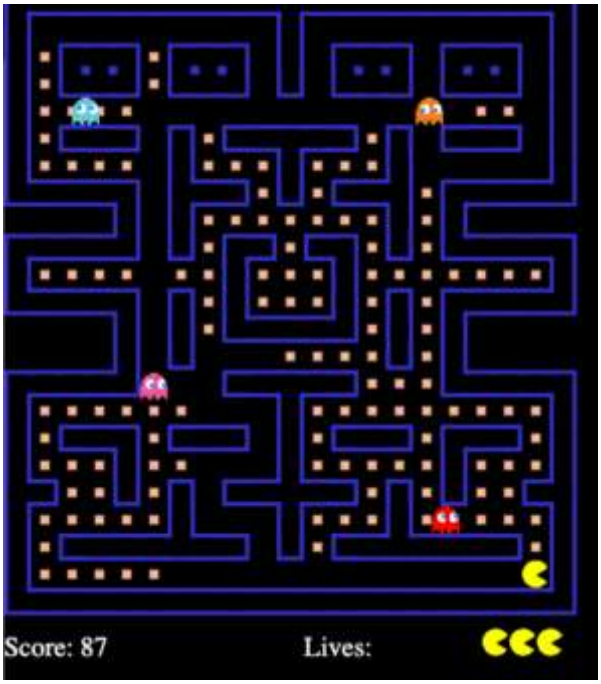


Figure 6: Screenshot of the Pacman game, showing the maze, Pacman and Ghosts.

Score and Lives Display

- Showing the current score and remaining lives.

Figure 7: Wireframe for score and lives display.

Description: The game screen layout includes the maze where Pacman navigates, the position of the ghosts, and the current score and lives remaining. The lives are represented by small Pacman Icons, displayed at the bottom of the screen along with the score.

4.5 Flowcharts

Pacman Movement Flowchart

Provides a visual representation of the logic and flow of key of algorithms used in the game.

- Details the decision–making process for moving Pacman and handling collisions

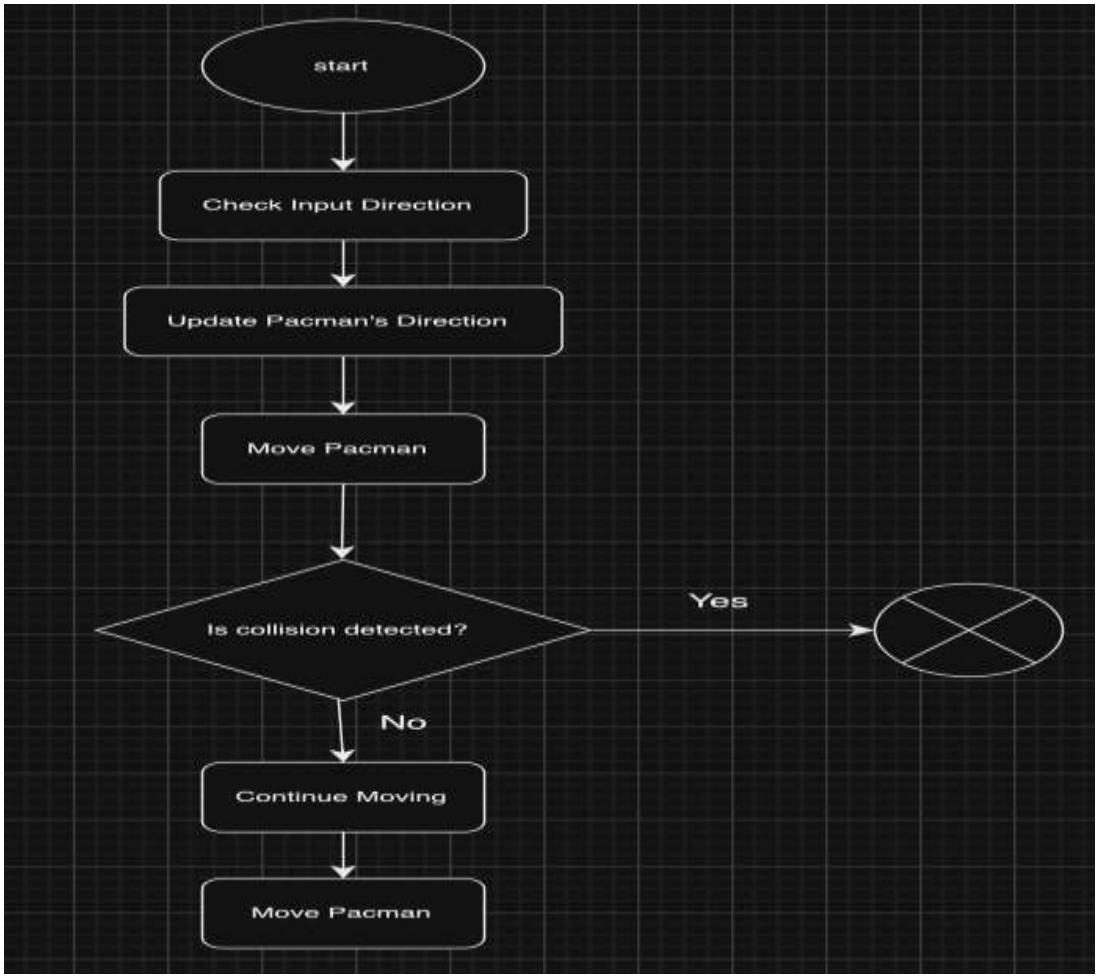


Figure 8: Flowchart for Pacman movement logic.

Ghost AI behaviour Flowchart

- Describes the logic for ghost movement and pathfinding using Dijkstra's algorithm.

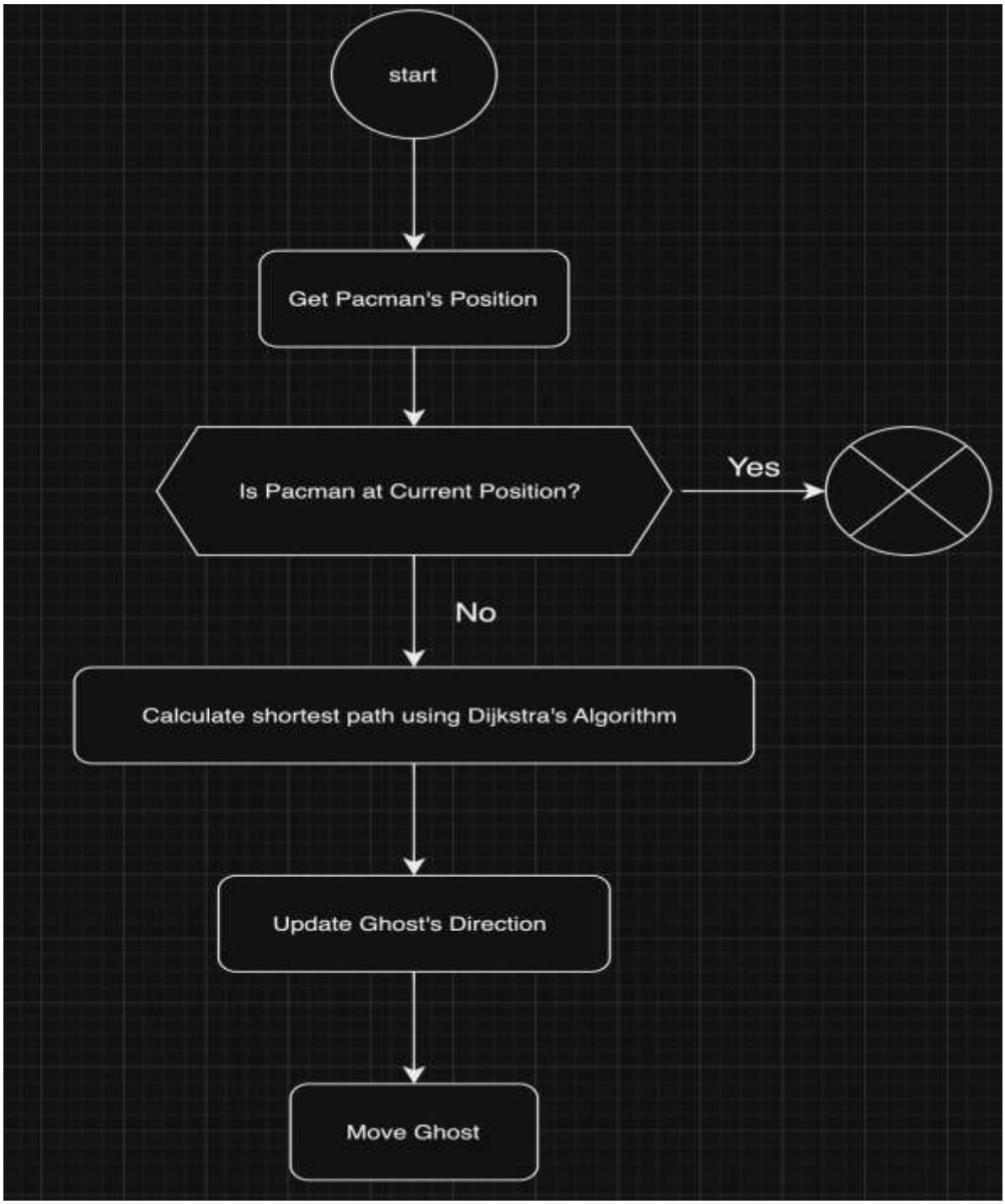


Figure 9: Flowchart for Ghost AI behaviour.

5. Implementation

5.1 Development Environment

The development of the Pacman game was carried out using the following tools and software.

- Languages and Frameworks: HTML, CSS, JavaScript
- Development Tools: Visual Studio Code for code editing, Chrome Developer Tools for debugging.

5.2 Code Structure

The codebase is organised into several files, each handling different aspects of the game.

- Index.html: The main HTML file that structures the web page and includes the game canvas
- Style.css: Contains the CSS styles for the game.
- Pacman.js: manages the behaviour and movement of Pacman.
- Ghost.js: Controls the AI behaviour and movement of Ghosts.
- Game.js: Handles the overall game logic, including initialising the game, updating the game state, and rendering the game.

5.3 Key Features and Implementation Details

Pacman Movement and Control:

Pacman's movement and control were implemented using event listens to capture keyboard inputs. The pacman.js file contains the core logic for Pacman's behaviour.

```
window.addEventListener("keydown", (event) => {
  let k = event.keyCode;

  setTimeout(() => {
    if (k == 37 || k == 65) {
      // left
      pacman.nextDirection = DIRECTION_LEFT;
    } else if (k == 38 || k == 87) {
      // up
      pacman.nextDirection = DIRECTION_UP;
    } else if (k == 39 || k == 68) {
      // right
      pacman.nextDirection = DIRECTION_RIGHT;
    } else if (k == 40 || k == 83) {
      // bottom
      pacman.nextDirection = DIRECTION_BOTTOM;
    }
  }, 1);
});
```

Figure 1: Code snippet for Pacman movement controls.

5.4 Ghost AI and Pathfinding:

The ghosts use Dijkstra's algorithm to find the shortest path to Pacman. This algorithm is implemented in the ghost.js file.

```
class Ghost {
  changeDirectionIfPossible() {
    let tempDirection = this.direction;
    this.direction = this.calculateNewDirection(
      map,
      parseInt(this.target.x / oneBlockSize),
      parseInt(this.target.y / oneBlockSize)
    );
    if (typeof this.direction == "undefined") {
      this.direction = tempDirection;
      return;
    }
    this.moveForwards();
    if (this.checkCollision()) {
      this.moveBackwards();
      this.direction = tempDirection;
    } else {
      this.moveBackwards();
    }
  }
}
```

Figure 2: Code snippet for Ghost AI using Dijkstra's algorithm.

5.5 Collision Detection:

Collision detection is handled to manage the interaction between Pacman, ghosts, and walls. The logic is found in the pacman.js file.

```
checkCollision() {
  if (
    map[this.getMapY()][this.getMapX()] == 1 ||
    map[this.getMapYRightSide()][this.getMapX()] == 1 ||
    map[this.getMapY()][this.getMapXRightSide()] == 1 ||
    map[this.getMapYRightSide()][this.getMapXRightSide()] == 1
  ) {
    return true;
  }
  return false;
}
```

Figure 3: Code snippet for collision detection in Pacman.

5.6 Game Initialisation and Loop:

The game is initiated and controlled through a continuous game loop, which updates the game state and renders the game frame by frame. This logic is handled in the game.js file.

```
let gameLoop = () => {
  draw();
  update();
};

let update = () => {
  pacman.moveProcess();
  pacman.eat();
  for (let i = 0; i < ghosts.length; i++) {
    ghosts[i].moveProcess();
  }
  if (pacman.checkGhostCollision()) {
    console.log("hit");
    restartGame();
  }
  if (score >= foodCount) {
    drawWin();
    clearInterval(gameInterval);
  }
};

let draw = () => {
  createRect(0, 0, canvas.width, canvas.height, "black");
  drawWalls();
  drawFoods();
  pacman.draw();
  drawScore();
  drawGhosts();
  drawLives();
};

let gameInterval = setInterval(gameLoop, 1000 / fps);
```

Figure 4: Code snippet for game loop and state updates.

5.7 Challenges and Solutions

Power Pellets and Ghost Behaviour:

Challenge: Implementing the logic for power pellets to turn ghosts blue and make them vulnerable proved to be a challenge.

- Frequent crashes when implementing the power pellet feature.
- Unknown command prompts due to implementing these features.
- Not enough research and understanding of how to manage the system.
- Ghosts did not turn blue as expected, and Pacman's consumption of power pellets did not trigger the intended behaviour of making ghosts run away; instead, ghosts continued to attack Pacman, leading to a game-over.

Solution: While the full implementation wasn't finished, extensive research on state management in JavaScript was done, and a basic framework was established for future development. The following snippet shows the initial attempt at implementing this feature.

```
if (map[this.getMapY()][this.getMapX()] == 2) {
  this.powerDotActive = true;
  this.powerDotAboutToExpire = false;
  this.timers = 0;
  for (let i = 0; i < ghosts.length; i++) {
    ghosts[i].isScared = true;
  }
  map[this.getMapY()][this.getMapX()] = 0;
}
```

Figure 5: Code snippet for power pellet consumption logic and ghost colour change.

5.8 Pathfinding Algorithm Optimization:

Challenge: Make sure the pathfinding algorithm (Dijkstra's) runs efficiently in real time during the game.

Solution: Improved the algorithm by limiting the search area and pre-calculating some paths when possible.

5.9 Development Photos

1. **Initial Layout:**

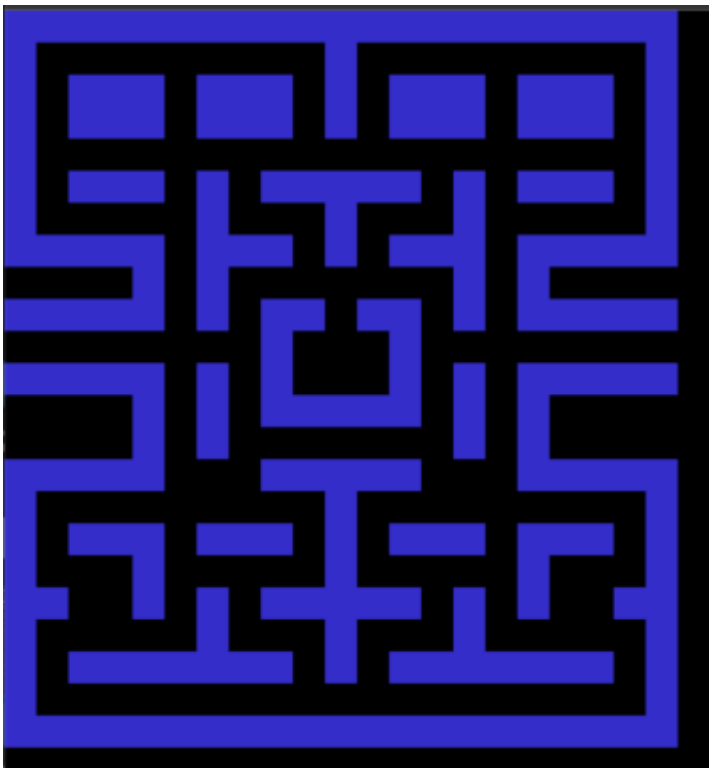


Figure 6: Initial layout of the Pacman game during development.

2. **Ghost AI Testing**

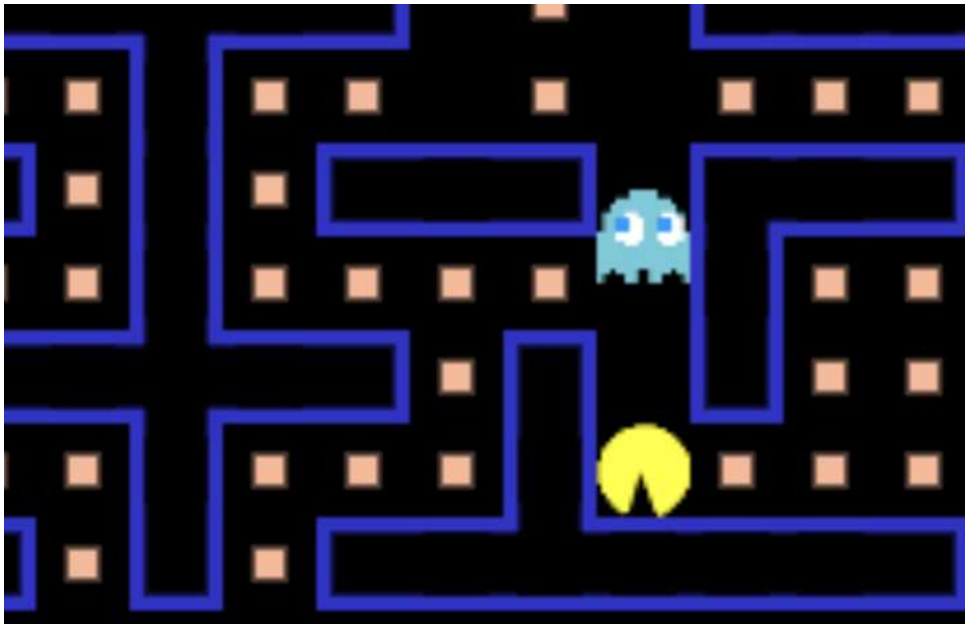


Figure 7: Testing ghost AI behaviour and pathfinding

3. Collision Detection Implementation:

Figure 8: Implementing and testing collision detection between Pacman and Ghosts.

4. Final Game Screen

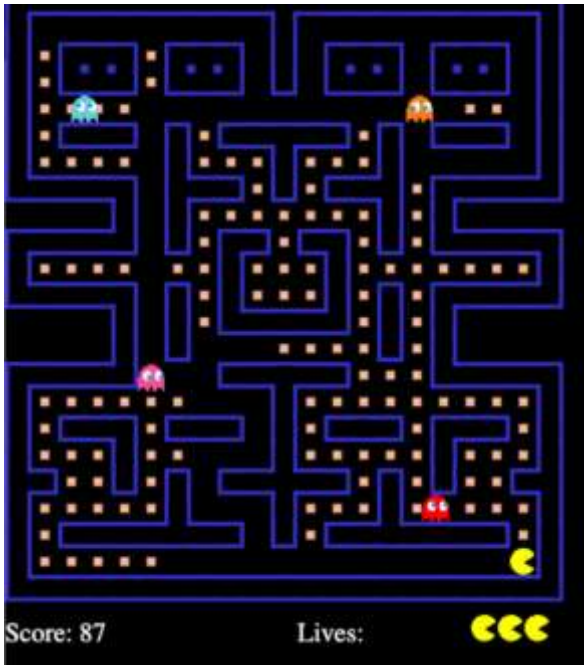


Figure 9: Final game screen showing Pacman, ghosts, score, and lives.

5.10 Conclusions

The implementation of the Pacman game involved developing key features such as Pacman's movement, ghost AI using Dijkstra's algorithm, and collision detection. Despite challenges with the power pellet functionality, significant progress was made, and the game provides a solid foundation for further improvement. With more research and experience, I would like to focus on refining ghost behaviour and implementing additional game features.

6. Testing and Evaluation

6.1 Testing Plan

A comprehensive testing plan was developed to ensure the Pacman game functions correctly and meets the specific requirements. The testing process involved several stages, including unit testing, integrating testing, and user acceptance testing.

6.2 Unit Testing

Unit testing was conducted to verify the functionality of individual components. The test focused on Pacman's movement, ghost AI behaviour, and collision detection.

6.2.1 Test Cases and Results

Key:

The requirement was successfully met	
The requirement was partially met	
The requirement was not met	

6.2.2 Pacman Movement

Test Case	Expected Result	Actual Result	Status
Move up when the up is pressed or W.	Pacman moves up	Pacman moves up	
Move down when the down key is pressed or S.	Pacman moves down	Pacman moves down.	
Move left when the left key is pressed or D	Pacman moves left	Pacman moves left.	
Move right when the right key is pressed or A	Pacman moves right.	Pacman moves right.	

6.2.3 Ghost AI Behaviour

Test Case	**Expected Result**	**Actual Result**	**Status**
Ghosts follow Pacman using the shortest path	**Ghost use Dijkstra's algorithm to follow Pacman**	**Ghosts follow Pacman**	**Pass**
Ghosts change direction when Pacman changes direction	**Ghost adjusts their path accordingly**	**Ghosts changes direction**	**Pass**
Ghost movement without causing crashes.	**Ghosts move smoothly without causing the game to crash.**	**Ghosts moves smoothly**	**Pass**

6.2.4 Collision Detection

Test Case	Expected Result	Actual Result	Status
Detection collisions between Pacman and walls	Pacman stops when hitting walls.	Collision detected	Pass
Detect collisions between Pacman and ghosts.	Game registers collisions and updates state.	Collision detected	Pass
Handles collisions without causing crashes.	Game continues running smoothly after collisions.	No Crashes	Pass

6.2.5 Integration Testing

Integration testing was performed to ensure different components of the game work together efficiently.

Test Case	Expected Result	Actual Result	Status
Pacman's movement and ghost AI interaction	Pacman and ghost interact correctly	Interactions correct	Pass
Collision detection integrated with game state	Game updates state correctly upon collisions	State updates correctly	Pass
Game initilisation and loop	Game starts and runs continuously without issues	Game runs smoothly	Pass

6.2.6 User Acceptance Testing

User acceptance testing was conducted to validate the game's functionality and user experience from an end-user perspective. Feedback was collected from my group of friends during testing sessions.

Test Case	Expected Result	Actual Result	Status
Game is engaging and controls are intuitive	Users find the game engaging and easy to control	Positive feedback	Pass
Ghost AI behaviour is challenging, strategic and fair.	Users find ghost AI behaviour challenging, strategic and fair.	Positive feedback	Pass
Power pellet functionality	Users can activate power pellets upon consumption and see ghosts turn blue and run away.	Features not fully implemented, crashes	Fail

6.2.7 Power Pellet Functionality

Despite significant efforts to implement this feature into the game, the power pellet was not fully implemented due to several issues:

Requirement	Expected Result	Actual Result	Status
Allow Pacman to consume power pellets	Pacman consumes power pellets	Pacman consumes power pellets (but crashes later)	Pass/Fail
Turn ghost blue when Pacman consumes a power pellet	Ghosts turn blue and flee from Pacman	Ghosts do not turn blue, crashes	Fail
Make ghosts run away from Pacman when they are blue	Ghosts run way from Pacman	Ghosts attack Pacman, game over and crashes later.	Fail
Handle power pellet effects without causing crashes	Game runs smoothly after consuming power pellets	Game crashes	Fail

6.3 Evaluation

The game demonstrates excellent performance, running smoothly on modern web browser with minimal lag. This was achieved through efficient coding practices and optimisation of the game loop. The rendering process ensures that the game maintains a consistent frame rate, providing a seamless and enjoyable experience for users. The game is stable and responsive under normal playing conditions, with core functionalities such as Pacman's movement and ghost AI being robust and reliable however, stability issues were observed with the power pellet feature, which caused game crashes due to incomplete and management problems.

The codebase is modular, making it scalable. Future features and enhancements can be integrated with ease. The separation of concern in the code structure allows for independent development and testing of different game components, facilitating future development. The code is well-documented, with clear comments and structured logic which makes it easy to understand and maintain. Best practices in coding were followed, such as using meaningful variable names, modular functions, and consistent indentation.

The game delivers an engaging, nostalgic experience, capturing the essence of the original Pacman. Intuitive controls allow easy navigation through the maze and the ghost AI behaviour adds a challenge, with ghosts actively perusing Pacman based on the shortest path calculated using Dijkstra's algorithm. Furthermore, Positive feedback was received regarding the overall gameplay experience. Users appreciated the responsive controls and the challenging ghost AI, and the users noted that the power pellet feature needs improvement, as the expected behaviour of ghosts turning blue and fleeing was not functioning correctly.

The game's visual design stays true to the original Pacman aesthetics, with vibrant colours and recognizable characters. The maze layout is clear, and the game elements are well-defined. Iconic audio including the waka waka sound, enhances nostalgia though further improvements could be made.

The implementation of power pellets faced significant challenges. The intended function of ghosts turning blue and fleeing from Pacman was not met due to frequent crashes and other issues. Multiple attempts to stabilise the game when power pellets were consumed resulted in a premature game over states instead of the desired behaviour. Furthermore, development encountered several unknown commands prompts and unexpected behaviours, hindering the implementation of complex features and requiring more debugging. The lack of comprehensive error handling and debugging tools highlighted an area for future improvement.

The project exposed gaps in understanding complex game logic and management. Advanced features such as ghost behaviour and power pellets needed extensive research, my failure to learn to do so has made me unable to implement these features.

In conclusion, the Pacman game project successfully implemented core features like Pacman's movement, ghost AI, and collision detection, delivering a smooth and engaging user experience. However, power pellet functionality, remains a challenge along with Ghost turning blue behaviour, needing further research and development. Despite this, the project lays a solid foundation for future work. The module code structure and positive user feedback indicate strong potential to improve. Addressing limitations and adding advanced features will improve the game's quality and enjoyment.

This concludes my evaluation, where the project's achievements and limitations were highlighted.

References

Pacman Development

1. Masswerk.at, Pacman Development, Available at: https://www.masswerk.at/JavaPac/pacman-howto.html
2. Reddit, How to Write a Pacman Game in JavaScript, Available at: https://www.reddit.com/r/javascript/comments/3nl803/how_to_write_a_pacman_game_in_javascript/
3. W3Schools, JavaScript Tutorial, Available at: https://www.w3schools.com/js/
4. W3Schools, JavaScript Tutorial, Available at:
- https://www.w3schools.com/graphics/game_score.asp
5. W3Schools, JavaScript Tutorial, Available at:
- https://www.w3schools.com/dsa/dsa_algo_graphs_dijkstra.php
6. W3Schools, Game Obstacles, Available at:
- https://www.w3schools.com/graphics/game_obstacles.asp
7. Wikipedia, Pac-Man, Available at:
- https://en.wikipedia.org/wiki/Pac-Man
8. Pacman Fandom, Pac-Man, Available at:
- https://pacman.fandom.com/wiki/Pac-Man
9. Mozilla Developer Network, Canvas API, Available at:
- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide
10. Mozilla Developer Network, Canvas API, Available at:
- https://developer.mozilla.org/en-US/docs/Web/API/Canvas_API
11. Reddit, Game Development Community, Available at:
- https://www.reddit.com/r/gamedev/
12. Stack Exchange, Game Development, Available at:
- https://gamedev.stackexchange.com/
13. HTML5 Game Dev Forum, Game Development with HTML 5, Available at:
- https://www.html5gamedevs.com/
14. JavaScript.info, JavaScript Tutorials and Information, Available at:
- https://javascript.info/
15. W3Schools, Keyboard Event, Available at:
- https://www.w3schools.com/jsref/obj_keyboardevent.asp
16. KeyJs, JavaScript Key Detection, Available at:
- https://keyjs.dev/
17. CodeWizardsHQ, JavaScript Games, Available at:

- https://www.codewizardshq.com/javascript-games/

18. freeCodeCamp, Learn JavaScript Game Development, Available at:
- https://www.freecodecamp.org/news/learn-javascript-game-development-full-course/

19. W3Schools, HTML Tutorial, Available at:
- https://www.w3schools.com/html/

20. GeeksforGeeks, Introduction to JavaScript, Available at:
- https://www.geeksforgeeks.org/introduction-to-javascript/

21. Arcade Blogger, The Development of Pac-Man, Available at:
- https://arcadeblogger.com/2016/03/12/the-development-of-pacman/

22. PacMan Code, Pac-Man Source Code and Analysis, Available at:
- https://pacmancode.com/

23. W3Schools, Introduction to JavaScript, Available at:
- https://www.w3schools.com/js/js_intro.asp

24. Quicksounds, Waka Waka (Pacman) sound, Available at
- https://quicksounds.com/sound/48/waka-waka-pacman

25. Pinterest, Maze Layout Idea, Available at
- https://in.pinterest.com/pin/313000242834176065/

26. Backofthecerealbox, Ghost Illustrations idea, Available at: https://www.backofthecerealbox.com/2008/10/blinky-pinky-inky-and-clyde-smarter.html

Appendix

Project Proposal

Pac Man

Computing Area

Programming

Description and Aim

The aim of this project is to recreate the classic arcade game Pac-Man using JavaScript and HTML, bringing this beloved retro gaming experience to modern web browsers. I'll be using HTML for structuring the game's layout and JavaScript for implementing the game logic which can be used to create a responsive and interactive gaming experience. In this project, key features include implementing core gameplay mechanics such as controlling Pac-Man's movement, collecting pellets, and avoiding ghosts to replicate their unique personalities from the original game, additionally, I'll add and design a scoring system, visually appealing animations into the game. This project will have a user-friendly interface with intuitive controls and an on-screen display for score and remaining lives.

Objectives

Programs Should:

- Implement Pac-Man movement mechanics, allowing the player to control Pac-Man using keyboard input.
- Create a maze layout, replicating the design of the original Pac-Man Maze by using HTML and JavaScript.
- Develop a collision detection algorithm to handle interactions between Pac-Man, pellets, fruits, and maze walls.
- Design ghost movement patterns and AI behaviour to mimic the original game's ghost behaviour using Dijkstra's algorithm, including chasing Pac-Man.
- Develop a scoring system to track points earned by consuming pellets.

Resources/Programming Languages

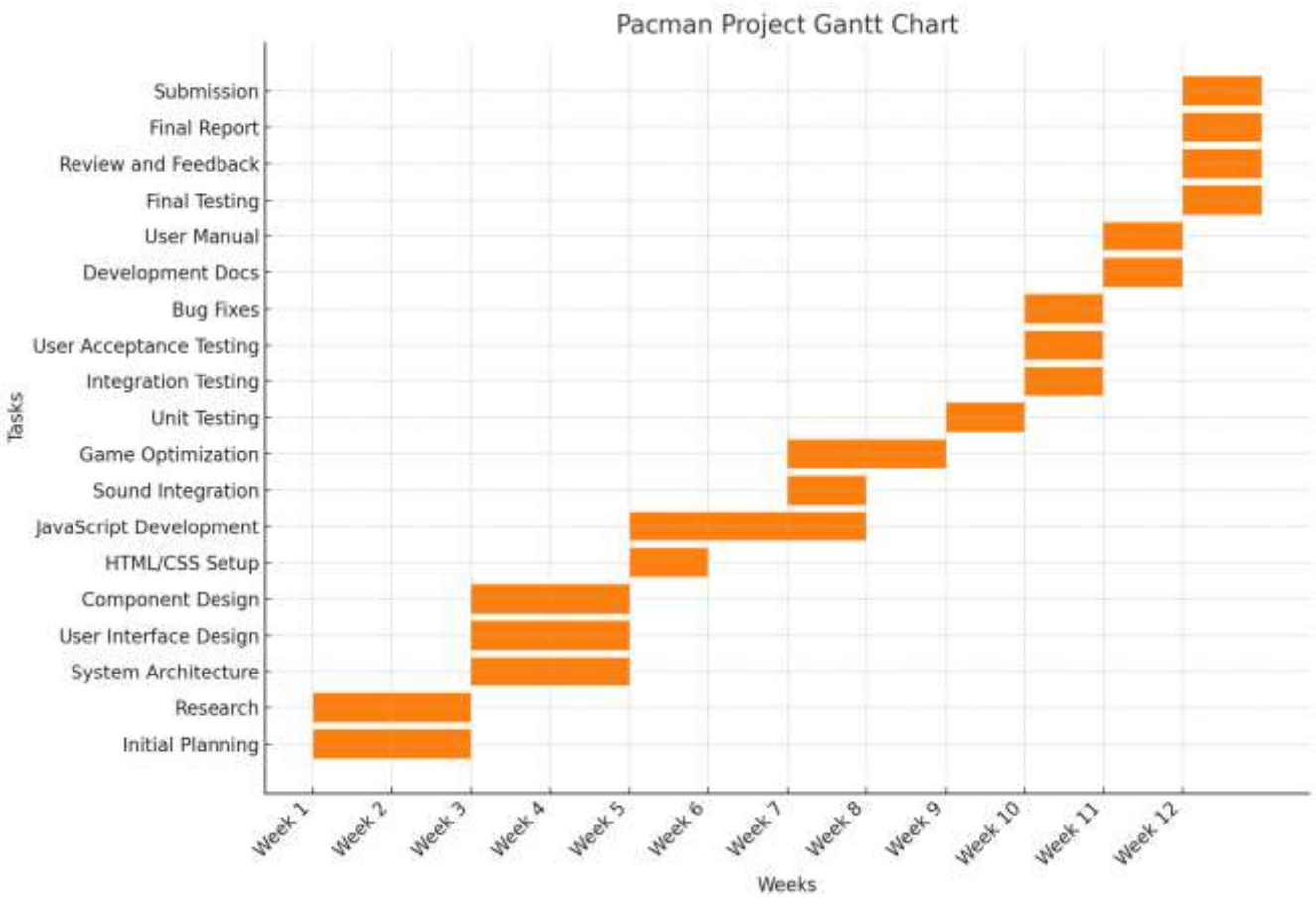
I will use the following languages to make this Pac-Man Possible.

- HTML
- JavaScript

For HTML, I've become familiar with the usage of HTML through this course. Available resources that I have access to in case I have trouble with HTML include canvas, where recordings and PowerPoint presentations are available for guidance should I require help and helpful websites such as W3Schools to help guide me throughout this project.

For JavaScript, I'm self-teaching this programming language which is vital for this project to come to fruition. Helpful resources for JavaScript include W3Schools and YouTube videos to guide me through my project.

Gannt Charts



February 28, 2024 - May 22, 2024. Both initial and final were incorporated into one.

Source Code

Game.js

```
const canvas = document.getElementById("canvas");
const canvasContext = canvas.getContext("2d");
const pacmanFrames = document.getElementById("animations");
const ghostFrames = document.getElementById("ghosts");

let userInteracted = false;

window.addEventListener("click", () => {
```

```
  userInteracted = true;
});

const playSound = (sound) => {
  if (userInteracted) {
    sound.play().catch((error) => {
      console.error("Failed to play sound:", error);
    });
  }
};

let createRect = (x, y, width, height, color) => {
  canvasContext.fillStyle = color;
  canvasContext.fillRect(x, y, width, height);
};

let fps = 30;
let oneBlockSize = 20;
let wallColor = "#342DCA";
let wallSpaceWidth = oneBlockSize / 1.5;
let wallOffset = (oneBlockSize - wallSpaceWidth) / 2;
let wallInnerColor = "black";
let foodColor = "#FEB897";
let score = 0;
let ghosts = [];
let ghostCount = 4;
let lives = 3;
let foodCount = 0;

const DIRECTION_RIGHT = 4;
const DIRECTION_UP = 3;
```

```
const DIRECTION_LEFT = 2;
const DIRECTION_BOTTOM = 1;

let ghostLocations = [
  { x: 0, y: 0 },
  { x: 176, y: 0 },
  { x: 0, y: 121 },
  { x: 176, y: 121 },
];

let map = [
  [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
  [1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 1],
  [1, 2, 1, 1, 1, 2, 1, 1, 1, 2, 1, 2, 1, 1, 1, 2, 1, 1, 1],
  [1, 2, 1, 1, 1, 2, 1, 1, 1, 2, 1, 2, 1, 1, 1, 2, 1, 1, 1],
  [1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1],
  [1, 2, 1, 1, 1, 2, 1, 2, 1, 1, 1, 1, 1, 2, 1, 2, 1, 1, 1],
  [1, 2, 2, 2, 2, 2, 1, 2, 2, 2, 1, 2, 2, 2, 1, 2, 2, 2, 1],
  [1, 1, 1, 1, 1, 2, 1, 1, 1, 2, 1, 2, 1, 1, 1, 2, 1, 1, 1],
  [0, 0, 0, 0, 1, 2, 1, 2, 2, 2, 2, 2, 2, 2, 1, 2, 1, 0, 0],
  [1, 1, 1, 1, 1, 2, 1, 2, 1, 1, 2, 1, 1, 2, 1, 2, 1, 1, 1],
  [1, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 1, 2, 2, 2, 2, 2, 1],
  [1, 1, 1, 1, 1, 2, 1, 2, 1, 2, 2, 2, 1, 2, 1, 2, 1, 1, 1],
  [0, 0, 0, 0, 1, 2, 1, 2, 1, 1, 1, 1, 1, 2, 1, 2, 1, 0, 0],
  [0, 0, 0, 0, 1, 2, 1, 2, 2, 2, 2, 2, 2, 2, 1, 2, 1, 0, 0],
  [1, 1, 1, 1, 1, 2, 2, 2, 1, 1, 1, 1, 1, 2, 2, 2, 1, 1, 1],
  [1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 1],
  [1, 2, 1, 1, 1, 2, 1, 1, 1, 2, 1, 2, 1, 1, 1, 2, 1, 1, 1],
  [1, 2, 2, 2, 1, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 1, 2, 2],
  [1, 1, 2, 2, 1, 2, 1, 2, 1, 1, 1, 1, 1, 2, 1, 2, 1, 2, 2],
  [1, 2, 2, 2, 2, 2, 1, 2, 2, 2, 1, 2, 2, 2, 1, 2, 2, 2, 2],
```

```
  [1, 2, 1, 1, 1, 1, 1, 1, 1, 2, 1, 2, 1, 1, 1, 1, 1, 1, 2, 1],
  [1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1],
  [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
];

for (let i = 0; i < map.length; i++) {
  for (let j = 0; j < map[0].length; j++) {
    if (map[i][j] == 2) {
      foodCount++;
    }
  }
}

let randomTargetsForGhosts = [
  { x: 1 * oneBlockSize, y: 1 * oneBlockSize },
  { x: 1 * oneBlockSize, y: (map.length - 2) * oneBlockSize },
  { x: (map[0].length - 2) * oneBlockSize, y: oneBlockSize },
  {
    x: (map[0].length - 2) * oneBlockSize,
    y: (map.length - 2) * oneBlockSize,
  },
];

let gameLoop = () => {
  draw();
  update();
};

let update = () => {
  pacman.moveProcess();
  pacman.eat();
```

```
  for (let i = 0; i < ghosts.length; i++) {
    ghosts[i].moveProcess();
  }

  if (pacman.checkGhostCollision()) {
    console.log("hit");
    restartGame();
  }
  if (score >= foodCount) {
    drawWin();
    clearInterval(gameInterval);
  }
};

let restartGame = () => {
  createNewPacman();
  createGhosts();
  lives--;
  if (lives == 0) {
    gameOver();
  }
};

let gameOver = () => {
  drawGameOver();
  clearInterval(gameInterval);
};

let drawGameOver = () => {
  canvasContext.font = "20px Emulogic";
  canvasContext.fillStyle = "white";
```

```
  canvasContext.fillText("Game Over!", 150, 200);
};

let drawWin = () => {
  canvasContext.font = "20px Emulogic";
  canvasContext.fillStyle = "white";
  canvasContext.fillText("YOU WIN!", 0, 200);
};

let drawLives = () => {
  canvasContext.font = "20px Emulogic";
  canvasContext.fillStyle = "white";
  canvasContext.fillText("Lives: ", 220, oneBlockSize * (map.length + 1) +
10);
  for (let i = 0; i < lives; i++) {
    canvasContext.drawImage(
      pacmanFrames,
      2 * oneBlockSize,
      0,
      oneBlockSize,
      oneBlockSize,
      350 + i * oneBlockSize,
      oneBlockSize * map.length + 10,
      oneBlockSize,
      oneBlockSize
    );
  }
};

let drawFoods = () => {
  for (let i = 0; i < map.length; i++) {
```

```
    for (let j = 0; j < map[0].length; j++) {
      if (map[i][j] == 2) {
        createRect(
          j * oneBlockSize + oneBlockSize / 3,
          i * oneBlockSize + oneBlockSize / 3,
          oneBlockSize / 3,
          oneBlockSize / 3,
          foodColor
        );
      }
    }
  }
};

let drawScore = () => {
  canvasContext.font = "20px";
  canvasContext.fillStyle = "white";
  canvasContext.fillText(
    "Score: " + score,
    0,
    oneBlockSize * (map.length + 1) + 10
  );
};

let drawGhosts = () => {
  for (let i = 0; i < ghosts.length; i++) {
    ghosts[i].draw();
  }
};

let draw = () => {
```

```
  createRect(0, 0, canvas.width, canvas.height, "black");
  drawWalls();
  drawFoods();
  pacman.draw();
  drawScore();
  drawGhosts();
  drawLives();
};

let gameInterval = setInterval(gameLoop, 1000 / fps);

let drawWalls = () => {
  for (let i = 0; i < map.length; i++) {
    for (let j = 0; j < map[0].length; j++) {
      if (map[i][j] == 1) {
        // then it is a wall
        createRect(
          j * oneBlockSize,
          i * oneBlockSize,
          oneBlockSize,
          oneBlockSize,
          wallColor
        );
        if (j > 0 && map[i][j - 1] == 1) {
          createRect(
            j * oneBlockSize,
            i * oneBlockSize + wallOffset,
            wallSpaceWidth + wallOffset,
            wallSpaceWidth,
            wallInnerColor
          );
```

```
      }
      if (j < map[0].length - 1 && map[i][j + 1] == 1) {
        createRect(
          j * oneBlockSize + wallOffset,
          i * oneBlockSize + wallOffset,
          wallSpaceWidth + wallOffset,
          wallSpaceWidth,
          wallInnerColor
        );
      }
      if (i > 0 && map[i - 1][j] == 1) {
        createRect(
          j * oneBlockSize + wallOffset,
          i * oneBlockSize,
          wallSpaceWidth,
          wallSpaceWidth + wallOffset,
          wallInnerColor
        );
      }
      if (i < map.length - 1 && map[i + 1][j] == 1) {
        createRect(
          j * oneBlockSize + wallOffset,
          i * oneBlockSize + wallOffset,
          wallSpaceWidth,
          wallSpaceWidth + wallOffset,
          wallInnerColor
        );
      }
    }
  }
}
```

```
};

let createNewPacman = () => {
  pacman = new Pacman(
    oneBlockSize,
    oneBlockSize,
    oneBlockSize,
    oneBlockSize,
    oneBlockSize / 5
  );
};

class Ghost {
  constructor(
    x,
    y,
    width,
    height,
    speed,
    imageX,
    imageY,
    imageWidth,
    imageHeight,
    range
  ) {
    this.x = x;
    this.y = y;
    this.width = width;
    this.height = height;
    this.speed = speed;
    this.direction = DIRECTION_RIGHT;
```

```
  this.imageX = imageX;
  this.imageY = imageY;
  this.imageHeight = imageHeight;
  this.imageWidth = imageWidth;
  this.range = range;
  this.randomTargetIndex = parseInt(
    Math.random() * randomTargetsForGhosts.length
  );
  setInterval(() => {
    this.changeRandomDirection();
  }, 10000);
}

changeRandomDirection() {
  this.randomTargetIndex += 1;
  this.randomTargetIndex = this.randomTargetIndex % 4;
}

moveProcess() {
  if (this.isInRangeOfPacman()) {
    this.target = pacman;
  } else {
    this.target = randomTargetsForGhosts[this.randomTargetIndex];
  }

  this.changeDirectionIfPossible();
  this.moveForwards();
  if (this.checkCollision()) {
    this.moveBackwards();
  }
}
```

```
moveBackwards() {
  switch (this.direction) {
    case DIRECTION_RIGHT:
      this.x -= this.speed;
      break;
    case DIRECTION_UP:
      this.y += this.speed;
      break;
    case DIRECTION_LEFT:
      this.x += this.speed;
      break;
    case DIRECTION_BOTTOM:
      this.y -= this.speed;
      break;
  }
}

moveForwards() {
  switch (this.direction) {
    case DIRECTION_RIGHT:
      this.x += this.speed;
      break;
    case DIRECTION_UP:
      this.y -= this.speed;
      break;
    case DIRECTION_LEFT:
      this.x -= this.speed;
      break;
    case DIRECTION_BOTTOM:
      this.y += this.speed;
```

```
      break;
    }
  }

  checkCollision() {
    if (
      map[this.getMapY()][this.getMapX()] == 1 ||
      map[this.getMapYRightSide()][this.getMapX()] == 1 ||
      map[this.getMapY()][this.getMapXRightSide()] == 1 ||
      map[this.getMapYRightSide()][this.getMapXRightSide()] == 1
    ) {
      return true;
    }
    return false;
  }

  isInRangeOfPacman() {
    let xDistance = Math.abs(pacman.getMapX() - this.getMapX());
    let yDistance = Math.abs(pacman.getMapY() - this.getMapY());
    if (
      Math.sqrt(xDistance * xDistance + yDistance * yDistance) <=
this.range
    ) {
      return true;
    }
    return false;
  }

  changeDirectionIfPossible() {
    let tempDirection = this.direction;
```

```
  this.direction = this.calculateNewDirection(
    map,
    parseInt(this.target.x / oneBlockSize),
    parseInt(this.target.y / oneBlockSize)
  );

  if (typeof this.direction == "undefined") {
    this.direction = tempDirection;
    return;
  }

  this.moveForwards();
  if (this.checkCollision()) {
    this.moveBackwards();
    this.direction = tempDirection;
  } else {
    this.moveBackwards();
  }
}

calculateNewDirection(map, destX, destY) {
  let mp = [];
  for (let i = 0; i < map.length; i++) {
    mp[i] = map[i].slice();
  }

  let queue = [
    {
      x: this.getMapX(),
      y: this.getMapY(),
      moves: [],
```

```
    },
  ];

  while (queue.length > 0) {
    let poped = queue.shift();
    if (poped.x == destX && poped.y == destY) {
      return poped.moves[0];
    } else {
      mp[poped.y][poped.x] = 1;
      let neighborList = this.addNeighbors(poped, mp);
      for (let i = 0; i < neighborList.length; i++) {
        queue.push(neighborList[i]);
      }
    }
  }

  return DIRECTION_UP; // default
}

addNeighbors(poped, mp) {
  let queue = [];
  let numOfRows = mp.length;
  let numOfColumns = mp[0].length;

  if (
    poped.x - 1 >= 0 &&
    poped.x - 1 < numOfRows &&
    mp[poped.y][poped.x - 1] != 1
  ) {
    let tempMoves = poped.moves.slice();
    tempMoves.push(DIRECTION_LEFT);
```

```
  queue.push({ x: poped.x - 1, y: poped.y, moves: tempMoves });
}

if (
  poped.x + 1 >= 0 &&
  poped.x + 1 < numOfRows &&
  mp[poped.y][poped.x + 1] != 1
) {
  let tempMoves = poped.moves.slice();
  tempMoves.push(DIRECTION_RIGHT);
  queue.push({ x: poped.x + 1, y: poped.y, moves: tempMoves });
}

if (
  poped.y - 1 >= 0 &&
  poped.y - 1 < numOfRows &&
  mp[poped.y - 1][poped.x] != 1
) {
  let tempMoves = poped.moves.slice();
  tempMoves.push(DIRECTION_UP);
  queue.push({ x: poped.x, y: poped.y - 1, moves: tempMoves });
}

if (
  poped.y + 1 >= 0 &&
  poped.y + 1 < numOfRows &&
  mp[poped.y + 1][poped.x] != 1
) {
  let tempMoves = poped.moves.slice();
  tempMoves.push(DIRECTION_BOTTOM);
  queue.push({ x: poped.x, y: poped.y + 1, moves: tempMoves });
```

```
  }

  return queue;
}

changeAnimation() {
  this.currentFrame =
    this.currentFrame == this.frameCount ? 1 : this.currentFrame + 1;
}

draw() {
  canvasContext.save();
  canvasContext.drawImage(
    ghostFrames,
    this.imageX,
    this.imageY,
    this.imageWidth,
    this.imageHeight,
    this.x,
    this.y,
    this.width,
    this.height
  );
  canvasContext.restore();
}

getMapX() {
  return parseInt(this.x / oneBlockSize);
}

getMapY() {
```

```
    return parseInt(this.y / oneBlockSize);
  }

  getMapXRightSide() {
    return parseInt((this.x + 0.9999 * oneBlockSize) / oneBlockSize);
  }

  getMapYRightSide() {
    return parseInt((this.y + 0.9999 * oneBlockSize) / oneBlockSize);
  }
}

let createGhosts = () => {
  ghosts = [];
  for (let i = 0; i < ghostCount; i++) {
    let newGhost = new Ghost(
      9 * oneBlockSize + (i % 2 == 0 ? 0 : 1) * oneBlockSize,
      10 * oneBlockSize + (i % 2 == 0 ? 0 : 1) * oneBlockSize,
      oneBlockSize,
      oneBlockSize,
      pacman.speed / 2,
      ghostLocations[i % 4].x,
      ghostLocations[i % 4].y,
      124,
      116,
      6 + i
    );
    ghosts.push(newGhost);
  }
};
```

```
createNewPacman();
createGhosts();
gameLoop();

window.addEventListener("keydown", (event) => {
  let k = event.keyCode;

  setTimeout(() => {
    if (k == 37 || k == 65) {
      // left
      pacman.nextDirection = DIRECTION_LEFT;
    } else if (k == 38 || k == 87) {
      // up
      pacman.nextDirection = DIRECTION_UP;
    } else if (k == 39 || k == 68) {
      // right
      pacman.nextDirection = DIRECTION_RIGHT;
    } else if (k == 40 || k == 83) {
      // bottom
      pacman.nextDirection = DIRECTION_BOTTOM;
    }
  }, 1);
});
```

Ghost.js

```
class Ghost {
  constructor(
    x,
    y,
```

```
  width,
  height,
  speed,
  imageX,
  imageY,
  imageWidth,
  imageHeight,
  range
) {
  this.x = x;
  this.y = y;
  this.width = width;
  this.height = height;
  this.speed = speed;
  this.direction = DIRECTION_RIGHT;
  this.imageX = imageX;
  this.imageY = imageY;
  this.imageHeight = imageHeight;
  this.imageWidth = imageWidth;
  this.range = range;
  this.randomTargetIndex = parseInt(
    Math.random() * randomTargetsForGhosts.length
  );
  setInterval(() => {
    this.changeRandomDirection();
  }, 10000);
}

changeRandomDirection() {
  this.randomTargetIndex += 1;
  this.randomTargetIndex = this.randomTargetIndex % 4;
```

```
}

moveProcess() {
 if (this.isInRangeOfPacman()) {
  this.target = pacman;
 } else {
  this.target = randomTargetsForGhosts[this.randomTargetIndex];
 }

 this.changeDirectionIfPossible();
 this.moveForwards();
 if (this.checkCollision()) {
  this.moveBackwards();
 }
}

moveBackwards() {
 switch (this.direction) {
  case DIRECTION_RIGHT:
   this.x -= this.speed;
   break;
  case DIRECTION_UP:
   this.y += this.speed;
   break;
  case DIRECTION_LEFT:
   this.x += this.speed;
   break;
  case DIRECTION_BOTTOM:
   this.y -= this.speed;
   break;
 }
```

```
}

moveForwards() {
  switch (this.direction) {
    case DIRECTION_RIGHT:
      this.x += this.speed;
      break;
    case DIRECTION_UP:
      this.y -= this.speed;
      break;
    case DIRECTION_LEFT:
      this.x -= this.speed;
      break;
    case DIRECTION_BOTTOM:
      this.y += this.speed;
      break;
  }
}

checkCollision() {
  if (
    map[this.getMapY()][this.getMapX()] == 1 ||
    map[this.getMapYRightSide()][this.getMapX()] == 1 ||
    map[this.getMapY()][this.getMapXRightSide()] == 1 ||
    map[this.getMapYRightSide()][this.getMapXRightSide()] == 1
  ) {
    return true;
  }
  return false;
}
```

```
 isInRangeOfPacman() {
  let xDistance = Math.abs(pacman.getMapX() - this.getMapX());
  let yDistance = Math.abs(pacman.getMapY() - this.getMapY());
  if (
   Math.sqrt(xDistance * xDistance + yDistance * yDistance) <=
this.range
  ) {
   return true;
  }
  return false;
 }

 changeDirectionIfPossible() {
  let tempDirection = this.direction;

  this.direction = this.calculateNewDirection(
   map,
   parseInt(this.target.x / oneBlockSize),
   parseInt(this.target.y / oneBlockSize)
  );

  if (typeof this.direction == "undefined") {
   this.direction = tempDirection;
   return;
  }

  this.moveForwards();
  if (this.checkCollision()) {
   this.moveBackwards();
   this.direction = tempDirection;
  } else {
```

```
    this.moveBackwards();
  }
}

calculateNewDirection(map, destX, destY) {
  let mp = [];
  for (let i = 0; i < map.length; i++) {
    mp[i] = map[i].slice();
  }

  let queue = [
    {
      x: this.getMapX(),
      y: this.getMapY(),
      moves: [],
    },
  ];

  while (queue.length > 0) {
    let poped = queue.shift();
    if (poped.x == destX && poped.y == destY) {
      return poped.moves[0];
    } else {
      mp[poped.y][poped.x] = 1;
      let neighborList = this.addNeighbors(poped, mp);
      for (let i = 0; i < neighborList.length; i++) {
        queue.push(neighborList[i]);
      }
    }
  }
```

```
  return DIRECTION_UP; // default
}

addNeighbors(poped, mp) {
  let queue = [];
  let numOfRows = mp.length;
  let numOfColumns = mp[0].length;

  if (
    poped.x - 1 >= 0 &&
    poped.x - 1 < numOfRows &&
    mp[poped.y][poped.x - 1] != 1
  ) {
    let tempMoves = poped.moves.slice();
    tempMoves.push(DIRECTION_LEFT);
    queue.push({ x: poped.x - 1, y: poped.y, moves: tempMoves });
  }

  if (
    poped.x + 1 >= 0 &&
    poped.x + 1 < numOfRows &&
    mp[poped.y][poped.x + 1] != 1
  ) {
    let tempMoves = poped.moves.slice();
    tempMoves.push(DIRECTION_RIGHT);
    queue.push({ x: poped.x + 1, y: poped.y, moves: tempMoves });
  }

  if (
    poped.y - 1 >= 0 &&
    poped.y - 1 < numOfRows &&
```

```
    mp[poped.y - 1][poped.x] != 1
  ) {
    let tempMoves = poped.moves.slice();
    tempMoves.push(DIRECTION_UP);
    queue.push({ x: poped.x, y: poped.y - 1, moves: tempMoves });
  }

  if (
    poped.y + 1 >= 0 &&
    poped.y + 1 < numOfRows &&
    mp[poped.y + 1][poped.x] != 1
  ) {
    let tempMoves = poped.moves.slice();
    tempMoves.push(DIRECTION_BOTTOM);
    queue.push({ x: poped.x, y: poped.y + 1, moves: tempMoves });
  }

  return queue;
}

changeAnimation() {
  this.currentFrame =
    this.currentFrame == this.frameCount ? 1 : this.currentFrame + 1;
}

draw() {
  canvasContext.save();
  canvasContext.drawImage(
    ghostFrames,
    this.imageX,
    this.imageY,
```

```
    this.imageWidth,
    this.imageHeight,
    this.x,
    this.y,
    this.width,
    this.height
  );
  canvasContext.restore();
 }

 getMapX() {
  return parseInt(this.x / oneBlockSize);
 }

 getMapY() {
  return parseInt(this.y / oneBlockSize);
 }

 getMapXRightSide() {
  return parseInt((this.x + 0.9999 * oneBlockSize) / oneBlockSize);
 }

 getMapYRightSide() {
  return parseInt((this.y + 0.9999 * oneBlockSize) / oneBlockSize);
 }
}

if (map[this.getMapY()][this.getMapX()] == 2) {
 this.powerDotActive = true;
 this.powerDotAboutToExpire = false;
 this.timers = 0;
```

```
  for (let i = 0; i < ghosts.length; i++) {
    ghosts[i].isScared = true;
  }
  map[this.getMapY()][this.getMapX()] = 0;
}
```

Pacman.js

```
class Pacman {
  constructor(x, y, width, height, speed) {
    this.x = x;
    this.y = y;
    this.width = width;
    this.height = height;
    this.speed = speed;
    this.direction = DIRECTION_RIGHT;
    this.nextDirection = this.direction;
    this.currentFrame = 1;
    this.frameCount = 7;

    setInterval(() => {
      this.changeAnimation();
    }, 100);
  }

  moveProcess() {
    this.changeDirectionIfPossible();
    this.moveForwards();
    if (this.checkCollision()) {
      this.moveBackwards();
```

```
  }
}

eat() {
  const eatSound = document.getElementById("eat-sound");
  for (let i = 0; i < map.length; i++) {
    for (let j = 0; j < map[0].length; j++) {
      if (map[i][j] == 2 && this.getMapX() == j && this.getMapY() == i) {
        map[i][j] = 3;
        score++;
        eatSound.play(); // Play sound effect
      }
    }
  }
}

moveBackwards() {
  switch (this.direction) {
    case DIRECTION_RIGHT:
      this.x -= this.speed;
      break;
    case DIRECTION_UP:
      this.y += this.speed;
      break;
    case DIRECTION_LEFT:
      this.x += this.speed;
      break;
    case DIRECTION_BOTTOM:
      this.y -= this.speed;
      break;
  }
```

```
}

moveForwards() {
  switch (this.direction) {
    case DIRECTION_RIGHT:
      this.x += this.speed;
      break;
    case DIRECTION_UP:
      this.y -= this.speed;
      break;
    case DIRECTION_LEFT:
      this.x -= this.speed;
      break;
    case DIRECTION_BOTTOM:
      this.y += this.speed;
      break;
  }
}

checkCollision() {
  if (
    map[this.getMapY()][this.getMapX()] == 1 ||
    map[this.getMapYRightSide()][this.getMapX()] == 1 ||
    map[this.getMapY()][this.getMapXRightSide()] == 1 ||
    map[this.getMapYRightSide()][this.getMapXRightSide()] == 1
  ) {
    return true;
  }
  return false;
}
```

```
checkGhostCollision() {
  for (let i = 0; i < ghosts.length; i++) {
    let ghost = ghosts[i];
    if (
      ghost.getMapX() == this.getMapX() &&
      ghost.getMapY() == this.getMapY()
    ) {
      return true;
    }
  }
  return false;
}

changeDirectionIfPossible() {
  if (this.direction == this.nextDirection) return;

  let tempDirection = this.direction;
  this.direction = this.nextDirection;
  this.moveForwards();
  if (this.checkCollision()) {
    this.moveBackwards();
    this.direction = tempDirection;
  } else {
    this.moveBackwards();
  }
}

changeAnimation() {
  this.currentFrame =
    this.currentFrame == this.frameCount ? 1 : this.currentFrame + 1;
}
```

```
draw() {
  canvasContext.save();
  canvasContext.translate(
    this.x + oneBlockSize / 2,
    this.y + oneBlockSize / 2
  );
  canvasContext.rotate((this.direction * 90 * Math.PI) / 180);

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

  canvasContext.drawImage(
    pacmanFrames,
    (this.currentFrame - 1) * oneBlockSize,
    0,
    oneBlockSize,
    oneBlockSize,
    this.x,
    this.y,
    this.width,
    this.height
  );

  canvasContext.restore();
}

getMapX() {
  return parseInt(this.x / oneBlockSize);
```

```
  }

  getMapY() {
    return parseInt(this.y / oneBlockSize);
  }

  getMapXRightSide() {
    return parseInt((this.x + 0.9999 * oneBlockSize) / oneBlockSize);
  }

  getMapYRightSide() {
    return parseInt((this.y + 0.9999 * oneBlockSize) / oneBlockSize);
  }
}

i;
```

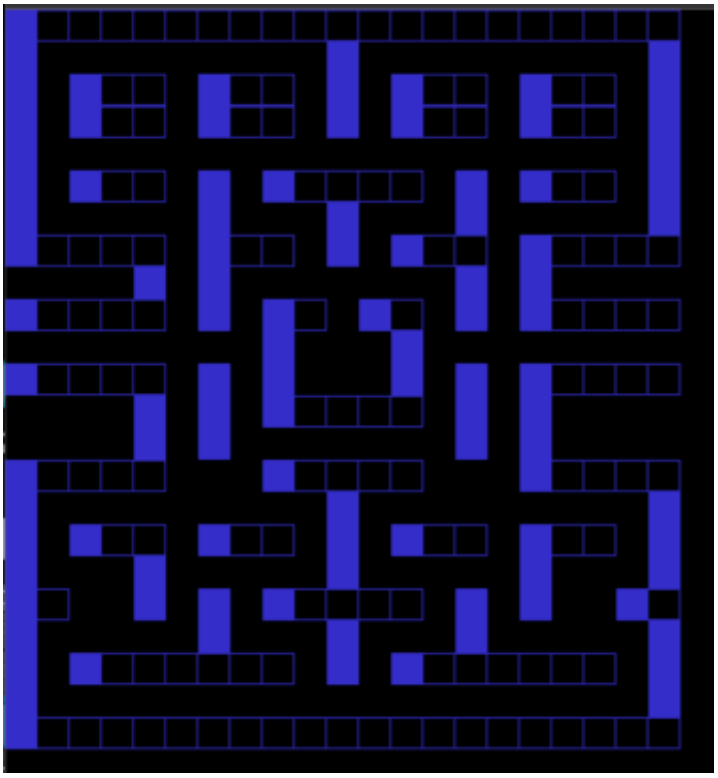
Index.html

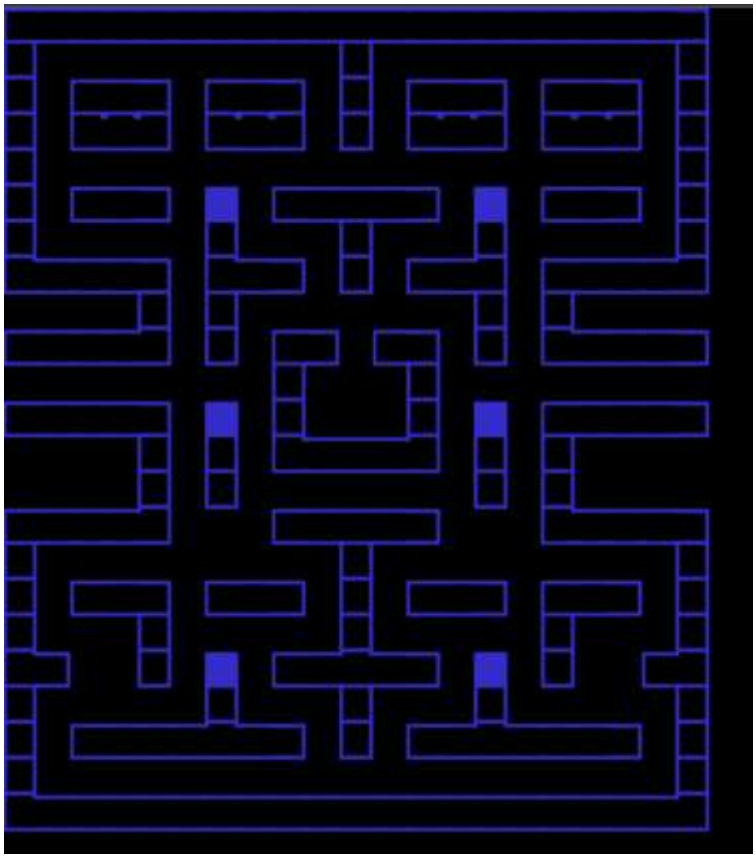
```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta http-equiv="X-UA-compatible" content="IE=edge" />
    <meta name="viewport" content="width=device-width, initial-
scale=1.0" />
    <title>Pacman</title>
  </head>

  <body
```

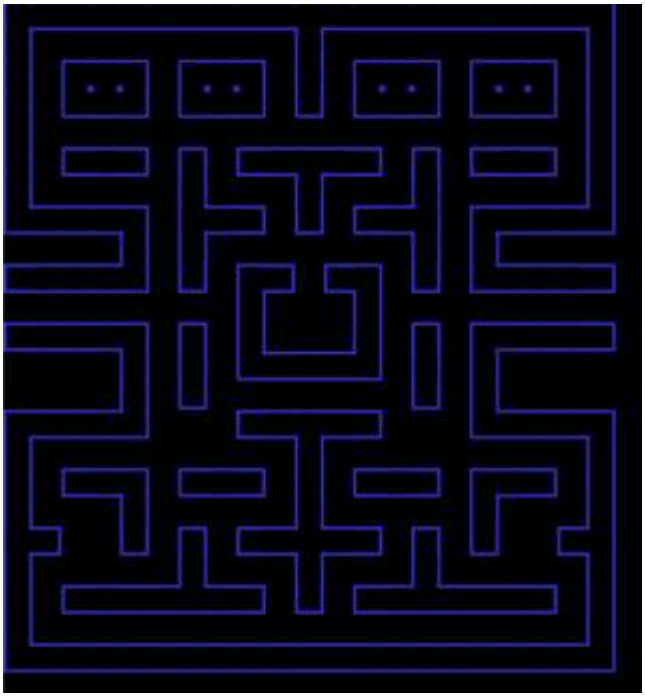
```
  style="
    margin: 0px;
    background-color: black;
    overflow-y: hidden;
    overflow-x: hidden;
  "
>
  <audio id="eat-sound" src="sounds/waka.wav"></audio>

  <canvas id="canvas" width="500" height="500"></canvas>
  <div style="display: none">
    <img id="animations" src="assets/animations.gif" alt="" />
    <img id="ghosts" src="assets/ghost.png" alt="" />
  </div>
  <sript src="ghost.js"></sript>
  <script src="pacman.js"></script>
  <script src="game.js"></script>
 </body>
</html>
```

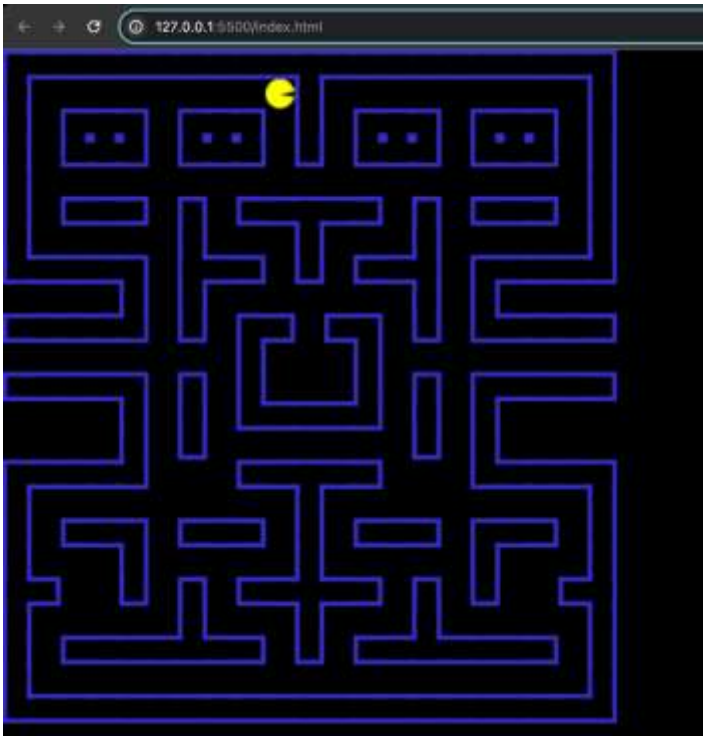
Development Photos

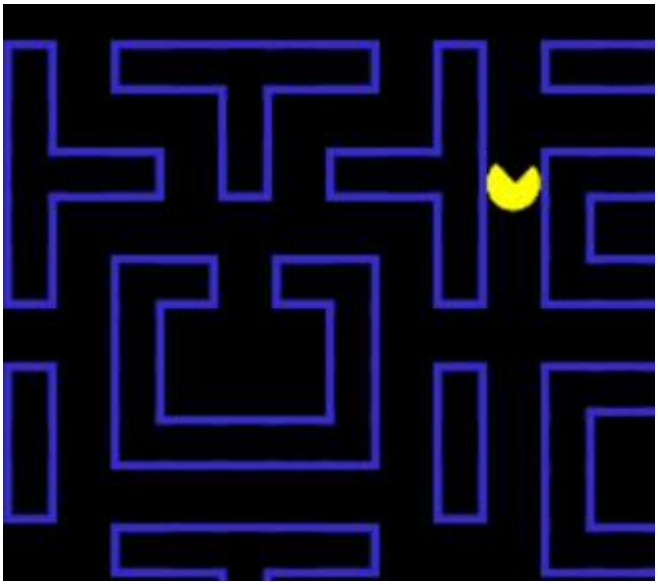
Development Photo’s during the developing of my Project.

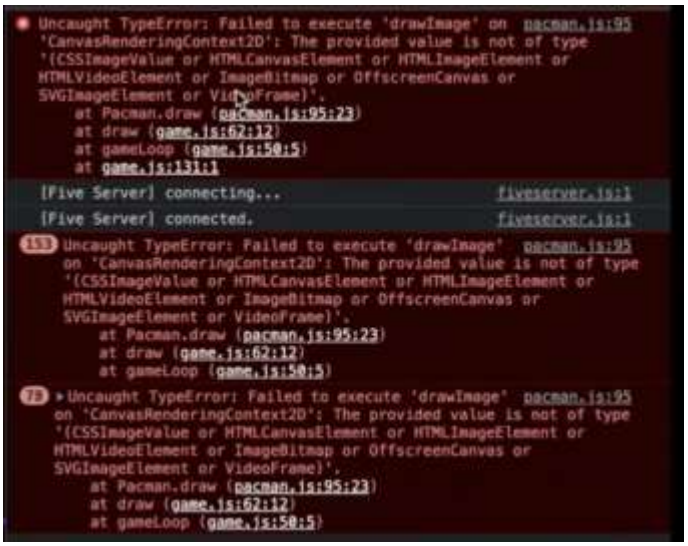


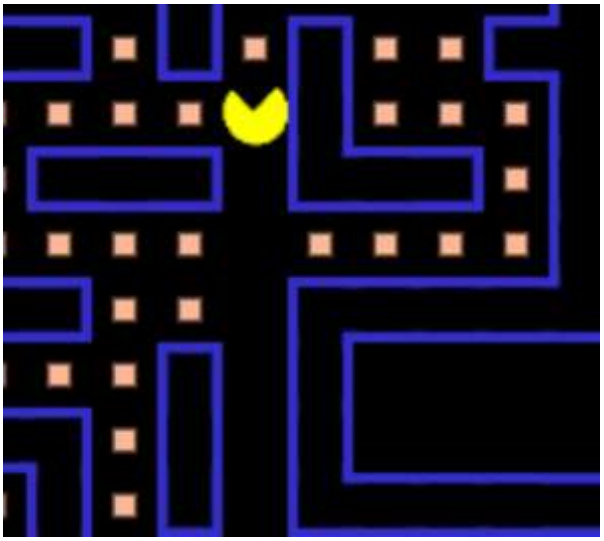


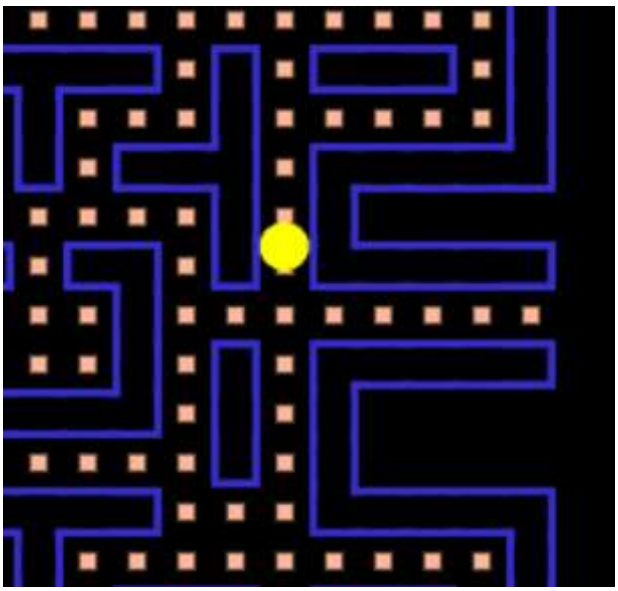


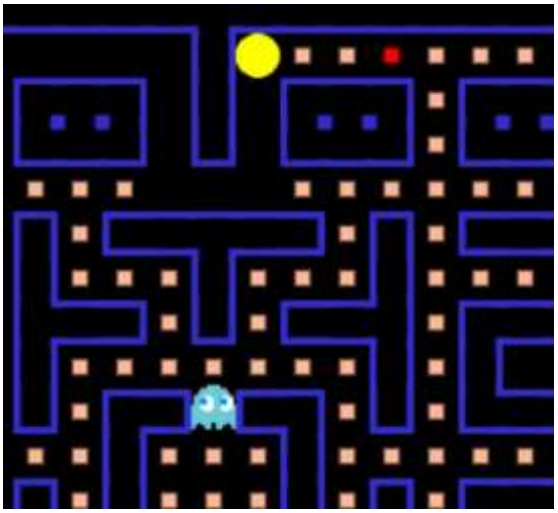


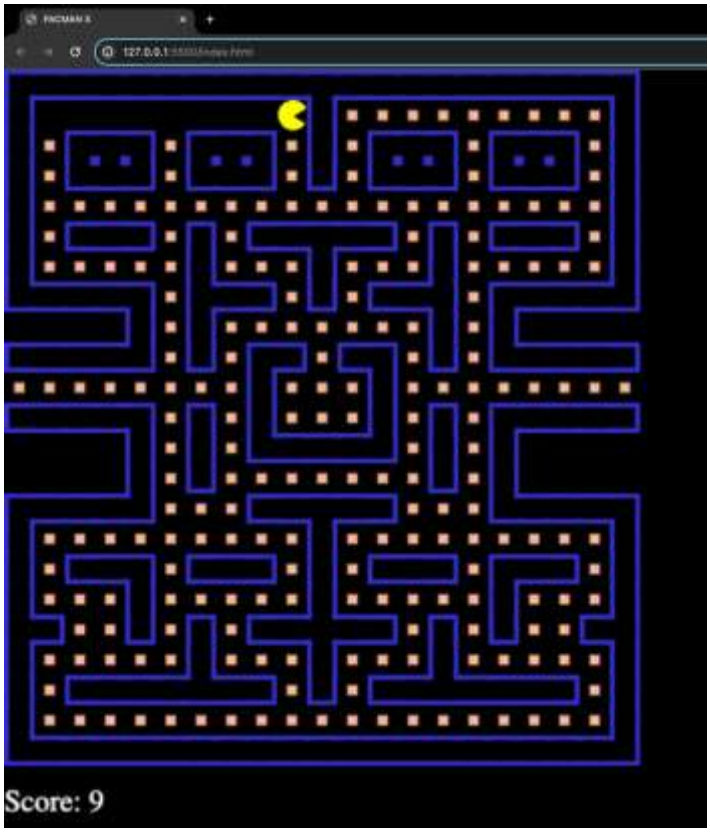


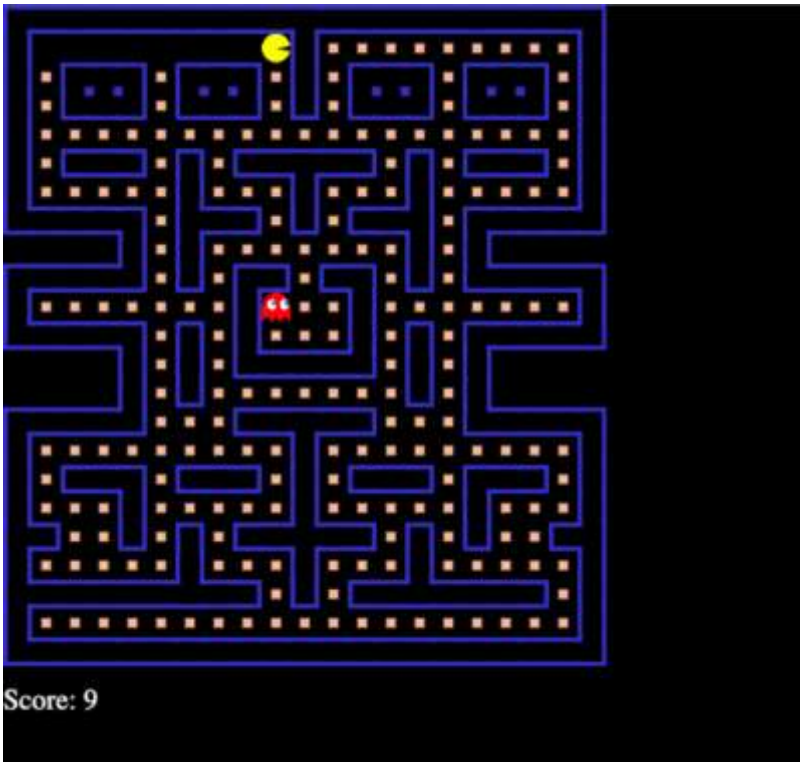

Uncaught TypeError: Failed to execute 'drawImage' on pacman.js:95
'CanvasRenderingContext2D': The provided value is not of type
'(CSSImageValue or HTMLCanvasElement or HTMLImageElement or
HTMLVideoElement or ImageBitmap or OffscreenCanvas or
SVGImageElement or VideoFrame)'.
at Pacman.draw (pacman.js:95:23)
at draw (game.js:62:12)
at gameLoop (game.js:50:5)
at game.js:131:1
[Five Server] connecting... fiveserver.js:1
[Five Server] connected. fiveserver.js:1
153 Uncaught TypeError: Failed to execute 'drawImage' pacman.js:95
on 'CanvasRenderingContext2D': The provided value is not of type
'(CSSImageValue or HTMLCanvasElement or HTMLImageElement or
HTMLVideoElement or ImageBitmap or OffscreenCanvas or
SVGImageElement or VideoFrame)'.
at Pacman.draw (pacman.js:95:23)
at draw (game.js:62:12)
at gameLoop (game.js:50:5)
73 Uncaught TypeError: Failed to execute 'drawImage' pacman.js:95
on 'CanvasRenderingContext2D': The provided value is not of type
'(CSSImageValue or HTMLCanvasElement or HTMLImageElement or
HTMLVideoElement or ImageBitmap or OffscreenCanvas or
SVGImageElement or VideoFrame)'.
at Pacman.draw (pacman.js:95:23)
at draw (game.js:62:12)
at gameLoop (game.js:50:5)

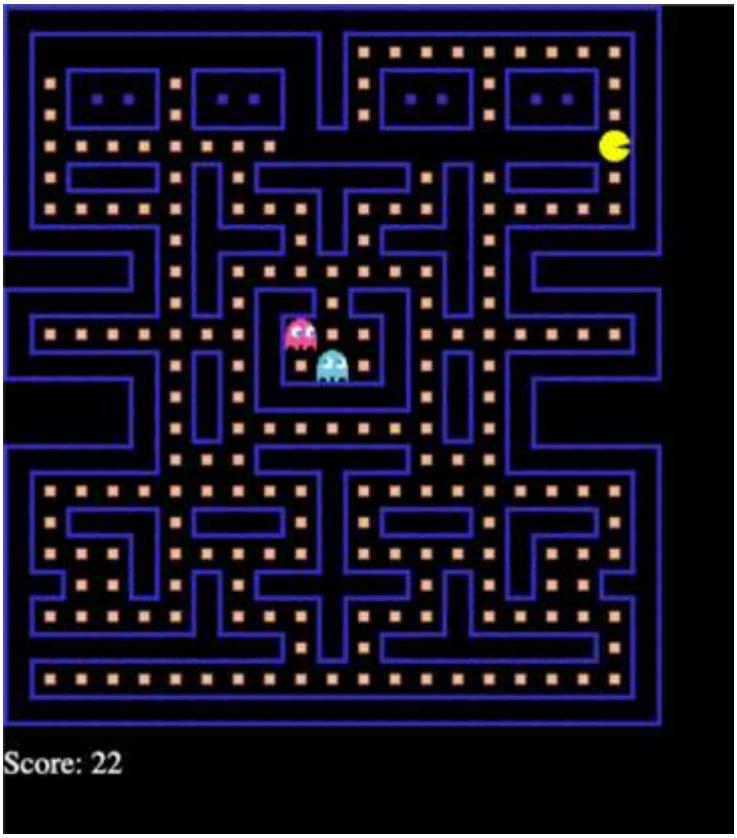


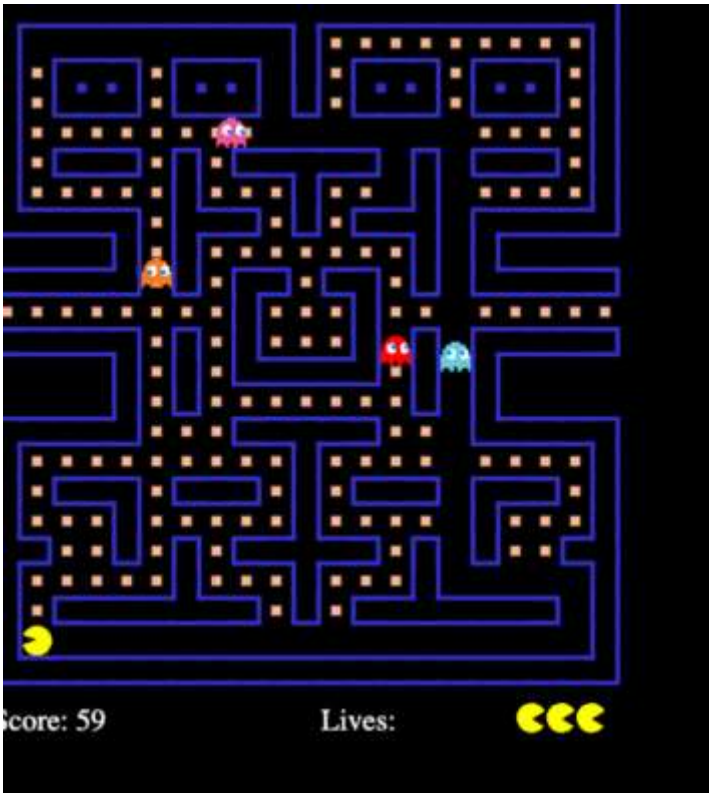




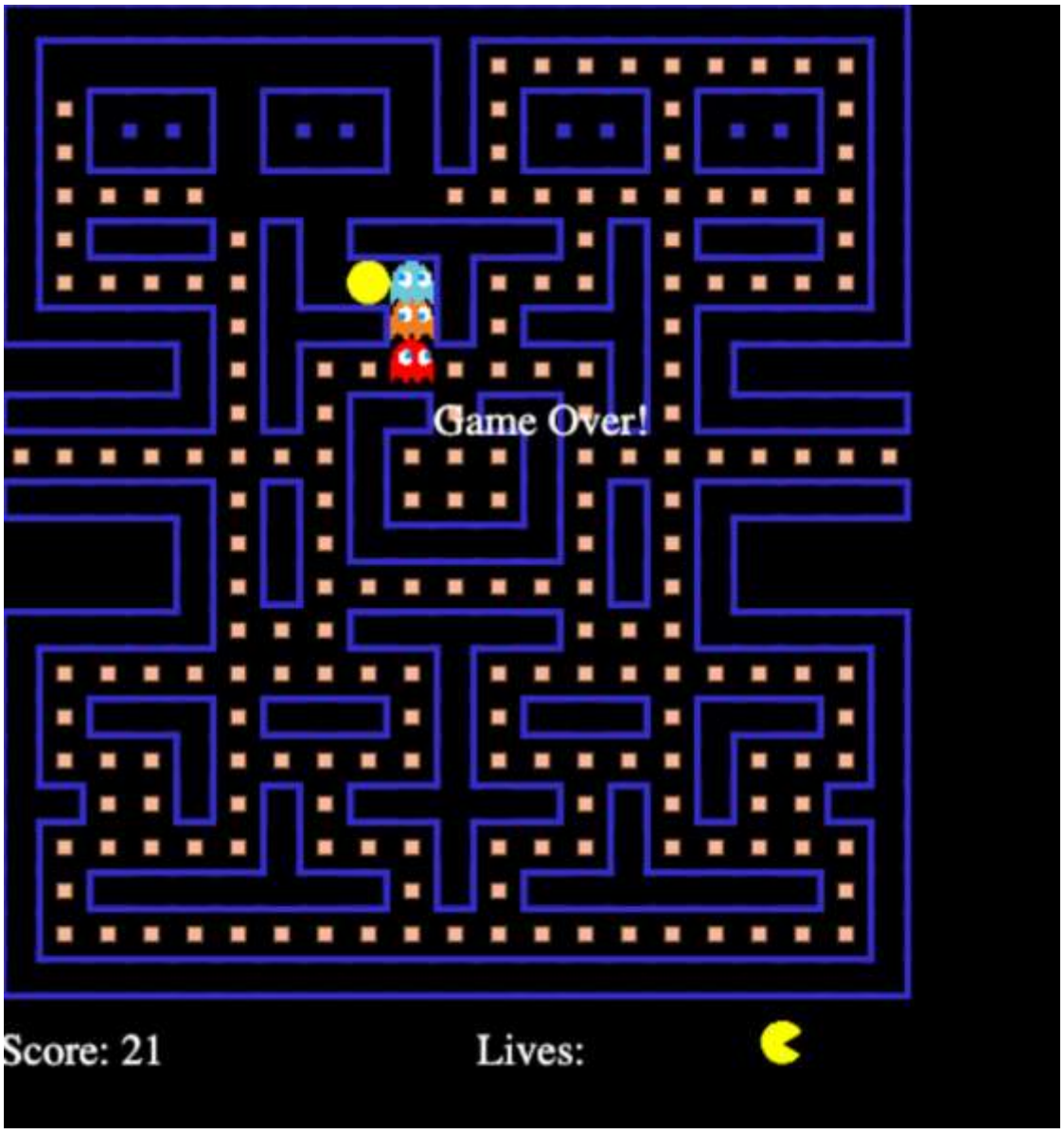

Score: 9


Score: 9


Score: 22


Lives:


Lives:


Game Over!
Score: 21
Lives:

Prototype

Pac-Man

Maze Wall

Yellow Pellets