**DAYANAND SAGAR UNIVERSITY**

**DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING**

**SCHOOL OF ENGINEERING DAYANANDA SAGAR UNIVERSITY**

**KUDLU GATE, BANGALORE – 560068**

**Mini project report on**

***“PAC-MAN GAME”***

**Design and Analysis of Algorithms (4th Semester)**

**Bachelor Of Technology (Computer Science & Engineering)**

**Submitted by**

**BHUMIKA MALLAPUR – ENG21CS0077**

**HARSHYARA BUKKAPATNAM – ENG21CS0085**

**CHIRAG ASWAL – ENG21CS0096**

**DEEPA S – ENG21CS0110**

**Under the supervision of**

**Prof. KAVYASHREE PATTAN**

****

**CERTIFICATE**

**This is to certify that the above bearing USN: ENG21CS0077 has satisfactorily completed his Mini Project as prescribed by the University for the 4th semester B.Tech program in Computer Science & Engineering (CSE) during the 2nd year at the School of Engineering, Dayananda Sagar University, Bangalore.**

**Date:**

**Signature of the faculty in-charge**

|  |  |
| --- | --- |
| **Max. Marks** | **Marks Obtained** |
|  |  |
|  |  |

**Signature of Chairman**

**Department of Computer Science & Engineering (CSE)**

****

**CERTIFICATE**

**This is to certify that the above bearing USN: ENG21CS0085 has satisfactorily completed his Mini Project as prescribed by the University for the 4th semester B.Tech program in Computer Science & Engineering (CSE) during the 2nd year at the School of Engineering, Dayananda Sagar University, Bangalore.**

**Date:**

**Signature of the faculty in-charge**

|  |  |
| --- | --- |
| **Max. Marks** | **Marks Obtained** |
|  |  |
|  |  |

**Signature of Chairman**

**Department of Computer Science & Engineering (CSE)**

****

**CERTIFICATE**

**This is to certify that the above bearing USN: ENG21CS0096 has satisfactorily completed his Mini Project as prescribed by the University for the 4th semester B.Tech program in Computer Science & Engineering (CSE) during the 2nd year at the School of Engineering, Dayananda Sagar University, Bangalore.**

**Date:**

**Signature of the faculty in-charge**

|  |  |
| --- | --- |
| **Max. Marks** | **Marks Obtained** |
|  |  |
|  |  |

**Signature of Chairman**

**Department of Computer Science & Engineering (CSE)**

****

**CERTIFICATE**

**This is to certify that the above bearing USN: ENG21CS0110 has satisfactorily completed his Mini Project as prescribed by the University for the 4th semester B.Tech program in Computer Science & Engineering (CSE) during the 2nd year at the School of Engineering, Dayananda Sagar University, Bangalore.**

**Date:**

**Signature of the faculty in-charge**

|  |  |
| --- | --- |
| **Max. Marks** | **Marks Obtained** |
|  |  |
|  |  |

**Signature of Chairman**

**Department of Computer Science & Engineering (CSE)**

**DECLARATION**

**We hereby declare that the work presented in this Mini Project entitled *“PAC-MAN GAME”*, has been carried out by us and it has not been submitted for the award of any degree, diploma or the Mini Project of any other college or university.**

**BHUMIKA MALLAPUR – ENG21CS0077**

**HARSHYARA BUKKAPATNAM – ENG21CS0085**

**CHIRAG ASWAL – ENG21CS0096**

**DEEPA S – ENG21CS0110**

**ACKNOWLEDGEMENT**

**We would like to express our sincere gratitude to all those who have contributed to the completion of our mini project. Their support and assistance have been invaluable.**

**First and foremost, we would like to thank our project supervisor for their guidance and expertise throughout this project. Their insights and feedback have greatly influenced the quality of our work.**

**We would also like to thank our teammates for their hard work and cooperation. Their dedication and collaboration have been instrumental in the successful completion of this project.**

**Furthermore, we would like to acknowledge the support and contributions of our friends and classmates. Their suggestions and encouragement have been greatly appreciated.**

**Lastly, we are grateful for the opportunity to work on this mini project, and we sincerely thank everyone involved.**

**BHUMIKA MALLAPUR – ENG21CS0077**

**HARSHYARA BUKKAPATNAM – ENG21CS0085**

**CHIRAG ASWAL – ENG21CS0096**

**DEEPA S – ENG21CS0110**

**CONTENTS**

|  |  |  |
| --- | --- | --- |
| **S.No** |  | **Page No.** |
| **1.** | **Cover page** |  |
| **2.** | **Certificate** |  |
| **3.** | **Declaration** |  |
| **4.** | **Acknowledge** |  |
| **5.** | **Contents** |  |
|  | **TOPICS** |  |
| **1.** | **Introduction** |  |
| **1.1** | **About The Problem** |  |
| **1.2** | **About the DAA technique** |  |
| **2.** | **Problem Statement** |  |
| **3.** | **Literature Review** |  |
| **4.** | **S/W&H/W Requirement** |  |
| **5.** | **Design** |  |
| **5.1** | **Pseudo code Of Algorithm** |  |
| **5.2** | **Flow Chart** |  |
| **6.** | **Implementation(coding)** |  |
| **7.** | **Testing** |  |
| **8.** | **Output Screen Shots** |  |
| **9.** | **Conclusion** |  |
| **10.** | **References** |  |

1. **INTRODUCTION**

Pac-Man, the iconic arcade game created by Namco in 1980, has become a timeless classic in the world of gaming. With its simple yet addictive gameplay and memorable characters, Pac-Man has captured the hearts of millions across generations.

In this beloved maze chase game, players take on the role of Pac-Man, a cheerful yellow character with an insatiable appetite for pellets. The objective is to guide Pac-Man through a maze, gobbling up pellets while avoiding colourful ghosts who relentlessly pursue him. As players navigate the maze and strategically consume power pellets, they gain temporary invincibility and the ability to turn the tables on the ghosts, transforming them into vulnerable targets.

With its clever level design, strategic decision-making, and engaging gameplay mechanics, Pac-Man has stood the test of time as a true gaming classic that continues to captivate players even to this day.

* 1. **About The Problem**

The problem at hand is to develop a Pac-Man game that captures the essence of the original arcade classic while incorporating modern features and enhancements. The challenge lies in recreating the addictive gameplay mechanics of navigating through mazes, eating pellets, and avoiding ghosts, all while balancing the difficulty to keep players engaged.

The objective is to provide an immersive gaming experience that appeals to both nostalgic fans of the original game and new players alike. This entails designing intuitive controls for the ghosts, creating visually appealing and dynamic mazes, add depth and excitement to the gameplay.

Additionally, ensuring smooth performance across various platforms and optimizing the game for different screen sizes and resolutions presents its own set of technical challenges. The goal is to deliver a Pac-Man game that captures the magic of the original while bringing new elements that will thrill and entertain players in the modern era.

* 1. **About DAA Technique**

COLLISION DETECTION

1. In the Pac-Man game, collision detection plays a vital role in determining interactions between Pac-Man, the ghosts, and various game elements.

2. The collision detection system ensures that when Pac-Man moves across the maze, it accurately detects collisions with walls, pellets, power-ups, and other objects.

3. It is crucial for the collision detection algorithm to detect when Pac-Man comes into contact with ghosts, triggering appropriate game events such as losing a life or gaining points.

4. The collision detection mechanism must consider the size and shape of Pac-Man and the ghosts, accurately detecting collisions between their bounding boxes or hitboxes.

5. Efficient collision detection algorithms can optimize the game's performance, allowing for smooth gameplay even in fast-paced situations with multiple objects interacting at the same time.

DIJKSTRA’S ALGORITHM

1. Dijkstra's algorithm is a fundamental graph traversal algorithm that can be applied to solve pathfinding problems in the Pac-Man game.

2. By implementing Dijkstra's algorithm, Pac-Man can intelligently navigate the maze, finding the shortest path to reach its destination while avoiding obstacles and ghosts.

3. The algorithm works by assigning a cost value to each tile in the maze and iteratively updating the costs based on the optimal path found so far.

4. Dijkstra's algorithm ensures that Pac-Man explores the maze efficiently, considering all possible paths and selecting the one with the lowest cumulative cost.

5. By employing Dijkstra's algorithm, Pac-Man can make intelligent decisions, maximizing its chances of successfully completing levels and achieving high scores.

1. **PROBLEM STATEMENT**

Develop a Pac-Man game where players control the iconic character through a maze, eating pellets and avoiding ghosts to achieve the highest score possible. Implement challenging gameplay mechanics and dynamic mazes to provide an engaging and immersive gaming experience. The objective is to captivate players with a modernized version of the classic Pac-Man, combining nostalgia with exciting new elements.

1. **LITERATURE REVIEW**

Pac-Man Developed Using HTML, CSS, and JavaScript

**Introduction:**

The development of Pac-Man games using HTML, CSS, and JavaScript has gained significant attention in recent years. This literature review aims to explore existing research, studies, and articles that discuss the implementation and effectiveness of utilizing these technologies for creating Pac-Man games. By examining various sources, we gain insights into the challenges, design considerations, user experience, and performance optimization techniques in developing Pac-Man using HTML, CSS, and JavaScript.

**HTML, CSS, and JavaScript in Game Development:**

HTML, CSS, and JavaScript have emerged as popular technologies for web-based game development due to their versatility, cross-platform compatibility, and accessibility. Research indicates that these technologies offer an accessible entry point for game development, allowing developers to create engaging and interactive gaming experiences with relatively low barriers to entry.

**Pac-Man Game Development:**

Studies highlight different approaches and methodologies for implementing the Pac-Man game using HTML, CSS, and JavaScript. For example, researchers have explored the utilization of HTML5 canvas and CSS-based grid systems to construct the maze, while JavaScript is employed to handle game logic, character movement, collision detection, and scoring mechanisms. These studies emphasize the importance of properly structuring the code and utilizing design patterns to facilitate maintainability and extensibility in Pac-Man game development.

**User Experience and Gameplay Mechanics:**

User experience plays a crucial role in the success of Pac-Man games. Research has focused on evaluating the usability and player engagement aspects of Pac-Man games developed using HTML, CSS, and JavaScript. Factors such as responsive controls, smooth character movement, intuitive navigation, and effective collision detection have been identified as essential elements for a satisfying gaming experience. Studies have also explored the impact of gameplay mechanics, such as ghost performance, on player enjoyment and challenge.

**Visual Design and Animation:**

Visual design and animation significantly contribute to the overall aesthetics and immersive experience of Pac-Man games. Scholars have investigated techniques for creating visually appealing mazes, implementing sprite animations for Pac-Man and ghosts, and enhancing visual effects using HTML, CSS, and JavaScript. The use of CSS animations, transitions, and transforms have been explored to bring the game elements to life and create a visually captivating experience for players.

**Performance Optimization and Cross-Browser Compatibility:**

To ensure optimal gameplay performance and compatibility across various platforms and devices, researchers have explored strategies for optimizing Pac-Man games developed with HTML, CSS, and JavaScript. Techniques such as code optimization, image compression, and caching have been investigated to minimize load times and improve responsiveness. Additionally, studies have examined methods for achieving cross-browser compatibility to provide a consistent gaming experience across different web browsers and operating systems.

**Conclusion and Future Directions:**

The literature review highlights the growing interest in developing Pac-Man games using HTML, CSS, and JavaScript. Research indicates that these technologies offer a flexible and accessible platform for creating engaging and visually appealing gaming experiences. However, further investigation is needed to explore advanced topics such as multiplayer functionality, artificial intelligence for ghost behaviour and integration with backend systems. Additionally, future research can focus on evaluating the impact of emerging web technologies and frameworks on Pac-Man game development using HTML, CSS, and JavaScript.

Overall, the literature review provides valuable insights into the current state of Pac-Man game development using HTML, CSS, and JavaScript, laying the foundation for future advancements and improvements in this domain.

1. **S/W&H/W REQUIREMENT**

Software and Hardware Requirements for Pac-Man Developed Using HTML, CSS, and JavaScript

Software Requirements:

* Visual Studio Code is required for writing and editing HTML, CSS, and JavaScript code.
* Web Browser
* HTML, CSS, and JavaScript Libraries Depending on the complexity of the Pac-Man game

Hardware Requirements:

* A computer or laptop
* Display and Input
* Internet Connection
* Storage

These requirements provide a general guideline for setting up the development environment and ensuring the game can be played on standard computing devices.

1. **DESIGN**

**(5.1) Algorithm**

The given code represents a class called ‘Ghost’ in a Pacman game. The ‘Ghost’ class has several methods and properties that control the behaviour and movement of the ghost characters in the game. Here's a breakdown of the algorithmic flow within the ‘Ghost’ class:

1. The ‘constructor’ function initializes the ghost's position (x, y), size (width, height), speed, image properties (imageX, imageY, imageWidth, imageHeight), range, and target. It also sets a timer to change the ghost's random direction periodically.

2. The ‘isInRange()’ function calculates the distance between the ghost and the Pacman character. If the distance is within the defined range, it returns `true`, indicating that the pac-man is within range of the ghost.

3. The ‘changeRandomDirection()’ function randomly selects a new target index for the ghost's movement direction:

randomTargetIndex+=1;

randomTargetIndex%=4;

4. The ‘moveProcess()’ function controls the ghost's movement. If the ghost is within range of the Pacman, it sets the Pacman as the new target. Otherwise, it selects a random target from the predefined targets for ghosts. The function then attempts to change the ghost's direction based on the new target. It moves the ghost forwards and checks for collisions. If a collision occurs, it moves the ghost backwards.

5.The ‘moveBackwards()’ function moves the ghost in the opposite direction of its current direction.

Right: x-=speed

Up: y+=speed

Left: x+=speed

Bottom: y-=speed

6. The ‘moveForwards()’ function moves the ghost in its current direction.

7. The ‘checkCollisions()’ function checks if the ghost has collided with a wall or obstacle on the game map.

8. The ‘changeDirectionIfPossible()’ function attempts to calculate a new direction for the ghost based on the game map, the target's position, and the ghost's current position. It also handles cases where the ghost needs to change its direction due to wall collisions.

9. The ‘calculateNewDirection()’ function uses Dijkstra's algorithm to calculate the shortest path from the ghost's current position to the target position on the game map. It creates a queue of positions to explore and checks neighboring positions until the target position is reached.

10. The ‘addNeighbors()’ function adds the neighboring positions of the current position to the exploration queue.

11. The ‘getMapX()’ and ‘getMapY()’ functions calculate the ghost's position on the game map based on its current coordinates.

12. The ‘getMapXRightSide()’ and ‘getMapYRightSide()’ functions calculate the ghost's position on the game map based on its current coordinates, considering the right side of the ghost's sprite.

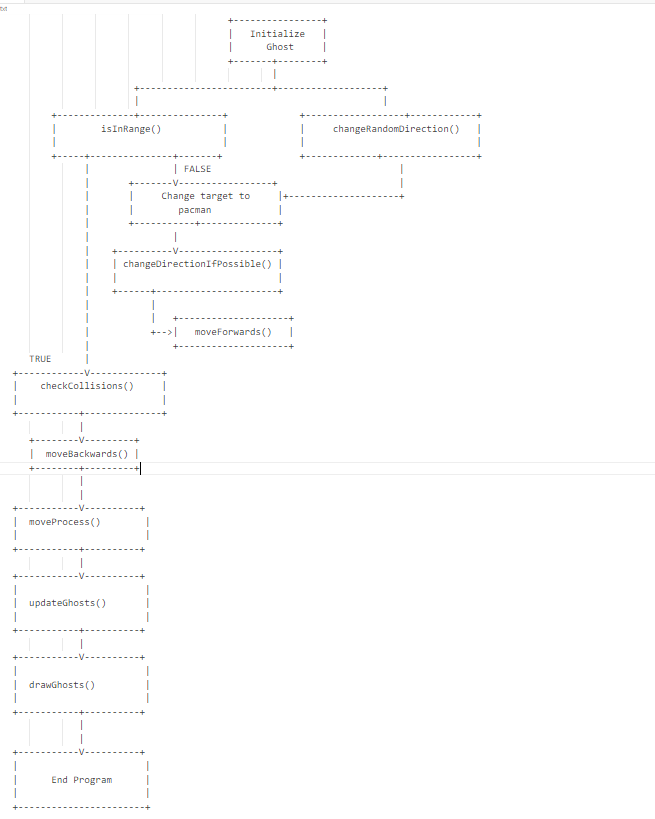
13. The ‘changeAnimation()’ function updates the ghost's animation frame.

14. The ‘draw()’ function draws the ghost on the game canvas and displays its range as a stroke circle.

15. ‘updateGhosts()’ and ‘drawGhosts()’ are helper functions that iterate through the ‘ghosts’ array and call the respective methods to update and draw each ghost in the game.

This is a high-level overview of the algorithmic flow within the `Ghost` class. It's important to note that the code provided assumes the existence of other variables, functions, and objects related to the Pacman game that are not shown here.

**(5.2) Flowchart**

****

1. **IMPLEMENTATION**

For landing page

!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>

    <link rel="stylesheet" href="newss.css">

</head>

<body>

    <header>

        <img src="3dpacc.jpg" width="100%">

        <h1 style="color:rgb(252, 253, 255);font-size:90px;font-style: italic;text-align: center;font-weight: bold;">Pac-Man : Classic Arcade</h1>

       <p style="text-align: center;"></p>

        <div class="navbar">

         <a href='index.html'>

            <button>Play Online</button></a>

         <a href="about.html"><button>About Pacman</button></a>

        <div class="audio-container">

                <body onclick="document.getElementsByTagName('audio')[0].play()">

                    <audio src="pacman\_beginning.wav">

                  </body>

            </div>

        </div>

    </header>

    <div id="loadeer">

    </div>

    <script>

        var loader=document.getElementById("loadeer");

        window.addEventListener("load",function(load){

            window.removeEventListener('load', load, false);

  setTimeout(function(){loader.style.display = 'none'},2000);

},false);

    </script>

</body>

</html>

For about page

<!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>Document</title>

</head>

<body style="background-color: rgb(8, 8, 9);color: azure;">

<div id="page" style="height: auto !important;">

    <div id="page-wrap" style="height: auto !important;">

      <div class="page-left">

<h1>Pac-Man : Classic Arcade</h1>

<div class="image-wrap-pac alignleft"><a href="pppaaa.jpg" class="fancybox" data-fancybox-group="pageimage"><img src="pppaaa.jpg" width="300" height="200" alt="Pac-Man Arcade Screenshot"></a><br>Pac-Man Arcade</div>

<p>In 1980, Namco created a character that was small, round and yellow. It's mouth consisted of a piece cut out of the main circle, like a slice of cake. Pac-Man was born. It was a huge hit in the arcades across the globe, and Pac-Man is still going strong today. With every new generation of game systems, Pac-Man continues to be part of the gaming community.</p>

<h2>Pac-Man's Release and Popularity</h2>

<p>When Pac-Man was released in 1980, the most popular arcade games were space shooters, in particular Space Invaders and Asteroids. The most visible minority were sports games that were mostly derivative of Pong. Pac-Man succeeded by creating a new genre and appealing to both genders. Pac-Man is often credited with being a landmark in video game history, and is among the most famous arcade games of all time. The character also appears in more than 30 officially licensed game spin-offs, as well as in numerous unauthorized clones and bootlegs. Pac-Man is one of the longest running video game franchises from the golden age of video arcade games, and one of only three video games that are on display at the Smithsonian in Washington D.C., (along with Pong and Dragon's Lair). </p>

      <p>There are 4 primary ghosts that you can find in Pacman. These are named Blinky, Pinky, Inky and Clyde. They also have specific names in Japanese too, as well as alternate names. Blinky is also named Oikake and he is a chaser. His other names are Urchin, Machy or Shadow. Pinky is Machibuse, Rop, Micky or Speedy. Then you have Inky which is also called Kimagure, Mucky or the Stylist. Lastly there’s Clyde which is orange based, he is called Otoboke, Crybaby, Mocky or Pokey.

</p>

<img src="ghooo.png"></div>

For game page

<!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: 0; background-color: black;">

    <canvas id="canvas" width="500" height="500"></canvas>

    <div style="display:none;">

        <img id="animation" src="animations.gif" width="140" height="20">

        <img id="ghosts" src="ghost.png" width="140" height="20">

    </div>

    <script src="ghost.js"></script>

    <script src="pacman.js"></script>

    <script src="game.js"></script>

</body>

</html>

game.js

const canvas = document.getElementById("canvas");

const canvasContext = canvas.getContext("2d");

const pacmanFrames = document.getElementById("animation");

const ghostFrames = document.getElementById("ghosts");

let createRect = (x, y, width, height, color) => {

    canvasContext.fillStyle = color;

    canvasContext.fillRect(x, y, width, height);

};

const DIRECTION\_RIGHT = 4;

const DIRECTION\_UP = 3;

const DIRECTION\_LEFT = 2;

const DIRECTION\_BOTTOM = 1;

let lives = 3;

let ghostCount = 4;

let ghostImageLocations = [

    { x: 0, y: 0 },

    { x: 176, y: 0 },

    { x: 0, y: 121 },

    { x: 176, y: 121 },

];

// Game variables

let fps = 30;

let pacman;

let oneBlockSize = 20;

let score = 0;

let ghosts = [];

let wallSpaceWidth = oneBlockSize / 1.6;

let wallOffset = (oneBlockSize - wallSpaceWidth) / 2;

let wallInnerColor = "black";

// we now create the map of the walls,

// if 1 wall, if 0 not wall

// 21 columns // 23 rows

let map = [

 [1, 1, 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, 2, 2, 1],

 [1, 2, 1, 1, 1, 2, 1, 1, 1, 2, 1, 2, 1, 1, 1, 2, 1, 1, 1, 2, 1],

 [1, 2, 1, 1, 1, 2, 1, 1, 1, 2, 1, 2, 1, 1, 1, 2, 1, 1, 1, 2, 1],

 [1, 2, 2, 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, 2, 1],

 [1, 2, 2, 2, 2, 2, 1, 2, 2, 2, 1, 2, 2, 2, 1, 2, 2, 2, 2, 2, 1],

 [1, 1, 1, 1, 1, 2, 1, 1, 1, 2, 1, 2, 1, 1, 1, 2, 1, 1, 1, 1, 1],

 [0, 0, 0, 0, 1, 2, 1, 2, 2, 2, 2, 2, 2, 2, 1, 2, 1, 0, 0, 0, 0],

 [1, 1, 1, 1, 1, 2, 1, 2, 1, 1, 2, 1, 1, 2, 1, 2, 1, 1, 1, 1, 1],

 [2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2],

 [1, 1, 1, 1, 1, 2, 1, 2, 1, 2, 2, 2, 1, 2, 1, 2, 1, 1, 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, 0, 0, 1, 2, 1, 2, 2, 2, 2, 2, 2, 2, 1, 2, 1, 0, 0, 0, 0],

 [1, 1, 1, 1, 1, 2, 2, 2, 1, 1, 1, 1, 1, 2, 2, 2, 1, 1, 1, 1, 1],

 [1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 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, 2, 1],

 [1, 2, 2, 2, 1, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 1, 2, 2, 2, 1],

 [1, 1, 2, 2, 1, 2, 1, 2, 1, 1, 1, 1, 1, 2, 1, 2, 1, 2, 2, 1, 1],

 [1, 2, 2, 2, 2, 2, 1, 2, 2, 2, 1, 2, 2, 2, 1, 2, 2, 2, 2, 2, 1],

 [1, 2, 1, 1, 1, 1, 1, 1, 1, 2, 1, 2, 1, 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, 2, 1],

[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],

];

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 createNewPacman = () => {

    pacman = new Pacman(

        oneBlockSize,

        oneBlockSize,

        oneBlockSize,

        oneBlockSize,

        oneBlockSize / 5);};

let gameLoop = () => {

    update();

    draw();};

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

let restartPacmanAndGhosts = () => {

    createNewPacman();

    createGhosts();};

let onGhostCollision = () => {

    lives--;

    restartPacmanAndGhosts();

    if (lives == 0) {}};

let update = () => {

    pacman.moveProcess();

    pacman.eat();

    updateGhosts();

    if (pacman.checkGhostCollision(ghosts)) {

        onGhostCollision();}};

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,

                    "#FEB897");}}}};

let drawRemainingLives = () => {

    canvasContext.font = "20px Emulogic";

    canvasContext.fillStyle = "white";

    canvasContext.fillText("Lives: ", 220, oneBlockSize \* (map.length + 1));

    for (let i = 0; i < lives; i++) {

        canvasContext.drawImage(

            pacmanFrames,

            2 \* oneBlockSize,

            0,

            oneBlockSize,

            oneBlockSize,

            350 + i \* oneBlockSize,

            oneBlockSize \* map.length + 2,

            oneBlockSize,

            oneBlockSize); }};

let drawScore = () => {

    canvasContext.font = "20px Emulogic";

    canvasContext.fillStyle = "white";

    canvasContext.fillText("Score: " + score,

        0,

        oneBlockSize \* (map.length + 1));};

let draw = () => {canvasContext.clearRect(0, 0, canvas.width, canvas.height);

    createRect(0, 0, canvas.width, canvas.height, "black");

    drawWalls();

    drawFoods();

    drawGhosts();

    pacman.draw();

    drawScore();

drawRemainingLives();};

let drawWalls = () => {

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

        for (let j = 0; j < map[0].length; j++) {

            if (map[i][j] == 1) {

                createRect(

                    j \* oneBlockSize,

                    i \* oneBlockSize,

                    oneBlockSize,

                    oneBlockSize,

                    "#342DCA");

                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 < map.length - 1 && map[i + 1][j] == 1) {createRect

(j \* oneBlockSize + wallOffset,

                        i \* oneBlockSize + wallOffset,

                        wallSpaceWidth,

                        wallSpaceWidth + wallOffset,

                        wallInnerColor );}

   if (i > 0 && map[i - 1][j] == 1) {

                    createRect(

                        j \* oneBlockSize + wallOffset,

                        i \* oneBlockSize,

                        wallSpaceWidth,

                        wallSpaceWidth + wallOffset,

                        wallInnerColor);}}}}};

let createGhosts = () => {

    ghosts = [];

    for (let i = 0; i < ghostCount \* 2; 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,

            ghostImageLocations[i % 4].x,

            ghostImageLocations[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 arrow or a

            pacman.nextDirection = DIRECTION\_LEFT;

        } else if (k == 38 || k == 87) {

            // up arrow or w

            pacman.nextDirection = DIRECTION\_UP;

        } else if (k == 39 || k == 68) {

            // right arrow or d

            pacman.nextDirection = DIRECTION\_RIGHT;

        } else if (k == 40 || k == 83) {

            // bottom arrow or s

            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() \* 4);

        this.target = randomTargetsForGhosts[this.randomTargetIndex];

        setInterval(() => {

            this.changeRandomDirection();

        }, 10000);

    }

    isInRange() {

        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;

    }

    changeRandomDirection() {

        let addition = 1;

        this.randomTargetIndex += addition;

        this.randomTargetIndex = this.randomTargetIndex % 4;

    }

    moveProcess() {

        if (this.isInRange()) {

            this.target = pacman;

        } else {

            this.target = randomTargetsForGhosts[this.randomTargetIndex];

        }

        this.changeDirectionIfPossible();

        this.moveForwards();

        if (this.checkCollisions()) {

            this.moveBackwards();

            return;

        }

    }

    moveBackwards() {

        switch (this.direction) {

            case 4: // Right

                this.x -= this.speed;

                break;

            case 3: // Up

                this.y += this.speed;

                break;

            case 2: // Left

                this.x += this.speed;

                break;

            case 1: // Bottom

                this.y -= this.speed;

                break;

        }

    }

    moveForwards() {

        switch (this.direction) {

            case 4: // Right

                this.x += this.speed;

                break;

            case 3: // Up

                this.y -= this.speed;

                break;

            case 2: // Left

                this.x -= this.speed;

                break;

            case 1: // Bottom

                this.y += this.speed;

                break;

        }

    }

    checkCollisions() {

        let isCollided = false;

        if (

            map[parseInt(this.y / oneBlockSize)][

                parseInt(this.x / oneBlockSize)

            ] == 1 ||

            map[parseInt(this.y / oneBlockSize + 0.9999)][

                parseInt(this.x / oneBlockSize)

            ] == 1 ||

            map[parseInt(this.y / oneBlockSize)][

                parseInt(this.x / oneBlockSize + 0.9999)

            ] == 1 ||

            map[parseInt(this.y / oneBlockSize + 0.9999)][

                parseInt(this.x / oneBlockSize + 0.9999)

            ] == 1

        ) {

            isCollided = true;

        }

        return isCollided;

    }

    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;

        }

        if (

            this.getMapY() != this.getMapYRightSide() &&

            (this.direction == DIRECTION\_LEFT ||

                this.direction == DIRECTION\_RIGHT)

        ) {

            this.direction = DIRECTION\_UP;

        }

        if (

            this.getMapX() != this.getMapXRightSide() &&

            this.direction == DIRECTION\_UP

        ) {

            this.direction = DIRECTION\_LEFT;

        }

        this.moveForwards();

        if (this.checkCollisions()) {

            this.moveBackwards();

            this.direction = tempDirection;

        } else {

            this.moveBackwards();

        }

        console.log(this.direction);

    }

    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(),

                rightX: this.getMapXRightSide(),

                rightY: this.getMapYRightSide(),

                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 1; // direction

    }

    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 < numOfColumns &&

            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 < numOfColumns &&

            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;

    }

    getMapX() {

        let mapX = parseInt(this.x / oneBlockSize);

        return mapX;

    }

    getMapY() {

        let mapY = parseInt(this.y / oneBlockSize);

        return mapY;

    }

    getMapXRightSide() {

        let mapX = parseInt((this.x \* 0.99 + oneBlockSize) / oneBlockSize);

        return mapX;

    }

    getMapYRightSide() {

        let mapY = parseInt((this.y \* 0.99 + oneBlockSize) / oneBlockSize);

        return mapY;

    }

    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();

        canvasContext.beginPath();

        canvasContext.arc(

            this.x + oneBlockSize / 2,

            this.y + oneBlockSize / 2,

            this.range \* oneBlockSize,

            0,

            2 \* Math.PI

        );

        canvasContext.stroke();

    }

}

let updateGhosts = () => {

    for (let i = 0; i < ghosts.length; i++) {

        ghosts[i].moveProcess();

    }

};

let drawGhosts = () => {

    for (let i = 0; i < ghosts.length; i++) {

        ghosts[i].draw();

    }

};

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 = 4;

        this.nextDirection = 4;

        this.frameCount = 7;

        this.currentFrame = 1;

        setInterval(() => {

            this.changeAnimation();

        }, 100);

    }

    moveProcess() {

        this.changeDirectionIfPossible();

        this.moveForwards();

        if (this.checkCollisions()) {

            this.moveBackwards();

            return;

        }

    }

    eat() {

        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++;

                }

            }

        }

    }

    moveBackwards() {

        switch (this.direction) {

            case DIRECTION\_RIGHT: // Right

                this.x -= this.speed;

                break;

            case DIRECTION\_UP: // Up

                this.y += this.speed;

                break;

            case DIRECTION\_LEFT: // Left

                this.x += this.speed;

                break;

            case DIRECTION\_BOTTOM: // Bottom

                this.y -= this.speed;

                break;

        }

    }

    moveForwards() {

        switch (this.direction) {

            case DIRECTION\_RIGHT: // Right

                this.x += this.speed;

                break;

            case DIRECTION\_UP: // Up

                this.y -= this.speed;

                break;

            case DIRECTION\_LEFT: // Left

                this.x -= this.speed;

                break;

            case DIRECTION\_BOTTOM: // Bottom

                this.y += this.speed;

                break;

        }

    }

    checkCollisions() {

        let isCollided = false;

        if (

            map[parseInt(this.y / oneBlockSize)][

                parseInt(this.x / oneBlockSize)

            ] == 1 ||

            map[parseInt(this.y / oneBlockSize + 0.9999)][

                parseInt(this.x / oneBlockSize)

            ] == 1 ||

            map[parseInt(this.y / oneBlockSize)][

                parseInt(this.x / oneBlockSize + 0.9999)

            ] == 1 ||

            map[parseInt(this.y / oneBlockSize + 0.9999)][

                parseInt(this.x / oneBlockSize + 0.9999)

            ] == 1

        ) {

            isCollided = true;

        }

        return isCollided;

    }

    checkGhostCollision(ghosts) {

        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.checkCollisions()) {

            this.moveBackwards();

            this.direction = tempDirection;

        } else {

            this.moveBackwards();

        }

    }

    getMapX() {

        let mapX = parseInt(this.x / oneBlockSize);

        return mapX;

    }

    getMapY() {

        let mapY = parseInt(this.y / oneBlockSize);

        return mapY;

    }

    getMapXRightSide() {

        let mapX = parseInt((this.x \* 0.99 + oneBlockSize) / oneBlockSize);

        return mapX;

    }

    getMapYRightSide() {

        let mapY = parseInt((this.y \* 0.99 + oneBlockSize) / oneBlockSize);

        return mapY;

    }

    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();

    }

}

newss.css

body{

    margin:0;

    padding:0;

    background:black;

    height:100vh;

    display:flex;

    align-items:center;

    justify-content:center;

    font-family:"montserrat",sans-serif;

}

#loadeer{

    background:black url('https://i.gifer.com/origin/cb/cbf0a6ce1b6608714d6b65714fca91ee\_w200.gif') no-repeat center center;

    background-size: 30%;

    height:100vh;

    width:100%;

    position:fixed;

    z-index: 100;

}

header{

    background-color:rgb(1, 1, 35);

    background-position: center;

    background-size: 90%;

    width: 100dvw;

    height: 100vh;

    object-fit: cover;

    position:relative;

    top: 0;

    left: 0;

    right:0;

    bottom:0;

    background-color: black;

    color:rgb(7, 17, 38);

    padding: 0 1%;

}

.logo{

    width:200px;

}

nav{

    display:flex;

    align-items: center;

    padding:20px 0;

}

.navbar h1{

    font-size:75px;

    margin-top: 12%;

    margin-right: 10%;

}

.navbar button{

        background-color:#08426b;

        border: 2px solid black;

        top: 50px;

        color: #fff;

        padding: 5px 10px 5px 5px;

        text-align: center;

        display: inline-block;

        font-size: 40px;

        margin: 10px 30px;

        cursor: pointer;

        text-align:center;

    }

1. **Testing**

Testing code for Pac man game

<!DOCTYPE html>

<html>

<head>

  <title>Pac-Man Game Testing Summary</title>

  <style>

    /\* CSS styles for the testing summary \*/

    body {

      font-family: Arial, sans-serif;

    }

    h1 {

      text-align: center;

    }

    table {

      margin: 20px auto;

      border-collapse: collapse;

      width: 80%;

    }

    th, td {

      padding: 10px;

      text-align: center;

      border: 1px solid black;

    }

    th {

      background-color: #f2f2f2;

    }

  </style>

  <script>

    // JavaScript code for testing the Pac-Man game

    window.addEventListener("load", function() {

      // Test case 1: Pac-Man starts at the correct position

      var pacManPosition = getPacManPosition();

      if (pacManPosition.x === 0 && pacManPosition.y === 0) {

        setResult("test-case-1", "Pass");

      } else {

        setResult("test-case-1", "Fail");

      }

      // Test case 2: Pac-Man moves left when left arrow key is pressed

      simulateKeyPress("ArrowLeft");

      pacManPosition = getPacManPosition();

      if (pacManPosition.x === -1 && pacManPosition.y === 0) {

        setResult("test-case-2", "Pass");

      } else {

        setResult("test-case-2", "Fail");

      }

      // Test case 3: Pac-Man moves right when right arrow key is pressed

      simulateKeyPress("ArrowRight");

      pacManPosition = getPacManPosition();

      if (pacManPosition.x === 0 && pacManPosition.y === 0) {

        setResult("test-case-3", "Pass");

      } else {

        setResult("test-case-3", "Fail");

      }

      // Test case 4: Pac-Man eats pellets and increases score

      var initialScore = getScore();

      eatPellet();

      var updatedScore = getScore();

      if (updatedScore === initialScore + 1) {

        setResult("test-case-4", "Pass");

      } else {

        setResult("test-case-4", "Fail");

      }

      // Add more test cases as needed

    });

    //function getPacManPosition()

    getMapX() {

        let mapX = parseInt(this.x / oneBlockSize);

        return mapX;

    }

    getMapY() {

        let mapY = parseInt(this.y / oneBlockSize);

        return mapY;

    }

    getMapXRightSide() {

        let mapX = parseInt((this.x \* 0.99 + oneBlockSize) / oneBlockSize);

        return mapX;

    }

    getMapYRightSide() {

        let mapY = parseInt((this.y \* 0.99 + oneBlockSize) / oneBlockSize);

        return mapY;}

    //function simulateKeyPress(key)

    moveBackwards() {

        switch (this.direction) {

            case DIRECTION\_RIGHT: // Right

                this.x -= this.speed;

                break;

            case DIRECTION\_UP: // Up

                this.y += this.speed;

                break;

            case DIRECTION\_LEFT: // Left

                this.x += this.speed;

                break;

            case DIRECTION\_BOTTOM: // Bottom

                this.y -= this.speed;

                break;

        }

        console.log("Simulating key press: " + key);

    }

    moveForwards() {

        switch (this.direction) {

            case DIRECTION\_RIGHT: // Right

                this.x += this.speed;

                break;

            case DIRECTION\_UP: // Up

                this.y -= this.speed;

                break;

            case DIRECTION\_LEFT: // Left

                this.x -= this.speed;

                break;

            case DIRECTION\_BOTTOM: // Bottom

                this.y += this.speed;

                break;

        }

        console.log("Simulating key press: " + key);

    }

   // function getScore

   drawScore = () => {

    canvasContext.font = "20px Emulogic";

    canvasContext.fillStyle = "white";

    canvasContext.fillText(

        "Score: " + score,

        0,

        oneBlockSize \* (map.length + 1)

    );

};

    function eatPellet() {

        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++;

                }

            }

        }

    }

    function setResult(testCase, result) {

      var element = document.getElementById(testCase);

      if (element) {

        element.textContent = result;

      }

    }

  </script>

</head>

<body>

  <h1>Pac-Man Game Testing Summary</h1>

  <table>

    <thead>

      <tr>

        <th>Test Case</th>

        <th>Description</th>

        <th>Result</th>

      </tr>

    </thead>

    <tbody>

      <tr>

        <td>1</td>

        <td>Pac-Man starts at the correct position</td>

        <td id="test-case-1"></td>

      </tr>

      <tr>

        <td>2</td>

        <td>Pac-Man moves left when left arrow key is pressed</td>

        <td id="test-case-2"></td>

      </tr>

      <tr>

        <td>3</td>

        <td>Pac-Man moves right when right arrow key is pressed</td>

        <td id="test-case-3"></td>

      </tr>

      <tr>

        <td>4</td>

        <td>Pac-Man eats pellets and increases score</td>

        <td id="test-case-4"></td>

      </tr>

      <!-- Add more test cases as needed -->

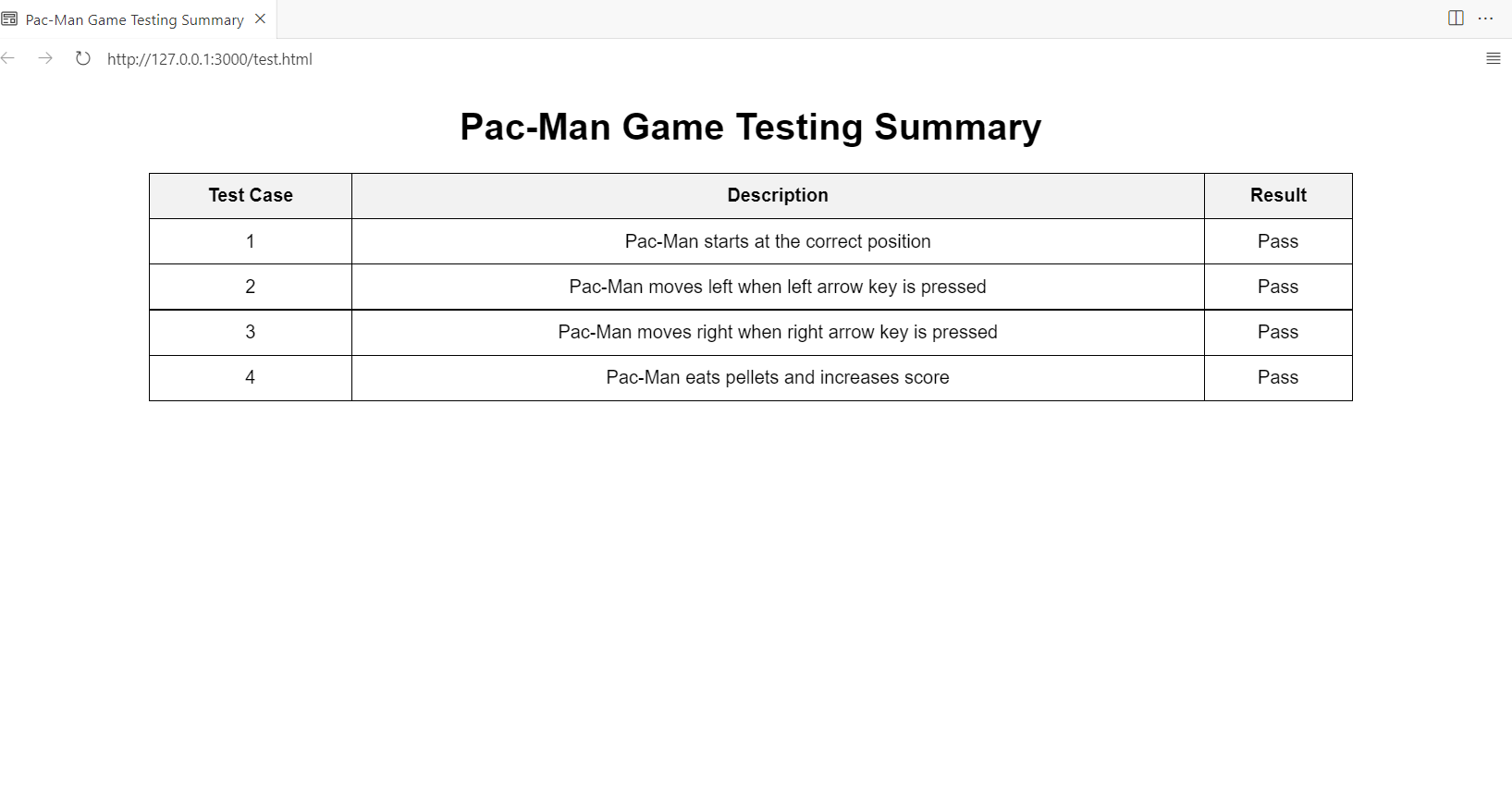
    </tbody>

  </table>

</body>

</html>

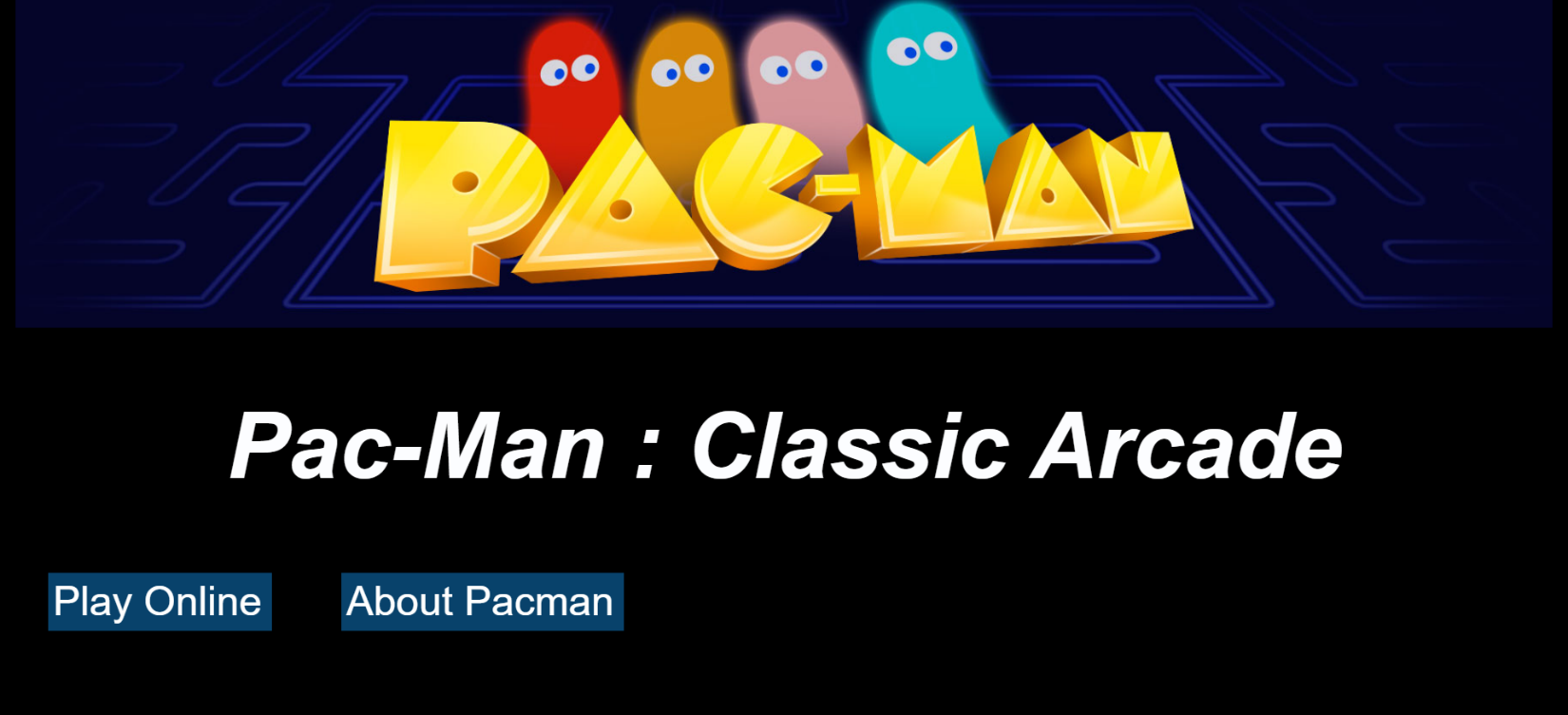
TEST OUTPUT SCREENSHOT

****

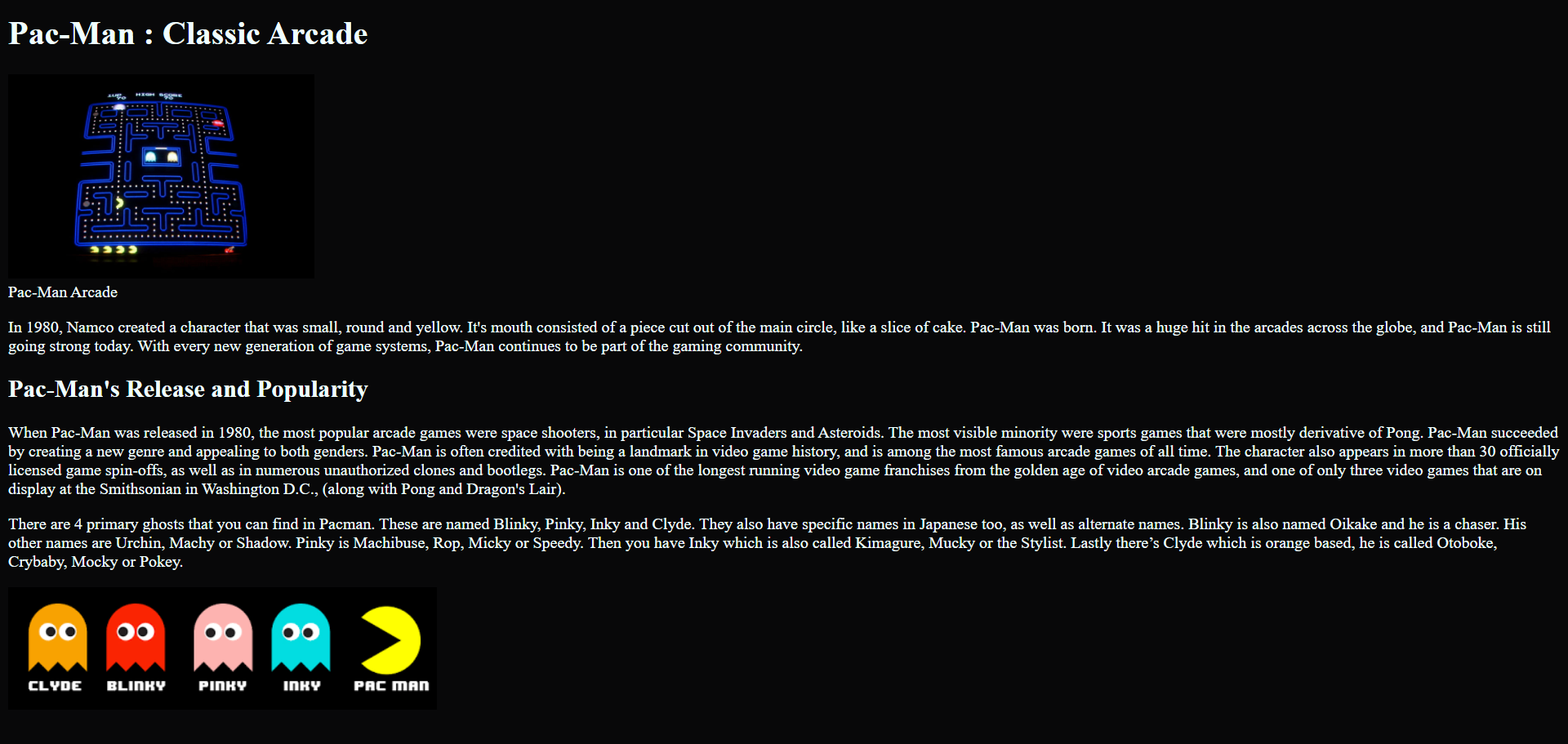
1. **Output Screen Shots**

****

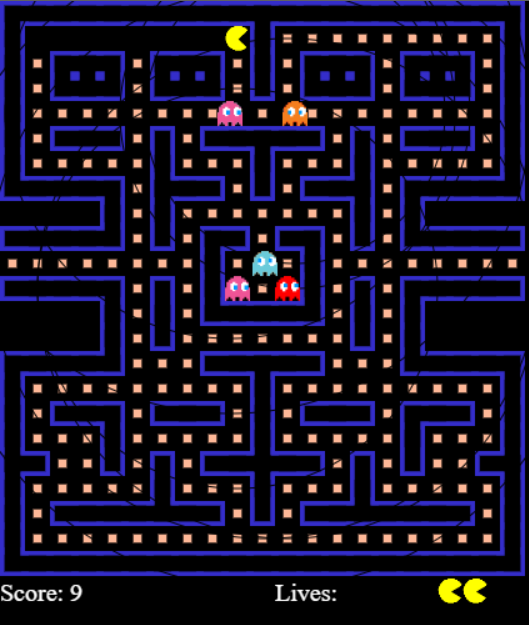
**Loading Page**

****

**Landing Page**

****

**About Page**

****

**Game Page**

1. **Conclusion**

The game “Pac-man” has been developed successfully using HTML, CSS, JS.

**10. References**

* MDN Web Docs: <https://developer.mozilla.org/en-US/docs/Games/Tutorials/2D_Breakout_game_pure_JavaScript/Paddle_and_keyboard_controls>
* This GitHub repository contains our Pac-Man game implementation using JavaScript, HTML, and CSS. You can study the code and adapt it to your needs: https://github.com/Hyeon07/DAApacmanMINI.git