~~~ ABSTRACT ~~~

Maze Game project is developed using HTML5, CSS3, and JavaScript. Here the player must escape from maze. Movements are done using Keyboard controls (W, A, S, D). Each level has different mazes.

About the game play, the player must start the game by clicking on the 'Play' button. Then, the player must move from one place to another through the empty path. The player can move as many times as he/she wants to, but he/she should not touch the walls. Movements are keyboard controlled using W, A, S, D Keys.

For the development of this simple web-based gaming project, no image is used, it is developed using JavaScript to bring the final output.All the gaming function is set from JavaScript whereas HTML and CSS are set for the layouts and other minor functions.

CHAPTER - 1

INTRODUCTION

1.1 Introduction to Web

Web consists of billions of clients and server connected through wires and wireless networks. The web clients make requests to web server. The web server receives the request, finds the resources and returns the response to the client. When a server answers a request, it usually sends some type of content to the client. The client uses web browser to send request to the server. The server often sends response to the browser with a set of instructions written in Hypertext Markup Language (HTML). All browsers know how to display HTML page to the client.

1.2 Hypertext Markup Language (HTML)

Hypertext Markup Language (HTML) is the standard markup language for creating web pages and web applications. HTML describes the structure of Web pages using markup. HTML elements are the building blocks of HTML pages. HTML elements are represented by tags. HTML tags label pieces of content such as "heading", "paragraph", "table", and so on. Browsers do not display the HTML tags but use them to render the content of the page.

Web browsers receive HTML documents from a web server or from local storage and render the documents into multimedia web pages. HTML describes the structure of a web page semantically and originally includes the cues for the appearance of the document.

1.3 Cascading Style Sheet (CSS)

Cascading Style Sheets (CSS) is a stylesheet language used to describe the presentation of a document written in HTML or XML. CSS describes how elements should be rendered on screen, on paper, in speech, or on other media. CSS is one of the core languages of the open Web and is standardized across Web browsers according to the W3C specification. Like HTML, CSS is not really a programming language. It is not a markup language either, it is a style sheet language. This means that it lets you apply styles selectively to elements in HTML documents.

CSS is designed to enable the separation of presentation and content, including layout, colors, fonts, etc. This separation of presentation can improve content accessibility, provide more flexibility and control in the specification of presentation characteristics, enable multiple web pages to share formatting by specifying the relevant CSS in a separate .css file, and reduce complexity and repetition in the structural content.

1.4 JavaScript

JavaScript is a prototype-based scripting language that is dynamic, weakly typed, general purpose programming language and has first-class functions. It is a multi-paradigm language, supporting object-oriented, imperative, and functional programming styles. JavaScript is primarily used in the form of client-side JavaScript, implemented as part of a Web browser in order to provide dynamic websites.

Alongside HTML and CSS, JavaScript is one of the core technologies of the World Wide Web. JavaScript enables interactive web pages and is an essential part of the web applications. Majority of the websites use JavaScript and major web browsers have a dedicated JavaScript engine to execute it.

As a multi-paradigm language, JavaScript supports event-driven, functional and imperative (including object-oriented and prototype-based) programming styles. It has APIs for working with texts, arrays, dates, regular expressions, and the DOM, but the language itself does not include any I/O, such as networking, storage, or graphics facilities. It relies upon the host environment in which it is embedded to provide these features.

CHAPTER - 2

SYSTEM ANALYSIS

2.1 Introduction to mini project

Maze Game project is developed using HTML5, CSS3 and JavaScript. Here the player must escape from the maze. The player must start the game by clicking on the 'Play' button. Then the player must move from one place to another through the empty path. The player can move as many times as he/she wants to, but he/she should not touch the walls. Movements are keyboard controlled using W, A, S, D keys.

The main objective of this mini project is to design and develop a dynamic webpage with user playable maze game.

2.2 Resource Requirements

Software Requirements:

- Front end tools: HTML5, CSS3, JavaScript
- Text editor (Atom / Visual Studio Code / Sublime / Other)
- Windows Operating system 7 and above
- Browser (Chrome / Mozilla / Edge / Opera / Other)

Hardware Requirements:

- Processor x86 or x64
- RAM: 512 MB (Minimum), 1 GB (Recommended)
- Hard disk: up to 4 GB of available space

CHAPTER – 3

SYSTEM DESIGN

3.1 System Perspective

Design is the first step in the development phase for any techniques and principles for the purpose of defining a device, a process or system in enough detail to permit its physical realization. Once the software requirements have been analyzed and specified the software design involves three technical activities – design, coding, implementation and testing that are required to build and verify the software.

The design activities are of main importance in this phase, because in this activity, decisions ultimately affecting the success of the software implementation and its ease of maintenance are made. These decisions have the final bearing upon reliability and maintainability of the system. Design is the only way to accurately translate the customer's requirements into finished software or a system.

3.2 Use case Diagram

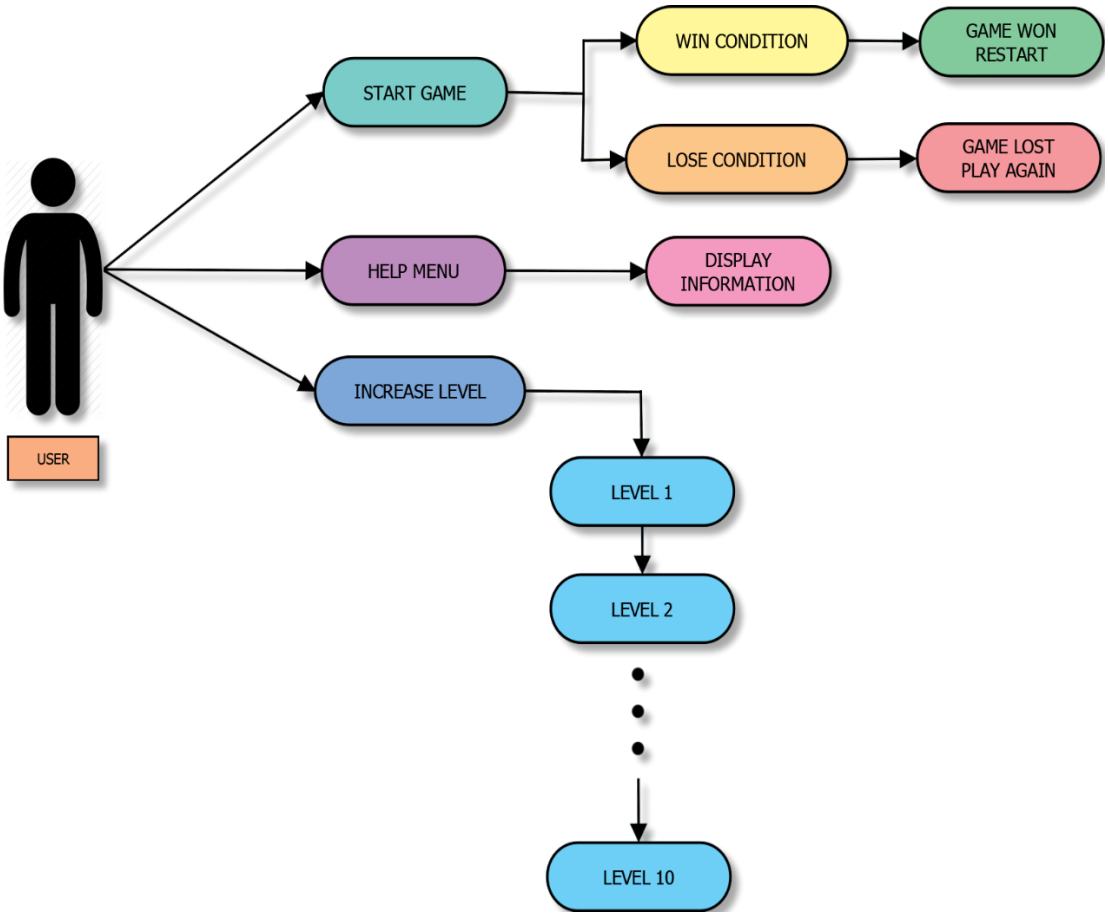

Figure 3.2 Use Case Diagram

3.3 Sequence Diagram

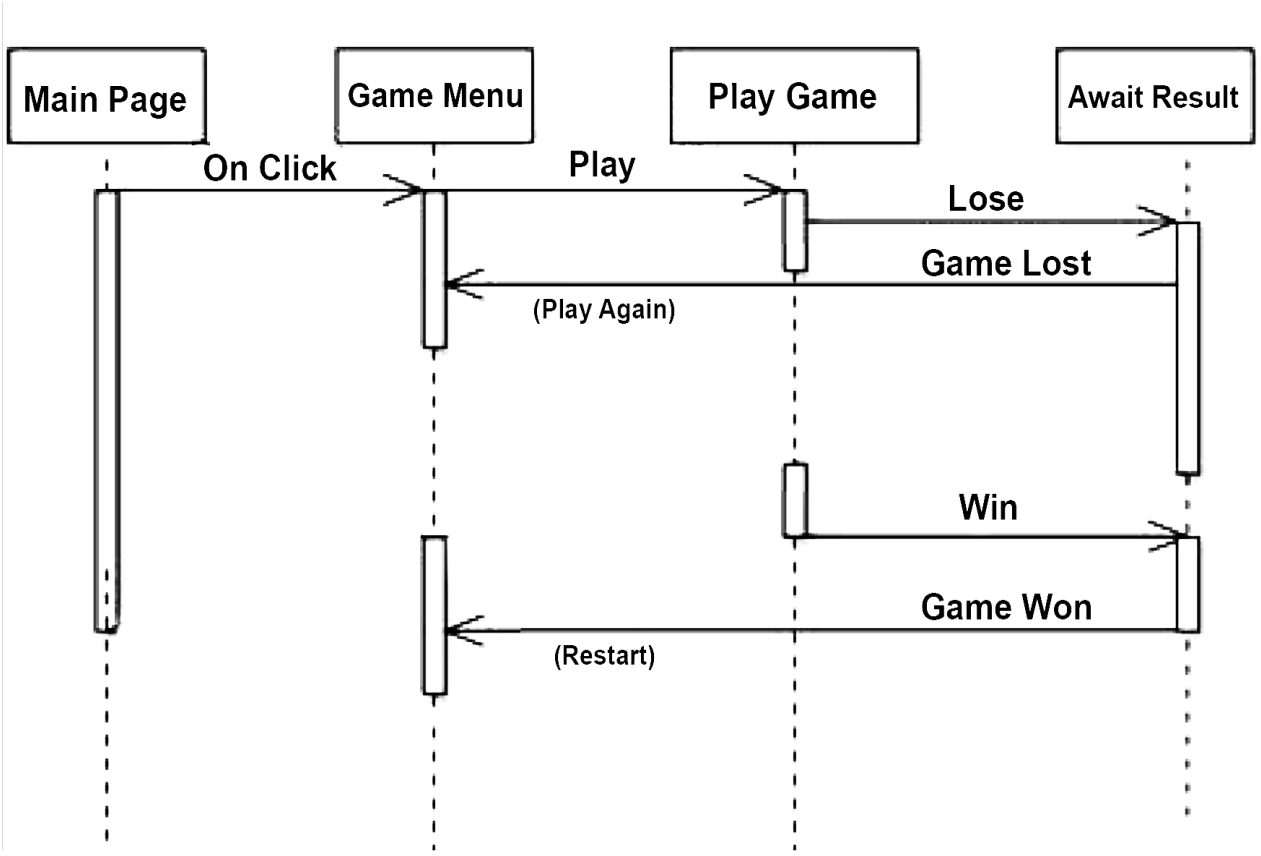

Figure 3.3 Sequence Diagram

CHAPTER - 4

IMPLEMENTATION

index.html

```
<!DOCTYPE html>
<html lang="en">

<head>
   <meta charset="UTF-8">
   <meta content="width=device-width, initial-scale=1.0" name="viewport">
   <script defer src="maze.js" type="text/javascript"></script>
   <link rel="stylesheet" href="maze.css">
   <title>Maze Game</title>
</head>

<body>

   <div id='cover'>
      <table>
/*maze*/
      </table>
   </div>
/*play*/
   <button id='one' type="button" name="Click to Play" onClick='loadPage();'>Play<br><br
>(Default Level 1)</button>
/*info*/
   <button id='two' type="button" name="controls" onClick='info();'>Help</button>
/*increase level*/
   <button id='three' type="button" name="select next level" onClick='nl();'>Increase level(s)
</button>

</body>

</html>
```

maze.css

```
html {
    box-sizing: border-box;
}
*,
* ::before,
*::after {
    box-sizing: border-box;
}
body {
    width: auto;
    height: auto;
    background-image:url("maze.png");
    background-repeat: no-repeat;
    background-size: cover;
    display: flex;
    flex-direction: column;
    align-items: center;
    justify-content: center;
    font-family: sans-serif;
}
h1 {
    font-weight: bold;
    color: whitesmoke;
    height: 100px;
    font-size: 40pt;
    float: center;
}
/*table styles*/
table {
    margin-left: 750px;
    margin-top: 200px;
    margin-right: auto;
```

```
        text-align: center;
        width: auto;
        height: auto;
        border-spacing: 0px;
        border-collapse: initial;
        position: absolute;
        text-align: center;
    }
    div#cover {
        position: none;
    }
/*for the moving object*/
    #player {
        position: absolute;
        width: 20px;
        height: 20px;
        display: block;
        background-color:  #35f835;
        color:  #35f835;
        z-index: 10000;
        border: 5px double  #35f835;
        border-radius: 25px;
    }
    button1 {
        margin-top: 159px;
        background-color: rgba(66, 64, 64, 0.815);
        color: whitesmoke;
        font-size: 25pt;
        text-align: center;
        padding-top: 50px;
        border: 5px solid  #35f835;
        width: 300px;
        height: 150px;
        border:3px solid #35f835;
        border-radius: 50px;
```

```
}
button1:hover{
    background-color: #085208ab;
}
button {
    margin-top: 91px;
    background-color: rgba(66, 64, 64, 0.815);
    color: whitesmoke;
    font-size: 25pt;
    border: 5px solid  #35f835;
    width: 300px;
    height: 150px;
    border:3px solid #35f835;
    border-radius: 50px;
}
button:hover{
    background-color: #085208ab;
}
/*sections which is for end game scenario*/
section {
    margin-top: 200px;
    width: 600px;
    height: 600px;
    border: 3px solid  #35f835;
    border-radius: 10px;
    background-image: url("Game won.png");
    z-index: 0;
    display: inline-flex;
    flex-direction: column;
    justify-content: center;
    align-items: center;
}
section1 {
    margin-top: 200px;
    width: 600px;
```

```
    height: 600px;
    border: 3px solid #35f835;
    border-radius: 10px;
    background-image: url("Game over.png");
    z-index: 0;
    display: inline-flex;
    flex-direction: column;
    justify-content: center;
    align-items: center;
}
p {
    color: whitesmoke;
    font-size: 15pt;
    height: 50px;
    font-weight: bolder;
}
p1 {
    color: whitesmoke;
    font-size: 15pt;
    height: 50px;
    font-weight: bolder;
    margin-top: 10px;
}
tr {
    border: none;
}
/*table element styles(maze)*/
td {
    width: 25px;
    height: 35px;
    margin: 0px;
    border: none;
    padding: 0px;
    text-align: center;
}
```

```
#win {
    background-color: green;
    color: green;
}
.wall {
    color: rgb(42, 41, 41) ;
    background-color:  rgb(42, 41, 41);
}
.freespace {
    color: rgba(255, 255, 255, 0);
    background-color: rgba(255, 255, 255, 0.486);
}
#start {
    color: rgb(148, 9, 9);
    background-color: rgb(148, 9, 9);
}
#level{
    color: green;
    font-weight: bolder;
    text-align: center;
    font-size: 20px;
    margin-left: 50px;
    background-color: rgb(22, 21, 21);
}
#one{
    margin-top: 250px;
}
#two{
    margin-top: 30px;
}
#three{
    margin-top: 30px;
}
```

maze.js

```
let currentLevel = maze1;
let nextLevel = 0;
let levels = [maze1, maze2, maze3, maze4, maze5, maze6, maze7, maze8, maze9, maze10];
let body = document.querySelector('body');
let divTable = document.getElementById('cover');
let tableEl = document.querySelector('table');
//if button clicked it loads this function for info on keys
let info = () => {
   let b1 = document.querySelector('#one')
   b1.textContent = 'WASD to move and press me to play';
};
let nl = () => {
   let b1 = document.querySelector('#one');
   nextLevel = (nextLevel + 1) % 10;
   b1.textContent = 'Play Level ' + (nextLevel + 1);
   if (nextLevel > 4)
      alert("Zoom out accordingly to play Level " + (nextLevel + 1));
};
//if button clicked it loads the game
let loadPage = () => {
   let getRideOfMenu = () => {
      let b1 = document.querySelector('#one');
      let b2 = document.querySelector('#two');
      let b3 = document.querySelector('#three');
      b1.style.display = 'none';
      b2.style.display = 'none';
      b3.style.display = 'none';
      body.style.flexDirection = 'row';
      body.style.justifyContent = 'flex-start';
      body.style.alignItems = 'flex-start';
   };
getRideOfMenu();

let won = () => {
      //needed variables for win condition
      let winP = document.createElement('section');
      let h1 = document.createElement('h1');
      let button = document.createElement('button1');
      clearTable(tableEl);
      //styles for win condition
      mover.style.display = 'none';
      h1.textContent = 'GAME WON!!';
      button.textContent = 'Play Again?';
      button.setAttribute('onclick', 'window.location.reload();');
      button.setAttribute('type', 'button');
      //adding win para to body and other child elements
      body.appendChild(winP);
      winP.appendChild(h1);
      winP.appendChild(button);
```

```
    body.style.justifyContent = 'center';
  };
  let lose = () => {
    //needed variables for end condition
    let looseP = document.createElement('section1');
    let para = document.createElement('p');
    let h1 = document.createElement('h1');
    let button = document.createElement('button');
    clearTable(tableEl);
    //styles for end condition
    mover.style.display = 'none';
    h1.textContent = 'OOPS!! GAME OVER';
    para.textContent = 'Press the below button to restart.';
    button.textContent = 'Restart?';
    button.setAttribute('onclick', 'window.location.reload();');
    button.setAttribute('type', 'button');
    //adding end para to body and other child elements
    body.appendChild(looseP);
    looseP.appendChild(h1);
    looseP.appendChild(para);
    looseP.appendChild(button);
    body.style.justifyContent = 'center';
  };

const clearTable = (tableEl) => {
    while (tableEl.firstChild) {
      tableEl.removeChild(tableEl.firstChild);
    }
  };
  let mover = document.createElement('div');

  function resetMover() {
    mover.style.left = '785px';
    mover.style.top = '250px';
    mover.setAttribute('id', 'player');
    mover.textContent = '`';
  }
  const drawMaze = () => {//creating a function to draw maze
    //defining basic layout
    divTable.appendChild(mover);
    resetMover();
    for (let i = 0; i < levels[nextLevel].length; i++) { //loop for tr's
      let rowEl = document.createElement('tr');
      tableEl.appendChild(rowEl);
      for (let x = 0; x < levels[nextLevel][i].length; x++) { //loop for td's
        let tdEl = document.createElement('td');
        rowEl.appendChild(tdEl)
        tdEl.innerHTML = levels[nextLevel][i].charAt(x);
        //conditionals below if the char is a specific character
        //then run the code below
        switch (levels[nextLevel][i].charAt(x)) {
          case '#':
```

```
                tdEl.setAttribute('class', 'wall');
                break;
            case '.':
                tdEl.setAttribute('class', 'freespace');
                break;
            case '_':
                tdEl.setAttribute('id', 'start');
                break;
            case '!':
                tdEl.setAttribute('id', 'win');
                break;
            default:
                tdEl.setAttribute('id','level');
                break;
        }
      }
    };
  };


// print the maze and table on the page
  drawMaze();
  window.addEventListener('keydown', event => {
    //mover based on which key is press then the action will occur
    switch (event.key) {
      case 'w':
        mover.style.top = parseInt(mover.style.top) - 10 + 'px';
        //the mover moves on left and top axis then parseInt gives a interger
        break;
      case 'a':
        mover.style.left = parseInt(mover.style.left) - 10 + 'px';
        break;
      case 's':
        mover.style.top = parseInt(mover.style.top) + 10 + 'px';

        break;
      case 'd':
        mover.style.left = parseInt(mover.style.left) + 10 + 'px';
        break;
    }
    //defining variables to use for win and lose conditions
    let pos = mover.getBoundingClientRect();
    let wins = document.getElementById('win').getBoundingClientRect();
    let walls = document.querySelectorAll('.wall');
    for (let wall of walls) {
      let wowWalls = wall.getBoundingClientRect();
      // checks for wall and player collision
      if (pos.x < wowWalls.x + wowWalls.width && pos.x + pos.width > wowWalls.x &&
pos.y < wowWalls.y + wowWalls.height && pos.y + pos.height > wowWalls.y) {
        lose();
      } else if (pos.x < wins.x + wins.width && pos.x + pos.width > wins.x && pos.y < wi
ns.y + wins.height && pos.y + pos.height > wins.y) {
```

```
            won();
            break;
        }
        if (pos.x == 0) {
            lose();
        }
    }
  }); //end of eventListener
}; //end of on click function
```

Sample maze layouts

```
let maze4 = [
    `################################`,
    `#_....###.........#........###`,
    `####..###..###..####..####..###`,
    `####.......###..#.....####..###`,
    `###############..####..#.......#`,
    `##..............####..#..###..#`,
    `##..##################..#..###..#`,
    `##..##........######..#....#..#`,
    `##..##..####..#.......######..#`,
    `##......####..#..###.......#..#`,
    `#############..#..############..#`,
    `##............#..#.......##...#`,
    `##..############..#..###..##..##`,
    `##..................###..#...!#`,
    `################################`,
    'LEVEL 4                        '
];
let maze8 = [
    `##############################################`,
    `#_.........................#................#`,
    `##..####################...#...##########...#`,
    `##..#.......#.......#...#...#.......#...#`,
    `##..#...#...#...#...#...#...#...#..##...#`,
    `##..#...#.......#...........#...#...#...#`,
    `##..##########...###############...#...#...#`,
    `##.........#...............#...#...#...#`,
    `########################..#...#...#...#...#`,
    `##.......................#...#...#.......#`,
    `##..####################...#...#...##########`,
    `##..#...............#...#...#...........#`,
    `##..###############...#...#...#####...#...#`,
    `##......#.......#.......#.......#...#...#`,
    `##..#...#...#...#...##########...#####...#`,
    `##..#.......#...#...........#.......#...#`,
    `###############...####################...#...#`,
    `##..............#.......#.......#...#...#`,
    `##..###############...#...#...#...#####...#`,
    `##..................#.......#..........!#`,
    `##############################################`,
    'LEVEL 8                                      '
];
```

CHAPTER - 6

Conclusion and Future Enhancement

We were able to create a 2D maze game that also required thinking and problem-solving skills. The application that was built on Web platform that can produce various proper maze layouts consistently, throught this process we have learned many interesting techniques involved in creating a web based 2d game and this project gave us an insight to what game creation process was like.

The game is developed using HTML, CSS and JavaScript. The player must escape the maze without colliding with the walls with the help of W, A, S, D keys.

In the future from the gaming side, this application can be improved by adding extra challenges such as: a timer may be introduced to keep an account on the time taken by the player to escape the maze. This information can be stored in a database and later can be fetched to display the scorecard.

Bonus items can be placed randomly all over the maze.

This game can be developed in a 3D environment to make users feel the game realistic and natural. And a ranking system can also be introduced by appending a database to keep a track and to display the ranks of multiple players based on their time interval i.e. time taken to escape the maze. And an option to solve the maze i.e. to show the path to be traversed in order to escape the maze can also be implemented maze can also be implemented.

REFERENCES

[1] Zak Ruvalcha Anne Boehm “Murach’s HTML5 and CSS3”

[2] https://www.w3schools.com/

[3] https://www.tutorialspoint.com/css/index.htm

[4] https://javascript.info/

[5] https://www.javatpoint.com/javascript-tutorial

[6] https://www.the-art-of-web