*Marko Stojkovski 191277*

**JavaScript & WebAssembly Integration**

**Research & Analysis**

**Tic-Tac-Toe Game**

**Table of Contents**

1. Introduction
2. Project Structure
3. Technologies Used
4. Functionalities

* Game Board
* Player Moves
* Score Tracking
* Winning Conditions
* Modal Pop-Up

1. Code Overview

* HTML
* CSS
* JavaScript
* Rust & WebAssembly

1. Conclusion

**Introduction**

This project is a web-based implementation of the classic Tic Tac Toe game. The game features a 3x3 grid where two players take turns marking the spaces with "X" and "O". The first player to get three of their marks in a row (vertically, horizontally, or diagonally) wins the game. The project is built using modern web technologies including HTML, CSS, JavaScript, Rust and WebAssembly (Wasm).

**Project Structure**

The project structure is organized as follows:

**tic-tac-toe/**

**├ pkg/**

**├** package.json

**├** tic\_tac\_toe\_bg.wasm

**├** tic\_tac\_toe\_bg.wasm.d.ts

**├** tic\_tac\_toe.d.ts

**├** tic\_tac\_toe.js

**├ src/**

**├** lib.rs

**├ static/**

**├** index.html

**├** style.css

**├** main.js

**├ Cargo.toml**

* pkg/: Directory generated by the wasm-pack tool, containing the WebAssembly package.
* package.json - The configuration file for the JavaScript.
* tic\_tac\_toe\_bg.wasm, tic\_tac\_toe\_bg.wasm.d.ts, tic\_tac\_toe.d.ts, tic\_tac\_toe.js - Files generated by WebAssembly.
* src/: Contains the Rust source files.
* lib.rs: The main Rust file where the game logic is implemented.
* static/: Contains the static files for the web application.
* index.html: The main HTML file for the web application.
* styles.css: The CSS file for styling the web application.
* main.js: The JavaScript file that interacts with the WebAssembly module.
* Cargo.toml: The configuration file for Rust.

**Technologies Used**

* **HTML**: The structure of the web page.
* **CSS**: Styling the web page to make it visually appealing.
* **JavaScript**: Handling the user interactions and dynamically updating the DOM.
* **Rust**: The core game logic being implemented for performance and safety.
* **WebAssembly**: Compiling the Rust code to WebAssembly to run in the browser.

**Functionalities**

**Game Board**

The game board is a 3x3 grid where players can place their marks. Each cell in the grid is interactive and listens for click events.

**Player Moves**

Players take turns to place their marks ("X" for Player 1 and "O" for Player 2) on the game board. The game logic ensures that players cannot place a mark on an already occupied cell.

**Score Tracking**

The game keeps track of each player's score. The scores are displayed on either side of the game board, with Player 1's score on the left and Player 2's score on the right.

**Winning Conditions**

The game checks for winning conditions after each move. A player wins if they manage to place three of their marks in a row, either horizontally, vertically, or diagonally. If all cells are filled without a winner, the game is considered a draw.

**Modal Pop-Up**

When a player wins or the game ends in a draw, a modal pop-up is displayed with a message indicating the result. The modal can be closed, which resets the game board for a new round.

**Code Overview**

**HTML**: provides the structure of the web page, including the game board, the score display, and the modal pop-up.

<!DOCTYPE *html*>

<html *lang*="en">

<head>

    <meta *charset*="UTF-8">

    <meta *name*="viewport" *content*="width=device-width, initial-scale=1.0">

    <title>Tic Tac Toe</title>

    <link *rel*="stylesheet" *href*="style.css">

</head>

<body>

    <div *id*="game">

        <div *class*="scoreboard-players">

            <p>Player 1 (X) Score: <span *id*="player1-score">0</span></p>

        </div>

        <div *class*="board-container">

            <div *class*="board">

                <div *class*="cell" *data-index*="0"></div>

                <div *class*="cell" *data-index*="1"></div>

                <div *class*="cell" *data-index*="2"></div>

                <div *class*="cell" *data-index*="3"></div>

                <div *class*="cell" *data-index*="4"></div>

                <div *class*="cell" *data-index*="5"></div>

                <div *class*="cell" *data-index*="6"></div>

                <div *class*="cell" *data-index*="7"></div>

                <div *class*="cell" *data-index*="8"></div>

            </div>

        </div>

        <div *class*="scoreboard-players">

            <p>Player 2 (O) Score: <span *id*="player2-score">0</span></p>

        </div>

    </div>

    <div *id*="myModal" *class*="modal">

        <div *class*="modal-content">

            <h2 *id*="modal-message"></h2>

            <button *id*="modal-close">Close</button>

        </div>

    </div>

    <script *type*="module" *src*="./main.js"></script>

</body>

</html>

**CSS**: provides the styles for the game, ensuring a visually appealing and responsive design.

body {

    display: flex;

    justify-content: center;

    align-items: center;

    height: 100vh;

    margin: 0;

    background-color: #323232;

}

#game {

    display: flex;

    align-items: center;

}

.board-container {

    display: flex;

    margin: 0 20px;

}

.board {

    display: grid;

    grid-template-columns: **repeat**(3, 100px);

    grid-template-rows: **repeat**(3, 100px);

    gap: 5px;

    background-color: white;

    box-shadow: 2px 2px 4px **rgba**(0, 0, 0, 0.9);

}

.cell {

    width: 100px;

    height: 100px;

    display: flex;

    align-items: center;

    justify-content: center;

    font-size: 2em;

    border: 1px solid #000;

}

.cell.x {

    color: red;

}

.cell.o {

    color: green;

}

.player1 {

    text-align: left;

}

.player2 {

    text-align: right;

}

.modal {

    display: none;

    position: fixed;

    z-index: 1;

    left: 0;

    top: 0;

    width: 100%;

    height: 100%;

    background-color: **rgba**(0,0,0,0.4);

    justify-content: center;

    align-items: center;

    transition: opacity 0.3s ease;

}

.modal-content {

    background-color: #fff;

    padding: 20px;

    border: 1px solid #888;

    text-align: center;

    border-bottom-left-radius: 10px;

    border-bottom-right-radius: 10px;

}

.modal h2 {

    margin: 0;

    padding: 0;

    font-size: 2em;

}

.modal button {

    margin-top: 20px;

    padding: 10px 20px;

    font-size: 1em;

    border: none;

    background-color: #007BFF;

    color: white;

    border-radius: 5px;

    cursor: pointer;

}

.modal button:hover {

    background-color: #0056b3;

}

.scoreboard-players {

    color: white;

}

**JavaScript**: handles user interactions, updates the DOM, and communicates with the WebAssembly module.

import init, { TicTacToe } from "../pkg/tic\_tac\_toe.js";

async function **run**() {

    await **init**();

    const game = new **TicTacToe**();

    const boardElement = document.**querySelector**(".board");

    const cells = document.**querySelectorAll**(".cell");

    const player1ScoreElement = document.**getElementById**("player1-score");

    const player2ScoreElement = document.**getElementById**("player2-score");

    const modal = document.**getElementById**("myModal");

    const modalMessage = document.**getElementById**("modal-message");

    const modalClose = document.**getElementById**("modal-close");

    function **render**() {

        const board = game.**get\_board**();

        cells.**forEach**((cell, index) => {

            cell.textContent = board[index] === 1 ? "X" : board[index] === 2 ? "O" : "";

            cell.classList.**remove**("x", "o");

            if (board[index] === 1) {

                cell.classList.**add**("x");

            } else if (board[index] === 2) {

                cell.classList.**add**("o");

            }

        });

        player1ScoreElement.textContent = game.**get\_player1\_score**();

        player2ScoreElement.textContent = game.**get\_player2\_score**();

    }

    function **showModal**(message) {

        modalMessage.textContent = message;

        modal.style.display = "block";

**setTimeout**(() => {

            modal.style.opacity = 1;

        }, 10);

    }

    function **hideModal**() {

        modal.style.opacity = 0;

**setTimeout**(() => {

            modal.style.display = "none";

        }, 300);

    }

    modalClose.**addEventListener**("click", () => {

**hideModal**();

        game.**reset\_board**();

**render**();

    });

    boardElement.**addEventListener**("click", (event) => {

        if (event.target.classList.**contains**("cell")) {

            const index = **parseInt**(event.target.**getAttribute**("data-index"), 10);

            if (game.**make\_move**(index)) {

**render**();

                const winner = game.**check\_winner**();

                if (winner !== 0) {

**setTimeout**(() => {

                        if (winner === 3) {

**showModal**("It's a draw!");

                        } else {

**showModal**(`Player ${winner} wins!`);

                        }

                    }, 10);

                }

            }

        }

    });

**render**();

}

**run**();

**Rust & WebAssembly**: The Rust code implements the core game logic and is compiled to WebAssembly to be used in the web application.

use **wasm\_bindgen**::**prelude**::\*;

#[**wasm\_bindgen**]

pub struct **TicTacToe** {

    board: [**u8**; 9],

    current\_player: **u8**,

    player1\_score: **u32**,

    player2\_score: **u32**,

}

#[**wasm\_bindgen**]

impl **TicTacToe** {

    #[**wasm\_bindgen**(constructor)]

    pub fn **new**() -> **TicTacToe** {

**TicTacToe** {

            board: [0; 9],

            current\_player: 1,

            player1\_score: 0,

            player2\_score: 0,

        }

    }

    pub fn **get\_board**(&self) -> **Vec**<**u8**> {

        self.board.**to\_vec**()

    }

    pub fn **get\_player1\_score**(&self) -> **u32** {

        self.player1\_score

    }

    pub fn **get\_player2\_score**(&self) -> **u32** {

        self.player2\_score

    }

    pub fn make\_move(&mut self, index: **usize**) -> **bool** {

        if self.board[index] == 0 {

            self.board[index] = self.current\_player;

            self.current\_player = 3 - self.current\_player;

            true

        } else {

            false

        }

    }

    pub fn check\_winner(&mut self) -> **u8** {

        const WINNING\_COMBINATIONS: [[**usize**; 3]; 8] = [

            [0, 1, 2], [3, 4, 5], [6, 7, 8], *// rows*

            [0, 3, 6], [1, 4, 7], [2, 5, 8], *// columns*

            [0, 4, 8], [2, 4, 6],            *// diagonals*

        ];

        for combo in WINNING\_COMBINATIONS.**iter**() {

            let [a, b, c] = \*combo;

            if self.board[a] != 0 && self.board[a] == self.board[b] && self.board[b] == self.board[c] {

                let winner = self.board[a];

                if winner == 1 {

                    self.player1\_score += 1;

                } else {

                    self.player2\_score += 1;

                }

                return winner;

            }

        }

        if self.board.**iter**().all(|&x| x != 0) {

            3

        } else {

            0

        }

    }

    pub fn reset\_board(&mut self) {

        self.board = [0; 9];

        self.current\_player = 1;

    }

}

**Conclusion**

This Tic Tac Toe project demonstrates how to build a simple, interactive web application using modern web technologies. The combination of HTML, CSS, and JavaScript provides a responsive and interactive user interface, while Rust and WebAssembly ensure efficient and safe game logic execution. This documentation covers the core functionalities, project structure, and the technologies used, providing a comprehensive guide for understanding and extending the project.