**מסמך SRS -אלגוריתמים ותזמון למערכות נבונות – מדמ"ח תשפ"ד**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ **המגישים:**

שי עיני 208635193

כפיר קורן 314627720

דניאל אזולאי 316085448

נבו אבישר 318305463  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**לינקים:**סרטון מצגת - <https://youtu.be/UTL9gkRGjnE?si=1NNCLVZjzRKNShEL>

סרטון משחק - <https://youtu.be/PNBPIbppokE?si=6Z77txcK5uxDvI08>

גיט - <https://github.com/nevoavisar97/submarine-game>

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**הבעיה:**

הפרויקט נדרש לייצר משחק אסטרטגי שבו שני שחקנים מתחרים בהזזת צוללות על גבי לוח משחק. כל שחקן צריך להגיע לנקודת היעד שלו במהירות האפשרית, תוך שהוא מתמודד עם צוללות מתחרות המונעות את התקדמותו. המשחק צריך לכלול מנגנון תור, ממשק משתמש דינמי, ותנאי ניצחון.

**הפתרון:**

* **ממשק משתמש:** נבנה באמצעות HTML ו-xr כדי ליצור לוח משחק רספונסיבי ודינמי המציג את הצוללות, השחקנים והיעדים.
* **לוגיקת משחק:** נכתבה ב-JavaScript כדי לטפל בתנועות השחקנים, מציאת מסלולים, ניהול תורים ותנאי ניצחון.
* **ספריות נוספות:** SweetAlert שימשה להצגת הודעות יפות ובכך שיפרה את חווית המשתמש.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**דרישות הפרויקט:**

* לוח משחק דינמי: לוח משחק בגודל 10x10 בו יופיעו הצוללות, השחקנים והיעדים.
* תנועת שחקנים: השחקנים יכולים להזיז את הצוללות במעלה, מטה, שמאלה וימינה בתורות.
* תנאי ניצחון: השחקן שמגיע לנקודת היעד שלו ראשון הוא המנצח.
* ממשק משתמש: הצגת הודעות התקדמות, ניצחון או הפסד.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**אלגוריתמים בהם השתמשנו**

**אלגוריתם A\***

* תיאור: אלגוריתם למציאת המסלול הקצר ביותר בין שתי נקודות על גבי גרף. האלגוריתם משתמש בפונקציית הערכה (Heuristic) כדי להעריך את המרחק מהנקודה הנוכחית לנקודת היעד.
* מימוש בפרויקט: האלגוריתם שימש למציאת המסלול הקצר ביותר של כל שחקן מנקודת ההתחלה לנקודת היעד שלו, תוך התחשבות במיקומי הצוללות על הלוח.

**אלגוריתם מותאם למקסום הצעד**

* האלגוריתם מחשב את המהלך האופטימלי של הצוללת בהתבסס על המסלול הקצר ביותר ובוחן את כל המהלכים האפשריים של הצוללת. בנוסף, מחשב את המרחק של השחקן והמחשב לנקודת היעד החדשה ומחזיר את המהלך שנותן את היתרון הגדול ביותר לשחקן הנוכחי.

game.js

import { Player } from './player.js';

import { aStar } from './utils.js';

import { Submarine } from './submarine.js';

//מחלקת משחק

export class Game {

    constructor(boardElement) {

        this.boardElement = boardElement;

        this.cells = [];

        this.players = [];

        this.grid = Array(10).fill().map(() => Array(10).fill(false));

        this.submarines = [];

        this.currentPlayerIndex = 0; // תור השחקן הנוכחי

        this.counterSwitches=0;

    }

    //בניית לוח המשחק

    buildBoard() {

        const board = document.getElementById('gameBoard');

        board.innerHTML = "";

        for (let y = 0; y < 10; y++) {

            for (let x = 0; x < 10; x++) {

                const cell = document.createElement('div');

                cell.classList.add('cell');

                cell.id = `${x},${y}`;

                // cell.textContent= cell.id;

                this.boardElement.appendChild(cell);

                this.cells.push(cell);

            }

        }

    }

    //מיקומי השחקנים והיעדים שלהם

    setPositions(isComputer) {

        this.cells.forEach(cell => cell.classList.remove('player', 'target', 'submarine'));

        let player1 = new Player('Player 1', { x: 1, y: 1 }, { x: 8, y: 8 },true);

        let player2 = new Player(isComputer ?'Player 2':'בן אדם', { x: 1, y: 8 }, { x: 8, y: 1 },isComputer);

        this.players = [player1, player2];

        document.getElementById(`${player1.getPosition().x},${player1.getPosition().y}`).classList.add('player');

        document.getElementById(`${player1.getPosition().x},${player1.getPosition().y}`).textContent="P1";

        document.getElementById(`${player2.getPosition().x},${player2.getPosition().y}`).classList.add('player');

        player2.getIsComputer() ? document.getElementById(`${player2.getPosition().x},${player2.getPosition().y}`).textContent="P2": document.getElementById(`${player2.getPosition().x},${player2.getPosition().y}`).textContent="P2 Human" ;

        document.getElementById(`${player1.getTarget().x},${player1.getTarget().y}`).classList.add('target');

        document.getElementById(`${player1.getTarget().x},${player1.getTarget().y}`).textContent="P1 Target";

        document.getElementById(`${player2.getTarget().x},${player2.getTarget().y}`).classList.add('target');

        document.getElementById(`${player2.getTarget().x},${player2.getTarget().y}`).textContent="P2 Target";

    }

    //מיקום הצוללות על הלוח

    placeSubmarinesRandomly() {

        let isValid = false;

        while (!isValid) {

            this.cells.forEach(cell => cell.classList.remove('submarine'));

            this.grid = Array(10).fill().map(() => Array(10).fill(false));

            this.submarines = this.getRandomSubmarines();

            if (this.isValidPlacement(this.submarines)) {

                this.submarines.forEach(submarine => {

                    submarine.getPosition().forEach(pos => {

                        document.getElementById(`${pos.x},${pos.y}`).classList.add('submarine');

                        this.grid[pos.x][pos.y] = true;

                    });

                });

                isValid = true;

            }

        }

    }

    //פונקצית עזר למיקום הצוללות

    getRandomSubmarines() {

        const submarines = [ ];

        const occupiedPositions = new Set();

        while (submarines.length < 4) {

            const x = Math.floor(Math.random() \* 4) + 3; // בטורים 3-6

            const y = Math.floor(Math.random() \* 9);

            const pos1 = `${x},${y}`;

            const pos2 = `${x},${y + 1}`;

            if (!occupiedPositions.has(pos1) && !occupiedPositions.has(pos2) && y + 1 < 10) {

                submarines.push(new Submarine([{ x, y }, { x, y: y + 1 }], 2));

                occupiedPositions.add(pos1);

                occupiedPositions.add(pos2);

            }

        }

        return submarines;

    }

    //ולידציה להנחת הצוללות

    isValidPlacement(submarines) {

        this.grid = Array(10).fill().map(() => Array(10).fill(false));

        const occupiedPositions = new Set();

        for (let submarine of submarines) {

            for (let pos of submarine.getPosition()) {

                if (occupiedPositions.has(`${pos.x},${pos.y}`)) {

                    return false;

                }

                this.grid[pos.x][pos.y] = true;

                occupiedPositions.add(`${pos.x},${pos.y}`);

            }

        }

        const player1Path = aStar(this.players[0].getPosition(), this.players[0].getTarget(), this.grid);

        const player2Path = aStar(this.players[1].getPosition(), this.players[1].getTarget(), this.grid);

        if (!player1Path || !player2Path) {

            return false;

        }

        if (player1Path.length === player2Path.length) {

            player1Path.forEach(node => {

                document.getElementById(`${node.x},${node.y}`).classList.add('player1-path');

            });

            player2Path.forEach(node => {

                document.getElementById(`${node.x},${node.y}`).classList.add('player2-path');

            });

        }

        return player1Path.length === player2Path.length;

    }

    //מהלך תור משחק בין 2 שחקני מחשב

    switchTurnComputerVSComputer(){

        if(this.counterSwitches==2){

            this.winGame();

            document.getElementById('startButtonComputerVsComputer').disabled = false;

            document.getElementById('startButtonHumanVsComputer').disabled = false;

            return;

        }

        this.counterSwitches++;

        this.currentPlayerIndex = (this.currentPlayerIndex + 1) % 2; // מעבר לשחקן הבא

        alert(`תור השחקן: ${this.getCurrentPlayer().name}`);

        setTimeout(() => {

            this.moveBestSubmarine();

        }, 1500);

        setTimeout(() => {

            this.switchTurnComputerVSComputer();

        }, 3000);

    }

    //מהלך תור משחק בין מחשב לבין בנאדם

    switchTurn() {

        this.currentPlayerIndex = (this.currentPlayerIndex + 1) % 2; // מעבר לשחקן הבא

        alert(`תור השחקן: ${this.getCurrentPlayer().name}`);

        console.log(this.getCurrentPlayer())

        if(this.getCurrentPlayer().isComputer){

            setTimeout(() => {

            this.moveBestSubmarine();

            // this.switchTurn();

        }, 1500);

        setTimeout(() => {

            this.winGame();

            document.getElementById('startButtonComputerVsComputer').disabled = false;

            document.getElementById('startButtonHumanVsComputer').disabled = false;

             // this.switchTurn();

        }, 1500);

        }

        else{

                ////ממתינים לשחקן

             let p=document.getElementById('in');

             p.textContent="בחר צוללת והזז באמצעות מקשי החצים"

        }

    }

    //פונקצית תחילת משחק

    startGame(isComputer) {

        let p=document.getElementById('in');

        p.textContent="" ;

        this.counterSwitches=0;

        this.buildBoard();

        this.setPositions(isComputer);

        this.placeSubmarinesRandomly();

        this.boardElement.focus();

    }

    //החזרת השחקן שתורו

    getCurrentPlayer() {

        return this.players[this.currentPlayerIndex];

    }

    //הזזת הצוללת למיקום האופטימלי

    moveBestSubmarine() {

        // שלב 1: קביעת השחקן הנוכחי

        const currentPlayer = this.getCurrentPlayer();

        // שלב 2: מציאת המהלך האופטימלי להזזת הצוללת בעזרת Alpha-beta pruning

        const bestMove = this.findBestMove(currentPlayer);

        if (bestMove) {

            // שלב 3: ביצוע המהלך

            const { submarine, newPosition } = bestMove;

            // console.log(bestMove)

            const oldPositions = submarine.getPosition();

            // עדכון מיקום הצוללת

            submarine.setPosition(newPosition);

            // שלב 4: עדכון לוח המשחק

            this.updateBoard(oldPositions, newPosition);

            this.updatePaths();

            // סימון שהצוללת הוזזה

            submarine.hasMoved = true;

        }

    }

    //פונקצית עזר למציאת התזוזה האופטימלית

    findBestMove(player) {

        let bestMove = null;

        let bestScore = -Infinity;

        for (let submarine of this.submarines) {

            if (!submarine.hasMoved) {

                 console.log("old")

                 console.log(submarine.getPosition())

                const possibleMoves = this.getPossibleMoves(submarine);

                for (let move of possibleMoves) {

                    // חישוב ציון למהלך הנוכחי

                    // console.log("Move")

                    // console.log(move)

                    const score = this.evaluateMove(player, submarine, move,submarine.getPosition());

                    console.log(`Submarine ID: ${submarine.id}, Move: ${JSON.stringify(move)}, Score: ${score}`);

                    if (score > bestScore) {

                        bestScore = score;

                        bestMove = { submarine, newPosition: move };

                    }

                }

            }

        }

        return bestMove;

    }

    // פונקציה עזר להחזרת מהלכים אפשריים לצוללת

    getPossibleMoves(submarine) {

    const possibleMoves = [];

    const directions = [

        { x: 1, y: 0 },

        { x: -1, y: 0 },

        { x: 0, y: 1 },

        { x: 0, y: -1 }

    ];

    for (let dir of directions) {

        const newPositions = submarine.getPosition().map(pos => ({

            x: pos.x + dir.x,

            y: pos.y + dir.y

        }));

        if (this.isValidMove(newPositions, submarine.getPosition())) {

            possibleMoves.push(newPositions);

        }

    }

    return possibleMoves;

}

    //בדיקת ולידציה להזזת הצוללות

isValidMove(newPositions, currentPositions) {

    return newPositions.every(pos =>

        pos.x >= 0 && pos.x < 10 && pos.y >= 0 && pos.y < 10 && (!this.grid[pos.x][pos.y] || currentPositions.some(currPos => currPos.x === pos.x && currPos.y === pos.y))

    );

}

    //מתן ניקוד לתזוזת צוללת

evaluateMove(player, submarine, newPosition,old) {

    const opponent = this.players[(this.currentPlayerIndex + 1) % 2];

    // שמירה של המיקום הנוכחי של הצוללת לשחזור מאוחר יותר

    const originalPosition = old;

    old.forEach(pos => {

        this.grid[pos.x][pos.y] = false;

    });

    newPosition.forEach(pos => {

        this.grid[pos.x][pos.y] = true;

    });

    // עדכון מיקום הצוללת למיקום החדש

    submarine.setPosition(newPosition);

    // חישוב מסלול חדש לשחקן הנוכחי וליריב בעזרת A\*

    const playerPath = aStar(player.getPosition(), player.getTarget(), this.grid);

    const opponentPath = aStar(opponent.getPosition(), opponent.getTarget(), this.grid);

      newPosition.forEach(pos => {

        this.grid[pos.x][pos.y] = false;

    });

    originalPosition.forEach(pos => {

        this.grid[pos.x][pos.y] = true;

    });

    // שחזור מיקום הצוללת למיקום המקורי

    submarine.setPosition(originalPosition);

    // אם לא נמצא מסלול לשחקן הנוכחי או ליריב, ציון נמוך מאוד

    if (!playerPath || !opponentPath) {

        return -Infinity;

    }

    // חישוב הציון: הפרש המסלולים

    const score = (opponentPath.length - playerPath.length);

    return score;

}

    //עדכון הלוח לאחר שינוי מיקומי הצוללות

updateBoard(oldPositions, newPositions) {

    // הסרת הצוללת מהמיקום הישן בלוח

    oldPositions.forEach(pos => {

        this.grid[pos.x][pos.y] = false;

        document.getElementById(`${pos.x},${pos.y}`).classList.remove('submarine');

    });

    // הוספת הצוללת למיקום החדש בלוח

    newPositions.forEach(pos => {

        this.grid[pos.x][pos.y] = true;

        document.getElementById(`${pos.x},${pos.y}`).classList.add('submarine');

    });

}

    //עדכון מסלולים הקצרים לכל שחקן לאחר הזזת הצוללות

updatePaths() {

    // ניקוי מסלולים קודמים

    this.cells.forEach(cell => {

        cell.classList.remove('player1-path');

        cell.classList.remove('player2-path');

    });

    // חישוב מסלול חדש לשחקן 1

    const player1Path = aStar(this.players[0].getPosition(), this.players[0].getTarget(), this.grid);

    if (player1Path) {

        player1Path.forEach(node => {

            document.getElementById(`${node.x},${node.y}`).classList.add('player1-path');

        });

    }

    // חישוב מסלול חדש לשחקן 2

    const player2Path = aStar(this.players[1].getPosition(), this.players[1].getTarget(), this.grid);

    if (player2Path) {

        player2Path.forEach(node => {

            document.getElementById(`${node.x},${node.y}`).classList.add('player2-path');

        });

    }

}

    //פונקציה לבדיקת מנצח המשחק

    winGame(){

        let p = document.getElementById('in');

        const player1Path = aStar(this.players[0].getPosition(), this.players[0].getTarget(), this.grid);

        const player2Path = aStar(this.players[1].getPosition(), this.players[1].getTarget(), this.grid);

        if (player1Path.length < player2Path.length) {

            player1Path.forEach(node => {

                document.getElementById(`${node.x},${node.y}`).classList.add('win');

            });

            p.innerHTML = `Player One Wins!<br>Player One Length: ${player1Path.length}<br>Player Two Length: ${player2Path.length}`;

        } else if (player1Path.length === player2Path.length) {

            p.innerHTML = "Tie";

        } else {

            player2Path.forEach(node => {

                document.getElementById(`${node.x},${node.y}`).classList.add('win');

            });

            p.innerHTML = `Player Two Wins!<br>Player Two Length: ${player2Path.length}<br>Player One Length: ${player1Path.length}`;

        }

    }

}

**Player.js**

export class Player {

    constructor(name, position, target,isComputer) {

        this.name = name;

        this.position = position;

        this.target = target;

        this.isComputer = isComputer;

    }

    setPosition(position) {

        this.position = position;

    }

    getPosition() {

        return this.position;

    }

    getTarget() {

        return this.target;

    }

    getIsComputer(){

        return this.isComputer;

    }

}

**Script.js**

import { Game } from './game.js';

document.addEventListener('DOMContentLoaded', () => {

    // הגדרת משתנים

    let selectedSubmarine = null;

    let selectedSubmarinePositions = null;

    // יצירת HTML בצורה דינמית

    const body = document.body;

    const board = document.getElementById('gameBoard');

    const startButtonComputerVsComputer = document.createElement('button');

    startButtonComputerVsComputer.id = 'startButtonComputerVsComputer';

    startButtonComputerVsComputer.textContent = 'Computer vs Computer';

    const startButtonHumanVsComputer = document.createElement('button');

    startButtonHumanVsComputer.id = 'startButtonHumanVsComputer';

    startButtonHumanVsComputer.textContent = 'Human vs Computer';

    const divButtons = document.createElement('div');

    divButtons.appendChild(startButtonComputerVsComputer);

    divButtons.appendChild(startButtonHumanVsComputer);

    divButtons.id='buttonsDiv';

    const divInstructions = document.createElement('div');

    const p = document.createElement('p');

    p.id='in'

    divInstructions.appendChild(p);

    body.insertBefore(divButtons, board);

    body.insertBefore(divInstructions, board);

    // יצירת אובייקט המשחק

    const game = new Game(board);

    // התחלת משחק בין שני שחקני מחשב

    startButtonComputerVsComputer.addEventListener('click', () => {

        game.startGame(true);

        document.getElementById('startButtonComputerVsComputer').disabled = true;

        document.getElementById('startButtonHumanVsComputer').disabled = true;

        game.switchTurnComputerVSComputer();

    });

    // התחלת משחק בין שחקן מחשב לבין בנאדם

    startButtonHumanVsComputer.addEventListener('click', () => {

        game.startGame(false);

        document.getElementById('startButtonComputerVsComputer').disabled = true;

        document.getElementById('startButtonHumanVsComputer').disabled = true;

        game.switchTurn();

    });

    // בחירת צוללת באמצעות קליק בעכבר

    board.addEventListener('click', (event) => {

        const currentPlayer = game.getCurrentPlayer();

        // אם זה תור המחשב, לא עושים כלום

        if (currentPlayer.isComputer) {

            return;

        }

        const target = event.target;

        const [x, y] = target.id.split(',').map(Number);

        // אם צוללת נבחרה, לא נזיז אותה

        if (selectedSubmarine) {

            selectedSubmarinePositions.forEach(pos => {

                document.getElementById(`${pos.x},${pos.y}`).classList.remove('selected-submarine');

            });

            selectedSubmarine = null;

            selectedSubmarinePositions = null;

        }

        // נבדוק אם יש צוללת בלחיצה ונבחר אותה

        const submarine = game.submarines.find(sub => sub.getPosition().some(pos => pos.x === x && pos.y === y) && !sub.hasMoved);

        if (submarine) {

            // אם יש צוללת אחרת שנבחרה, נבטל את הבחירה בה

            if (selectedSubmarine) {

                selectedSubmarinePositions.forEach(pos => {

                    document.getElementById(`${pos.x},${pos.y}`).classList.remove('selected-submarine');

                });

            }

            // נבחר את הצוללת החדשה

            selectedSubmarine = submarine;

            selectedSubmarinePositions = submarine.getPosition();

            selectedSubmarinePositions.forEach(pos => {

                document.getElementById(`${pos.x},${pos.y}`).classList.add('selected-submarine');

            });

        }

    });

    // האזנה לאירועי מקלדת להזזת הצוללת

    document.addEventListener('keydown', (event) => {

        const currentPlayer = game.getCurrentPlayer();

        // אם זה תור המחשב, לא עושים כלום

        if (currentPlayer.isComputer || !selectedSubmarine) {

            return;

        }

        let direction = null;

        switch (event.key) {

            case 'ArrowUp':

                direction = { x: 0, y: -1 };

                break;

            case 'ArrowDown':

                direction = { x: 0, y: 1 };

                break;

            case 'ArrowLeft':

                direction = { x: -1, y: 0 };

                break;

            case 'ArrowRight':

                direction = { x: 1, y: 0 };

                break;

            default:

                return; // אם זה לא אחד מהחיצים, לא עושים כלום

        }

        const newPositions = selectedSubmarinePositions.map(pos => ({

            x: pos.x + direction.x,

            y: pos.y + direction.y

        }));

        if (game.isValidMove(newPositions, selectedSubmarinePositions)) {

            selectedSubmarinePositions.forEach(pos => {

                document.getElementById(`${pos.x},${pos.y}`).classList.remove('selected-submarine');

            });

            game.updateBoard(selectedSubmarinePositions, newPositions);

            selectedSubmarine.setPosition(newPositions);

            selectedSubmarine.hasMoved = true;

            selectedSubmarine = null;

            selectedSubmarinePositions = null;

            p.textContent = "";

            game.switchTurn();

        } else {

            alert("צעד לא חוקי");

        }

    });

});

**Submarine.js**

//מחלקת צוללת

export class Submarine {

    static counter = 0;

    constructor(position, size) {

        this.id=Submarine.counter++; // הגדרת ID שונה לכל צוללת

        this.position = position;

        this.size = size;

        this.hasMoved = false; // תכונה לציון אם הצוללת הוזזה בעבר

    }

    getPosition() {

        return this.position;

    }

    setPosition(position) {

        this.position = position;

    }

    getSize() {

        return this.size;

    }

    static resetCounter() {

        Submarine.counter = 0;

    }

}

**Utils.js**

//פונקצית הערכה לחישוב מרחק לשימוש ב A\*

// פונקציה לחישוב מרחק אוקלידי

export function euclideanDistance(x1, y1, x2, y2) {

    return Math.sqrt(Math.pow(x1 - x2, 2) + Math.pow(y1 - y2, 2));

}

// אלגוריתם A\*

export function aStar(start, goal, grid) {

    const openSet = [start];

    const cameFrom = new Map();

    const gScore = new Map();

    gScore.set(`${start.x},${start.y}`, 0);

    const fScore = new Map();

    fScore.set(`${start.x},${start.y}`, euclideanDistance(start.x, start.y, goal.x, goal.y));

    while (openSet.length > 0) {

        openSet.sort((a, b) => fScore.get(`${a.x},${a.y}`) - fScore.get(`${b.x},${b.y}`));

        const current = openSet.shift();

        if (current.x === goal.x && current.y === goal.y) {

            let path = [];

            let temp = current;

            while (cameFrom.has(`${temp.x},${temp.y}`)) {

                path.push(temp);

                temp = cameFrom.get(`${temp.x},${temp.y}`);

            }

            return path.reverse();

        }

        const neighbors = getNeighbors(current, grid);

        for (const neighbor of neighbors) {

            const tentativeGScore = gScore.get(`${current.x},${current.y}`) + 1;

            if (!gScore.has(`${neighbor.x},${neighbor.y}`) || tentativeGScore < gScore.get(`${neighbor.x},${neighbor.y}`)) {

                cameFrom.set(`${neighbor.x},${neighbor.y}`, current);

                gScore.set(`${neighbor.x},${neighbor.y}`, tentativeGScore);

                fScore.set(`${neighbor.x},${neighbor.y}`, tentativeGScore + euclideanDistance(neighbor.x, neighbor.y, goal.x, goal.y));

                if (!openSet.some(n => n.x === neighbor.x && n.y === neighbor.y)) {

                    openSet.push(neighbor);

                }

            }

        }

    }

    return null; // No path found

}

// פונקצית עזר למציאת שכנים ש A\* משתמש בה

function getNeighbors(node, grid) {

    const neighbors = [];

    const directions = [

        { x: 1, y: 0 },

        { x: -1, y: 0 },

        { x: 0, y: 1 },

        { x: 0, y: -1 },

        { x: 1, y: 1 },

        { x: 1, y: -1 },

        { x: -1, y: 1 },

        { x: -1, y: -1 }

    ];

    for (const dir of directions) {

        const newX = node.x + dir.x;

        const newY = node.y + dir.y;

        if (newX >= 0 && newX < 10 && newY >= 0 && newY < 10 && !grid[newX][newY]) {

            neighbors.push({ x: newX, y: newY });

        }

    }

    return neighbors;

}

**Index.html**

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

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

    <title>משחק הצוללן</title>

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

    <div class="board" id="gameBoard"></div>

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

</head>

<body>

</body>

</html>

**Stylysheet**

body {

    display: flex;

    flex-direction: column;

    justify-content: center;

    align-items: center;

    height: 100vh;

    margin: 0;

    background: linear-gradient(to right, #1e3c72, #2a5298);

    color: #313131;

    font-family: 'Heebo', sans-serif;

    text-align: center;

    padding-bottom: 150px;

}

.board {

    display: grid;

    grid-template-columns: repeat(10, 50px);

    grid-template-rows: repeat(10, 50px);

    gap: 2px;

    margin: 20px auto;

    padding: 10px;

    background-color: #ffffff;

    border-radius: 10px;

    box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);

}

.cell {

    width: 50px;

    height: 50px;

    background-color: #87CEEB;

    border: 1px solid #4682B4;

    border-radius: 4px;

    display: flex;

    justify-content: center;

    align-items: center;

    transition: background-color 0.3s;

}

.cell:hover {

    background-color: #00BFFF;

}

.player {

    background-color: #FFD700;

}

.target {

    background-color: #FF4500;

}

.submarine {

    background-color: #808080;

}

#startButtonComputerVsComputer, #startButtonHumanVsComputer {

    margin: 10px;

    padding: 12px 24px;

    font-size: 18px;

    cursor: pointer;

    background-color: #ff5722;

    color: white;

    border: none;

    border-radius: 25px;

    transition: background-color 0.3s, transform 0.3s;

}

#startButtonComputerVsComputer:hover, #startButtonHumanVsComputer:hover {

    background-color: #e64a19;

    transform: scale(1.05);

}

#startButtonHumanVsComputer {

    background-color: #4CAF50;

}

#startButtonHumanVsComputer:hover {

    background-color: #45A049;

}

.player1-path {

    background-color: rgba(255, 0, 0, 0.5); /\* Transparent Red \*/

}

.player2-path {

    background-color: rgba(0, 0, 255, 0.5); /\* Transparent Blue \*/

}

.win {

    background-color: rgba(21, 255, 0, 0.5);

}

.selected-submarine {

    border: 3px solid rgb(0, 0, 0);

    box-shadow: 0 0 10px rgba(0, 0, 0, 0.5);

}

#in {

    color:#ffffff;

    font-size: 20px;

    font-weight: bold;

    margin-top: 20px;

}

#buttonsDiv {

    display: absoulute;

    justify-content: center;

    align-items: center;

    margin-top: 20px;

    padding-top: 20px;

    gap: 20px;

}