

Міністерство освіти і науки України
Національний технічний університет України «Київський політехнічний
інститут імені Ігоря Сікорського"
Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

Звіт

з лабораторної роботи № 5 з дисципліни
«Проектування алгоритмів»

„Пошук в умовах протидії, ігри з повною інформацією”

Виконав(ла)

ІП-02 Василенко Павло Олександрович
(шифр, прізвище, ім'я, по батькові)

Перевірив

Вєчерковська А.С.
(прізвище, ім'я, по батькові)

Київ 2021

ЗМІСТ

1	МЕТА ЛАБОРАТОРНОЇ РОБОТИ	3
2	ЗАВДАННЯ	4
3	ВИКОНАННЯ.....	5
3.1	ПРОГРАМНА РЕАЛІЗАЦІЯ АЛГОРИТМУ	6
3.1.1	<i>Вихідний код.....</i>	6
3.1.2	<i>Приклади роботи</i>	6
3.3	ТЕСТУВАННЯ АЛГОРИТМУ	6
	ВИСНОВОК	7
	КРИТЕРІЇ ОЦІНЮВАННЯ	8

1 МЕТА ЛАБОРАТОРНОЇ РОБОТИ

Мета роботи - вивчити основні підходи до формалізації алгоритмів знаходження рішень задач в умовах протидії. Ознайомитися з підходами до програмування алгоритмів штучного інтелекту в іграх з повною інформацією.

2 ЗАВДАННЯ

Згідно варіанту (таблиця 2.1) реалізувати візуальний ігровий додаток для гри користувача з комп'ютерним опонентом. Для реалізації стратегії гри комп'ютерного опонента використовувати алгоритм альфа-бета-відсікань.

Реалізувати три рівні складності (легкий, середній, складний) + 1 балл.

Зробити узагальнений висновок з лабораторної роботи.

Таблиця 2.1 – Варіанти

№	Варіант
2	Нейтріко http://www.iggamecenter.com/info/ru/neutreeko.html

3 ВИКОНАННЯ

3.1 Програмна реалізація алгоритму

3.1.1 Вихідний код

Index.html

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8" />
    <link rel="icon" href="%PUBLIC_URL%/favicon.ico" />
    <meta name="viewport" content="width=device-width, initial-scale=1" />
    <meta name="theme-color" content="#000000" />
    <meta
      name="description"
      content="Web site created using create-react-app"
    />
    <link rel="apple-touch-icon" href="%PUBLIC_URL%/logo192.png" />
    <!--
      manifest.json provides metadata used when your web app is installed on a
      user's mobile device or desktop. See
https://developers.google.com/web/fundamentals/web-app-manifest/
    -->
    <link rel="manifest" href="%PUBLIC_URL%/manifest.json" />
    <!--
      Notice the use of %PUBLIC_URL% in the tags above.
      It will be replaced with the URL of the `public` folder during the build.
      Only files inside the `public` folder can be referenced from the HTML.

      Unlike "/favicon.ico" or "favicon.ico", "%PUBLIC_URL%/favicon.ico" will
      work correctly both with client-side routing and a non-root public URL.
      Learn how to configure a non-root public URL by running `npm run build`.
    -->
    <title>React App</title>
  </head>
  <body>
    <noscript>You need to enable JavaScript to run this app.</noscript>
    <div id="root"></div>
    <!--
      This HTML file is a template.
      If you open it directly in the browser, you will see an empty page.

      You can add webfonts, meta tags, or analytics to this file.
      The build step will place the bundled scripts into the <body> tag.

      To begin the development, run `npm start` or `yarn start`.
      To create a production bundle, use `npm run build` or `yarn build`.
    -->
```

```
</body>
</html>
```

Index.tsx

```
import React from "react";
import ReactDOM from "react-dom";
import "./index.css";
import App from "./App";
import { Provider } from "react-redux";
import { setupStore } from "./store/store";

const store = setupStore();

ReactDOM.render(
  <Provider store={store}>
    <App />
  </Provider>,
  document.getElementById("root")
);
```

Index.css

```
body {
  margin: 0;
  font-family: -apple-system, BlinkMacSystemFont, "Segoe UI", "Roboto", "Oxygen",
    "Ubuntu", "Cantarell", "Fira Sans", "Droid Sans", "Helvetica Neue",
    sans-serif;
  -webkit-font-smoothing: antialiased;
  -moz-osx-font-smoothing: grayscale;
  background-color: #444;
}

code {
  font-family: source-code-pro, Menlo, Monaco, Consolas, "Courier New",
    monospace;
}

* {
  box-sizing: border-box;
}
```

App.tsx

```
import React from "react";
import "./App.css";
import Board from "./components/board/Board";
import { useAppSelector, useAppDispatch } from "./hooks/redux";
import { BoardSlice } from "./store/reducers/boardSlice";
import { GameStateSlice, TDifficulty } from "./store/reducers/gameStateSlice";
import { MoveSlice } from "./store/reducers/moveSlice";
```

```

import { TurnSlice } from "../store/reducers/turnSlice";

function App() {
  const board = useAppSelector((store) => store.board);
  const { state: gameState, difficulty } = useAppSelector(
    (store) => store.game
  );
  const dispatch = useAppDispatch();
  const restartGame = () => {
    dispatch(BoardSlice.actions.restart());
    dispatch(GameStateSlice.actions.restart());
    dispatch(MoveSlice.actions.restart());
    dispatch(TurnSlice.actions.restart());
  };
  const concede = () => {
    dispatch(GameStateSlice.actions.setResultOfGame("whiteWin"));
  };
  return (
    <>
      <button
        onClick={() =>
          alert(`The players have three pieces each. They are placed as shown in
the figure.
Movement: A piece slides orthogonally or diagonally until stopped by
an occupied square or the border of the board. Black always moves first.

Objective: To get three in a row, orthogonally or diagonally. The row must be
connected.`)
        }
      >
        Rules
      </button>
      <div className="app">
        <h1>Neutreeko</h1>
        {gameState === "start" ? (
          <>
            <p>You can choose the difficulty</p>
            <select
              onChange={(e) =>
                dispatch(
                  GameStateSlice.actions.setDifficulty(
                    +e.target.value as TDifficulty
                  )
                )
              }
              value={difficulty}
            >
              <option value={1}>Easy</option>
              <option value={2}>Medium</option>
              <option value={3}>Hard</option>
            </select>
          </>
        ) : null}
      </div>
    </>
  );
}

```

```

        <option value={4}>Impossible</option>
      </select>
    </>
  ) : null}
  {gameState === "going" ? (
    <>
      {" "}
      <p>The game is hard..</p>
      <button onClick={concede}>Concede</button>
    </>
  ) : null}
  {gameState === "blackWin" ? (
    <>
      <p>Black win the game!</p>
      <button onClick={restartGame}>Restart</button>
    </>
  ) : null}
  {gameState === "whiteWin" ? (
    <>
      <p>white win the game!</p>
      <button onClick={restartGame}>Restart</button>
    </>
  ) : null}
  {gameState === "draw" ? (
    <>
      <p>Draw!</p>
      <button onClick={restartGame}>Restart</button>
    </>
  ) : null}

  <Board board={board} />
</div>
</>
);
}

export default App;

```

app.css

```

.app {
  width: 100%;
  height: 100vh;
  display: flex;
  justify-content: center;
  align-items: center;
  flex-direction: column;
}

```



```
h1 {
  color: white;
}
p {
  color: white;
}
```

Board.ts

```
export type TBoard = number[][];

export interface ICell {
  row: number;
  col: number;
}

export const SCORE = {
  USER_WIN: -1,
  DRAW: 0,
  AI_WIN: 1,
};
```

Colors-enums.ts

```
export enum Colors {
  white = "white",
  black = "black",
}
```

Store.ts

```
import { combineReducers, configureStore } from "@reduxjs/toolkit";
import boardReducer from "../reducers/boardSlice";
import turnReducer from "../reducers/turnSlice";
import moveReducer from "../reducers/moveSlice";
import GameStateReducer from "../reducers/gameStateSlice";

const rootReducer = combineReducers({
  turn: turnReducer,
  board: boardReducer,
  move: moveReducer,
  game: GameStateReducer,
});

export const setupStore = () => {
  return configureStore({
```

```

    reducer: rootReducer,
  });
};

export type RootState = ReturnType<typeof rootReducer>;
export type AppStore = ReturnType<typeof setupStore>;
export type AppDispatch = AppStore["dispatch"];

```

boardSlice.ts

```

import { createSlice, PayloadAction } from "@reduxjs/toolkit";
import { ICell, TBoard } from "../../types/board";

const initialState: TBoard = [
  [0, 1, 0, 1, 0],
  [0, 0, -1, 0, 0],
  [0, 0, 0, 0, 0],
  [0, 0, 1, 0, 0],
  [0, -1, 0, -1, 0],
];

export const BoardSlice = createSlice({
  name: "board",
  initialState,
  reducers: {
    moveFrom: (state, action: PayloadAction<ICell>) => {
      const { row, col } = action.payload;
      state[row][col] = 0;
    },
    setWhite: (state, action: PayloadAction<ICell>) => {
      const { row, col } = action.payload;
      state[row][col] = 1;
    },
    setBlack: (state, action: PayloadAction<ICell>) => {
      const { row, col } = action.payload;
      state[row][col] = -1;
    },
    restart: (state) => {
      state[0] = [0, 1, 0, 1, 0];
      state[1] = [0, 0, -1, 0, 0];
      state[2] = [0, 0, 0, 0, 0];
      state[3] = [0, 0, 1, 0, 0];
      state[4] = [0, -1, 0, -1, 0];
    },
  },
});

export default BoardSlice.reducer;

```

gameStateSlice.ts

```
import { createSlice, PayloadAction } from "@reduxjs/toolkit";

type TGameState = "start" | "going" | "blackWin" | "whiteWin" | "draw";
export type TDifficulty = 1 | 2 | 3 | 4;
interface IGameState {
  state: TGameState;
  difficulty: TDifficulty;
}

const initialState: IGameState = {
  state: "start",
  difficulty: 1,
};

export const GameStateSlice = createSlice({
  name: "gameState",
  initialState,
  reducers: {
    setResultOfGame: (state, action: PayloadAction<TGameState>) => {
      state.state = action.payload;
    },
    setDifficulty: (state, action: PayloadAction<TDifficulty>) => {
      state.difficulty = action.payload;
    },
    restart: (state) => {
      state.state = "start";
    },
  },
});

export default GameStateSlice.reducer;
```

moveSlice.ts

```
import { createSlice, PayloadAction } from "@reduxjs/toolkit";
import { ICell } from "../../types/board";
interface IMoveState {
  chosenFigure: ICell | null;
  possibleMoves: ICell[];
}

const initialState: IMoveState = {
  chosenFigure: null,
  possibleMoves: [],
};
```

```

export const MoveSlice = createSlice({
  name: "move",
  initialState,
  reducers: {
    chooseFigure: (state, action: PayloadAction<ICell | null>) => {
      state.chosenFigure = action.payload;
    },
    setPossibleMoves: (state, action: PayloadAction<ICell[]>) => {
      state.possibleMoves = action.payload;
    },
    restart: (state) => {
      state.chosenFigure = null;
      state.possibleMoves = [];
    },
  },
});

export default MoveSlice.reducer;

```

turnSlice.ts

```

import { createSlice, PayloadAction } from "@reduxjs/toolkit";

const initialState = {
  yourTurn: true,
};

export const TurnSlice = createSlice({
  name: "turn",
  initialState,
  reducers: {
    giveTurn: (state, action: PayloadAction<boolean>) => {
      state.yourTurn = action.payload;
    },
    restart: (state) => {
      state.yourTurn = true;
    },
  },
});

export default TurnSlice.reducer;

```

redux.ts

```

import { TypedUseSelectorHook, useDispatch, useSelector } from "react-redux";
import { AppDispatch, RootState } from "../store/store";

```

```
export const useAppDispatch = () => useDispatch<AppDispatch>();
export const useAppSelector: TypedUseSelectorHook<RootState> = useSelector;
```

gameHelper.t

```
import { ICell } from "../types/board";

interface ITurnInfo {
  from: ICell;
  to: ICell;
}

class GameHelper {
  private lastThreePlayerTurns: ITurnInfo[] = [];
  private lastThreeAITurns: ITurnInfo[] = [];

  calculatePossibleMoves = (
    row: number,
    col: number,
    board: number[][]
  ): ICell[] => {
    const result: ICell[] = [];
    //TOP
    for (let i = row; i > 0; i--) {
      if (board[i - 1][col] !== 0) {
        result.push({ row: i, col });
        break;
      } else if (i - 1 === 0) {
        result.push({ row: i - 1, col });
        break;
      }
    }
    //BOT
    for (let i = row; i < 4; i++) {
      if (board[i + 1][col] !== 0) {
        result.push({ row: i, col });
        break;
      } else if (i + 1 === 4) {
        result.push({ row: i + 1, col });
        break;
      }
    }
    //LEFT
    for (let i = col; i > 0; i--) {
      if (board[row][i - 1] !== 0) {
        result.push({ row, col: i });
        break;
      } else if (i - 1 === 0) {
```

```

        result.push({ row, col: i - 1 });
        break;
    }
}
//RIGHT
for (let i = col; i < 4; i++) {
    if (board[row][i + 1] !== 0) {
        result.push({ row, col: i });
        break;
    } else if (i + 1 === 4) {
        result.push({ row, col: i + 1 });
        break;
    }
}
//TOP RIGHT DIAGONAL
for (let i = row, j = col; i > 0 && j < 4; i--, j++) {
    if (board[i - 1][j + 1] !== 0) {
        if (i !== row || j !== col) result.push({ row: i, col: j });
        break;
    } else if (i - 1 === 0 || j + 1 === 4) {
        result.push({ row: i - 1, col: j + 1 });
        break;
    }
}
//TOP LEFT DIAGONAL
for (let i = row, j = col; i > 0 && j > 0; i--, j--) {
    if (board[i - 1][j - 1] !== 0) {
        if (i !== row || j !== col) result.push({ row: i, col: j });
        break;
    } else if (i - 1 === 0 || j - 1 === 0) {
        result.push({ row: i - 1, col: j - 1 });
        break;
    }
}
//BOT RIGHT DIAGONAL
for (let i = row, j = col; i < 4 && j < 4; i++, j++) {
    if (board[i + 1][j + 1] !== 0) {
        if (i !== row || j !== col) result.push({ row: i, col: j });
        break;
    } else if (i + 1 === 4 || j + 1 === 4) {
        result.push({ row: i + 1, col: j + 1 });
        break;
    }
}
//BOT LEFT DIAGONAL
for (let i = row, j = col; i < 4 && j > 0; i++, j--) {
    if (board[i + 1][j - 1] !== 0) {
        if (i !== row || j !== col) result.push({ row: i, col: j });
        break;
    } else if (i + 1 === 4 || j - 1 === 0) {

```

```

        result.push({ row: i + 1, col: j - 1 });
        break;
    }
}
return result.filter((cell) => {
    if (!(cell.row === row && cell.col === col)) return cell;
});
});
checkForWin = (board: number[][], side: number): boolean => {
    //left -> right
    for (let i = 0; i < 5; i++) {
        let sum = 0;
        for (let j = 0; j < 5; j++) {
            if (board[i][j] === side) {
                sum += board[i][j];
            }
        }
        if (sum === side * 3) {
            const resultCollumn = [];
            for (let k = 0; k < 5; k++) resultCollumn.push(board[i][k]);
            // console.log(resultCollumn.join(""));

            let count = 0;
            for (let m = 0; m < resultCollumn.length - 1; m++) {
                if (resultCollumn[m] === side && resultCollumn[m + 1] === side)
                    count++;
            }
            if (count === 2) {
                return true;
            }
        }
    }
}
//top -> bot
for (let i = 0; i < 5; i++) {
    let sum = 0;
    for (let j = 0; j < 5; j++) {
        if (board[j][i] === side) {
            sum += board[j][i];
        }
    }
    if (sum === side * 3) {
        const resultCollumn = [];
        for (let k = 0; k < 5; k++) resultCollumn.push(board[k][i]);
        // console.log(resultCollumn.join(""));

        let count = 0;
        for (let m = 0; m < resultCollumn.length - 1; m++) {
            if (resultCollumn[m] === side && resultCollumn[m + 1] === side)
                count++;
        }
    }
}

```

```

        if (count === 2) {
            return true;
        }
    }
}

// right top diagonal
for (let i = 2; i < 5; i++) {
    let sum = 0;
    for (let j = 0; j < i + 1; j++) {
        if (board[i - j][j] === side) {
            sum += board[i - j][j];
        }
    }
    if (sum === side * 3) {
        const resultDiagonal = [];
        for (let k = 0; k < i + 1; k++) {
            resultDiagonal.push(board[i - k][k]);
            // console.log(resultDiagonal.join(""));

            let count = 0;
            for (let m = 0; m < resultDiagonal.length - 1; m++) {
                if (resultDiagonal[m] === side && resultDiagonal[m + 1] === side)
                    count++;
            }
            if (count === 2) {
                return true;
            }
        }
    }
}

// right bot diagonal
for (let j = 2; j >= 0; j--) {
    let sum = 0;
    for (let i = 0; i < 5 - j; i++) {
        if (board[i][j + i] === side) {
            sum += board[i][j + i];
        }
    }
    if (sum === side * 3) {
        const resultDiagonal = [];
        for (let k = 0; k < 5 - j; k++) {
            resultDiagonal.push(board[k][j + k]);
            // console.log(resultDiagonal.join(""));

            let count = 0;
            for (let m = 0; m < resultDiagonal.length - 1; m++) {
                if (resultDiagonal[m] === side && resultDiagonal[m + 1] === side)
                    count++;
            }
            if (count === 2) {

```



```

        return true;
    }
}
}
}
return false;
};

addPlayerTurn = (from: ICell, to: ICell) => {
    if (this.lastThreePlayerTurns.length === 3) {
        this.lastThreePlayerTurns.shift();
    }
    this.lastThreePlayerTurns.push({ from, to });
};

addAITurn = (from: ICell, to: ICell) => {
    if (this.lastThreeAITurns.length === 3) {
        this.lastThreeAITurns.shift();
    }
    this.lastThreeAITurns.push({ from, to });
};

checkForDraw = (): boolean => {
    if (
        this.lastThreeAITurns.length < 3 ||
        this.lastThreePlayerTurns.length < 3
    )
        return false;
    const aiFirst = this.lastThreeAITurns[0];
    const sameAITurns = this.lastThreeAITurns.filter(
        (info) =>
            info.from.row === aiFirst.from.row &&
            info.from.col === aiFirst.from.col &&
            info.to.row === aiFirst.to.row &&
            info.to.col === aiFirst.to.col
    );
    if (sameAITurns.length !== 2) return false;
    if (
        this.lastThreeAITurns[0].from.col !== this.lastThreeAITurns[1].to.col ||
        this.lastThreeAITurns[0].from.row !== this.lastThreeAITurns[1].to.row
    )
        return false;

    const playerFirst = this.lastThreePlayerTurns[0];
    const samePlayerTurns = this.lastThreePlayerTurns.filter(
        (info) =>
            info.from.row === playerFirst.from.row &&
            info.from.col === playerFirst.from.col &&
            info.to.row === playerFirst.to.row &&
            info.to.col === playerFirst.to.col
    );
};

```

```

);
if (samePlayerTurns.length !== 2) return false;
if (
  this.lastThreePlayerTurns[0].from.col !==
    this.lastThreePlayerTurns[1].to.col ||
  this.lastThreePlayerTurns[0].from.row !==
    this.lastThreePlayerTurns[1].to.row
)
  return false;
return true;
};
estimateNow = (board: number[][]): number => {
  const side = 1;
  let result = 0;
  //left -> right
  for (let i = 0; i < 5; i++) {
    let sum = 0;
    for (let j = 0; j < 5; j++) {
      if (board[i][j] === side) {
        sum += board[i][j];
      }
    }
    if (sum === 2) result += 0.2;
    else if (sum === 3) result += 0.3;
  }
  //top -> bot
  for (let i = 0; i < 5; i++) {
    let sum = 0;
    for (let j = 0; j < 5; j++) {
      if (board[j][i] === 1) {
        sum += board[j][i];
      }
    }
    if (sum === 2) result += 0.2;
    else if (sum === 3) result += 0.3;
  }
  // right top diagonal
  for (let i = 2; i < 5; i++) {
    let sum = 0;
    for (let j = 0; j < i + 1; j++) {
      if (board[i - j][j] === side) {
        sum += board[i - j][j];
      }
    }
    if (sum === 2) result += 0.2;
    else if (sum === 3) result += 0.3;
  }
  // right bot diagonal
  for (let j = 2; j >= 0; j--) {

```

```

    let sum = 0;
    for (let i = 0; i < 5 - j; i++) {
      if (board[i][j + i] === side) {
        sum += board[i][j + i];
      }
    }
    if (sum === 2) result += 0.2;
    else if (sum === 3) result += 0.3;
  }
  return result;
};
}

export default new GameHelper();

```

game.ts

```

import { ICell } from "../types/board";
import GameHelper from "../helpers/GameHelper";
import { SCORE } from "../types/board";
import { TDifficulty } from "../store/reducers/gameStateSlice";

class Game {
  private USER_TURN = false;
  private AI_TURN = true;
  private MAX_DEPTH = 1;
  computerTurn(board: number[][], difficulty: TDifficulty): ICell[] {
    switch (difficulty) {
      case 1:
        this.MAX_DEPTH = 1;
        break;
      case 2:
        this.MAX_DEPTH = 3;
        break;
      case 3:
        this.MAX_DEPTH = 5;
        break;
      case 4:
        this.MAX_DEPTH = 6;
        break;
      default:
        this.MAX_DEPTH = 1;
    }

    let result: ICell[] = [];
    let bestScore = -2;
    const boardCopy: number[][] = [];
    for (let i = 0; i < 5; i++) {
      boardCopy.push([...board[i]]);
    }
  }
}

```

```

    }
    const exception = {};
    try {
      boardCopy.forEach((row, i) => {
        row.forEach((cell, j) => {
          if (cell === 1) {
            const possibleMoves = GameHelper.calculatePossibleMoves(
              i,
              j,
              boardCopy
            );
            possibleMoves.forEach((possibleMove) => {
              const { row: moveRow, col } = possibleMove;
              boardCopy[moveRow][col] = 1;
              boardCopy[i][j] = 0;
              const score = this.minimax(boardCopy, 0, -20, 20, this.USER_TURN);
              if (
                score === 1 &&
                GameHelper.checkForWin(boardCopy, SCORE.AI_WIN)
              ) {
                result = [possibleMove, { row: i, col: j }];
                throw exception;
              }
              boardCopy[moveRow][col] = 0;
              boardCopy[i][j] = 1;
              if (score >= bestScore) {
                bestScore = score;
                result = [possibleMove, { row: i, col: j }];
              }
            });
          }
        });
      });
    } catch (e) {
      if (e !== exception) throw e;
    }
    return result;
  }

  minimax = (
    board: number[][],
    depth: number,
    alpha: number,
    beta: number,
    isAiTurn: boolean
  ): number => {
    if (GameHelper.checkForWin(board, SCORE.USER_WIN)) {
      return SCORE.USER_WIN;
    } else if (GameHelper.checkForWin(board, SCORE.AI_WIN)) {
      return SCORE.AI_WIN;
    }
  }

```

```

    } else if (GameHelper.checkForDraw()) {
        return SCORE.DRAW;
    } else if (depth === this.MAX_DEPTH) {
        return GameHelper.estimateNow(board);
    }
    let bestScore = isAiTurn ? -20 : 20;
    if (isAiTurn) {
        //chose move that is the best for us
        board.forEach((row, i) => {
            row.forEach((cell, j) => {
                if (cell === 1) {
                    const possibleMoves = GameHelper.calculatePossibleMoves(
                        i,
                        j,
                        board
                    );
                    for (let possibleMove of possibleMoves) {
                        const { row: moveRow, col } = possibleMove;
                        board[moveRow][col] = 1;
                        board[i][j] = 0;
                        const score = this.minimax(
                            board,
                            depth + 1,
                            alpha,
                            beta,
                            this.USER_TURN
                        );
                        board[moveRow][col] = 0;
                        board[i][j] = 1;
                        bestScore = Math.max(score, bestScore);
                        alpha = Math.max(alpha, score);
                        if (beta <= alpha) break;
                    }
                }
            });
        });
    } else {
        //enemy chooses the move the is the best for him and worst for us
        board.forEach((row, i) => {
            row.forEach((cell, j) => {
                if (cell === -1) {
                    const possibleMoves = GameHelper.calculatePossibleMoves(
                        i,
                        j,
                        board
                    );
                    for (let possibleMove of possibleMoves) {
                        const { row: moveRow, col } = possibleMove;
                        board[moveRow][col] = -1;
                        board[i][j] = 0;

```

```

        const score = this.minimax(
            board,
            depth + 1,
            alpha,
            beta,
            this.AI_TURN
        );
        board[moveRow][col] = 0;
        board[i][j] = -1;
        bestScore = Math.min(score, bestScore);
        beta = Math.min(beta, score);
        if (beta <= alpha) {
            break;
        }
    }
}
});
});
}

return bestScore;
};
}

export default new Game();

```

game.tsx

```

import * as React from "react";
import { useAppSelector, useAppDispatch } from "../../hooks/redux";
import { SCORE, TBoard } from "../../types/board";
import { Colors } from "../../types/colors-enum";
import Figure from "../../figure/Figure";
import { MoveSlice } from "../../store/reducers/moveSlice";
import clsx from "clsx";
import gameHelper from "../../helpers/GameHelper";
import { TurnSlice } from "../../store/reducers/turnSlice";
import { BoardSlice } from "../../store/reducers/boardSlice";
import game from "../../game/Game";
import gameStateSlice, {
    GameStateSlice,
} from "../../store/reducers/gameStateSlice";

interface GameProps {
    board: TBoard;
}

const Game: React.FC<GameProps> = ({ board }) => {
    const { chosenFigure, possibleMoves } = useAppSelector((store) => store.move);

```

```

const { yourTurn } = useAppSelector((store) => store.turn);
const { state: gameState, difficulty } = useAppSelector(
  (store) => store.game
);
const dispatch = useAppDispatch();
const checkPossibleMove = React.useCallback(
  (row: number, col: number) => {
    return Boolean(
      possibleMoves.find((item) => item.row === row && item.col === col)
    );
  },
  [possibleMoves]
);
const onFigureBlackClick = (row: number, col: number) => {
  if (yourTurn) {
    dispatch(MoveSlice.actions.chooseFigure({ row, col }));
    const calculatedPossibleMoves = gameHelper.calculatePossibleMoves(
      row,
      col,
      board
    );
    dispatch(MoveSlice.actions.setPossibleMoves(calculatedPossibleMoves));
  }
};

const computerTurn = (boardLocal: number[][][]) => {
  if (!yourTurn) {
    const [moveToDo, moveFrom] = game.computerTurn(boardLocal, difficulty);
    dispatch(
      BoardSlice.actions.moveFrom({
        row: moveFrom.row,
        col: moveFrom.col,
      })
    );
    dispatch(
      BoardSlice.actions.setWhite({ row: moveToDo.row, col: moveToDo.col })
    );
    const boardCopy: number[][] = [];
    for (let i = 0; i < 5; i++) {
      boardCopy.push([...boardLocal[i]]);
    }
    boardCopy[moveToDo.row][moveToDo.col] = 1;
    boardCopy[moveFrom.row][moveFrom.col] = 0;
    gameHelper.addAITurn(moveFrom, moveToDo);
    if (gameHelper.checkForWin(boardCopy, SCORE.AI_WIN)) {
      dispatch(GameStateSlice.actions.setResultOfGame("whiteWin"));
    } else if (gameHelper.checkForDraw()) {
      dispatch(GameStateSlice.actions.setResultOfGame("draw"));
    } else {
      dispatch(TurnSlice.actions.giveTurn(true));
    }
  }
};

```

```

    }
  }
};
React.useEffect(() => {
  if (!yourTurn) {
    computerTurn(board);
  }
}, [yourTurn]);

const playerTurn = (i: number, j: number) => {
  if (yourTurn && (gameState === "going" || gameState === "start")) {
    if (checkPossibleMove(i, j)) {
      if (chosenFigure) {
        if (gameState === "start") {
          dispatch(GameStateSlice.actions.setResultOfGame("going"));
        }
        dispatch(
          BoardSlice.actions.moveFrom({
            row: chosenFigure.row,
            col: chosenFigure.col,
          })
        );
        dispatch(BoardSlice.actions.setBlack({ row: i, col: j }));
        dispatch(MoveSlice.actions.chooseFigure(null));
        dispatch(MoveSlice.actions.setPossibleMoves([]));
        const boardCopy: number[][] = [];
        for (let i = 0; i < 5; i++) {
          boardCopy.push([...board[i]]);
        }
        boardCopy[i][j] = -1;
        boardCopy[chosenFigure.row][chosenFigure.col] = 0;
        gameHelper.addPlayerTurn(chosenFigure, { row: i, col: j });
        if (gameHelper.checkForWin(boardCopy, SCORE.USER_WIN)) {
          dispatch(GameStateSlice.actions.setResultOfGame("blackWin"));
        } else if (gameHelper.checkForDraw()) {
          dispatch(GameStateSlice.actions.setResultOfGame("draw"));
        } else {
          dispatch(TurnSlice.actions.giveTurn(false));
        }
      }
    } else {
      dispatch(MoveSlice.actions.chooseFigure(null));
      dispatch(MoveSlice.actions.setPossibleMoves([]));
    }
  }
};

return (
  <>
    {board.map((row, i) => {
      return (

```



```

    <div className="row" key={i}>
      {row.map((cell, j) => {
        return (
          <div
            className={clsx({
              cell: true,
              green: checkPossibleMove(i, j),
            })}
            key={j}
            onClick={(e) => {
              playerTurn(i, j);
            }}
          >
            {cell === 1 ? (
              <Figure
                color={Colors.white}
                chosen={i === chosenFigure?.row && j === chosenFigure.col}
              />
            ) : null}
            {cell === -1 ? (
              <Figure
                color={Colors.black}
                chosen={i === chosenFigure?.row && j === chosenFigure.col}
                onFigureClick={(e) => {
                  e.stopPropagation();
                  if (gameState === "going" || gameState === "start") {
                    onFigureBlackClick(i, j);
                  }
                }}
              />
            ) : null}
          </div>
        );
      })}
    </div>
  );
});
</>
);
};

export default Game;

```

figure.css

```

.figure {
  width: 85px;
  height: 85px;
  border-radius: 50%;
}

```

```

}

.white {
  background-color: white;
}

.black {
  background-color: black;
  cursor: pointer;
}

.chosen {
  background-color: #222;
}

```

Figure.tsx

```

import * as React from "react";
import { Colors } from "../../types/colors-enum";
import "./figure.css";
import clsx from "clsx";

interface FigureProps {
  color: Colors;
  chosen?: boolean;
  onFigureClick?: (e: any) => void;
}

const Figure: React.FC<FigureProps> = ({ color, chosen, onFigureClick }) => {
  return (
    <div
      className={clsx({
        figure: true,
        white: color === Colors.white,
        black: color === Colors.black,
        chosen,
      })}
      onClick={onFigureClick}
    ></div>
  );
};

export default Figure;

```

board.css

```

.board {
  user-select: none;
  display: grid;
  grid-template-areas:
    "empty letter letter letter letter"
    "number game game game game"
    "number game game game game"
    "number game game game game"
    "number game game game game"
    "number game game game game";
  grid-template-columns: repeat(6, 100px);
  grid-template-rows: repeat(6, 100px);
  color: white;
}

.empty {
  grid-area: empty;
}

.top {
  grid-area: letter;
  display: flex;
}

.top-letter {
  width: 100px;
  display: flex;
  justify-content: center;
  align-items: center;
  border-left: 1px solid white;
}

.column {
  grid-area: number;
  display: flex;
  flex-direction: column;
}

.column-number {
  height: 100px;
  display: flex;
  justify-content: center;
  align-items: center;
  border-top: 1px solid white;
  border-right: 1px solid white;
}

.game {
  grid-area: game;
  display: grid;
  grid-template-rows: repeat(5, 100px);
}

```

```

.row {
  display: grid;
  grid-template-columns: repeat(5, 100px);
}

.cell {
  border-top: 1px solid white;
  border-right: 1px solid white;
  display: flex;
  justify-content: center;
  align-items: center;
}

.cell:last-child {
  border-right: none;
}

.green {
  background: rgba(78, 141, 72, 0.808);
}

```

Board.tsx

```

import * as React from "react";
import { TBoard } from "../../types/board";
import Game from "../game/Game";
import "./board.css";

interface BoardProps {
  board: TBoard;
}

const Board: React.FC<BoardProps> = ({ board }) => {
  return (
    <div className="board">
      <div className="empty"></div>
      <div className="top">
        <div className="top-letter">A</div>
        <div className="top-letter">B</div>
        <div className="top-letter">C</div>
        <div className="top-letter">D</div>
        <div className="top-letter">E</div>
      </div>
      <div className="column">
        <div className="column-number">1</div>
        <div className="column-number">2</div>
        <div className="column-number">3</div>
        <div className="column-number">4</div>
        <div className="column-number">5</div>

```

```

    </div>
    <div className="game">
      <Game board={board} />
    </div>
  </div>
);
};

export default Board;

```

Встановлені пакети:

```

"dependencies": {
  "@reduxjs/toolkit": "^1.6.2",
  "@testing-library/jest-dom": "^5.11.4",
  "@testing-library/react": "^11.1.0",
  "@testing-library/user-event": "^12.1.10",
  "@types/jest": "^26.0.15",
  "@types/node": "^12.0.0",
  "@types/react": "^17.0.0",
  "@types/react-dom": "^17.0.0",
  "clsx": "^1.1.1",
  "react": "^17.0.2",
  "react-dom": "^17.0.2",
  "react-redux": "^7.2.6",
  "react-scripts": "4.0.3",
  "typescript": "^4.1.2",
  "web-vitals": "^1.0.1"
},

```

3.1.2 Приклади роботи

На рисунках 3.1 і 3.2 показані приклади роботи програми.

Neutreeko

Black win the game!

Restart







	A	B	C	D	E
1					
2					
3					
4					
5					

Рисунок 3.1 – На легкому рівні складності гравець переміг комп'ютер

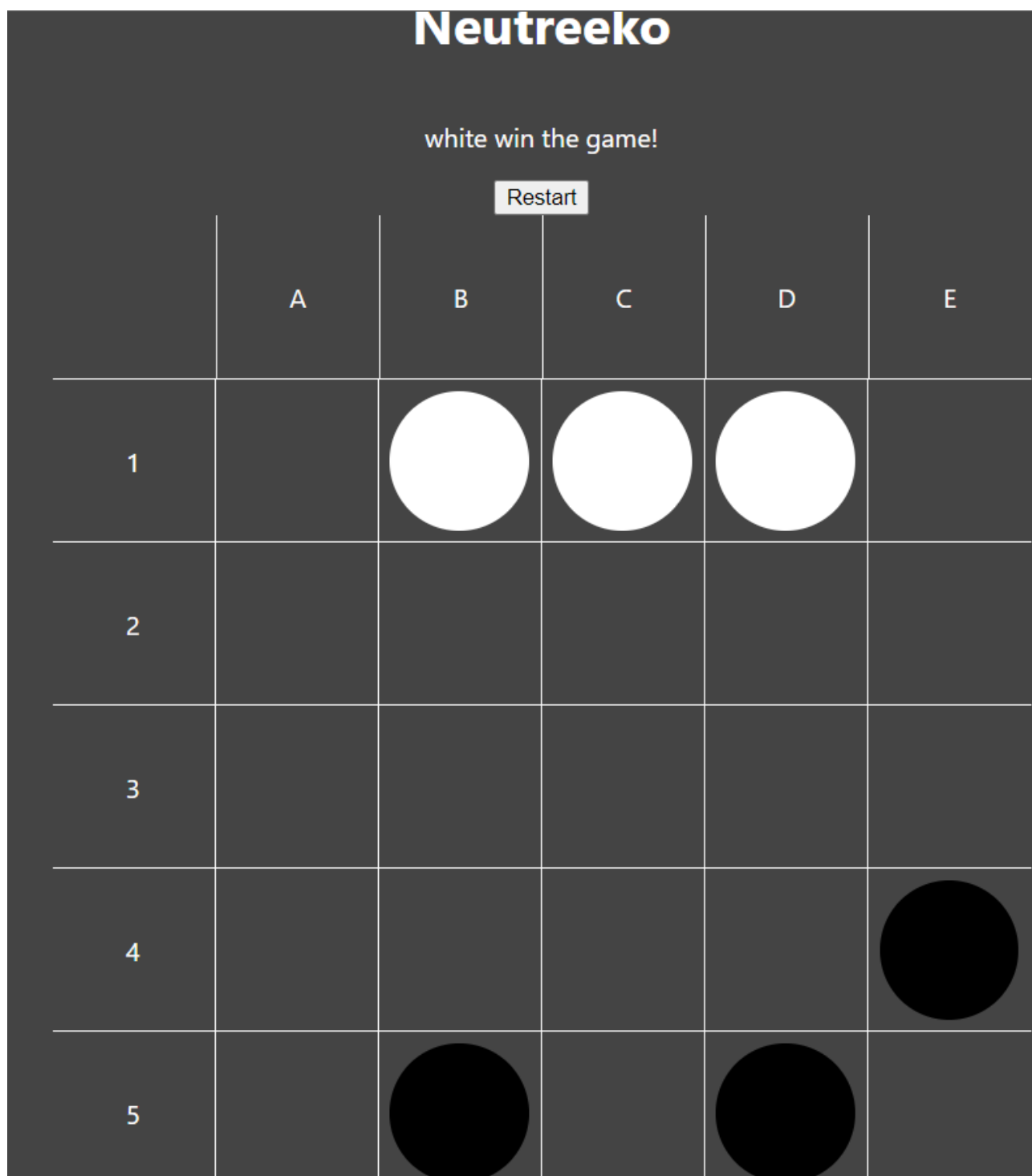


Рисунок 3.2 – На складному рівні складності комп'ютер переміг гравця

ВИСНОВОК

В рамках даної лабораторної роботи я створив програмну реалізацію гри Neutreeko за допомогою веб технологій таких як React, Redux, Typescript. Було реалізовано можливість гри з комп'ютерним опоненти з можливістю вибору рівня складності. Комп'ютерний опонент обирає ходи за допомогою відомого алгоритму `minimax`, а точніше його вдосконаленої версії «Альфа-бета відсікання».

Спробувати пограти в цю гру можна за посиланням: <https://neutreeko-game.vercel.app/>

КРИТЕРІЇ ОЦІНЮВАННЯ

При здачі лабораторної роботи до 10.12.2020 включно максимальний бал дорівнює – 5. Після 10.12.2020 максимальний бал дорівнює – 1.

Критерії оцінювання у відсотках від максимального балу:

- програмна реалізація алгоритму – 95%;
- висновок – 5%.

+1 додатковий бал можна отримати за реалізацію рівнів складності.