รายงานวิชาDistributed Computer And Web Technology

รหัสวิชา 240-311

Socket Assignment

Tic Tac Toe

โดย

นายแวรอฮีม บินแวมุสตอปา 6210110690

นายนุอ์มาน เบ็ญอับดุลรอชีด 6210110176

นายไซยิดฮารูน อัลอิดรุส 6210110650

นายอาดิล วาเต๊ะ 6210110429

เสนอ

ผศ.สุธน แซ่ว่อง

ภาควิชาวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์

มหาวิทยาลัยสงขลานครินทร์

คำนำ

รายงานเล่มนี้จัดทำขึ้นเพื่อเป็นส่วนหนึ่งของรายวิชา240-311 DISTRIBUTED COMP & WEB TECH เพื่อให้ได้ศึกษาหาความรู้ในการทำงานของ Client server จากการปฏิบัติด้วยการออกแบบทดลองเขียนโปรแกรมด้วย Node.js ด้วย Javascript และได้ศึกษาอย่างเข้าใจเพื่อเป็นประโยชน์กับการเรียนและการนำไปต่อยอดใช้งานในอนาคตตามจุดประสงค์ของอาจารย์ผู้สอนในรายวิชานี้

ผู้จัดทำหวังว่ารายงานเล่มนี้จะเป็นประโยชน์กับผู้อ่าน หรือ นักศึกษา ที่กำลังหาข้อมูลเรื่องนี้อยู่ หากมีข้อแนะนำหรือข้อผิดพลาดประการใด ผู้จัดทำขอน้อมรับไว้และขออภัยมา ณ ที่นี้ด้วย

ผู้จัดทำ

นายแวรอฮีม บินแวมุสตอปา

นายนุอ์มาน เบ็ญอับดุลรอชีด

นายไซยิดฮารูน อัลอิดรุส

นายอาดิล วาเต๊ะ

สารบัญ

เรื่อง **หน้า**

เนื้อหาที่เกี่ยวข้อ

Node.js คืออะไร ?

Node.js คือ Cross Platform Runtime Environment สำหรับฝั่ง Server และเป็น Open Source ซึ่งเขียนด้วยภาษา JavaScript สรุปรวมๆ Node.js ก็คือ Platform ตัวหนึ่งที่เขียนด้วย JavaScript สำหรับเป็น Web Server นั่นเอง

Client/Server คืออะไร

Client คือ เครื่องคอมพิวเตอร์ที่ไปร้องขอบริการและรับบริการอย่างใดอย่างหนึ่งจาก Server server คือเครื่องคอมพิวเตอร์หรือระบบปฏิบัติการหรือโปรแกรมคอมพิวเตอร์ที่ทำหน้าที่ให้บริการอย่างใดอย่างหนึ่งหรือหลายอย่าง โดยอาศัยโปรแกรม Web serverแก่เครื่องคอมพิวเตอร์หรือโปรแกรมคอมพิวเตอร์ที่เป็นลูกข่าย ในระบบเครื่อข่าย

Tic Tac Toe

เกมจะมีลักษณะคือ มีเส้นขนานสองเส้นแนวตั้งและแนวนอนมาตัดกัน

พื้นที่ทั้งหมด 3*3 = 9 ช่อง มีผู้เล่น 2 คน

กติกาการเล่น

1. ต้องตกลงกันว่าใครจะเป็นคนใช้รูปอะไร O , X

2. เมื่อตกลงกันเสร็จแล้วก็ต้องตกลงกันอีกครั้งว่าฝ่ายไหนจะเริ่มก่อน

3. ผลัดกันลงตามช่องที่ว่างอยู่คนละ 1 ครั้ง

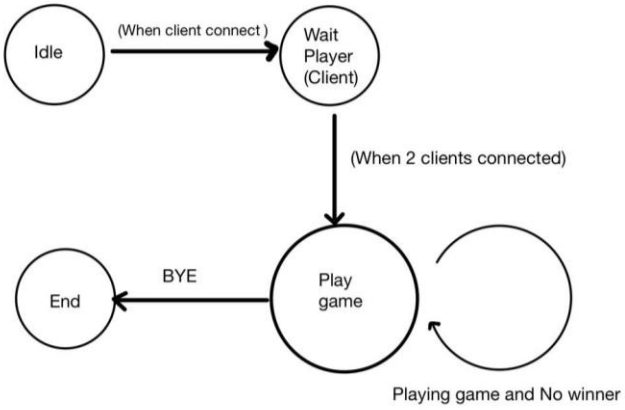
4. ผู้ที่ชนะเกมนี้จะต้องมีรูป O , X ของตัวเองนั้นเรียงติดกัน 3 ช่อง แต่ถ้าไม่มีฝั่งไหนเรียงติดกันจะถือว่าเสมอ

การทำงานของโปรแกรม

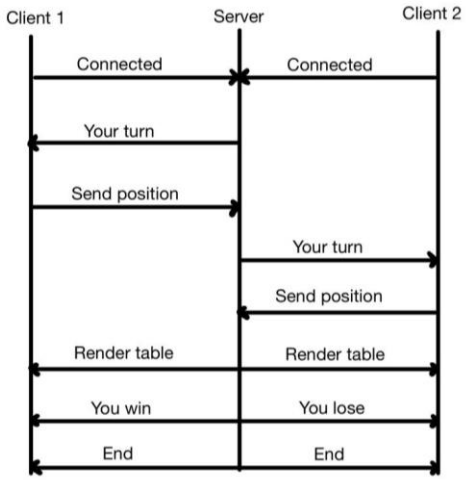
การทำงานของโปรแกรมแบ่งออกเป็น 2 ส่วน ในส่วนแรกคือ server.js คือส่วนที่ทำหน้าที่ในการทำงานของserver ขอให้บริการการทำงานของจัวโปรแกรม ในส่วนที่สองคือส่วนของ client.js เป็นส่วนของผู้เล่นซึ่งในเกมนี้มีผู้เล่นทั้งหมด 2 คนด้วยกันเพื่อให้เกมสามารถดำเนินการต่อไปได้โดยที่ ผู้ที่ชนะเกมนี้จะต้องมีรูป O , X ของตัวเองนั้นเรียงติดกัน 3 ช่อง และเกมจะดำเนินไปเรื่อยๆจนกว่าจะมีผู้ชนะ และแสดงสถานะว่าผู้เล่นฝ่ายใดเป็นผู้ชนะหรือแพ้ เมื่อได้ผลการตัดสินก็ถือว่าสิ้นสุดารเล่นเกม ตัวผู้เล่นทั้งสองฝ่ายจะถูกออกจากเกมโดยอัตโนมัติ

Diagram

State diagram



Sequence diagram



ส่วนของโปรแกรม

Client.js

```
const net = require('net');
const delay = require('delay');


const HOST = 'localhost';
const PORT = 8000;


var client = new net.Socket();
client.connect(PORT, HOST, () => {
  console.log(`Connected to ${HOST}:${PORT}`);
  client.write(`Hello, I am ${client.address().address}`);
});


client.on('data', (data) => {
  console.log(`Client received: ${data}`);
  if (data == 'game has already begun') client.destroy();
  else if (data == 'prepare for game') client.write('Ready');
  else if (
    data.includes('You win') ||
    data.includes('You lose') ||
    data.includes('Draw')
  ) {
    client.write('Bye'); // disconnect เมื่อจบเกม
    client.destroy();
  } else if (
    data.includes('Your turn') ||
    data.includes('Enter available position')
  ) {
    (async () => {
      await delay(1000);
      client.write(`${Math.floor(Math.random() * 9)}`);
     })();
  }
});


client.on('close', () => {
```

```
  console.log('Client closed');
});


client.on('error', (err) => {
  console.error(err);
});
```

Server.js

```
const net = require('net');


const host = 'localhost';
const port = 8000;
let playerNum = 0;
let players = Object.create(null);
let symbol = '';


const server = net.createServer();
server.listen(port, host, () => {
  console.log(`TCP server listening on ${host}:${port}`);
});


let sockets = [];
let phase = 1;
let board = ['-', '-', '-', '-', '-', '-', '-', '-', '-'];
let currentPlayer = 1;


let renderedBoard = `
      | ${board[0]} | ${board[1]} | ${board[2]} |
      | ${board[3]} | ${board[4]} | ${board[5]} |
      | ${board[6]} | ${board[7]} | ${board[8]} |
  `;


server.on('connection', (socket) => {
  if (sockets.length == 2) { //reject client ที่เชื่อมต่อหลังจากเกมเริ่ม
    socket.write('the game has already begun');
```

```
    return;
  }
  playerNum++;
  socket.nickname = `player${playerNum}`;
  var clientName = socket.nickname;


  players[clientName] = socket;
  socket.on('close', () => {
    delete players[socket.nickname];
  });
  var clientAddress = `${socket.remoteAddress}:${socket.remotePort}`;
  console.log(`new client connected: ${clientAddress}`);
  sockets.push(socket);


  socket.on('data', (data) => {
    console.log(`Client ${clientAddress}: ${data}`);
    switch (phase) {
      case 1: // เฟสแรกคือรอให้ client มาเชื่อมต่อให้ครบ 2 client
        if (Object.keys(players).length < 2)
          socket.write(`wait for other player`);
        if (Object.keys(players).length == 2) {
          players[`player${1}`].write('prepare for game');
          players[`player${2}`].write('prepare for game');
          phase += 1;
        }
        break;


      case 2:
// เฟส 2 คือเมื่อมี client มาเชื่อมต่อครบ 2 client ก็ทำการ broadcast ตารางและเริ่มเกม
        if (data == 'Ready') {
          players[`player${currentPlayer}`].write('Your turn');
        }
        players[`player${1}`].write(renderedBoard);
        players[`player${2}`].write(renderedBoard);
        phase += 1;
        break;
      case 3:
// เฟส 3 คือให้ client แต่ละตัวส่งตำแหน่งของตารางที่ว่างให้ server เป็นฝ่าย update ตาราง
        if (parseInt(data) >= 0 && parseInt(data) <= 8) {
          if (isConflict(parseInt(data))) {
            players[`player${currentPlayer}`].write('Enter available position');
          } else {
```

```
            if (currentPlayer == 1) {
              symbol = 'X';
              board[parseInt(data)] = symbol;
            } else {
              symbol = 'O';
              board[parseInt(data)] = symbol;
            }
            renderedBoard = reRenderBoard();


            if (isBoardFilled()) {
              players[`player1`].write(renderedBoard);
              players[`player1`].write('Draw');
              players[`player2`].write(renderedBoard);
              players[`player2`].write('Draw');
              phase += 1;
            }


            if (
              isrowStraight(parseInt(data), symbol) ||
              isColumnStraight(parseInt(data), symbol) ||
              isDiagonalStraight(parseInt(data), symbol)
            ) {
              players[`player${currentPlayer}`].write(renderedBoard);
              players[`player${currentPlayer}`].write('You win');
              if (currentPlayer == 1) currentPlayer = 2;
              else currentPlayer = 1;
              players[`player${currentPlayer}`].write(renderedBoard);
              players[`player${currentPlayer}`].write('You lose');
              phase += 1;
            }


            if (currentPlayer == 1) players[`player${2}`].write(renderedBoard);
            else players[`player${1}`].write(renderedBoard);


            if (currentPlayer == 1) currentPlayer = 2;
            else currentPlayer = 1;


            console.log(renderedBoard);
```

```
            players[`player${currentPlayer}`].write('Your turn');
          }
        }
        break;
      case 4:
// เฟส 4 คือจบเกมและทำการล้างค่าต่างๆ และกลับไปอยู่ใน phase 1 เพื่อรอเริ่มเกมต่อไป
        (async () => {
          await clearBoard();
          renderedBoard = reRenderBoard();
          playerNum = 0;
          for (const key in players) {
            delete players[key];
          }
          phase = 1;
        })();
    }
  });


  socket.on('close', () => {
    let index = sockets.findIndex((disconnectedClient) => {
      return (
        disconnectedClient.remoteAddress === socket.remoteAddress &&
        disconnectedClient.remotePort === socket.remotePort
      );
    });
    if (index !== -1) sockets.splice(index, 1);
    sockets.forEach((sock) => {
      sock.write(`${clientAddress} disconnected\n`);
    });
    console.log(`connection closed: ${clientAddress}`);
  });


  socket.on('error', (err) => {
    console.log(`Error occurred in ${clientAddress}: ${err.message}`);
  });
});


const reRenderBoard = () => {
  return `
  | ${board[0]} | ${board[1]} | ${board[2]} |
  | ${board[3]} | ${board[4]} | ${board[5]} |
  | ${board[6]} | ${board[7]} | ${board[8]} |
```

```
`;
};


const isConflict = (index) => {
  if (board[index] != '-') return true;
  return false;
};


const isrowStraight = (index, currentSymbol) => {
  switch (index) {
    case 0:
    case 3:
    case 6:
      if (
        board[index] == currentSymbol &&
        board[index + 1] == currentSymbol &&
        board[index + 2] == currentSymbol
      )
        return true;
      return false;


    case 1:
    case 4:
    case 7:
      if (
        board[index] == currentSymbol &&
        board[index - 1] == currentSymbol &&
        board[index + 1] == currentSymbol
      )
        return true;
      return false;


    case 2:
    case 5:
    case 8:
      if (
        board[index] == currentSymbol &&
        board[index - 1] == currentSymbol &&
        board[index - 2] == currentSymbol
      )
        return true;
```

```
      return false;
  }
};


const isColumnStraight = (index, currentSymbol) => {
  switch (index) {
    case 0:
    case 1:
    case 2:
      if (
        board[index] == currentSymbol &&
        board[index + 3] == currentSymbol &&
        board[index + 6] == currentSymbol
      )
        return true;
      return false;


    case 3:
    case 4:
    case 5:
      if (
        board[index] == currentSymbol &&
        board[index - 3] == currentSymbol &&
        board[index + 3] == currentSymbol
      )
        return true;
      return false;


    case 6:
    case 7:
    case 8:
      if (
        board[index] == currentSymbol &&
        board[index - 3] == currentSymbol &&
        board[index - 6] == currentSymbol
      )
        return true;
      return false;
  }
};
```

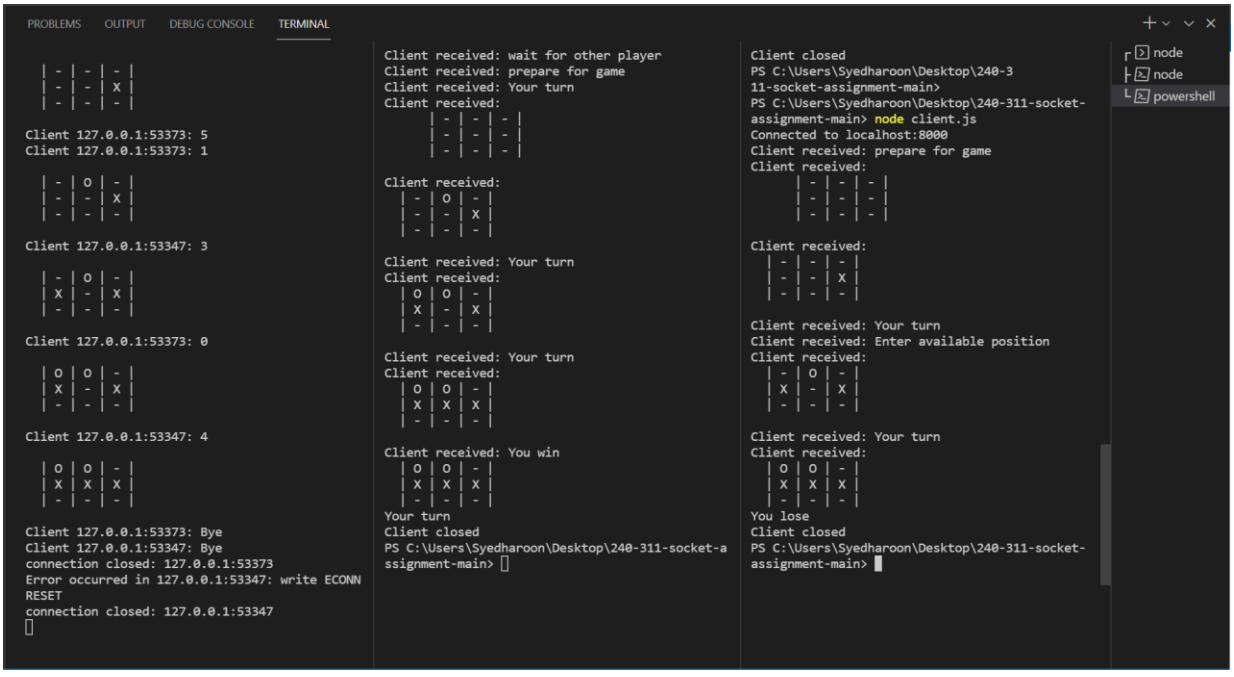
```
const isDiagonalStraight = (index, currentSymbol) => {
  switch (index) {
    case 0:
      if (
        board[index] == currentSymbol &&
        board[4] == currentSymbol &&
        board[8] == currentSymbol
      )
        return true;
      return false;
    case 2:
      if (
        board[index] == currentSymbol &&
        board[4 == currentSymbol &&
        board[6] == currentSymbol
      )
        return true;
      return false;
    case 4:
      if (
        board[index] == currentSymbol &&
        board[0] == currentSymbol &&
        board[8] == currentSymbol
      )
        return true;
      if (
        board[index] == currentSymbol &&
        board[2] == currentSymbol &&
        board[6] == currentSymbol
      )
        return true;
      return false;
    case 6:
      if (
        board[index] == currentSymbol &&
        board[4] == currentSymbol &&
        board[2] == currentSymbol
      )
        return true;
      return false;
    case 8:
      if (
        board[index] == currentSymbol &&
        board[4] == currentSymbol &&
        board[0] == currentSymbol
```

```
      )
        return true;
      return false;
  }
};


const isBoardFilled = () => {
  return !board.some((element) => element == '-');
};


const clearBoard = async () => {
  board = ['-', '-', '-', '-', '-', '-', '-', '-', '-'];
};
```

ผลลัพธ์ของโปรแกรม



อ้างอิง

Client Server คืออะไร ไคลเอ็นท์ เซิร์ฟเวอร์
คือระบบเครือข่ายคอมพิวเตอร์ซึ่งประกอบด้วยเครื่องคอมพิวเตอร์ที่ให้บริการ server เครื่องคอมพิวเตอร์ที่ขอรับบริการ client (mindphp.com)

วิธีเล่นเกม Tic Tac Toe ยังไงให้มีชัยชนะ!!! | TrueID In-Trend

https://www.knowledgehut.com/tutorials/node-js/socket-services

https://stackoverflow.com/questions/35367897/how-to-send-data-to-specific-sockets-created-using-net-createserver-connection