دانشگاه صنعتی امیرکبیر

(پلی تکنیک تهران)

دانشکده ریاضی و علوم کامپیوتر

گزارش چهارم

پیاده سازی یک بازی دو نفره با قابلیت های جستجوی تخاصمی

نگارش

سروش آریانا

استاد

دکتر مهدی قطعی

فروردین ۱۴۰۰

مقدمه

در بازی های کامپیوتری، ما قادر به تولید درخت آن ها هستیم و حل کـردن مسـئله، بـازی ، و یـا پـازل مورد نظر، به جستجو در درخت reduce میشود. پس جستجو نقش اساسی در ایـن گونـه مسـائل بـازی میکند. روش های مختلفی برای بازی های مختلفی ارائه شده است؛ اما ما در این گـزارش بـه یـک بـازی کلاسیک به نام tic tac toe با استفاده از الگوریتم minimax می پردازیم و زوایای این مسئله را بررسی میکنیم.

۱- پیاده سازی

به علت ساده بودن Back-end این بازی و نیز نیاز به داشتن front-end از زبان برنامه نویسی Javascript و در کنار آن برای مصور سازی از HTML,CSS ساده ای استفاده شده است. سورس کد ها در قسمت پیوست آورده شده اند و علاوه بر این، در کنار فایل pdf گزارش ارسال شده اند. در ادامه توضیحاتی درباره برنامه و نحوه اجرا ارائه شده است.

۲- توضیحاتی درباره بازی و جزئیات پیاده سازی و یک نمونه از اجرای بازی

همان طور که میدانیم این بازی یک بازی دو نفره است که شرط برنده شدن در آن تصاحب یک سطر، ستون ،و یا یک قطر است.

نمایی از چک کردن حالت های مختلف برای برنده شدن در تصویر زیر آمده است. (در پیاده سازی این تصویر بهینه سازی هم شده است)

```
// winning combinations using the board indexies
function winning(board, player){
 if (
 (board[0] == player && board[1] == player && board[2] == player) ||
 (board[3] == player && board[4] == player && board[5] == player) ||
 (board[6] == player && board[7] == player && board[8] == player) ||
 (board[0] == player && board[3] == player && board[6] == player) ||
 (board[1] == player && board[4] == player && board[7] == player) ||
 (board[2] == player && board[5] == player && board[8] == player) ||
 (board[0] == player && board[4] == player && board[8] == player) ||
 (board[2] == player && board[4] == player && board[6] == player)
 ) {
 return true;
 } else {
 return false;
 }
}
```

نمایه سازی الگوریتم مینیمکس در تصویر زیر انجام شده است:

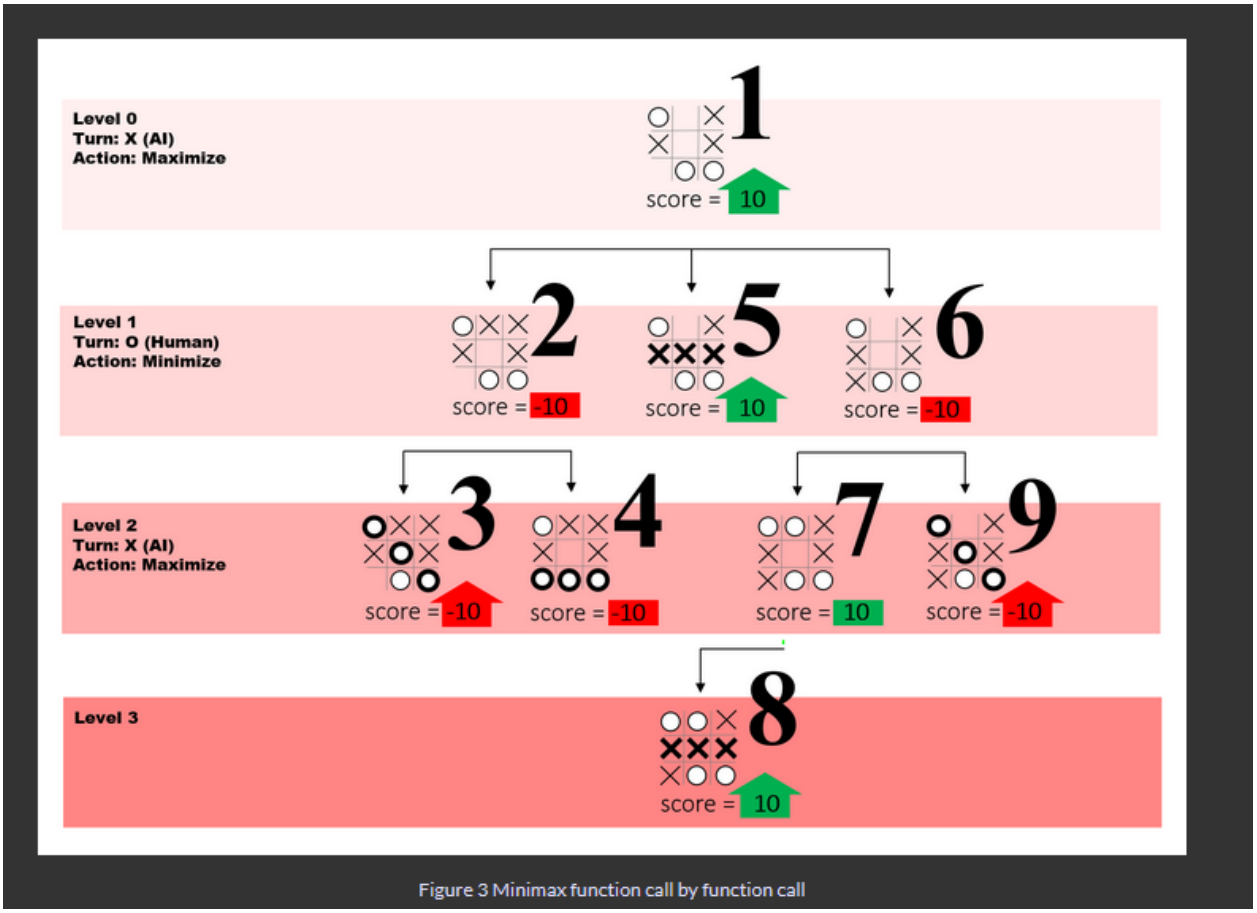
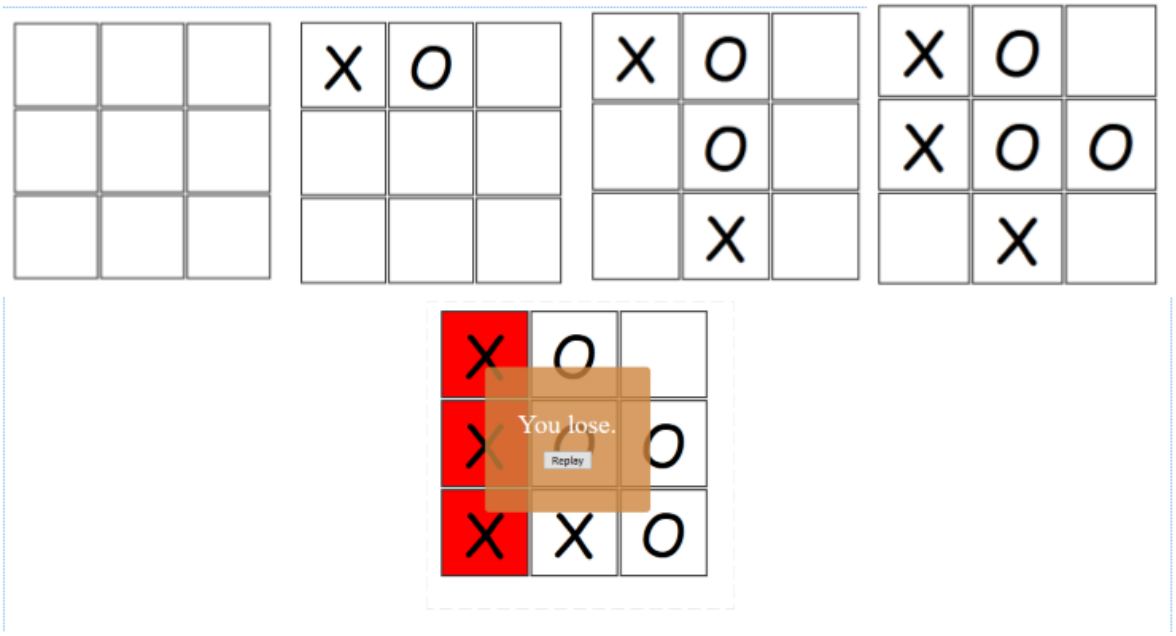


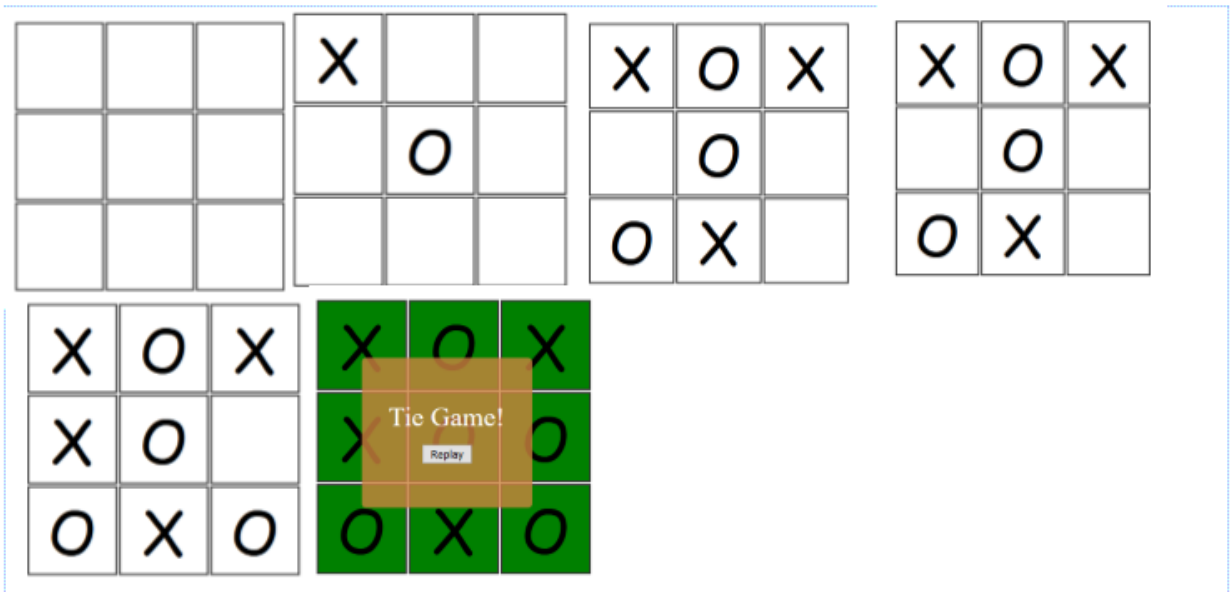
Figure 3 Minimax function call by function call

همان طور که در تصویر بالا قابل مشاهده است، ما در هر راس مشخص میکنیم که نوبت انتخاب با کامپیوتر است و یا با انسان و سپس بر اساس آن سعی میکنیم بهترین امتیاز را برای خودمان در لایه بعدی برداریم و امتیاز برد ها در لایه های بعدی هم توجه به شخص انتخاب کننده در لایه بعدی و برد های شاخه بعدی آن دارد. به طور خلاصه یک تابع recursive پیاده سازی میکنیم که شرط خاتمه نود های آن این است که یا یکی از بازیکن ها "برنده" شده باشد و یا بازی " Tie game " شده باشد.

حال به سراغ بازی خود میرویم و سعی میکنیم با الگوریتم غیرقابل شکست ناپذیر خود بازی کنیم:



در بازی اول انسان به کامپیوتر باخت.



در این بازی مساوی شده ایم.

منابع و مراجع

در توضیح الگوریتم minimax برای بازی tic-tac-toe از این وبسایت ها استفاده شده است:

https://www.freecodecamp.org/news/how-to-make-your-tic-tac-toe-game-unbeatable-by-using-the-minimax-algorithm-9d690bad4b37/

برای پیاده سازی الگوریتم کل برنامه از یوتوب ویدئو های زیر استفاده شده است:

https://www.youtube.com/watch?v=Y-GkMjUZsmM

https://www.youtube.com/watch?v=P2TcQ3h0ipQ&t=435s

برای فهم عمیق تر minimax الگوریتم از جزوه استاد و ویکی پیدا و یوتوب ویدئو زیر استفاده شده است:

https://www.youtube.com/watch?v=l-hh51ncgDI

پیوست‌ها

سورس کد html

```html
<html>
<head>
  <meta charset="UTF-8">
  <title>Tic Tac Toe</title>
  <link rel="stylesheet" href="style.css">
</head>
<body>
  <table>
    <tr></tr>
    <tr></tr>
    <tr></tr>
  </table>
  <div class="endgame">
    <div class="text"></div>
    <button onClick="startGame()">Replay</button>
  </div>

  <script src="script.js"></script>
</body>

</html>
```

سورس کد css برای قالب صفحه

```css
td {
  border:  2px solid #333;
  height:  100px;
  width:  100px;
  text-align:  center;
  vertical-align:  middle;
  font-family:  "Comic Sans MS", cursive, sans-serif;
  font-size:  70px;
  cursor: pointer;
}
table {
  position: absolute;
  left: 50%;
  margin-left: -155px;
  top: 50px;
}
.endgame {
  display: none;
  width: 200px;
  top: 120px;
  background-color: rgba(205,133,63, 0.8);
  position: absolute;
  left: 50%;
  margin-left: -100px;
  padding-top: 50px;
  padding-bottom: 50px;
  text-align: center;
  border-radius: 5px;
  color: white;
  font-size: 2em;}
```

سورس کد جاوا اسکریپت برنامه

```
// Phase 1: Basi setup
var origBoard;
const huPlayer = 'O';
const aiPlayer = 'X';

// All combination of winning
const winCombos = [
  [0, 1, 2],
  [3, 4, 5],
  [6, 7, 8],
  [0, 3, 6],
  [1, 4, 7],
  [2, 5, 8],
  [0, 4, 8],
  [6, 4, 2]
]
// list of all cells on the HTML page
const cells = document.querySelectorAll('.cell');
// forward declaration
startGame();

// Phase 2: efficient communication with the page
function startGame() {
  document.querySelector(".endgame").style.display = "none";
  // a fancy way of creating new board = [0,1,2,3,4,5,6,7,8]
  origBoard = Array.from(Array(9).keys());
  for (var i = 0; i < cells.length; i++) {
    cells[i].innerText = '';
    cells[i].style.removeProperty('background-color');
    // Click listener
    cells[i].addEventListener('click', turnClick, false);
  }
}
// The reason that we don't directly go to the Turn func and we need
```

```
    cells[i].addEventListener('click', turnClick, false);
  }
}
// The reason that we don't directly go to the Turn func and we need
// to implement turnClick funtion is we need to chck some stuff
function turnClick(square) {
  if (typeof origBoard[square.target.id] == 'number') {
    turn(square.target.id, huPlayer)
    if (!checkWin(origBoard, huPlayer) && !checkTie()) turn(bestSpot(), aiPlaye
  }
}

function turn(squareId, player) {
  origBoard[squareId] = player;
  // updating the cell of the table that we just clicked on it
  document.getElementById(squareId).innerText = player;
  let gameWon = checkWin(origBoard, player)
  if (gameWon) gameOver(gameWon)
}

function checkWin(board, player) {
  // we're gonna iterate winCombos and check if it's satisfied
  let plays = board.reduce((a, e, i) =>
    (e === player) ? a.concat(i) : a, []);
  let gameWon = null;
  for (let [index, win] of winCombos.entries()) {
    if (win.every(elem => plays.indexOf(elem) > -1)) {
      gameWon = {index: index, player: player};
      break;
    }
  }
  return gameWon;
}
```

```
// What will happen if somebody win the game and the game become over :
function gameOver(gameWon) {
  for (let index of winCombos[gameWon.index]) {
    document.getElementById(index).style.backgroundColor =
      gameWon.player == huPlayer ? "blue" : "red";
  }
  for (var i = 0; i < cells.length; i++) {
    cells[i].removeEventListener('click', turnClick, false);
  }
  declareWinner(gameWon.player == huPlayer ? "You win!" : "You lose.");
}

function declareWinner(who) {
  document.querySelector(".endgame").style.display = "block";
  document.querySelector(".endgame .text").innerText = who;
}

function emptySquares() {
  return origBoard.filter(s => typeof s == 'number');
}
// AI part of the code
function bestSpot() {
  return minimax(origBoard, aiPlayer).index;
}

function checkTie() {
  if (emptySquares().length == 0) {
    for (var i = 0; i < cells.length; i++) {
      cells[i].style.backgroundColor = "green";
      cells[i].removeEventListener('click', turnClick, false);
    }
    declareWinner("Tie Game!")
    return true;
  }
  return false;
```

```
    declareWinner("Tie Game!")
    return true;
  }
  return false;
}

function minimax(newBoard, player) {
  var availSpots = emptySquares();
  // termination conditino
  if (checkWin(newBoard, huPlayer)) {
    return {score: -10};
  } else if (checkWin(newBoard, aiPlayer)) {
    return {score: 10};
  } else if (availSpots.length === 0) {
    return {score: 0};
  }
  // for ALL empty spots we create a child and find their score
  var moves = [];
  for (var i = 0; i < availSpots.length; i++) {
    var move = {};
    move.index = newBoard[availSpots[i]];
    newBoard[availSpots[i]] = player;

    if (player == aiPlayer) {
      var result = minimax(newBoard, huPlayer);
      move.score = result.score;
    } else {
      var result = minimax(newBoard, aiPlayer);
      move.score = result.score;
    }

    newBoard[availSpots[i]] = move.index;

    moves.push(move);
```

```
      var result = minimax(newBoard, huPlayer);
      move.score = result.score;
    } else {
      var result = minimax(newBoard, aiPlayer);
      move.score = result.score;
    }

    newBoard[availSpots[i]] = move.index;

    moves.push(move);
  }
  // Picking the best child (best move)
  var bestMove;
  if(player === aiPlayer) {
    var bestScore = -10000;
    for(var i = 0; i < moves.length; i++) {
      if (moves[i].score > bestScore) {
        bestScore = moves[i].score;
        bestMove = i;
      }
    }
  } else {
    var bestScore = 10000;
    for(var i = 0; i < moves.length; i++) {
      if (moves[i].score < bestScore) {
        bestScore = moves[i].score;
        bestMove = i;
      }
    }
  }

  return moves[bestMove];
  // end of recursion
}
```