

Department of Electrical & Computer Engineering

North South University

**Tittle name : Javascript**

**Course name : Cse299 ( junior project )**

**Sec : 12**

**Name and Id:**

**1 . Pinky Biswas – 1621276042**

**2 . Dipu Chandra malo - 1712419042**

**3 . Khorshadul Islam – 1521333642**

**4 . Fahad Ahmed Pranto** – **1612607042**

**Submitted To** - **Mohammad Ashrafuzzaman Khan Sir**

Date : 24 – 12 – 2019

**ABSTRACT** :

We study in javascript language . This is totally new challenging task for us because before we don’t have any idea about this language . We basically worked on javascript language how to works in the game . First we select one easy game than in the game we implement callbacks function and successfully run the game . Than second task was how to minimax algorithm works in the game using by javascript language . Finally we search that time complexity between 3:3 and 5:5 tic tac toe game.

* **List of figure :**

**Figure1: Javascript ( how to works)**



**Figure2: 3 by 3 tic tac toe game**





Figure 3 : Minimax algorithm

**List Of Table:**

* **Introduction:** Everyone knows that JavaScript is eating the world. The problem is, there are lots of developers responsible for building software with JavaScript without a particularly deep knowledge of the language. It is easy to learn the basics and be productive, but newer programmers have a tendency to misuse some of JavaScript. Asynchronous JavaScript is particularly misunderstood. In this post, I will show you the most common ways to make asynchronous JavaScript calls and when to use each one. There are a lot of idiosyncrasies when using JavaScript in the browser, so for the purposes of this post, I will be discussing JavaScript on the server using Node.
* **Background :**
* [Callbacks (The Old Way)](https://developer.okta.com/blog/2019/01/16/history-and-future-of-async-javascript?fbclid=IwAR3VzltYe4WunNvs6iAcaHyn_CWqWLJabUa7h1JXYCfolrfAA_Em6QjroIE):

The setTimeout() function’s first parameter is a function to call when the timeout finishes. This is a simple example of a callback. The function passed is known as a “callback” function and the work being done asynchronously is the timer.

* [Promises (The Newer Way)](https://developer.okta.com/blog/2019/01/16/history-and-future-of-async-javascript?fbclid=IwAR3VzltYe4WunNvs6iAcaHyn_CWqWLJabUa7h1JXYCfolrfAA_Em6QjroIE)

Promises were introduced to fix a lot of the problems with using callbacks to do asynchronous operations. A promise is just that: a promise to let the calling code know when the asynchronous call has finished.

* [AsyncHYPERLINK "https://developer.okta.com/blog/2019/01/16/history-and-future-of-async-javascript?fbclid=IwAR3VzltYe4WunNvs6iAcaHyn\_CWqWLJabUa7h1JXYCfolrfAA\_Em6QjroIE"/Await (The Newest Way)](https://developer.okta.com/blog/2019/01/16/history-and-future-of-async-javascript?fbclid=IwAR3VzltYe4WunNvs6iAcaHyn_CWqWLJabUa7h1JXYCfolrfAA_Em6QjroIE):

The Async keyword in JavaScript adds a whole new level of cool to promises: the await keyword! This can make your code much more readable because the await keyword lets the code end up reading like traditional synchronous JavaScript.

**3. Methodology:**

* **3:3 Tic-Tac-Toe game Ai vs Hp:**

Tic-tac-toe Xs and Os is a [paper-and-pencil game](https://en.wikipedia.org/wiki/Paper-and-pencil_game) for two players, X and O, who take turns marking the spaces in a 3×3 grid. The player who succeeds in placing three of their marks in a horizontal, vertical, or diagonal row is the winner.

The following example game is won by the first player, X:



In tic tac toe game, implement javascript’s oldest way (callback ) and with out any error successfully run the game

Tic tac toe –creating unbeatable ai by using minimax algorithm.

* **5x5 Tic-tac-toe Ai vs human:**

 Build Ai vs human tic-tac-toe  5x5 where ai never lose with minimax algorithm. Compare this game with the previous one In terms of time complexity.

**4.Result and Discussion:**

* **Discussion :**

-Minimax is a recursive algorithm which is used to choose an optimal move for a player assuming that the opponent is also playing  optimally.

-its goal is to minimize the maximum loss. This algorithm sees a few steps ahead and puts itself in the shoes of its opponent. It keeps playing ahead until it reaches a terminal arrangement of the board resulting in a tie, a win, or a loss.

Once in a terminal state, the AI will assign an arbitrary positive score (+10) for a win, a negative score (-10) for a loss, or a neutral score (0) for a tie.

* Result :

Minimax algorithm that searches game trees. It assumes that the players take alternate moves. The time complexity of Minimax is O(b^m), where b is the number of legal moves at each point and m is the maximum depth of the tree.

Comparison:

For 3x3 board needs minimum 9! (factorial approximately 3,68,000) function calls to calculate best move for every iteration .

On the other hand, 5x5 board needs minimum 25! (factorial approximately 1.55\*10^25) function calls to calculate best move for every iteration . It just goes exponentially as the board size goes up.

**5 .Conclusion :**

JavaScript is a wonderful technology to use on the web. It is not *that* hard to learn and it is very versatile. It plays nicely with other web technologies — such as HTML and CSS . JavaScript allows us to build highly responsive user interfaces, prevent frustrating page reloads, and even fix support issues for CSS.

**References:**

**1.https://webplatform.github.io/docs/concepts/programming/the\_purpose\_of\_javascript/**

**2.** [**https://www.geeksforgeeks.org/minimax-algorithm-in-game-theory-set-1-introduction/**](https://www.geeksforgeeks.org/minimax-algorithm-in-game-theory-set-1-introduction/)

**3.** [**https://www.coursereport.com/blog/history-of-javascript**](https://www.coursereport.com/blog/history-of-javascript)

**Appendix :**

//defining minimax function

function minimax(newBoard, player) {

var availSpots = emptySquares(newBoard); //defining the indexes of the available spots in the board

if(checkWin(newBoard, player)) { //checking who wins

return {score: -10}; //if O wins we return -10

} else if (checkWin(newBoard, aiPlayer)) {

return {score: 10} // if X wins we return 10

} else if (availSpots.length === 0) {

return {score: 0} //tie, we return 0

}