**SYNOPSIS REPORT**

**on**

**GAME- TIC TAC TOE**

**Submitted by**

Ramya Mihir (Enroll No. 500067119)

Isha Mittal (Enroll. No. 500068884)

Shubhangi Gupta (Enroll. No. 500068875)

Sahil Verma (Enroll. No. 500067012)

**Under the guidance of**

Dhiviya Rose J.

Assistant Professor - Senior Scale

SCHOOL OF COMPUTER SCIENCE



**SCHOOL OF COMPUTER SCIENCE**

UNIVERSITY OF PETROLEUM & ENERGY STUDIES

Bidholi Campus, Energy Acres, Dehradun – 248007.

**August – 2020**

 **School of Computer Science**

**University of Petroleum & Energy Studies, Dehradun**

**Project Proposal Approval Form (2020)**

****I

**Minor**

**PROJECT TITLE:** Game- TIC TAC TOE

**ABSTRACT**

The game Tic Tac Toe uses the algorithm “Minimax”. The game will be prepared using the C language, where we will provide a user interface to input X and Os and it will also decide whether the game is a tie or whther player 1 won or player 2.

The Minimax algorithm is a type of backtracking algorithm that is generally used in decision making and for the game development theory, so that the player can find the optimal move, also keeping into mind that the opponent plays fair and optimally. This algorithm is widely used in two-player games like Tic-Tac-Toe, Chess, etc.

The schedule of the complete project will be from September 2020- December 2020, which will include the creation of the entire project along with its working.

**Keywords:** C language, Minimax Algorithm, Input from user- X & O, Display the output of who is the winner or whether the game is a draw, Requirement gathering, Testing

**INTRODUCTION**

Tic Tac Toe is a famously known game, which uses the Minimax Algorithm, so as to take the input from the user. This game is a two player game, where one player chooses cross (X) and the other player uses zero (0). In this game, each player seek an alternate turn to complete either a row, or a column or a diagonal with either three 0’s or three X’s that is drawn in the grid of nine squares.

Minimax algorithm which is used by this game, is a type of a backtracking algorithm that is used to make decisions and is also used in game theories to find the best optimal move for the player, supposing that the opponent player also plays optimally. This algorithm is widely used in the coding of games that posses only two players and not more than that.

The games that use this algorithm are: Tic-Tac-Toe, Chess, Manchala, etc.

In this, the two players are termed as Maximizer and Minimizer, wherein the Maximizer tries to get the highest score possible and the minimizer tends to get the lowest score possible.

Tic Tac Toe will be prepared using the C language, where the user will provide an input of X’s and O’s, then depending upon the inputs, the system will generate its empty spaces and if not, it will declare the result- tie/ winner, and the particular message will be displayed to the user.

The material will be gathered and the coding will be started. After the coding is done, the program will be tested and looked into errors if any, and will check whether the desired output is achieved or not, so depending upon the output, the changes will be done.

**PROBLEM STATEMENT**

Constructing a Tic-Tac-Toe game having Minimax algorithm in C language.

**LITERATURE REVIEW**

Saying that a game is solved usually indicates in general, that a property with regard to the outcome of the game has been determined. Even for two-player, zero-sum games with perfect information, at least three different types of definitions could be intended, which are ultra- weakly solved, weakly solved and strongly solved. Ultraweakly solved game indicates that at the start of the game, it is known what the outcome of the game would be with optimal play by both sides. It is not necessarily known how either player can achieve the optimal outcome. Weakly solved game indicates that the player need to be able to achieve a draw, in every game played. It is not necessary for player to win against a non-optimally playing opponent, when player is given a winning opportunity. Strongly solved game demands a strategy not just from the initial positions, but from all possible legal positions. Thus, against a non-optimally playing opponent, the strategy should be such that each mistake must be capitalized upon.

Some of the relevant references for more detail are:

**Brute-force Methods:** Brute-force methods have been important tools helpful in solving games. Many solving programs use basic brute-force methods such as α-β and their enhancements in some way or another. Two methods which have their application especially in solving games are the construction of databases by retrograde analysis and enhanced transposition-table methods.

**Retrograde analysis:** Retrograde analysis is a method in which for each position of some specific game or endgame the number of moves towards the best reachable goal is stored. For instance, in Chess, assuming perfect counter play, the number of moves to be played by the stronger side up to mate or conversion is stored. Checkers databases sometimes only contain an indication of won, drawn, or lost per position. A database is constructed by starting in terminal positions and then working backwards. Once database is constructed, perfect play is guaranteed: the stronger side chooses a move with the shortest distance-tomate and the weaker side opts for moves with the longest distance-to-mate. Perfect play by a computer in a position which is game-theoretically drawn or lost does not guarantee the best performance against imperfect opponents, as was demonstrated by Jansen. Nowadays the use of retrograde analysis is commonplace for the construction of endgame databases. It has deepened the understanding of such endgames considerably, and resulted in notions as max- to-mate, max-to-conversion, max-to-zeroing-move, and max-to-the- rule.

**Knowledge-based Methods:** -Next to brute-force methods it is often beneficial to incorporate knowledge-based methods in game-solving programs. Their main advantage is that it provides an appropriate move ordering or selection in the search trees. Some of the methods are:

Threat-space search and λ-search: Allis generalized the idea of threat-space search to a method called dependency-based search. Threat-space search investigates whether by a sequence of threats, to which the opponent at any time has only a limited set of replies, a win can be forced. Since the opponent effectively has no real choices, this search algorithm represents the application of single-agent search to two-player games. A recent successor of threat-space search, called λ-search, has been proposed by Thomsen. This method uses null moves combined with different orders of threat sequences, called λ-trees. Thomsen introduces λ1-moves, which threaten to end the game or reach a specified sub goal immediately, followed by λ2-moves threatening a winning λ1- sequence, and so on. The method behaves as a goal-directed searcher, with a favorable tree size relative to standard α- β trees. It can be combined with any search method to search the λ-trees. As a relevant

example Thomsen mentions proof-number search. A combination of null moves and proof numbers seems a promising method for solving Go endgames.

**OBJECTIVES**

To create a GUI based tic tac toe game with the help of Minimax Algorithm using C language.

Sub Objectives

* To provide an easy user interface to input X and Os.
* User should be able to move across the screen
* System should be able to preprocess the given input
* System should detect text regions present in the image
* System should retrieve text and display them to the user

**METHODOLOGY**

For this project, the technology being used is basic C language having Minimax Algorithm (backtracking algorithm) , wherein the player (user) inputs X and Os as per his/ her choice. The input is provided to the convolutional layer where the parameters are chosen, filters are applied with strides and padding is applied as well if required.

The class generates an output using required output functions.

The entire implementation of this project can be summarized into the following steps:

1. Import the libraries and load the dataset: All the modules needed for training our model are loaded. The library being used here is standard input output.

2. Preprocess the data: The input data has to undergo certain process in order to be displayed on screen.

3. Create the model: This step marks the creation of the game board layout. This layer consists of the convolutional and pooling layers and works with data that is represented as grid structures.

4. Train the model: To initiate the training of the model, player\_turn function is used to let the player make choice between X and O.

5. Evaluate the model: To evaluate the whole game there are Boolean functions used to

decide whether the game is a tie or player 1 won the game. Thus, displaying the desired output.

6. Testing the program- After the coding is done, the program will be checked- run and compiled and the errors will be resolved so that the desired output is generated.

**SYSTEM REQUIREMENTS**

Hardware:

* RAM: 4GB
* Disk Space: 4GB

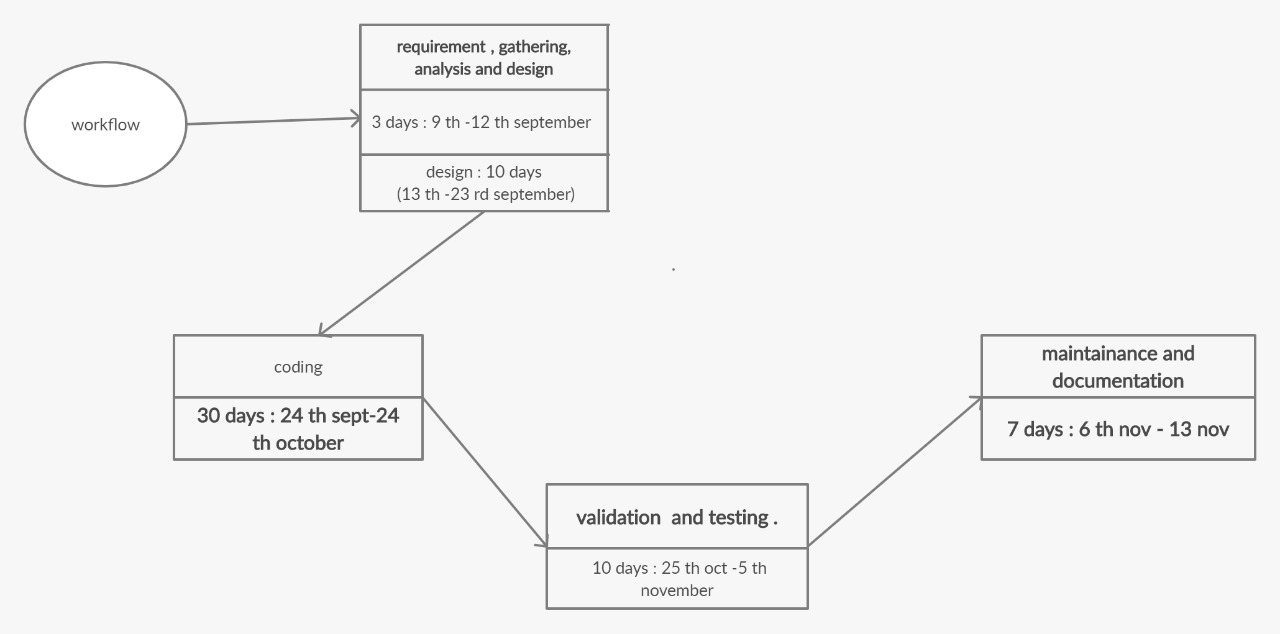
Software:

* Dev C++ or any online compiler using C language

Operating System:

* Windows 7,8,10, XP

**SCHEDULE**



**ALGORITHM**

* **Step-1:**In the first step, the algorithm generates the entire game-tree and apply the utility function to get the utility values for the terminal states.
* **Step 2:**Now, first we find the utilities value for the Maximizer, its initial value is -∞, so we will compare each value in terminal state with initial value of Maximizer and determines the higher nodes values.
* **Step 3**: In the next step, it’s a turn for minimizer, so it will compare all nodes value with +∞, and will find the 3rd  layer node values.
* **Step 4:**Now it’s a turn for Maximizer, and it will again choose the maximum of all nodes value and find the maximum value for the root node. In this game tree, there are only 4 layers, hence we reach immediately to the root node, but in real games, there will be more than 4 layers.

**PSEUDOCODE**

//find the best move on the board

function findbestmove(board):

bestmove = NULL

for each move in board :

if move is better than bestmove

bestmove = move //current move made by the user

return bestmove

//minimax function to find whether the move is better

function minimax(board, depth, maximizer): //to check whether current move made is better than the best move

if current board state is a terminal state :

return value of the board

if maximizer : //maximizer is the one who tries to score highest, and minimizer tries opposite, ie, gets the lowest score

bestvalue = -INFINITY

for each move in board :

value = minimax(board, depth+1, false)

bestvalue = max( bestvalue, value) //gives maximum of the values

return bestvalue

else : //for minimizer

bestvalue = +INFINITY

for each move in board :

value = minimax(board, depth+1, true)

bestvalue = min( bestvalue, value) //gives minimum of the values

return bestvalue

// to find the moves left

function movesleft(board):

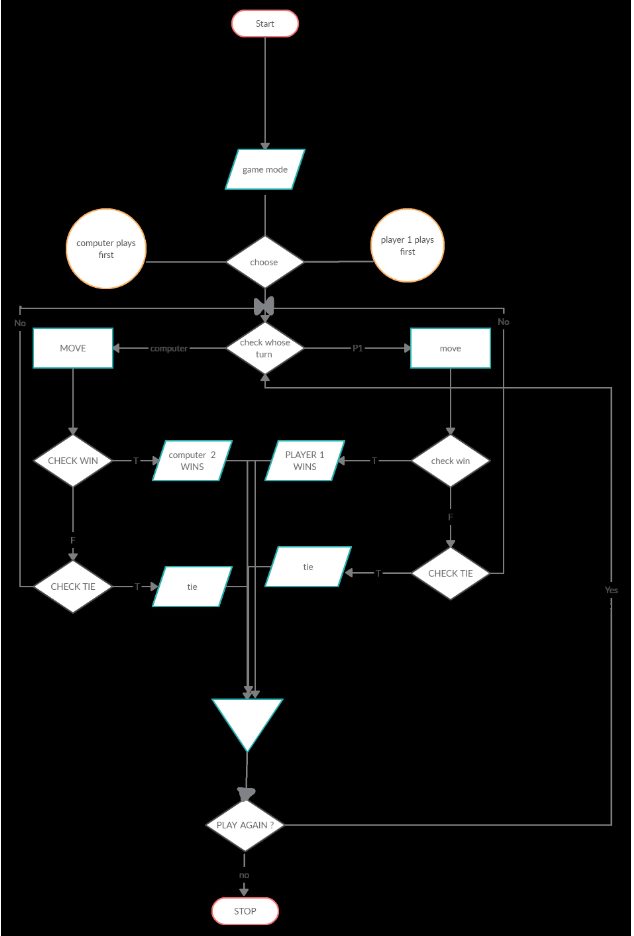
for each cell in board:

if current cell is empty:

return true

return false

**FLOWCHART**

****

**REFERENCES**

* <https://www.geeksforgeeks.org/minimax-algorithm-in-game-theory-set-1-introduction/#:~:text=Minimax%20is%20a%20kind%20of,%2C%20Mancala%2C%20Chess%2C%20etc>.
* <http://twjgame.blogspot.com/2015/09/literature-review_5.html?m=1>
* <https://shodhganga.inflibnet.ac.in/bitstream/10603/16330/6/09_chapter%202.pdf>
* <https://www.slideshare.net/UjjawalPoudel/mini-max-algorithm-proposal-document>
* <https://www.geeksforgeeks.org/minimax-algorithm-in-game-theory-set-3-tic-tac-toe-ai-finding-optimal-move/>
* <https://www.javatpoint.com/mini-max-algorithm-in-ai>

**Synopsis Draft verified by**

**Project Guide HOD**

**(Name & Sign) (Dept. of Systemics)**

**INSTRUCTIONS**

1. Min/Max. no. pages: **7** pages excluding Front Page.
2. The contents in the document should be provided in such a way that the **Verification of the Synopsis Report** should be available in the last page along with the contents and NOT in a separate page.
3. Type your content in this document itself with all Pages except the front page numbered. Bottom, Center, Times New Roman, Bold, Font size 10.
4. Margins: Left(1”), Right (0.75), Top & Bottom(0.6”).
5. Chapter Heading: Center Aligned, Bold, All Caps, Time New Roman, Font size 16
6. Main Heading: Heading under a chapter should be Left Aligned, Bold, Not underlined & not italicized, Times New Roman, Font size 14
7. Sub-Heading: Left Aligned, Bold, Not underlined & not italicized, Times New Roman, Font size 12
8. Content Format: Justified, Not Bold & Underlined, Not Italicized, Time New Roman, Font size: 12, line spacing: 1.5