**GitHub Link:** <https://github.com/raghavsharma804/AIproject>

**Artificial Intelligence Project Report**

**BACHELOR OF TECHNOLOGY**

**in**

**COMPUTER SCIENCE AND ENGINEERING**

**Raghav Sharma 11813297 A16**

**Agrima Agnihotri 11902957 A17**

**Aakarshit Shukla 11801782 A18**

**Section: K18AW**



**School of Computer Science and Engineering**

Lovely Professional University

Phagwara, Punjab (India)

TABLE OF CONTENT

1. Introduction

1.1 Purpose

1.2 Product Scope

2. Overall Description

2.1 Product Functions

2.2 User classes

2.3 User interface

3. Algorithm Used- Minimax

4. How minimax is used?

4. Coding

**INTRODUCTION**

**Purpose:** The main purpose of this project is to build a game based on AI. Game selected is Tic Tac Toe. We basically want to teach the program how to play the game so that it makes moves that are logical, in a way that a human think. The program will be built on py3. The IDE chosen is Jupyter Notebook.

**Product Scope:** This will be a simple AI based Tic Tac Toe game that involves console based output. We put together an algorithm that works the same way a human brain does.

**OVERALL DESCRIPTION**

**User classes and Product Functions:**

The program is built up of total of 10 functions which are written in the code mentioned at the last of this report. Each one of the functions is described in the code itself so that there is a fair amount of readability ease.

The main idea of the program goes this way:

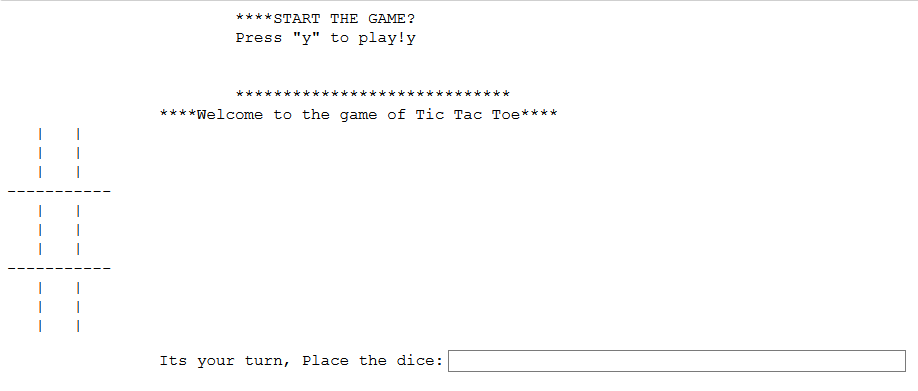
1. There is base of interface for the game. 9 columns.
2. One function that prints the interface of the game on the output screen.
3. A function check if the board is full
4. A function check if a particular space on the base is available.
5. A function puts the dice on the space
6. A function defines all the winning conditions.
7. One function takes input from the player and use above mentioned function to check if all conditions are valid.
8. One function teaches the program how to take decisions.
9. One random function for CPU to take decision

**User Interface:**

The program has set the symbols for human and cpu. Alphabets ‘X’ is human and ‘O’ is CPU. The function base() defines the empty spaces in the interface and the function baseInterface() prints the base every time a move is completely executed.

The player can move their dice by the use of numpad; 1 is the upper left most corner in the board and 9 is the lower right most corner of the board. The function check if only integer value is being entered .

The interface is actually on the py3 console itself and not on a GUI.

**Interface:**

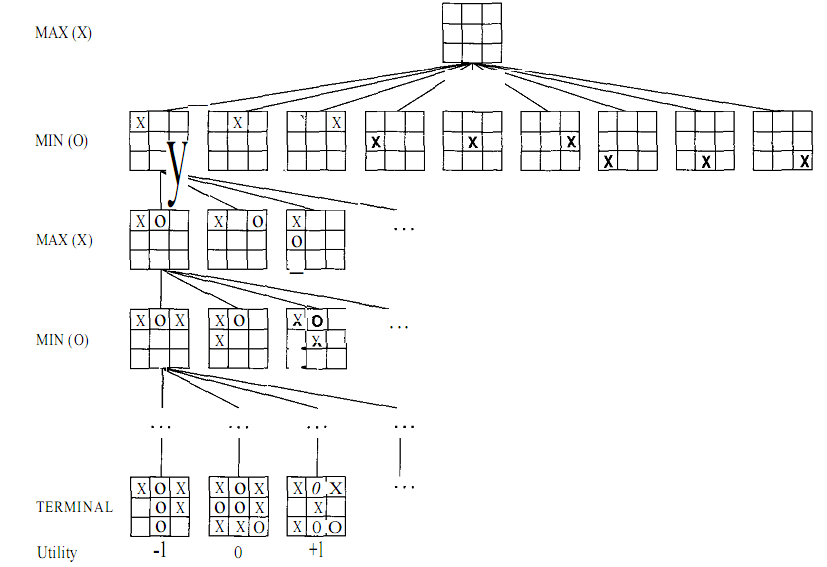
**ALGORITHM USED:**

1. **Minimax algorithm:**

It is mainly used in games like chess, checkers, tic tac toe etc.

A game is defined as a kind of search problem with the following functions:

* An initial state, which is the base position and an indication of whose chance it is.
* A set of operators or functions, which declares the valid moves a player can make.
* A test function which checks if the game is over.
* A final function or utility function which provides us a number value for outcome of the game, win, loss or tie.



**HOW IS MINIMAX ALGORITHM USED IN THIS PROGRAM?**

In the cpu() function, we teach the program how to make the best decision for itself so that either it can win or stop us from winning and lead to a tie game.

**def** cpu():

*#creating a list that will contain all the possible moves that the cpu can have*

pm = [a **for** a, letter **in** enumerate(base) **if** letter == ' ' **and** a != 0]

*#enumerate will give me all the combinations of letters and its position on the base*

*#This is a for loop, returning the letter for which it is blank and it is not zero.*

*#pm will be a list that helps the cpu to decide what are the various spaces it can move to.*

chance = 0 *# this is default variable; if at the end it is not changed then we print that is TIE GAME*

*#create a copy of the base interface and check for all the possible moves*

**for** let **in** ['O', 'X']:*#checking who is going to win first for human then cpu*

**for** i **in** pm:

baseCopy = base[:] *#base copy created using slicing of list.*

baseCopy[i] = let *#place symbol in the base copy*

**if** win(baseCopy, let): *#now checking if that position will win the game*

chance = i *#change the chance variable to this winning letter position*

**return** chance

*#if after checking that in the next step nobody can win*

*#check if open corners*

corner = []

**for** i **in** pm: *#if any corners are available then we add them to the list corner*

**if** i **in** [7,1,3,9]:

corner.append(i)

**if** len(corner) > 0:*#if multiple corners are available then select any one at random*

chance = ran(corner) *#use of random function*

**return** chance

*#check if center is open*

**if** 5 **in** pm:

chance = 5

**return** chance

*#check if any other space is open*

edge = []

**for** i **in** pm: *#if space are available then we add them to the list a*

**if** i **in** [2,4,6,8]:

edge.append(i)

**if** len(edge) > 0:*#If multiple edges are available then select one at random*

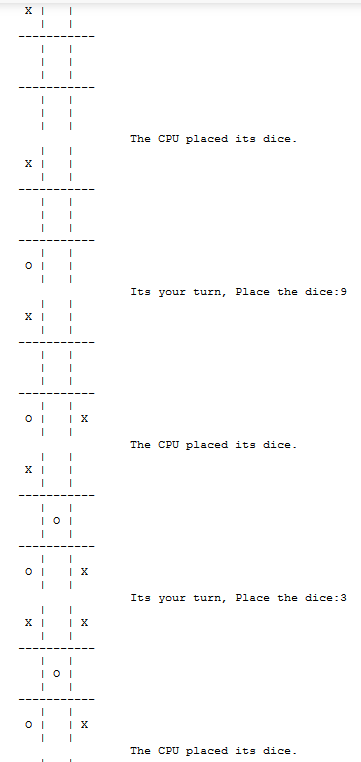
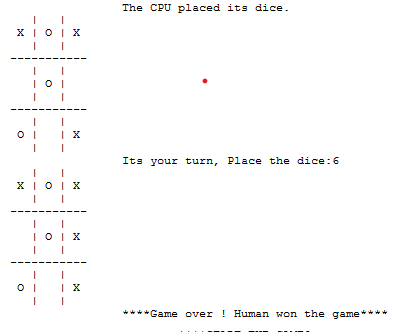
chance = ran(edge)

**return** chance

As in the function, list of possible moves is created and every time it is checked for different sets of move and then it decides to move to a space in the base via following algorithm:

1. If there is a win move, take it.
2. If in the next chance, human player is going to win, move to that space in the board
3. If above mentioned moves are not valid then move to one of the corners.
4. Take the center position.
5. Take one of the edges.
6. If no possible move is left then game over.

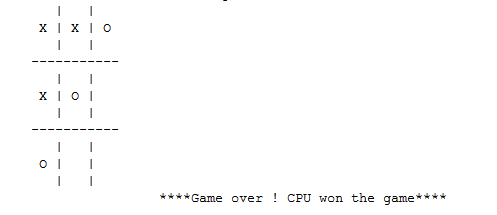
**TEST CASES**

Test case 1: Human wins

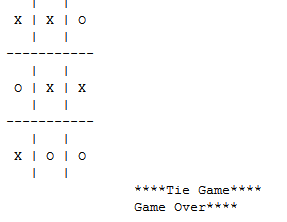
CPU places dice at one random corner

CPU places dice so that human cant win in next turn

Test case 2: CPU wins



Test case 3: Tie game



[Jupyter Notebook](http://localhost:8888/tree?token=0f68f088d873fb80c92dc911626a5e9b9870e5130a08b0ee)**FULL CODE FOR TIC TAC TOE (py3 )**

The coding is done in such a way that before executing a function I’ve already defined everything used with inside comments in each line of code and the reason for doing to for reading ease.

[Jupyter Notebook](http://localhost:8888/tree?token=0f68f088d873fb80c92dc911626a5e9b9870e5130a08b0ee)

**Coding done by: Raghav Sharma K18AW 16**

**Simple Artificial Intelligence Game- Tic Tac Toe**

**Introduction**

In this project I am going got built a game of tic tac toe. In this python program, there will be two players; one human; you and other the cpu which we will train to play in such a way that it either try to win or try not to lose. This will be a function based python code in which I will be using different sub functions to call one main function. Lets start

**Creating a base of game**

First off, I will create a base on which the game will be played. this will be list based variable in which there are empty spaces " " ranging from 1 to 9

base = [' ' **for** i **in** range(10)]*#this is going to store our symbols*

**Check if space is free**

Next, I am going to create my first function that will be to check whether the base we create is empty or not. I will name this as used() this will take one argument and that will be the position of the dice on the base. It will return me a boolean value , if the position is filled on the base of not. True or False

**def** used(position):

**return** base[position] == ' '*#this will return a boolean value*

**Add the dice to position**

Now, I want to crete a function name symbol() that will take 2 arguments. symbol() will put our dice on the position. And those will be the 2 arguments; which is the dice and the postion.

**def** symbol(dice, position):*#this is going to store the posistion at which the dice is placed*

base[position] = dice

**Defining the winning conditions**

Now, I create a function called win() that will take 2 arguments. The purpose of this function is to whether the player that just moved its dice satisfy the winning condition or not. 2 arguments will be the position and the dice. there are total of 8 winning conditions.

**def** win(p,s): *#defines the winning conditions*

**return** (p[1] == s **and** p[2] == s **and** p[3] == s)**or**(p[1] == s **and** p[4] == s **and** p[7] == s)**or**(p[1] == s **and** p[5] == s **and** p[9] == s)**or**(p[2] == s **and** p[5] == s **and** p[8] == s)**or**(p[3] == s **and** p[6] == s **and** p[9] == s)**or**(p[4] == s **and** p[5] == s **and** p[6] == s)**or**(p[7] == s **and** p[5] == s **and** p[3] == s)**or**(p[7] == s **and** p[8] == s **and** p[9] == s)

**Interface of the game**

Now, we print the base interface for tha game. This function printInterface() will take the base list as argument that I created earlier in the first line

**def** printInterface(base):

print(' | |')

print(' ' + base[1] + ' | ' + base[2] + ' | ' + base[3])

print(' | |')

print('-----------')

print(' | |')

print(' ' + base[4] + ' | ' + base[5] + ' | ' + base[6])

print(' | |')

print('-----------')

print(' | |')

print(' ' + base[7] + ' | ' + base[8] + ' | ' + base[9])

print(' | |')

**Taking input from user**

Next up,I will create a function called human() that will take no arguments. This function will do 3 things: 1.Ask the user for input of position of their dice. 2.Validating the input from the user. 3.If valid input then add it to the baseInterface

This will use the following functions: 1.used() for checking the postion whether valid or not 2.symbol() for adding the dice to the position if valid.

**def** human():

takeinput = **True** *#*

**while** takeinput:

chance = input('**\t\t**Its your turn, Place the dice:')

**try**: *#for checking that user enters only an integer*

chance = int(chance)

**if** chance > 0 **and** chance < 10: *#check whether the input is within the range of 0 to 9*

**if** used(chance):

takeinput= **False** *#if user has entered valid input then we need not run the loop to check that*

symbol('X', chance) *#then we add the dice into the base*

**else**: *#if the space is used loop again*

print('**\t\t**This space has already been used, try again!')

**else**: *#if the integer is >9 or 0 ;invalid loop again*

print('**\t\t**Number is greater than 9 or 0. Try again!')

**except**: *#if the user does not enters a number*

print('**\t\t**Please type an integer only.')

**Teaching the CPU how to play**

Next, I will actually move to teaching the cpu how to play the game. "The real AI stuff". In this we will create a function called as cpu() taking no argument. The cpu will decide and try to make the best move.

**def** cpu():

*#creating a list that will contain all the possible moves that the cpu can have*

pm = [a **for** a, letter **in** enumerate(base) **if** letter == ' ' **and** a != 0]

*#enumerate will give me all the combinations of letters and its position on the base*

*#This is a for loop, returning the letter for which it is blank and it is not zero.*

*#pm will be a list that helps the cpu to decide what are the various spaces it can move to.*

chance = 0 *# this is default variable; if at the end it is not changed then we print that is TIE GAME*

*#create a copy of the base interface and check for all the possible moves*

**for** let **in** ['O', 'X']:*#checking who is going to win first for human then cpu*

**for** i **in** pm:

baseCopy = base[:] *#base copy created using slicing of list.*

baseCopy[i] = let *#place symbol in the base copy*

**if** win(baseCopy, let): *#now checking if that position will win the game*

chance = i *#change the chance variable to this winning letter position*

**return** chance

*#if after checking that in the next step nobody can win*

*#check if open corners*

corner = []

**for** i **in** pm: *#if any corners are available then we add them to the list corner*

**if** i **in** [7,1,3,9]:

corner.append(i)

**if** len(corner) > 0:*#if multiple corners are available then select any one at random*

chance = ran(corner) *#use of random function*

**return** chance

*#check if center is open*

**if** 5 **in** pm:

chance = 5

**return** chance

*#check if any other space is open*

edge = []

**for** i **in** pm: *#if space are available then we add them to the list a*

**if** i **in** [2,4,6,8]:

edge.append(i)

**if** len(edge) > 0:*#If multiple edges are available then select one at random*

chance = ran(edge)

**return** chance

**random function**

**def** ran(lst):*#it will help in selecting a random number for selecting a space*

**import** **random**

l = len(lst)

r = random.randrange(0,l)

**return** lst[r]

**Checking if all spaces are filled, and base is full.**

Now, I create a function called baseout() it will take 1 argument of the base. This will return a boolean value. if base is full, then True if not then False. simple as that

**def** baseout(base):

**if** base.count(' ') > 1:

**return** **False**

**else**:

**return** **True**

**creating the main function**

Now that every sub function is created I am ready to create my main function This will not take any arguments

**def** play():

print('**\t\t**\*\*\*\*Welcome to the game of Tic Tac Toe\*\*\*\*')

printInterface(base) *#start off by printing a blank baseInterface*

**while** **not**(baseout(base)): *#this loop will run until base is not out of spaces*

**if** **not**(win(base, 'O')): *#check if CPU has won the game and if not then continue*

human() *#run the human() and take inputs from the user*

printInterface(base) *#print the base again*

**else**: *#if CPU has won the game*

print('**\t\t**\*\*\*\*Game over ! CPU won the game\*\*\*\*')

**break**

**if** **not**(win(base, 'X')): *#check if human has won the game and if not continue*

chance = cpu() *#assign the cpu() to a variable*

**if** chance == 0: *#if there are nowhere to put the dice then end*

print('**\t\t**\*\*\*\*Tie Game\*\*\*\*')

**else**:

symbol('O', chance)

print('**\t\t**The CPU placed its dice.')

printInterface(base)

**else**: *#if Human has already won the game*

print('**\t\t**\*\*\*\*Game over ! Human won the game\*\*\*\*')

**break**

**if** baseout(base): *#if base is full, then game is Over.*

print('**\t\t**Game Over\*\*\*\*')

**Finishing up**

Now every function is complete and we need to start the game and prompt the user if he wishes to play again

**while** **True**: *#game loop*

restart= input('**\t\t\t**\*\*\*\*START THE GAME?**\n\t\t\t**Press "y" to play!') *#take input from user to play*

**if** restart.lower() == 'y': *#only accpets y as an answer otherwise quit*

base= [' ' **for** x **in** range(10)]

print('**\n\n\t\t\t**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*')

play()

**else**:

**break**

**Coding done by: Raghav Sharma K18AW 16**