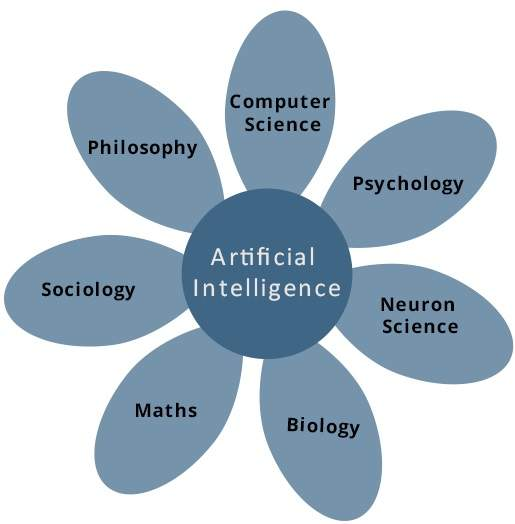
**EXPERIMENT-01**

**OBJECTIVE-** To study the concepts of Artificial Intelligence, Soft Computing and Semantic Web.

**ARTIFICIAL INTELLIGENCE**

**Artificial intelligence** is a science and technology based on disciplines such as Computer Science, Biology, Psychology, Linguistics, Mathematics, and Engineering.

It is a branch of computer science that aims to create intelligent machines. It has become an essential part of the technology industry.



Research associated with artificial intelligence is highly technical and specialized. The core problems of artificial intelligence include programming computers for certain traits such as:

1.Knowledge engineering is a core part of AI research. Machines can often act and react like humans only if they have abundant information relating to the world. Artificial intelligence must have access to objects, categories, properties and relations between all of them to implement knowledge engineering. Initiating common sense, reasoning and problem-solving power in machines is a difficult and tedious task.

2. Reasoning- To reason is to draw [inferences](https://www.merriam-webster.com/dictionary/inferences) appropriate to the situation. Inferences are classified as either [deductive](https://www.britannica.com/topic/deduction-reason) or [inductive](https://www.britannica.com/topic/induction-reason). An example of the former is, “Fred must be in either the museum or the café. He is not in the café; therefore he is in the museum,” and of the latter, “Previous accidents of this sort were caused by instrument failure; therefore this accident was caused by instrument failure.” The most significant difference between these forms of reasoning is that in the deductive case the truth of the [premises](https://www.merriam-webster.com/dictionary/premises) guarantees the truth of the conclusion, whereas in the inductive case the truth of the [premise](https://www.merriam-webster.com/dictionary/premise) lends support to the conclusion without giving absolute [assurance](https://www.merriam-webster.com/dictionary/assurance). Inductive reasoning is common in [science](https://www.britannica.com/science/science), where data are collected and tentative models are developed to describe and predict future behaviour—until the appearance of anomalous data forces the model to be revised. Deductive reasoning is common in [mathematics](https://www.britannica.com/science/mathematics) and [logic](https://www.britannica.com/topic/logic), where elaborate structures of irrefutable theorems are built up from a small set of basic axioms and rules.

3. Problem solving- Problem solving, particularly in artificial intelligence, may be characterized as a systematic search through a range of possible actions in order to reach some predefined goal or solution. Problem-solving methods divide into special purpose and general purpose. One general-purpose technique used in AI is means-end analysis—a step-by-step, or [incremental](https://www.merriam-webster.com/dictionary/incremental), reduction of the difference between the current state and the final goal. The program selects actions from a list of means—in the case of a simple robot this might consist of PICKUP, PUTDOWN, MOVEFORWARD, MOVEBACK, MOVELEFT, and MOVERIGHT—until the goal is reached.

4. Perception- In perception the [environment](https://www.merriam-webster.com/dictionary/environment) is scanned by means of various sensory organs, real or artificial, and the scene is decomposed into separate objects in various spatial relationships. At present, artificial perception is sufficiently well advanced to enable optical sensors to identify individuals, [autonomous](https://www.merriam-webster.com/dictionary/autonomous) vehicles to drive at moderate speeds on the open road, and robots to roam through buildings collecting empty soda cans. One of the earliest systems to [integrate](https://www.merriam-webster.com/dictionary/integrate) perception and action was FREDDY, a stationary robot with a moving television eye and a pincer hand.

5. Learning- The simplest is learning by trial and error. For example, a simple [computer](https://www.britannica.com/technology/computer) program for solving mate-in-one [chess](https://www.britannica.com/topic/chess) problems might try moves at random until mate is found. The program might then store the solution with the position so that the next time the computer encountered the same position it would recall the solution. This simple memorizing of individual items and procedures—known as rote learning—is relatively easy to [implement](https://www.merriam-webster.com/dictionary/implement)ation a computer. More challenging is the problem of [implementing](https://www.merriam-webster.com/dictionary/implementing) what is called [generalization](https://www.britannica.com/topic/generalization). Generalization involves applying past experience to [analogous](https://www.merriam-webster.com/dictionary/analogous) new situations. For example, a program that learns the past tense of regular English verbs by rote will not be able to produce the past tense of a word such as jump unless it previously had been presented with jumped, whereas a program that is able to generalize can learn the “add ed” rule and so form the past tense of jump based on experience with similar verbs.

6.Planning

**Machine learning** is also a core part of AI. Learning without any kind of supervision requires an ability to identify patterns in streams of inputs, whereas learning with adequate supervision involves classification and numerical regressions. Classification determines the category an object belongs to and regression deals with obtaining a set of numerical input or output examples, thereby discovering functions enabling the generation of suitable outputs from respective inputs. Mathematical analysis of machine learning algorithms and their performance is a well-defined branch of theoretical computer science often referred to as computational learning theory.

**Goals of AI**

To Create Expert Systems − The systems which exhibit intelligent behavior, learn, demonstrate, explain, and advice its users.

To Implement Human Intelligence in Machines − Creating systems that understand, think, learn, and behave like humans.

**AI Technique** is a manner to organize and use the knowledge efficiently in such a way that −

1. It should be perceivable by the people who provide it.

2. It should be easily modifiable to correct errors.

3. It should be useful in many situations though it is incomplete or inaccurate.

**Applications of AI**

**Gaming −** AI plays crucial role in strategic games such as chess, poker, tic-tac-toe, etc., where machine can think of large number of possible positions based on heuristic knowledge.

**Speech Recognition** − Some intelligent systems are capable of hearing and comprehending the language in terms of sentences and their meanings while a human talks to it. It can handle different accents, slang words, noise in the background, change in human’s noise due to cold, etc.

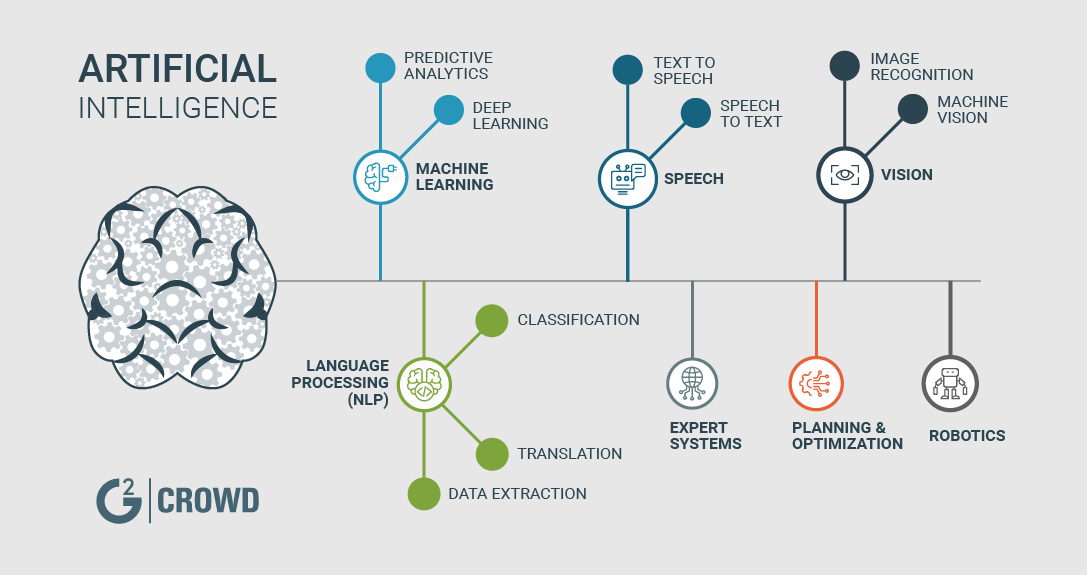
**Handwriting Recognition** − The handwriting recognition software reads the text written on paper by a pen or on screen by a stylus. It can recognize the shapes of the letters and convert it into editable text.

**Intelligent Robots** − Robots are able to perform the tasks given by a human. They have sensors to detect physical data from the real world such as light, heat, temperature, movement, sound, bump, and pressure. They have efficient processors, multiple sensors and huge memory, to exhibit intelligence. In addition, they are capable of learning from their mistakes and they can adapt to the new environment.

**WORKING OF AI**

AI works by combining large amounts of data with fast, iterative processing and intelligent algorithms, allowing the software to learn automatically from patterns or features in the data. AI is a broad field of study that includes many theories, methods and technologies, as well as the following major subfields:

* [**Machine learning**](https://www.sas.com/en_in/insights/analytics/machine-learning.html)automates analytical model building. It uses methods from neural networks, statistics, operations research and physics to find hidden insights in data without explicitly being programmed for where to look or what to conclude.
* **A neural network** is a type of machine learning that is made up of interconnected units (like neurons) that processes information by responding to external inputs, relaying information between each unit. The process requires multiple passes at the data to find connections and derive meaning from undefined data.
* [**Deep learning**](https://www.sas.com/en_in/insights/analytics/deep-learning.html) uses huge neural networks with many layers of processing units, taking advantage of advances in computing power and improved training techniques to learn complex patterns in large amounts of data. Common applications include image and speech recognition.
* [**Cognitive computing**](https://www.sas.com/en_in/insights/articles/big-data/executives-guide-to-cognitive-computing.html) is a subfield of AI that strives for a natural, human-like interaction with machines. Using AI and cognitive computing, the ultimate goal is for a machine to simulate human processes through the ability to interpret images and speech – and then speak coherently in response.
* **Computer vision** relies on pattern recognition and deep learning to recognize what’s in a picture or video. When machines can process, analyze and understand images, they can capture images or videos in real time and interpret their surroundings.
* **Natural language processing**(NLP) is the ability of computers to analyze, understand and generate human language, including speech. The next stage of NLP is natural language interaction, which allows humans to communicate with computers using normal, everyday language to perform tasks.



**SOFT COMPUTING**

Soft computing encompasses a set of computational techniques and algorithms that are used to deal with complex systems. Soft computing exploits the given tolerance for imprecision, partial truth, and uncertainty for a particular problem.

Soft computing is an emerging approach to computing which parallel the remarkable ability of the human mind to reason and learn in a environment of uncertainty and imprecision.

It is a field of computer science which is characterized by the use of inexact solutions to computationally hard tasks such as the NP-complete problems.

**PRINCIPLE'S**

* Neural Network(NN)
* Fuzzy Logic(FL)
* Genetic Algorithm(GA)

**GOALS**

* The main Goal of soft computing is to develop intelligent machines to provide solutions to real world problems, which are hard or difficult to model mathematically.
* To exploit the tolerance for approximation, Uncertainty, Imprecision and Partial Truth in order to achieve close resemblance with human like decision making.

**Neural Network(NN)** is a highly interconnected network of a large number of processing elements called neurons in an architecture inspired by the brain.

ANNs are composed of multiple nodes, which imitate biological neurons of human brain. The

**EXPERIMENT-02**

**OBJECTIVE-** To implement the concepts of Breadth First Search(BFS) and Depth First Search(DFS).

**THEORY OF BFS :-**

[Breadth First Traversal (or Search)](http://en.wikipedia.org/wiki/Breadth-first_search) for a graph is similar to Breadth First Traversal of a tree . The only catch here is, unlike trees, graphs may contain cycles, so we may come to the same node again. To avoid processing a node more than once, we use a boolean visited array. For simplicity, it is assumed that all vertices are reachable from the starting vertex.  
For example, in the following graph, we start traversal from vertex 2. When we come to vertex 0, we look for all adjacent vertices of it. 2 is also an adjacent vertex of 0. If we don’t mark visited vertices, then 2 will be processed again and it will become a non-terminating process. A Breadth First Traversal of the following graph is 2, 0, 3, 1.

[](https://cdncontribute.geeksforgeeks.org/wp-content/uploads/bfs-5.png)

**ALGORITHM OF BFS :-**

1. Start from a vertex **S**. Let this vertex be at, what is called…. “Level 0”.
2. Find all the other vertices that are immediately accessible from this starting vertex **S**, i.e., they are only a single edge away (the adjacent vertices).
3. Mark these adjacent vertices to be at “Level 1”.
4. There will be a challenge that you might be coming back to the same vertex due to a loop or a ring in the graph. If this happens your BFS will take **∞** time. So, you will go only to those vertices who do not have their “Level” set to some value.
5. Mark which is the parent vertex of the current vertex you’re at, i.e., the vertex from which you accessed the current vertex. Do this for all the vertices at Level 1.

**THEORY OF DFS :-**

The DFS algorithm is a recursive algorithm that uses the idea of backtracking. It involves exhaustive searches of all the nodes by going ahead, if possible, else by backtracking.

Here, the word backtrack means that when you are moving forward and there are no more nodes along the current path, you move backwards on the same path to find nodes to traverse. All the nodes will be visited on the current path till all the unvisited nodes have been traversed after which the next path will be selected.

This recursive nature of DFS can be implemented using stacks. The basic idea is as follows:  
Pick a starting node and push all its adjacent nodes into a stack.  
Pop a node from stack to select the next node to visit and push all its adjacent nodes into a stack.  
Repeat this process until the stack is empty. However, ensure that the nodes that are visited are marked. This will prevent you from visiting the same node more than once. If you do not mark the nodes that are visited and you visit the same node more than once, you may end up in an infinite loop.

**ALGORITHM OF DFS :-**

1. **Step 1**: Push the root node in the Stack.
2. **Step 2**: Loop until stack is empty.
3. **Step 3**: Peek the node of the stack.
4. **Step 4**: If the node has unvisited child nodes, get the unvisited child node, mark it as traversed and push it on stack.
5. **Step 5**: If the node does not have any unvisited child nodes, pop the node from the stack.

**CODE IMPLEMENTATION OF DFS:-**

#include<iostream>

#include<conio.h>

**EXPERIMENT-03**

**OBJECTIVE-** To understand the concept of Tic-Tac-Toe Game implementation using Minimax algorithm.

**THEORY OF MINIMAX ALGORITHM:-**

Minimax is a kind of [backtracking](https://www.geeksforgeeks.org/tag/backtracking/) algorithm that is used in decision making and game theory to find the optimal move for a player, assuming that your opponent also plays optimally. It is widely used in two player turn based games such as Tic-Tac-Toe, Backgamon, Mancala, Chess, etc.

In Minimax the two players are called maximizer and minimizer. The **maximizer** tries to get the highest score possible while the **minimizer** tries to get the lowest score possible while minimizer tries to do opposite.

Every board state has a value associated with it. In a given state if the maximizer has upper hand then, the score of the board will tend to be some positive value. If the minimizer has the upper hand in that board state then it will tend to be some negative value. The values of the board are calculated by some heuristics which are unique for every type of game.

**ALGORITHM FOR TIC-TAC-TOE:-**

1. **Computer v/s Human**

The program will consider all possible scenarios and makes the most optimal move. It uses the Minimax Algorithm that plays a perfect game.

**Finding the Best Move :**

We shall be introducing a new function called **findBestMove()**. This function evaluates all the available moves using **minimax()** and then returns the best move the maximizer can make. The pseudocode is as follows :

**function**findBestMove(board):

bestMove = NULL

**for each** move in board :

if current move is better than bestMove

bestMove = current move

**return**bestMove

if(game\_over()) {

find\_winner();

display\_result();

set\_game\_statistic();

reset\_state();

reset\_board();

display\_intro();

}

select\_game\_type();

}

}

return 0;

}

// selects game type

void select\_game\_type() {

std::cout << " 1 - play a game against the computer" << std::endl;

std::cout << " 2 - continue with the same player" << std::endl;

std::cout << " 3 - play against another player" << std::endl;

std::cout << " 4 - display game statistic" << std::endl;

std::cout << " 5 - quit the program" << std::endl;

std::cout << "\nselection: ";

int choice;

std::cin >> choice;

if(!std::cin.good()) {

std::cout << "please notice that you can only enter integers for the selection" << std::endl;

update\_screen();

}

switch(choice) {

case 1:

game\_type = "human vs computer";

break;

case 2:

if(game\_type == "") {

update\_screen();

}

break;

case 3:

game\_type = "human vs human";

break;

case 4:

display\_game\_statistic();

update\_screen();

break;

case 5:

state = QUIT;

break;

default:

std::cout << "Invalid Selection." << std::endl;

update\_screen();

}

if(choice > 0 && choice < 5) {

if(prev\_game\_type != "" && game\_type != prev\_game\_type) {

reset\_game\_statistic();

reset\_player\_name();

get\_player\_name();

get\_player\_symbol();

}

if(game\_type.length() > 0) {

prev\_game\_type = game\_type;

}

}

}

// gets current player name

void get\_player\_name() {

std::cin.sync();

if(game\_type == "human vs computer") {

std::cout << "\nplease enter your name: ";

std::getline(std::cin, player1.name);

if(player1.name.length() == 0) {

get\_player\_name();

}

player2.name = "the computer";

} else if(game\_type == "human vs human") {

while(player1.name.length() == 0) {

std::cout << "\nplayer1 please enter your name: ";

std::getline(std::cin, player1.name);

}

while(player2.name.length() == 0) {

std::cout << "player2 please enter your name: ";

std::getline(std::cin, player2.name);

}

}

}

void reset\_player\_name() {

player1.name.erase();

player2.name.erase();

}

// gets symbol for the current player

void get\_player\_symbol() {

if(game\_type == "human vs computer") {

int selection = rand() % 2;

if(selection == 0) {

rand() % 2 == 0 ? player2.symbol = 'X' : player2.symbol = 'O';

cSymbol = player2.symbol;

player2.selected = 1;

std::cout << player2.name << " will play \'" << player2.symbol << "\'" << std::endl;

} else if(selection == 1) {

std::cout << player1.name << " please enter your symbol (X, O): ";

std::cin >> player1.symbol;

player1.symbol = toupper(player1.symbol);

cSymbol = player1.symbol;

player1.selected = 1;

}

} else if(game\_type == "human vs human") {

int sel = rand() % 2;

std::string player\_name = "";

if(sel == 0) {

player\_name = player1.name;

player1.selected = 1;

} else if(sel == 1) {

player\_name = player2.name;

player2.selected = 1;

}

std::cout << "\n" << player\_name << " please enter your symbol (X, O): ";

if(sel == 0) {

std::cin >> player1.symbol;

player1.symbol = toupper(player1.symbol);

cSymbol = player1.symbol;

} else {

std::cin >> player2.symbol;

player2.symbol = toupper(player2.symbol);

cSymbol = player2.symbol;

}

}

if(!std::cin.good() || wrong\_symbol()) {

std::cout << "please notice that your symbol can only be X or O" << std::endl;

system("pause");

get\_player\_symbol();

}

if(!player2.selected) {

player1.symbol == 'X' ? player2.symbol = 'O' : player2.symbol = 'X';

player1.symbol == 'O' ? player2.symbol = 'X' : player2.symbol = 'O';

} else if(!player1.selected) {

player2.symbol == 'X' ? player1.symbol = 'O' : player1.symbol = 'X';

player2.symbol == 'O' ? player1.symbol = 'X' : player1.symbol = 'O';

}

state = PLAYING;

}

// gets move for the current player

void get\_move() {

std::cin.sync();

if(game\_type == "human vs human") {

if(player1.selected) {

std::cout << player1.name << " please enter your move (1 - 9): ";

std::cin >> player1.move;

nMove = player1.move;

cSymbol = player1.symbol;

player1.selected = 0;

player2.selected = 1;

current\_player = player1;

} else if(player2.selected) {

std::cout << player2.name << " please enter your move (1 - 9): ";

std::cin >> player2.move;

nMove = player2.move;

cSymbol = player2.symbol;

player1.selected = 1;

player2.selected = 0;

current\_player = player2;

}

} else if(game\_type == "human vs computer") {

if(player1.selected) {

std::cout << "\n" << player1.name << " please enter your move (1 - 9): ";

std::cin >> player1.move;

if(!std::cin.good()) {

std::cin.clear();

std::cin.sync();

}

nMove = player1.move;

cSymbol = player1.symbol;

current\_player = player1;

player1.selected = 0;

player2.selected = 1;

Sleep(1000);

} else if(player2.selected) {

player2.move = MiniMax(board, player2);

nMove = player2.move;

cSymbol = player2.symbol;

current\_player = player2;

player1.selected = 1;

player2.selected = 0;

reset\_state();

Sleep(1500);

}

}

verify\_move();

if(game\_over()) {

return;

}

}// set game statististic for current match

void set\_game\_statistic() {

if(state == START) {

player1.game\_win = 0;

player1.draw\_num = 0;

player1.win = 0;

player2.game\_win = 0;

player2.draw\_num = 0;

player2.win = 0;

} else if(state == XWIN || state == OWIN) {

if(player1.win) {

player1.game\_win++;

player1.win = 0;

} else if(player2.win) {

player2.game\_win++;

player2.win = 0;

}

} else if(player2.move != -1) {

board[player2.move - 1] = player2.symbol;

}

}

}

// displays outcome the game on the screen

void display\_result() {

if(player1.win) {

std::cout << player1.name << " has won the game!" << std::endl;

} else if(player2.win) {

std::cout << player2.name << " has won the game!" << std::endl;

} else if(player1.win == 0 && player2.win == 0) {

std::cout << "no winner, this game is a draw." << std::endl;

}

system("pause");

system("cls");

}

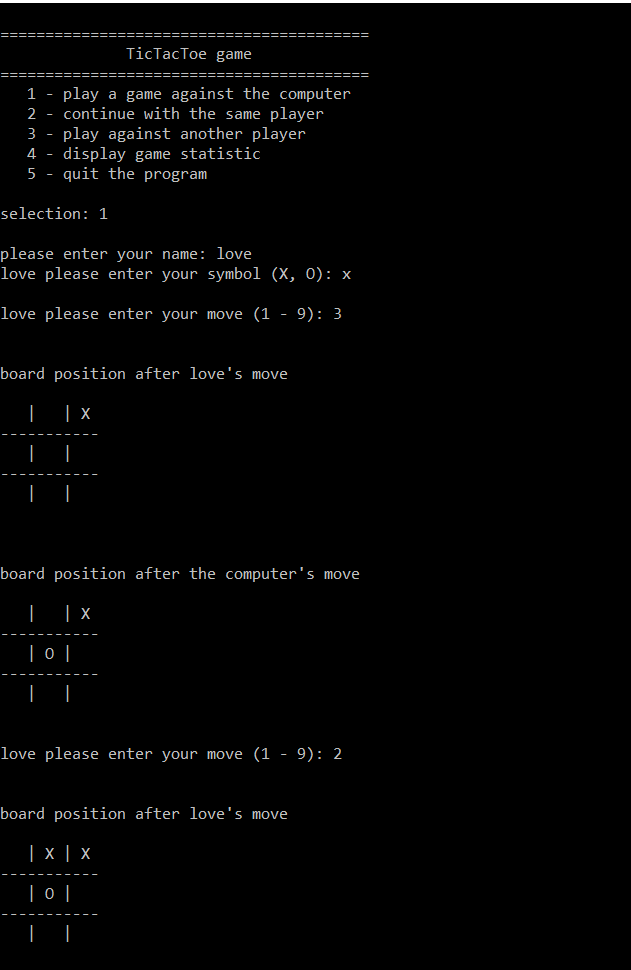
// make updates of the current game

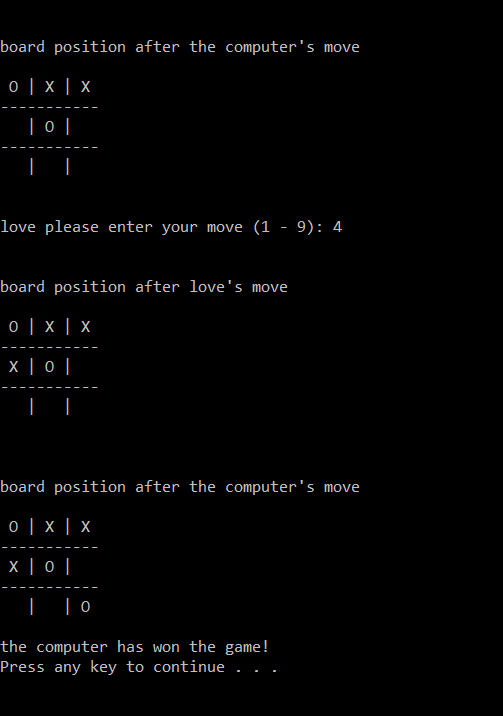
void update\_game() {

update\_board();

display\_game\_progress();

**OUTPUT OF TIC-TAC-TOE**



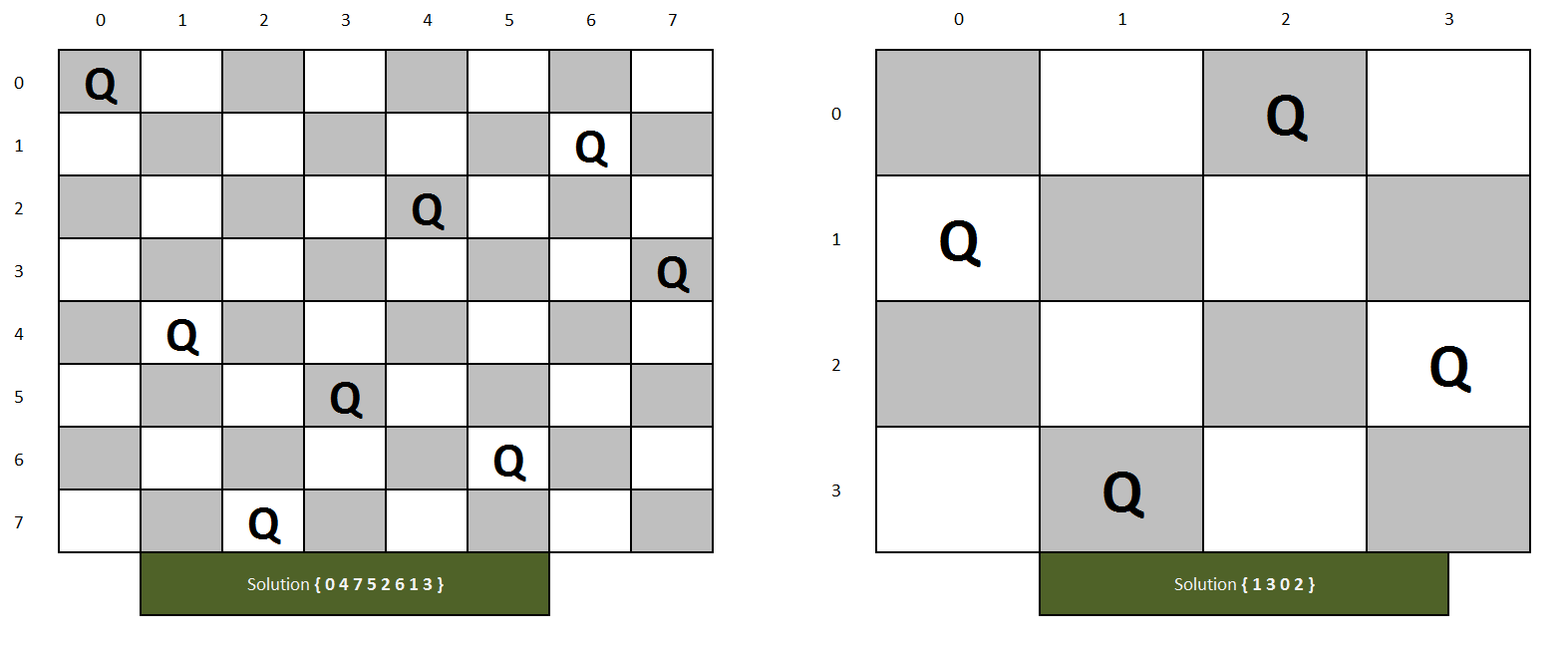


**EXPERIMENT-04**

**OBJECTIVE-** To implement the concept of Artificial Intelligence in N-Queen .

**THEORY OF N-Queen:-**

The n-queens problem consists in placing n non-attacking queens on an n-by-n chess board. A queen can attack another queen vertically, horizontally, or diagonally. E.g. placing a queen on a central square of the board blocks the row and column where it is placed, as well as the two diagonals (rising and falling) at whose intersection the queen was placed.



**ALGORITHM OF N-Queen :-**

1. Place the queens column wise, start from the left most column
2. If all queens are placed.
   1. return true and print the solution matrix.
3. Else
   1. Try all the rows in the current column.
   2. Check if queen can be placed here safely if yes mark the current cell in solution matrix as 1 and try to solve the rest of the problem recursively.
   3. If placing the queen in above step leads to the solution return true.
   4. If placing the queen in above step does not lead to the solution , BACKTRACK, mark the current cell in solution matrix as 0 and return false.
4. If all the rows are tried and nothing worked, return false and print NO SOLUTION.

**CODE IMPLEMENTATION OF N-Queen:-**

#include<iostream>

#include<cmath>

#include<stdlib.h>

using namespace std;

int i,k,j,n;

int x[30];

int count=0;

// for checking the non-attacking condition for placing the queen

bool place(int,int);

void printSolution(int x[])

{

static int count = 0;

count++;

cout<<"#"<<count ;

cout<<endl<<endl;;

for(i = 1; i<=n; i++)

{

for(j= 1; j<=n;j++)

{

if(x[i]== j)

{

cout<<" "<<1;

}

else

{

cout<<" "<< "\_\_\_";

}

}

cout<<endl<<endl;

}

cout<<endl;

}

// Applies the NQueens algorithm for each queen recursively using backtracking

void Nqueen(int k, int n){

// arguement1: k represents the queen number as well as the row to place queen in

// arguement2: n represents the total number of queens

for(int i=1;i<=n;i++)

{

// i is a column number where queen is placed

if(place(k,i))

{

// place queen k in column i

x[k]=i;

if(k==n)

{

cout << "\n solution found!\n";

for(i=1;i<=n;i++){

cout<<"Column number of queen "<<i<<": "<<x[i]<<endl;

}

cout<<"Placement is succesfull"<<endl;

cout<<endl;

printSolution(x);

exit(0);

return;

}

else

{

Nqueen(k+1,n);

}

}

}

}

// Checking if placing a queen at row k and column[row] position on the board is safe

bool place(int k,int i)

{

for(j=1;j<=k-1;j++)

{

// This condition checks if another queen lies on the same column or on the diagonal

cout<<"Checking Position for Queen "<<k<<" in column "<<i<<endl;

if(x[j]==i || abs(x[j]-i)==abs(j-k))

{

cout<<"Column "<<i<<" is not suitable"<<endl<<endl;

return false;

}

}

cout<<"COLUMN "<<i<<" IS SAFE FOR QUEEN:"<< k<<endl<<endl;

return true;

}

int main()

{

cout<<"Enter the number of queens :";

cin>>n;

//Initialize

for(i=0;i<n;i++)

{

x[i]=0;

}

Nqueen(1,n);

}

**EXPERIMENT-05**

**OBJECTIVE-** To understand the concepts of Prolog Programming.

**THEORY**

There are four kinds of term in Prolog: atoms,numbers, variables, and complex terms (or structures). Atoms and numbers are lumped together under the heading constants, and constants andvariables together make up the simple terms of Prolog.

**Atoms**

An atom is either:

1. A string of characters made up of upper-case letters, lower-case letters, digits, and the underscore character, that begins with a lower-case letter. Here are some examples: butch ,

big\_kahuna\_burger , listens2Music and playsAirGuitar .

2. An arbitrary sequence of characters enclosed in single quotes. For example ’ Vincent ’, ’ The Gimp ’, ’ Five\_Dollar\_Shake ’, ’ &^%&#@$ &\* ’, and ’ ’. The sequence of characters between the single quotes is called the atom name. Note that we are allowed to use spaces in such atoms; in fact, a common reason for using single quotes is so we can do precisely that.

3. A string of special characters. Here are some examples: @= and ====> and ; and :- are all atoms. As we have seen, some of these atoms, such as ; and :- have a pre-defined meaning.

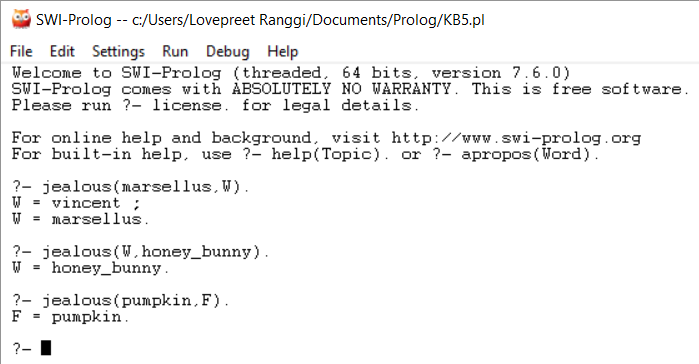
**Numbers**

Real numbers aren’t particularly important in typical Prolog applications. So although most Prolog implementations do support floating point numbers or floats (that is, representations of real numbers such as 1657.3087 or π ) we say little about them in this book. But integers (that is: …,-2, -1, 0, 1, 2, 3,…) are useful for such tasks as counting the elements of a list. Their Prolog syntax is the obvious one: 23 , 1001 , 0 , -365 ,and so on.

**Variables**

A variable is a string of upper-case letters, lower-case letters, digits and underscore characters that starts either with an upper-case letter or with an underscore. For example, X , Y , Variable , \_tag , X\_526 , List , List24 , \_head , Tail , \_input and Output are all Prolog variables. The variable \_ (that is, a single underscore character) is rather special. It’s called the anonymous variable .

**OUTPUT FOR K5**

****

**PROLOG PROJECT:-**

**Write clauses in prolog to solve water jug problem using Depth First Search.**

**Source Code:-**

min(X,Y,X):-X<Y,!.

min(\_,Y,Y).

rev(L,R):-revacc(L,[],R).

revacc([],R,R):-!.

revacc([H|T],A,R):-revacc(T,[H|A],R).

%Solve water jug problem using DFS

%X,Y are initial contents, Nx,Ny are final contents of jug1 of capacity \_ and jug2 of capacity My respectively after pouring from jug1 into jug2

chk(\_,X,My,Y,Nx,Ny):- X>0,Y<My,Ey is My-Y,min(X,Ey,P),

Nx is X-P,Ny is Y+P.

%Given 3 jugs of capacities Mx,My,Mz and filled with X,Y,Z units of a liquid respectively,give steps so that finally they contain Fx,Fy,Fz units of the liquid respectively.

jug(Mx,My,Mz,X,Y,Z,Fx,Fy,Fz):-jug(Mx,X,My,Y,Mz,Z,Fx,Fy,Fz,[],['Initially']).

jug(\_,Fx,\_,Fy,\_,Fz,Fx,Fy,Fz,T,R):-!,rev([[Fx,Fy,Fz],[Fx,Fy,Fz]|T],TR),rev(['Finally'|R],RR),display(TR,RR).

jug(Mx,X,My,Y,Mz,Z,Fx,Fy,Fz,T,R):-chk(Mx,X,My,Y,Nx,Ny),not(member([Nx,Ny,Z],T)) ,jug(Mx,Nx,My,Ny,Mz,Z,Fx,Fy,Fz,[[X,Y,Z]|T],['Pour liquid from jug1 into jug2'|R]).

jug(Mx,X,My,Y,Mz,Z,Fx,Fy,Fz,T,R):-chk(Mx,X,Mz,Z,Nx,Nz),not(member([Nx,Y,Nz],T)) ,jug(Mx,Nx,My,Y,Mz,Nz,Fx,Fy,Fz,[[X,Y,Z]|T],['Pour liquid from jug1 into jug3'|R]).

jug(Mx,X,My,Y,Mz,Z,Fx,Fy,Fz,T,R):-chk(My,Y,Mz,Z,Ny,Nz),not(member([X,Ny,Nz],T)) ,jug(Mx,X,My,Ny,Mz,Nz,Fx,Fy,Fz,[[X,Y,Z]|T],['Pour liquid from jug2 into jug3'|R]).

jug(Mx,X,My,Y,Mz,Z,Fx,Fy,Fz,T,R):-chk(My,Y,Mx,X,Ny,Nx),not(member([Nx,Ny,Z],T)) ,jug(Mx,Nx,My,Ny,Mz,Z,Fx,Fy,Fz,[[X,Y,Z]|T],['Pour liquid from jug2 into jug1'|R]).

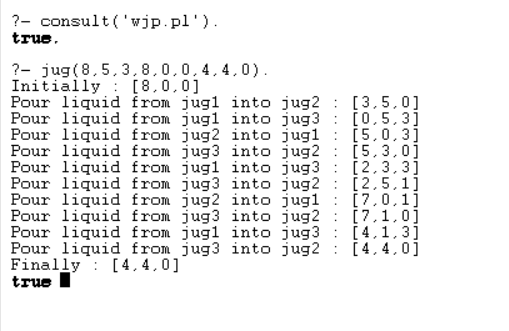
jug(Mx,X,My,Y,Mz,Z,Fx,Fy,Fz,T,R):-chk(Mz,Z,Mx,X,Nz,Nx),not(member([Nx,Y,Nz],T)) ,jug(Mx,Nx,My,Y,Mz,Nz,Fx,Fy,Fz,[[X,Y,Z]|T],['Pour liquid from jug3 into jug1'|R]).

jug(Mx,X,My,Y,Mz,Z,Fx,Fy,Fz,T,R):-chk(Mz,Z,My,Y,Nz,Ny),not(member([X,Ny,Nz],T)) ,jug(Mx,X,My,Ny,Mz,Nz,Fx,Fy,Fz,[[X,Y,Z]|T],['Pour liquid from jug3 into jug2'|R]).

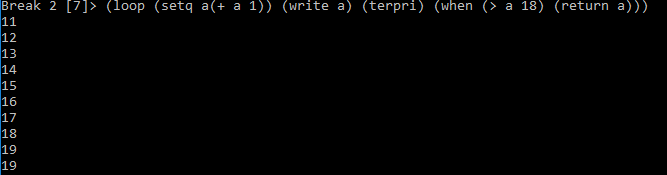
display([],[]):-!.

display([T1|T],[R1|R]):-write(R1),write(' : '),write(T1),nl,display(T,R).

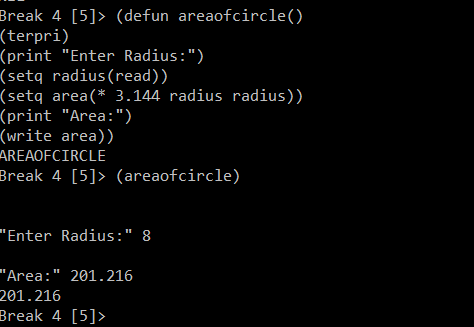
**Output :-**

****

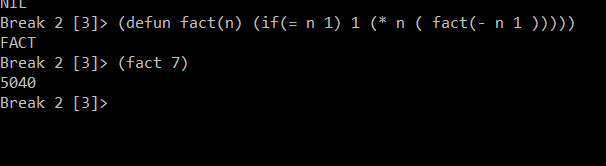
1. **(loop (setq a(+ a 1)) (write a) (terpri) (when (>a 18) (return a))).**

****

**AREA OF A CIRCLE:**



**FACTORIAL OF A NUMBER:**

****

print("\nIts a draw")

**OUTPUT;-**

