CSE
NORS

## AB SESSION 10: Hangman game

ate of the Session: __/__/__ Time of the Session: _____to_____

re-Lab:
What is the search where we examine the problem which arises when we try to plan ahead of the world nd other agents are planning against us and write its algorithm and provide a code as an example.

Adversal search is a search, where we examine the problem which arises when we try to plan ahead of the world and other agents are planning aganist. We have studied the search strategies which are only associated with a single agent that to find the solution which as then expressed in the form of a sequence of actions

L CSE
NORS

-Lab:
angman is a classic word-guessing game. The user should guess the word correctly by entering phabets of the user choice. The Program will get input as single alphabet from the user and it will tchmaking with the alphabets in the original word. If the user given alphabet matches with the phabet from the original word then user must guess the remaining alphabets. Once the user ccessfully found the word he will be declared as winner otherwise user lose the game.

```
import random
words = ("which", 'their', 'about', 'would', 'these',
'others', 'words', 'could', 'write')
def isWord(user_word, wordly_word)
    for x in user_word:
        print(x, end=" ")
    print()
for i in range(len(uses_word))
if (user word(i) == wordly_word(i):)
    print("????", end=" ")
else if user_word(i) in wordy_word
    print("????", end=" ")
else:
    print(" ", end="")
    if user_word == wordly_word:
        return;
    else:
        return 0;
```

**Post-Lab:**
The following problems that can be solved using Constraint Satisfaction Problems (CSP):
Test Case 1: Magic Square ([[1,2,3],[4,5,6],[7,8,9]])
False
This is not a magic square. The numbers in the rows/columns/diagonals do not add up to the same value.
Let's try another list of lists.
Test Case 2: Magic Square ([[2,7,6],[9,5,1],[4,3,8]])
True
Develop a python program that satisfies below operations.

| | | | |
|---|---|---|---|
| 2 | 7 | 6 | →15 |
| 9 | 5 | 1 | →15 |
| 4 | 3 | 8 | →15 |
| ↓15 | ↓15 | ↓15 | ↘15 |

```
import csp
x = csp variable (range (1,2,1))
y = csp variable (range (1,2,1))
z = csp variable (range (1,2,1))
pythagoran triple_constraint = csp Constraint (x,y,z).
and values = values (0) ↓↓ 2 + values (c)
                * * 2 - = values (2) ↓ +2
if len(values) == 3:
else:
true
total_order_constraint = csp Constraint
(x,y,z)
```

```
value[i]
if lens(values) == 3;
  else
True
pythagoran_triples_problem = clp.constraint problems
  for solution_assignments in csp_heuristic bacteria:
  print ("x", x.values)
  print ("y", y.values)
print ("z", z.values)
print()
```

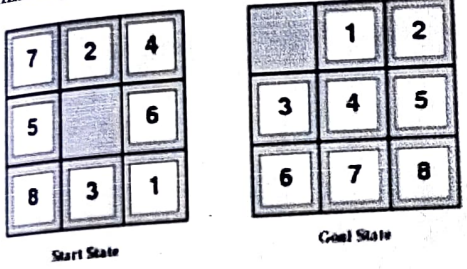
**LAB SESSION 11: Optimal Decisions in Games**

**Date of the Session:** ___/___/___ **Time of the Session:** _____to_____

**Pre Lab:**

Define Cost Function and idea cost function in 8- puzzle problem.

Take the below picture as an example and calculate the heuristic functions for it (using the most two commonly used techniques).




Cost function = returns the actual case from the start node to the current node

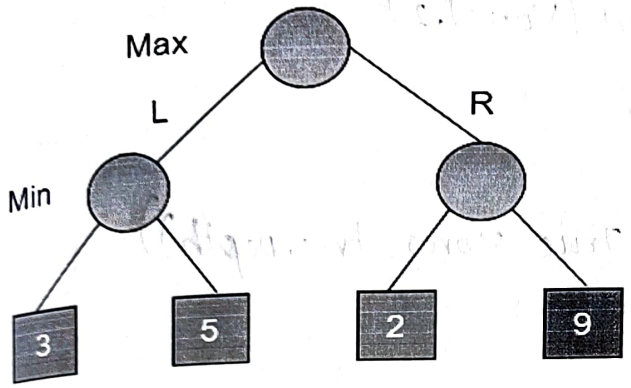
- $h(n)$ = number of misplaced tiles
- $h_2(n)$ = meh hatten distance
- $h_3(n)$ = Gaschnig's

$h_1(s)$ = ?

$h_2(s)$ = ? $3+1+2+2+3+3+2 = 18$

$h_3(s) = 8$

Consider a game which has 4 final states and paths to reach the final state are from root to 4 leaves of a perfect binary tree as shown below. Assume you are the maximizing player and you get the first chance to move, i.e., you are at the root and your opponent at the next level. Which move you would make as a maximizing player considering that your opponent also plays optimally?




```
import math
def minimet (co Depth, node Index, max Turn, scoes,
    + arget Depth):
if (cur Depth == (budget Depth):
    return scores [node Index]
if (max Turn):
    return max (minimax (ear Depth+1, node
Index*2, false, scores, target Depth),
minimax (Cur Depth +1, node Index *2 +1, false,
scores, target Depth))
else:
return min (minima* (ar Depth+1, node Index*2,
True scores, target Depth),
```

```
minimax(asDepth+1, nodeIndex*2+1, True, scores,
targetDepth)

scores = (3,5,2,5,12,5,23,25)

Depth = math.log(len(scores),2)

    print(end=" ")

print(minimax(0,0, True, scores, tree.Depth))
```

```
class Tic Tac Toe:
    def __init__(self):
        self.board:[]
    def     create_board(self):
        for i in range(5):
    row:[]
for j in range(3):
    row.append('_')
self.board.append(row)
def get_random_firstplayer(self):
    return random.randint(0,0i)
def fir_spot(self, row, col, player)
    def self.board[row][col]:player
    def is_player_win(self, player):
        win:none
        n: len(self.board)
    for i in range(n):
        win: True
    for j in range(n):
        if self.board[i][j] != player
if self is.board.filled():
```

```
        print ("match Draw!")
        break
    player = self.swap_player_turn(player)
    print()
    self.show_board()
    tic_tac_toe = TicTacToe()
    tic_tac_toe.ster()
    return False
    break
if win:
    return win
for i in range(n):
    win = True
for j in range(n):
    if self.board[j][i] != player
        win = False
        break
if win:
    return win
win = True
    for i in range(n):
        if self.board[i][j] != player:
win = False
n = len(self.board)
```

```
for i in range(n):
    win = True
for j in range(n):
    if self.board[i][j] != player:
        win = false
        break
if win:
    return win
for i in range(n):
    win = True
for j in range(n):
    if self.board[j][i] != player:
        win = false
        break
if win:
    return win
win = True
for i in range(n):
    if self.board[i][j] != player
        win = false
        break
if win:
    return win
if self's player.win(player)
    print(f) break.
```

```
N = 9

def printing(arr):
    for i in range(N):
        for j in range(N):
            print(arr[i][j], end=" ")
        print()

def issafe(grid, row, col, num):
    for x in range(9):
        if grid[x][col] == num:
            return false
    startrow = row - row % 3
    startcol = col - col % 3
    for i in range(3):
        for j in range(3):
            if grid[i + startrow][j + startcol] == num
                return false
    return True

solve sudoko(grid, row, col):
```

Writing space for the Problem:(For Student's use only)

```
if (row == P-1 and col==N)
        return True

if  col == N :
    row+=1
    col=0
  if grid[row][col]=0
return  solve Solution (grid, row, col+1)
for  num in range (1, w+1,1):
        if issafe (grid, row, col, num):
            grid[row][col]= num
    if  solve sudoko Grid, row, col sol):
            return  True
    grid [row][col] = 0
        return false.
```

```
grid = [[3,0,6,5,0,8,4,0,0],
        [5,2,0,0,0,0,0,0,0],
        [0,8,7,0,0,0,0,3,1],
        [0,0,3,0,1,0,0,8,0],
        [9,0,0,8,6,3,0,0,5],
        [0,5,0,0,9,0,6,0,0],
        [1,3,0,0,0,0,2,5,0],
        [0,0,0,0,0,0,0,7,4],
        [0,0,5,2,0,6,3,0,0]]
if (solveSudoko(grid,0,0)):
    print(grid)
else:
    print("no solution exists")
```

# LAB SESSION 12: Constraint Satisfaction problems

Date of the Session: __/__/__ Time of the Session: _____to_____

**Pre-Lab:**

1. Solve the CSP problem:

TWO+TWO=FOUR

A constraint satisfaction problem (csp) is defined by x is a set of n variables $x_1, x_2, x_3 \ldots x_n$ each defined by a finite domain $D_1, D_2 \ldots D_n$ of possible values. A solution is an assignment of values to be variables that satisfies all

Constraints

⇒ problem = TWO + TWO = FOUR

variables =

$T \in \{0, \ldots 9\}$; $W \in \{0 \ldots 9\}$; $O \in \{0, \ldots 9\}$

$F \in \{0, \ldots 9\}$; $U \in \{0, \ldots 9\}$, $R \in \{0, \ldots 9\}$

$x_1 \in \{0, \ldots 9\}$; $x_2 \in \{0, \ldots 9\}$; $x_3 \in \{0, \ldots 9\}$;

Constraints:

$O + O = R + 10 * x_1$

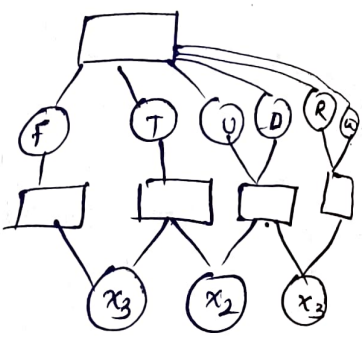
$x_1 + W + W = U + 10 * x_2$

$x_2 + T + T = O + 10 * x_3$

$x_3 = F$

each letter has a different digit

- By cryptarithmetic




```
  TWO
+ TWO
-------
 FOUR
-------
```

Variables: F, T, U, W, R, O, $x_1$, $x_2$, $x_3$

Domains: {0, 1, 2, 3, 4, 5, 6, 7, 8, 9} (same domain para11)

Example Constraints:

all dif (F, T, U, W, R, O)

A unary constraint: F ≠ 0

An n-ary constraint: O + O = R + 10 * $x_1$.

Can add constraints to restrict the $x_i$'s to 0

```
  TWO          765
+ TWO        + 765
-------      -----
 FOUR         1530
-------      -----
```

A solution is an assignment to all variables from their domain so that all constraints are satisfied.

for any csp, there might be a single, multiple or no solutions at all.

2. Discuss Zero-sum game rule with an example.

Zero-sum games are found in many contents. poker and gambling are popular examples of zero-sum games since the sum of amounts won by some players equals the combined losers of others. Games like chess and tennis, where there is one winner and one loser, are also zero-sum games.

Example of a zero-sum game:

The game of matching pennies is often cited as an example of zero-sum game, according to game theory. The game involves two players A and B simultaneously placing a penny on the table. The playoff depends on whether the pennies match or not. if both pennies are heads or tails, player A wins and keeps player B's penny. if they donot match

then player B win and keeps player A's penny

| A / B | Heads | Tails |
|---|---|---|
| Heads | (a) +1, −1 | (b) −1, +1 |
| Tails | (c) −1, +1 | (d) +1, −1 |

As can be seen, the combined playoff for A and B in all four cell is zero.