Build X : Algorithms
Week 3

- Week 2 - Winner && Challenges

- Lee's Algorithm
- Mouse Labyrinth
- Minesweeper Game
- Subarray of Maximum Sum

Week 2 - Challenges

- The Dragon (Not the Android One) – 100pts
  - Solved with a simple formula : 3*D + H
    - D = the number of days our hero fights the dragon
    - H = the number of heads the dragon initial had
  - Another solution would be to use a loop to pass through each day and calculate the number of heads the dragon will have after each day.

Week 2 - Challenges

- **Let's First Sort – 100pts**
  - Input : 5 6 4 7 2 1
    - 5 elements : 6 4 7 2 1
  - Sort the elements using any sorting algorithm learned.
    - New list : 1 2 4 6 7
  - Select and print the elements that are on even positions.
    - Elements to print : 2 6

And the winner is …Amandine Jala

- Alex Clapa
- Jerome De Chillaz
- Tania Copocean
- Amandine Jala
- Razvan-Gabriel Ceangu
- Ruxandra Anghel

Lee’s Algorithm – Short Description

- Identifies the minimum number of steps we need to make in a matrix.

  - Example of usage:
- Labyrinth Problem – Determine the number of steps we need to make to find an exit

Lee's Algorithm - Characteristics

- Identical with BF but used on a 2D data structure such as a matrix.
- Considered to be efficient:
  - Time Complexity : $O(m*n)$
- Used quite often

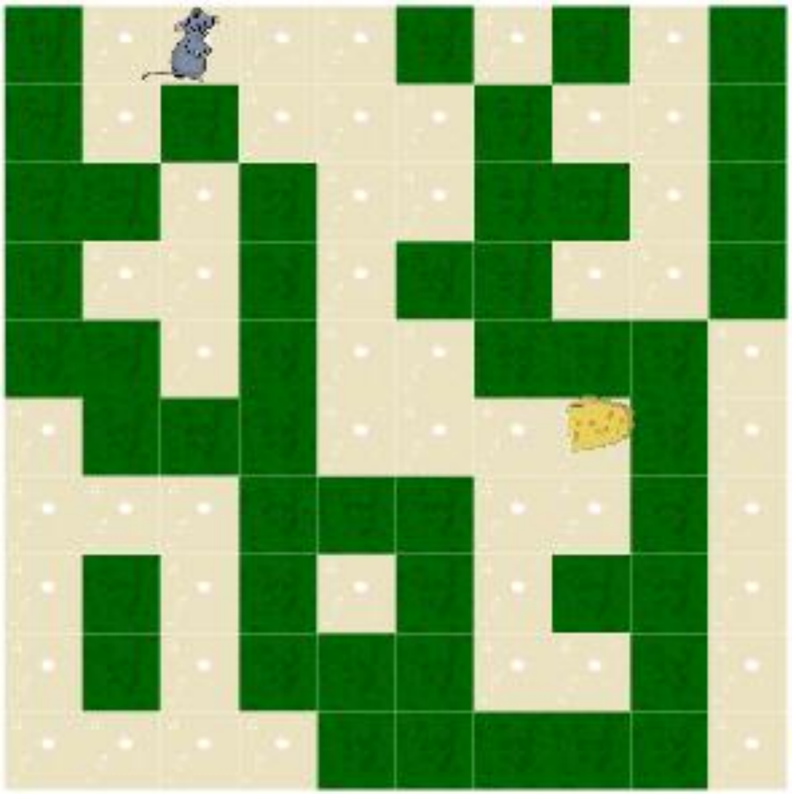
Lee’s Algorithm - Representation

Row/ Column	0	1	2	3	4
0	X	X	X	X	X
1	X	0	0	0	-1
2	X	0	0	X	X
3	X	0	0	X	X
4	X	0	X	X	X

Labyrinth Problem - Description

- Given a labyrinth find a path to the solution.
- Steps :
  - Represent your labyrinth;
  - Specify the possible moves;
  - Apply the algorithm;

Labyrinth Problem - Representation



	0	1	2	3	4	5	6	7	8	9	10	11
0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
1	-1	-1	0	0	0	0	-1	0	-1	-1	-1	-1
2	-1	-1	0	-1	0	0	0	-1	0	0	-1	-1
3	-1	-1	-1	0	-1	0	0	-1	-1	0	-1	-1
4	-1	-1	0	0	-1	0	-1	-1	0	0	-1	-1
5	-1	-1	-1	0	-1	0	0	-1	-1	-1	0	-1
6	-1	0	-1	-1	-1	0	0	0	0	-1	0	-1
7	-1	0	0	0	-1	-1	-1	0	0	-1	0	-1
8	-1	0	-1	0	-1	0	-1	0	-1	-1	0	-1
9	-1	0	-1	0	-1	-1	-1	0	0	-1	0	-1
10	-1	0	0	0	0	-1	-1	-1	-1	-1	0	-1
11	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1

Time to code

-Matrix Representation-

Labyrinth Problem – Possible Moves

- a. N, S, E, V (4 moves)
  - Use 2 lists for xOy representations
    - dx = {-1, 0, 1, 0};
    - dy = {0, 1, 0, -1};
- b. N, S, E, V, NE, NV, etc. (8 moves)
  - Use the same 2 lists but with more coordinates:
    - dx = {-1, -1, 0, 1, 1, 1, 0, -1};
    - dy = {0, -1, -1, -1, 0, 1, 1, 1};

Time to code

-Lee Algorithm-

Different variations

- Add obstacles
- Add more solutions
- Use a Queue structure to expand only certain nodes

Break
-10 min-

Queue

- It is an abstract data structure which works after the FIFO technique
- Elements are kept are kept in order

- Has only 2 methods
  - Enqueue :
    - addition of elements to the rear of the queue;
  - Dequeue :
    - Removal of elements from the front of the queue;

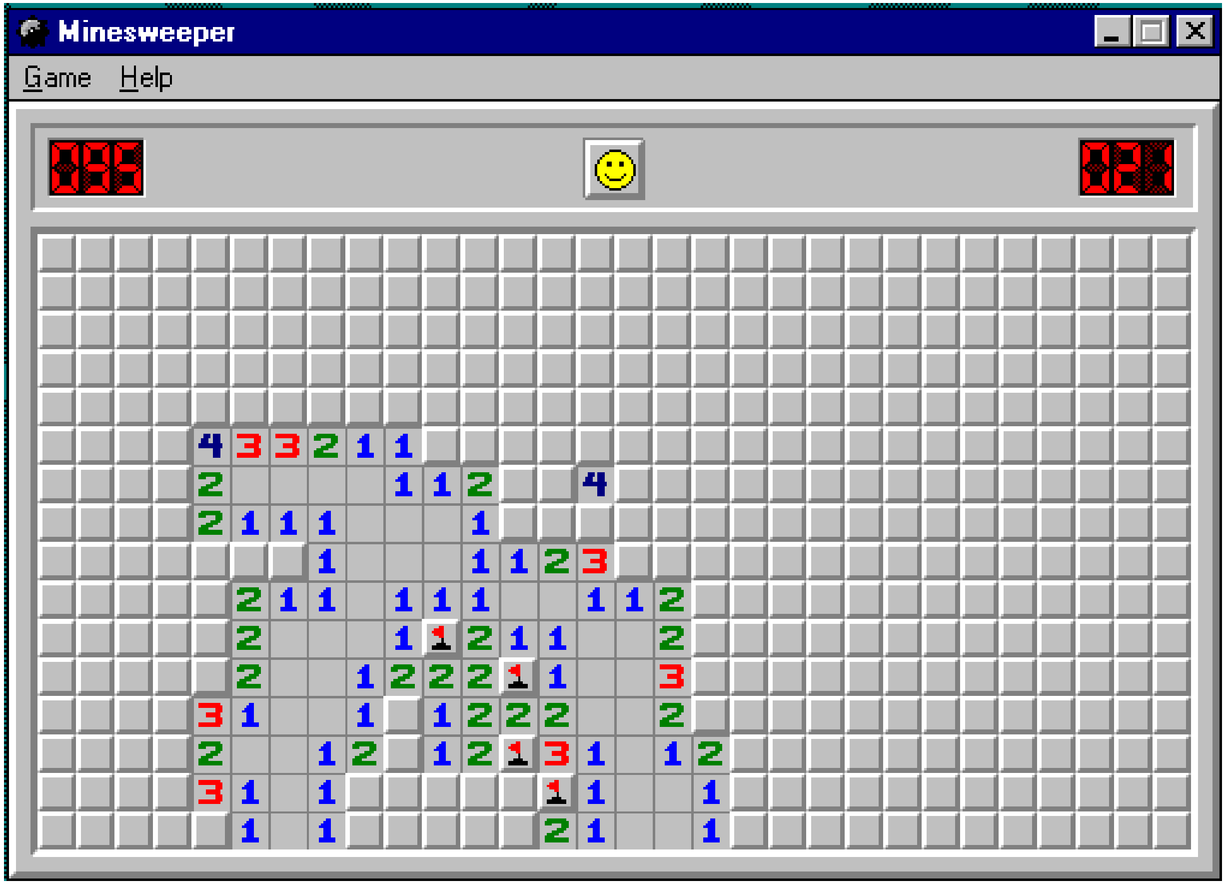
Queue – Advantages

- No real limit on the memory
- Used when we need to use the elements in the same order we got them
  - E.g.
    - Manage print commands from multiple users connected to a printer
    - Hotel Booking Systems

Queue in Lee’s Algorithm

- We can use Queues in order to keep track of the nodes we need to expand next or nodes that have already been expanded

Minesweeper



Minesweeper - Steps

- Create the field
- Border it
- Place the bombs
- Initialize the rest of the places with the number of neighbours that contains a bomb
- Use Lee’s Expansion Algorithm to interact with the user’s click

Maximum Subarray - Examples

- E.g.
  - 5 4 1 -10 2
    - 5 4 1
  - 1 2 3 -4 10
    - 1 2 3 -4 10
  - -2 -1 -3 4 -1 2 1 -5 4
    - 4 -1 2 1

Maximum Subarray - Description

- Given a list of elements find the sub-list that have the maximum sum
- Let's take:
  - S[] = (s1, s2, s3, … , sn), S contains n elements
  - A sub-list of S is a list of the form :
    - (s(i), s(i+1), s(i+2), … , s(j)); $0 <= i <= j < n$
  - Sum of the sub-list is
    - Sum = s(i) + s(i+1) + s(i+2) + … + s(j);

Maximum Subarray - Solutions

- 1. O (n^3)
  - Set 2 indexes i and j and iterate between them
- 2. O (n^2)
  - Use the same 2 indexes but calculate the sum while you iterate
- 3. O(n * log(n))
  - Use the Divide and Conquer technique.

Maximum Subarray - Solutions

- 4. O(n)
  - Using a new structure to calculate all the sums up to the position I

- 5. O(n)
  - Dynamic Programming

Next week

Special Guest : **Luca Vigano**

-Algorithms in Security-