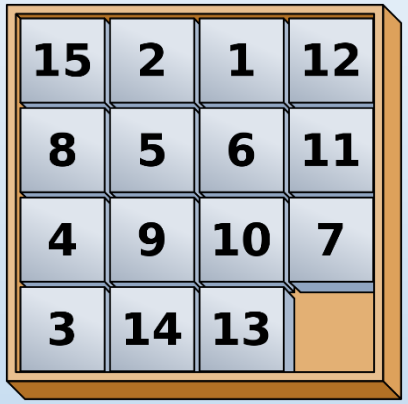
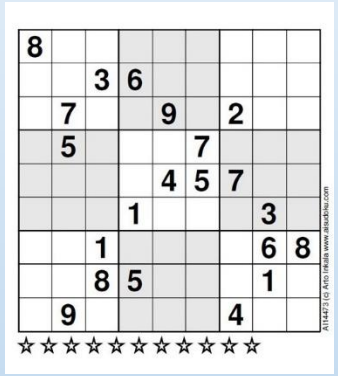
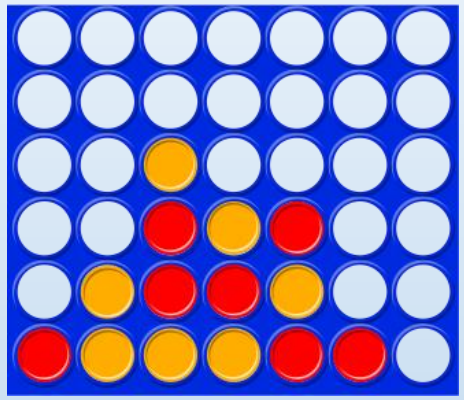
The three fondamental mechanisms of AI: Search, Optimisation, Learning

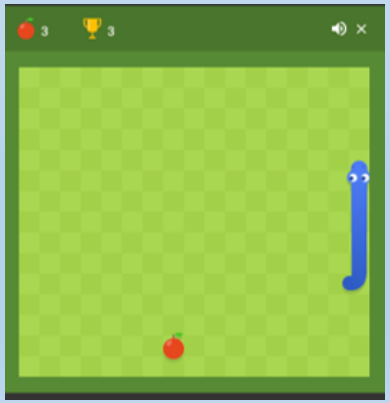
Hugues Bersini
IRIDIA-CODE
ULB

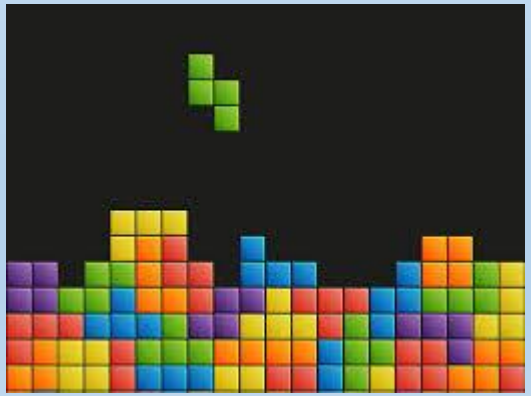
Use indifferently different techniques for the same problems

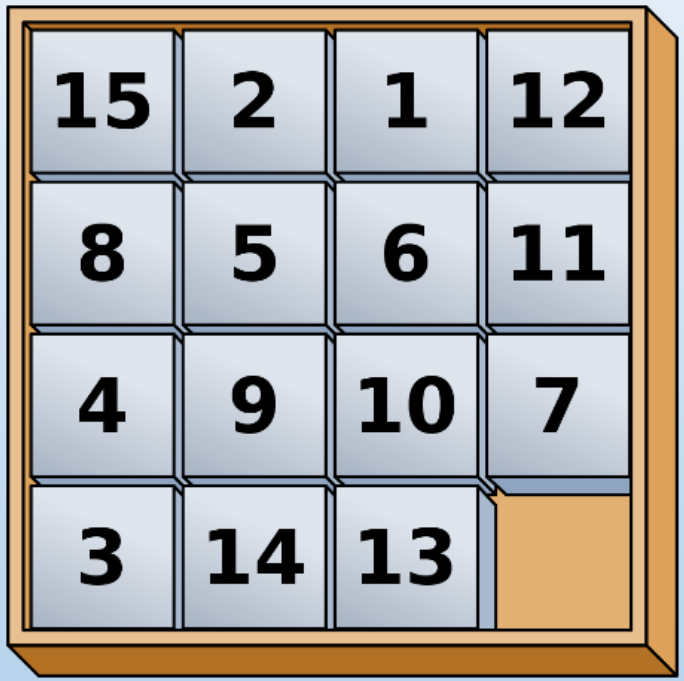












- A* = search
- Genetic Algorithm = optimisation
- Q-Learning = learning

How to differentiate them:

- CPU Time and Memory
- Algorithm Complexity
- Ease to integrate Human Expertise

Multiple ways to mix or to hybridize them

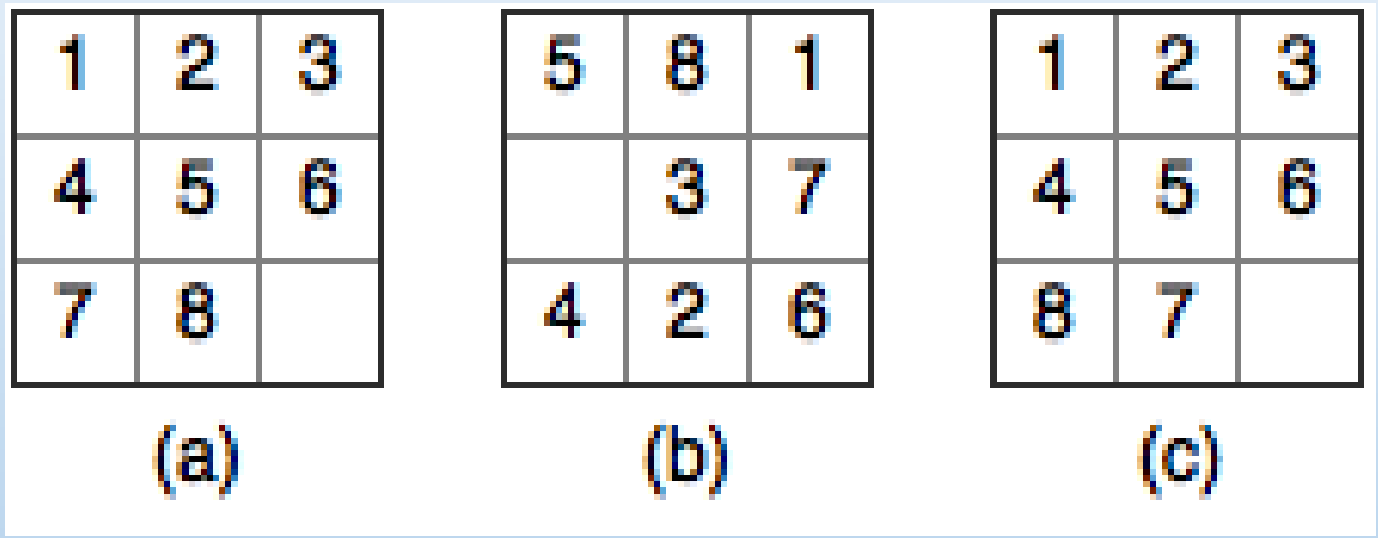
- Make AI becoming a software tool box
- For instance:
  - Learn the right heuristic for A*
  - Improve the optimisation algorithm for the learning phase of Neural Nets
  - Learn the best parameters of a Genetic Algorithm
  - Combine them: Optimisation + Neural Nets

  + Classical problem solving algorithms not originally coming from and with AI

Not immune to fashion

- Neural Nets many successive deads and resurrections: 50’, 80’, 2000’
  - Connectionims, NN, Deep Learning
- Fuzzy Sets in the 80’
- Complexity Theory: Chaos, Multi-Agents
- The Deep Learning current fashion

First Exemple: the 15 puzzle game



1	2	3
4	5	6
7	8	

(a)

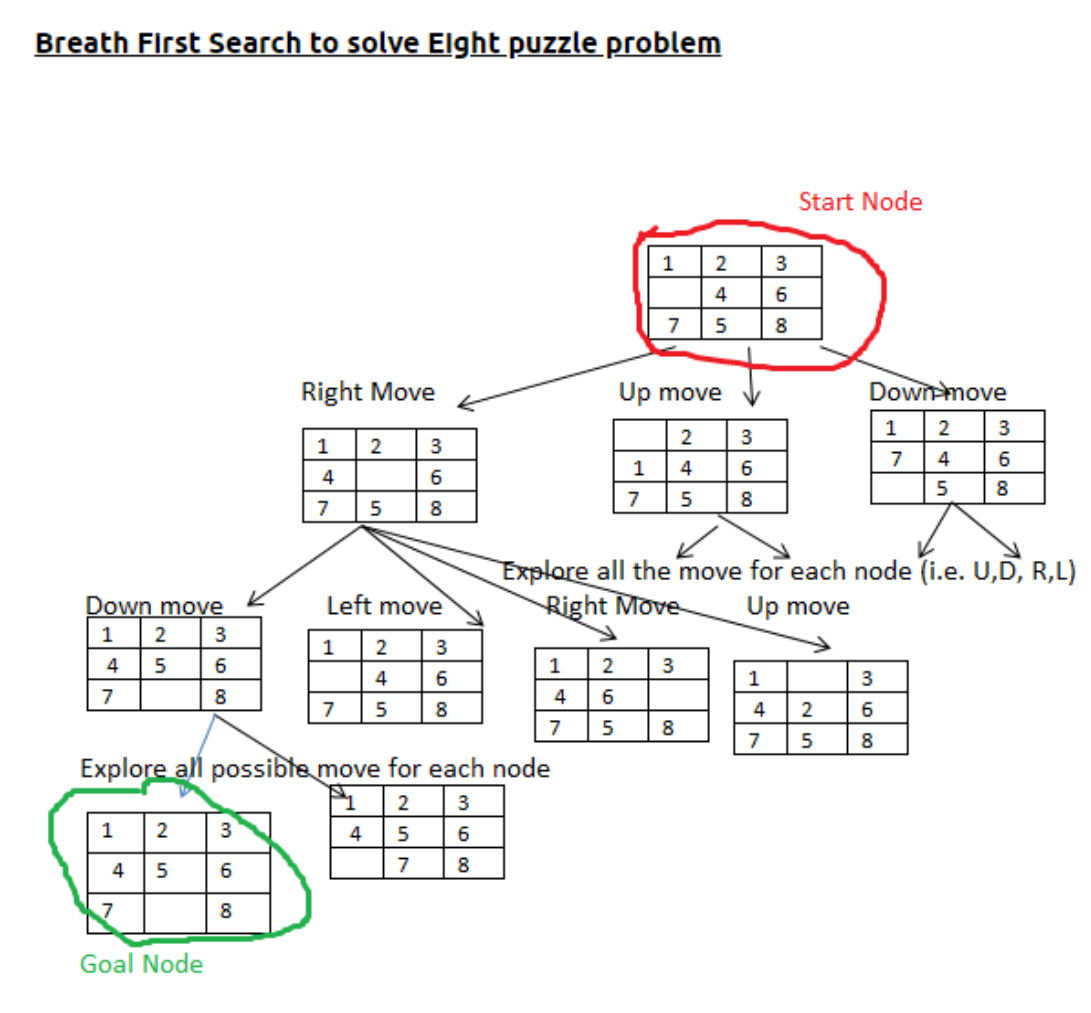
5	8	1
	3	7
4	2	6

(b)

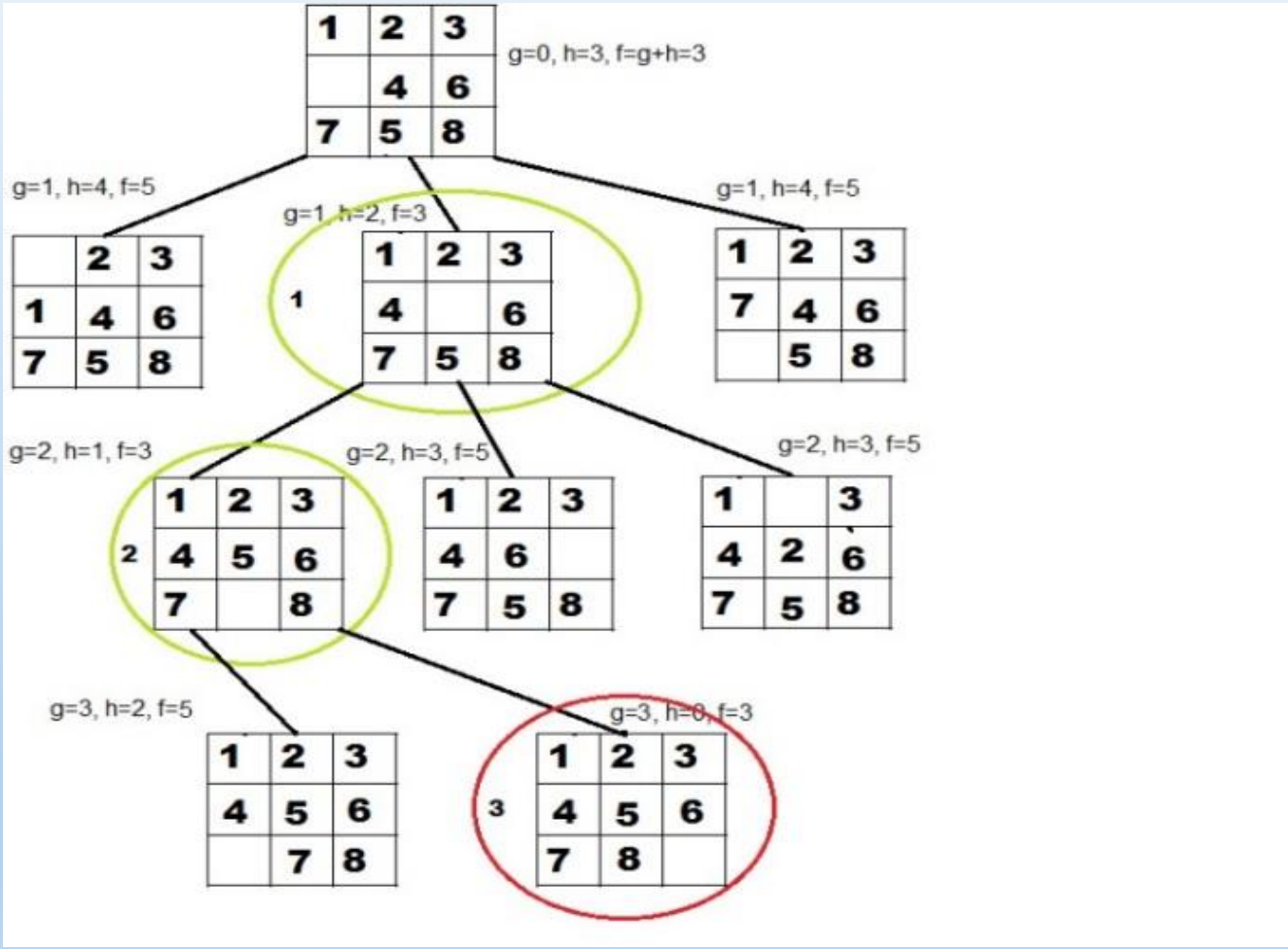
1	2	3
4	5	6
8	7	

(c)

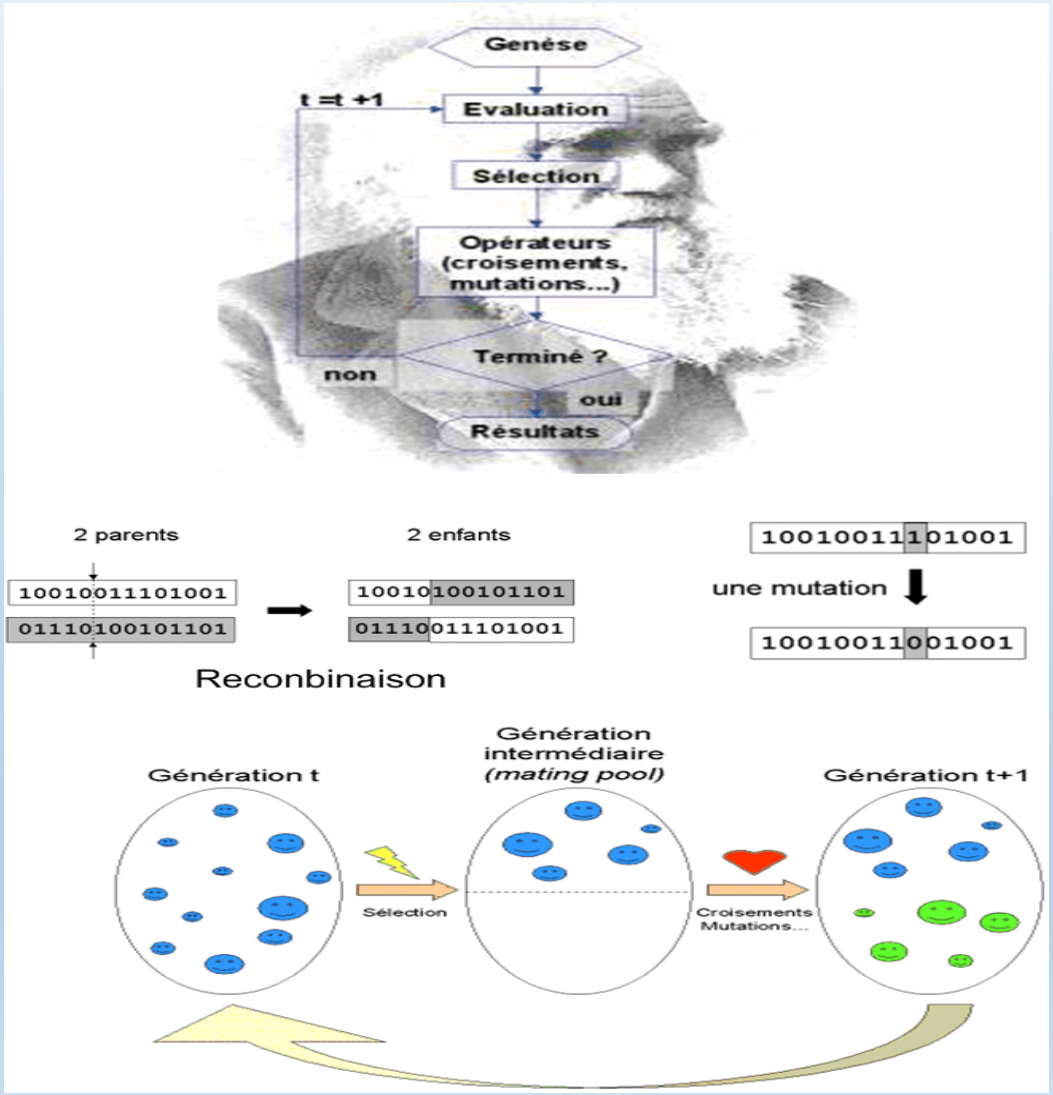
Breath First



A*


g=0, h=3, f=g+h=3
g=1, h=4, f=5
g=1, h=2, f=3
g=1, h=4, f=5
1
g=2, h=1, f=3
g=2, h=3, f=5
g=2, h=3, f=5
2
g=3, h=2, f=5
g=3, h=0, f=3
3

Genetic Algorithms



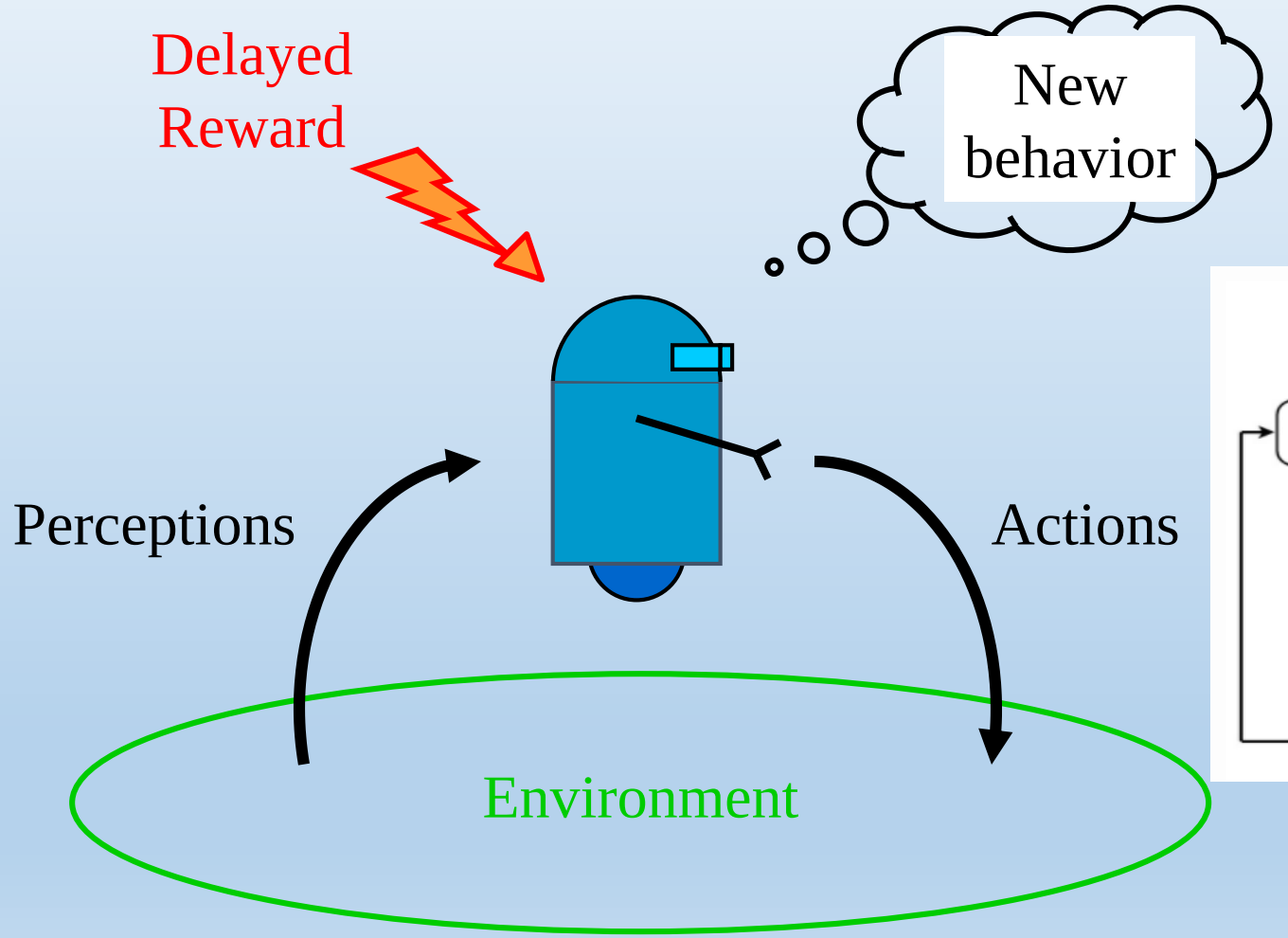
[1,3,4,2,3,1] et [1,4,3,3]
pour donner [1,3,4,3,3] et [1,3,2,3,1]

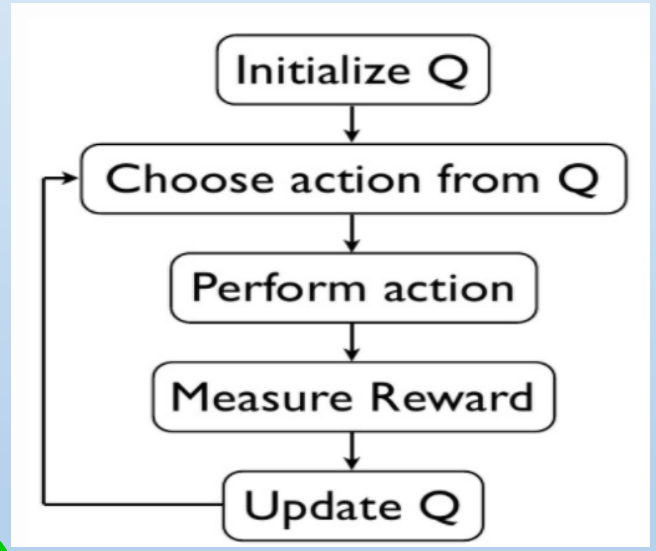
Fitness = *Individual size: nbr of moves + nbr of bad positions at the end of the sequence of moves*

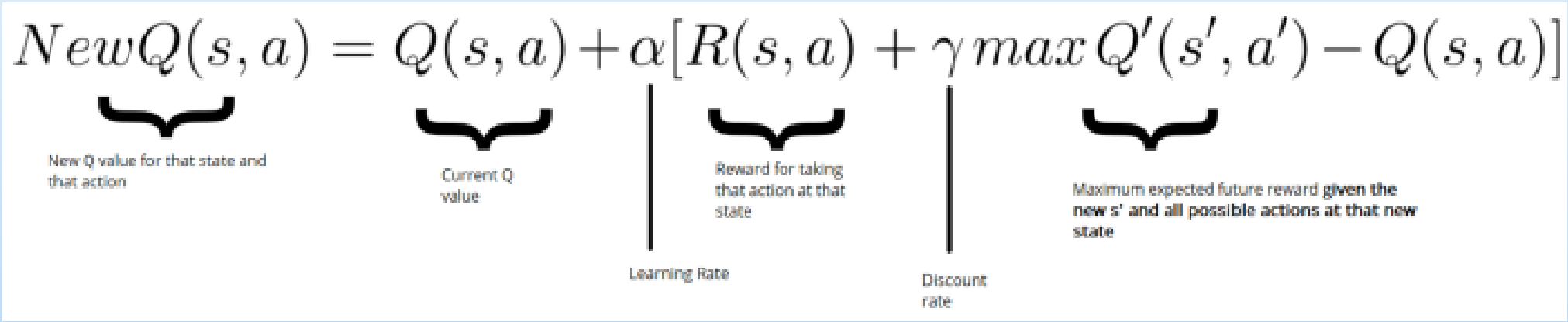
Drawbacks and Discussions

- Disappear the notion of states and trees that guide the search
- Possibility of very long sequences or sequences composed of cycles.
- In A*, possibility of backtracking, very guided construction of moves
- It’s very possible that after a very long CPU time, solution will be found
- Many AI users choose to favour CPU time instead of Cognitive time
- But this CPU time saving still makes a lot of sense and honors and gives tribute to human intelligence

Q-Learning


Delayed
Reward
New
behavior
Perceptions
Actions
Environment


Initialize Q
Choose action from Q
Perform action
Measure Reward
Update Q



The state is each possible configuration

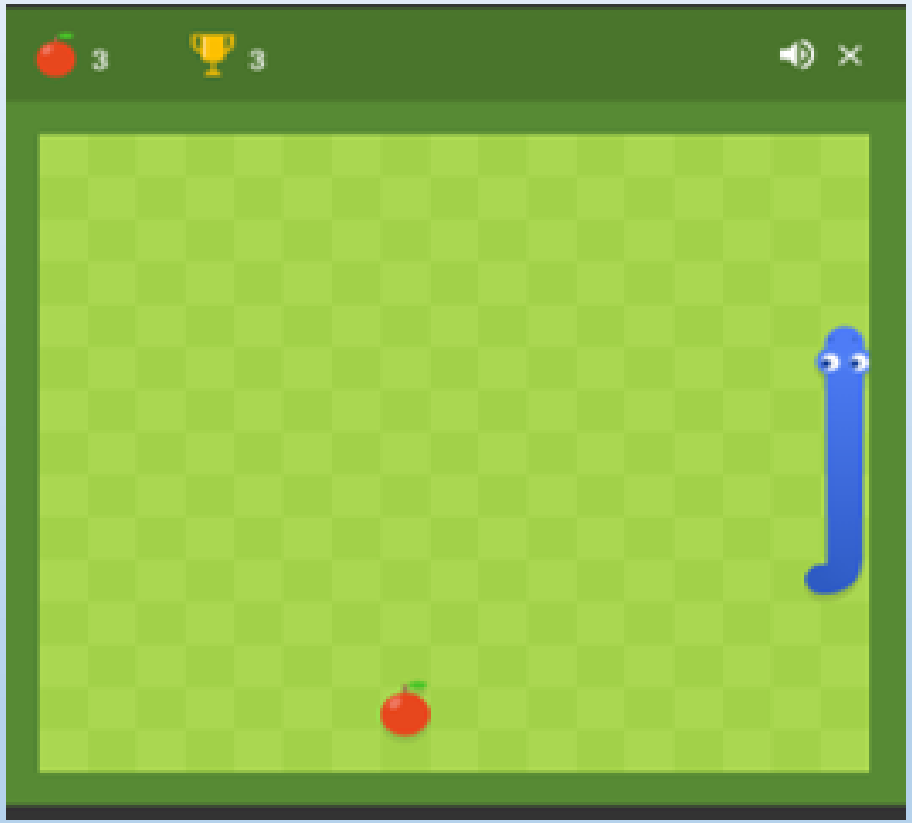
The actions are all possibles moves from that state

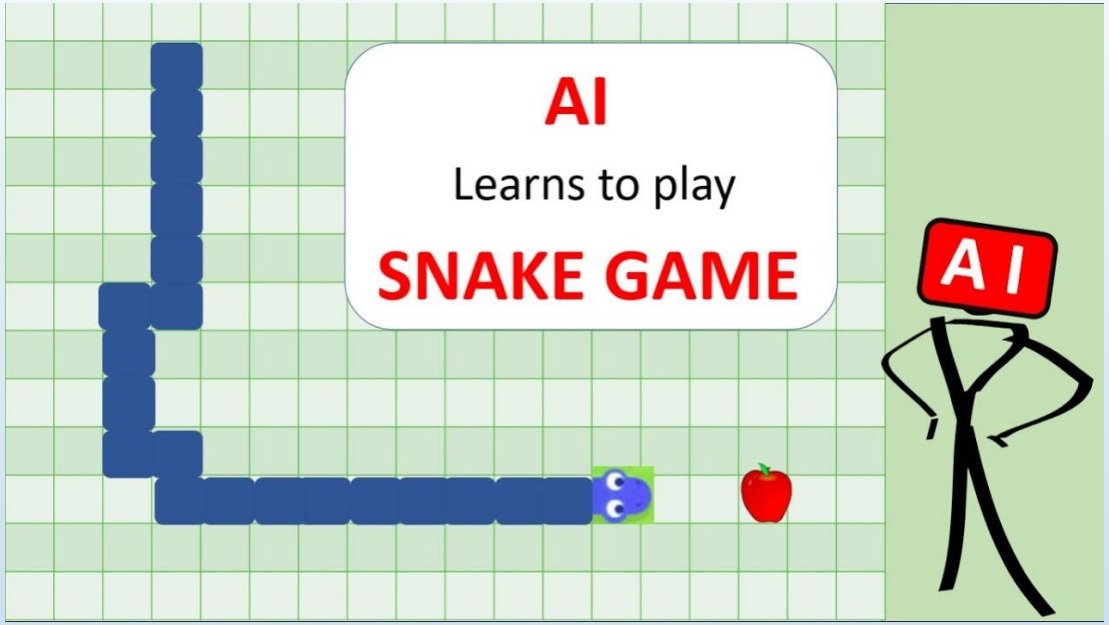
Drawbacks and discussions

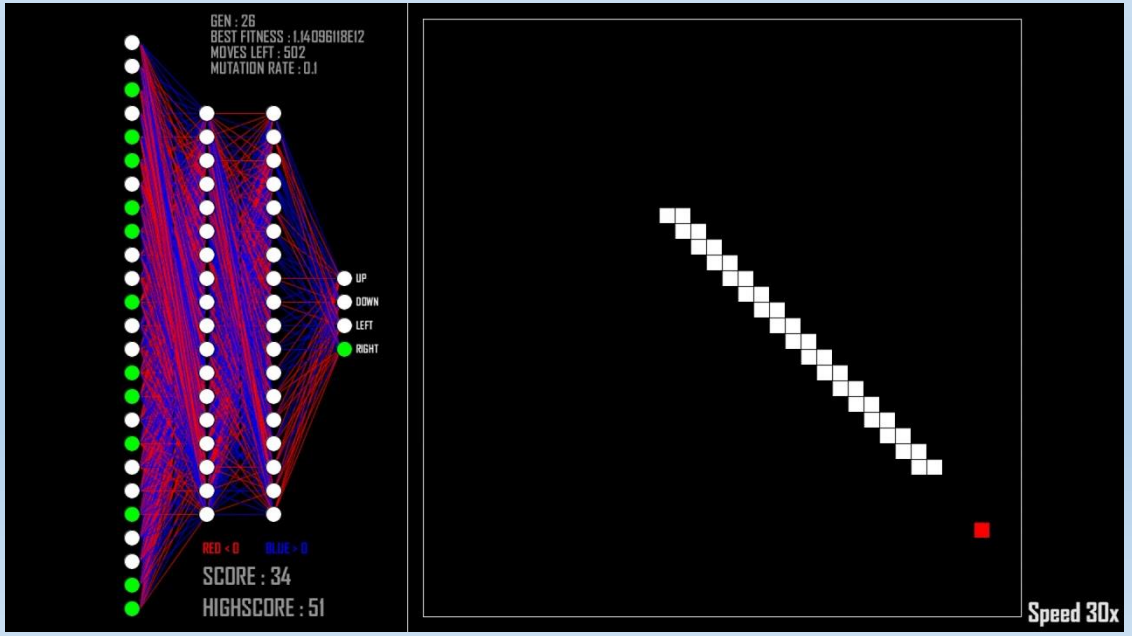
- The notion of state is well preserved
- The best sequence of moves will be even found for teach initial configuration (each initiat state)
- Learning to compensate for the need for an intermediary evolution of the state
- The only possible feedback is at the end like with GA
- But even so, it could be possible to run an invert A* (called also dynamic programming) to accelerate the search.
- But still a learning of the best A* heuristic might still be an interesting idea.

Snake and Tetris

Snake: A* or Q-Learning



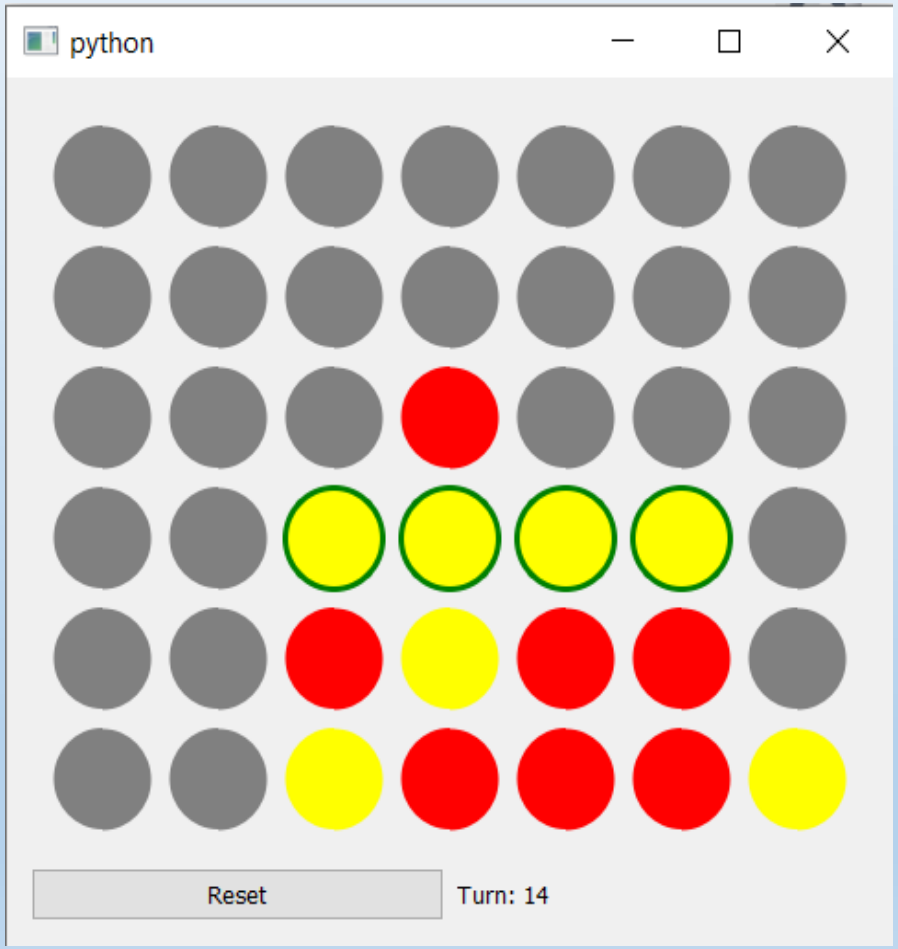


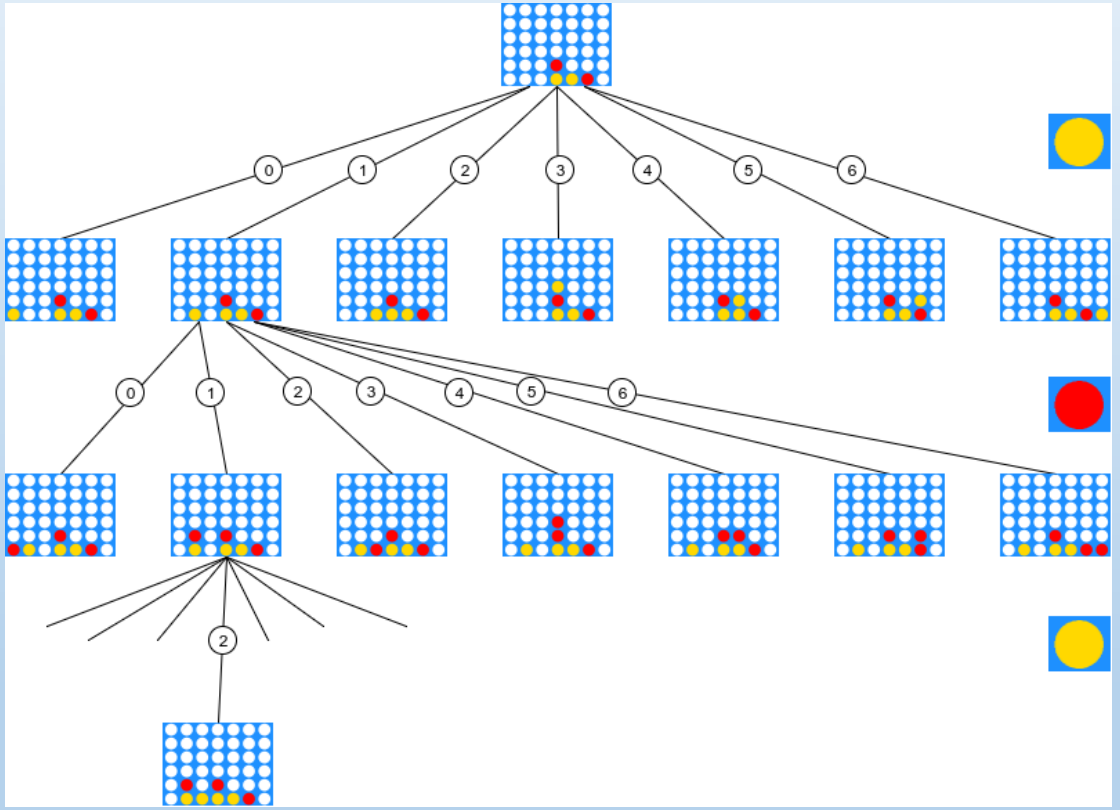


- The states can be defined relatively to the snake head, to limit the size.
- The four actions are easy to define
- The snake moves independently of the actions. Hard to draw the search tree.
- A* becomes very problematic since no final state really exists.
- Much easier to use reinforcement or a cost function to evaluate the performance of the snake and thus the player.
- -> Reinforcement Learning or NN + GA are classical ways to find the best player.

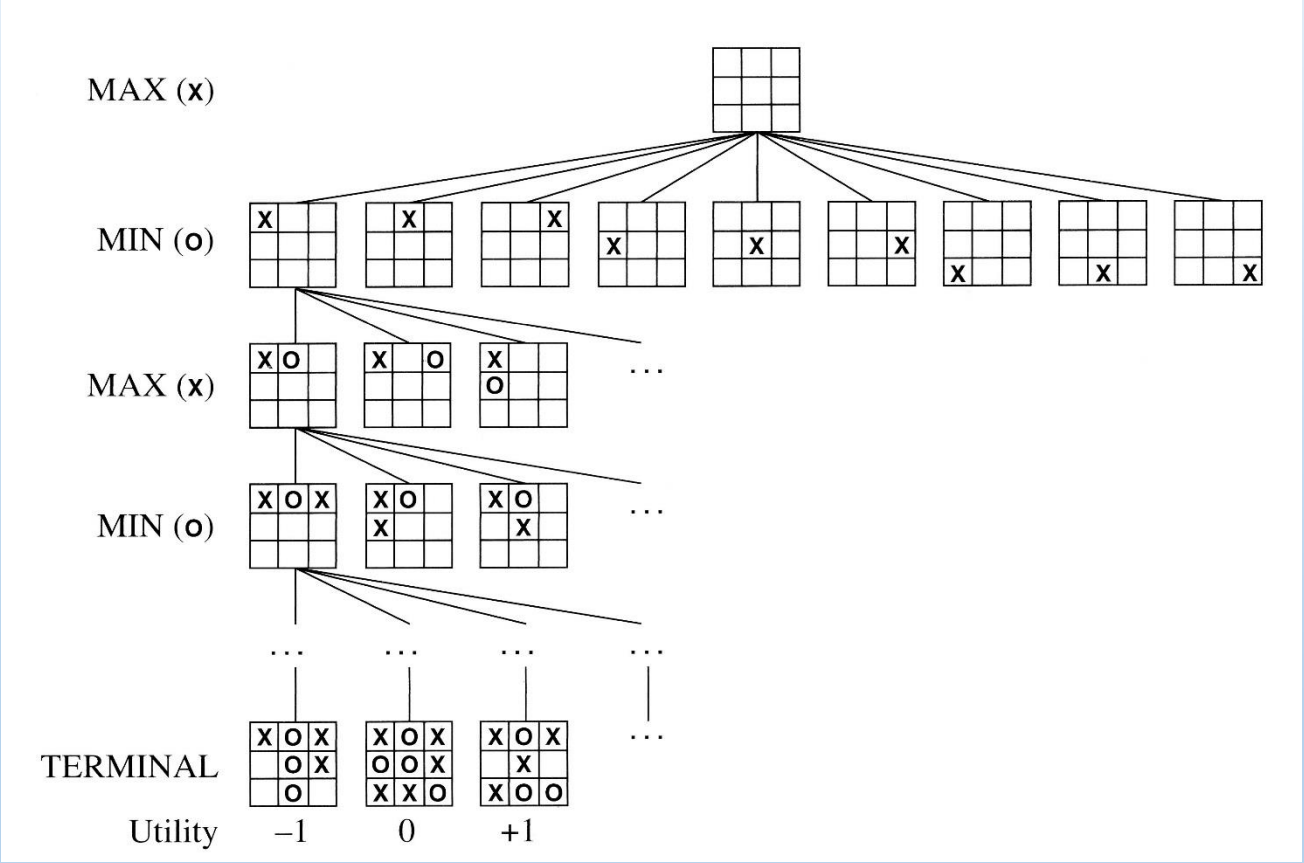
Connect-4 and two players game.

Connect-4

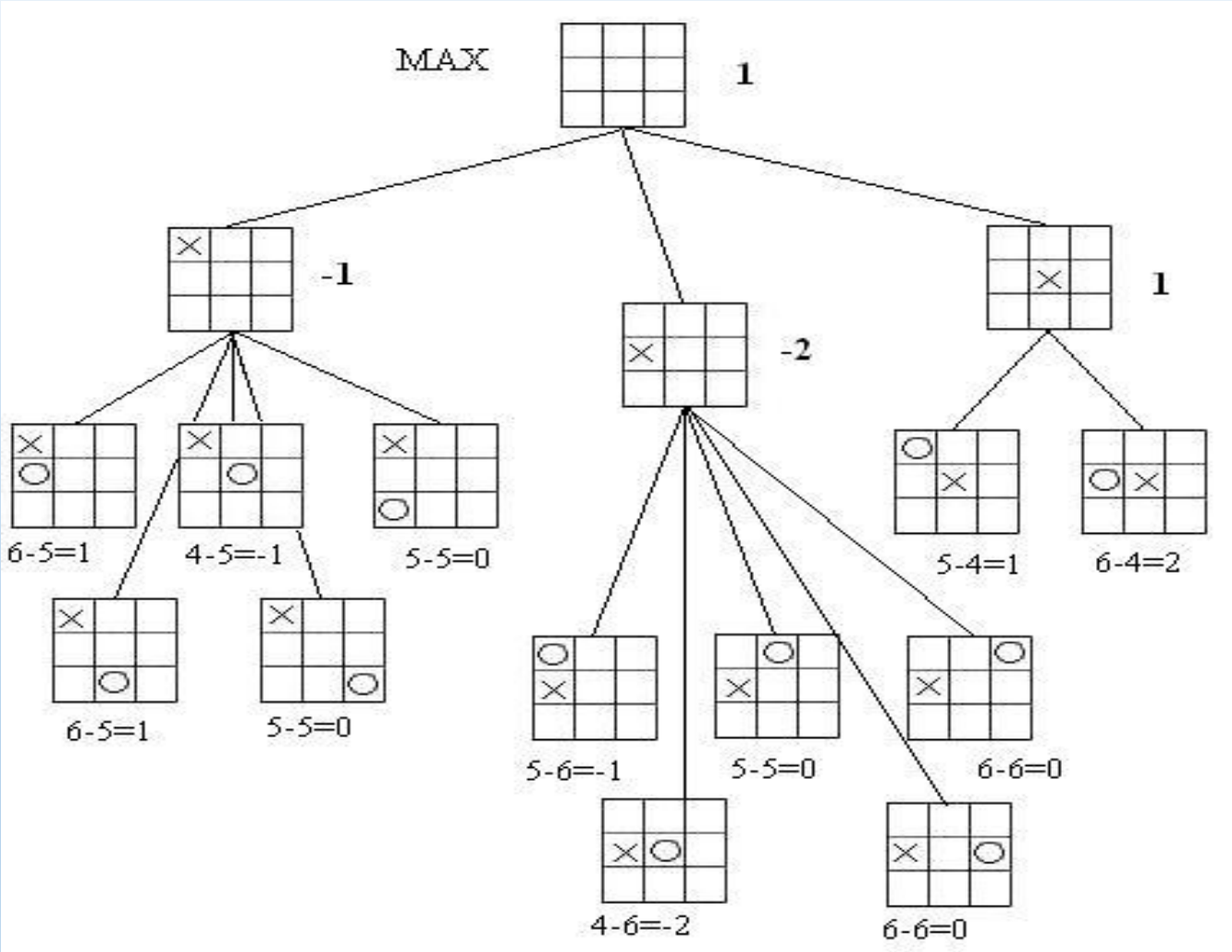




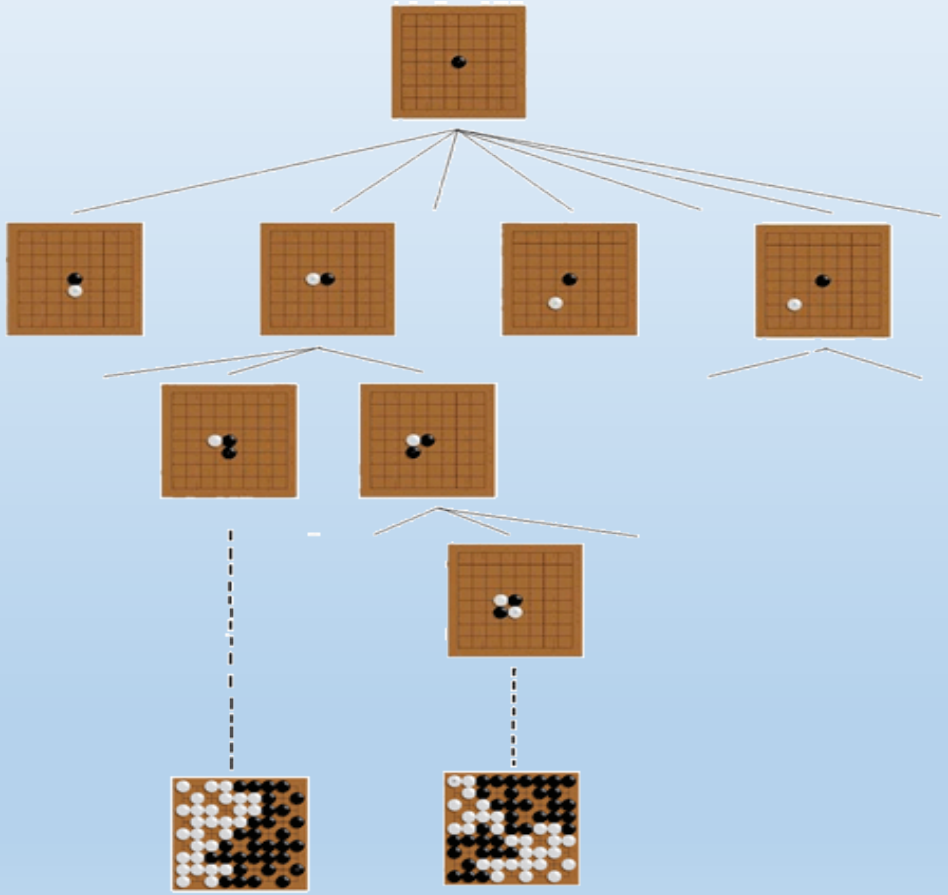
The Min-Max Strategy for the Tic-Tac-Toe



With heuristics

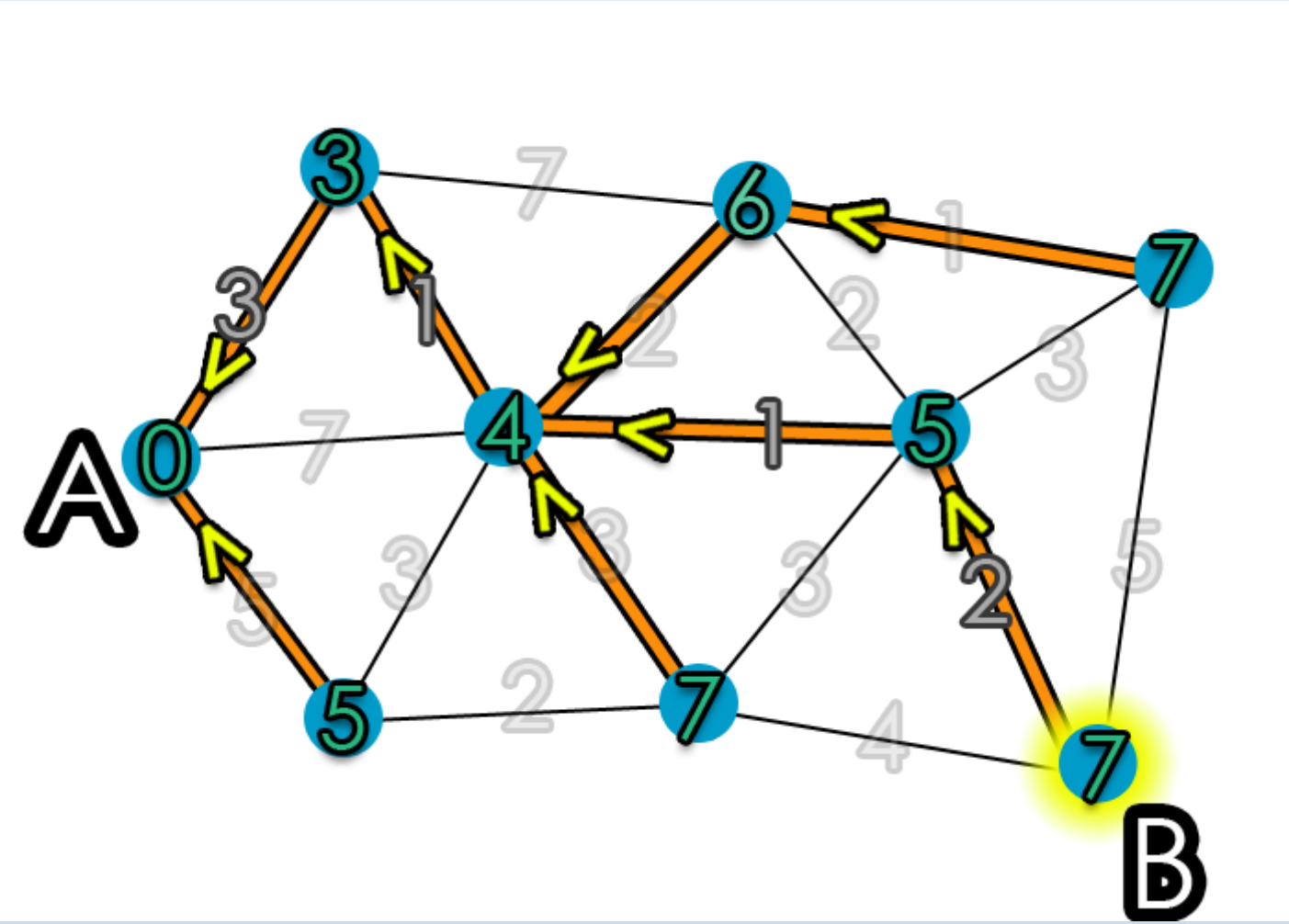


Learning the heuristics



Learning by statistics.
Playing randomly and
computing the number of lost games
vs the number of winning games.

Discovering the shortest path: Dijkstra or A*

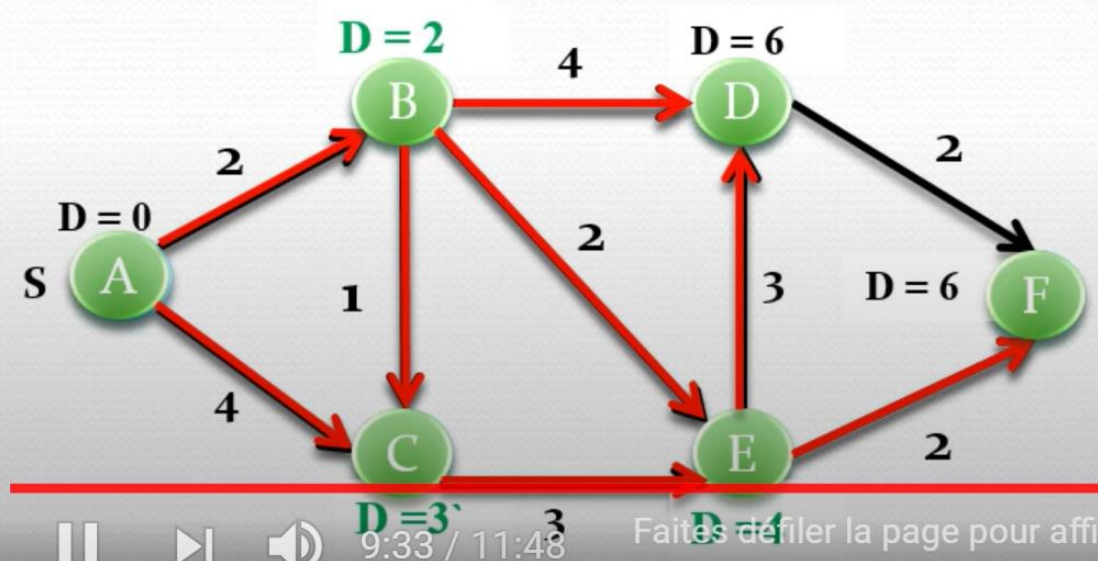


Graph: Dijkstra's Algorithm With Animation (Shortest Path Search)

Dijkstra's Rules

Rule 1: Make sure there is no negative edges. Set distance to source vertex as zero and set all other distances to infinity.
Rule 2: Relax all vertices adjacent to the current vertex.
Rule 3: Choose the closest vertex as next current vertex.
Rule 4: Repeat Rule2 and Rule 3 until the queue or reach the destination.

If (D[C] + D[AdjEdge]} < D [Adj]) {Update Adj's D with new shortest path}

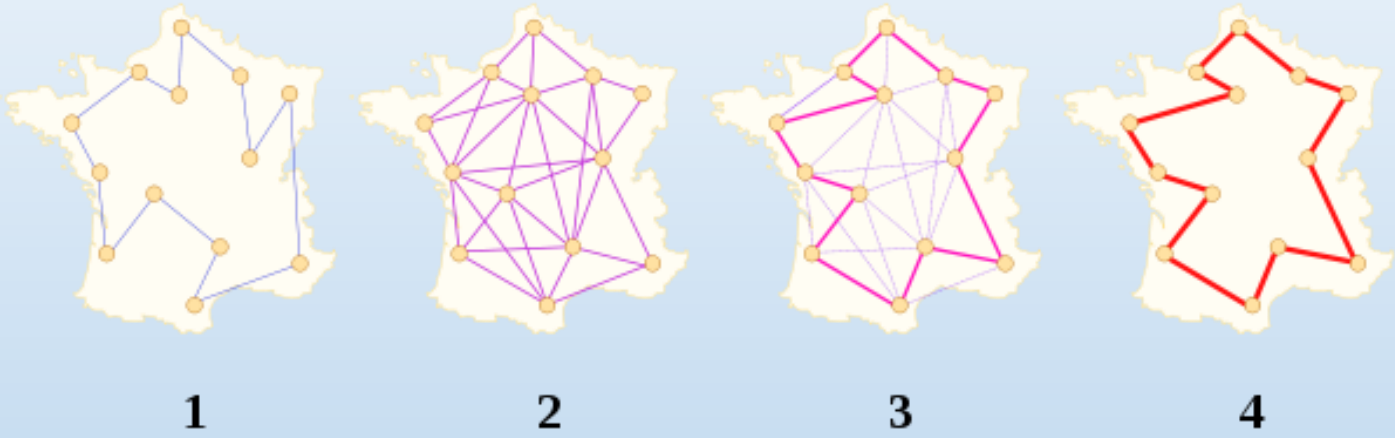


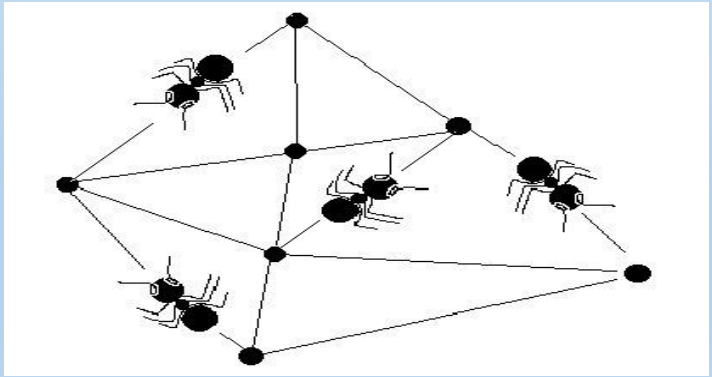
Q <= V	A	B	C	D	E	F
A	0^A	2^A	4^A	∞^A	∞^A	∞^A
B	0^A	2^A	3^B	6^B	4^B	∞^A
C	0^A	2^A	3^B	6^B	4^B	∞^A
E				6^B	4^B	6^E

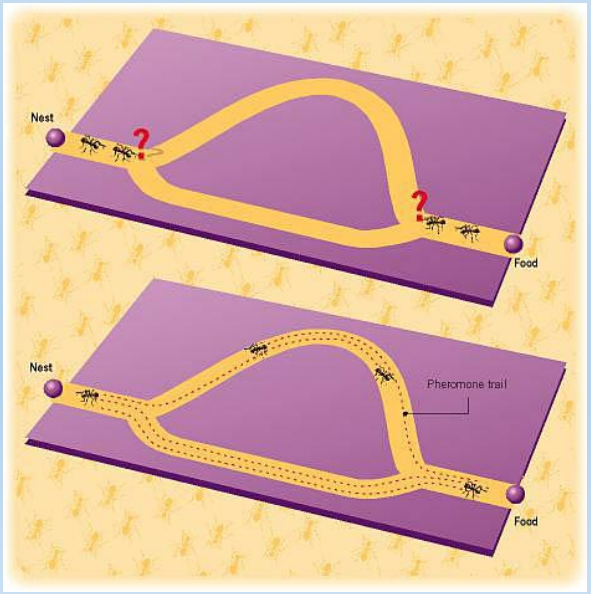
9:33 / 11:48

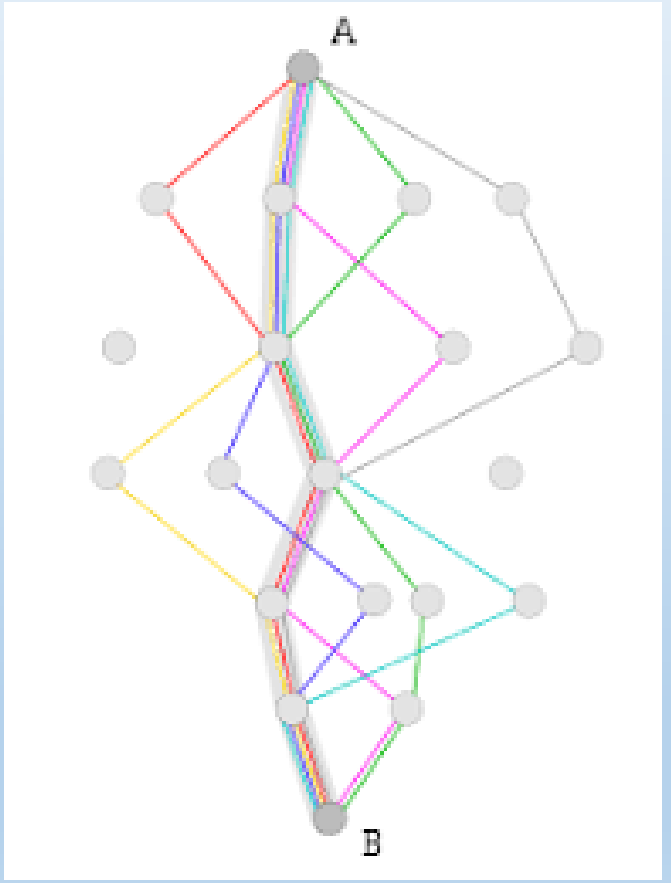
Faites défiler la page pour afficher plus de détails

GA and ACO for shortest path









Distinguishing a cat from a dog: NN and DL