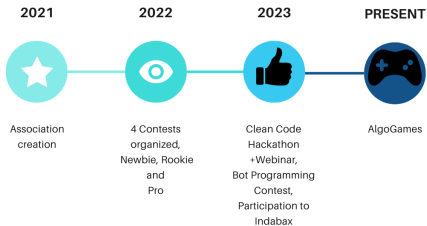


Graph traversal and Path-finding Algorithms used in Video Games

Marius Rabenarivo



A brief history of AlgoMada



About me



- ▶ Telecommunications, ESPA Alumni
- ▶ Computer Science, University of Reunion Island Alumni
- ▶ FaceDev Admin since 2012
- ▶ Founder member of AlgoMada
- ▶ Clojure dev
- ▶ Computer Science Enthusiast
- ▶ Current interests: Cryptocurrency, Clojure programming language
- ▶ Side project: BetaX Community

What is a graph?

A data structure to represent link between objects. A graph is defined by a set of nodes V and a set of edges E .

We can summarize this definition by the following formula:

$$G = (E, V)$$

Example:

<https://www.redblobgames.com/pathfinding/grids/graphs.html>

What's the difference between a graph and a tree?

A graph can contain cycles (a node can be visited twice).

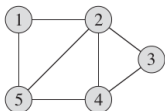
Different type of graphs

- ▶ Acyclic Graph A graph that has no cycle.
- ▶ Cyclic Graph A graph that has at least one cycle.
- ▶ Directed Graph A graph in which edge has direction. That is the nodes are ordered pairs in the definition of every edge.
- ▶ Undirected Graph A graph in which edge are not directed. Meaning, the edges are defined by an unordered pair of nodes.
- ▶ Directed Acyclic Graph A graph that is both directed and acyclic.
- ▶ Connected graph Every pair of nodes has a path linking them. Put in another way, there are no inaccessible node.
- ▶ Disconnected graph A graph in which there is at least one inaccessible node.
- ▶ A multigraph A graph that can have multiple edges between the same nodes.

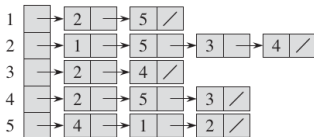
Different way to represent a graph

There are 2 ways to represent a graph:

- ▶ adjacency list For each node, provide a list of other nodes that are adjacent to it.
- ▶ adjacency matrix A matrix construct by aligning the nodes in the row and the columns and putting a value if the nodes are linked by an edge.



(a)



(b)

	1	2	3	4	5
1	0	1	0	0	1
2	1	0	1	1	1
3	0	1	0	1	0
4	0	1	1	0	1
5	1	1	0	1	0

(c)

Figure 1: (a) Undirected graph with 5 vertices and 7 edges (b) Adjacency-list representation (c) Adjacency-matrix representation

Graph traversal algorithms

DFS (Depth-First Search)

A graph traversal algorithm in which one start with a root node (arbitrarily chosen) then explore as far as possible along each branch before backtracking.

BFS (Breadth-First Search)

A graph traversal algorithm in which one explore every possible node in the current depth level before going to the next.

BFS (Breadth-First Search)

```
BFS( $G, s$ )
1  for each vertex  $u \in G.V - \{s\}$ 
2       $u.color = \text{WHITE}$ 
3       $u.d = \infty$ 
4       $u.\pi = \text{NIL}$ 
5   $s.color = \text{GRAY}$ 
6   $s.d = 0$ 
7   $s.\pi = \text{NIL}$ 
8   $Q = \emptyset$ 
9  ENQUEUE( $Q, s$ )
10 while  $Q \neq \emptyset$ 
11      $u = \text{DEQUEUE}(Q)$ 
12     for each  $v \in G.Adj[u]$ 
13         if  $v.color == \text{WHITE}$ 
14              $v.color = \text{GRAY}$ 
15              $v.d = u.d + 1$ 
16              $v.\pi = u$ 
17             ENQUEUE( $Q, v$ )
18      $u.color = \text{BLACK}$ 
```

Figure 2: Breadth-first search pseudo-code

BFS (Breadth-First Search)

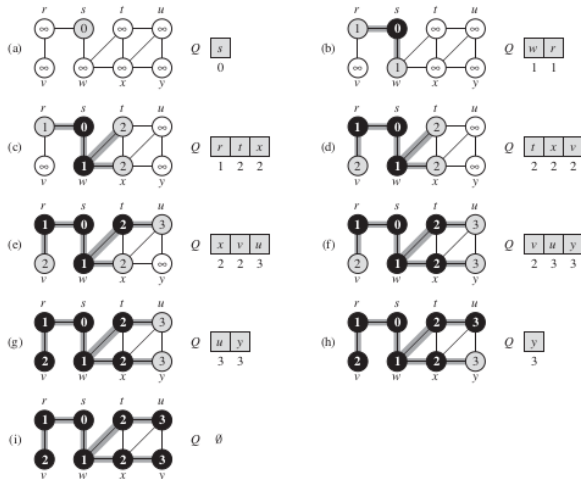


Figure 3: Operation of BFS on an undirected graph

Depth-First Search

DFS(G)

```
1  for each vertex  $u \in G.V$ 
2       $u.color = \text{WHITE}$ 
3       $u.\pi = \text{NIL}$ 
4   $time = 0$ 
5  for each vertex  $u \in G.V$ 
6      if  $u.color == \text{WHITE}$ 
7          DFS-VISIT( $G, u$ )
```

DFS-VISIT(G, u)

```
1   $time = time + 1$                 // white vertex  $u$  has just been discovered
2   $u.d = time$ 
3   $u.color = \text{GRAY}$ 
4  for each  $v \in G.Adj[u]$           // explore edge  $(u, v)$ 
5      if  $v.color == \text{WHITE}$ 
6           $v.\pi = u$ 
7          DFS-VISIT( $G, v$ )
8   $u.color = \text{BLACK}$               // blacken  $u$ ; it is finished
9   $time = time + 1$ 
10  $u.f = time$ 
```

Figure 4: Depth-First Search Pseudocode

Depth-First Search

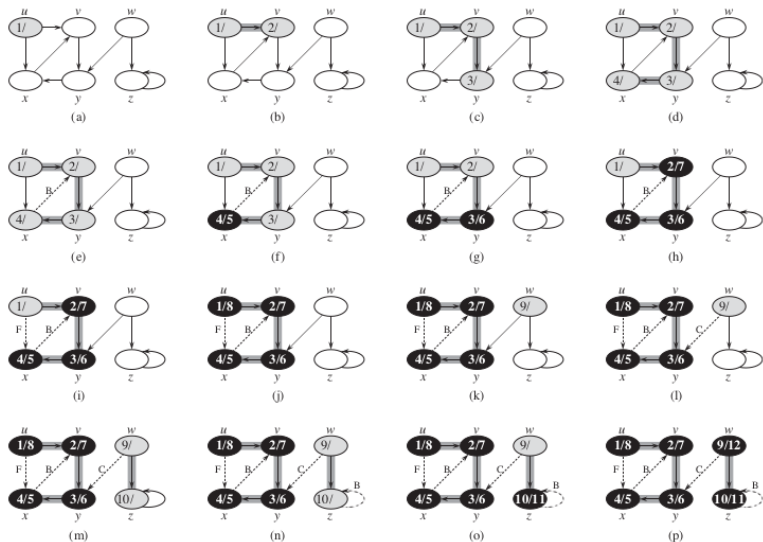


Figure 5: Depth-First Search progress on a directed graph

Path finding algorithms

A* algorithm

A* (pronounced “A-Star”) is a graph traversal and path-finding algorithm. Given a source and a goal node, the algorithm find the shortest-path (with respect to given weights) from source to goal.

Dijkstra algorithm

Dijkstra algorithm solves the single-source shortest-paths problem on a weighted directed graph for the case in which all weights are non-negative.

Path-finding

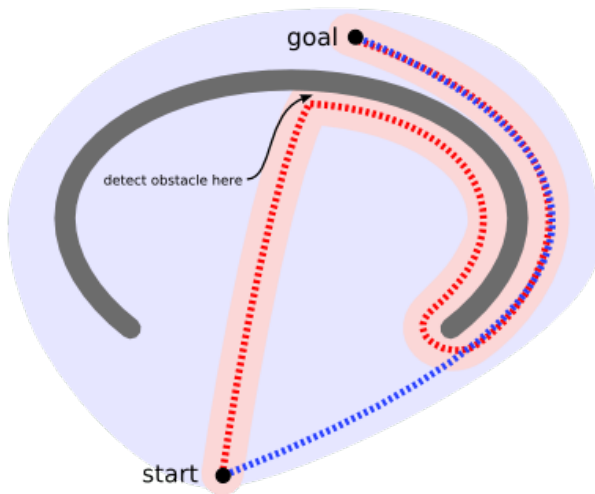


Figure 6: Example path-finding situation

Path-finding

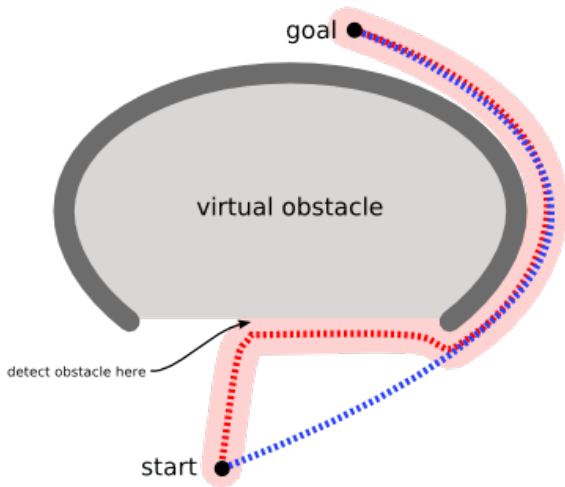


Figure 7: Example path-finding situation

A* Algorithm History

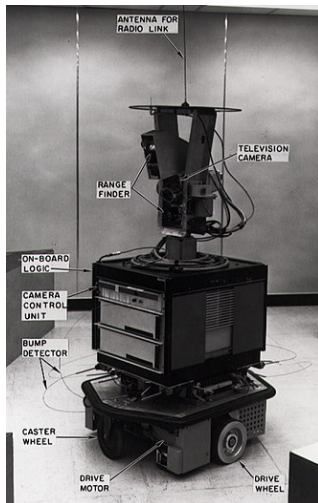


Figure 8: A* was invented by researchers working on Shakey the Robot's path planning.

References



https://en.wikipedia.org/wiki/A*_search_algorithm



<http://theory.stanford.edu/~amitp/GameProgramming/AStarCompa>