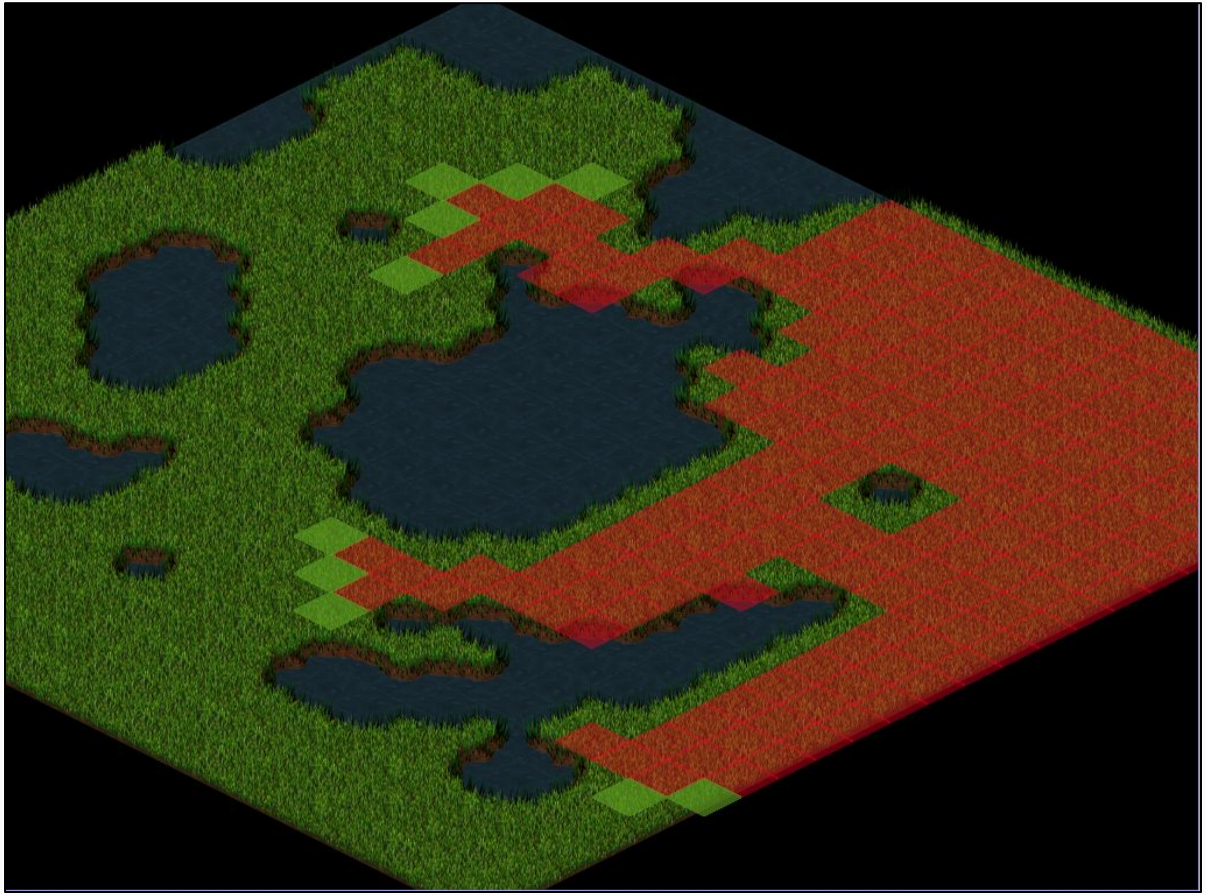
Game Development

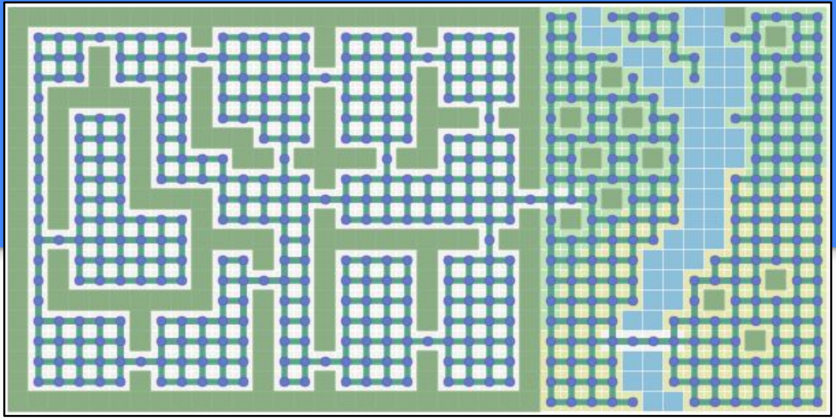
Introduction to Pathfinding

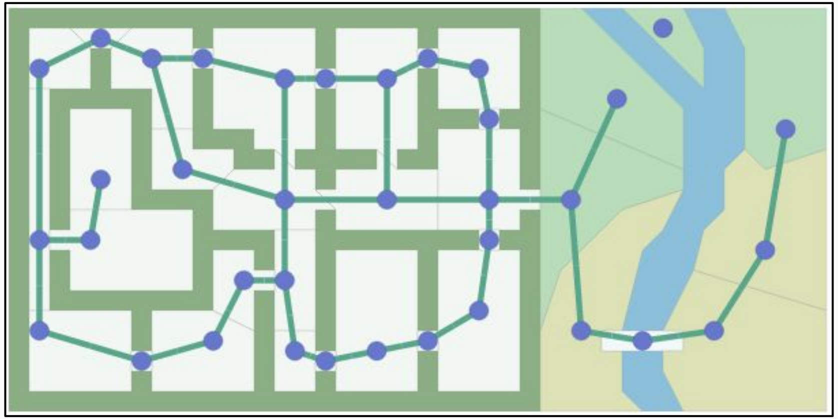
Solution



Navigation meshes

- For navigating we abstract a graph
- Graph could be regular or irregular
- They are dealt in the same way
- Irregular are simpler/faster
- … but are hand made
- We will use regular (grids) for simplicity

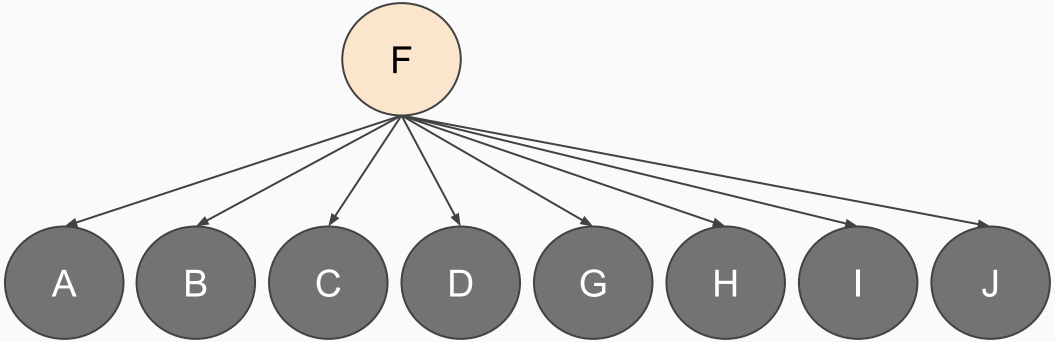




Navigation Mesh -> Tree

- We will apply it to regular grids for visualization:

A	B	C
D	F	G
H	I	J

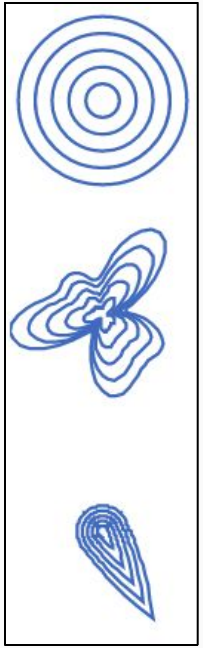


Navigation Algorithms: BFS

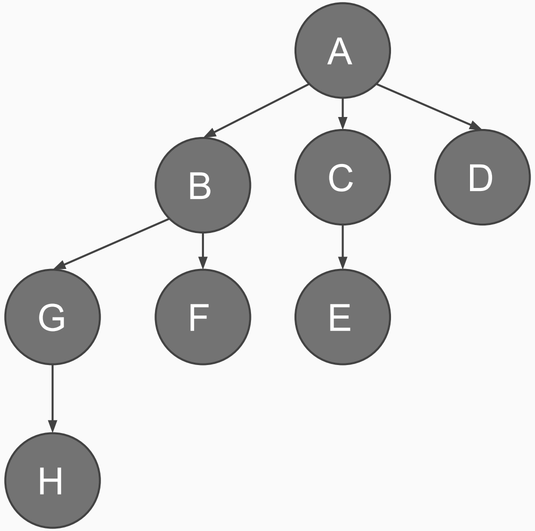
Breadth First Search explores equally in all directions.

Dijkstra is like BFS but favors lower cost nodes.

A* is like Dijkstra but favor nodes closer to a single destination:



Breadth First Search vs Deep First Search

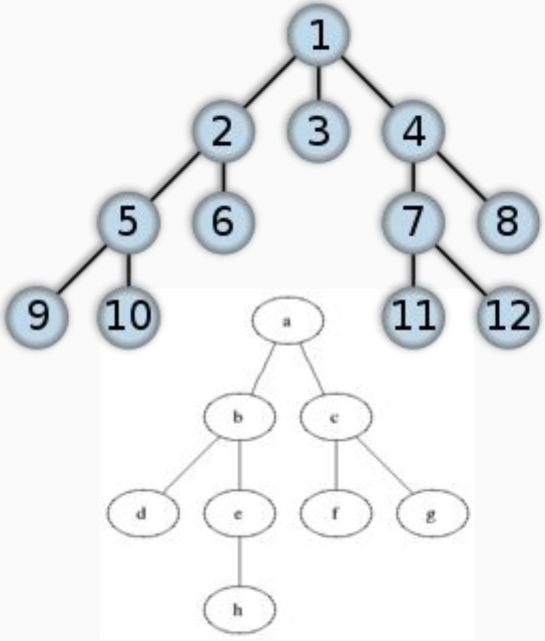


DFS: A,B,G,H,F,C,E,D

BFS: A,B,C,D,G,F,E,H

Breadth First Search or BFS

- It is the simplest pathfinding algorithm
- Method for generic search in a tree/graph
- Explores all child/neighbors before moving on
- Opposite of Depth First algorithms



Iterative Breadth First Search

```
frontier = Queue()
frontier.put(start )
visited = {}
visited[start] = True

while not frontier.empty():
   current = frontier.pop()
   for next in graph.neighbors(current):
    if next not in visited:
         frontier.push(next)
         visited[next] = True
```

BFS in action

A	B	C
D	F	G
H	I	START

Frontier	Visited

BFS in action: STEP 0

A	B	C
D	F	G
H	I	START *Step 0*

Add START to both frontier and visited

Frontier	Visited
START	START

BFS in action: STEP 1

A	B	C
D	F	G *Step 1*
H	I *Step 1*	START *Step 0*

POP START and add all neighbours to frontier and visited

Frontier	Visited
G	START
I	G
	I

BFS in action: STEP 2

A	B	C *Step 2*
D	F *Step 2*	G *Step 1*
H	I *Step 1*	START *Step 0*

POP G and add all ***not-visited*** *neighbours to frontier and visited*

Frontier	Visited
I	START
C	G
F	I
	C
	F

BFS in action: STEP 3

A	B	C *Step 2*
D	F *Step 2*	G *Step 1*
H *Step 3*	I *Step 1*	START *Step 0*

*POP I and add all **not-visited** neighbours to frontier and visited*

Frontier	Visited
C	START
F	G
H	I
	C
	F
	H

BFS in action: STEP 4

A	B *Step 4*	C *Step 2*
D	F *Step 2*	G *Step 1*
H *Step 3*	I *Step 1*	START *Step 0*

*POP C and add all **not-visited** neighbours to frontier and visited*

Frontier	Visited
F	START
H	G
B	I
	C
	F
	H
	B

BFS in action: STEP 5

A	B *Step 4*	C *Step 2*
D *Step 5*	F *Step 2*	G *Step 1*
H *Step 3*	I *Step 1*	START *Step 0*

*POP F and add all **not-visited** neighbours to frontier and visited*

Frontier	Visited
H	START
B	G
D	I
	C
	F
	H
	B
	D

BFS in action: STEP 6

A	B *Step 4*	C *Step 2*
D *Step 5*	F *Step 2*	G *Step 1*
H *Step 3*	I *Step 1*	START *Step 0*

*POP H and add all **not-visited** neighbours to frontier and visited (there is none!)*

Frontier	Visited
B	START
D	G
	I
	C
	F
	H
	B
	D

BFS in action: STEP 7

A *Step 7*	B *Step 4*	C *Step 2*
D *Step 5*	F *Step 2*	G *Step 1*
H *Step 3*	I *Step 1*	START *Step 0*

*POP B and add all **not-visited** neighbours to frontier and visited*

Frontier	Visited
D	START
A	G
	I
	C
	F
	H
	B
	D
	A

BFS in action: STEP 8

A *Step 7*	B *Step 4*	C *Step 2*
D *Step 5*	F *Step 2*	G *Step 1*
H *Step 3*	I *Step 1*	START *Step 0*

POP D and add all ***not-visited*** *neighbours to frontier and visited (there is none!)*

Frontier	Visited
A	START
	G
	I
	C
	F
	H
	B
	D
	A

BFS in action: STEP 9

A *Step 7*	B *Step 4*	C *Step 2*
D *Step 5*	F *Step 2*	G *Step 1*
H *Step 3*	I *Step 1*	START *Step 0*

*POP A and add all **not-visited** neighbours to frontier and visited (there is none!)*

Frontier	Visited
	START
	G
	I
	C
	F
	H
	B
	D
	A

BFS in action: STEP 10

A *Step 7*	B *Step 4*	C *Step 2*
D *Step 5*	F *Step 2*	G *Step 1*
H *Step 3*	I *Step 1*	START *Step 0*

We finish since frontier is empty

Frontier	Visited
	START
	G
	I
	C
	F
	H
	B
	D
	A

TODO 1

“If frontier queue contains elements, pop() one and calculate its 4 neighbors”

- We are doing ONE iteration of the BFS expand at a time (like solution.exe)
- Frontier queue is already created and reset to the first element **ResetBFS()**
- Remember that all points are in **tile coordinates**

```
bool Queue::Pop(tdata& item)
```

TODO 2

“For each neighbor, if not visited, add it to the frontier queue and visited list”

- The list already contains a find() method to search for elements
- Just add to visited list and frontier queue the new unexplored node
- You may test the game already, should see a forever expanding search

```
int List::find(const tdata& data)
```

TODO 3

“return true only if x and y are within map limits and the tile is walkable (tile id 0 in the navigation layer)”

- This method makes sure we never get out of the map
- And that we do not visited non-walkable nodes!
- Mind that navigation layer is the second one in this map!
- You need to go back to *PropagateBFS()* and add the walkability check

Homework (check an interesting video here)

- We only did BFS expanding, not really pathfinding
- Try stopping when you reach certain node
- Try remembering from which tile you came from each visited node
- Then reconstruct the path from destination to source

Really good article about the three basic navigation methods here