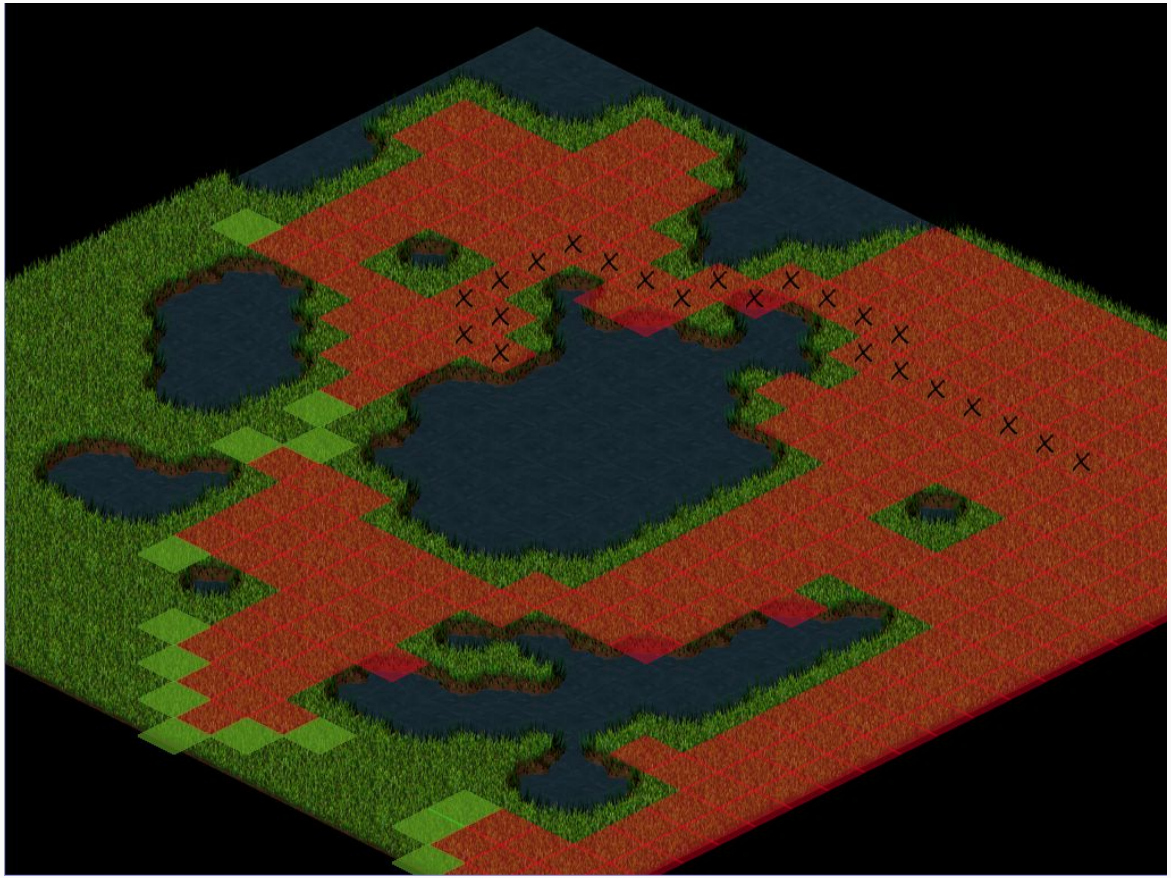
Game dev: BFS to Dijkstra

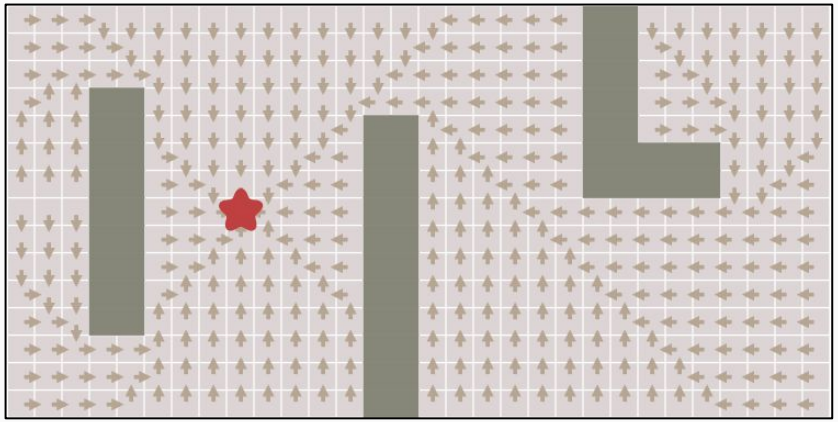
Ricard Pillosu - UPC

Solution BFS



Creating a path out of BFS

- BFS only navigates the whole map
- It’s actually calculating the path to **all** other nodes
- Let’s keep on “the node I come from”
- It should give us a map like



BFS in action

A	B	C
D	F	G
H	I	START

We add new array parallel to visited to record were every tile comes from called “breadcrumbs”

Frontier	Visited	Breadcrumbs

BFS in action: STEP 0

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

Add START to both frontier and visited. Adding START to breadcrumbs is optional.

Frontier	Visited	Breadcrumbs
START	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	Breadcrumbs
G	START	START
I	G	START
	I	START

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	Breadcrumbs
I	START	START
C	G	START
F	I	START
	C	G
	F	G

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	Breadcrumbs
C	START	START
F	G	START
H	I	START
	C	G
	F	G
	H	I

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	Breadcrumbs
F	START	START
H	G	START
B	I	START
	C	G
	F	G
	H	I
	B	C

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	Breadcrumbs
H	START	START
B	G	START
D	I	START
	C	G
	F	G
	H	I
	B	C
	D	F

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	Breadcrumbs
B	START	START
D	G	START
	I	START
	C	G
	F	G
	H	I
	B	C
	D	F

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	Breadcrumbs
D	START	START
A	G	START
	I	START
	C	G
	F	G
	H	I
	B	C
	D	F
	A	B

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	Breadcrumbs
A	START	START
	G	START
	I	START
	C	G
	F	G
	H	I
	B	C
	D	F
	A	B

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	Breadcrumbs
	START	START
	G	START
	I	START
	C	G
	F	G
	H	I
	B	C
	D	F
	A	B

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	Breadcrumbs
	START	START
	G	START
	I	START
	C	G
	F	G
	H	I
	B	C
	D	F
	A	B

BFS in action: PATH 1

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 reconstruct the path backwards.
We arrived to A from B

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

BFS in action: PATH 2

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 arrived from B from C

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

BFS in action: PATH 3

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 arrived from C from G

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

BFS in action: PATH 4

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 arrived from G from START

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

BFS in action: PATH 4

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*

Our final path is:
START > G > C > B > A

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

TODO 1

“Record the direction to the previous node with the new list "breadcrumps"

- The list **breadcrumps** is already created
- Note the change of name of few functions
- For each neighbor, remember that you come from “current” cell
- Just one line of code somewhere in the method

Reconstructing the path

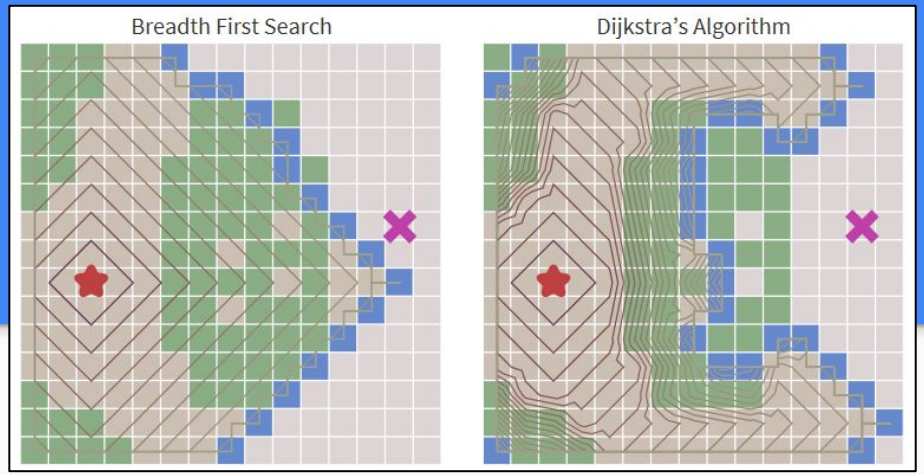
```
current = goal
path = [current]

while current != start:
    current = came_from[current]
    path.append(current)

path.append(start)
```

TODO 2

“Follow the breadcrumps to goal back to the origin add each step into "path" dyn array (it will then draw automatically)"

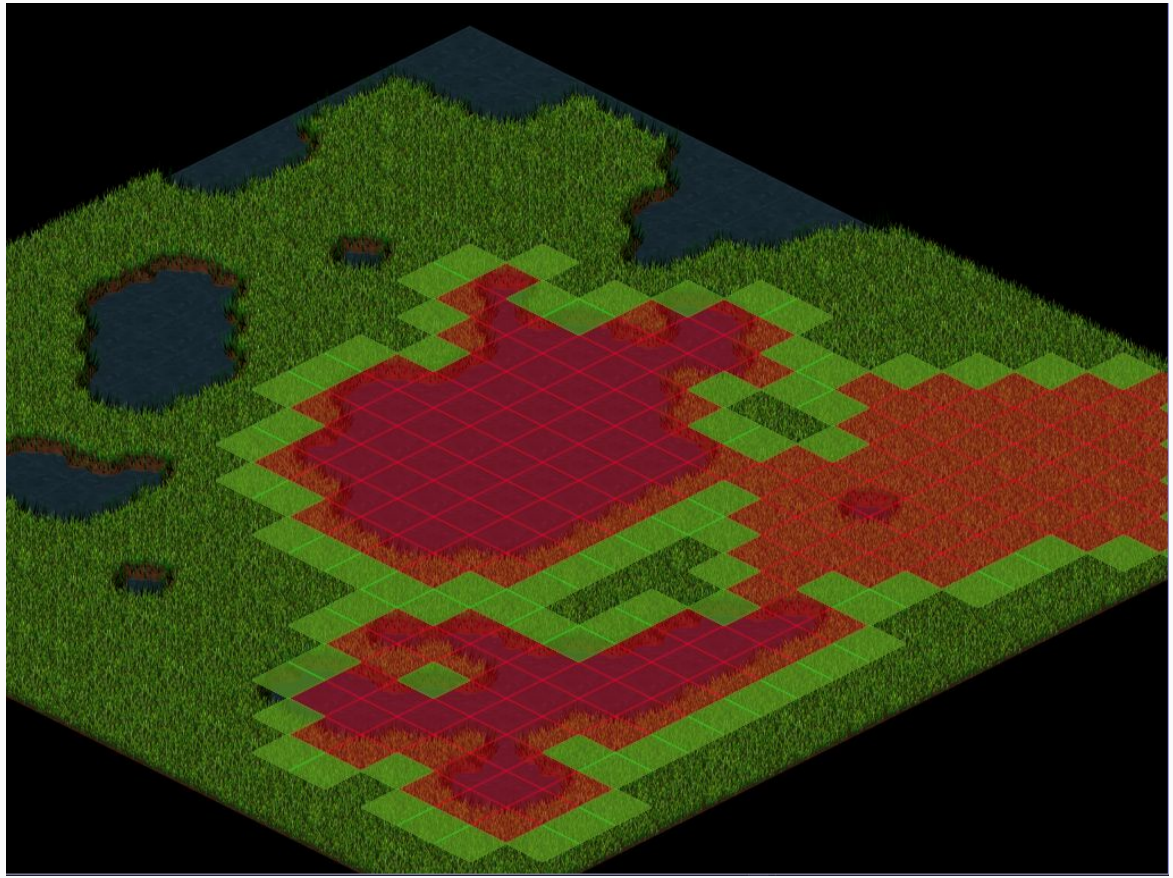
- The dyn array for path already exists
- If filled, it will draw “X” on each tile
- The mouse position when clicked is already calculated for you

Dijkstra



- Expands in all directions like BFS
- But will prefer low cost nodes
- We will simulate that water has a lower cost
- Check solution.exe keys j & k
- We could re-visit a node more than once
- We need to write down in each cell the latest accumulated score

Solution Dijkstra (press J/K to expand)



Dijkstra

```
frontier = PriorityQueue()
frontier.put(start, 0)
came_from = {}
cost_so_far = {}
came_from[start] = None
cost_so_far[start] = 0

while not frontier.empty():
   current = frontier.get()

   for next in graph.neighbors(current):
      new_cost = cost_so_far[current] + graph.cost(current, next)
      if next not in cost_so_far or new_cost < cost_so_far[next]:
        cost_so_far[next] = new_cost
        frontier.put(next, new_cost)
        came_from[next] = current
```

TODO 3

“Taking BFS as a reference, implement the Dijkstra algorithm use the 2 dimensional array "cost_so_far" to track the accumulated costs on each cell (is already reset to 0 automatically)"

- Frontier is already a **priority** queue
- Be sure to understand MovementCost() method
- Cost_so_far is just a big fat array, *just* ok for now :)

Homework

- Try stopping when you reach certain node
- Experiment with an orthographic map with different tile weights

Really good article about the three basic navigation methods here