Dijkstra Maps in Roguelikes

Jeremy Mates

March 13, 2020

Simple Pathfinding

Consider the following map, wherein there are three velociraptors `R` surrounding the player `@` on an open field[Munroe(2006)]. How can the raptors reach the player?

```
......R
.R.....
.......
...@...
.......
.......
..R....
```

One solution to this problem is the so-called Dijkstra Map[Pender(2010)]. A map this small can easily be worked out on graph paper; set the location of the player to the value `0`. Then in the squares adjacent to that point write the next highest number, `1`.

```
.......
.......
...1...
..101..
...1...
.......
.......
```

And so forth until the map is filled. This is a Dijkstra Map (with Manhattan distances).

```
6543456
5432345
4321234
3210123
4321234
5432345
6543456
```

Note that diagonal moves were not considered–only those points North South East or West of the given square. This has ramifications on more complicated maps that contain obstacles. Meanwhile, each velociraptor can use this map to find the player by walking "downhill" to `0`, or can flee by picking a cell with a higher value.

Obstacles

Many maps will be more complicated than the above and will contain obstacles, typically walls, or lava, or there may be monsters that can pass through walls but not across holy water–specters perhaps–or anything else you can imagine. Let us for now only consider physical walls, represented using # as is typical for roguelikes.

```
R....
.###.
.#@..
.#.##
.#.#.
...#.
```

A Dijkstra Map for this might look like

8	7	6	5	4
9	-1	-1	-1	3
8	-1	0	1	2
7	-1	1	-1	-1
6	-1	2	-1	?
5	4	3	-1	?

where the impassable walls are represented by `-1`. One may use `Inf` or an object for such cells, but I assume here an integer or fixnum array. Another option would be to leave walls at the maximum integer or `MOST-POSITIVE-FIXNUM` value and to ignore them while calculating the costs. Note the two squares lower right that have no path to the player; these after calculation will typically remain at the maximum integer value the field was filled with before the pathfinding pass. This map by the way was made with my `Game::DijkstraMap` module[Mates(2018b)].

```
#!/usr/bin/env perl
use strict;
use warnings;
use Game::DijkstraMap;
my $dm = Game::DijkstraMap->new( str2map => <<'EOM' );
.....
.###.
.#x..
.#.##
.#.#.
...#.
EOM
print $dm->to_tsv;
```

Pathfinding on this map must ignore `-1` and look for values equal to or greater than `0` when routing to a goal. Here, the raptor should move horizontally to reach the player in as few moves as possible.

Multiple Goals

A Dijkstra Map may contain multiple goals; pathfinding will find the (or a) nearest goal. Crabs c for example may wish to retreat to water ~ when threatened by the player.

```
..~~......~.....
.~~~~......~~...
...~~..c..~~....
..........c.....
...~~...........
..~~...c........
...~~...........
.....~.........@
```

2	1	0	0	1	2	3	4	2	1	0	1	1	2	3	4
1	0	0	0	0	1	2	3	4	2	1	0	0	1	2	3
2	1	1	0	0	1	2	3	2	1	0	0	1	2	3	4
3	2	2	1	1	2	3	4	3	2	1	1	2	3	4	5
4	3	1	0	0	1	2	3	4	3	2	2	3	4	5	6
2	1	0	0	1	2	3	4	5	4	3	3	4	5	6	7
3	2	1	0	0	1	2	3	4	5	4	4	5	6	7	8
4	3	2	1	1	0	1	2	3	4	5	5	6	7	8	9

This map however does not consider other crabs–can two occupy the same square, or will one need to find a longer route to a free water cell?–nor the influence of the player; a monster may realistically have multiple desires: get to water while also avoiding the player. This can be solved in various ways; moves could be ranked by utility value–corners might be scored poorly for a fleeing monster (see e.g. "Behavioral Mathematics for Game AI"[Mark(2009)] for ideas)–or by using a combination of maps.

Combining Dijkstra Maps

Multiple maps may be added together and the combined score used to move an entity towards or away from goals. Suppose that some Knight[Cervantes(2003)] seeks to make justice upon a fleeing goblin. The monster, however, has been cornered in a room so simple "move to the highest cost cell" code will leave the monster trapped in a corner. There should be, in this condition, some means for the monster to slip past the player.

```
######
#....#
#.g.@.
#....#
######
```

A monster whose state is fleeing needs to be anywhere else; this can be represented by a map with the monster as the goal, higher values thus being more desirable. This map can be combined with the map centered on the player–and also any of their allies, if necessary.

Since the player map will have undesirably low values close to the player, the resulting map should give a path of higher values for a monster to flee along step by step.

```
######   ######   ######
#2123#   #4321#   #6444#
#1g123 + #321@1 = #42224
#2123#   #4321#   #6444#
######   ######   ######
```

The best single move in this case would be for the goblin to move to one of the corners (combined cost 6) and for the player to step towards them, so

```
######   ######
#g...#   #....#
#.....   #..@..
#....#   #....#
######   ######
 g123     3212     3335
 11345 +  21@12 =  33357
 2345     3212     5557
```

The goblin is now stuck on an island (cost 3), so the player advances again,

```
######   ######
#g...#   #....#
#.....   #.@...
#....#   #....#
######   ######
 g123     2123     2246
 12345 +  1@123 =  22468
 2345     2123     4468
```

This is not productive as even with combined maps the goblin still gets stuck. Perhaps if we give more weight to the values of one map or the other, say multiplying the goblin costs by 2?

```
######   ######
#g...#   #....#
#.....   #.@...
#....#   #....#
######   ######
 g123     2123     2369
 11234 +  1@123 =  347AD
 2223     2123     67AD
  * 2
```

This looks promising, though the best move is to move into the player, which might be bad. The easiest solution would be to treat the player (and any allies) as impassable squares on the goblin's flee map

```
######   ######
#g...#   #....#
#.#...   #.@...
#....#   #....#
######   ######
 g123     2123     2369
 1#234 +  1@123 =  3#7AD
 2223     2123     67AD
  * 2
```

Better. The goblin might here roll unlucky and move down instead of right towards the exit, but that should create conditions that will cause them to try to move past the player:

```
######   ######
#....#   #....#
#g#...   #.@...
#....#   #....#
######   ######
 1234     2123     458B
 1#456 +  1@123 =  1#9CF
 1234     2123     458B
  * 2
```

However the monster may move back and forth between the back wall and an edge of the room instead of only moving towards the door if the RNG dictates that, but at least it is not always stuck, and it may roll well and escape.

Another method would be to construct a line or path to the highest value on the monster's map instead of only looking for the single best move in any given turn; this would avoid the possibility of the monster getting stuck over multiple turns though may be more expensive to calculate and store. The monster's map also need not comprise the entire field; it may only need to be slightly larger than the player's field of vision.

Different or non-integer weights can be used, or the maps could be subtracted instead of being added, though experimentation shows that adding the maps and applying a small positive integer weight to the monster flee map better avoids problems such as the monster islanding itself in a corner.

Play testing with different approaches and map weights will likely be necessary. Level design may also be a factor, 3x3 rooms give an obvious path for the player to try to box a monster in with; larger rooms or rooms with multiple exits or rooms with rounded corners should create different weights for the monsters to flee along.

The Diagonal

Costs in the above maps have been done without consideration for diagonal moves. Consider the following map, where `x` is the goal.

```
@#..
#...
...x
```

?	-1	3	2
-1	3	2	1
3	2	1	0

The player here cannot pathfind to the goal as the diagonal move was not considered by the 4-way algorithm that only consults cells North South East and West. Various roguelikes (Angband, Dungeon Crawl Stone Soup) permit such diagonal moves, so will need to use an 8-way algorithm when constructing a Dijkstra Map. Other roguelikes (Brogue, POWDER) may deny such diagonal moves so can use the 4-way algorithm. This choice also influences level design. 4-way and 8-way roguelikes require rather different diagonal corridors:

```
8-way    @####   4-way    @.###
corridor #.###   corridor #..##
         ##x##            ##x##
```

Brogue (like rogue) is complicated in that it allows some diagonal moves. A 4-way Dijkstra Map algorithm can be used with 8-way motion provided 4-way moves are possible to everywhere that must be reached. Diagonal moves in such a case exist as shortcuts–moving diagonally along the above 4-way corridor (which Brogue does not permit, nor POWDER unless one is ploymorphed into a grid bug).

Diagonal Maps

8-way maps typically assign equal costs to all adjacent squares. The original raptor map instead with Chebyshev distances:

```
......R  3333333
.R.....  3222223
.......  3211123
...@...  3210123
.......  3211123
.......  3222223
..R....  3333333
```

This while traditional for roguelikes is not actually correct; diagonals under euclidean geometry should instead use $\sqrt{x^2 + y^2}$ or $\sqrt{2}$ instead of 1 for the closest diagonal. Various roguelikes are actually non-euclidean: Brogue and Dungeon Crawl Stone Soup apply the same cost to a move in any direction, diagonal or otherwise. Anyways! Our original diagonal map that stumped the 4-way map under (non-euclidean) 8-way is:

```
@#..
#...
...x
```

3	-1	2	2
-1	2	1	1
3	2	1	0

And the player can path to the goal.

Not Just for Animates

Dijkstra Map can be used for other purposes. Given a map with two rooms in it, how to connect them?

```
####################
###########XXXXXX###
###########X....X###
###########X....X###
#XXXXXX####XXXXXX###
#X....X#############
#X....X#############
#X....X#############
#XXXXXX#############
####################
```

Place a goal anywhere in the wall space, and then considering only walls # as passable construct a Dijkstra Map with the goal as the destination, here shown partially complete.

```
           XXXXXX
           X....X8
      DCBA98X....X7
  XXXXXXA987XXXXXX6
  X....X98765432345
  X....X87654321234
  X....X76543210123
  XXXXXX 765432123
```

Then pick a random door location in each room, and pathfind from each door to the goal, drawing a corridor along the way.

```
####################
###########XXXXXX###
###########X....+.##
#####...###X....X.##
#XXXX+X.###XXXXXX.##
#X....X...#######.##
#X....X##.........##
#X....X#######..####
#XXXXXX#############
####################
```

This will run into complications if rooms are adjacent or worse two rooms block a third from reaching a goal (there should likely be one goal per unconnected wall space in a map) though there are various solutions to these challenges, such as pathfinding through rooms or using additional code to find and link up adjacent rooms.

Super Dimensional Dijkstra Maps

We need not confine ourselves to two, or even three dimensions; Dijkstra Maps can be built in arbitrary numbers of dimensions (memory requirements, implementation demands, and sanity permitting). The following is a four-dimensional map with an implementation[Mates(2018a)] that does not consider diagonal moves legal.

```
% sbcl --noinform --load dijkstramap.lisp
* (setf *dimap-cost-max* 99)

99
* (defparameter level
    (make-array '(3 3 3 3)
                :initial-contents
                '((((99 -1 99) (-1 99 -1) (99 -1 99))
                   ((-1 99 99) (99 99 99) (99 99 -1))
                   ((99 99 99) (99 99 99) (99 99 99)))
                  (((-1 99 99) (99 99 99) (99 99 -1))
                   ((99 99 99) (99  0 99) (99 99 99))
                   ((99 99 99) (99 99 99) (99 99 99)))
                  (((99 99 99) (99 99 99) (99 99 99))
                   ((99 99 99) (99 99 99) (99 99 99))
                   ((99 99 99) (99 99 99) (99 99 99))))))

LEVEL
* (dimap-calc level)

4
* level

#4A((((99 -1 4) (-1 2 -1) (4 -1 99))
     ((-1 2 3) (2 1 2) (3 2 -1))
     ((4 3 4) (3 2 3) (4 3 4)))
    (((-1 2 3) (2 1 2) (3 2 -1))
     ((2 1 2) (1 0 1) (2 1 2))
     ((3 2 3) (2 1 2) (3 2 3)))
    (((4 3 4) (3 2 3) (4 3 4))
     ((3 2 3) (2 1 2) (3 2 3))
     ((4 3 4) (3 2 3) (4 3 4))))
*
```

Productive uses for such maps are left as an exercise to the reader.

References

[Cervantes(2003)] Cervantes. *The History and Adventures of the Renowned Don Quixote*. University of Georgia Press, 2003. ISBN 9780820324302.

[Mark(2009)] Dave Mark. *Behavioral Mathematics for Game AI*. Course Technology Cengage Learning, 2009. ISBN 1584506849.

[Mates(2018a)] Jeremy Mates. dijkstramap.lisp. `https://github.com/thrig/ministry-of-silly-vaults/blob/master/dijkstramap.lisp`, 2018a. [Online; accessed 20-September-2018].

[Mates(2018b)] Jeremy Mates. Game::DijkstraMap. `https://metacpan.org/pod/Game::DijkstraMap`, 2018b. [Online; accessed 20-September-2018].

[Munroe(2006)] Randall Munroe. Substitute. `https://www.xkcd.com/135/`, 2006. [Online; accessed 20-September-2018].

[Pender(2010)] Pender. The Incredible Power of Dijkstra Maps. `http://www.roguebasin.com/index.php?title=The_Incredible_Power_of_Dijkstra_Maps`, 2010. [Online; accessed 20-September-2018].