

Reinforcement Learning 2: Grid World

Eckel, TJHSST AI2, Spring 2024

Background & Explanation

I haven't had time to put this all in writing, but to briefly summarize the lecture you've hopefully seen, every state/action pair gets a q-value. To get all q-values correct, we need to satisfy this equation on all of them:

For any valid state/action pair, $q(s, a) = \text{immediate reward of taking } a + \max \text{ of resulting state's } q \text{ values}$

And it turns out repeatedly applying this equation (using zero as the max q of any terminal state) will actually converge to the correct set of q values! Just keep looping over every q value and changing it according to that equation until you get a whole loop through every q value where nothing changes.

You'll apply this to grid world, a basic game where there is a grid of locations, the player can move to any adjacent location at each step, some squares are goals, and we want to reach the goal with as little "effort" as possible.

As a reminder, be sure to code this as separate move and update functions – the move function takes the current state & possibly a policy, and returns "up", "down", "left", or "right". The update function takes the current state and the move and returns the reward and the new environment.

Round 1: Basic Rules

Any move gives an immediate reward of -1. Landing on a goal square ends the game.

Your input is:

- n (side length of the square)
- a list of goal squares

Your output is:

- a grid of the max q value at each state

For example:

- 4
- [0, 15]

...would produce:

x	-1	-2	-3
-1	-2	-3	-2
-2	-3	-2	-1
-3	-2	-1	x

(Spacing isn't especially important here, just get the numbers right and displayed with each row on a separate line.)

Round 2: Quicksand

Add quicksand pits. Leaving a quicksand pit is much harder than the usual move; leaving a quicksand pit gives a reward of -100.

This should change the **reward** part of your update function.

Your input is:

- n (side length of the square)
- a list of goal squares
- a list of quicksand pits

Your output is:

- a grid of the max q value at each state

For example:

- 4
- [0]
- [1, 5, 9]

...would produce:

x	-100	-8	-9
-1	-101	-7	-8
-2	-102	-6	-7
-3	-4	-5	-6

Round 3: Dragons

Uh oh, the goal squares have been occupied by dragons. Landing on a goal square gives a reward of -1,000,000 instead of -1 or -100 now, because the dragon eats you. (This does end the game as you'd expect.)

Thankfully, there are also magical items scattered around the grid, and if you get enough of them, you can kill the dragon, at which point landing on a goal square gives -1 or -100 as it normally would, depending on whether you're moving from a quicksand pit or a normal square. You get a magical item when you move OFF a square where a magical item is located.

This should change **your list of possible states**, as well as the **update** function to move between them, and the **reward** function to take into account possibly getting eaten by the dragon. Now, each state must contain the player's location on the grid as well as a separate Boolean value for each distinct magical item the player may or may not have. So, a single key in your dictionary of states might be (5, True, True, False) – this is the game state of being at location 5 owning the first and second magic items but not the third. Moving off of a location that contains a magical item switches to a state where that item's Boolean is set to True. For example, if square 5 contains the third magical item, moving left from the state above would move to (4, True, True, True).

Your input is:

- n (side length of the square)
- a list of goal squares (all now occupied by dragons)
- a list of quicksand pits
- the number of magical items necessary to defeat the dragon (which may be less than or equal to the total number of items available)
- a list of the locations of all magical items (which may be on normal squares or quicksand squares, but won't be on goal/dragon squares)

Your output is:

- a grid of the max q value of the game state at each locations **where all item Booleans are set to False** (in other words, you do not need to display q values for any state where the agent has one or more magical items, though of course you'll need to get those right within your code to get correct answers for the states without items)

For example:

- 4
- [0]
- [1, 5, 9]
- 1
- [3]

...would produce:

x	-110	-10	-9
-17	-111	-11	-10
-16	-112	-12	-11
-15	-14	-13	-12

Round 4: Warps

Dragons, as we all know, are jerks. Now, the dragons have picked a few locations, and anyone who tries to move to that location is *sometimes* picked up and moved to a different location. Call this a warp.

A warp comes in the form of a tuple, for instance (4, 9, 0.1) would mean that any attempt to move to location 4 has a 10% chance of being taken by a dragon and dropped on location 9 instead. (If there is a magic item on 4, the player would not pick up that item in this case.)

This should change your **q value update step** to take into account this probability. For example, if the above warp is given, then any action that normally moves to square 4 would need to update like so:

$q(s, a) = \text{immediate reward} + (0.9 * \text{max } q \text{ of location 4 with same magic items} + 0.1 * \text{max } q \text{ of location 9 with same magic items})$

Your input is the same as Round 3, except for now it also has a list of tuples defining some number of warps. There will not be two different warps originating at the same location. If a warp ends on a location that also has a warp, that warp doesn't apply. (For instance, if the above warp is given and also the warp (9, 10, 0.1) is given, then someone trying to move to square 4 might end up at square 9, and someone trying to move to square 9 might end up at square 10, but someone trying to move to square 4 does not have a small chance to end up at square 10.)

I'll update this with a sample output soon; I need to do some code tweaking.

Specification

Submit a **single python script** to the link on the course website.

You'll get command line arguments like so; use `ast.literal_eval` to convert each to the appropriate data type automatically.

- `sys.argv[1]` converts to an int – n, the side length of the square
- `sys.argv[2]` converts to a list – locations of goal/dragon squares
- `sys.argv[3]` converts to a list – quicksand pits
- `sys.argv[4]` converts to an int – the number of magic items necessary to kill a dragon
- `sys.argv[5]` converts to a list – locations of magic items
- `sys.argv[6]` converts to a list of tuples – each tuple represents a warp, as defined above

You should output, with each row on a separate line, the max q value of the game state at each location with no magic items.

This assignment is **complete** if:

- You follow the instructions on the submission form to format your submission properly.
- Your script works as described.