TDDE15 Lab 3

vgustafs97

October 2020

1 Q-Learning

1.1 Environment A

1.1.1 Greedy Policy

```
GreedyPolicy <- function(x, y){
    action = which(q_table[x,y,]==max(q_table[x,y,]))
    if(length(action)>1){
        action = sample(action, 1)
    }
}
```

1.1.2 Epsilon Greedy Policy

```
EpsilonGreedyPolicy <- function(x, y, epsilon){
    if (runif(1)>epsilon){
        return(GreedyPolicy(x,y))
    }else{
        return(sample(1:4))
    }
}
```

1.1.3 Q-Learning Algorithm

I implement the Q-Learning algorithm with update rules according to:

$$Q(S,a) \leftarrow Q(S,a) + \alpha * (r + \gamma \max_a Q(S^{'},a) - Q(S,a))$$

```
q_learning <- function(start_state, epsilon = 0.5,
                alpha = 0.1, gamma = 0.95, beta = 0){
```

```
  current_state = start_state

  repeat{
    # Follow policy, execute action, get reward.

    # Q-table update.
    episode_correction = 0
    action = EpsilonGreedyPolicy(current_state[1],
                                 current_stat[2], epsilon)
    new_state = transition_model(current_state[1],
                                 current_state[2], action=action, beta=beta)

    reward = reward_map[new_state[1],new_state[2]]

    # Calculate correction
    correction = alpha*(gamma * max(q_table[new_state[1],new_state[2],]) +
     reward-q_table[current_state[1],current_state[2],action])

    # Update current Q-table
    q_table[current_state[1],current_state[2],action] <<-
        q_table[current_state[1],current_state[2],action] + correction

    #Sum all corrections
    episode_correction = episode_correction + correction
    current_state = new_state

    if(reward!=0)
      # End episode.
      return (c(reward,episode_correction))

  }
}
```

1.1.4 Run Environment A

$\epsilon = 0.5 \quad \beta = 0 \quad \alpha = 0.1 \quad \gamma = 0.95$

- *What has the agent learned after the first 10 episodes ?*

Figure 1: Environment A 10 Iterations

The agent has learned a basic strategy of avoiding the negative awards by traveling away from it (except for one square), but has not yet explored all possible states so the policies for the last row and right side of the state space remains unchanged. The agent has not yet reached the positive reward so not policy for reaching this state has been found. The agent's current policy will not get it to the goal.

– *Is the final greedy policy (after 10000 episodes) optimal? Why / Why not ?*

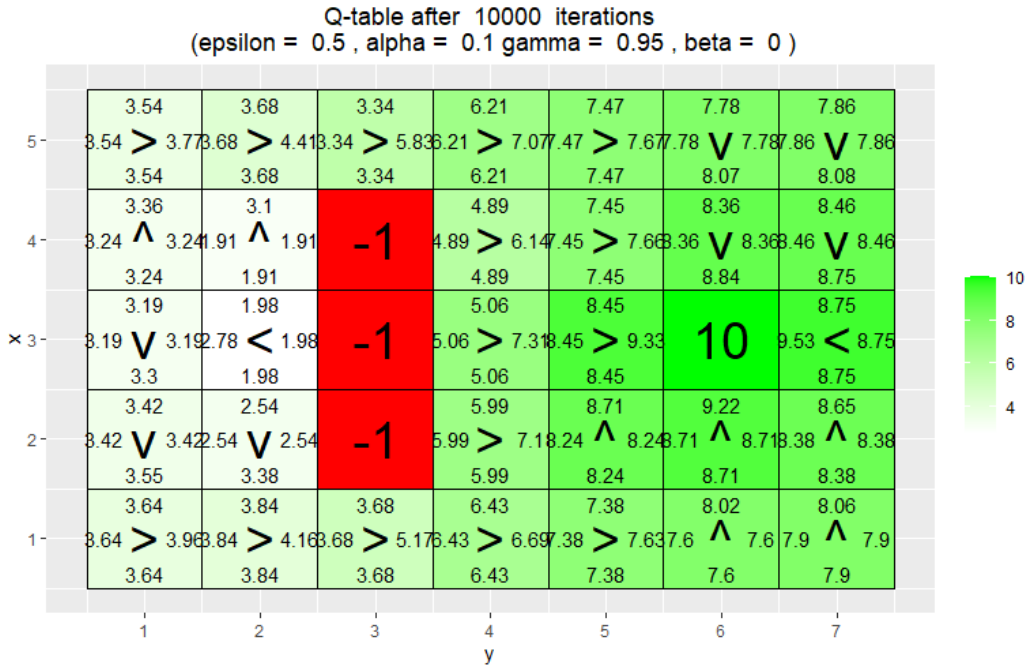

Figure 2: Environment A 10000 Iterations

It is not entirely optimal, an optimal path would have arrows pointing towards the goal at all states (except for the states to the left of the negative rewards). It has also not yet explored most states and have therefor not learned the optimal paths at these states yet.

– Does the agent learn that there are multiple paths to get to the positive reward ? If not, what could be done to make the agent learn this ?

The agent has learned a policy that is close to optimal, with the exception of the square 2,3 as it should point downwards to reach the goal faster. With more iterations, the algorithm should learn this as well. Otherwise, the paths to the goal is optimal as it reaches the goal as fast as it can without traversing the squares with negative rewards.

1.2 Environment B

In Environment B we try different settings for γ (discount rate) and ϵ (chance of randomizing action).

The general behaviour observed was the following: Increasing γ gave larger values to all tiles and gives less of a distance in Q-Value for states further away

from the goal. More importantly, a larger gamma would produce an optimal policy that leads the agent to the reward of 10 instead of 5. This is explained as a higher value of gamma means the agent will discount states further away from the goal less, and thus mean that the agent "thinks long term". A higher epsilon gave faster convergence and would find better policies for more states. This makes sense as the agent explores more states with larger epsilon. An ϵ value of 0.1 would always result in an optimal policy that leads the agent to the reward of 5, even with a large γ value. This is probably because the agent has not explored enough during the episodes to learn that the reward of 10 is a better policy. The agent probably finds the goal of 5 first, and the paths leading towards it are reinforced. Using an ϵ of 0.5 would make the model find the optimal policy that leads it to reward of 10 with $\gamma = 0.95$. This is probably because the model explores more and often finds the 10 reward, and with a large gamma values the reward increase from goal 5 to 10 is larger than the discount factors for the states needed to move from the position of goal 5 to goal 10.

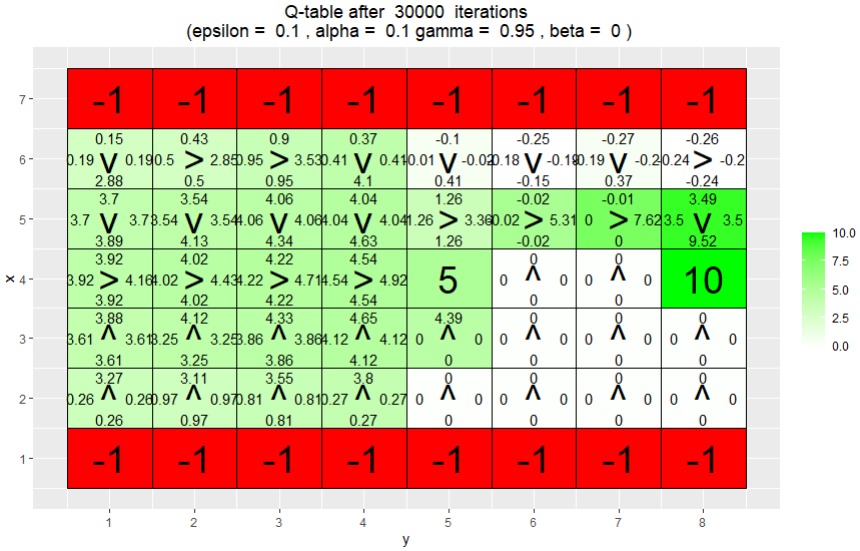

Figure 3: Environment C $\beta = 0$

The model almost finds the goal of 10 here.

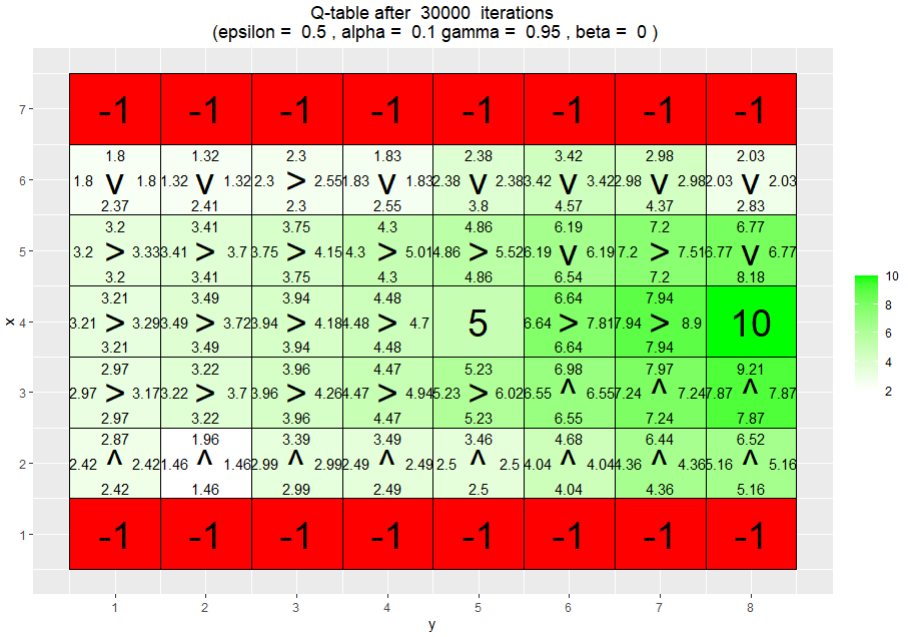

Figure 4: Environment C $\beta = 0$

This is the only parameter configuration that finds the reward of 10 in the optimal policy. This makes sense as the agent is most long term oriented and explores a lot. In comparison.

1.3 Environment C

$\epsilon = 0.5$ $\beta \in (0,\ 0.2,\ 0.4,\ 0.66)$ $\alpha = 0.1$ $\gamma = 0.6$ In this task we vary the chance of the agent slipping to the right or left of their intended action. The start state in the environment is (1,1)

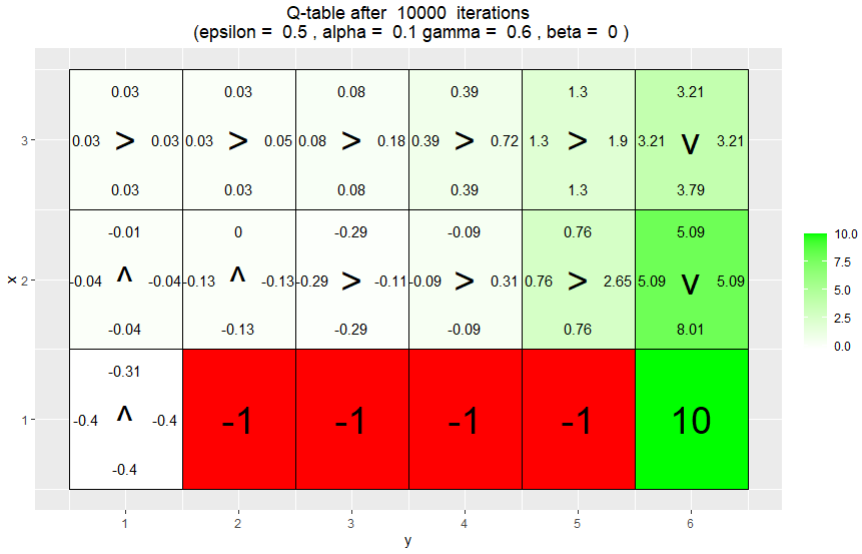

Figure 5: Environment C $\beta = 0$

This configuration almost learns an optimal policy. The optimal in this environment would be to go right at 1,2 however and continue right. The reason the model does not learn this is probably due to the agent having a high chance of stepping into the negative reward because of the exploration rate if it takes this route, and the final reward for this route is therefor not reinforced enough.

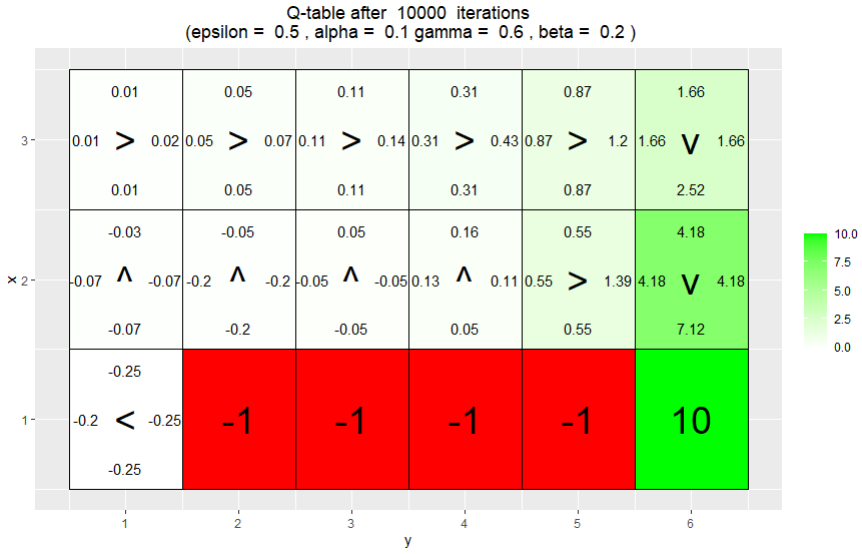

Figure 6: Environment C $\beta = 0.2$

The the chance of slipping into the negative reward when moving right next to these states makes the agent prefer moving at the top, thus if the agent slips down while moving at the top its optimal policy is moving up again. The agent has also learnt a clever trick to prevent it from ever slipping into the negative reward when moving up from its initial position. It's optimal policy at 1,1 is to move to the left, meaning it relies on the slip up to move it upwards.

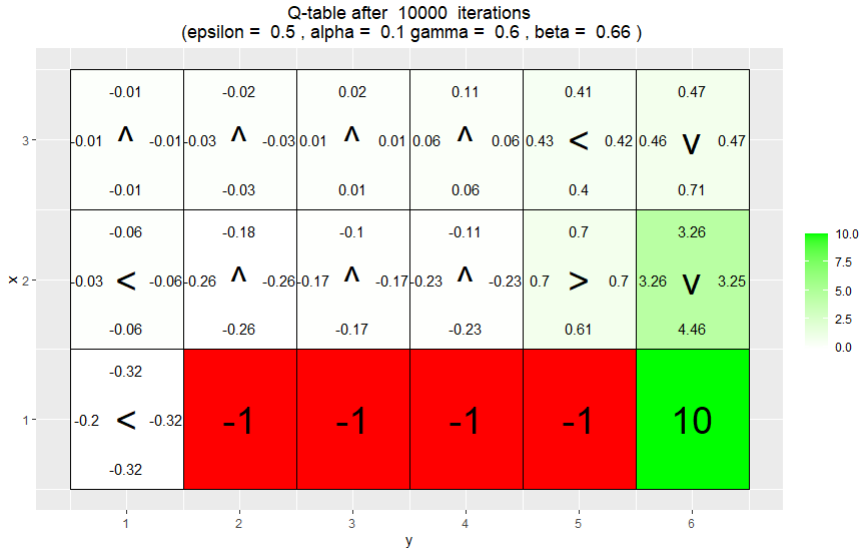

Figure 7: Environment C $\beta = 0.66$

For this beta, the chance of slip up is large enough to make the agent want to always have a zero chance of slipping closer to the negative reward, thus moving opposite the direction of the negative rewards and relying on the slip up to move horizontally. This same tendency could be seen for beta=0.4, though with fewer policies. This means the agent relies entirely on randomness to move to the towards the goal until the last four states.

2 Reinforce

2.1 Environment D

Has the agent learned a good policy? Why / Why not ?

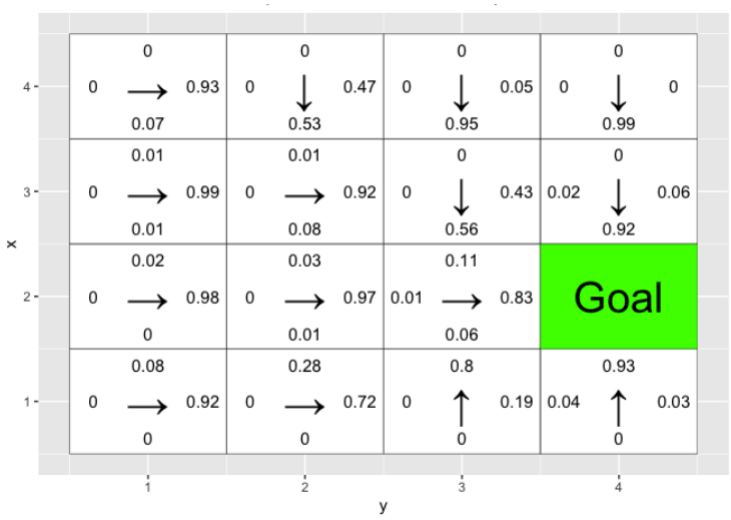

Figure 8: Environment D

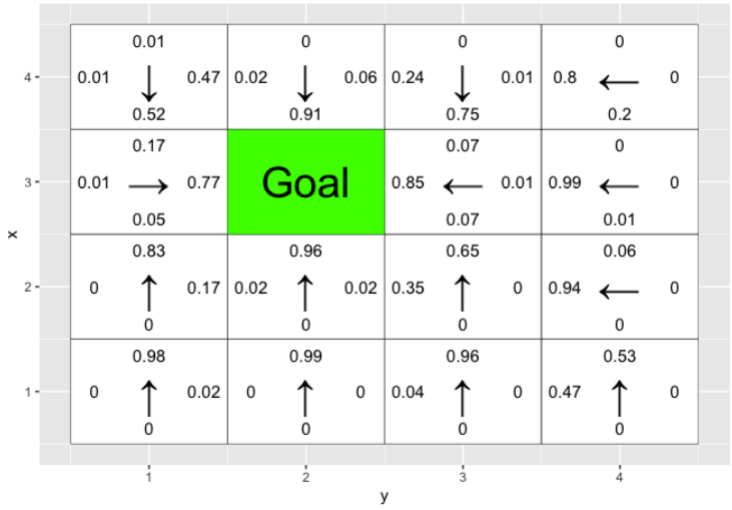

Figure 9: Environment D

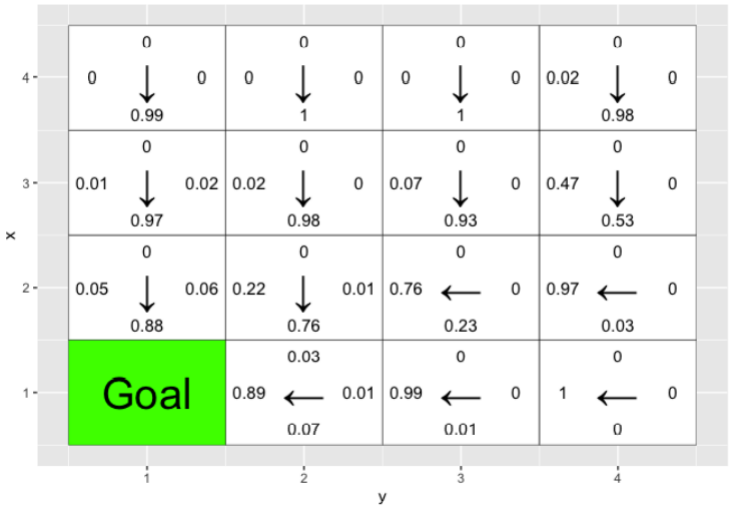

Figure 10: Environment D

Yes, the agent can output generalised policies quite well for the different unseen goals. As seen in the plots, the agent finds close to optimal paths.

Could you have used the Q-learning algorithm to solve this task ?

As the goal changes, the Q-learning algorithm would probably find a path that covers all goal states, depending on the exploration rate. This would not be considered a good solution as the path would be unnecessary long for all goals except the first one reached by the path. Q-learning could also not *generalize* for other goals then the ones learned, as it will learn one single optimal policy for all goals.

The reinforce algorithm learns a generalised policy for any goal, and is therefor preferable in this problem.

2.2 Environment E

Has the agent learned a good policy? Why / Why not ?

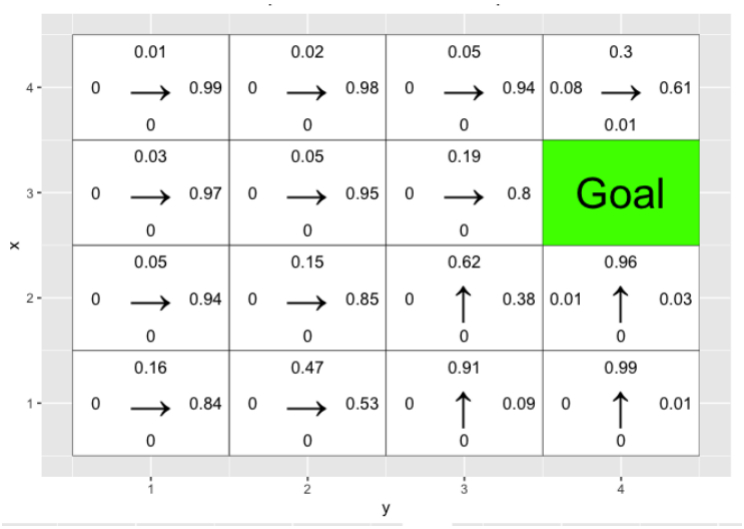

Figure 11: Environment E

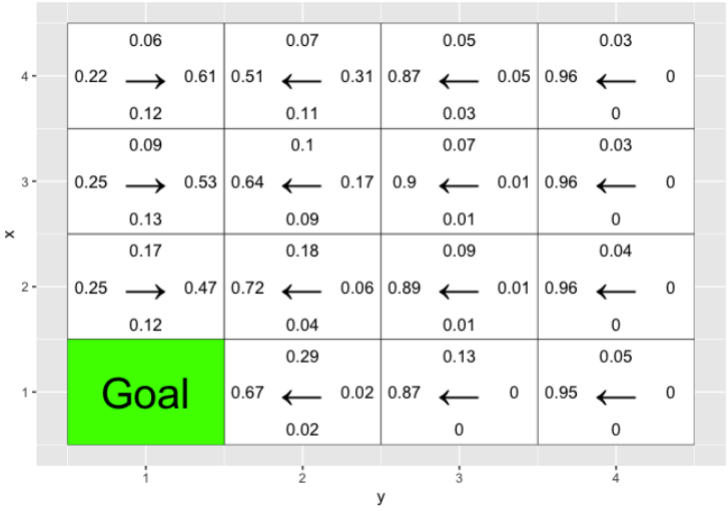

Figure 12: Environment E

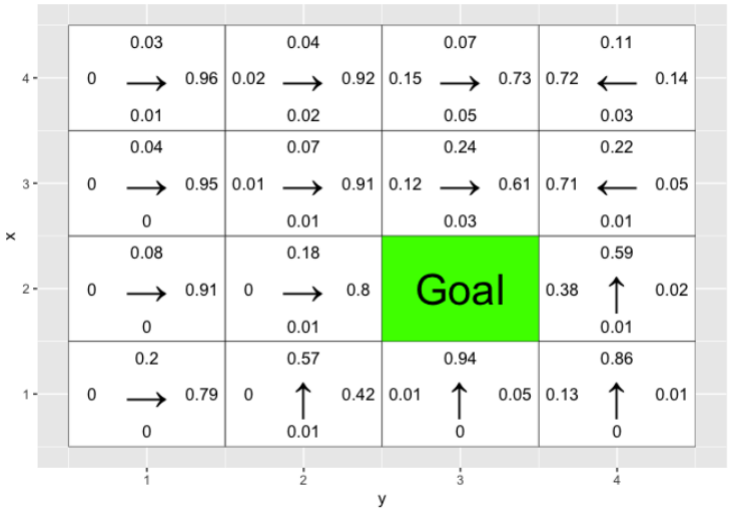

Figure 13: Environment E

The agent can no longer generalise a good policy for the goal. If the goal is in another row than the one trained at (row 2), it will usually not even reach the reward state with the policy if it starts on a row other than the goal state. It can however usually find its way to the goal column on the goal's row, which seems to indicate that the policy for column traversal on the goal's row is generalized, which makes sense as the agent is trained on different column goals, although not different rows.

If the results obtained for environments D and E differ, explain why.

The model can no longer find a good policy for any goal. This is because the model has been trained on a fixed row state. It can therefore not generalise policies for different rows, as it "assumes" that all future goals will be in the same row. All the model in E learns is the policy for the different column goal positions, as the row is a constant. In comparison, the model in environment D is trained on goals with varying locations in both dimensions and the model then has data to generalise the connection between state and goal position for both column and row.