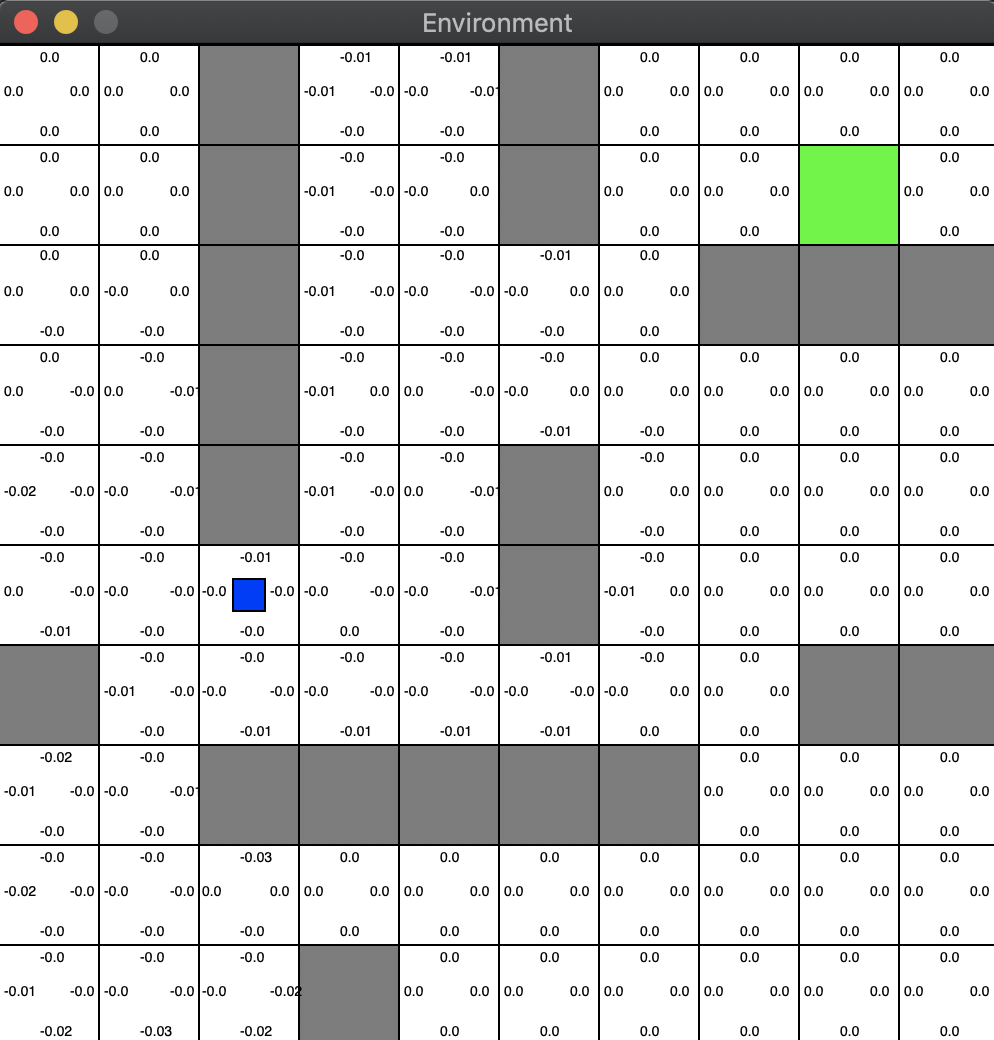
# RL Q-Learning Grid World Example:

## Environment:

First of all, I designed an environment with the characteristics we talked about. A 10x10 grid with some walls and a goal state.

Running the script *display\_grid\_world.py*, the environment will be shown and the agent is going to start trying to reach the goal state.



The initial state is the one with the cross drawn in the image while the goal state is the green square.

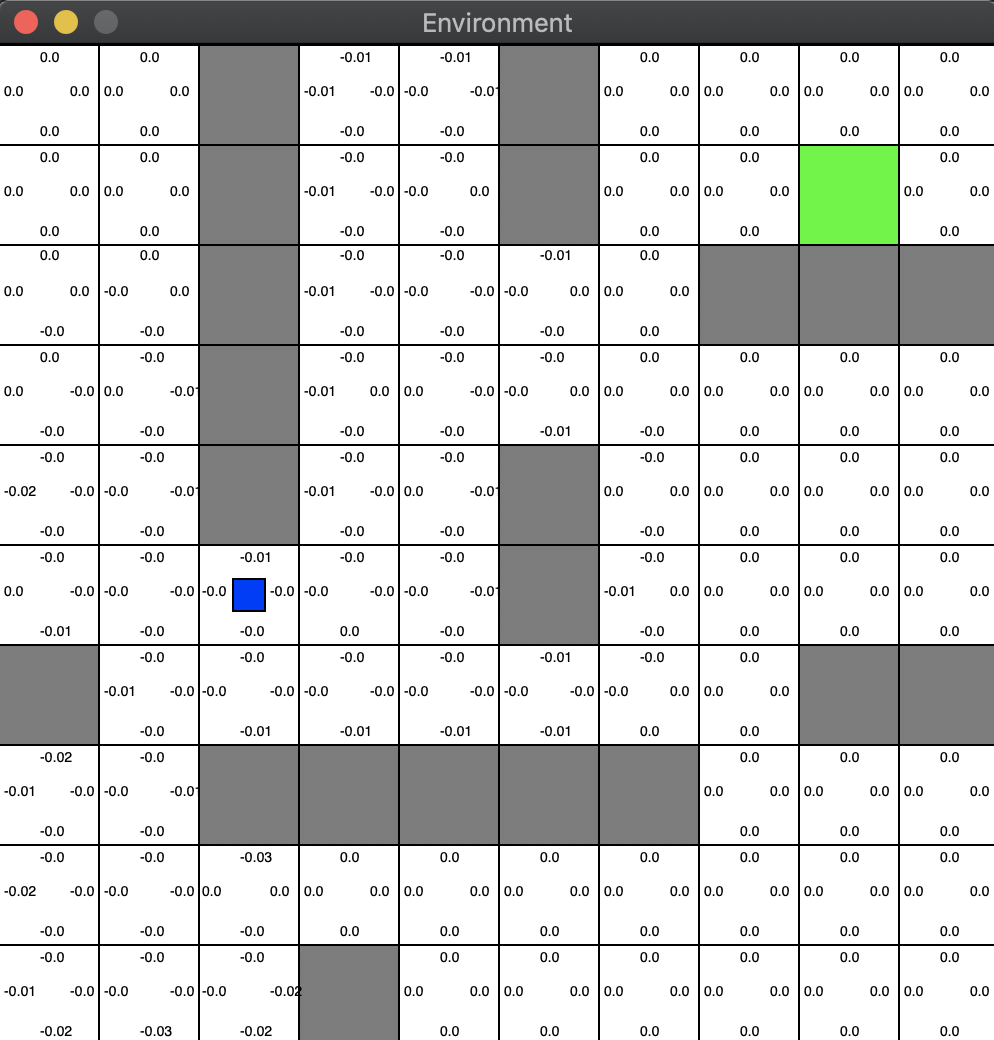
The reward system used is based on the following rules:

* Reaching the goal state = +10
* Trying to go to a wall or outside the 10x10 grid = -1
* Each time-step = -0.01

The environment won’t let the agent be in a wall or go through it. So, in case the agent choses an action that would lead it to a state occupied by a wall or outside the grid, the state won’t be modified and the reward for that action-state pair will be -1.

Questions:

* Is it correct to give more reward when reaching the goal state than when hitting a wall?
* If I don’t penalize the agent every time-step, the system usually converged to a suboptimal solution. But if I penalize too much per time-step, the results don’t make sense at all. That’s why I chose that small negative reward (-0.01). Is there any concrete explanation for this behaviour?
  + Optimal solution: 15 time-steps
  + Sub-optimal solutions: 17, 19, 21 time-steps…



## Agent:

After designing the environment and testing its behaviour, I started implementing the agent with a policy based on Q-Learning algorithm. In the beginning, the epsilon-greedy method was chosen in order to pick the action the agent must take. But using that method the system never converged to the optimal solution. It was always converging very far away from the optimal. After that, I realized that I should use the same method but with an epsilon decay rate in order to stop exploring while getting to the optimal behaviour. That worked. Could be that the reason?

I’ve also been evaluating the agent’s behaviour depending on other parameters such as the learning rate, discount factor, epsilon… And the ones I finally chose are the ones that were always converging to the best (optimal) solution (15 time-steps).

I haven’t analysed how all of these parameters affect the convergence to that optimal solutions. Should I?

## Results:

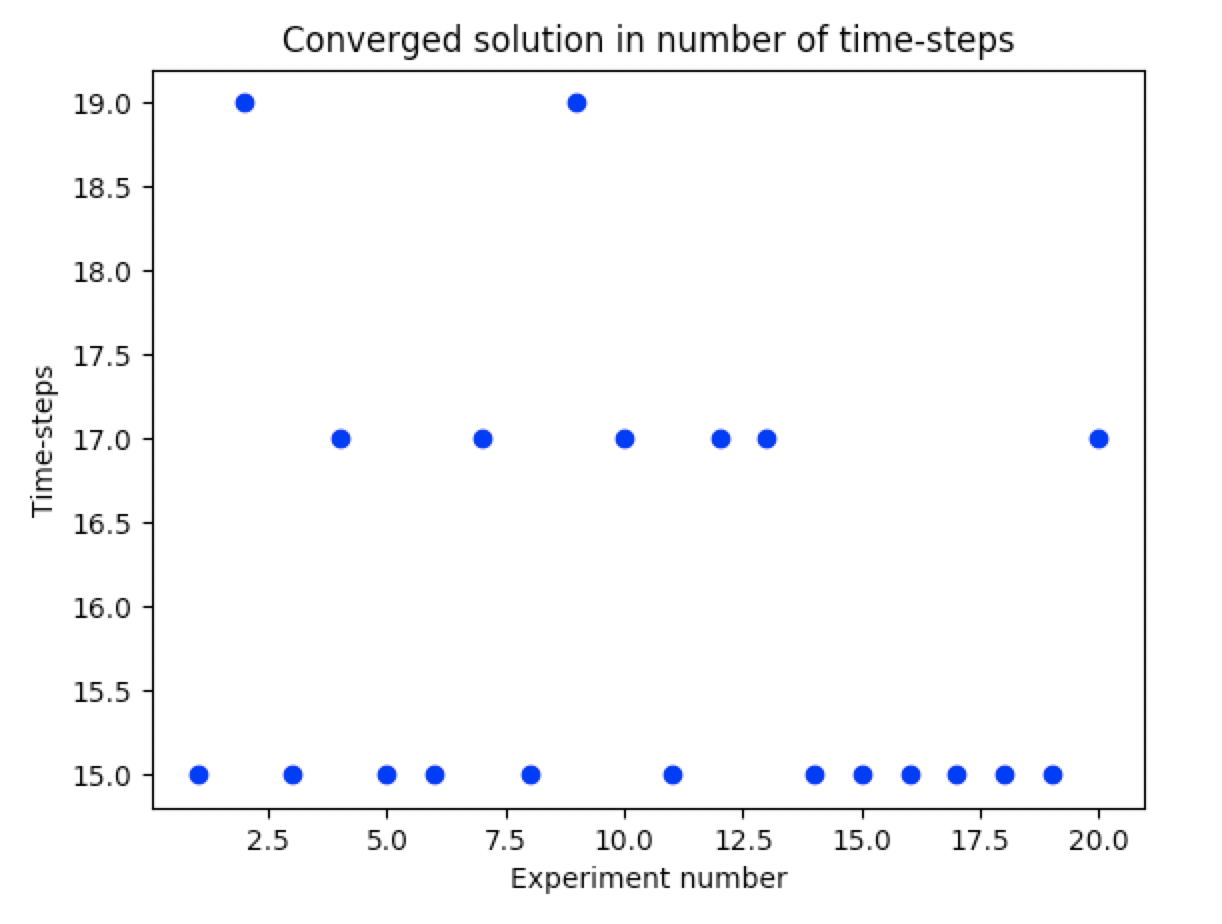
In every experiment are simulated 1000 episodes in order to ensure convergence. To proof that the system is converging “always” to the same value, I ran 20 experiments per each test. (1 test 🡪 20 experiments, 1 experiment 🡪 1000 episode).

Parameters used:

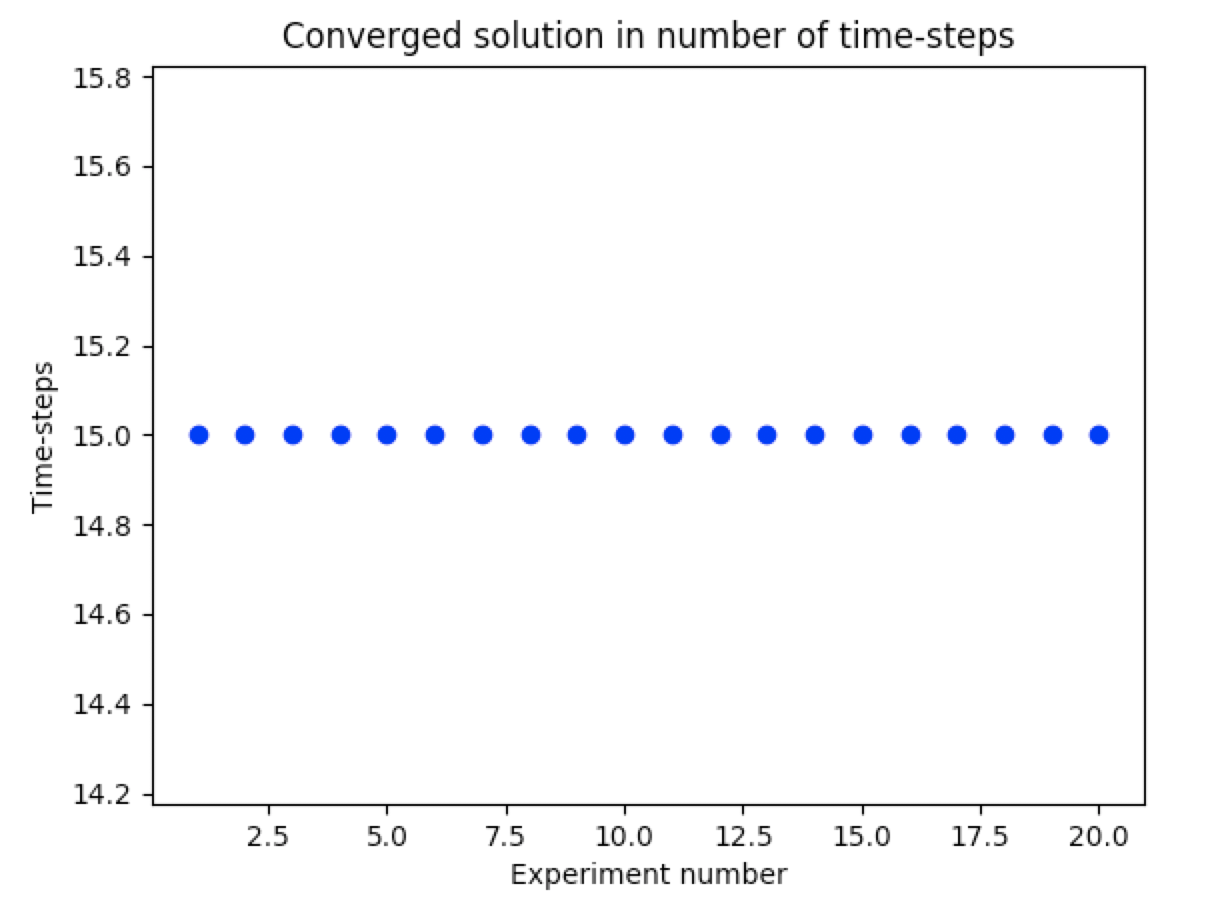
* GOAL\_STATE\_REWARD = 10
* WALL\_STATE\_REWARD = -1
* EPSILON = 0.9
* EPSILON\_DECAY\_RATE = 0.99
* DISCOUNT\_FACTOR = 0.9
* LEARNING\_RATE = 0.01

### With/Without time-step penalization:

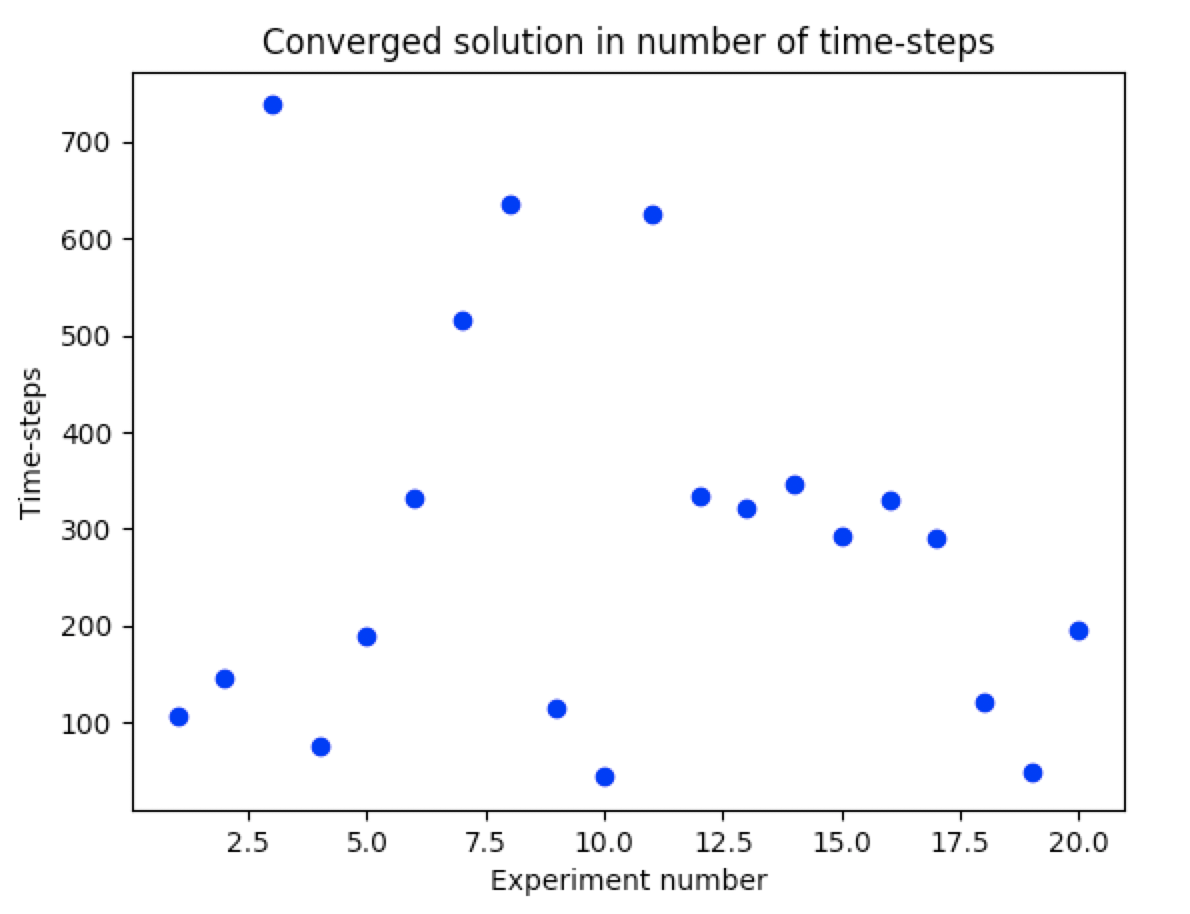
Without time-step penalization



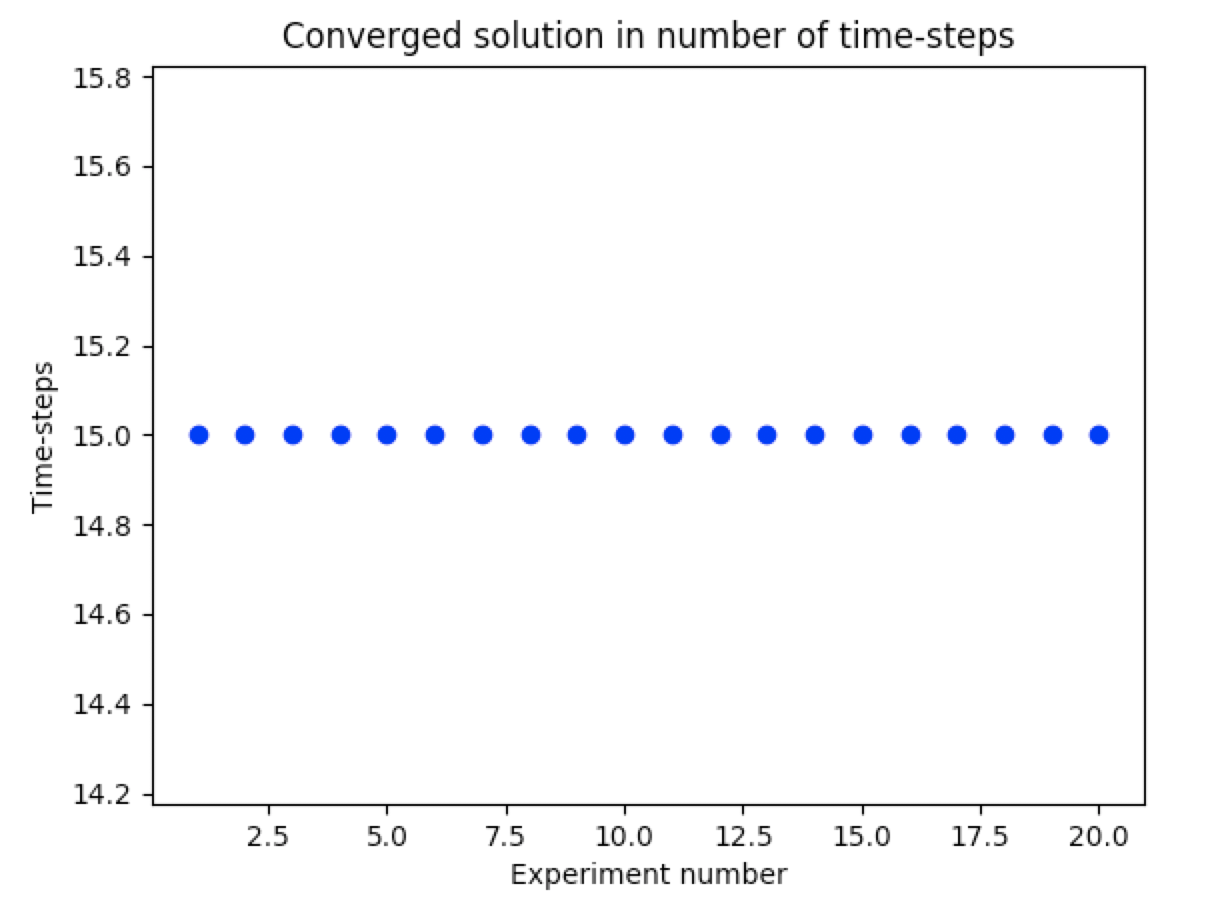
With time-step penalization



### With/Without epsilon greedy decay rate:

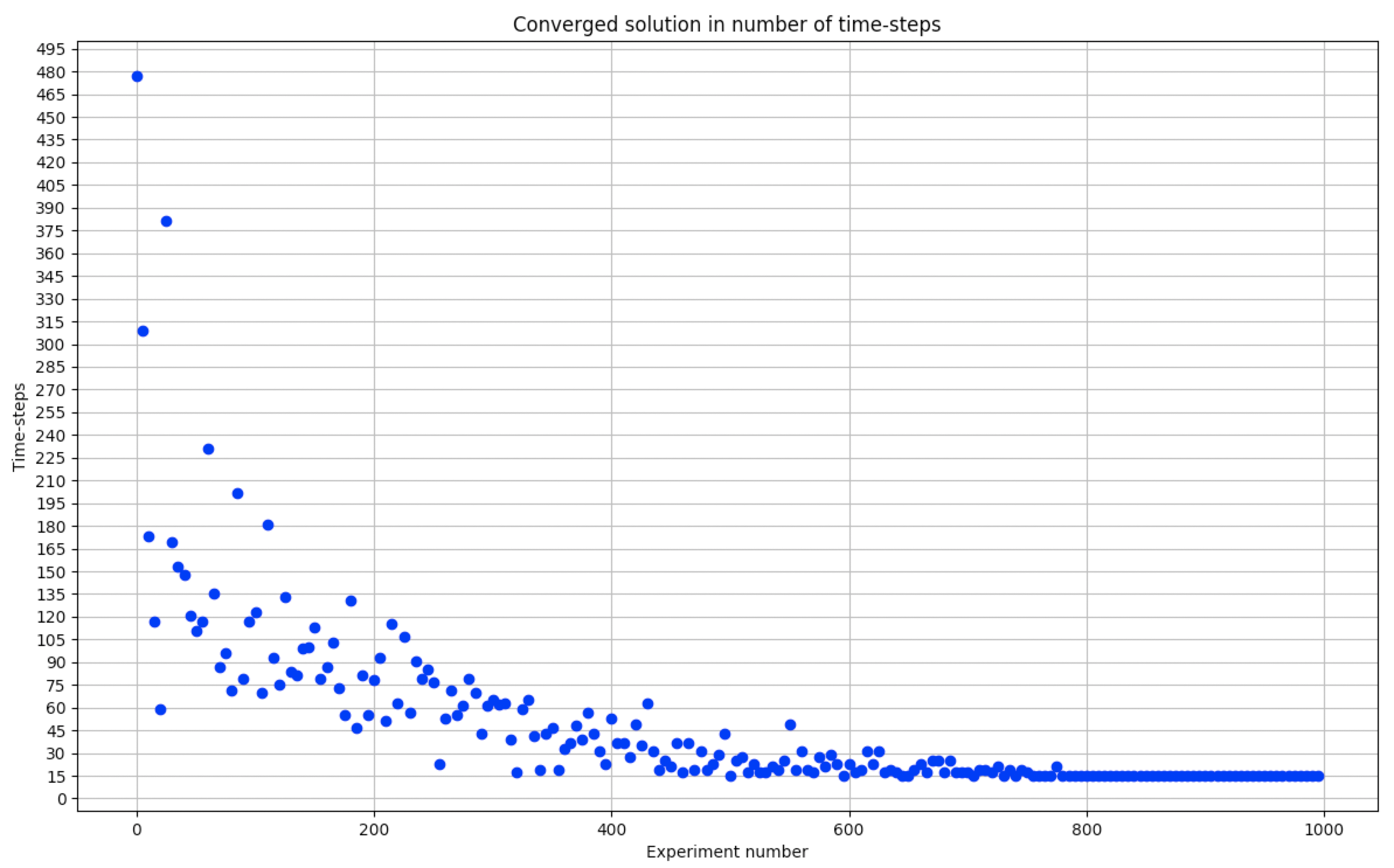


With epsilon-greedy decay rate

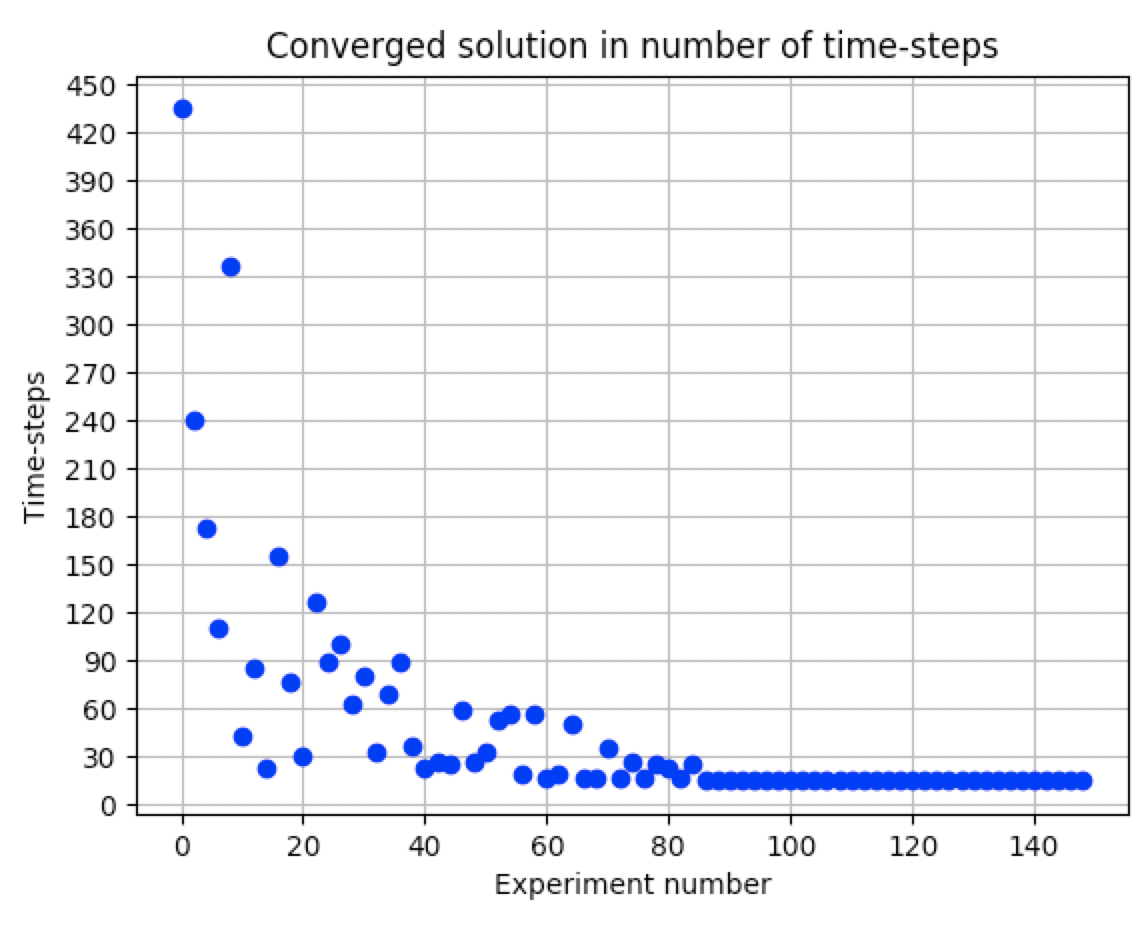


Without epsilon-greedy decay rate

### Convergence of the “optimal” system found:



Learning rate = 0.01



Learning rate = 0.1