Reasoning Agents Project

Policy Networks for Non Markovian Deep RL

SAPIENZA
UNIVERSITÀ DI ROMA

Faculty of Information Engineering, Informatics, and Statistics

Department of Computer, Control, and Management Engineering "Antonio Ruberti"

Master's Degree in Artificial Intelligence and Robotics

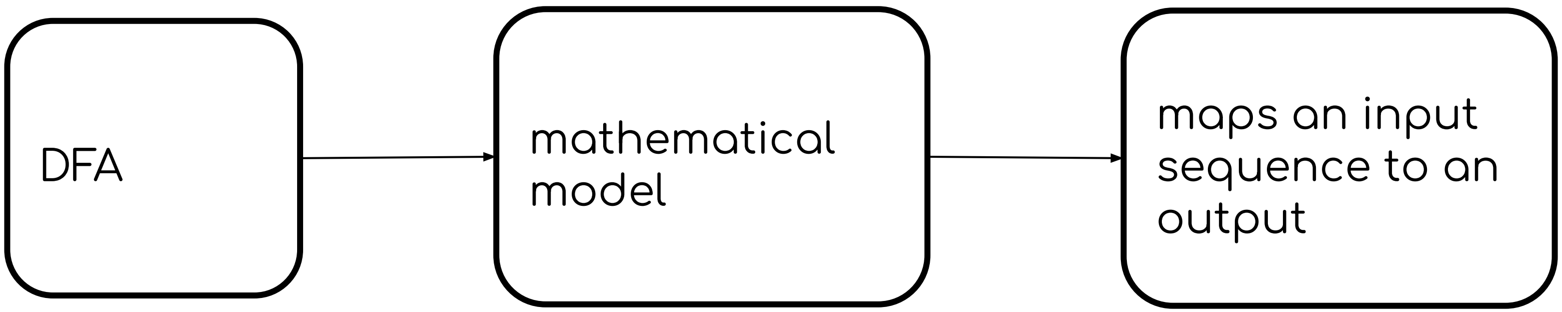
Federica Coppa
Francesca Cardia
Andrea Del Vecchio

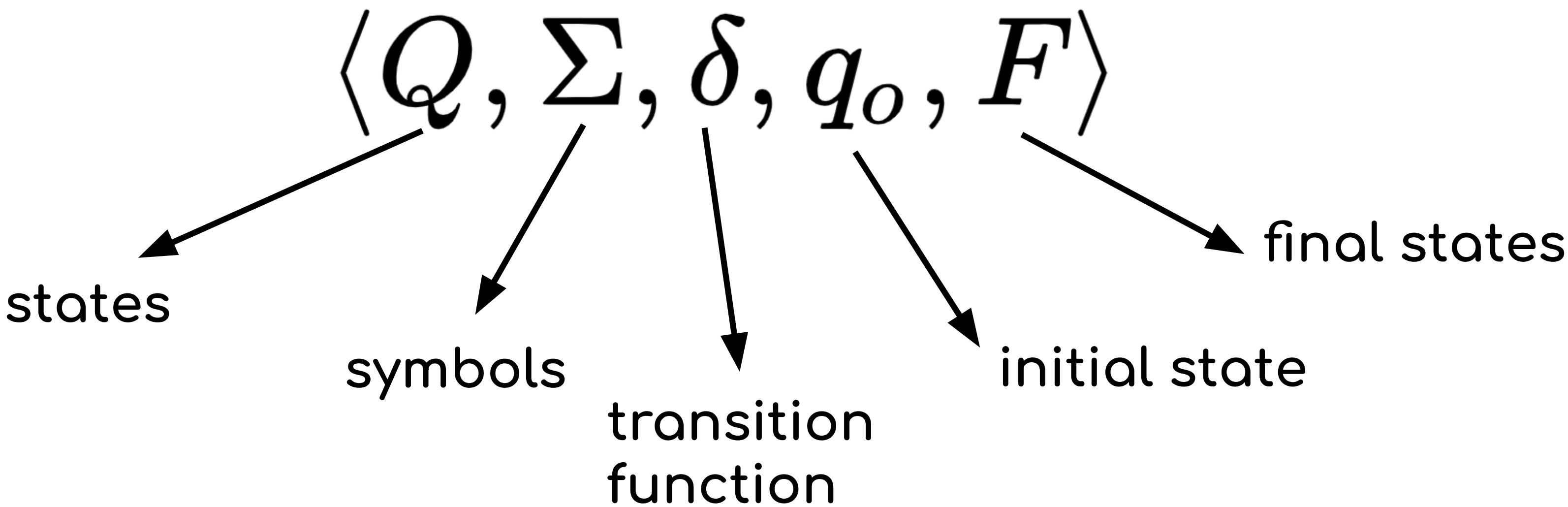
Introduction

- Goal: develop a non-Markovian agent that solves a **navigation task with non-Markovian rewards:**

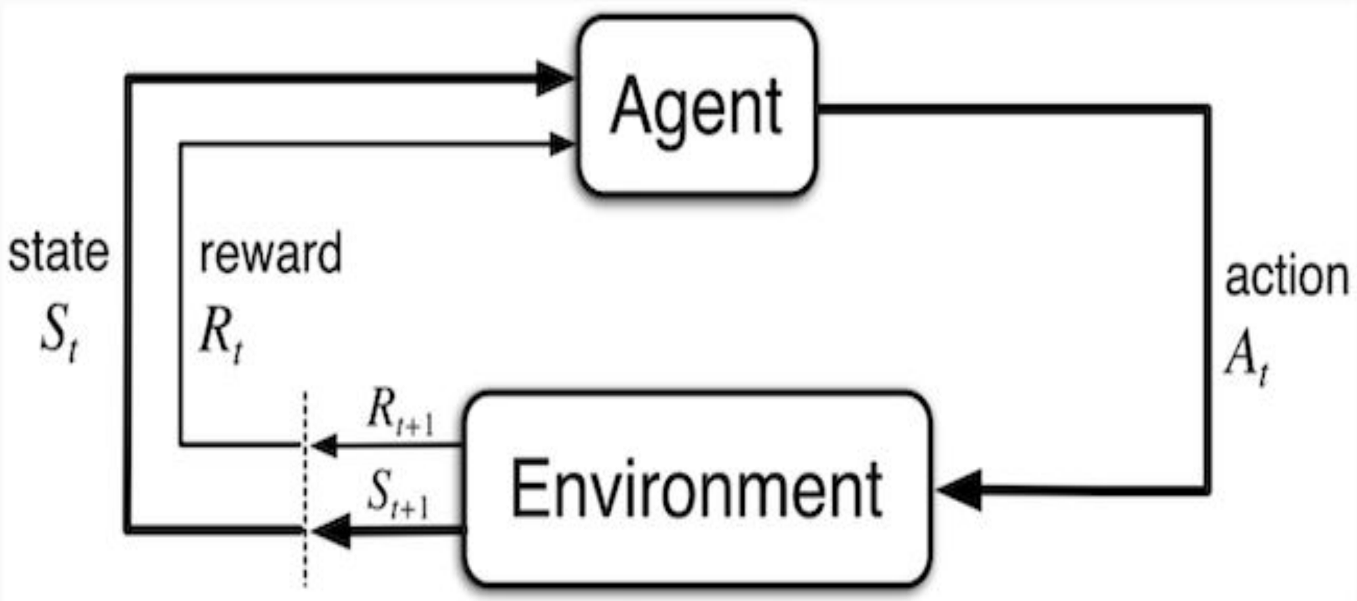
- Combination of RL agent with an automaton: PPO + DFA.

Deterministic Finite Automata (DFA)

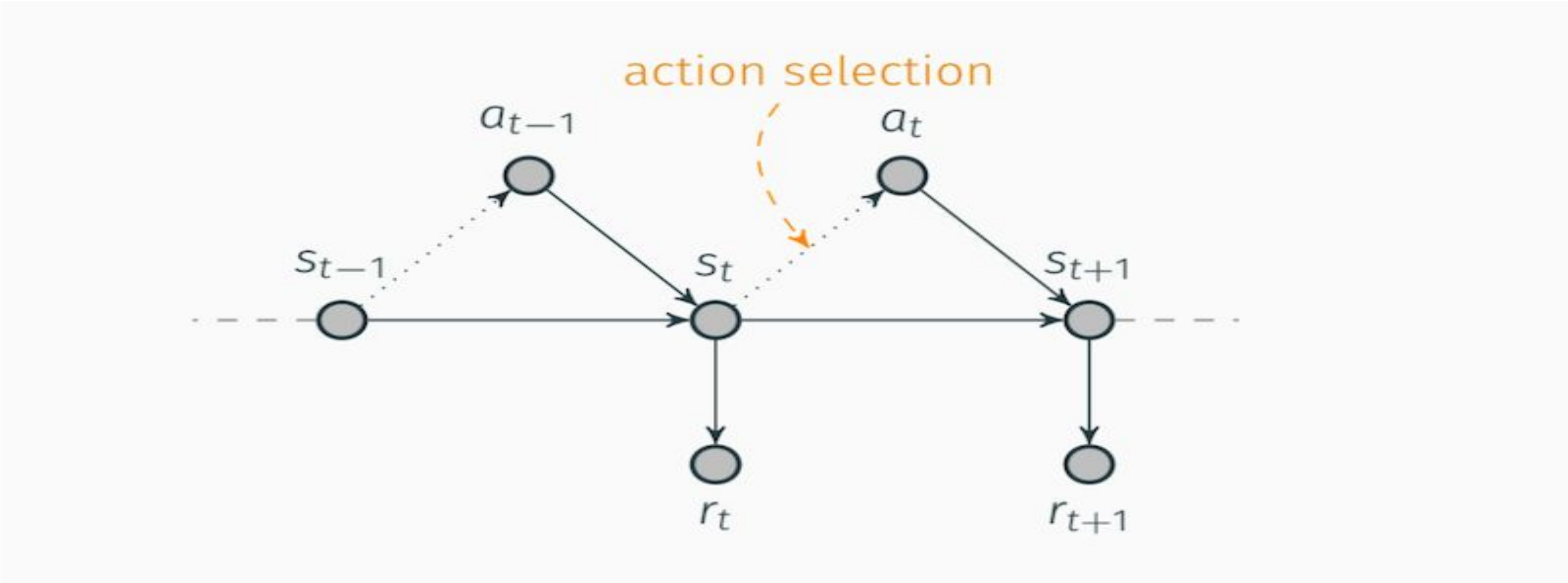




RL general setting



Markov Decision Processes (MDPs)



$$\langle S, A, T, R, \gamma \rangle$$

states

actions

transition function

reward function

$R : (S \times A) \longrightarrow \mathbb{R}$

discount factor

Properties of MDPs

$$s_{t+1} \perp s_0, \ldots, s_{t-1} | s_t \quad \forall t$$

$$r_{t+1} \perp s_0, \ldots, s_{t-1} | s_t \quad \forall t$$

- A policy is a solution for a MDP which assigns an action to each state. Moreover each MDP has an optimal policy able to maximizes the expected rewards for every starting state $s \in S$

Non Markovian Rewards Decision Processes (NMRDPs)

- The reward are the real valued function over finite state-action sequences (according to a specific history):

$$\bar{R} : \langle s_1, a_1, \ldots . a_{t-1}, s_t \rangle \longrightarrow \mathbb{R}$$

- **Temporal Rewards Specification (given $\{(\phi_i, r_i)\}_{i=1}^{m}$)**

$$\bar{R}(\pi) := \sum_{i : \pi \models \varphi^{(i)}} r^{(i)}$$

- **Rewarding with automata** $\phi \longrightarrow A_\phi$

$$S' = S \times Q_{\phi(1)} \times \ldots . \times Q_{\phi(m)}$$

Proximal Policy Optimization (PPO)

- A policy gradient algorithm for Deep RL.

$$\hat{g} = \hat{\mathbb{E}}_t \left[\nabla_\theta \log \pi_\theta(a_t \mid s_t) \hat{A}_t \right]$$

- **Advantage** is a measure of how remunerative a certain action was in a determined time instant.
    - Difference between the action value in a state and the average value of that state

$$A_t = \sum_{k=1}^{T} \gamma^k r_{t+k} - V_{\theta_V}(s_t)$$

Proximal Policy Optimization (PPO)

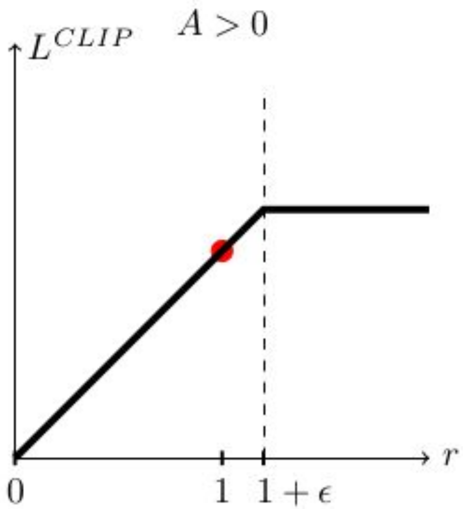
- Problem: "too big" policy updates often lead to unstable behaviour
  - Solution: policy gradient objective is replaced with **clipped surrogate loss.**

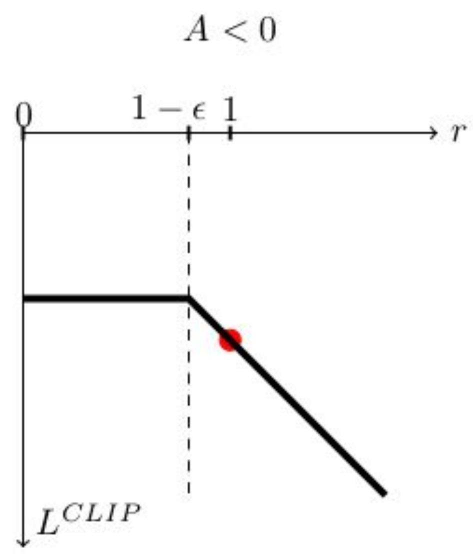
$$r_t(\theta) = \frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}(a_t \mid s_t)}$$

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t\Big[\min(r_t(\theta)\hat{A}_t, \operatorname{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)\Big]$$

Proximal Policy Optimization (PPO)

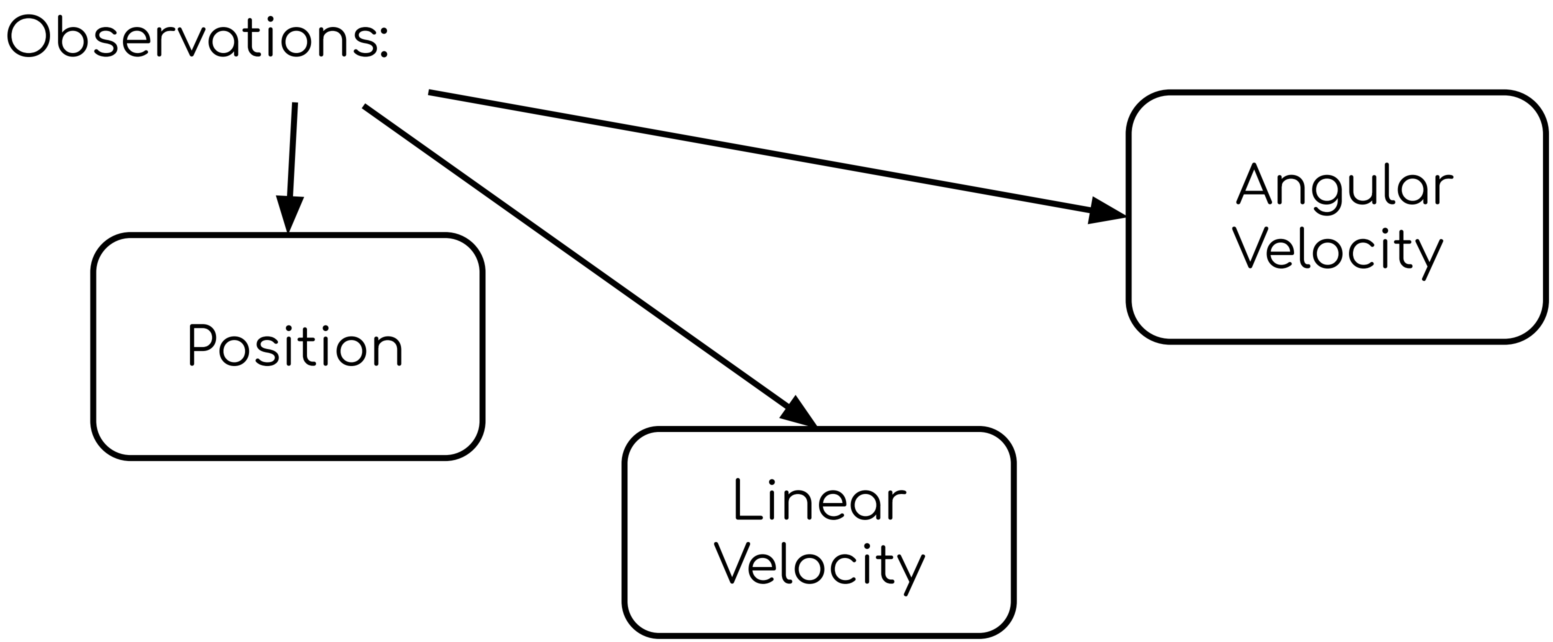
- Clipping forces the policy updates to be conservative: the updated policy not “too far away” from the older policy





Gym-Sapientino environment

- It is a RL discrete action environment
- The actions continuous on the state.



Temporal Goals on Gym-Sapientino

- The agent can reach temporal goals characterized by a visited sequence of colors:

- When the agent reaches a temporal goal it receives a reward from the environment.
- It is possible to specify which colors are included in the temporal goal. SINK state is signed as 2 to indicate failure .

Temporal Goals on Gym-Sapientino (contd.)

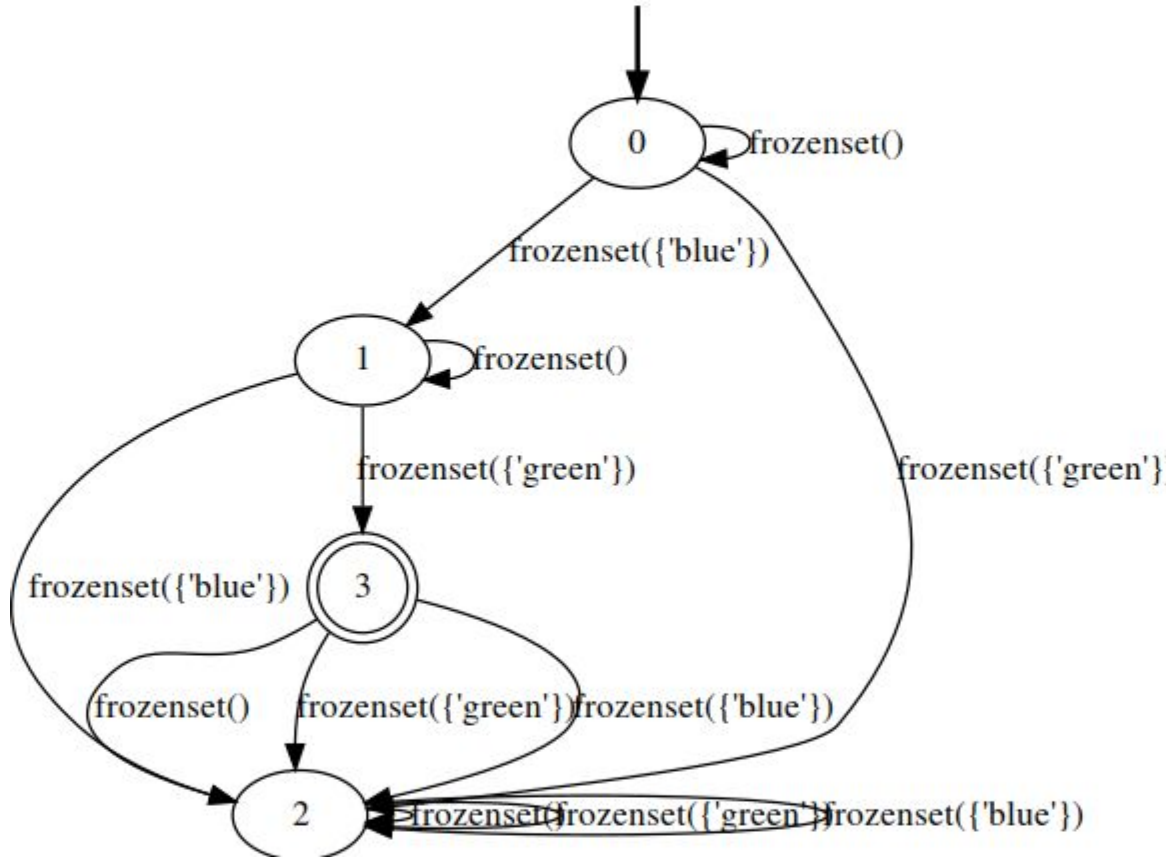
- Actions: *LEFT, RIGHT, FORWARD, NULL, BEEP.*

- Double *BEEP* problem: in order to test the temporal goal satisfaction, we have interfaced with this problem. In fact we have controlled, during the training, that the previous visited automaton state was different from the current visited automaton state.

Automaton example (considering only 2 colors, ***blue*** and ***green***):

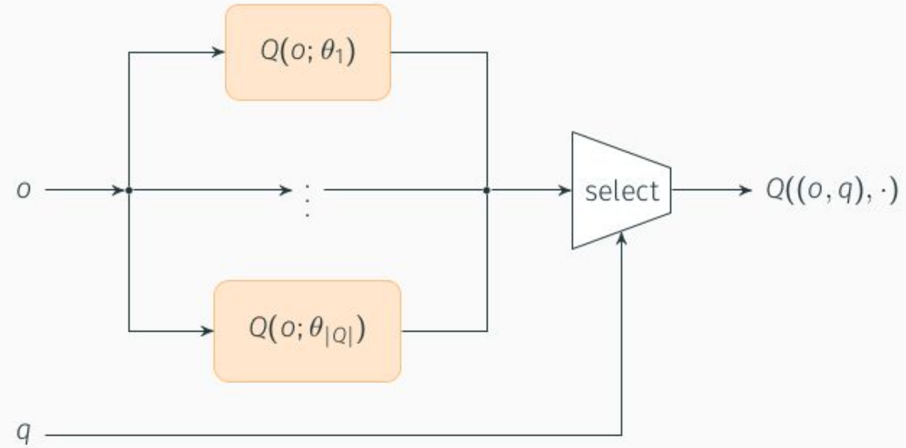
right sequence to visit:
{0,1,3}

no **memoryless** property



The non markovian agent

As baseline, we'll consider this implementation:



A number of $|Q|$ separate "experts".

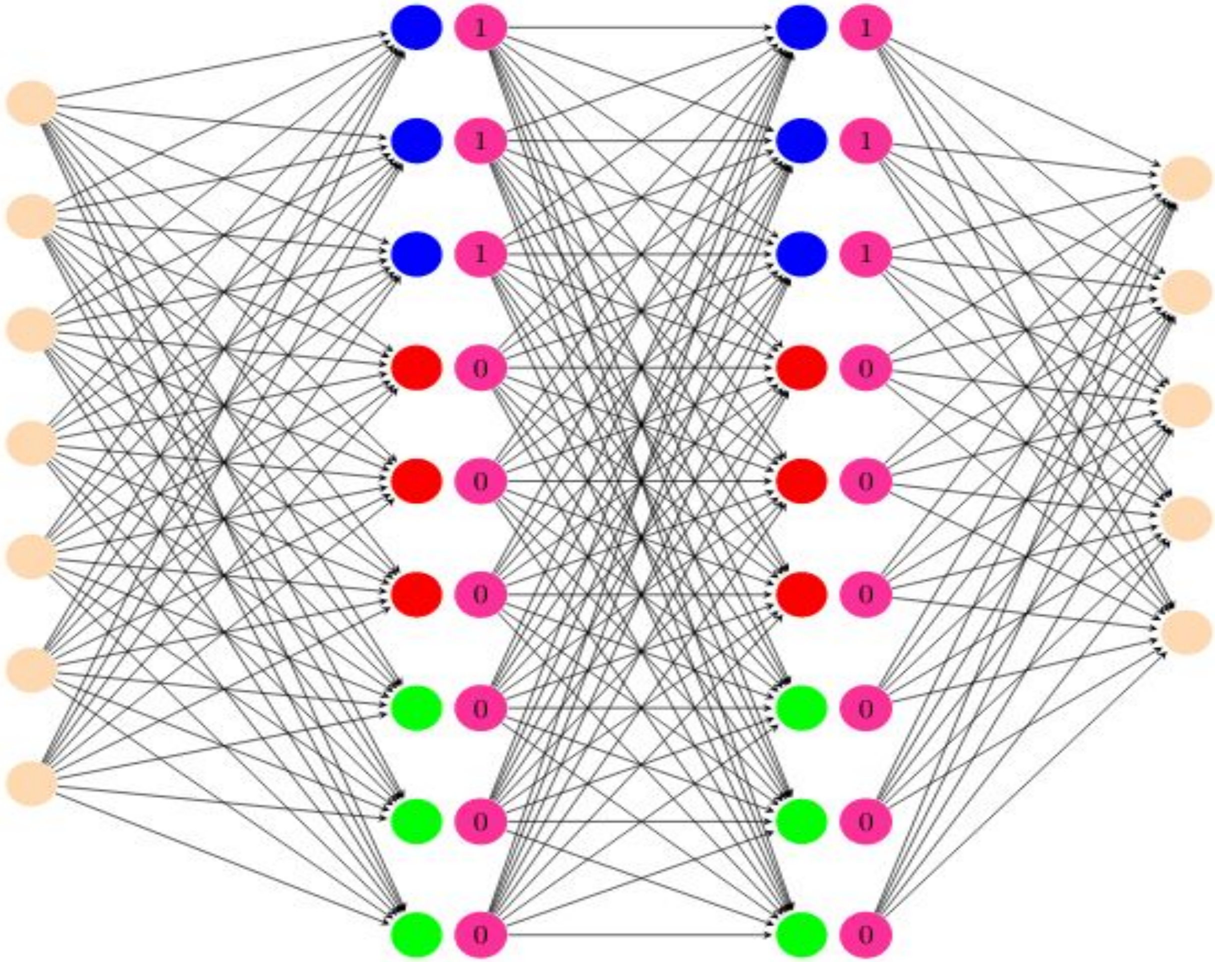
- different **sub-problems** division (markovian sub-problems), given the non markovian task.

- according to the state of the automaton, it is possible to select one of the |Q| separate *experts* for the action selection phase.

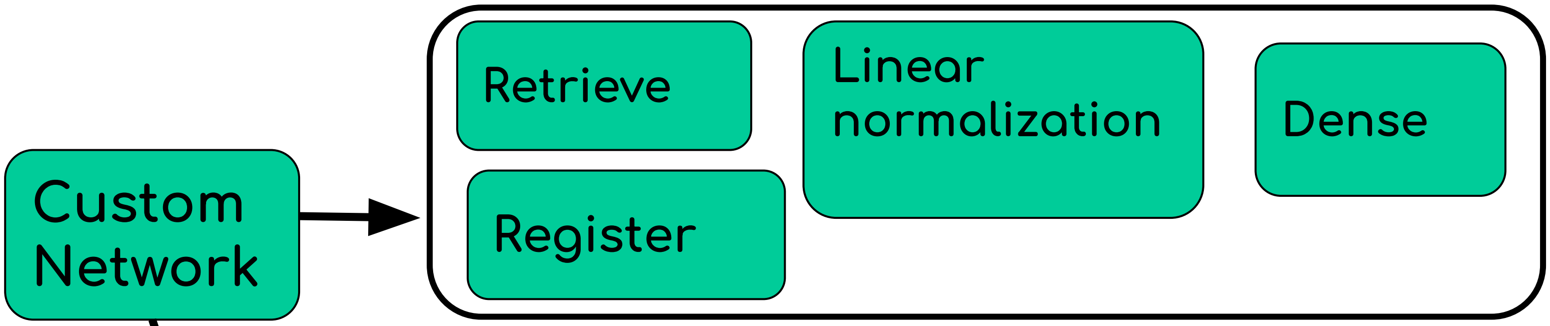
Non markovian agent: implementation details

- We use a **single policy network** for the non markovian agent.
- The core part of the network is **divided amongst the different color expert.**
- Select the expert, the network portion which contributes to action selection we multiply a **binary vector** to the output of each hidden layer.

Non markovian agent: implementation details



Network Description: implementation details




```
network=dict(type = 'custom',

    layers= [

    dict(type = 'retrieve',tensors= ['gymtpl0']),
    dict(type = 'linear_normalization'),

    dict(type='dense', bias = True,activation ='tanh',size=
    AUTOMATON_STATE_ENCODING_SIZE),

    dict(type= 'register',tensor = 'gymtpl0-dense1'),

    #Perform the product between the one hot encoding of the
    automaton and the output of the dense layer.
    dict(type = 'retrieve',tensors=['gymtpl0-dense1','gymtpl1'],
    aggregation = 'product'),

    dict(type='dense', bias = True,activation ='tanh',size=
    AUTOMATON_STATE_ENCODING_SIZE),

    dict(type= 'register',tensor = 'gymtpl0-dense2'),

    dict(type = 'retrieve',tensors=['gymtpl0-dense2','gymtpl1'],
    aggregation = 'product'),

    dict(type='register',tensor = 'gymtpl0-embeddings'),],)
```

Reward shaping : implementation details

This method is useful to incorporate domain knowledge in the RL agents. In reward shaping, the domain knowledge is represented as a supplementary reward that allows the RL agent to learn more efficiently.

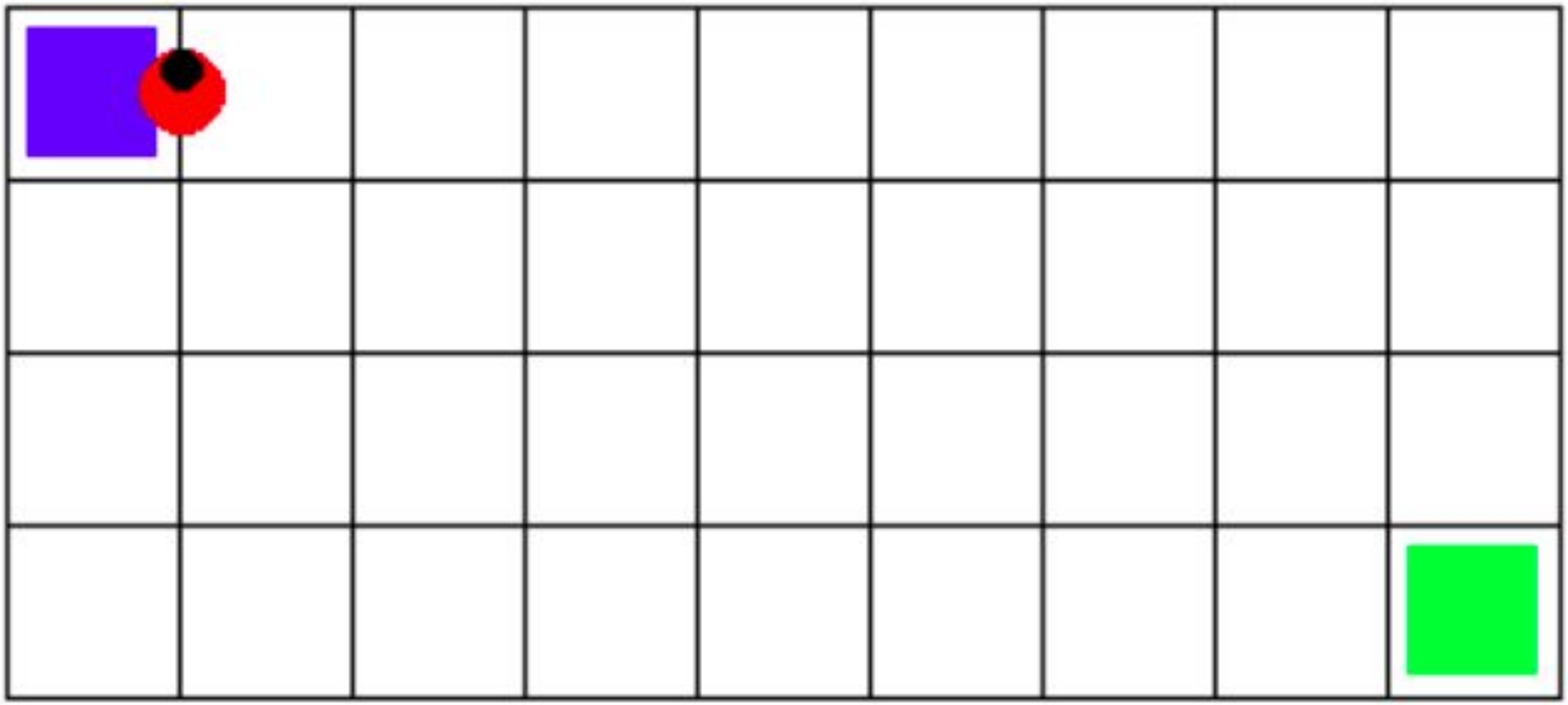
```
#Reward Shaping
if automaton_state == SINK_ID:
    reward = -500.0
    terminal = True


elif automaton_state == 1 and prevAutState != 1:
    reward = 500.0

elif automaton_state == 3:
    reward = 500.0
    print("Visited the goal in episode: ", episode)
```

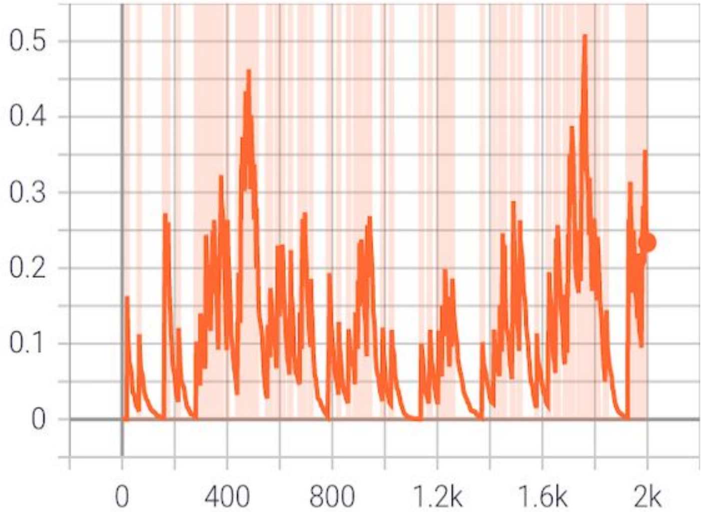
Experiments with two colors

- Gym-Sapientino 4x9 map for the experiment {*blue* ,*green*}:

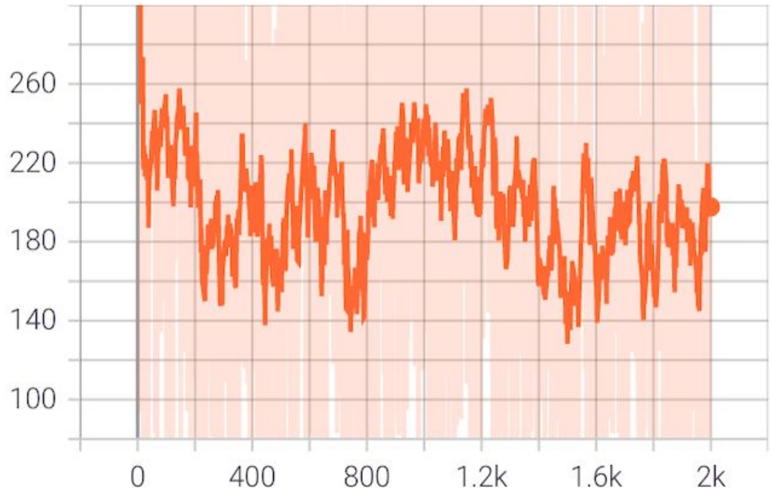


Trial 1 -Plots-

agent/cond/episode-return
tag: agent/cond/episode-return

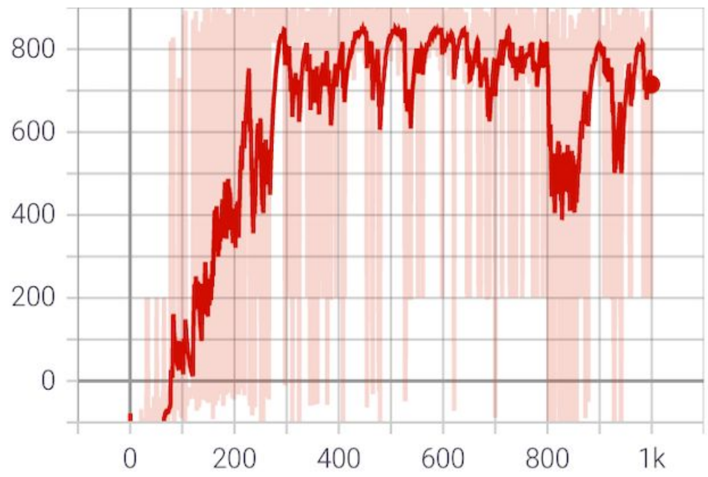


agent/cond/episode-length
tag: agent/cond/episode-length

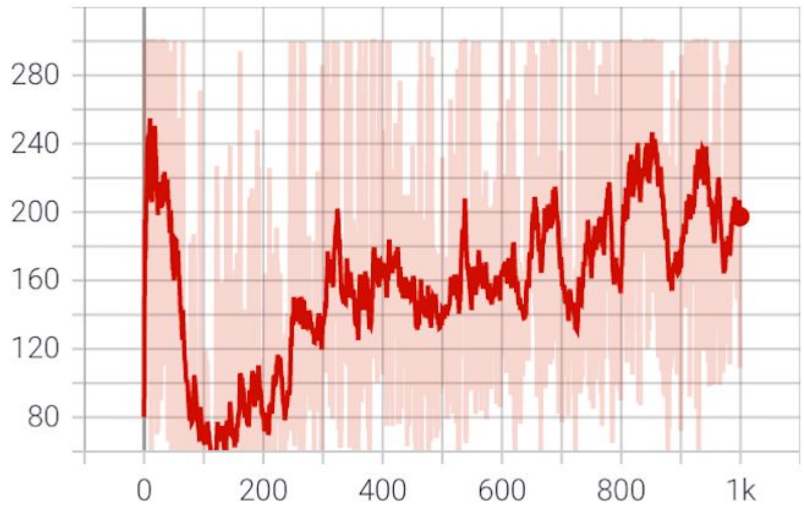


Trial 2 -Plots-

agent/cond/episode-return
tag: agent/cond/episode-return



agent/cond/episode-length
tag: agent/cond/episode-length

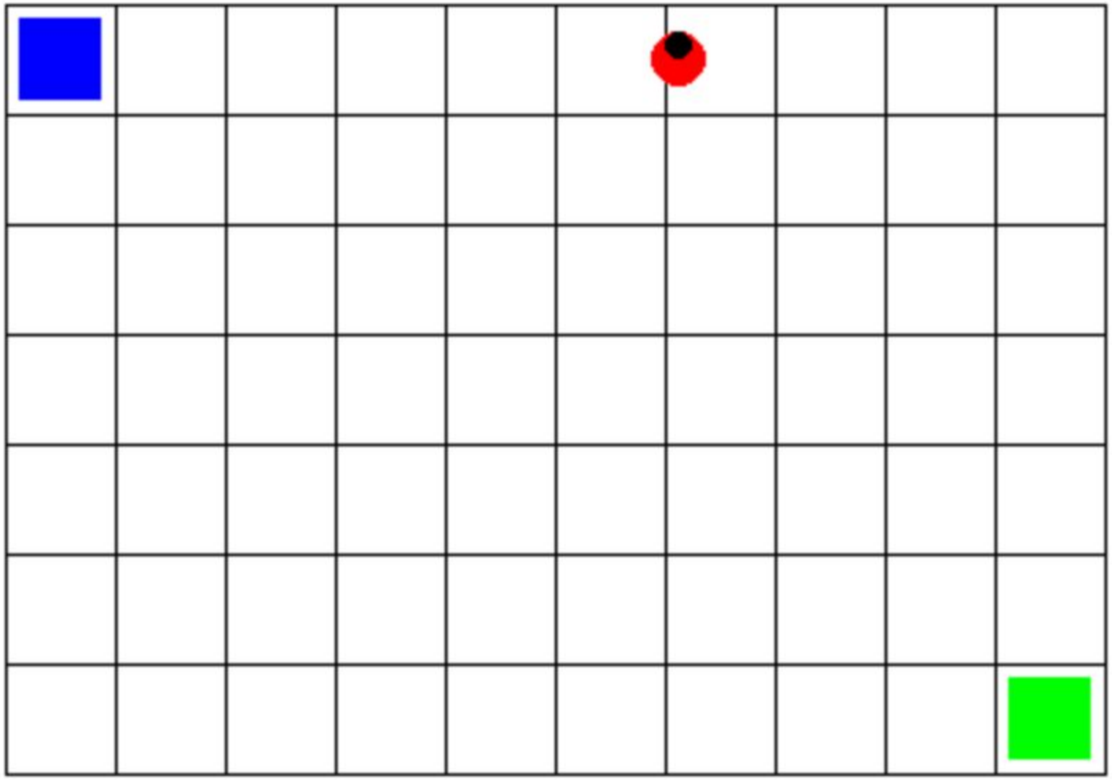


Trial 1 vs Trial 2 -Comparison-

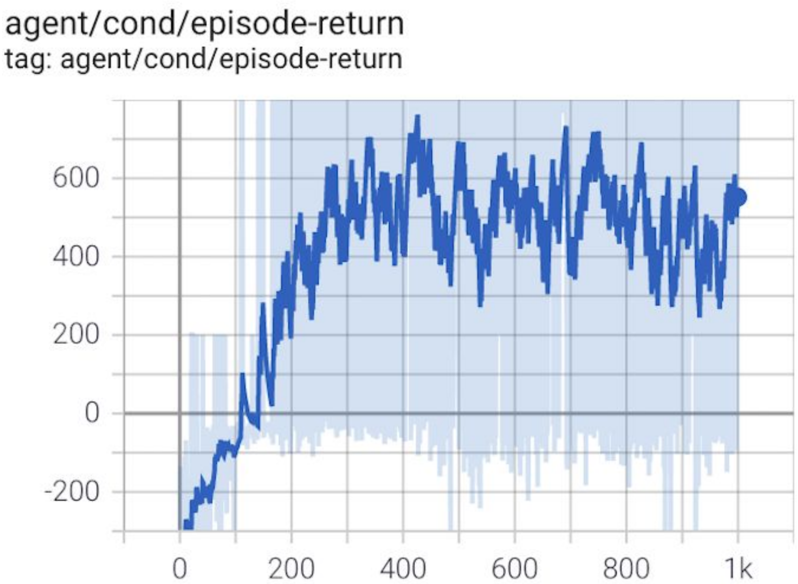
Agent parameters								
Trial	Algorithm name	batch size	memory	exploration	lr	update fre-quency	reward shap-ing	episodes
trial1	PPO	32	20000	0.4	0.001	32	no	2000
trial2	PPO	64	64	0.0	0.001	20	yes	1000

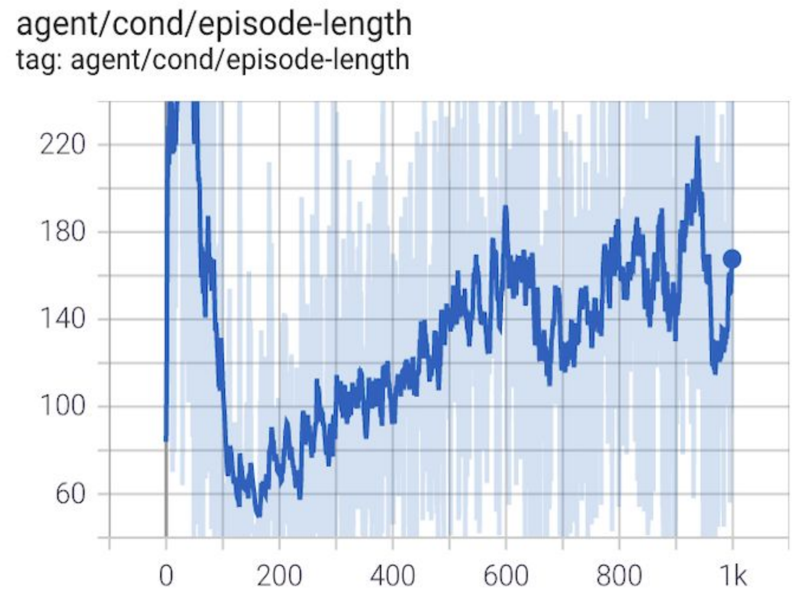
First experiment with two colors with a bigger map (slightly worse behaviour)

- Gym-Sapientino 7x10 map for the first experiment {*blue*, *green*}:



Plots and Table -First experiment-

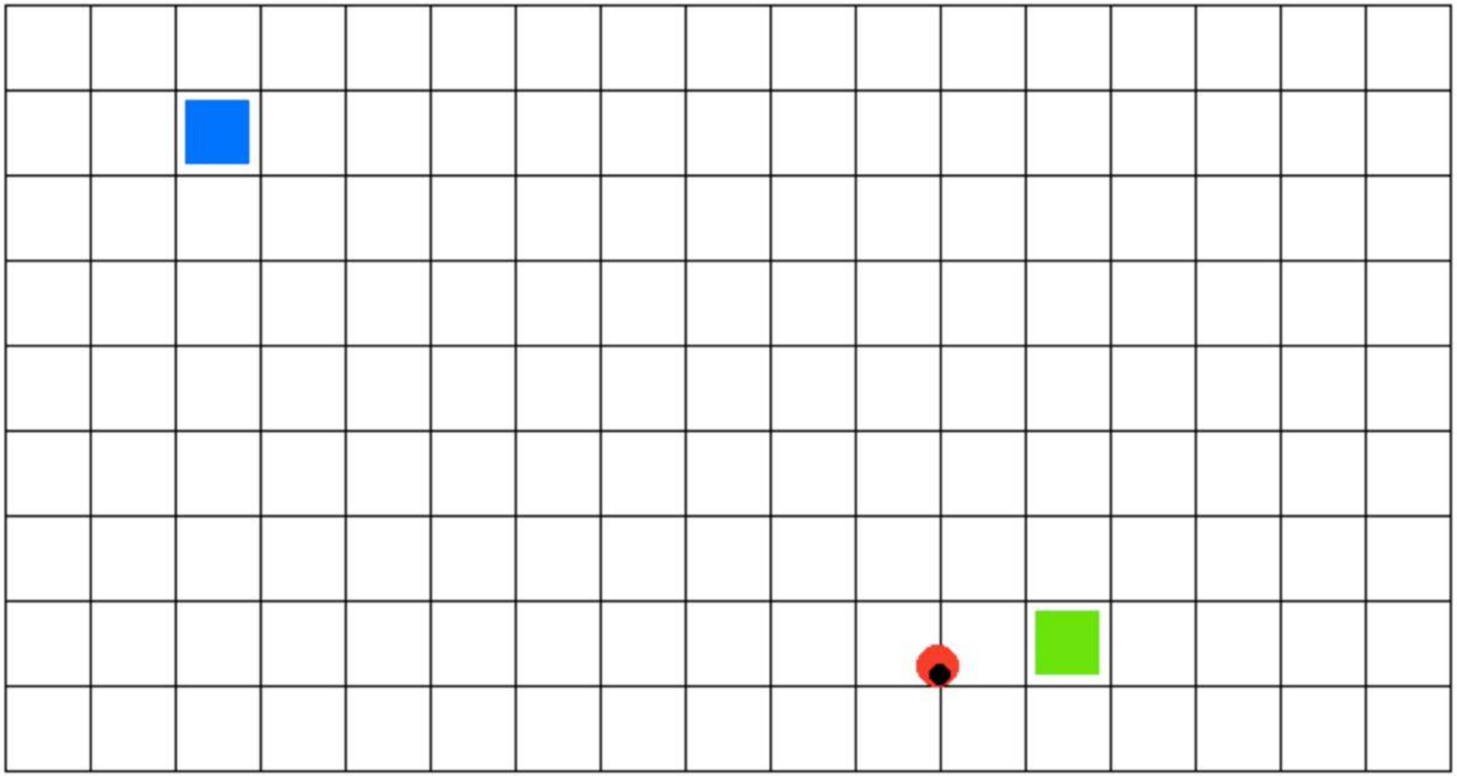




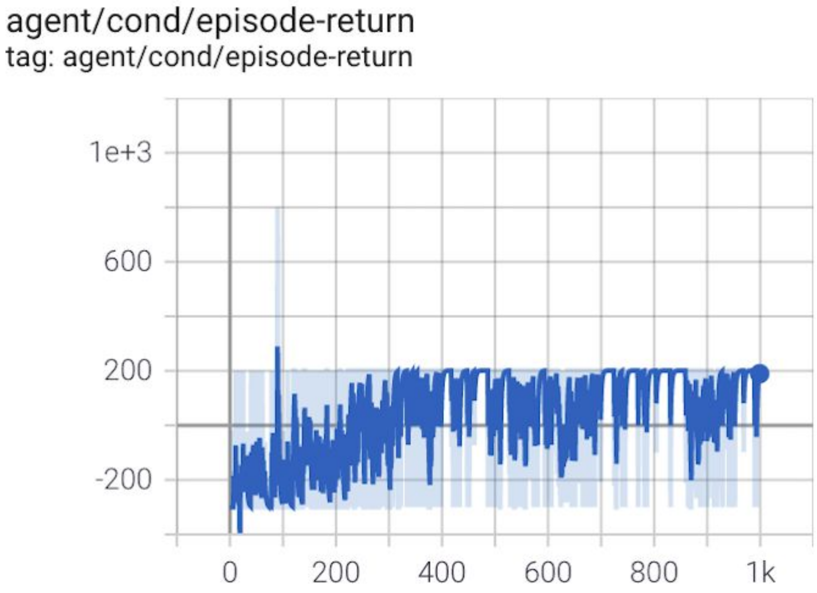
Agent parameters								
Trial	Algorithm name	batch size	memory	exploration	lr	update fre-quency	reward shap-ing	episodes
trial1	PPO	64	64	0.25	0.001	20	yes	1000

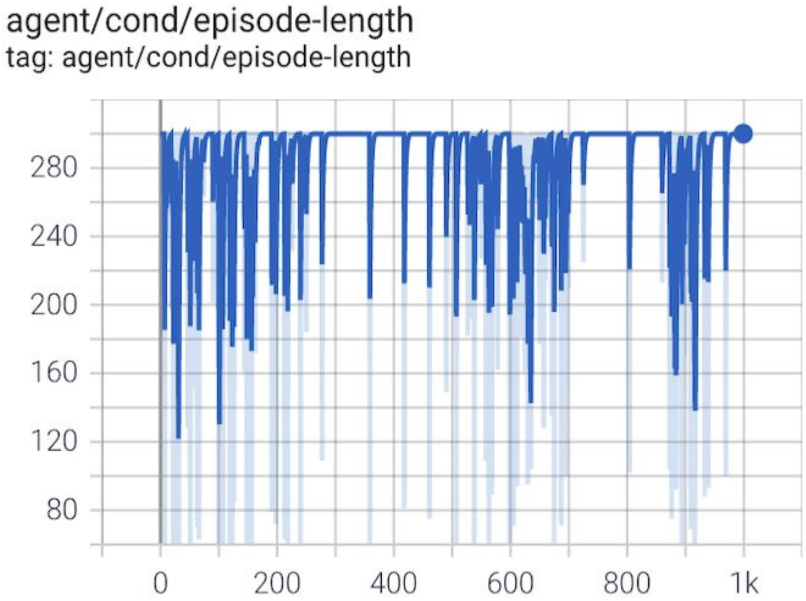
Second experiment with two colors with a bigger map (worse behaviour)

- Gym-Sapientino 9x17 map for this experiment {*blue*, *green*}:



Plots and Table -Second experiment-

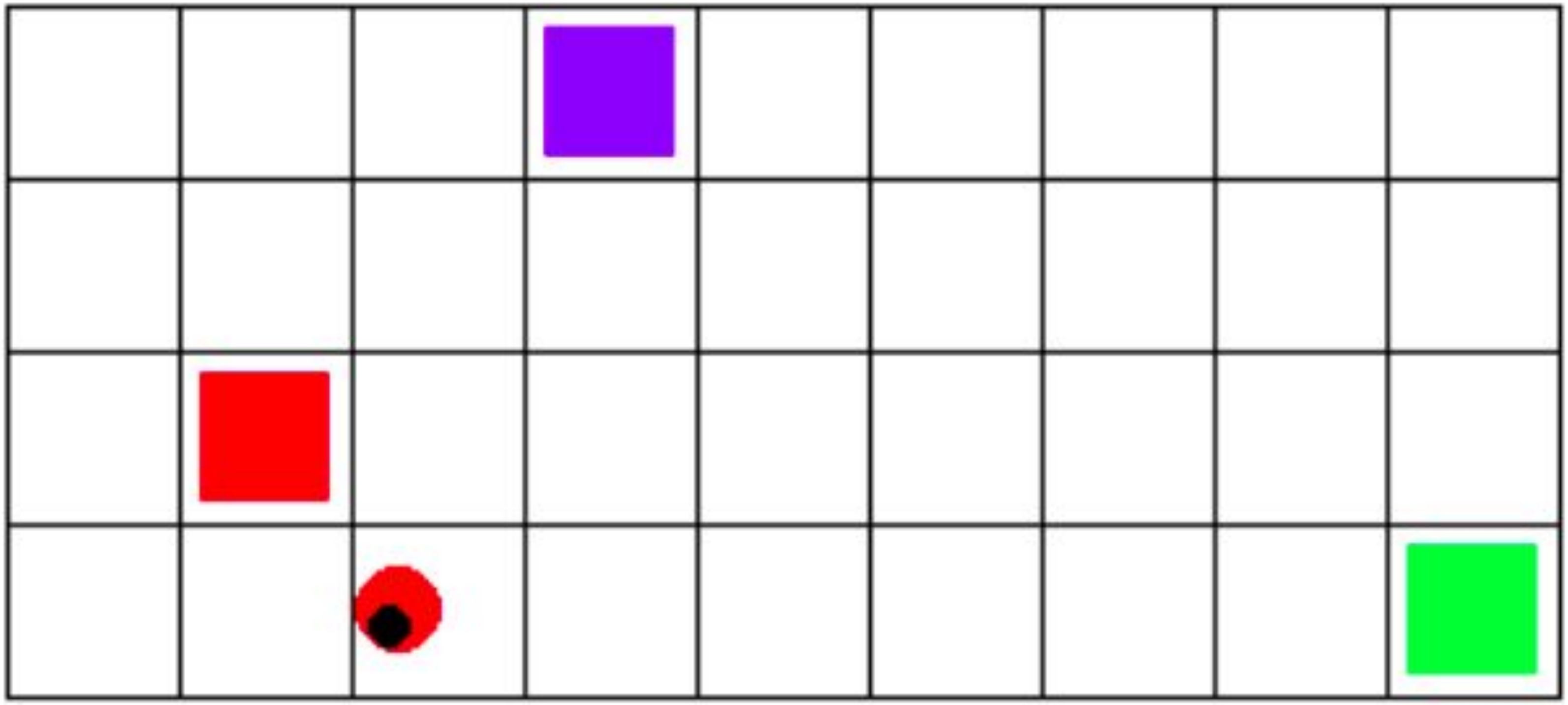




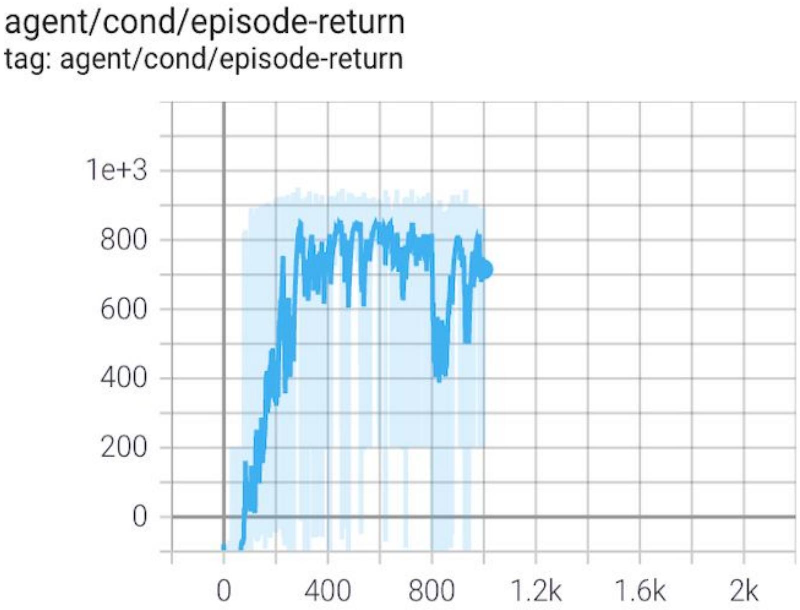
Agent parameters								
Trial	Algorithm name	batch size	memory	exploration	lr	update frequency	reward shaping	episodes
trial1	PPO	64	64	0.3	0.001	20	yes	1000

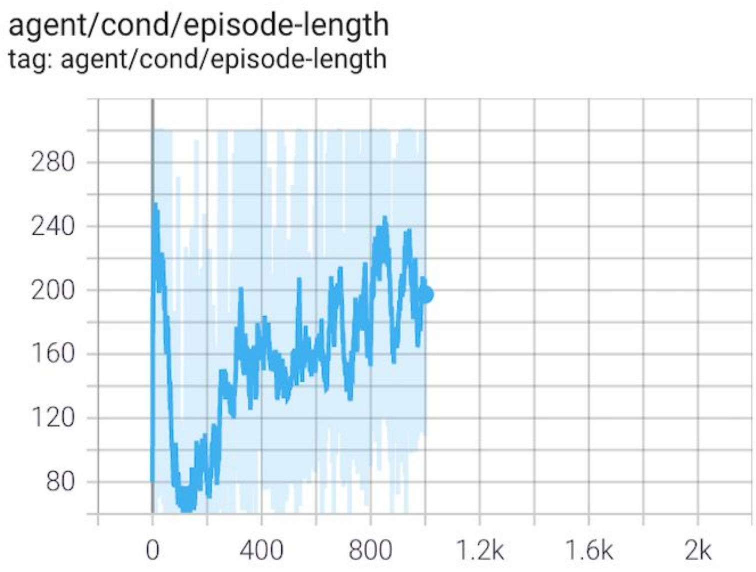
Experiments with three colors

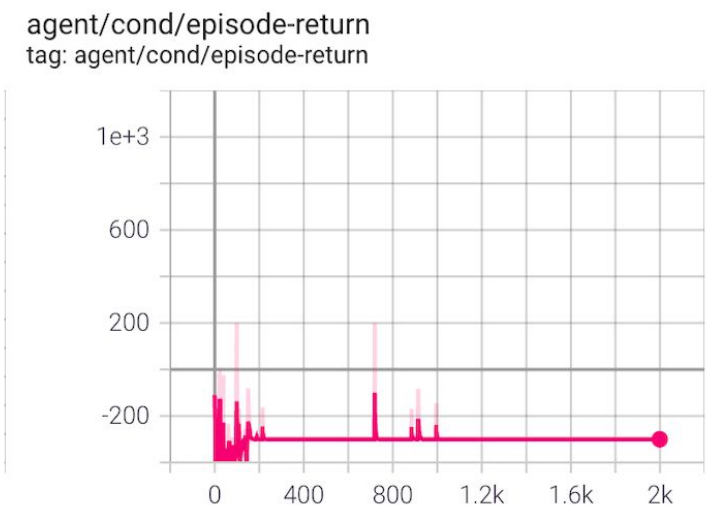
- Gym Sapientino 4x9 map for the experiment {*blue*, *red*, *green*}:

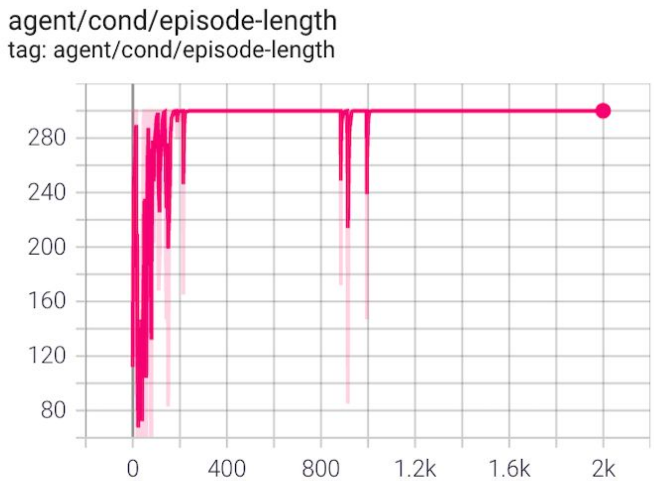


Trials (custom network vs auto network)

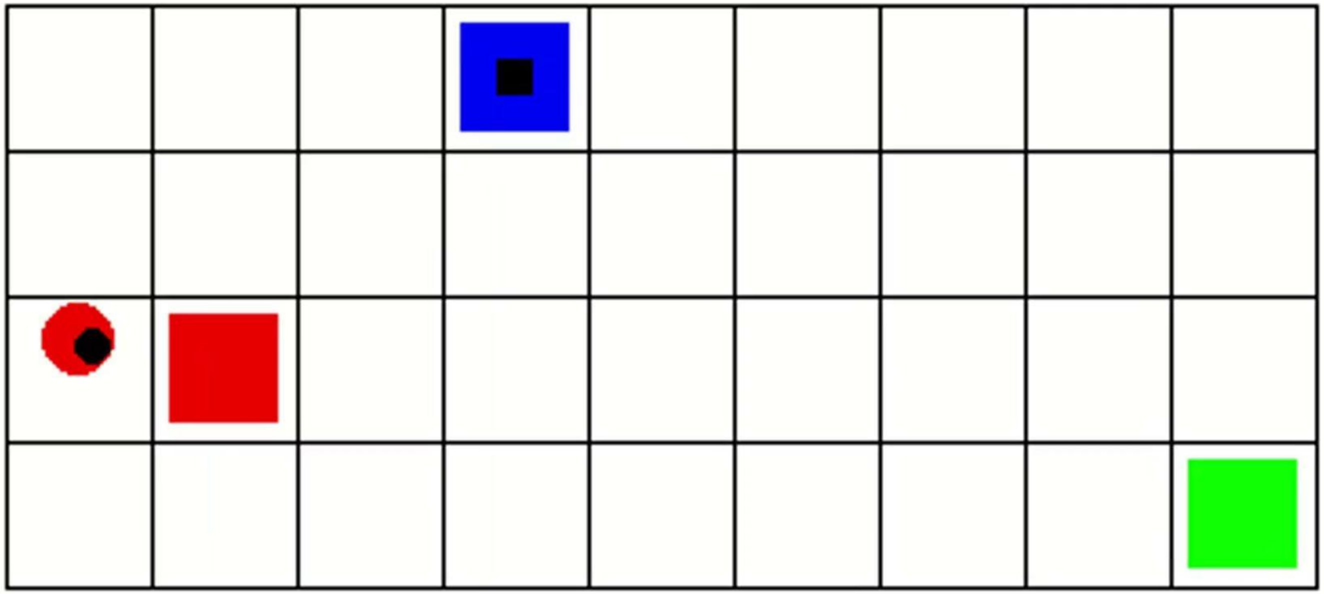








Three colors: convergence



Trial 1 vs Trial 2 -Comparison-

Agent parameters for the three color goal experiments in the 4X9 map.								
Agent name	batch size	memory	hidden size	expl.	lr	update freq.	rew. shap.	episodes
PPO baseline	64	64	64	0.0	10^{-3}	20	yes	2000
Our approach	64	64	192	0.0	10^{-3}	20	yes	1000

Conclusion

- Problems with larger maps: the agent does not frequently sample the goal.
    - Solution: Adding exploration, the agent can probe better the action space.

- Sparse rewards → suboptimal convergence → negative reward action selection.
    - Solution: Reward shaping.

References:

[1] G. De Giacomo and M. Favorito. Compositional approach to translate ltlf/ldlf into deterministic finite automata. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 31, pages 122–130, 2021.

[2] A. D. Laud. *Theory and application of reward shaping in reinforcement learning*. University of Illinois at Urbana-Champaign, 2004.

[3] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[4] https://github.com/cipollone/gym-sapientino-case/tree/master/gym_sapientino_case