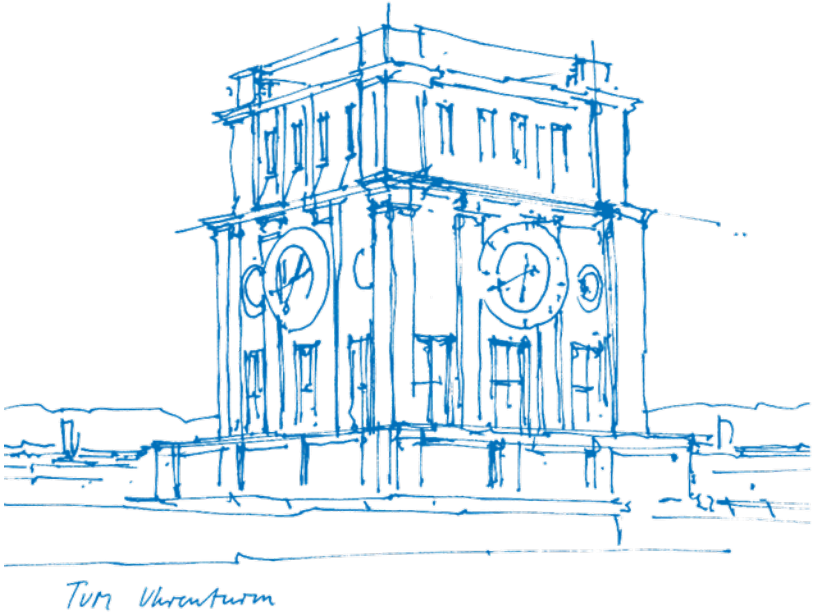
Multi-Agent Reinforcement Learning for Logistics

Anja Kirschner, Leo Tappe, Victor Caceres, Andres Becker, Iheb Belgacem

TUM Data Innovation Lab
MaibornWolff GmbH
Project Lead: Dr. Ricardo Acevedo Cabra

Scientific Lead: Dr. Lenz Belzner, Jorrit Posor
Co-Mentor: Oleh Melnyk
Supervisor: Prof. Dr. Massimo Fornasier

July 30, 2020



Agenda

Introduction

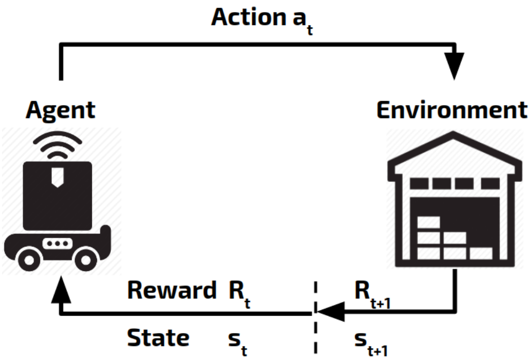
Challenge Proposed

- Project Focus: Chaotic Warehouse
- The Chaotic Warehouse contains bins, item types and transactions of items
- Agents to handle these transactions
- Real Problem scenario to apply Reinforcement Learning

Goals

- Implement working environment that resembles a chaotic warehouse
- Implement visualization capabilities of this environment
- Implement a single agent reinforcement learning algorithm that handles different warehouse complexities
- Implement a multi agent reinforcement learning algorithm mirroring single agent cases
- Compare the performance achieved with a heuristic baseline

Reinforcement Learning Overview



Reinforcement Learning's (RL) idea is to learn from interaction. An agent interacts with the environment in order to maximize the reward in the lung run.

Key elements:

- Reward $R(s_t, a_t) := \mathbb{E}[R_{t+1}|s_t, a_t] = \sum_{r \in \mathcal{R}} r \sum_{s' \in \mathcal{S}} P(s', r|s_t, a_t)$.
- Return $G_t := \sum_{k=0}^{T} \gamma^k R_{t+k+1}$, where $\gamma \in [0, 1]$ is the discount factor and T is the maximum number of time steps per episode; Reward in the long run.
- Policy $\pi : \mathcal{S} \to \mathcal{A}$; Tells the agent which action to take given a state s.
- State-value $V_\pi(s) := \mathbb{E}_\pi[G_t|s_t = s]$; expected return by following π when in state s.
- Action-value $Q_\pi(s, a) := \mathbb{E}_\pi[G_t|s_t = s, a_t = a]$; expected return when taking action a from state s by following π.

This RL problem can be formulated as a Markov Decision Process (MDP). Therefore, all states $s \in \mathcal{S}$ holds the Markov property: $P(s_{t+1}|s_1, \dots, s_t) = P(s_{t+1}|s_t)$ (i.e. future only depends on the current state).

Single Agent RL and Deep Q-Learning

Initialize replay memory D to capacity N;
Initialize action-value function Q with random weights θ;
Initialize target action-value function $\hat{Q}$ with weights $\hat{\theta} = \theta$;
for *episode=1* **to** *M* **do**
  Initialize state s_1;
  for *t=1* **to** *T* **do**
    ε-greedy: With probability ε select a random action a_t, otherwise select a_t=argmax$_a Q(s_t, a|\theta)$;
    Execute action a_t in the Environment and observe reward r_t and next state s_{t+1};
    Store transition (s_t, a_t, r_t, s_{t+1}) in D;
    Sample random minibatch of transitions (s_j, a_j, r_j, s_{j+1}) from D;

$$y_j = \begin{cases} r_j, & \text{if episode terminates at step } j+1 \\ r_j + \gamma \max_{a'} \hat{Q}(s_{j+1}, a'|\hat{\theta}), & \textit{otherwise} \end{cases};$$

    Perform gradient descent step on $(y_j - Q(s_j, a_j|\theta))^2$ w.r.t. θ;
    Set $s_t = s_{t+1}$;
    Every C steps reset $\hat{Q} = Q$;
  end
end

Deep Q-Learning Algorithm was used to address single agent approach.

Function Q_π is critical since it governs the actions taken by the agent on each time step. Then, how to approximate Q_π? Answer: use a deep neural network!

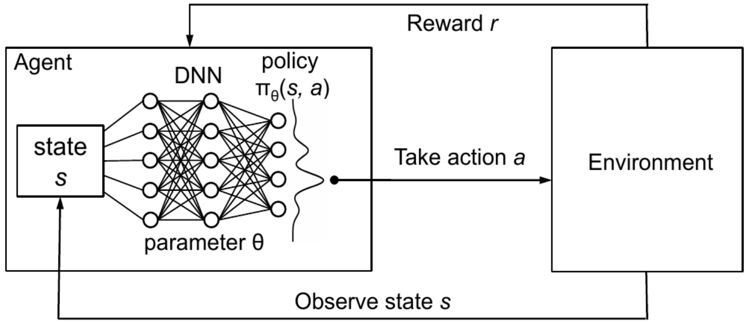
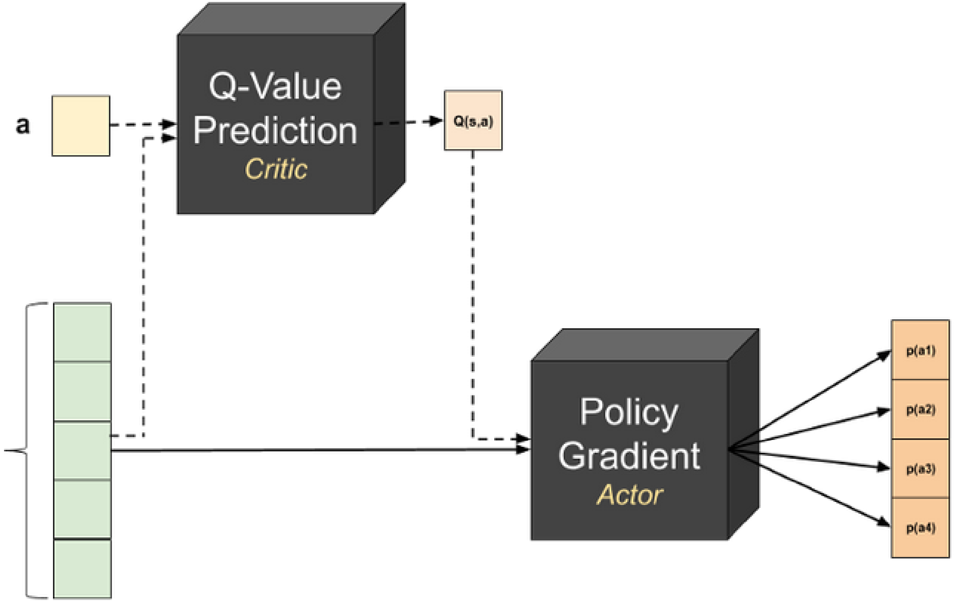


Image source: Mao et al. 2016.

Advantage Actor Critic (A2C)

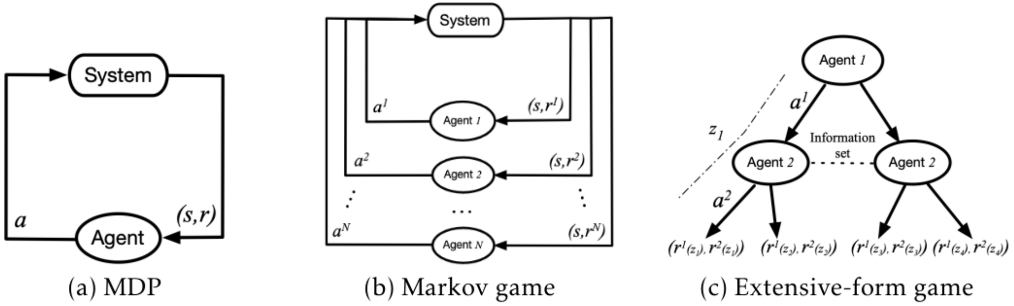
Combines:

- Actor-critic methods:
  - *Critic* estimates value function
  - *Actor* updates the policy distribution in direction suggested by critic
  - Critic and actor functions parametrized with neural networks
- Parallelized training:
  - Multiple agents (actors) run on multiple instances of environment in parallel
  - Synchronously updated global network parameters
  - Parallel actors can start from same policy in next iteration



Architecture of A2C. Image source: *Qrash Course II: From Q-Learning to Gradient Policy Actor-Critic in 12 Minutes* n.d.

Multi-Agent Reinforcement Learning



Multi-Agent Reinforcement Learning Frameworks. Image source: Zhang, Yang, and Başar 2019

Joint Action Learner / Markov Games

- All agents choose their next action simultaneously
- All agents know the actions chosen by the other agents
- Cooperative/Competitive/Mixed settings

Independent Learner / Extensive Form Games

- Agents choose their next actions alternately
- Agents do not know the actions chosen by the other agents
- Agents handle other agents as part of the environment
- Non-cooperative settings (in general)

Curiosity-driven Exploration

Curiosity-driven exploration is a popular approach to address the sparse rewards problem.
Goal : To increase the agent's knowledge of the environment.
Main Idea : The agent learns to predict S_{i+1} using S_i and A_i .
The prediction error is larger for regions the agent has not explored well yet.
The agent tries to optimize the sum between the instrinsic and extrinsic rewards.
Intrinsic reward = prediction error.

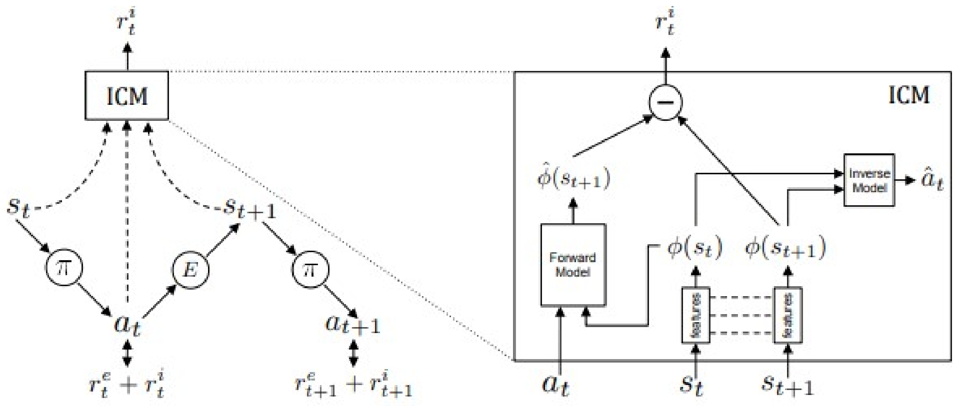
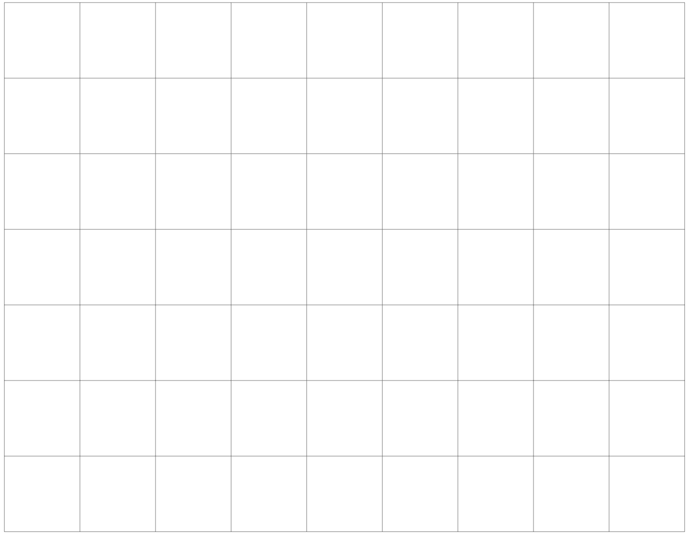


Image source: Pathak et al. 2017.

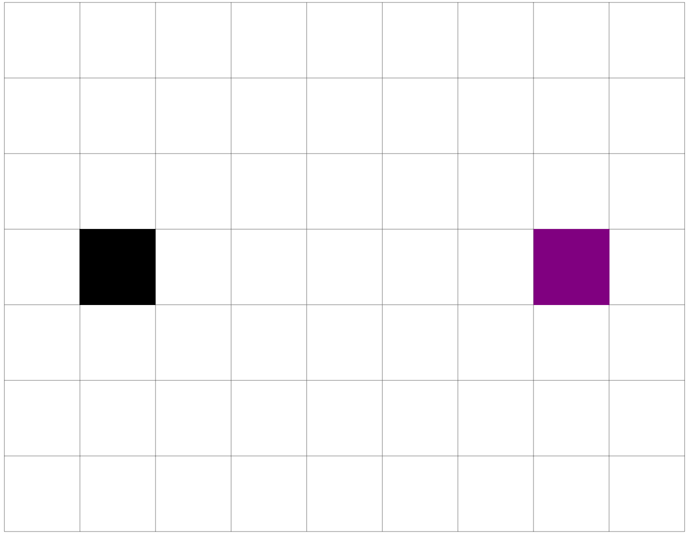
Modelling

We model the warehouse as a bounded 2D grid.



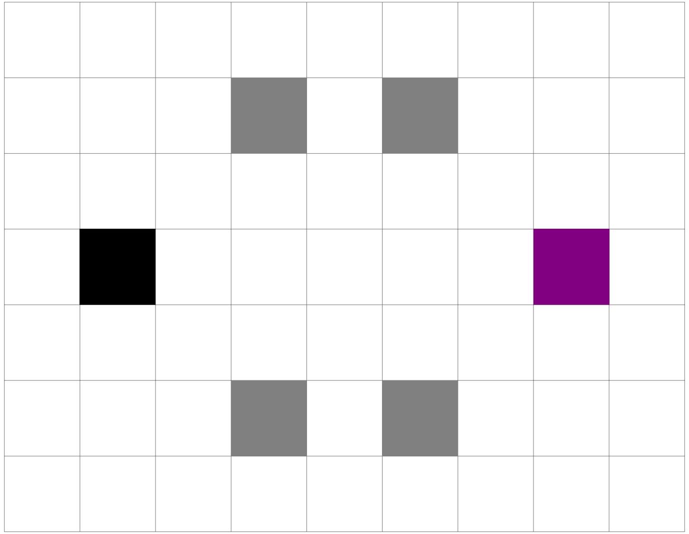
Staging Areas

Items to be stored in warehouse appeared in *staging-in area* ●. (*Inbound transactions*)
Requests for items appear in *staging-out area* ●. (*Outbound transactions*)



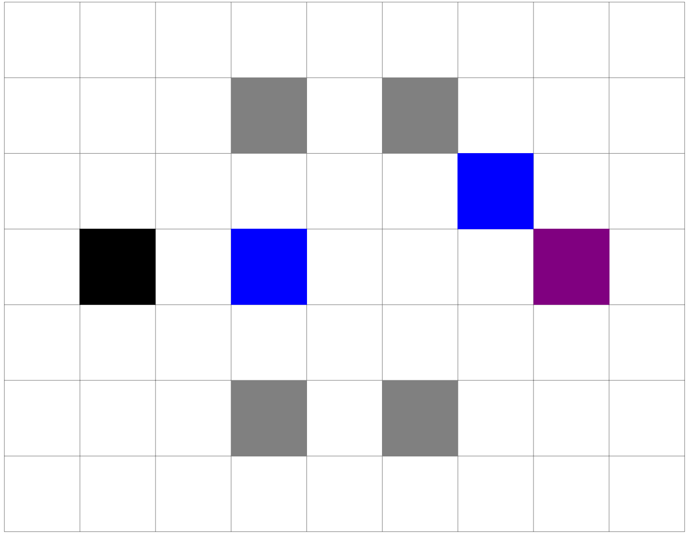
Bins

Items can be stored in *bins* ●.



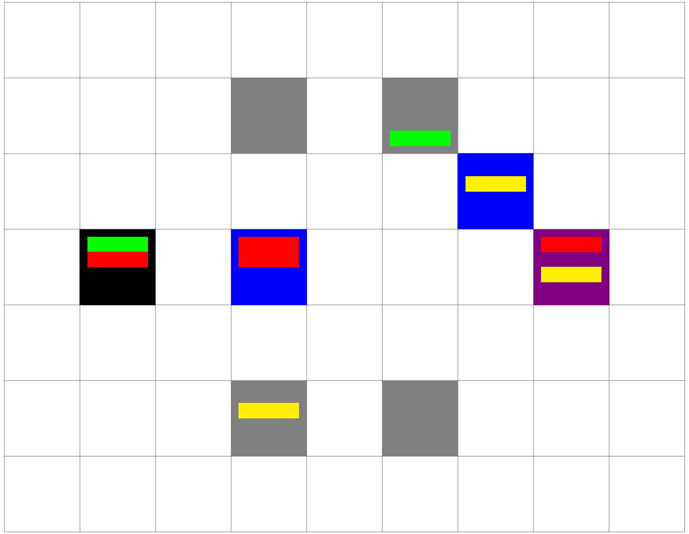
Agents

One or multiple *agents* ● navigate the warehouse with the purpose of satisfying transactions.



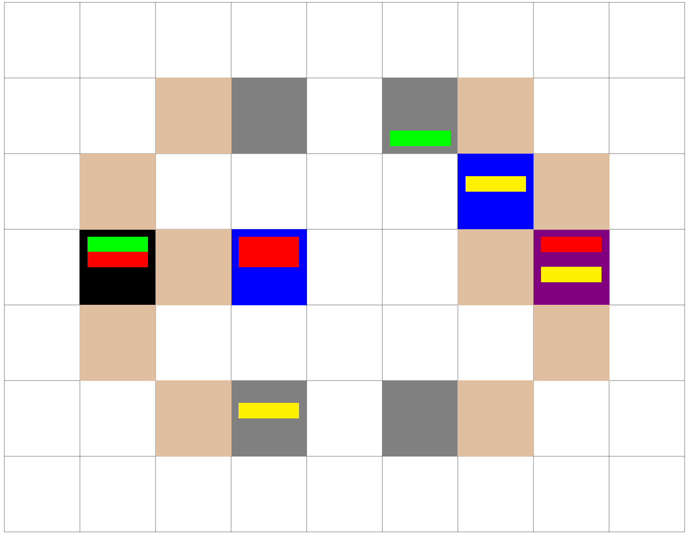
Items

Agents, staging-area and bins have *slots* that can hold *items* of different types (●, ●, ●).



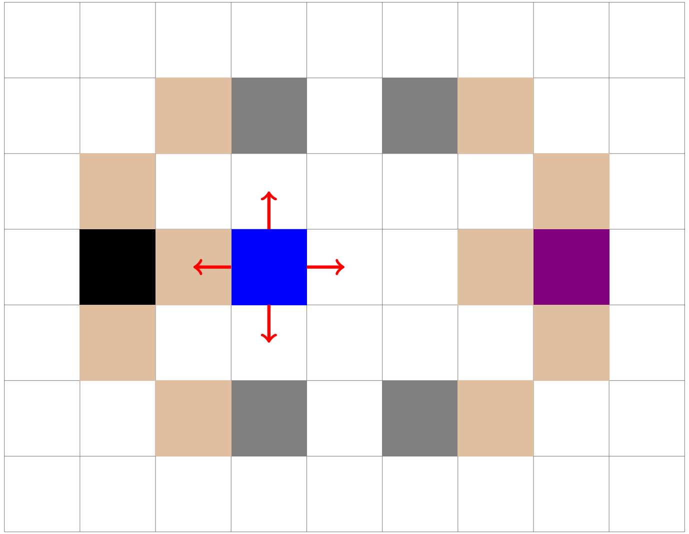
Access spots

In order to interact with a bin or staging-area, an agent has to be in a designated *access spot* ●.



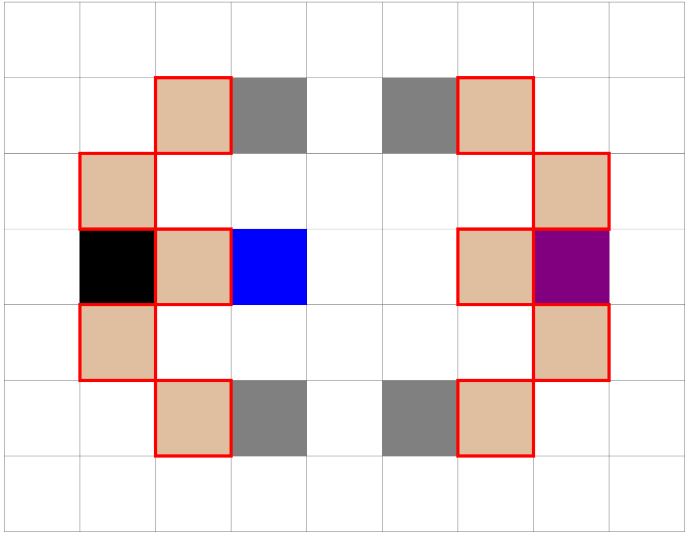
Movement - Low level

In the *low-level movement model*, an agent picks a direction to move in.



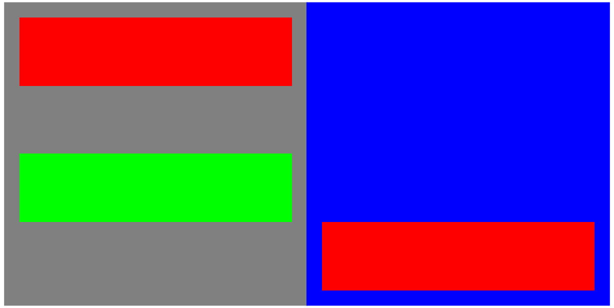
Movement - High level

In the *high-level movement model*, an agent picks a goal location to move to (from the set of access spots).



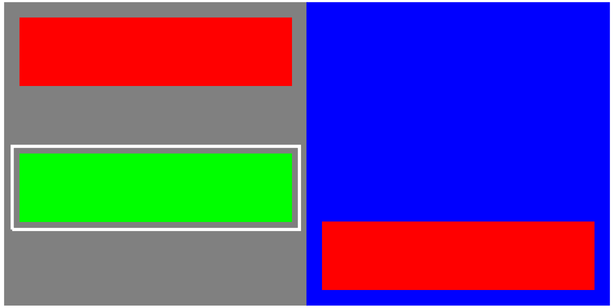
Picking / Putting

In order to interact with a container, an agent has to be in one of the container's access spots. Then, it has to provide three pieces of information:



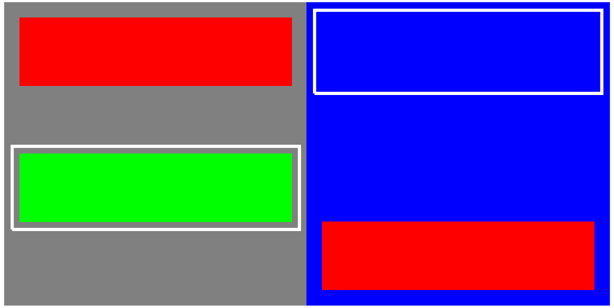
Picking / Putting (1)

1. Which slot of the container to interact with.



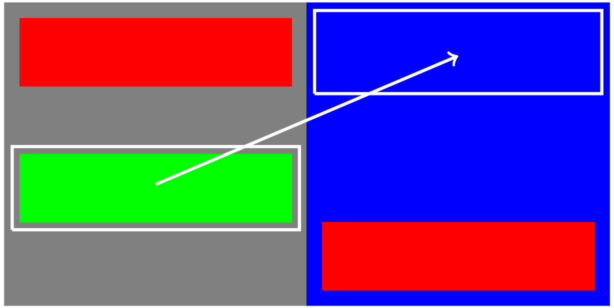
Picking / Putting (2)

2. Which slot of itself to interact with.



Picking / Putting (3)

3. Whether to pick or to put.



Warehouse as MDP

Component	Description
$\mathcal{S}$	Discrete vectors containing positions of agents, holding status of staging areas, bins, and agents ($\mathcal{S} \subset \mathbb{Z}^N$).
$\mathcal{A}$	Movement / pick / put actions, enocoded by a single integer ($\mathcal{A} \subset \mathbb{Z}$).
$\mathcal{R}$	Rewards for completion of transactions, either “dense” (reward for picking/putting single item correctly) or “sparse” (reward only for clearing entire staging area).
$\mathcal{P}$	Actions have deterministic consequences, transactions are generated in stochastic manner. Different transaction generation models were implemented.
T	The environment is episodic with a fixed number of time steps.

The `WarehouseEnv` Gym environment

Standard interface for Reinforcement Learning: *OpenAI Gym* (Brockman et al. 2016).

```
import gym

class WarehouseEnv(gym.Env):

    def __init__(self, filename):
        ...

    def step(self, action):
        ...

    def reset(self):
        ...

    def render(self):
        ...
```

The `step` method

```python
def step(self, action):
    """
    Parameters
    ----------
    action : Movement or pick / put action, encoded as a single integer.

    Returns
    -------
    state : a discrete vector containing the positions of the agents
            and the holding status of the warehouse
    reward : a numerical reward the agent gets for the action
    done : a boolean flag indicating whether the episode is over
    info : additional metadata
    """
    # Modify the warehouse according to the chosen action
    ...
    return state, reward, done, info
```

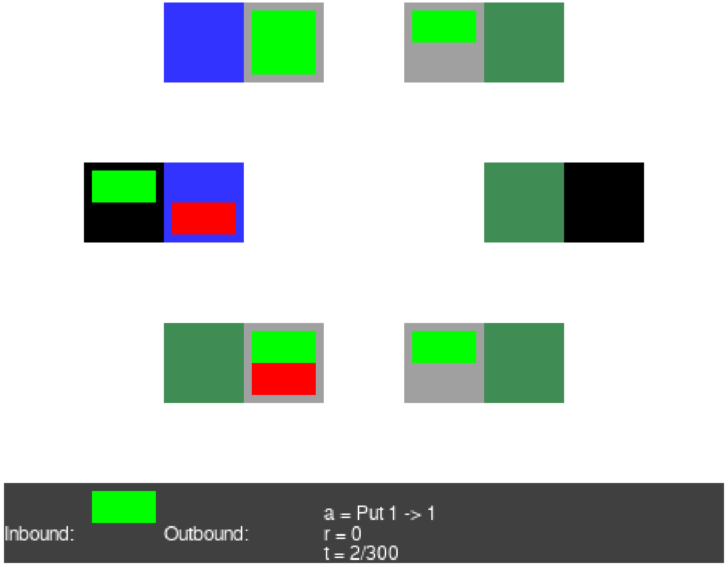
Using the environment

```python
# Create a warehouse environment and an RL agent
env = WarehouseEnv('my-warehouse.json')
agent = MyReinforcementLearningAlgorithm(env)

# Train the agent
agent.train()

# Run the trained agent for an episode
state = env.reset()
done = False
env.render()

while not done:
    action = agent.policy(state)
    state, reward, done, info = env.step(action)
    env.render()
```



A warehouse frame rendered using `pygame` (Shinners 2011).

Heuristic Baseline

A simple heuristic algorithm is developed in order to have a baseline to compare the RL agent against.

Principle :

1. If it can perform a good pick action, do so.
2. Else, if it can perform a good put action, do so.
3. Else, if it can move somewhere where it could perform a good pick action, do so.
4. Else, if it can move somewhere where it could perform a good put action, do so.
5. Else, perform a random action.

Stable Baselines

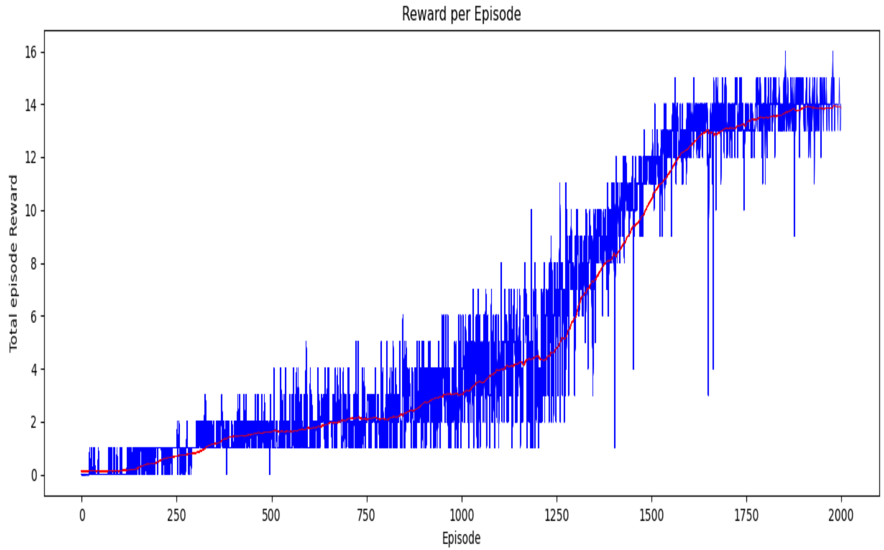
We have used throughout the project for the different RL algorithms the implementations from **stable-baselines**

- Fork of OpenAI Baselines
- Offers instantiations of various Deep Reinforcement Learning Algorithms : Deep Q-Learning, Actor Critic, Deep Deterministic Policy Gradient ...
- Uses Tensorflow to construct the Deep Neural Networks
- Its instentiation of DQN presents several standard enhancement :replay buffer, double Q-Learning, dueling ...
- Provides support for Gym Environments

1 Slot & 1 Item

Environment Conditions

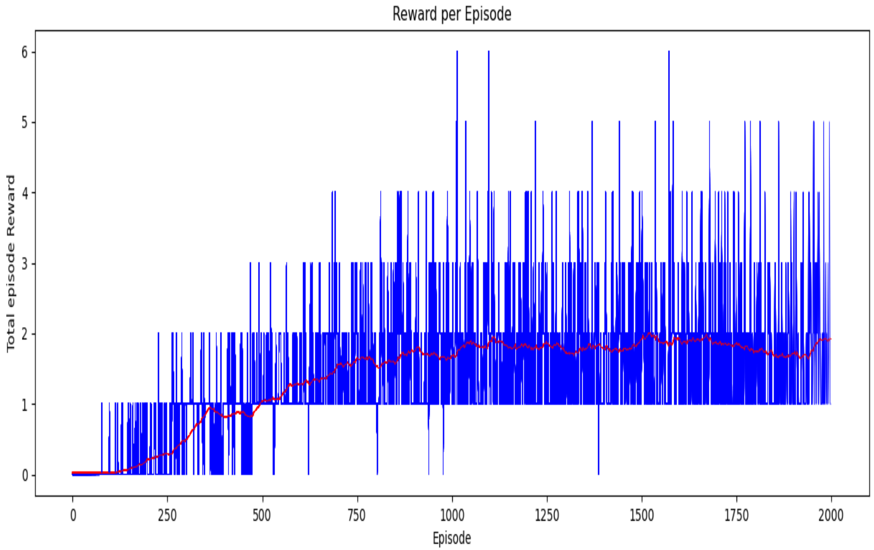
- Environment: 7x9 grid, 4 bins, 1 slot (bin & agent), 1 Item
- Transaction Scheme: Only 1 (random) transaction at a time, with an initial 2
- Sparse Rewards: Only a +1 reward on finishing an outbound transaction
- Network sizes: 32x16
- States: 7000 aprox



Growing Complexity - 2 slots & 2 Items

Environment Conditions

- Environment: 7x9 grid, 4 bins, 2 slots (bin & agent), 2 Items
- Transaction Scheme: Only 1 (random) transaction at a time, with an initial 2
- Sparse Rewards: Only a +1 reward on finishing an outbound transaction
- Network sizes: 512x128x32
- States: 60 000 000 aprox



Simplifying Scenarios

High Level Movement

- Switch to High Level Movement
- Eliminates task of learning how to move
- Able incorporate Distance

Episodic Transactions

- Single one Transaction Episode
- Randomized warehouse
- Continuous Flow in Testing

Curiosity

- Environment side curiosity
- Encourage Exploration
- Attempt to improve training time

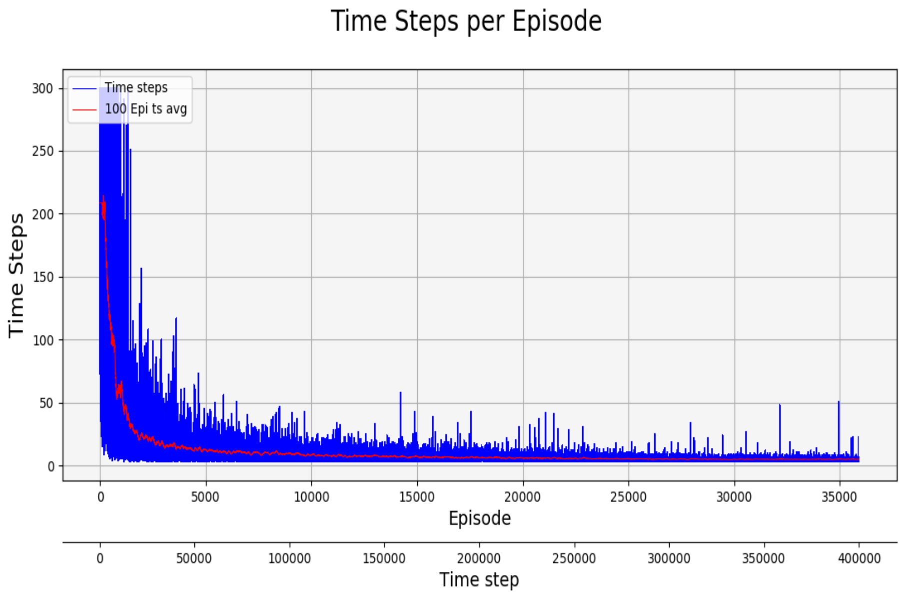
Other Considerations

- Continuous State with one-hot encoding
- Automatic Picking and Putting schemes
- Intermediate Rewards, Punishments

2 Slots & 2 Items

Environment Conditions

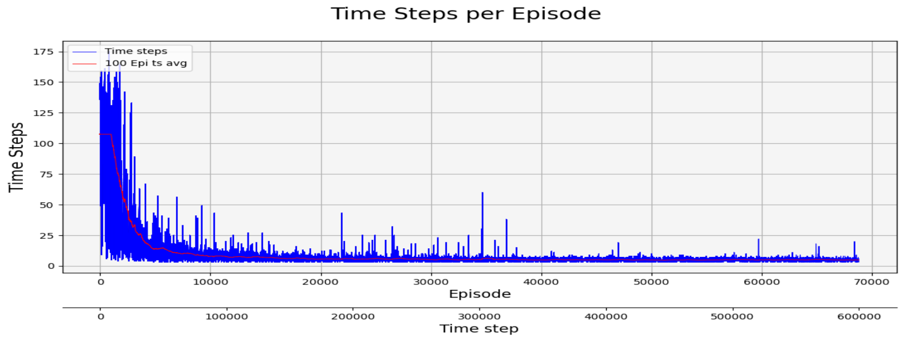
- Environment: 7x9 grid, 4 bins, 2 slots (bin & agent), 2 Items
- Transaction Scheme: 1 Transaction per Episode
- Sparse Rewards: +1 reward on finishing an outbound or inbound transaction
- Network sizes: 512x128x32
- States: 1 000 000 aprox

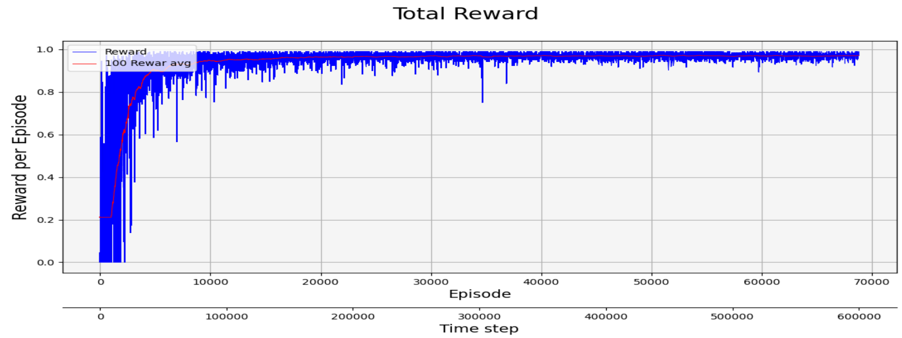


2 Slots & 2 Items - Distance

Environment Conditions

- Environment: 7x9 grid, 4 bins, 2 slots (bin & agent), 2 Items
- Transaction Scheme: 1 Transaction per Episode
- Takes Distance into account
- Sparse Rewards: +1 reward on finishing an outbound or inbound transaction
- Network sizes: 512x128x32
- States: 1 000 000 aprox

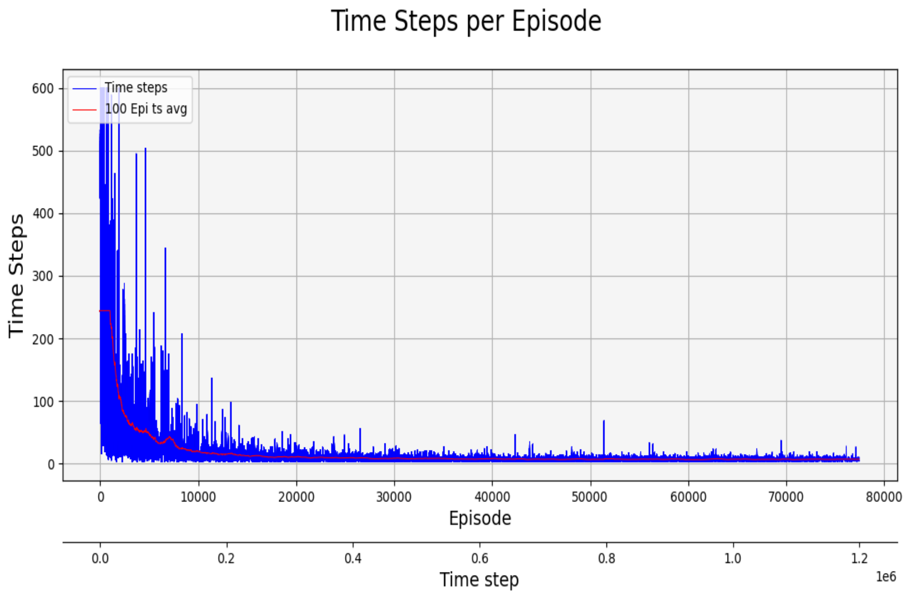




2 Slots & 3 Items

Environment Conditions

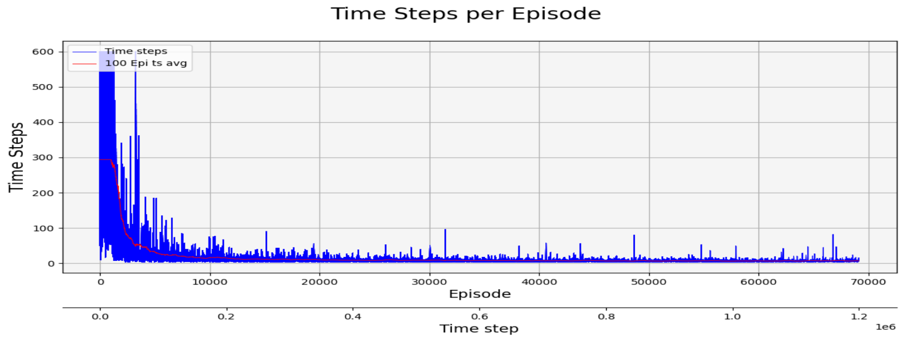
- Environment: 7x9 grid, 4 bins, 2 slots (bin & agent), 2 Items
- Transaction Scheme: 1 Transaction per Episode
- Sparse Rewards: +1 reward on finishing an outbound or inbound transaction
- Training time: Around 14 hours
- Network sizes: 1024x512x128x32
- States: 200 000 000 aprox

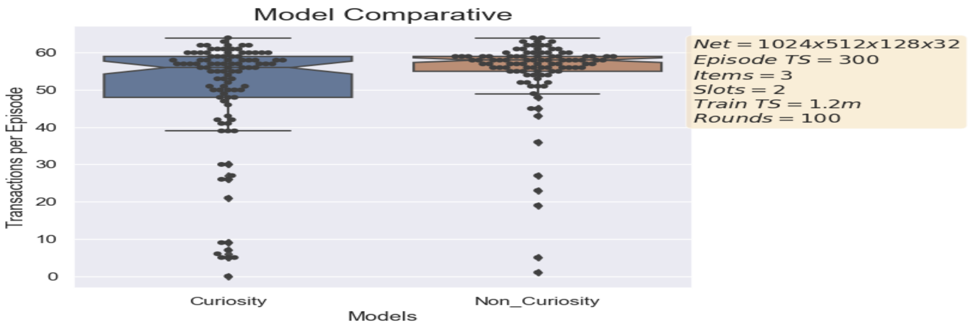


2 Slots & 3 Items - Curiosity

Environment Conditions

- Environment: 7x9 grid, 4 bins, 2 slots (bin & agent), 2 Items
- Transaction Scheme: 1 Transaction per Episode
- Sparse Rewards: +1 reward on finishing an outbound or inbound transaction
- Training time: Around 21 hours
- Incorporates Curiosity Module
- Network sizes: 1024x512x128x32
- States: 200 000 000 aprox



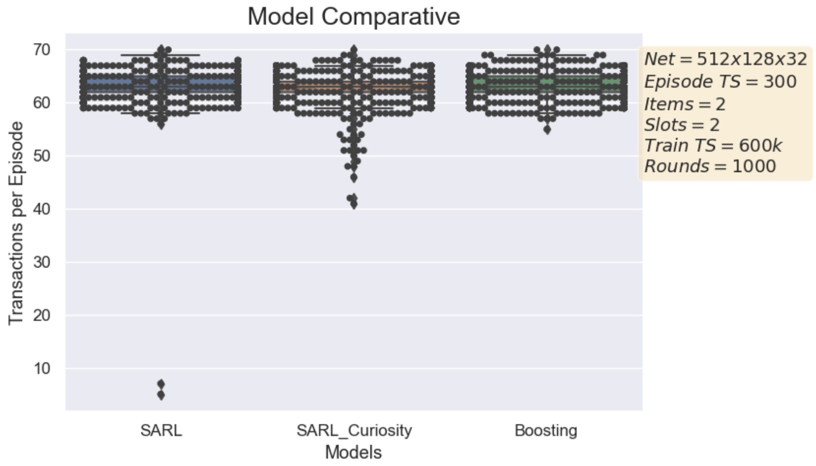


Ensemble Learning

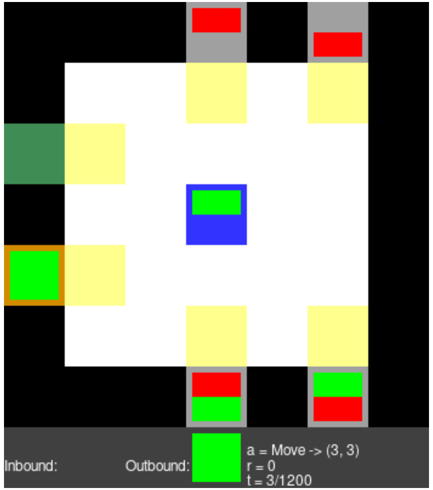
Main idea : Achieving better results by combining several base methods.

- Several RL agents are trained.
- The meta model is obtained by summing up the action selection probabilities for the different agents and selecting the action with the highest probability.

Results : The meta agent has a more stable performance (No rounds with only few successful transactions).



Low Level Movement Environment



Key Elements:

- Action Space size (Movement+Interaction): size of $\{\uparrow, \downarrow, \leftarrow, \rightarrow\} \cup \{Picking\_Actions, Dropping\_Actions\}$, i.e. $4 + 2 \cdot S_{bin} \cdot S_{agent}$.
- Observation Space Size: $Agent\_Possible\_Positions \cdot (N_{item} + 1)\hat{}(S_{bin} \cdot (N_{bin} + 2) + S_{agent})$.

General Environment conditions:

- Single agent.
- 7x7 space grid (but only 5x5 for movement) with 4 bins.
- Episodic transactions i.e. episode start with a random transaction, bin status and agent position; episode finish after transaction completion or time limit (1200 time steps).
- Sparse Rewards (reward obtained only after completing current transaction).
- Q-Network architecture: [128, 64, 32].

The number of items, number of bin slots, number of agent slots and training time steps, vary among the 3 experiments.

Increasing Complexity

Environment 1:

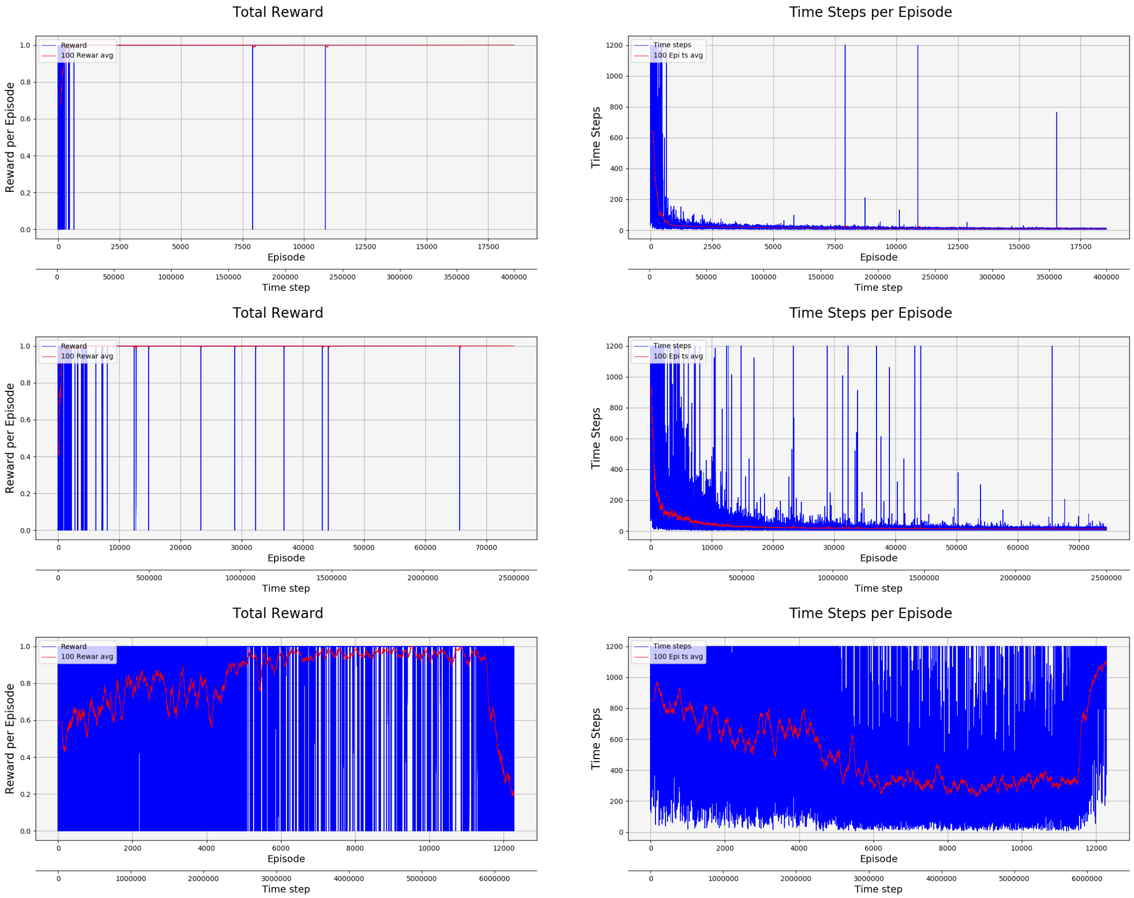
- $400k$ time steps of training.
- 1 bin and agent slot, 3 Items.
- $25 \cdot 4^7 \approx 410k$ environment states.
- $4 + 2 \cdot 1 \cdot 1 = 6$ actions.

Environment 2:

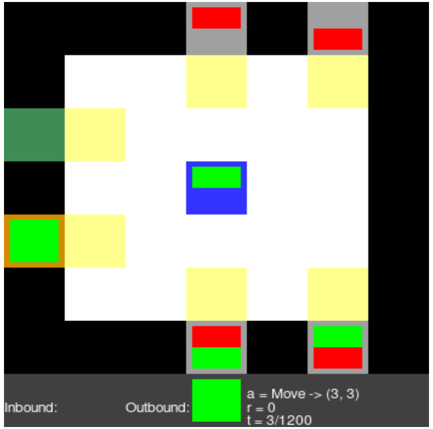
- $2.5m$ time steps of training.
- 2 bin slots, 1 agent slot, 2 Items.
- $25 \cdot 3^{13} \approx 40m$ environment states.
- $4 + 2 \cdot 2 \cdot 1 = 8$ actions.

Environment 3:

- $7m$ time steps of training.
- 2 bin and agent slots, 2 Items.
- $25 \cdot 3^{14} \approx 120m$ environment states.
- $4 + 2 \cdot 2 \cdot 2 = 12$ actions.



Low Level Movement; Best results



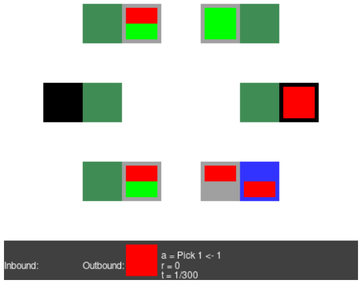
Environment 2 was the most complex environment successfully trained for a moving and interacting agent:

- $2.5m$ time steps of training.
- 2 bin slots, 1 agent slot, 2 Items.
- $25 \cdot 3^{13} \approx 40m$ environment states.
- $4 + 2 \cdot 2 \cdot 1 = 8$ actions.

To achieve satisfactory results for environments more or equally complex than environment 3, more training steps were needed. To make the training process more robust, a curiosity module was implemented. However, the behavior during training was very similar to the one obtained with no curiosity.

Comparison with Heuristic baseline

Performance metric: Number of random transaction completed in 300 time steps. 100 rounds performed on each scenario.

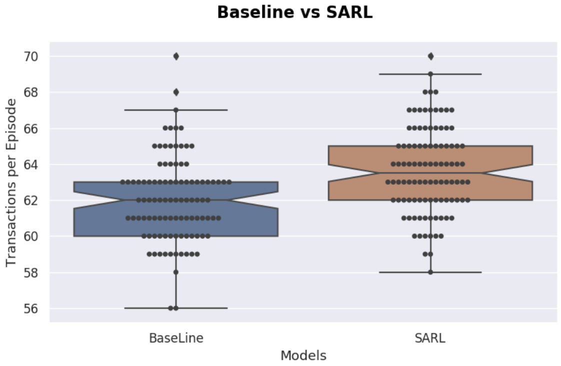


General Environment conditions:

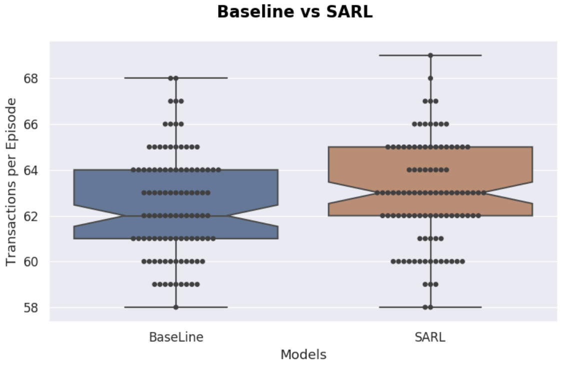
- High-level movement environment
- 4 bins with 2 slots each.
- Single agent with 2 slots.
- Episodic transactions.
- 300 time steps limit.
- Sparse Rewards.

- First scenario:
  2 items (28.7m states), Q-Net. arch. [512, 128, 32], 1.4m training ts.
- First scenario:
  3 items (1611m states), Q-Net. arch. [1024, 512, 128, 32], 2.4m training ts.

Action space size for both scenarios was 14.
RL approach performed better than the heuristic baseline in both cases.



First scenario: 2 items.

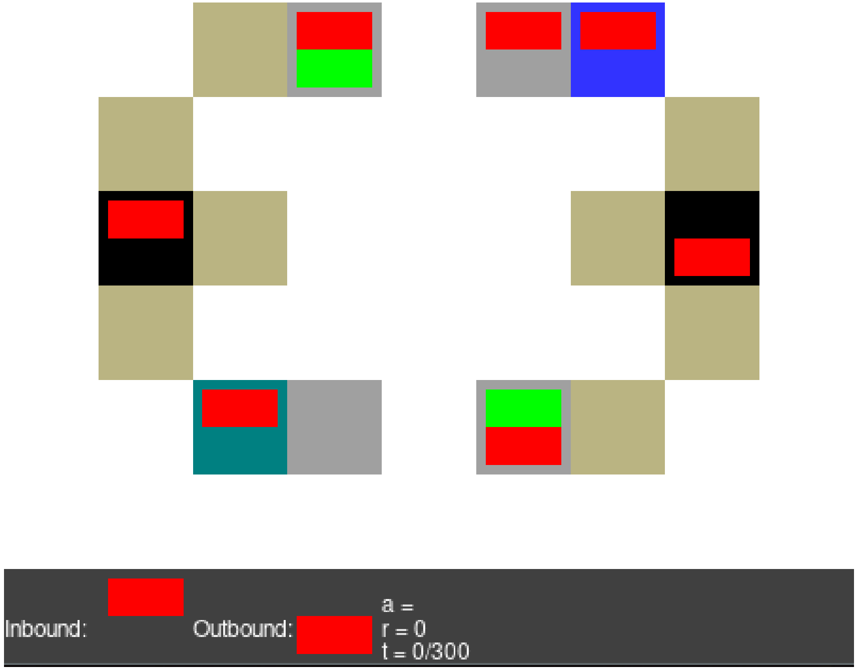


Second scenario: 3 items.

MARL medium

Environment Conditions

- Environment: 7x9 grid, 4 bins, 2 slots (bin & agent), 2 Items, 2 Agents
- Transaction Scheme: Episodic
- Rewards: +1 reward on finishing an outbound or inbound transaction or putting an item to a bin
- A2C: Number of vectorized environments: 4
- DQN: Network sizes: 512x128x32
- Training time: 600,000 steps

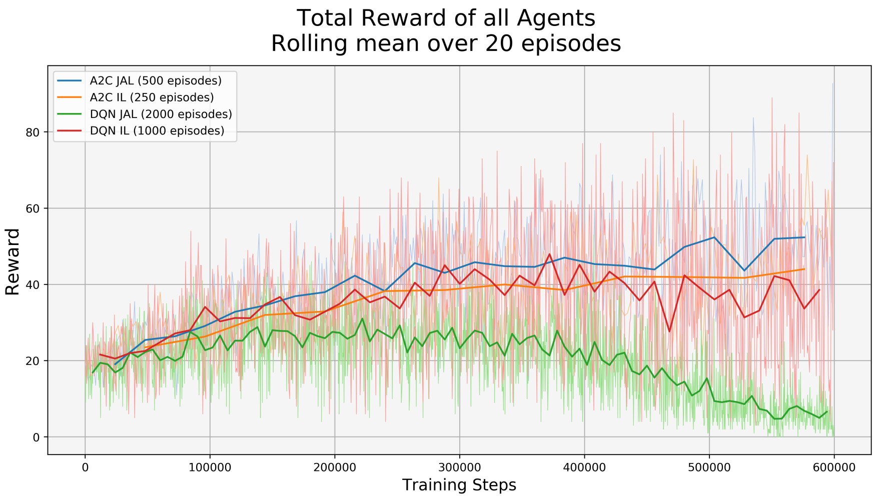


Environment

MARL medium

Environment Conditions

- Environment: 7x9 grid, 4 bins, 2 slots (bin & agent), 2 Items, 2 Agents
- Transaction Scheme: Episodic
- Rewards: +1 reward on finishing an outbound or inbound transaction or putting an item to a bin
- A2C: Number of vectorized environments: 4
- DQN: Network sizes: 512x128x32
- Training time: 600,000 steps

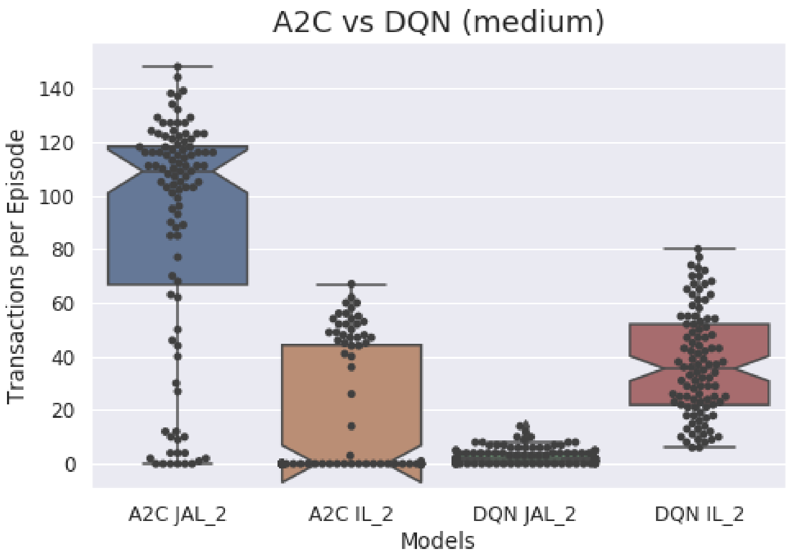


Total reward over training time.

MARL medium

Environment Conditions

- Environment: 7x9 grid, 4 bins, 2 slots (bin & agent), 2 Items, 2 Agents
- Transaction Scheme: Episodic
- Rewards: +1 reward on finishing an outbound or inbound transaction or putting an item to a bin
- A2C: Number of vectorized environments: 4
- DQN: Network sizes: 512x128x32
- Training time: 600,000 steps

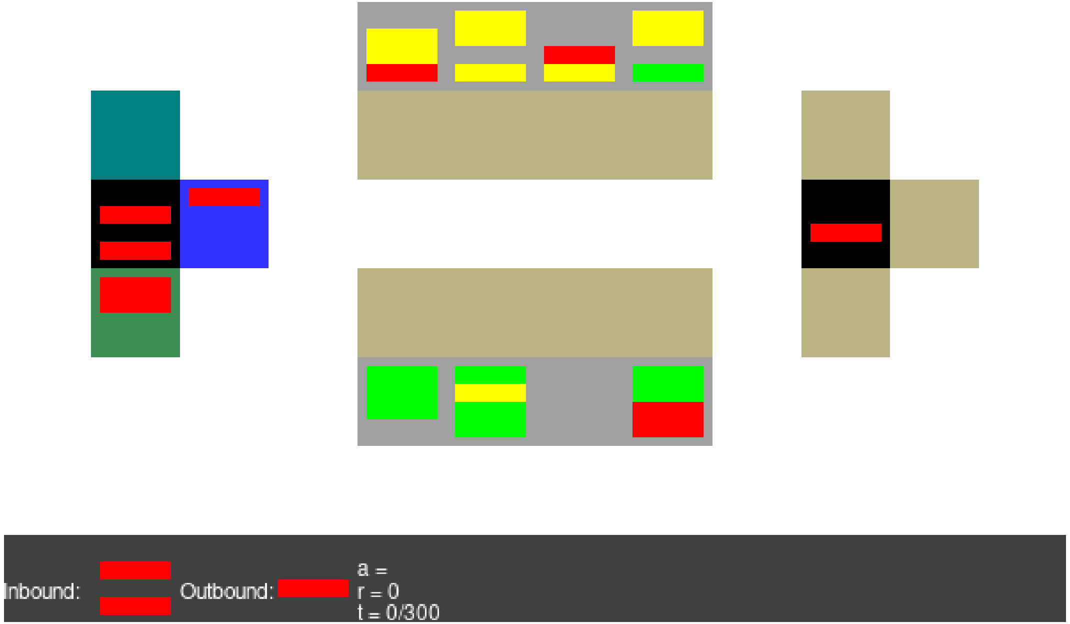


Completed random transactions in 100 rounds of 300 time steps.

MARL large

Environment Conditions

- Environment: 9x12 grid, 8 bins, 4 bin slots, 3 items, 3 agents, 2 agent slots
- Transaction Scheme: Episodic
- Rewards: +1 reward on finishing an outbound or inbound transaction or putting an item to a bin
- A2C: Number of vectorized environments: 4
- DQN: Network sizes: 512x128x32
- Training time: 600,000 steps

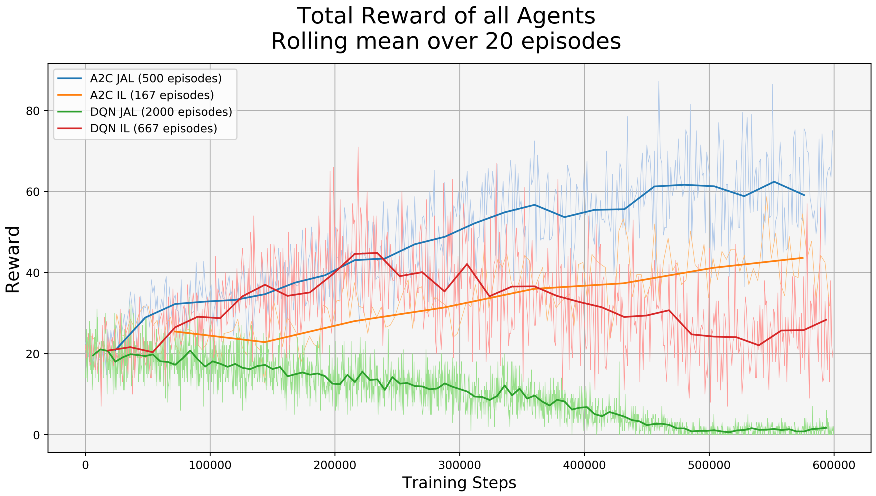


Environment

MARL large

Environment Conditions

- Environment: 9x12 grid, 8 bins, 4 bin slots, 3 items, 3 agents, 2 agent slots
- Transaction Scheme: Episodic
- Rewards: +1 reward on finishing an outbound or inbound transaction or putting an item to a bin
- A2C: Number of vectorized environments: 4
- DQN: Network sizes: 512x128x32
- Training time: 600,000 steps

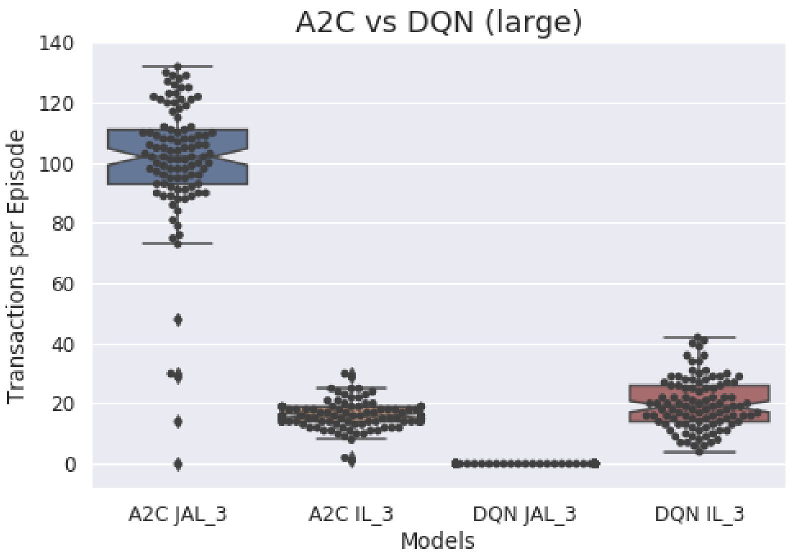


Total reward over training time.

MARL large

Environment Conditions

- Environment: 9x12 grid, 8 bins, 4 bin slots, 3 items, 3 agents, 2 agent slots
- Transaction Scheme: Episodic
- Rewards: +1 reward on finishing an outbound or inbound transaction or putting an item to a bin
- A2C: Number of vectorized environments: 4
- DQN: Network sizes: 512x128x32
- Training time: 600,000 steps



Completed random transactions in 100 rounds of 300 time steps.

Conclusions and Outlook

Conclusions

- A flexible Gym warehouse environment was implemented to resemble a Chaotic Warehouse
- Reinforcement Learning can be applied to a Chaotic Warehouse, however, there is exponential growth in complexity
- Key is to reduce complexity in scenarios: Our solutions: High level movement and Episodic Transactions
- Curiosity might not be helpful in this type of scenario, as it does not change with progress
- Single agent case is able to surpass our baseline’s performance
- Multi agent adds complexity and additional modeling problems

Outlook

- Keep increasing complexity, assisted with better hardware, software and other scenario simplifications
- Explore issues with local optimas due to simplifying scenarios
- Further investigate multi agent reinforcement learning using suitable supporting frameworks

References

- Brockman, G. et al. (2016). *OpenAI Gym*. eprint: `arXiv:1606.01540`.
- Mao, H. et al. (2016). "Resource Management with Deep Reinforcement Learning". In: pp. 50–56.
- Pathak, D. et al. (2017). *Curiosity-driven Exploration by Self-supervised Prediction*. arXiv: `1705.05363 [cs.LG]`.
- *Qrash Course II: From Q-Learning to Gradient Policy Actor-Critic in 12 Minutes* (n.d.). `https://towardsdatascience.com/qrash-course-ii-from-q-learning-to-gradient-policy-actor-critic-in-12-minutes-8e8b47129c8c`. Accessed: 2020-07-22.
- Shinners, P. (2011). *PyGame*. `http://pygame.org/`.
- Zhang, K., Z. Yang, and T. Başar (2019). "Multi-Agent Reinforcement Learning: A Selective Overview of Theories and Algorithms". In: *ArXiv* abs/1911.10635.