Flatland-RL : Multi Agent Reinforcement Learning on Trains

Sharada Mohanty* Erik Nygren* Adrian Egli Marcel Salathé

mohanty@aicrowd.com
erik.nygren@sbb.ch
adrian.egli@sbb.ch
marcel@aicrowd.com

March 9, 2019

Abstract

The *Flatland Challenge* is a competition to facilitate the progress of multi-agent reinforcement learning for any vehicle re-scheduling problem (VRSP)[5]. The challenge addresses a real world problem faced by many transportation and logistics companies around the world (e.g. Swiss Federal Railway). Using reinforcement learning (or operations research methods) the participants must solve different tasks related to VRSP on a simplified 2D multi-agent railway simulations environment called as *Flatland*, released just in context of this competition. Insights and prototypes from this challenge will influence and shape the way modern traffic management systems are implemented not only in railway but also in other areas of transportation and logistics. We aim to establish these series of proposed tasks as a long-term benchmark for the Multi-Agent Reinforcement Learning Community to measure their progress over time.

Keywords

multi-agent reinforcement learning, operations research, vehicle re-scheduling problem, automated traffic management system

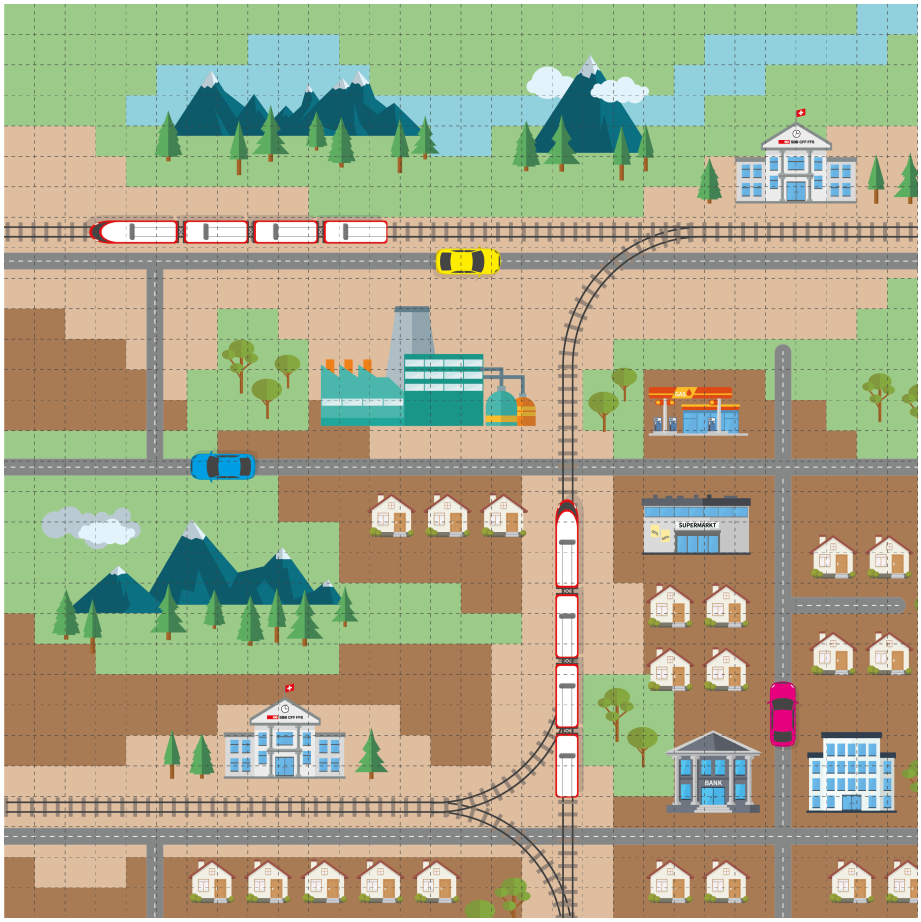
Competition type

The *Flatland* - Challenge will be a regular challenge.

*These authors have equal contribution

FLATLAND

1 Competition description



1.1 Background and impact

The Swiss Federal Railway Company (SBB) operates the most dense mixed railway traffic in the world. SBB maintains and operates the biggest railway infrastructure in Switzerland. Today, there are more than 10,000 trains running each day, being rerouted at over 13,000

switches and managed by over 32,000 signals. Almost half of all goods within Switzerland and 1.2 million passengers are transported on this railway network each day. Due to the growing demand for mobility, SBB needs to increase the transportation capacity of the network by approximately 30%.
The increase in transport capacity can be achieved through different measures such as denser train schedules, large infrastructure investments, and/or investments in new rolling stock[12]. However, the SBB currently lacks suitable technologies and tools to quantitatively asses these different measures.
A promising solution to this dilemma is a complete railway simulation that efficiently evaluates the consequences of infrastructure changes or schedule adaptations for network stability and traffic flow. A complete railway simulation consists of a full dynamical physics simulation as well as an automated traffic management system (TMS).
The research group at SBB has developed a high performance simulator which simulates the dynamics of train traffic as well as the necessary elements of the railway infrastructure. We are currently investigating the possibility of an automated TMS [1]. The role of the traffic management system is to select routes for all trains and decide on their priorities at switches in order to optimize traffic flow across the full network.
The same high performance physical simulation also unfortunately brings with itself enough real world complexity to inhibit fast iteration cycles in experimenting new ideas. Hence, in this work, we propose a simpler 2D grid environment which allows for fast experimentation of ideas, which can eventually be applied to the high performance physical simulation.
At the core of this work, we are trying to solve the more general vehicle re-scheduling problem proposed by Li, Mirchandani and Borenstein in 2007 [5] :

The vehicle rescheduling problem (VRSP) arises when a previously assigned trip is disrupted. A traffic accident, a medical emergency, or a breakdown of a vehicle are examples of possible disruptions that demand the rescheduling of vehicle trips. The VRSP can be approached as a dynamic version of the classical vehicle scheduling problem (VSP) where assignments are generated dynamically.

The "Flatland" Competition aims to address the VRSP by providing a simplistic grid world environment and allowing for diverse solution approaches from the fields of reinforcement learning and operations research.
The problems are formulated as **a 2D grid environment with restricted transitions between neighbouring cells** to represent railway networks. On the 2D grid, multiple agents with different objectives must collaborate to maximize global reward. There are different tasks that need to be solved as explained in section 1.4.
The vehicle re-scheduling problem is of the same nature as the job shop scheduling problem [3], aircraft landing problem[17] and many more problems connected to logistics and transportation. Therefore, any promising solution found in this challenge will find many applications across a broad range of industries.

The re-scheduling problem has been a main research topic for operational researches for many decades [8, 2]. In recent years, the advancement of deep learning and machine learning in general have sparked interest for this problem in the deep learning and specifically in the reinforcement learning domain [4, 10, 11].
One of the key goals of this competition is to design the proposed competition in a way that is equalling appealing to both the operations research community and the deep reinforcement learning community.

1.2 Novelty

There have been other multi-agent reinforcement learning challenges where the objective was to learn either collaboration or competition (or both) among the agents, e.g. Pommerman [9], MarlÖ[7], SC2LE [15]. The key aspects in which the proposed challenge differs from the previous challenges are:

- In *Flatland*, the tasks have to be solved by a large group of agents ($n > 1000$), which means that collaboration becomes a much more complex problem.
- The allowed transitions in *Flatland* differ vastly from free movement on a grid. This requires participants to come up with novel approaches to represent the observations spaces in contrast to pixel or grid view observations.
- *Flatland* directly correlates to a real world problem. A feasible solution to this challenge will influence the evolution of railway re-scheduling around the globe.
- *Flatland* provides an easily extensible framework to define novel tasks in an easy and accessible way.

And most importantly, many of the competitions around reinforcement learning and multi agent reinforcement learning, are designed in a way that the challenge organizers of the competitions have the role of asking many of the difficult research questions, and also doing many of the key design decisions for the participants, for example : what is the best way to represent the observation space, reward function, etc. In this competition, we open up the many important questions around representation of observation spaces, etc to the participants. And make sure they have all the information accessible to them to efficiently compute the same at their end. The idea will be to gather many such alternate representations, and eventually include them in the parent library as general purpose functions.
That said, the proposed competition also shares a lot of similarities with above mentioned competitions, and in fact takes inspirations from many of the design decisions of the above mentioned and many other Multi-Agent Reinforcement Learning Environments.

1.3 Data/Resources

The competition revolves around a two-dimensional grid world environment for multi-agent problems, built on top of the MAgent [16] environment. The easily extensible environment can represent a broad array of diverse problems from infectious disease spreading to train traffic management. Our hope is for the environment to serve as an easy to use and accessible test-bed for many multi-agent reinforcement problems from various domains. Much of this work, and our earlier experiments on this library are inspired by the previously mentioned MAgent environment [16], and also by many ideas introduced by Lowe et. al [6] in their work around Mixed Cooperative-Competitive Environments. In the current scope of this work, we focus on the just Cooperative Agents. The following sections describe the formalisms and assumptions around our implementation of this multi agent environment.

1.3.1 Environment

Flatland is a 2D grid environment of arbitrary size, where the most primitive unit is a **cell**. A *cell* is a location in the grid environment represented by two integral coordinates x and y, where $x \in [0, W]$ and $y \in [0, H]$ (where W is the maximum width of the grid world, and H is the maximum height of the grid world).
Each cell has the capacity to hold a single **agent**. [1]
An agent is an entity which resides at a particular cell, and has a direction d which represents the direction the agent is facing, where the integral value $d \in [0, 4)$, each of values representing one of the 4 directions (North, East, South, West) an agent can face in a grid world.
An agent can move to a subset of adjacent cells. The subset of adjacent cells that an agent is allowed to transition to, is determined by a **transition map**.
A single *transition map* is stored as a 4-bit bitmask representing the legality of moving to any of the 4 adjacent cells. As the movements of agents in *Flatland* is directional (eg. a train cannot move backwards/back-and-forth on the track), each cell has 4 such 4-bit transition maps, to encode the information of directionality (e.g. If the agent is facing east, the transition maps available to the agent are different than when the agent is facing south).
This setup allows us to store the information about the constrained transitions using 4x4 bits of information per cell, and at the same time allows us efficient access to this data in $O(1)$ time at a minimal memory expense ($O(WH)$). For example, to store the transition maps for a grid size of 1000x1000, the cost in terms of memory will be 1.9 MB for the whole grid. Section 1.3.3 discusses in more detail about the movement patterns that emerge from this particular approach of representing legality of moves using transition maps.

[1] The implementational details of the environment do allow for the cells to hold multiple agents, but for the sake of simplicity we assume the capacity of a single cell to be 1 in context of this proposed competition.

Flatland is a discrete time simulation. A discrete time simulation performs all actions with constant time step. A single simulation step synchronously moves the time forward in equal duration of time.

1.3.2 Tile Types of Wall-Based Cell Games

(Theseus and Minotaurs puzzle, Labyrinth Game)
The Flatland approach can also be used the describe a variety of cell based logic games. Without going into any detail, it is still worthwhile noting that the games are usually visualized using cell grid with wall describing forbidden transitions (negative formulation).

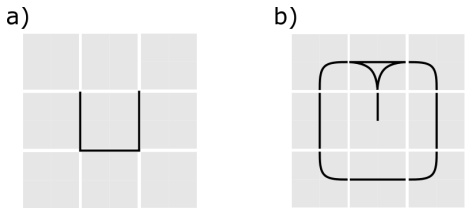


Figure 1: The sub figure on the left (a) shows a classical way of denoting forbidden transitions in grid world environments by a wall. But, we use a transition encoding based approach, where every cell encodes the information about transitions are possible from that cell.

1.3.3 Transition maps and emergent patterns

Each Cell within the simulation grid consists of a distinct transition map which limits the movement possibilities of the agent through the cell.

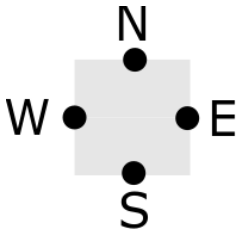


Figure 2: *Each cell can have transitions to its* 4 *neighbouring cells located North,East, South or West.*

For this railway specific problem 8 basic transition maps (as outlined in Figure 3) are defined which describe a real world railway network. For railway network, transition maps allow at most for a binary decision, this means that given the entry direction to a given

cell, at most two transitions are possible. A curved line in a transition map means that the agent also changes orientation while moving into the cell. Changes of agent orientation are only allowed at cells with transition maps including curved lines. Figure 1.3.3 gives an overview of the eight basic types.

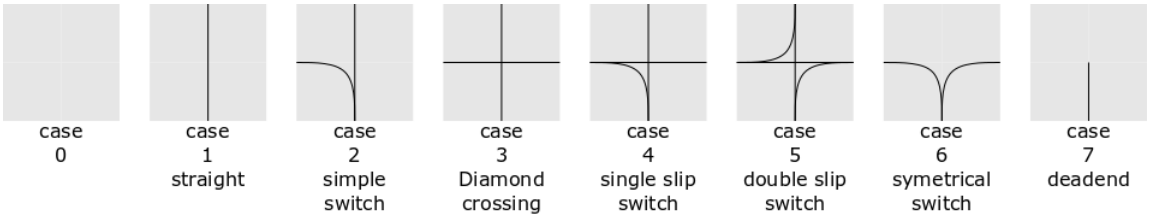


Figure 3: *The railway configuration of Flatland consists of 8 basic transition maps. For each type of transition map all orientations are possible. Curves are a special type of case 1 transition maps where the transition includes a 90 degree rotation. Only transitions between cells connected through lines with no sharp bend are allowed.*

- **Case 0** represents an empty cell, thus no agent can occupy the tile at any time.
- **Case 1** represent a passage through the tile. While on the tile the agent can make no navigation decision. The agent can only decide to either continue, i.e. passing on to the next connected tile, wait or move backwards (moving the tile visited before).
- **Case 2** represents a simple switch thus when coming the top position (south in the example) a navigation choice (West or North) must be taken. Generally the straight transition (S $\rightarrow$ N in the example) is less costly than the bent transition. Therefore in Case 2 the two choices may be rewarded differently in future implementations. Case 6 is identical to case 2 from a topological point of view, however the is no preferred choice when coming from South.
- **Case 3** can be seen as a superposition of Case 1. As with any other tile at maximum one agent can occupy the cell at a given time.
- **Case 4** represents a single-slit switch. In the example a navigation choice is possible when coming from West or South.
- In **Case 5** coming from all direction a navigation choice must be taken.
- **Case 6** : represents a symmetrical switch without a preferred direction. At this switch the agent must either deviate left or right if coming from the South in this example. Coming from East or West this cell acts like curve.
- **Case 7** represents a dead-end, thus only stop or backwards motion is possible when an agent occupies this cell.

As a general consistency rule, it can be said that each connection out of a cell must be joined by a connection of a neighboring cell. We use this consistency rule to generate random valid railway network configurations (Figure 4). These random configurations are used for training while a set of fixed, secret network configurations are used for validation and evaluation.

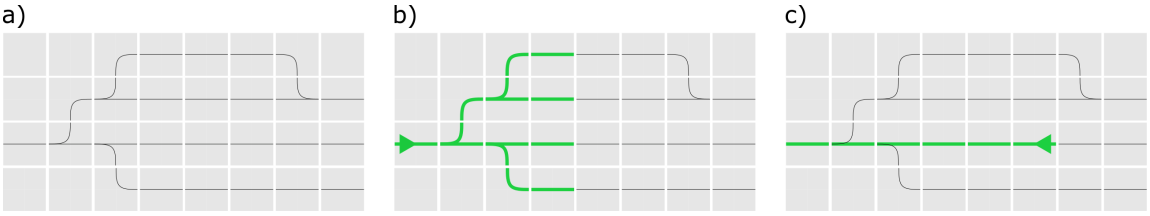


Figure 4: ***a)*** *The transition maps can only be connected if neighbouring cells are connected by an allowed transition. In this way any real world railway network can be implemented within the Flatland environment. The transition maps are direction dependent and thus the possible paths for an agent on Flatland depend not only on its position but also its current direction as shown in sub figures* ***b) & c).***

Agents can only travel in the direction of their current facing direction (details in action-section 1.3.4 below). Thus, the permitted routes for any given agent depend both on its position and on its direction. This is illustrated in the figure 4 on a simple railway network configuration.

1.3.4 Action

The action space is *discrete*(4) for this railway specific implementation of Flatland consisting of:

- Go forward
- Deviate Left
- Deviate Right
- No operation (Halt)

This is distinctly different from freely moving agents in other environments where the heading of an agent can be changed by rotation of the agent.
Deviate left and deviate right can only be executed when a switch is present (in the transition map of the current cell that the agent resides in) with an allowed transition either to the left or to the right (see Fig. 3, cases 2,3,4 and 6). In cases where the deviation is chosen, the heading of the agent is changed simultaneously with the move to the next cell.

Any action is executed if it is allowed by the transition map of the current cell and the subsequent cell is not occupied by another agent. In the case an action cannot be executed the environment executes a *No operation* action and *illegal move* reward is returned to the agent.

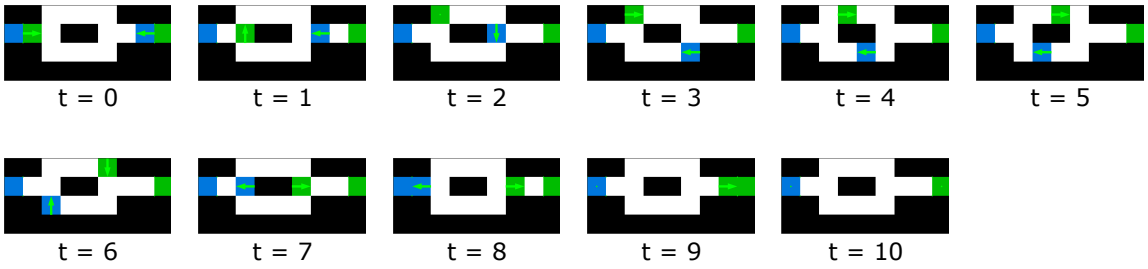


Figure 5: *This figure illustrates a typical situation that can occur between two or more agents. A routing decision needs to be made in order to avoid deadlocks between multiple agents. The complexity grows vastly when considering the effects of each such decision on the full network traffic. Thus problems of this nature cannot be solved using only local information. The figures are take from an earlier version of Flatland.*

1.3.5 Observation

This challenge differs from most other reinforcement learning challenges because the objective for participants is two fold. On one hand they are asked to produces an efficient agent which solves the formulated tasks (Section 1.4) and on the other hand we encourage the participants to come up with their own observation space to improve the performance of their agent.
This is a very strong differentiating factor of this proposed competition from many other similar competitions.
As an inspiration and baseline we provide three different implementations of observation space representations as well as access to the full underlying information of the environment. In addition we provide wrapper functions that can be used to generate novel observation space representations by the participants (something that is greatly encouraged in this competition).

Global observation
The global observation is the simplest observation to get started. In this case every agent is provided a global view of the full flatland environment. This can be compared to raw-pixel data used in many Atari games. The size of the observation space is $h \times w \times c$, where h is the height of the environment, w is the width of the environment and c are the number of channels of the environment. These channels can be modified by the participants but in the initial configuration we include the following 5 channels:

Transition maps	This channel provides a unique value for each type of transition map and its orientation. This gives the agent an overview of the current map.
Agent position	This channel is an one-hot representation of the agents position in the map.
Agent target	This channel is an one-hot representation of the agents target on the map.
Other agents	This is an one-hot representation of all other agent positions on the map.
Other targets	This is an one-hot representation of all the other targets on the map.

This observation space is well suited for single-agent navigation but does not provide enough information to solve the multi-agent navigation task (see Section 1.4), thus participants must improve on this observation space to solve the challenge.

Local grid observation

The local grid observation is very similar to the global observation where we only replace h and w by agent specific dimensions. Each agent has an observation grid of uneven width w_i and arbitrary height h_i which is spanned from its current position. The agent is always situated at the position $(0, (width + 1)/2)$ within the observation grid (Fig. 6), and the observation grid is rotated according to the agents heading such that the full height h_i of the observation grid is in front of the agent.

This is in line with many other similar grid world based Multi Agent RL works [16][6].

The initial local grid view provides the same channels as the initial global view introduced above.

This observation space offers benefits over the global view mostly by reducing the amount of irrelevant information in the observation space. Global navigation with this local observation would be impossible if no indication of target direct where given. We therefore compute a distance map for every agent-target and provide this distance map as an additional channel.

Additional channel:

Distance map	This additional channel in the local observation grid allows for a sense of direction without the need of a global view.

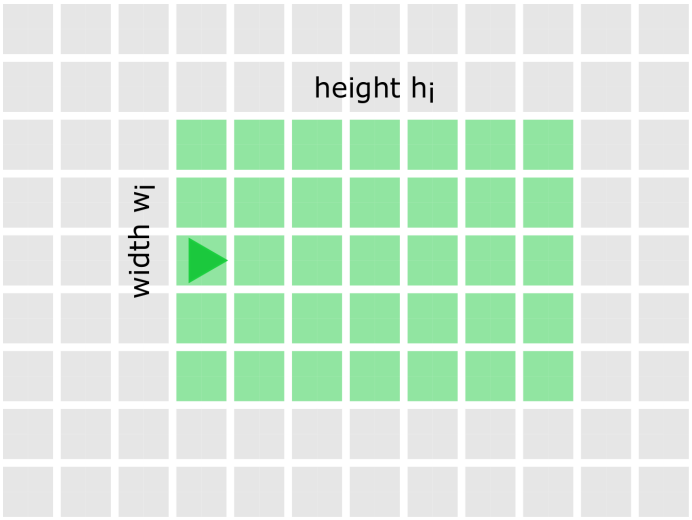


Figure 6: *An abstract visualization of the local field of view of an agent. The green boxes represent visible cells in the agents field of view. This field of view is turned as the agents heading changes.*

Local tree search
The most task specific observation space is a local tree search on the railway network. This method takes the transition maps into account when generating the observation for each agent. There are several different possibilities how to aggregate information along the tree search. For the Flatland environment with railway transition maps we provide the following tree search observation.

1. Start at the agent position.
2. Take all allowed transitions given the initial heading.
3. Collect information at given cell.
4. If max tree depth is reached stop.
5. Check if cell is a switch (more than one allowed transition detected)
    - If switch store aggregated information in previous tree node, branch tree and continue process from step 2 for each branch
    - If no switch is detected return to step 2

What information is collected at each cell can be adjusted by the participants of the challenge. However, the initial configuration will collect the following information:

Agent detected distance	Current depth
Agent detected	0 or 1
Target detected	Current depth
Switch detected	Current depth
Distance map value	$\in [0, 1]$

In order to aggregate this information to the appropriate tree node we always select the minimal value detected, except for **Agent detected** which will be summed, before reaching the next switch. The reasoning behind this is to give the agents an implicit sense of their surrounding and the traffic situation around them. The aggregated tree is flattened to an array, and empty nodes are represented by 0.
Figure 7 illustrates the tree search on a simple network example where the switches are highlighted and numbered in order to identify them with their corresponding nodes in the tree.

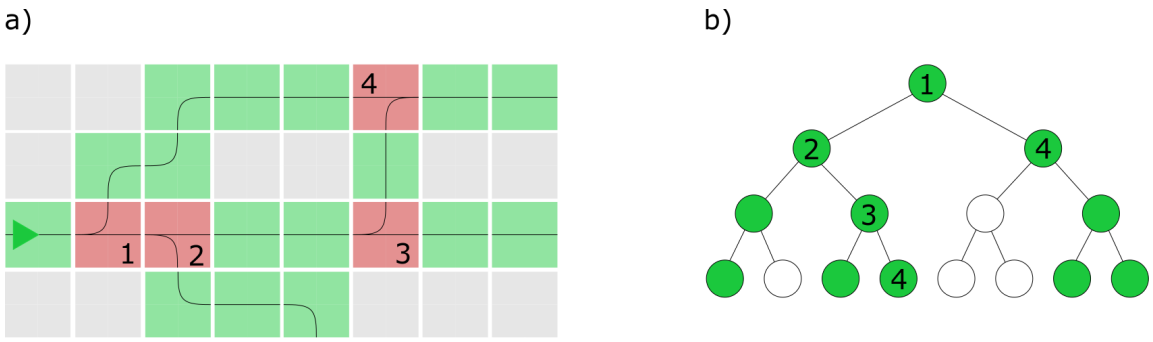


Figure 7: *An abstract visualization of the local tree search observation space. The tree is calculated from the position of the agent along the heading of the agent. Figure **a)** shows the switches (in red) where the tree search branches and the intermediate cells (green) where information is aggregated. On the right (Figure **b)**) we see the corresponding tree, where green circles indicate nodes with an aggregated value and white circles indicate that no value has been collected at the corresponding node.*

Individual observations

By investigating the performance of the three different observation spaces it quickly became clear that one of the important questions that needs to answer is indeed what is the best way to represent the observation space. We therefore decided to include the observation space as part of the challenge objective. Participants are incentivized to contribute observations spaces representations by setting aside a community contribution award as mentioned in section 3.2.

All predefined observations can be modified and combined in order to achieve better performance in this challenge. Furthermore, the full Flatland state information can be accessed and aggregated into new agent observations by using our predefined wrapper functions or implementing new ones.

1.3.6 Reward

Each agent receives combined reward consisting of a local and global reward signal. Locally the agent receives $r_l = -1$ for each time step t and $r_l = 0$ for each time step after it has reached its target location. In addition an illegal move reward $r_i = -2$ is received when the action cannot be executed. The global reward signal $r_g = 0$ only returns a non-zero value when all agents have reached their targets $r_g = 1$. Thus, every agent i receives a reward

$$r_i(t) = \alpha\, r_l(t) + \beta\, r_g(t) + r_i(t) \qquad \in [-\alpha - 2, \beta], \tag{1}$$

where α and β are factors for tuning collaborative behavior. This reward creates an objective of finishing the episode as quickly as possible in a collaborative way. At the end of each episode the return of each agent is calculated as

$$g_i = \sum_{t=1}^{T} r_i(t), \tag{2}$$

where T is the time step when the episode terminated. Submissions by participants are measured according to the metrics in section 1.5.

1.3.7 Environment interface

Participants can interact with the environment using a familiar gym-flavoured API inspired by numerous other Multi Agent RL environments. Figure 8 shows the basic usage example. The environment will be made available on PyPi, and also at :

https://gitlab.aicrowd.com/flatland/flatland-rl

1.4 Tasks and application scenarios

The competition is set up to gradually introduce the vehicle re-scheduling problem to the participants. Through the different tasks, the participants will become familiar with the "Flatland" environment and the specific difficulties of the re-scheduling problem. Each task addresses important questions in order to arrive at an automatic traffic management system in the future. **It is important to note that task 1, is a toy task, which can be deterministically solved using numerous grid search approaches. Task 2 will be the major focus of this competition. Task 3 and Task 4, are more**

```
# Create a new Flatland environment
env = Flatland(...)

# Get all agent handles
handle = env.get_agent_handles()

# Create a agent (e.g. Deep Q-Learning Agent)
agent = Agent(...)

done = False
while not done:
    # The Flatland environment calculates observations in parallel
    for i in range(env.number_of_agents):
        observation = env.get_observation(handle[i])
        agent.infer_action(observation, handle[i])

    # The action is determined one after the other
    for i in range(env.number_of_agents):
        action = agent.get_action(handle[i])
        env.register_action(action,handle[i])

    # Perform one simulation step
    done = env.step()

    # Calculate the reward and do one training step
    rewards = []
    for i in range(env.number_of_agents):
        _reward = env.get_reward(handle[i])
        rewards.append(_reward)
    env.render()
```

Figure 8: Example API for interacting with the Flatland Multi Agent environment.

difficult problem formulations which we eventually want to solve, but are only included here in this proposal (and will be made available to the participants) to provide a broad view of the future directions.

1. **Single agent navigation on railway grid:** The first (toy) task is used as an entry level to the "Flatland" environment. In this setting, an agent simply needs to find the shortest path from a randomly chosen starting position to a randomly chosen target position. Movement of the agent is restricted to the railway network. The objective for the participants is to find the shortest path given limited computational resources and time. This task illustrates that shortest path calculations can be done efficiently.

2. **Multi-agent navigation on railway grid:** As an extension of the simple navigation task, many more agents are added to the railway network. This is the main task to be solved by the participants. Final ranking of the competition will be based on the score obtained on this task. In this task multiple agents must collaborate to avoid deadlocks (when two trains face each other and don't have any alternative routes to avoid each other) and navigate to all targets efficiently. This task is significantly harder due to the interactions between the agents. The objective here is to show that a feasible solution, meaning all agents arrive at their target, can be found for any given starting configuration. The constraints on computational resources and time

are relaxed for this task.

Due to the complexity of the multi-agent navigation task we will not include the following additional tasks in this challenge. They are two additional tasks that highlight the flexibility of Flatland as well as close the gap between the abstract Flatland environment and the fully dynamic railway simulation:

- **Multi-agent traffic with schedule on railway grid:** Adding a schedule for all agents to follow adds complexity in form of diversity among the agents. Beyond avoiding deadlocks, agents have to collaboratively agree on priorities amongst themselves in order to stay as close as possible to the dictated schedule. The objective for the participants is to prove that a multi-agent system can collaboratively solve an asymmetric problem with a global reward.
- **Multi-agent mixed railway traffic with schedule:** Minor change to the previous task by adding different types of agents with different properties such as velocity. This implementation of "Flatland" comes very close to real-world railway traffic. Any feasible solution found for this task will illustrate the feasibility of an automatic traffic management system for the real railway network.

1.5 Metrics

Submitted agents by the participants will be evaluated by a fixed set of environment configurations for each task. The participants can submit a different agent for each task. The metrics used to evaluate the performance are as follows:

1. **Single agent navigation on railway grid:** The score (S_0) is the accumulated total return (section 1.3.6) over $N = 100$ predefined, secret railway network configurations and over all agents (M):

$$S_0 = \sum_{i=0}^{N} \sum_{j=0}^{M} g_j^i. \tag{3}$$

   A higher score is better. This score will not be part of the ranking as this task is only a toy example for participants to get used to the restricted transitions between cells.

2. **Multi-agent navigation on railway grid:** The score (S_1) is the accumulated total return (section 1.3.6) over $N = 100$ predefined, secret railway network configurations and over all agents (M)::

$$S_1 = \sum_{i=0}^{N} \sum_{j=0}^{M} g_j^i. \tag{4}$$

   The challenge ranking will be based on this score (S_1 in equation 4), where a higher score is better.

1.6 Baselines, code, and material provided

Baselines will be provided for all initial observation spaces explained in section 1.3.5. They are based on previous work done by the SBB research lab on an earlier version of the Flatland environment. We will adapt them to the improved environment. Baselines include executable code with documentation explaining the code as well as a documentation about the corresponding observation space. This documentation contains learnings from our observation space investigations and provides insights for participants about different aspects of the vehicle re-scheduling problem. Some of the new baselines will be provided by PhD students from a collaborating Swiss applied university (ZHAW), also in form of executable code and an accompanying paper or explanatory blogpost.
The baselines and documentation will be bundled into a github starter kit as with our previous SBB challenge [13].
The environment will be packaged and easily installable through pip.
Examples of policy learnt by multiagent-DQN with local tree search as the observation space :

- https://s3.eu-central-1.amazonaws.com/aicrowd-static/mohanty/trained_policy.mp4
- https://s3.eu-central-1.amazonaws.com/aicrowd-static/mohanty/trained_policy_1.mp4
- https://s3.eu-central-1.amazonaws.com/aicrowd-static/mohanty/10_agent_side_by_side.mp4

1.7 Tutorial and documentation

A starter kit introducing the problem, with the basic getting started instructions and along with the above mentioned baselines will be available for the participants at:
https://gitlab.aicrowd.com/flatland/flatland-starter-kit

The code for the environment will be available for the participants at:
https://gitlab.aicrowd.com/flatland/flatland-rl

Participants can then easily make their first submission by following the instructions at:
https://gitlab.aicrowd.com/flatland/flatland-rl-sample-submission

2 Organizational aspects

2.1 Protocol

2.1.1 Platform

The evaluation of the submissions will be managed by AIcrowd.com. AIcrowd is built by the same team that created crowdAI, the (now retired) challenge platform developed

at EPFL. The platform offers state-of-the art technology and management specifically for running challenges like the one proposed here. The technology underlying AIcrowd has been tried and tested in over 50 challenges, including some at NeurIPS (the Learning to Run Challenge in 2017, the AI for Prosthetics Challenge in 2018, and the Adversarial Vision Challenge in 2018). The AIcrowd platform offers to the full range of services for a challenge, from user management to discussion forums and leaderboards. The evaluator infrastructure can be flexibly adapted to any execution requirements, and allows for full traceability and reproducibility of the submissions.

2.1.2 Submission Protocol

Throughout the competition, participants can work on their code bases as private git repositories on `https://gitlab.aicrowd.com`. The requirements of the AIcrowd evaluators require participants to package their intended software runtime in their repositories, to ensure that the evaluators can automatically build relevant Docker images from their repositories, and orchestrate them as needed by the evaluators of the particular round. This approach also ensures that all the user submitted code that is successfully evaluated in context of this competition is both versioned across time, and also completely reproducible.

Software Runtime Packaging. Packaging and specification of the software runtime is among the most time consuming (and frustrating) tasks for many participants. And hence, to make this step easier, we will be supporting numerous approaches of easily packaging the software runtime by the help of aicrowd-repo2docker (https://pypi.org/project/aicrowd-repo2docker/). A tool which lets participants specify their runtime using Anaconda environment exports, requirements.txt , or even with the fall back of a traditional Dockerfile. This significantly decreases the barrier to entry for the less technically inclined participants by transforming an irritating debug cycle to a deterministic one-liner which does all the work behind the scenes.

Submission Mechanism. Participants can collectively collaborate together on their private git repository throughout the competition, and whenever they wish to make a submission, they have to create and push a *git tag*, which triggers the evaluation pipeline : image building, orchestration of the containers with the required evaluation context. On the successful evaluation of the submission, the scores and any necessary artifacts

Orchestration of the Submissions. The ability to reliably orchestrate user submissions over large periods of time will be a key determining feature of the success of the proposed competition. We will be using the evaluators of AIcrowd.com which use custom Kubernetes clusters to orchestrate the submissions against pre-agreed resource usage constraints. The same setup has been successfully used previously in numerous other

complicated machine learning competitions in the past. The evaluation setup allows for evaluations of arbitrarily long time-periods, and also can privately provide feedback about the current state of the evaluation to the respective participants.
The challenge will run on the challenge platform AIcrowd (www.aicrowd.com).

2.2 Rules

- Participants are allowed up to 5 submissions a day
- Participant submitted code cannot access the external network during evaluation
- Participants will have to open source their submitted code to be eligible for prizes

2.3 Schedule and readiness

The competition will start in June 2019 and run till mid October to allow for enough time before NeurIPS 2019 to review and publish the results.
At the time of this proposal submission we have an early version of Flatland running and the development of the improved environment started a few weeks ago. Thorough investigation about the performance of different observation spaces have been made on the earlier Flatland environment as shown in the linked videos of this submission.
An international team of reinforcement learning experts are involved in the open source development of the Flatland environment and both the EPFL and ZHAW are supporting baseline investigations with PhD students.

2.4 Competition promotion

The competition will be promoted in the social media channels of SBB and AIcrowd (with a reach to numerous RL researchers from the many RL competitions organized in the past). Competition will also be circulated among researchers at EPFL and ETH-Zurich.

3 Resources

3.1 Organizing team

Sharada Prasanna Mohanty is the CEO & Co-founder of AIcrowd, an opensource platform for encouraging Reproducible research in Artificial Intelligence. He has been the co-organizer of numerous large scale machine learning competitions like NIPS 2017: Learning to Run Challenge, NIPS 2018: AI for Prosthetics Challenge, NIPS 2018: Adversarial Vision Challenge, MarLO Challenge 2018. While doing his PhD at EPFL, he was working on numerous problems at the intersection of AI and Health, with a strong interest in Reinforcement Learning. In his current role, he focusses on building better engineering tools for AI researchers, and on making research in AI accessible to a much larger community of engineers. He will be responsible for advise on general challenge design, and for integration with AIcrowd evaluators.

- Design & implement submission and evaluation backend
- Advise on challenge design
- Provide Baselines
- Platform Administration

Erik Nygren is an AI researcher at the Swiss federal railway company (SBB) research group. His research focus is on applied machine learning, in particular the use of deep reinforcement learning to tackle the vehicle re-scheduling problem. He was involved in a previous crowdsourcing Challenge by SBB and AIcrowd. He will help designing and implementing the challenge as well as provide baselines.

- Design & implementation challenge
- Provide baselines
- Administration

Adrian Egli is a high performance computing expert at the Swiss federal railway company (SBB) research group. His focus is the design and implementation of high performance simulations for real world challenges. He will design and optimize the *Flatland* to guarantee good reinforcement learning compatibility and high computational performance.

- Design & implementation of challenge
- Optimization of *Flatland* performance

Marcel Salathé is a professor at the Swiss Institue of Technology Lausanne (EPFL) where he heads the Digital Edpidemiology Lab. His group created the challenge platform crowdAI and co-organized multiple NeurIPS challenges before (Learning to Run Challenge in 2017, AI for prosthetics Challenge in 2018, Adverserial Vision Challenge 2018). He is a co-founder of AIcrowd, which has spun out off crowdAI. He will advise the team and help with the promotion in the academic community.

- Advise on design & implementation
- Promotion
- Administration

3.2 Resources provided by organizers, including prizes

Just like our previous SBB challenge [14] with crowdAI we will provide monetary prices to the top three ranks as well as at least 4 travel grants to visit a relevant conference. We will also try to find sponsoring for machine learning hardware (e.g. Nvidia graphics cards) for the remaining top ten contributions.
Participants solutions will be evaluated using docker images on cloudservices (e.g. AWS) provided by SBB.

3.3 Support and facilities requested

We would request full conference tickets for the winners and top participants of the competitions.

References

[1] Adrian Egli and Erik Nygren. *Real World Application of Multi-Agent Deep Reinforcement Learning: Autonomous traffic flow management*. Oct. 2018. DOI: 10.13140/RG.2.2.13164.62085.

[2] Brian A Foster and David M Ryan. "An integer programming approach to the vehicle scheduling problem". In: *Journal of the Operational Research Society* 27.2 (1976), pp. 367–384.

[3] Albert Jones, Luis Rabelo, and Abeer T. Sharawi. "Survey of Job Shop Scheduling Techniques". In: Dec. 1999. ISBN: 9780471346081. DOI: 10.1002/047134608X.W3352.

[4] Eric Larsen et al. "Predicting Tactical Solutions to Operational Planning Problems under Imperfect Information". In: *arXiv preprint arXiv:1901.07935* (2019).

[5] Jing-Quan Li, Pitu B Mirchandani, and Denis Borenstein. "The vehicle rescheduling problem: Model and algorithms". In: *Networks: An International Journal* 50.3 (2007), pp. 211–229.

[6] Ryan Lowe et al. *Multi-Agent Actor-Critic for Mixed Cooperative-Competitive Environments*. 2017. eprint: arXiv:1706.02275.

[7] Diego Perez-Liebana et al. "The Multi-Agent Reinforcement Learning in Malmfffdfffd (MARLfffdfffd) Competition". In: (2019). eprint: arXiv:1901.08129.

[8] Jean-Yves Potvin and Jean-Marc Rousseau. "A parallel route building algorithm for the vehicle routing and scheduling problem with time windows". In: *European Journal of Operational Research* 66.3 (1993), pp. 331–340.

[9] Cinjon Resnick et al. *Pommerman: A Multi-Agent Playground*. 2018. eprint: arXiv:1809.07124.

[10] Darja Šemrov et al. "Reinforcement learning approach for train rescheduling on a single-track railway". In: *Transportation Research Part B: Methodological* 86 (2016), pp. 250–267.

[11] Jamal Shahrabi, Mohammad Amin Adibi, and Masoud Mahootchi. "A reinforcement learning approach to parameter estimation in dynamic job shop scheduling". In: *Computers & Industrial Engineering* 110 (2017), pp. 75–82.

[12] *smartrail 4.0: An innovation programme from the Swiss railway industry*. https://smartrail40.ch/index.asp?inc=&lang=en.

[13] *Starter Kit*. https://github.com/crowdAI/train-schedule-optimisation-challenge-starter-kit/#starter-kit-repo-for-the-sbb-train-schedule-optimisation-challenge-on-crowdai.

[14] *Train Schedule Optimisation Challenge*. https://www.crowdai.org/challenges/train-schedule-optimisation-challenge.

[15] Oriol Vinyals et al. *StarCraft II: A New Challenge for Reinforcement Learning*. 2017. eprint: arXiv:1708.04782.

[16] Lianmin Zheng et al. *MAgent: A Many-Agent Reinforcement Learning Platform for Artificial Collective Intelligence*. 2017. eprint: arXiv:1712.00600.

[17] Li Zipeng and Wang Yanyang. "A Review for Aircraft Landing Problem". In: *MATEC Web of Conferences* 179 (Jan. 2018), p. 03016. DOI: 10.1051/matecconf/201817903016.

A Appendix : Multi-agent movement

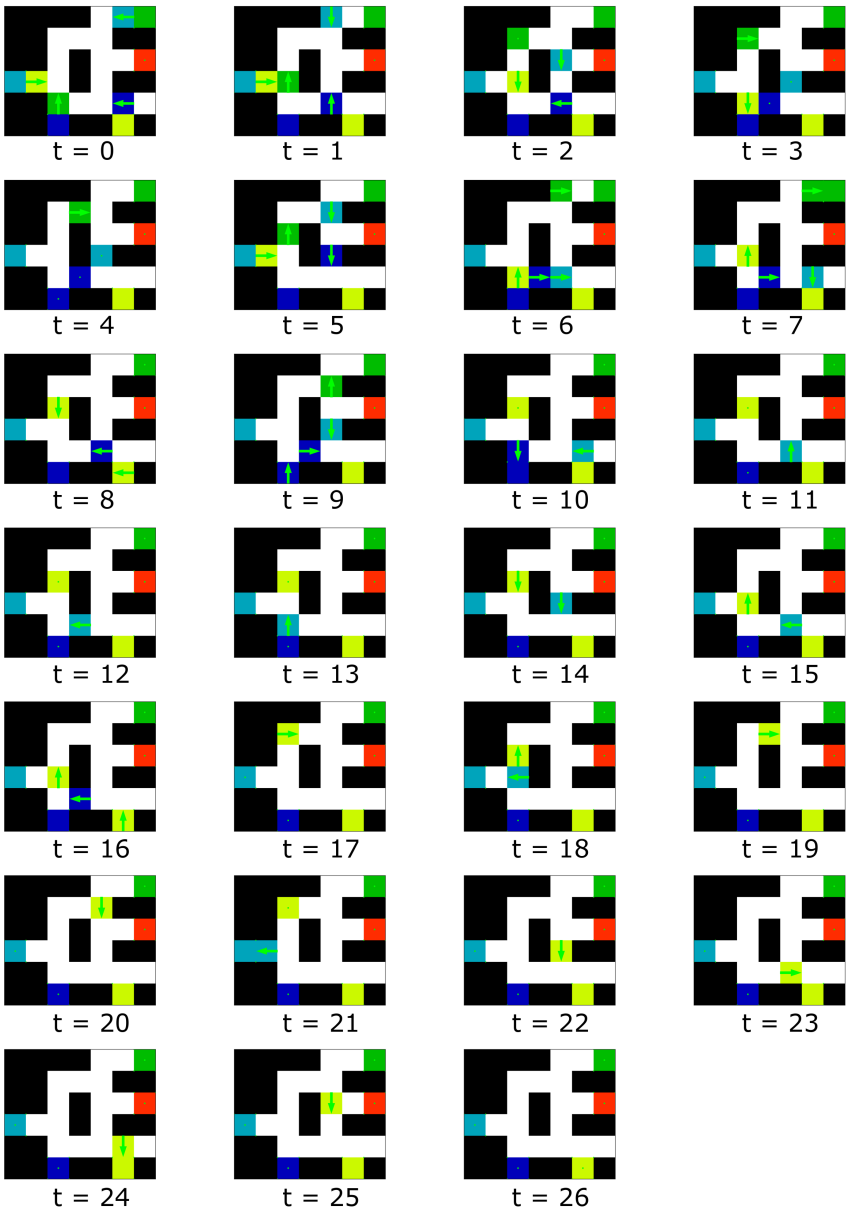


Figure 9: *This figure illustrates a typical situation that can occur between two or more agents. A routing decision needs to be made in order to avoid deadlocks between multiple agents. The complexity grows vastly when considering the effects of each such decision on the full network traffic. Thus problems of this nature cannot be solved using only local information.*