Reinforcement Learning for Autonomous Agents Exploring Environments: an Experimental Framework and Preliminary Results

Nassim Habbash, Federico Bottoni and Giuseppe Vizzari

Department of Informatics, Systems and Communication (DISCo), University of Milano-Bicocca, Milan, Italy

Abstract
Reinforcement Learning (RL) is being growingly investigated as an approach to achieve autonomous agents, where the term autonomous has a stronger acceptation than the current most widespread one. On a more pragmatic level, recent developments and results in the RL area suggest that this approach might even be a promising alternative to current agent-based approaches to the modeling of complex systems. This work presents an investigation of the level of readiness of a state-of-the-art model to tackle issues of orientation and exploration of a randomly generated environment, as a toy problem to evaluate the adequacy of the RL approach to provide support to modelers in the area of complex systems simulation, and in particular pedestrian and crowd simulation. The paper presents the adopted approach, the achieved results, and discusses future developments on this line of work.

Keywords
agent-based modeling and simulation, reinforcement learning, complex-systems

1. Introduction

Reinforcement Learning (RL) [1] is being growingly investigated as an approach to implement autonomous agents, where the acceptation of the term "autonomous" is closer to Russell and Norvig's [2] than the most widely adopted ones in agent computing. Russell and Norvig state that:

> *A system is autonomous to the extent that its behavior is determined by its own experience*

A certain amount of initial knowledge (in an analogy to built-in reflexes in animals and humans) is reasonable, but it should be sided by the ability to learn. RL approaches, reinvigorated by the energy, efforts, and promises brought by the *deep learning* revolution, seems one of the most promising ways to investigate how to provide an agent this kind of autonomy. On a more pragmatic level, recent developments and results in the RL area suggest that this approach might even be a promising alternative to current agent-based approaches to the modeling of complex

WOA 2020: Workshop "From Objects to Agents", September 14–16, 2020, Bologna, Italy
n.habbash@campus.unimib.it (N. Habbash); f.bottoni1@campus.unimib.it (F. Bottoni); giuseppe.vizzari@unimib.it (G. Vizzari)
0000-0002-7916-6438 (G. Vizzari)

systems [3]: whereas currently behavioral models for agents are carefully hand crafted, often following a complicated interdisciplinary effort involving different roles and types of knowledge, as well as validation processes based on the acquisition and analysis of data describing the studied phenomenon, RL could simplify this work, focusing on the definition of an environment representation, the definition of a model for agent perception and action, and defining a reward function. The learning process could, in theory, be able to explore the potential space of the *policies* (i.e. agent behavioral specifications) and converge to the desired decision making model. While the definition of a model of the environment, as well as agent perception and action, and the definition of a reward function are tasks requiring substantial knowledge about the studied domain and phenomenon, the learning process could significantly simplify modeler's work, and at the same time it could solve issues related to model calibration.

The present work is set in this scenario: in particular, we want here to explore the level of readiness of state-of-the-art models to tackle issues of orientation and exploration of an environment [4] by an agent that does not own prior knowledge about its topology. The environment is characterised by the presence of obstacles, generated randomly, and by a target for agent's movement, a goal that must be reached while, at the same time, avoiding obstacles. This represents a toy problem allowing us to investigate the adequacy of the RL approach to support modelers in the area of complex systems simulation, and in particular pedestrian and crowd simulation [5]. We adopted Proximal Policy Optimization (PPO) [6] and trained agents in the above introduced type of environment: the achieved decision making model was evaluated in new environments analogous to the ones employed for training, but we also evaluated the adequacy of the final model to guide agents in different types of environment, less random and more similar to human built environments (i.e. including rooms, corridors, passages) to start evaluating if agents for simulating typical pedestrians could be trained through a RL approach.

2. Problem Statement

The main focus of this paper is automatic exploration of environments without a-priori knowledge of their topology. This is modeled through a single-agent system, where an agent is encouraged to look out for a target placed randomly in a semi-randomly generated environment. This environment presents an arbitrary number of obstacles placed randomly on its space. The environment can be seen as a rough approximation of natural, mountainous terrain, or artificial, post-disaster terrain, such as a wrecked room. The agent can observe the environment through its front-mounted sensors and move on the XY Cartesian plane. In order to solve the problem of automatic exploration in an ever changing obstacle-ridden environment, the main task is to generalize the exploration procedure, to achieve an agent able to explore different environments from the ones it was trained on.

In this paper we develop a framework around this task using Reinforcement Learning. Section 3 provides a definition of the agent, the environment and their interactions. Section 4 goes through Reinforcement Learning and the specific technique adopted for this work. Section 5 provides an architecture to the system, with some details on the tools used. Section 6 reports the experimental results obtained, and Section 7 provides some considerations on the work and possible future developments.

3. Agent and Environment

3.1. Agent model

The agent is modeled after a simplified rover robot with omnidirectional wheels, capable of moving on the ground in all directions. The location of the agent in the environment is described by the triple (x, y, ϑ), where (x, y) denotes its position on the XY plane, and ϑ denotes its orientation. The agent size is 1x1x1 units.

3.2. Observation space

The agent can observe the environment through a set of LIDARs that create a array of surveying rays: these are time-of-flight sensors which provide information on both the distance between the agent and the collided object and the object's type. If a ray is not long enough to reach an object because it is too far away, the data provides the over-maximum-range information to the agent. The standard LIDAR length is 20 units. The agent is equipped with 14 LIDARs, placed radially on a plane starting from the middle of its front-facing side, giving it a field of view of [-$\frac{2}{3}\pi$; $\frac{2}{3}\pi$] for 20 units of range.

More formally, we can define an observation or state as a set of tuples, as follows:

$$s = \{(x_1, o_1), (x_2, o_2), ..., (x_n, o_n)\}, s \in S \tag{1}$$

Where n is the number of LIDARs on the agent, x_i represents the distance from a certain LIDAR to a colliding object in range, and o_i is the type of said object, with $o_i \in \{obstacle, target, \emptyset\}$.

The observations (or state) space is hence defined as S, the set of all possible states s.

3.3. Action space

The possible actions that the agent can perform are:

- Move forward or backward
- Move to the left or to the right, stepping aside (literally), without changing orientation
- Rotate counterclockwise or clockwise (yaw)

The agent can also combine the actions, for example going forward-right while rotating counterclockwise.

More formally, we can define the action space as:

$$A = \{Forward, Side, Rotation\} \tag{2}$$

Where $Forward$, $Side$ and $Rotation$ represent the movement on the associated axes, and their value can be either $\{-1, 0, 1\}$, where 0 represents no movement, and -1 and 1 represent movement towards one or the other reference point on the axis.

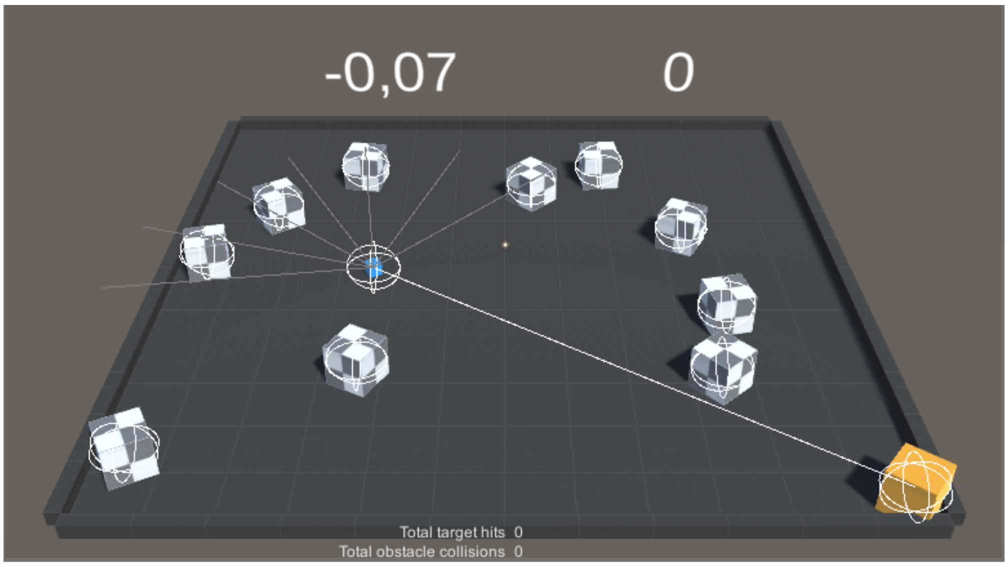

Figure 1: Environment: the blue cube is the agent, the checkerboard pattern cubes are the obstacles and the orange cube is the target. The environment is generated considering the distance between agent and target (white line connecting both) and the minimum spawn distance (the white sphere around objects) to ensure a parametric distribution between objects in the environment. The agent collects observations through its LIDAR array (white rays exiting the agent). On the top are present the episode's cumulative reward on the left and the current number of collisions on the right.

3.4. Environment model

The environment is a flat area of 50x50 units, bounded on its extremities by walls tall 1 unit. A set of gray cubes of 3x3x3 units each are randomly placed on this area as obstacles. The target - the goal the agent must reach - is positioned randomly between this set of obstacles, and is an orange cube of 3x3x3 units.

The provided interactions between agent and environment are collisions. The agent collides with another object in the environment if there's an intersection between the bounding boxes of the two entities. The floor is excluded from collisions. If a collision happens between agent and obstacles, the agent suffers a penalty, while if a collision happens between agent and target, the episode ends successfully, as the agent achieved its goal.

The environment is generated every time the episodes ends, successfully or not, so no two identical episodes are played by the agent. This generation is parametric, allowing for more or less dense obstacle distribution in the environment, or longer or shorter distance between agent and the target.

4. Reinforcement Learning

In the past couple of years Reinforcement Learning has seen many successful and remarkable applications in the robot and locomotive field, such as [7] and [8]. This approach provides many benefits: experimentation can be done in a safe, simulated environment, and it's possible to train models through millions of iterations of experience to learn an optimal behaviour. In some fields - such as robot movement - the RL approach currently outperforms classic heuristic and evolutionary methods [1].

Reinforcement Learning is a technique where an agent learns by interacting with the environment. The agent ought to take actions that maximize a reward, selecting these from its past experiences (Exploitation) and completely new choices (Exploration), making this essentially a trial-and-error learning strategy. After sufficient training, an agent can generalize an optimal strategy, allowing itself to actively adapt to the environment and maximize future rewards. Generally, an RL algorithm is composed of the following components:

1. A policy function, which is a mapping between the state space and the action space of the agent
2. A reward signal, which defines the goal of the problem, and is sent by the environment to the agent at each time-step
3. A value function, which defines the expected future reward the agent can gain from the current and all subsequent future states
4. A model, which defines the behaviour of the environment

At any time, an agent is in a given states of the overall environment $s \in S$ (that it should be able to perceive; from now on, we can consider the state the portion of the environment that is perceivable by the agent), and it can choose to take one of many actions $a \in A$, to cause a change of state to another one with a given probability P. Taken an action a chosen by an agent, the environment returns a reward signal $r \in R$ as a feedback on the goodness of the action. The behaviour of the agent is regulated by what's called a policy function π, which can be defined as

$$\pi_\Theta(a|s) = P(A_t = a|S_t = s) \tag{3}$$

and represents a distribution over actions given states at time t with parameters Θ - in this case the policy function is stochastic, as it maps over probabilities. Following is a brief introduction to the two main algorithms used in this work.

4.1. Proximal Policy Optimization

RL presents a plethora of different approaches. Proximal Policy Optimization (PPO) [6], the one used in this work, is a policy gradient algorithm which works by learning the policy function π directly. These methods have a better convergence properties compared to dynamic programming methods, but need a more abundant set of training samples. Policy gradients work by learning the policy's parameters through a policy score function, $J(\Theta)$, through which is then possible to apply gradient ascent to maximize the score of the policy with respect to the policy's parameters, Θ. A common way to define the policy score function is through a loss function:

$$L^{PG}(\Theta) = E_t[log\pi_\Theta(a_t|s_t)]A_t \tag{4}$$

which is the expected value of the log probability of taking action a_t at state s_t times the advantage function A_t, representing an estimate of the relative value of the taken action. As such, when the advantage estimate is positive, the gradient will be positive as well; through gradient ascent the probability of taking the correct action will increase, while decreasing the probabilities of the actions associated to negative advantage, in the other case. The main issue with this vanilla policy gradient approach is that gradient ascent might eventually lead out of the range of states where the current experience data of the agent has been collected, changing completely the policy. One way to solve this issue is to update the policy conservatively, so as to not move too far in one single update. This is the solution applied by the Trust Region Policy Optimization algorithm [9], which forms the basis of PPO. PPO implements this update constraint in its objective function through what it calls Clipped Surrogate Objective. First, it defines a probability ratio between new and old policy $r_t(\Theta)$, which tells if an action for a state is more likely or less likely to happen after the policy update, and it is defined as $r_t(\Theta) = \frac{\pi_\Theta(a_t|s_t)}{\pi_{\Theta_{old}}(a_t|s_t)}$. PPO's loss function, is then defined as:

$$L^{CLIP}(\Theta) = E_t[min(r_t(\Theta)A_t, clip(r_t(\Theta), 1-\epsilon, 1+\epsilon)A_t)] \tag{5}$$

The Clipped Surrogate Objective presents two probability ratios, one non clipped, which is the default objective as expressed in 4 in terms of policy ratio, and one clipped in a range. The function presents two cases depending on whether the advantage function is positive or negative:

1. $A > 0$: the action taken had a better than expected effect, therefore, the new policy is encouraged to taking this action in that state;
2. $A < 0$: the action had a negative effect on the outcome, therefore, the new policy is discouraged to taking this action in that state.

In both cases, because of the clipping, the actions will only increase or decrease in probability of $1 \pm \epsilon$, preventing updating the policy too much, while allowing the gradient updates to undo bad updates (e.g. the action was good but it was accidentally made less probable) by choosing the non-clipped objective when it is lower than the clipped one. Note that the final loss function for PPO adds two other terms to be optimized at the same time, but we suggest the original paper [6] for a more complete overview of PPO.

4.2. Curiosity

Reward sparseness is one of the main issues with RL. If an environment is defined with a sparse reward function, the agent won't get any feedback about whether its actions at the current time step are good or bad, but only at the end of the episode, where it either managed to succeed in the task or fail. This means that the reward signal is 0 most of the time, and is positive in only few states and actions. One simple example is the game of chess: the reward could be obtained only at the end of the match, but at the beginning, when the reward might be 10, 50, 100 time steps away, if the agent can't receive feedback for its current actions it can only move randomly until, by sheer luck, it manages to get a positive reward; long range dependencies must then be learned, leading to a complicated and potentially overly long learning process. There are many ways to solve the problem of reward sparseness, such as reward shaping, which requires domain-specific knowledge on the problem, or intrinsic reward signals, additional reward signals to mitigate the sparseness of extrinsic reward signals.

Curiosity [10] falls into the second category, and its goal is to make the agent actively seek out and explore states of the environment that it would not explore. This is done by supplying the default reward signal with an additional intrinsic component which is computed by a curiosity module. This module is comprised of a forward model, which takes in s_t and a_t, and tries to predict the features of next state the agent will find itself in $\hat{\Phi}(s_{t+1})$. The more different this value compared to the features of the real next state $\Phi(s_{t+1})$, the higher the intrinsic reward.

To avoid getting stuck into unexpected states that are produced by random processes non influenced by the agent, the module is comprised of an inverse model, which takes $\Phi(s_t)$, $\Phi(s_{t+1})$ and tries to predict the action $\hat{a_t}$ that was taken to get from s_t to s_{t+1}. By training the encoder (Φ) together with the inverse model, it s possible to make it so that the feature extracted ignore those states and events that are impossible to influence, retaining only features actually influenced by the agent's actions.

5. System Architecture

The system has been developed using Unity3D as the main simulation platform and ML-Agents for Reinforcement Learning[1].

Unity3D is a well-established open-source game engine. It provides a lot of out-of-the-box functionalities including tools to assemble a scene, the 3D engine to render it, a physics engine to physically simulate object interaction under physical laws, and many plugins and utilities. In Unity an object is defined as a GameObject, and it can have attached different components according to necessity, such as RigidBodies for physical computations, Controllers for movement, decision-making and elements of the learning system (Agent, Academy and others). An object's life-cycle starts with an initialization and then a cyclic refresh of its state, while the engine provides handler methods for these phases for customizing them through an event-driven programming.

Unity keeps track of time and events on a stratified pipeline: physics, game logic and scene rendering logic are each computed sequentially and asynchronously:

1. Objects initialization.
2. Physics cycle (triggers, collisions, etc). May happen more than once per frame if the fixed time-step is less than the actual frame update time.
3. Input events
4. Game logic, co-routines
5. Scene rendering
6. Decommissioning (objects destruction)

One notable caveat is that physics updates may happen at a different rate than game logic. This, in a game development scenario is sometimes treated by buffering either inputs or events, to result in smoother physical handling, while for more simulations in which a precise correspondence between simulated and simulation time is necessary this might pose a slight inconvenience.

[1]The project's source code is available on Github: https://github.com/nhabbash/autonomous-exploration-agent.

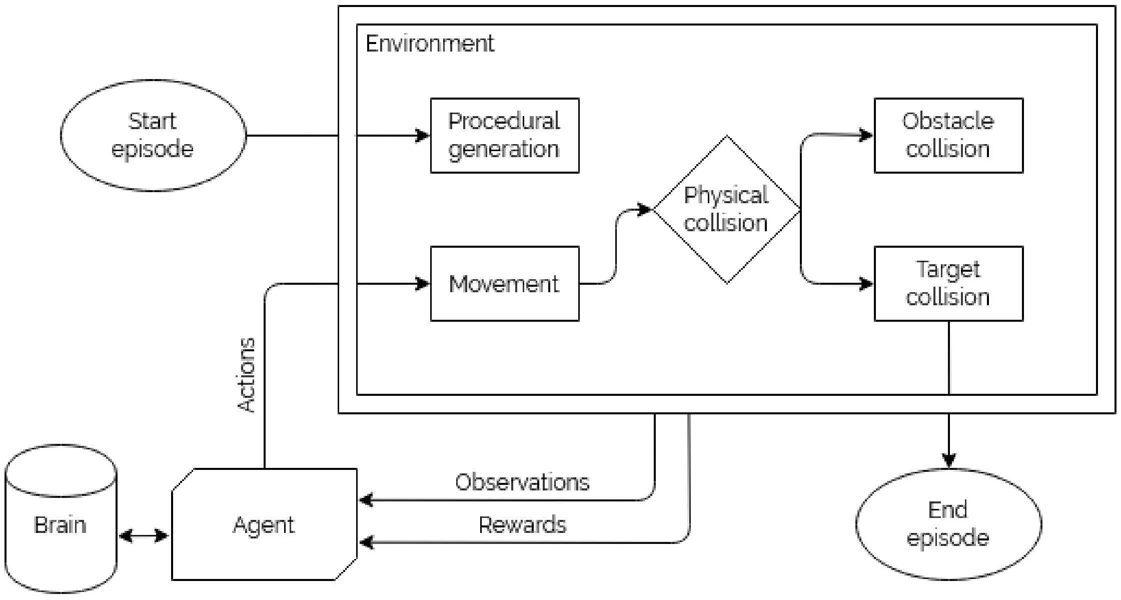

Figure 2: Schematized system architecture, shows the main interactions between the actors in the system

However, for the goals of the present work, this consideration does not represent a crucial problem.

ML-Agents is an open-source Unity3D plugin that enables games and simulations to serve as environments for training intelligent agents. It provides different implementations of state-of-the-art Reinforcement Learning algorithms implemented in TensorFlow, including PPO and Curiosity. ML-Agents comes with an SDK that can be integrated seamlessly into Unity and provides utilities for controlling the agent(s) and environment.

5.1. System design

Figure 2 represents the main actors of the system and their interactions. At the start of each episode the environment is randomly generated according to its parameters, placing obstacles, the target and the agent. The agent can then perform its actions, moving around the environment, while collecting observations through its sensors and receiving rewards according to the goodness of its actions. Physical collisions trigger negative or positive instantaneous rewards according to the time of collision: obstacle collisions produce negative rewards, while target collisions produce positive rewards and end the episode, as the task is successful. The agent class, ExplorationAgent, is responsible of the agent initialization, observation collection, collision detection and physical movement through the decisions received by the Brain interface. The decision-making of the agent is made through its Brain interface, which provides the actions produced by the model to the agent class. The environment is comprised of two classes, ExplorationArea is responsible for resetting and starting every episode, rendering the UI, logging information on the simulation process and placing every object in the environment

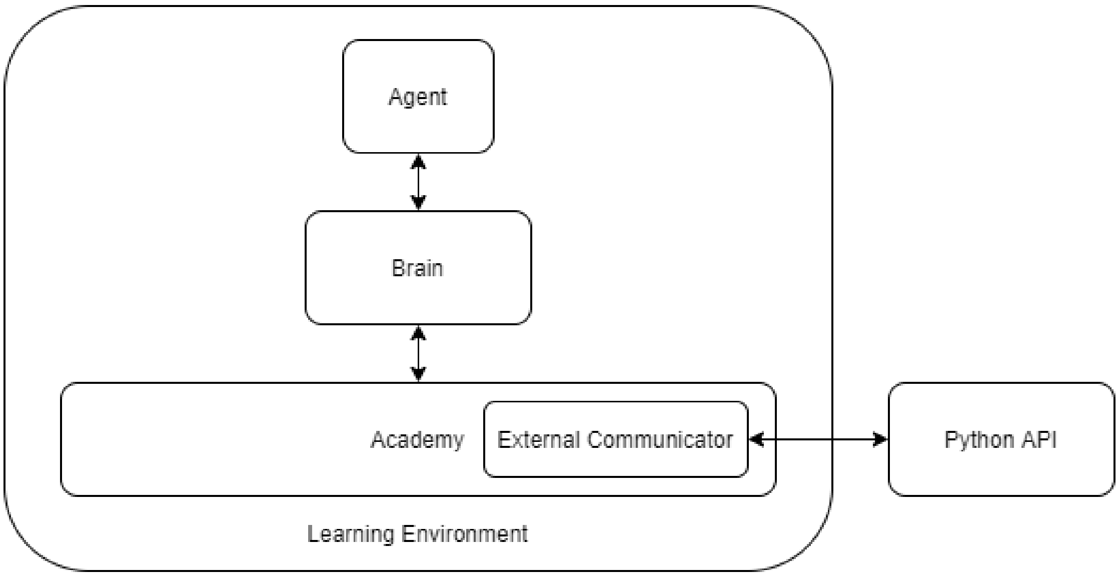

Figure 3: Learning system: the agent receives the actions from the Brain interface, which communicates with the Academy to send and receive observations and actions. The Academy communicates through an external communicator to the ML-Agents back-end, which wraps over Tensorflow and holds the various RL models involved, receiving training/inference data and returning the models outputs

according to its parameters, while the Academy works in tandem with the Brain to regulate the learning process, acting as a hub routing information - whether it's observations from the environment or inferences made by the models - between the system and the RL algorithms under the hood.

Once the training phase ends, ML-Agents generates a model file which can be connected to the brain and used directly for inference. Figure 3 explains how exactly the learning system is structured between Unity and ML-Agents.

5.2. Reward Signal

The main RL algorithm used in this work is PPO. The reward signal is composed of:

- Extrinsic signal: the standard reward produced by the environment according to the goodness of the agent's actions
- Intrinsic signal: the Curiosity of the agent

The extrinsic signal presents some reward shaping, and is defined as: $r = 5 * c_t - p$. The p term stands for penalty, and is a negative reward formulated as $p = c_o * 0.1 + time * 0.001$. Every time the agent reaches a target, indicated by c_t (target collisions), the episode ends, and the positive reward is 5, while if it hits an obstacle, indicated by c_o (obstacle collisions), it receives a penalty of 0.1 for each collision. The agent is also penalized as time passes, receiving a 0.001 negative reward for each timestep, to incentive the agent to solve the task faster.

The intrinsic signal is the Curiosity module, which as section 4.2 goes through, provides an additional reward signal to encourage the exploration of unseen states.

6. Experiments

Different scenarios for the analysis of the effectiveness of the proposed system have been investigated.

The main differences between scenarios consist in:

1. Curriculum environment parameters
2. Penalty function
3. Observation source (LIDAR or camera)
4. Knowledge transferability to structured environments

Comparison between the scenarios is conducted on two aspects: the firsts depends on the canonical RL charts built in training-time in order to define the reward, their trends over times, their balance and other information about the system; the other aspect is an environmental performance comparison, conducted through three performance measures pertaining to the investigated setting, these being CPM (collisions per minute), measuring the mean number of collisions made by the agent on obstacles, TPM (targets per minute), measuring the mean number of goal targets successfully reached by the agent and CPT (collisions per target), measuring the mean number of collisions the agents does before getting to a target. As models between scenarios have been trained with varying curricula and environments, these measures estimate the performance of every model on a shared environment, making comparison between the models happen on the same plane and circumstances. This environmental performances have been measured in a parallel fashion in order to gather more accurate data.

An interactive demo of the system is also available to allow visual comparison of the different scenarios[2].

6.1. Baseline

The first experiment acts as a baseline for other variations, and was conducted on the following parameters:

1. Static parameters:

Number of obstacles	10
Min spawn distance	2
Target distance	45

2. Penalty function: $p = c_o * 0.1 + time * 0.001$
3. Observations source: LIDAR set

[2]Demo at https://nhabbash.github.io/autonomous-exploration-agent/

The parameters in this setting generate fairly open environments with at most 10 obstacles. The minimum distance between obstacles is 2 units, while the target spawns at a distance of 45 units, which is more then half of the environment size.

Table 1 shows how the model collides roughly 5 times per minute and manages to reach a target about 4 and times per minute, making roughly **1.25** obstacles per target reached.

6.2. Curriculum

The second experiment implemented curriculum learning into the training pipeline. The curriculum was structured in seven lessons scaling along with the cumulative reward. Following are its settings:

1. Dynamic parameters:

Reward thresholds	1	2	2.5	2.8	3	3.5	4
Number of obstacles	8	10	13	15	17	18	20
Min spawn distance	6	6	4	4	3	3	2
Target distance	25	28	30	33	35	37	40

2. Penalty function: $p = c_o * 0.1 + time * 0.001$
3. Observations source: LIDAR set

The parameters in this setting generate an increasingly harder environment, with the target getting farther from the agent, and the obstacles getting more cluttered and closer together.

Table 1 shows how the model collides roughly 10.6 times per minute and manages to reach a target about 9.6 and times per minute, making roughly **1.09** obstacles per target reached. This is a significant improvement compared to the baseline, as not only the agent is able to reach the target faster, reaching 2.6 times more targets, but also with less collisions.

6.3. Harder penalty

The third experiment implemented curriculum learning into the training pipeline, and added a harsher penalty to the agent. The curriculum is structured as the above experiment, with the exception of the new parameter Penalty offset. Following are its settings:

1. Curriculum parameters:

Reward thresholds	1	2	2.5	2.8	3	3.5	4
Number of obstacles	8	10	13	15	17	18	20
Min spawn distance	6	6	4	4	3	3	2
Target distance	25	28	30	33	35	37	40
Penalty offset	0.5	1.5	2	2.5	2.5	2.5	2.5

2. Harder penalty function: $p = p_{offset} + c_o + time * 0.001$
3. Observations source: LIDAR set

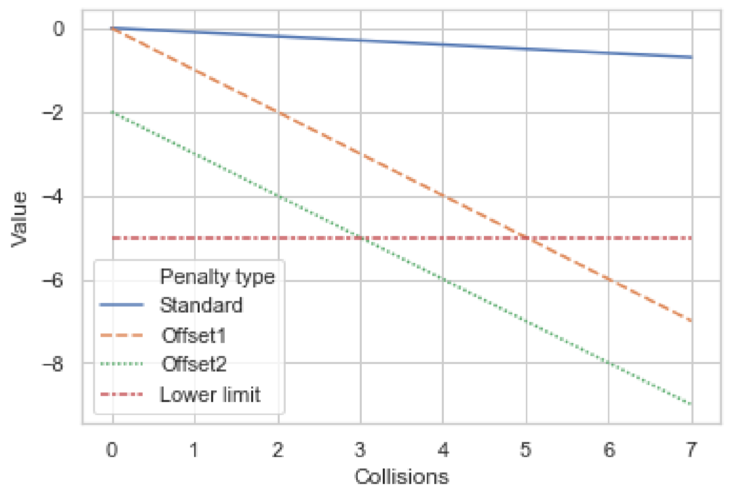

Figure 4: Penalty comparison: standard is the penalty used in most settings in the project, Offset1 and Offset2 are the penalties from the third experiment at different stages of difficulty, and the Cumulative Reward lower limit shows the number of collision before the episode aborts: Offset1 can get out with at most 5 collisions, Offset2 has a maximum of 3

As for the previous situation, the parameters generate a harder environment as the cumulative reward increase, but this time the penalty function too increases in difficulty. The rationale of this experiment is that, as the agent learns how to move to reach the target, the agent should learn to not collide frequently, but instead just search in the environment for the target smoothly. Figure 4 shows how the different penalty relate to each other and the Cumulative Reward lower limit without considering time decrease (same in all penalties).

Table 1 shows how the model collides roughly 4.5 times per minute and manages to reach a target about 3.9 and times per minute, making roughly **1.18** obstacles per target reached. This model obtains results similar to the baseline, while staying below the performances the curriculum model obtained. This may be due to the harsher penalty that does not give the model time to adapt to an optimal policy.

6.4. Camera sensors

The fourth experiment implemented the same curriculum and penalty as the second experiment. The main difference consists in the use of a camera sensor instead of the LIDAR array, thus generating images as observations.

1. Curriculum parameters: same as the second experiment
2. Penalty function: same as the second experiment
3. Observations source: Camera 84x84 RGB

The model performs significantly worse than the others. This is plausibly due to the low number of steps taken by the training of the model (74k) compared to the others (700k), which

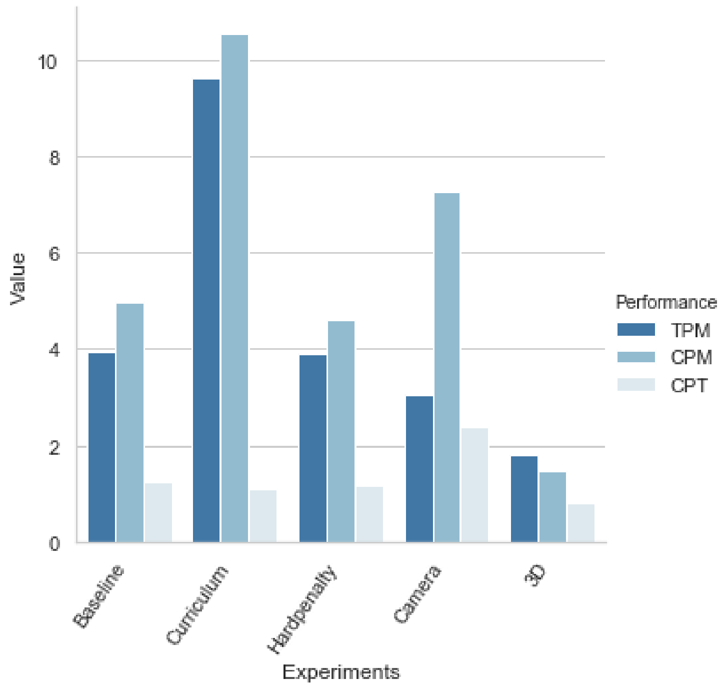

Figure 5: Environmental performances comparisons between the different random experiments

Table 1
Comparison of performance of the different experimented training approaches.

Training approach	CPM	TPM	CPT
Baseline	4,984	3,96	1,25
Curriculum	10,568	9,616	1,09
Hardpenalty	4,592	3,896	1,18
Camera	7,264	3,048	2,38

did not let the algorithm converge to an optimal policy. We must add that the choice not to investigate a longer training phase is due to the fact that the need to analyse this heavier form of input, that nonetheless might not necessarily be more informative, made the training much more expensive in terms of computation time, so whereas the steps are less than in the other experiments, the overall computation time for learning is very similar. The model manages to reach a target with roughly 2.5 obstacles hits each time, but taking roughly 3 times as much as the optimal curriculum model.

6.5. Structured environment transferability

Taking the best performing model (i.e. Curriculum), the experiment consisted in testing how well does the model generalize its task to structured environments - and how well does what

Table 2
Summary of the performance of the curriculum based model (achieved by training in random scenarios) applied to structured environments.

Environment	CPM	TPM	CPT
Rooms	7,2	1,8	4
Corridor	4,8	14,2	0,34
Corridor tight	29,6	2,2	13,45
Turn	10,4	10,4	1
Crossroad	23,4	9,6	2,44

the model learned during training in the chaotic environments transfer to other structured environments, which consist in:

1. "Rooms", two rooms are linked by a tight space in a wall, the agent has to pass through the opening to reach the target;
2. "Corridor" is a long environment - literally a corridor - that the agent has to run across to reach the target;
3. "Corridor tight" is similar to the previous environment but tighter
4. "Turn" is a corridor including a 90° left turn
5. "Crossroad" represents two ways cross and the agent has either to go straight on, to turn left or to turn right.

These environments tested the capability of the agent to follow directions and to look for the target in every space of the scene.

The model does not perform as well as in every environments. This seems to be due the strategy that the model has learned to be optimal for the resolution of the task at hand, seems to be random exploration, and will be addressed later on. The agent seems able to follow a linear path, if the environment is wide enough to allow it to stay away from the borders, it is apparent comparing Corridor and Corridor Tight outcomes: the second experiment has less targets and more collisions per minute then the first one. The model's performances in Crossroad and Turn are similar but, in the first case the agent has to change path more often then in the second one, so collisions happen more frequently. Rooms experiment has low both TPM and CPM because the agent tends to stay in the spawn room, without attempting to pass from the door.

6.6. Performance measures comparison

The models have comparable performances: the curriculum, hardpenalty and camera models reach a similar cumulative reward - with the camera ending on top, followed by the curriculum and then hardpenalty models. This comparison shows the first caveat of the experiments: cumulative reward notwithstanding, the curriculum model achieved way higher environmental performances than the other two models. The comparison makes even more sense if compared to the baseline model, which almost perfectly converged on a higher cumulative reward, but even then, the curriculum model achieved better performances, even with a lower cumulative reward.

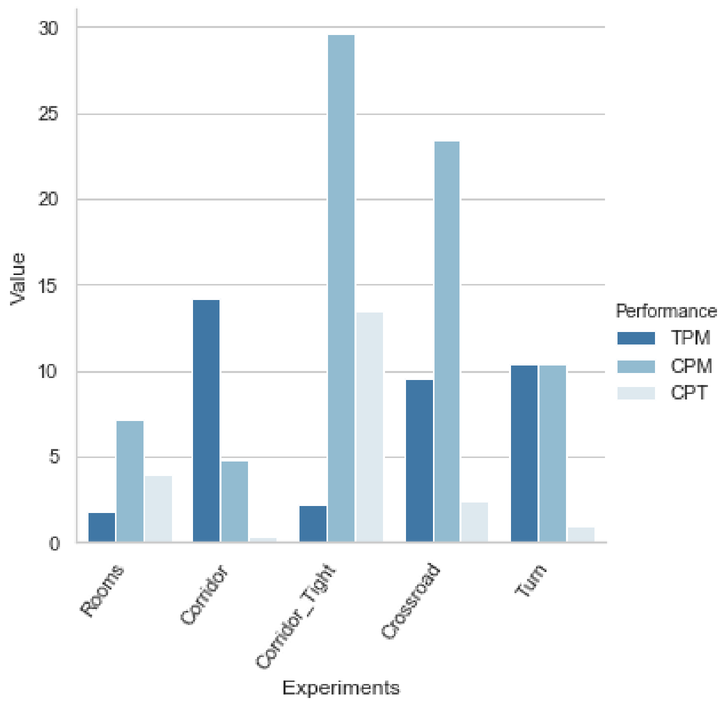

Figure 6: Environmental performances comparisons between the different structured experiments

We also performed experiments in a 3D version of the environment, whose complete description is omitted for sake of space, but in which (intuitively) actions included the possibility to maneuver in the Z axis as well, increasing significantly the dimension of the state space. The measurements of such a 3D maneuvering model took a very long time to converge and its lack of progress in the curriculum shows how the model still hasn't converged to an optimal policy.

For the models employing curriculum learning, we discovered a tendency for the models to reach a plateau early on in terms of steps. This may be due to the manual thresholding setup, which does not accurately increase difficulty. The only model which the learning process managed to reach the last difficulty level is the curriculum model.

It is of note how the distribution of reward of the models capable of obtaining a working policy shifts in time from the curiosity itself to the environmental reward, meaning a significant shift between curiosity-driven exploration and exploitation of the strategy learnt by the policy.

7. Conclusions

The empirical results show a certain decoupling between the environmental measures and the classical performance measures. Measuring performance in reinforcement learning settings is well-known to be tricky, as not always cumulative rewards, policy losses and other low-level measures are able to capture the effectiveness of the behaviour of the agent in the setting.

In the proposed setting, the experiments showed how curriculum learning can be an effective solution for improving the generalizing capabilities of the model, improving significantly how the agent behaves.

The proposed learning system shows that the policy that most commonly is converged to is essentially a random search strategy: the agent randomly explores the environment to find the target. This is demonstrated by the behaviours of the different models - to different levels of performances - which shows the agent randomly moving between obstacles, revisiting previously seen areas until he manages to get the target in the range of its sensors. This is probably due to the random nature of the environment generation, as no two episodes present the same environment, and as such the agent isn't able to memorize the layout of the environment (or portions of it), but is only able to generally try to avoid obstacles until the targets comes into sight.

This consideration represents a reasonable way to interpret results in environments whose structure is closer to the human built environment: whenever looking around to see if the target is finally in sight and moving towards it is possible without excessive risk of hitting an obstacles, the process yields good results, otherwise it leads to a failure. While this does not represent a negative result per se, it is a clear warning that the learning procedure, the environments employed to shape the learning process, can lead to unforeseen and undesirable results. It must be stressed that, whereas this is a sort of malicious exploitation of a behavioural model that was trained on some types of environments and that is being tested in different situations, other state of the art approaches specifically devised and trained to achieve proper pedestrian dynamics still do not produce results that are competitive with hand crafted models [11].

Possible future developments on the RL model side are:

1. **Implement memory**: adding memory to the agent (in the form of a RNN module) might allow it to form a sort of experience buffer for the current episode and allows it to explore the environment in a non-random fashion.
2. **Rework the reward and penalty functions**: the proposed reward and penalty are pretty simplistic, a possible enhancement to the penalty could be implementing soft-collisions, that is, scaling the negative reward obtained by the agent in a collision according to the velocity of the collision - safe, soft touches can be allowed.
3. **Compare different RL algorithms**: different reinforcement learning algorithms (A3C, DQN) might show different insights on the optimal way to implement intelligent agents in the proposed setting.

On the other hand, a different training procedure, including specific environments aimed at representing a sort of *grammar* of the built environment, that could be used to define a sort of curriculum for training specifically pedestrian agents, should be defined to more specifically evaluate the plausibility of the application of this RL approach to pedestrian modeling and simulation.

References

[1] R. S. Sutton, A. G. Barto, Reinforcement Learning: an Introduction, MIT press Cambridge, 2018.

[2] S. J. Russell, P. Norvig, Artificial Intelligence: A Modern Approach (4th ed.), Pearson, 2020.

[3] S. Bandini, S. Manzoni, G. Vizzari, Agent based modeling and simulation: An informatics perspective, Journal of Artificial Societies and Social Simulation 12 (2009) 4.

[4] D. Weyns, A. Omicini, J. Odell, Environment as a first class abstraction in multiagent systems, Autonomous Agents Multi-Agent Systems 14 (2007) 5–30.

[5] G. Vizzari, L. Crociani, S. Bandini, An agent-based model for plausible wayfinding in pedestrian simulation, Eng. Appl. Artif. Intell. 87 (2020). URL: https://doi.org/10.1016/j.engappai.2019.103241. doi:10.1016/j.engappai.2019.103241.

[6] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, O. Klimov, Proximal policy optimization algorithms, CoRR abs/1707.06347 (2017). URL: http://arxiv.org/abs/1707.06347. arXiv:1707.06347.

[7] N. Heess, D. TB, S. Sriram, J. Lemmon, J. Merel, G. Wayne, Y. Tassa, T. Erez, Z. Wang, S. M. A. Eslami, M. A. Riedmiller, D. Silver, Emergence of locomotion behaviours in rich environments, CoRR abs/1707.02286 (2017). URL: http://arxiv.org/abs/1707.02286. arXiv:1707.02286.

[8] W. Yu, G. Turk, C. K. Liu, Learning symmetric and low-energy locomotion, ACM Trans. Graph. 37 (2018) 144:1–144:12. URL: https://doi.org/10.1145/3197517.3201397. doi:10.1145/3197517.3201397.

[9] J. Schulman, S. Levine, P. Abbeel, M. I. Jordan, P. Moritz, Trust region policy optimization, in: F. R. Bach, D. M. Blei (Eds.), Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015, volume 37 of *JMLR Workshop and Conference Proceedings*, JMLR.org, 2015, pp. 1889–1897. URL: http://proceedings.mlr.press/v37/schulman15.html.

[10] D. Pathak, P. Agrawal, A. A. Efros, T. Darrell, Curiosity-driven exploration by self-supervised prediction, in: D. Precup, Y. W. Teh (Eds.), Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017, volume 70 of *Proceedings of Machine Learning Research*, PMLR, 2017, pp. 2778–2787. URL: http://proceedings.mlr.press/v70/pathak17a.html.

[11] F. Martinez-Gil, M. Lozano, F. Fernández-Rebollo, Emergent behaviors and scalability for multi-agent reinforcement learning-based pedestrian models, Simul. Model. Pract. Theory 74 (2017) 117–133. URL: https://doi.org/10.1016/j.simpat.2017.03.003. doi:10.1016/j.simpat.2017.03.003.