From one to many: Simulating groups of agents with Reinforcement Learning controllers.

Luiselena Casadiego and Nuria Pelechano

Universitat Politecnica de Catalunya

Abstract. Simulation of crowd behavior has been approached through many different methodologies, but the problem of mimicking human decisions and reactions remains a challenge for all. We propose an alternative model for simulation of pedestrian movements using Reinforcement Learning. Taking the approach of microscopic models, we train an agent to move towards a goal while avoiding obstacles. Once one agent has learned, its knowledge is transferred to the rest of the members of the group by sharing the resulting *Q-Table*. This results in individual behavior leading to emergent group behavior. We present a framework with states, actions and reward functions general enough to easily adapt to different environment configurations.

Keywords: crowd simulation, reinforcement learning

1 Introduction

The basic requirement for most crowd simulation applications, is to model pedestrian movements with perception and placement. The perception encloses all knowledge the agent has about the environment including the situation of the other agents and its own status, while placement defines the desired position for an agent given the perception. Perception and placement could be defined as *state* and *action*.

Traditional microscopic methods specify each individual behavior and obtain emergent behavior from their autonomous interactions [1]. Data driven methods [2] imitate human movement patterns, but there is no learning involved. In contrast, reinforcement learning approaches aim at allowing the agents to learn individual behaviors based on what they experience [3]. Torrey [4] proposes reinforcement learning as a viable alternative method for crowd simulation. Cuayáhuitl et al. [5] presented an approach for inducing adaptive behavior of route instructions. Martínez et al. [6] propose a new methodology that uses different iterative learning strategies, combining a vector quantization with Q-Learning algorithm.

We present a Reinforcement Learning (RL) approach where by training one agent and transferring its knowledge to a group of agents, we can achieve emergent behavior while reducing the training time. We discuss in detail the main challenges when designing crowd behavior based on RL and provide ideas for future work in this field.

2 Reinforcement Learning for agent simulation

We start with one agent that learns to walk avoiding static obstacles to reach a goal position. This knowledge is then transferred to the rest of the agents, so that they can either use it as it is or further build upon it.

Our method is based on *Q-Learning* with ϵ-greedy policy. At each time step t, the agent receives some representation of the environment *state*, $s_t \in \mathcal{S}$, where $\mathcal{S}$ is the set of possible states, and on that basis selects an *action*, $a_t \in \mathcal{A}(s_t)$, where $\mathcal{A}(s_t)$ is the set of actions available in state s_t. After performing the action, the agent receives a *reward,* $r_{t+1} \in \mathbb{R}$, and moves to a new state, s_{t+1}. The agent seeks to maximize the R_t which is defined as a function of the reward sequence $r_{t+1}, r_{t+2}, r_{t+3}, ...$

The behavior learned by an agent is stored in a *Q-table* of $n \times m$ entries, where $n = |\mathcal{S}|$ and $m = |\mathcal{A}|$. Each cell of this table contains the value learned for each pair (s, a). Once this table is filled with learned behaviors, it can be transferred to other agents. Therefore the other agents do not need to learn from scratch, but can still continue the learning process by using small $\epsilon > 0$. This provides heterogeneity in behaviors while still allowing fast learning for groups of agents.

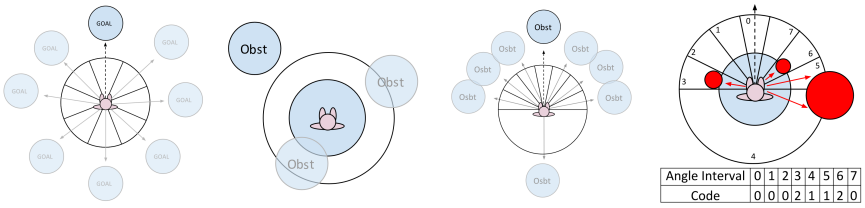
3 Learning problem definition

The action set defines the movements the agents can make during the simulation and it is discretized in 8 possible directions. The states encode the nearby environment and we evaluated two approaches: nearest obstacle and occupancy code. Depending on the state definition, we also need to define different reward functions which must be carefully chosen with two principles in mind: collision must be punished, and moving towards the goal has to be rewarded positively.

Nearest obstacle: The states are defined by the goal position $\mathcal{S}_{goal}$ (in range $[0, 7]$, figure 1a) , the distance to the nearest obstacle $\mathcal{S}^d_{obs}$ (in range $[0, 2]$, figure 1b), and the relative position of this obstacle $\mathcal{S}^p_{obs}$ (figure 1c). The obstacle position state is encoded with seven values for the front half of the circle (which simulates human perception) and one value for the back half.

The number of states is $S = |\mathcal{S}_{goal}| \times |\mathcal{S}^d_{obs}| \times |\mathcal{S}^p_{obs}|$ (192 states). Since the *Q-table* is small, the learning process is very fast, but there are several problems with such simple approach: (i) it only considers the nearest object at each time step, which can cause instabilities and local minima when several obstacles are too close to the agent and ii) there is no knowledge about the size of obstacles.

Occupancy code: The code is defined by $|\mathcal{S}^p_{obs}|$ values, where each value of the code $o[i]$ with $i \in [0, \mathcal{S}^p_{obs} - 1]$ indicates the distance to an obstacle according to $|\mathcal{S}^d_{obs}|$ (distance states). Figure 1d shows graphically this distribution of space with an example of occupancy code. The upper row depicts the number of angle interval and the bottom row shows the occupancy level of each interval. Note how obstacle size is represented by the number of regions being occupied. This representation also correctly encodes gaps between obstacles and aids the agent to learn whether it can walk through or needs to walk around a set of obstacles. The number of states is: $S = |\mathcal{S}_{goal}| \times |\mathcal{S}^d_{obs}|^{|\mathcal{S}^p_{obs}|}$ (52,488 states).



(a) (b) (c) (d)

Fig. 1: Goal state definition (a), distance to obstacle state (b), and goal position states for the two approaches: nearest obstacle (c) and occupancy code (d).

Reward Function The reward function is calculated as: $r(s_t, a_t) = r_g + r_o$, where r_g represents the positive reward gained by moving towards the goal and r_o the negative reward obtained from moving towards an obstacle. We tested two different r_g functions, (i) based on distance gained, and (ii) based on velocity vector being close to the goal direction. We observed that the second one resulted in straighter trajectories. Since the occupancy code encodes obstacles better we will focus only on the reward function for this case:

$$r_o(t) = \begin{cases} 0, & \text{if } d_o > \tau_2 \\ -\sum_{i=0}^{|\mathcal{S}^d_{obs}|-1} \frac{10}{10^{(o[i]-1)*2}}, & \text{otherwise} \end{cases} \tag{1}$$

where d_o is the distance between the agent and the obstacle.

4 Results

In this work we presented an RL approach to train one agent, and then apply knowledge transfer to the rest of the individuals in the crowd. The other agents can further learn during the simulation phase, by keeping a small $\epsilon > 0$ to leave room for some exploration. This allows us to reduce the time needed to train groups of agents, by separating the learning phase from the size of the simulated crowd.

During the training phase, we have one agent moving towards a goal and an obstacle somewhere in the scenario. Avoiding obstacles successfully depends on having an accurate state representation and finding a good equilibrium in the reward function between the positive feedback of moving towards the goal and the negative feedback of a collision. Our approach based on occupancy code better captures the complexity of the environment, and it is able to successfully reach the goal despite encountering more challenging situations (figure 2 top). Once the knowledge is transferred to several agents, we can simulate a group of agents reaching their individual goals while avoiding each other (figure 2 (bottom)). Note how agents are able to avoid moving obstacles (i.e. other agents), despite the training being done with static obstacles.

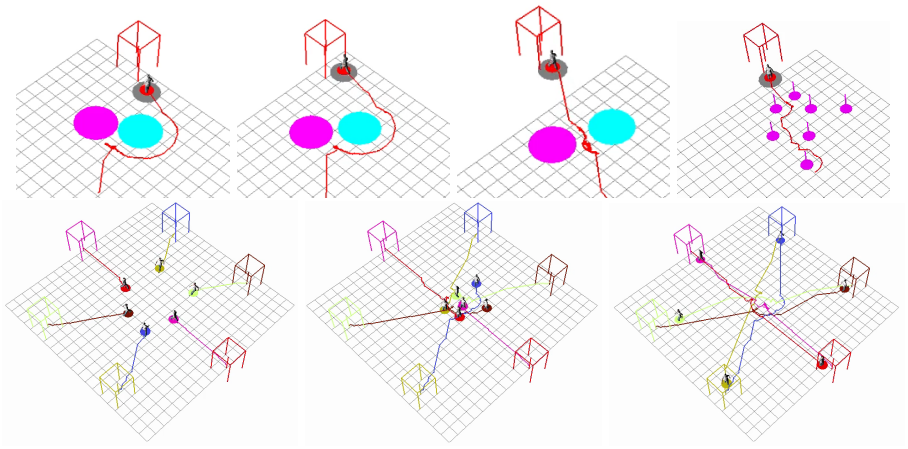


Fig. 2: Results of agents' reaching their goals while avoiding obstacles and agents.

Our preliminary results look promising, as we manage to have a group of agents wandering a virtual environment avoiding moving obstacles and reaching their destinations. But this work has also shown the challenges regarding choosing the right state representation and more importantly the right reward function. The goal of this research was to escape the cumbersome work of tweaking parameters in crowd simulation models, and while we had success with this, we did encounter a number of challenges in finding and tuning reward functions. Future work will focus on partially automating this selection process by incorporating inverse reinforcement learning techniques.

Acknowledgment: Funded by the Spanish Ministry, TIN2014-52211-C2-1-R.

References

1. Pelechano, N. and Allbeck, J. and Badler, N.: Virtual Crowds: Methods, Simulation, and Control. Morgan & Claypool Publishers. 2008.
2. Charalambous, P. and Chrysanthou, Y.: The PAG Crowd: A Graph Based Approach for Efficient Data-Driven Crowd Simulation. Computer Graphics Forum, 33(8), 95–108 (2014)
3. Sutton, R.S and Barto, A.G. : Introduction to reinforcement learning. MIT Press. (1998)
4. Torrey, L. : Crowd Simulation Via Multi-Agent Reinforcement Learning. Artificial Intelligence and Interactive Digital Entertainment (AIIDE). (2010)
5. Cuayáhuitl, H., Dethlefs, N., Frommberger, L., Richter, K. F., and Bateman, J.: Generating adaptive route instructions using hierarchical reinforcement learning. Spatial Cognition vii. 319–334 (2010)
6. Martínez-Gil, F., Lozano, M., and Fernández, F. : Strategies for simulating pedestrian navigation with multiple reinforcement learning agents. Autonomous Agents and Multi-Agent Systems, 29(1), 98–130 (2015)