Towards a multi-agent non-player character road network: a Reinforcement Learning approach

Stela Makri
CYENS - Centre of Excellence
Nicosia, Cyprus
s.makri@cyens.org.cy

Panayiotis Charalambous
CYENS - Centre of Excellence
Nicosia, Cyprus
p.charalambous@cyens.org.cy

***Abstract*—Creating detailed and interactive game environments is an area of great importance in the video game industry. This includes creating realistic Non-Player Characters which respond seamlessly to the players actions. Machine learning had great contributions to the area, overcoming scalability and robustness shortcomings of hand-scripted models. We introduce the early results of a reinforcement learning approach in building a simulation environment for heterogeneous, multi-agent non-player characters in a dynamic road network game scene.**

***Index Terms*—reinforcement learning, multi-agent, dynamic environment, Non-Player Characters, traffic simulation**

I. Introduction

An ongoing challenge in video game design, is the creation of believable, dynamic environments which respond to the user's decisions, in real time. Detailed and interactive gaming environments aid the feeling of immersiveness in the fictional world, enhancing the gaming experience and the overall user enjoyment [1]. Machine learning has several applications in generating such content, Non-Player Character (NPC) modelling being one of them [2].

As their name suggests, NPCs are dynamic characters whose behaviour is not controlled by the player. NPCs either interact with the player as enemies or allies, or they just contribute to the aesthetics of the game [3]. In either case, as the game scene evolves and progresses, NPCs should adapt seamlessly and in real-time to the new environment [4]. Therefore, scripting their behaviour by hand during the game development stage is often not feasible, as such models are not easily scalable to new environments and are prone to errors. Instead, artificial intelligence provided the means for significant advances in controlling NPCs.

In this study, we work towards developing a system of heterogeneous, interacting agents [5] in a dynamic environment, modelling traffic flow. The agents may include both vehicles and pedestrians. An agent's behaviour is influenced by its interaction with other agents and external parameters. The latter could be static, e.g. environment boundaries, or dynamic, such as traffic light signals and road disruptions.

This project has received funding from the European Union's Horizon 2020 Research and Innovation Programme under Grant Agreement No 739578 and the Government of the Republic of Cyprus through the Deputy Ministry of Research, Innovation and Digital Policy.

Ultimately, we are interested in the collaborative-competitive scenario, where the agents exhibit social behaviour. The goal is to infer efficiently and in real-time a model for traffic flow, that captures, at least in the macroscopic scale, patterns seen in real life scenarios - e.g., a sense of priority at the intersections or velocity perception to avoid collisions - and which scales well with the number of agents in the scene.

We adopt a Reinforcement Learning (RL) strategy [6] to train the multi-agent system, which allows exploring the continuous state space thoroughly, without the need to process large, detailed and exhaustive datasets to capture the complexity of the agent interactions. Moreover, RL permits training the heterogeneous interacting agents concurrently, reaching a common optimal solution which is our ultimate goal. RL was previously used to model traffic signal synchronisation for traffic congestion [7], [8], but our goal is to attack a more general problem, with different categories of agents interplaying in a dynamic environment.

II. Preliminaries

A. Reinforcement Learning

Like any machine learning approach, RL seeks to learn a rule or mapping; in the particular case to associate observations to actions. It does so by maximising a reward function expressing a long-term objective [6], [9]. As already mentioned, contrary to traditional machine learning techniques, RL does not require large exemplary datasets describing the mapping in question. Motivated from human intelligence, in RL, agents learn by trial-and-error as they explore their environment, by making educated guesses based on the feedback received from their last actions [6].

To be more precise, at any time $t \in \mathbb{N}$, an agent is observed to be in a state s_t. The rule underlying how the agent should act upon s_t is expressed by the policy π, associating s_t to an action a_t. The sequence of states and actions $(s_0, a_0, s_1, a_1, \dots)$ is the trajectory τ. Each time an action is cast and a state is observed, a reward $r_t = r(s_t, a_t, s_{t+1})$ is returned, informing the agent how good the transition from state s_t to state s_{t+1} given action a_t was. The objective of RL is to find the policy π that maximises a notion of the cumulative reward, the *return* $R(\tau)$. One may use an artificial Neural Network together with an optimisation method to find an expression for π [10].

B. Proximal Policy Optimisation

Different RL methods use different *Value functions* as objective functions to perform optimisation and these typically fall into one of the two: On-policy and Off-Policy Value functions [9]. For our preliminary results we employed an On-Policy Value Function but we will not be restricted to this necessarily.

There are several methods in literature to solve the RL optimisation problem. We restrict our attention to the Proximal Policy Optimisation (PPO) method [11], which enjoys algorithmic simplicity, data efficiency and robustness. The method converges to an optimal solution by iterating between sampling from the current policy estimate and performing several optimisation steps on the sampled data, using stochastic gradient ascent.

C. Long Short-Term Memory

Delving into the problem-specific algorithmic details, our agents can benefit from acquiring some notion of memory of previous states. For example, a vehicle agent could benefit from knowledge of the traffic light sequence, or knowledge of the speed of approaching agents. Long Short-Term Memory (LSTM) [12] is a class of Recurrent Neural Networks (RNN) incorporating feedback architecture to retain 'memory' of past events over arbitrary time intervals. This is achieved by preventing back-propagating error information from becoming arbitrarily small [12]. Thus, using the LSTM architecture to train our RL algorithm will allow our agents to retain information of past events over long or short periods of time.

D. Sparse Rewards

A common obstacle in RL is converging to a suboptimal solution. This is strongly correlated to the choice of reward function. Thus constructing a good reward function is crucial. However, there might be factors that are hard to circumvent. A frequently occurring issue is that of *sparse reward signals*. That is, the agent spends most of its time exploring the space while receiving nearly zero or simply constant reward signals because no important event occurred. However if a significant amount of time passes with little change in the cumulative reward the agent might be 'discouraged' to explore the environment any further. Essentially this means that the RL algorithm converged to some local maximum of the expected return, but not the intended one [13], [14].

Consider the following: agents in our environment have a predefined destination. Suppose our agent is a vehicle and its target is located straight ahead after it crosses an intersection. The agent should navigate straight ahead, wait at the crossing for the green light and then move forwards a bit more until it reaches the goal. For the most part of its journey, the agent moves only forwards, while receiving some constant reward (possibly zero) as not much has changed. We can help the agent learn by planning an appropriate learning strategy, e.g. instantiating the agent near the target during the first few training episodes. But that might not be sufficient, as it is hard to predict all local optimal solutions an agent might come across during its learning journey.

We will use a *curiosity* driven approach [13]. An intrinsic reward is assigned to each agent in addition to the extrinsic one introduced in Sec. II-A. This intrinsic reward favours unfamiliar actions to be explored, by amounting for the prediction error of the next state given the current state and an auxiliary action. The intrinsic and extrinsic rewards are added together to make up the total reward fed to the RL training algorithm. Generating intrinsic reward signals is achieved with the aid of two coupled neural networks, comprising the Intrinsic Curiosity Module (ICM) [13].

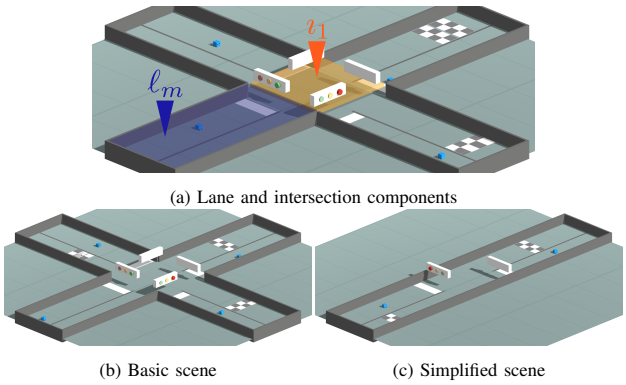
III. Methodology and Results

We implemented our game environment with the help of the ML-Agents Toolkit of the Unity Engine [15]. Training was performed using the PyTorch library [16]. A demonstration can be found at https://youtu.be/-qwFddCkyDo.

We are assuming the left-hand traffic system. Our game scene consists of bi-directional *lane components*, $\{\ell_m\}_{m=1}^{M_1}$, and *intersection components*, $\{\imath_m\}_{m=1}^{M_2}$, complemented by a set of traffic lights. We marked these on diagram Fig. 1a. For the purposes of training we use a simple crossroad scene, comprised of a single intersection component and four lane components, illustrated in Fig. 1b. We call this the basic scene. We will also use a simplified intersection scene, as shown in Fig. 1c to assist training. An episode begins with one agent in each of the lane components in the scene. We will describe the set of actions allowed for each agent, as well as the set of observations describing a state and the rewards received.

A. Actions

The action space of an agent is spanned by 5 discrete movements: moving forward by a fixed amount $d \in \mathbb{R}$, moving backwards by the same amount d, turning clockwise at a 2° angle, turning anticlockwise at a 2° angle and staying idle. In our implementation, the ratio of d to the length of a lane component is $1:100$.



(a) Lane and intersection components

(b) Basic scene (c) Simplified scene

Fig. 1: Stills from the game scene: (a) shows a lane component ℓ_m shaded in blue and intersection component $\imath_1$ shaded in orange, (b) shows the basic crossroad scene and (c) the simplified scene.

B. Observations

Three stacked observations will be used to feed the LSTM network, capturing the information of positions, velocities and accelerations of agents in the environment (see Sec. II-C).

Each agent makes a total of $5\times(7+2)+2 = 47$ observations. More specifically, an agent is assigned 5-rays of overlapping sphere perception sensors, spanning a 70° angle (see Fig. 2), to detect the following 7 objects in its vicinity: other agents, scene boundaries, the traffic lights' waiting line and the four traffic light indications: green, amber, red, amber-red. Each of the 5 rays returns $7+2$ observations at each learning time step. The first 7 return binary information of hitting any of the 7 detectable objects. For the latter 2 observations, the first returns true if nothing was hit and false otherwise; this is important since a non-detectable object may intercept the rays sensors (for example one of the targets), and the second returns the fraction of the ray at which the hit occurred if any and 1.0 otherwise.

The remaining 2, out of the total of 47 observations, encode a notion of distance and direction. In particular, consider a single agent, located in lane component, ℓ_{A} and target destination located in, say ℓ_{T}. Let $\hat{\boldsymbol{e}}_{\mathrm{A}}$ and $\hat{\boldsymbol{n}}_{\mathrm{A}}$ be the unit vectors along and normal to the direction of motion of ℓ_{A}. Similarly, define $\hat{\boldsymbol{e}}_{\mathrm{T}}$ and $\hat{\boldsymbol{n}}_{\mathrm{T}}$ for ℓ_{T}. Let $\boldsymbol{x}_A$ and $\boldsymbol{x}_T$ be the positions of the agent and target respectively. We define the following two scalar observations:

$$\delta_{\mathrm{A}}(\boldsymbol{x}_{\mathrm{T}}, \boldsymbol{x}_{\mathrm{A}}) = (\boldsymbol{x}_{\mathrm{T}} - \boldsymbol{x}_{\mathrm{A}}) \cdot (\hat{\boldsymbol{e}}_{\mathrm{A}} + \hat{\boldsymbol{n}}_{\mathrm{A}}), \tag{1}$$

$$\delta_{\mathrm{T}}(\boldsymbol{x}_{\mathrm{T}}, \boldsymbol{x}_{\mathrm{A}}) = (\boldsymbol{x}_{\mathrm{T}} - \boldsymbol{x}_{\mathrm{A}}) \cdot (\hat{\boldsymbol{e}}_{\mathrm{T}} + \hat{\boldsymbol{n}}_{\mathrm{T}}). \tag{2}$$

To elaborate on this choice, let $h_1, h_2 \geq 0$ be the distances the agent has to traverse to reach its target as illustrated in Fig 3. As shown in Table I, the pair of observations $\{\delta_{\mathrm{A}}, \delta_{\mathrm{T}}\}$ uniquely defines the distances h_1 and h_2 and the three possible trajectories: no turn, right turn or left turn. However, we need to be careful with the above choice, as this is not the case when $h_1 \ll h_2$, i.e after the intersection, and the agent might end up in a different lane than the one of its desired destination. We observed that this issue is resolved in most cases due to the memory of the agent's velocity (see Sec. II-C), although this is not necessarily an issue for us as long as the agent does not drive back into its starting lane component. Identifying the correct target is not our goal for the purposes of constructing a model for controlling NPCs. What is important is that, the agents follow acceptable trajectories and the target is used only to guide the agent during training.

In practice, feeding the RNN with these 2 observations to describe the position of the target instead of the actual position of the target relative to the agent was more robust during training and required minimal hyperparameter tuning to achieve convergence of the RL algorithm. This can be attributed to fact that, in the second case, not enough information is encoded regarding the geometry of the system.

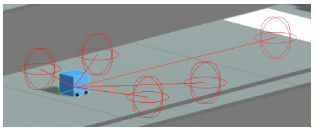


Fig. 2: Still from the scene view, showing the 5 ray sensors cast by an agent, to detect nearby objects.

C. Reward

In a time step t of an episode k, we consider 9 events in which the agent is awarded a reward $r_{t,k}$, listed in Table II. We terminate each episode after a maximum of $T = 3000$ time steps, after which an episode is considered unsuccessful. A total reward $R_k = \sum_{t=0}^{T} \gamma^t r_{t,k} < 1$ for a discount factor $\gamma \in (0, 1)$ is assigned to each episode k. Deciding the values of $r_{t,k}$ was achieved by hand-tuning.

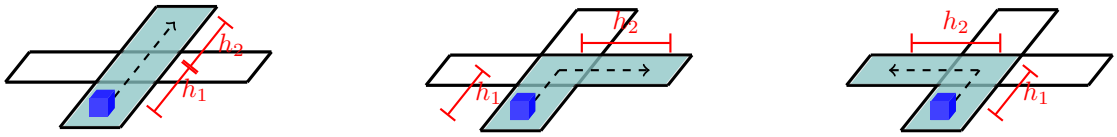
D. Training and Results

We trained an LSTM RNN architecture using the PPO method, as explained in Sec. II. In particular, the RNN consists of 2 hidden layers, each having 128 units. We keep a sequence of 64 experiences and a memory buffer of size 128 for the LSTM. A linearly decaying learning rate of 3×10^{-4} and a batch of 128 experiences is used for each iteration of gradient descent. The policy is updated after a total of 1024 experiences are collected. For the PPO implementation, we set the entropy coefficient $\beta = 0.01$, the change in policy is restricted within a threshold of $\epsilon = 0.2$, the regularisation parameter for the Generalised Advantage Estimator (GAE) [17] is set to $\lambda = 0.95$ and the number of epochs through the experience buffer while performing gradient descent optimisation is set to 3. Extrinsic rewards carry a strength of 1 and are discounted by a factor of $\gamma = 0.99$ in calculating the expected return. Curiosity intrinsic rewards carry a strength of 0.1 and are discounted by a factor of $\gamma = 0.99$. The ICM network is made up of 2 hidden layers each with 128 units and this is trained with a learning rate of 3×10^{-4}. Rewards are truncated after a time horizon of 64. Training was allowed to reach a total of 7×10^6 episodes.

To motivate learning, we start training on a restricted scene: lane components are kept short and both the agent and the target are located near the intersection. After a while, the lanes' length increases progressively, until the lanes reach their full extend. To prevent the agent from converging to an idle state before it explores the scene thoroughly (see Sec. II-D on sparse events), the termination of an episode varies through training. In particular, termination is always invoked if the number of training time steps t exceeds $T = 3000$, if the target is reached,

TABLE I: Observed values, δ_{A} and δ_{T} defined by (1) and (2) of the agent's and target's position $\boldsymbol{x}_{\mathrm{A}}$ and $\boldsymbol{x}_{\mathrm{T}}$. The values of $|\delta_{\mathrm{A}}| - |\delta_{\mathrm{T}}|$, $\delta_{\mathrm{A}} + \delta_{\mathrm{T}}$ and $\delta_{\mathrm{A}} - \delta_{\mathrm{T}}$ are also depicted, illustrating that they define uniquely the trajectory and distances the agent should traverse to reach its target.

Quantity	**Trajectory**		
	No turn	***Right turn***	***Left turn***
$\delta_{\mathrm{A}}(\boldsymbol{x}_{\mathrm{T}}, \boldsymbol{x}_{\mathrm{A}})$	$h_1 + h_2$	$h_1 + h_2$	$h_1 - h_2$
$\delta_{\mathrm{T}}(\boldsymbol{x}_{\mathrm{T}}, \boldsymbol{x}_{\mathrm{A}})$	$-h_1 - h_2$	$h_1 - h_2$	$-h_1 - h_2$
$\lvert\delta_{\mathrm{A}}\rvert - \lvert\delta_{\mathrm{T}}\rvert$	$+$	0	$-$
$\delta_{\mathrm{A}} + \delta_{\mathrm{T}}$	0	$2h_1$	$-2h_2$
$\delta_{\mathrm{A}} - \delta_{\mathrm{T}}$	$2(h_1 + h_2)$	$-2h_2$	$2h_1$



(a) No turn (b) Right turn (c) Left turn

Fig. 3: Schematic representation of the desired trajectories (dashed line) the agent (blue cube) should follow: (a) no turn required, (b) right turn, and, (c) left turn. The teal tiles represent the lane components the agent will trace. The agent is required to move a distance $\sim h_1$ up to the intersection and a distance $\sim h_2$ thereafter to reach the target.

TABLE II: Events giving a reward $r_{t,k}$ to the agent in time step t out of a total of T time steps, of an episode k. Rewards are summed to give a cumulative reward $R_k = \sum_{t=0}^{T} \gamma^t r_{t,k}$, where $\gamma \in (0, 1)$ is a discount factor. All episodes begin with $R_k = 0$.

Event	**Reward** $(r_{t,k})$
reached target	1
crossed green light	$R_k + 1.0/T$
crossed amber light	$R_k - 0.8/T$
crossed red light	-1
crossed red-amber light	-0.5
crossed lights against stream	-1
collision with obstacles	-1
collision with agent	-1
unsuccessful step (target not reached)	$R_k - 0.1/T$

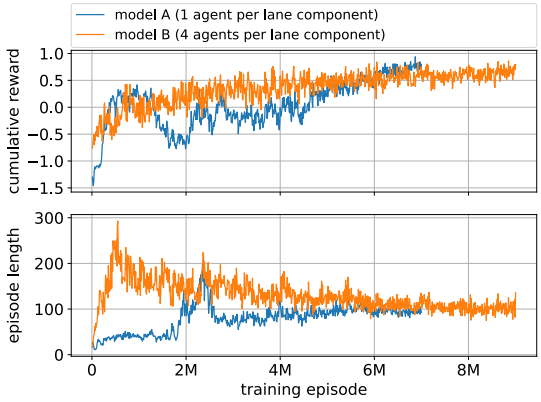


Fig. 4: Cumulative reward received (top) and episode length (bottom) during training. In blue we show the results for training session of *model A*, where at $t = 0$, each lane component is occupied by 1 agent, and in orange we have the results for the training session of *model B* with 4 agents per lane component at $t = 0$ and with each agent's network initialised from *model A*.

if the agent collided with a boundary or an agent, and if the agent crossed the intersection line in the opposite direction. At later episodes, crossing the intersection at the red or amber-red lights also causes the episode to terminate. When inferring a trained model the latter termination condition is ignored.

During training, 12 replicas of the basic scene were used, together with 4 replicas of a simplified scene (refer to Fig. 1). Fig. 4 shows, in blue, the cumulative reward and respective episode length per training episode. We refer to the resulting model as *model A*. The results of inferring the model are shown in the first column of Table III, where from a sample of 5000 episodes we record the number of times per episode an agent crosses a traffic light, reaches its target, collides with another agent or a scene boundary, as well as, the mean number of time steps needed to complete a single episode. As seen in Table III and expected from the plots of Fig. 4, in *model A*, agents respond to the traffic lights and traverse the correct trajectory with high success rate. However, the agents' motion is not smooth, which is reflected in the relatively large episode length. Instead, the agents often change their orientation abruptly and have a tendency to wonder while waiting at the traffic signals. This is reflected in the results of Table III by the fact that summing the numbers the agents cross the traffic lights per episode is greater than 1. Finally, even though in Table III we see that the number of collisions per episode is small, in reality the agents rarely avoid collisions with other agents, a consequence of the small number of agents in the scene, resulting in agents rarely meeting other agents during training and inference.

To treat in-between agents collisions, we subsequently trained a new model, by initialising the network of each agent from *model A*. This time we populated the scene with 4 agents in each lane and we use the full extend of each lane component throughout training (no need to restrict the boundaries of the scene, since the agents inherit knowledge of the environment from *model A*) and we also relaxed the termination condition when agents cross the intersection at the red or amber-red lights. The rest of the configuration parameters are kept the same, as during training of *model A*, but we terminated training at 9×10^6 episodes. We refer to the resulting model as *model B*. The cumulative reward and episode length per training episode are plotted in orange in Fig. 4. The remaining columns of Table III, summarise the results of inferring *model B* on a scene of 1,2,3 and 4 agents per lane, recorded from samples of 5000 episodes.

Comparing the results of Table III for the two models, *model B* performs similarly to *model A* with respect to crossing the traffic lights and reaching the agent's target, independently of the number of agents in the scene and despite relaxing the episode termination condition when crossing the red and amber-red lights when training *model B*. The two models perform similarly with respect to the mean episode length, as well. Further we see that *model B* successfully reduces the number of collision with scene boundaries by 5 times, independently of the number of agents in the scene and comparing the models for the cases where 1 agent per lane is used, the number of collisions with other agents is 5 times smaller for *model B* (the number of collisions with agents appears to be increasing linearly with the number of agents per lane). However, even though interactions between agents are improved for *model B*, there are other shortcomings yet to be resolved, like achieving smooth motion for the agents.

TABLE III: Number of times per episode an agent crosses a light, reaches its target and collides with other agents or scene boundaries when inferring *model A* on a scene of 1 agent per lane and and *model B* on a scene of 1,2,3 and 4 agents per lane on samples of 5000 episodes. The mean episode length for each case is also depicted.

Event per episode	**Model (agents per lane)**				
	A (1)	**B (1)**	**B (2)**	**B (3)**	**B (4)**
green light	0.72	0.72	0.69	0.67	0.63
amber light	0.29	0.29	0.29	0.27	0.28
red light	0.033	0.024	0.026	0.021	0.024
amber-red light	0.0056	0.0018	0.003	0.0016	0.002
reached target	0.88	0.96	0.91	0.85	0.81
agent collision	0.024	0.0048	0.052	0.11	0.16
boundary collision	0.054	0.0086	0.012	0.013	0.012
Mean episode length	600	570	570	570	590

IV. Conclusions and Future Work

We presented a reinforcement learning strategy to train a multi-agent environment of vehicles, navigating a simple four-way road intersection to reach a target destination, whilst interacting with other vehicles and obeying the traffic lights.

The trained model succeeds in classifying the 4 traffic light signals and identifying the correct trajectory to reach the target destination relatively fast. However it falls short in resolving prospective collisions and often this leads to agents 'forgetting' their planned trajectory upon meeting another agent. This was improved after training with a larger number of agents in the scene. Calibrating the LSTM hyperparameters will likely help treating this further. Moreover, an agent's motion exhibits undesirable fluctuations along its axis of motion and at the red traffic lights, agents tend to move around until the lights turn green. Nonetheless, the results are promising in respect to successfully navigating the road network. Achieving better scalability seems plausible, as suggested by the convergence rate and stability of the second training session.

After resolving the above shortcomings, we will proceed to achieve our ultimate goal which is also the most challenging part. That is, to introduce different families of agents in the system, like pedestrians, stray animals, etc. (see for example [18]), and to enhance the environment with parameters such as pavements, traffic signs, road disruptions, etc. Until now, we focused on agents sharing the same self-centred policy, resulting in competitive agents, each seeking to maximise their own reward function. The only sense of collaboration comes intrinsically from each agent's goal to avoid other agents, without explicit communication. For our next steps we seek to generalise the reward function to one that allows communication across agents and thus enhance collaboration between them. This will allow, not only to enhance collision avoidance, but also to describe more complex behaviours when our system involves different types of agents and more perplexed environments.

References

[1] P. Sweetser and P. Wyeth, "Gameflow: A model for evaluating player enjoyment in games," *Comput. Entertain.*, vol. 3, no. 3, p. 3, Jul. 2005.

[2] N. Justesen, P. Bontrager, J. Togelius, and S. Risi, "Deep learning for video game playing," *IEEE Transactions on Games*, vol. 12, no. 1, pp. 1–20, 2020.

[3] J. E. Laird and M. v. Lent, "Human-level ai's killer application: Interactive computer games," in *Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on Innovative Applications of Artificial Intelligence*. AAAI Press, 2000, p. 1171–1178.

[4] K. Merrick and M. L. Maher, "Motivated reinforcement learning for non-player characters in persistent computer game worlds," in *Proceedings of the 2006 ACM SIGCHI International Conference on Advances in Computer Entertainment Technology*, ser. ACE '06. New York, NY, USA: Association for Computing Machinery, 2006, p. 3–es.

[5] G. Weiss, Ed., *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*. Cambridge, MA, USA: MIT Press, 1999.

[6] R. Sutton and A. Barto, *Reinforcement Learning: An Introduction*, ser. Adaptive Computation and Machine Learning series. MIT Press, 2018.

[7] X. Liang, X. Du, G. Wang, and Z. Han, "A deep reinforcement learning network for traffic light cycle control," *IEEE Transactions on Vehicular Technology*, vol. 68, no. 2, pp. 1243–1253, 2019.

[8] W. Genders and S. Razavi, "Using a deep reinforcement learning agent for traffic signal control," 2016.

[9] C. Szepesvári, "Algorithms for reinforcement learning," *Synthesis Lectures on Artificial Intelligence and Machine Learning*, vol. 4, no. 1, pp. 1–103, 2010.

[10] V. François-Lavet, P. Henderson, R. Islam, M. G. Bellemare, and J. Pineau, *An Introduction to Deep Reinforcement Learning*, 2018.

[11] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *CoRR*, vol. abs/1707.06347, 2017.

[12] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, p. 1735–1780, Nov. 1997.

[13] D. Pathak, P. Agrawal, A. A. Efros, and T. Darrell, "Curiosity-driven exploration by self-supervised prediction," in *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ser. ICML'17. JMLR.org, 2017, p. 2778–2787.

[14] Y. Burda, H. Edwards, D. Pathak, A. J. Storkey, T. Darrell, and A. A. Efros, "Large-scale study of curiosity-driven learning," *CoRR*, vol. abs/1808.04355, 2018.

[15] A. Juliani, V. Berges, E. Vckay, Y. Gao, H. Henry, M. Mattar, and D. Lange, "Unity: A general platform for intelligent agents," *CoRR*, vol. abs/1809.02627, 2018.

[16] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds. Curran Associates, Inc., 2019, pp. 8024–8035.

[17] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel, "High-dimensional continuous control using generalized advantage estimation," 2018.

[18] Z. Ren, P. Charalambous, J. Bruneau, Q. Peng, and J. Pettré, "Group modeling: A unified velocity-based approach," *Computer Graphics Forum*, vol. 36, no. 8, pp. 45–56, 2017.