Multi-agent reinforcement learning for character control

The Visual Computer (2021) 37:3115–3123
https://doi.org/10.1007/s00371-021-02269-1

SURVEY

Multi-agent reinforcement learning for character control

Cheng Li[1] · Levi Fussell[1] · Taku Komura[1,2]

Accepted: 23 July 2021 / Published online: 4 December 2021

Abstract
Simultaneous control of multiple characters has been a research topic that has been extensively pursued for applications in computer games and computer animations, for applications such as crowd simulation, controlling two characters carrying objects or fighting with one another and controlling a team of characters playing collective sports. With the advance in deep learning and reinforcement learning, there is a growing interest in applying multi-agent reinforcement learning for intelligently controlling the characters to produce realistic movements. In this paper we will survey the state-of-the-art MARL techniques that are applicable for character control. We will then survey papers that make use of MARL for multi-character control and then discuss about the possible future directions of research.

Keywords Multi-agent reinforcement learning · Character control · Computer animation

1 Introduction

Controlling multiple agents under adversarial and cooperative scenarios has been a research topic that has been intensively pursued by researchers in artificial intelligence, computer graphics and robotics in the past few decades. There are a wide range of applications of which include autonomous driving [13], swarm robotics control in warehouses [10], NPC control in computer games [37,43] and background character control in films [35,48].

Especially in computer animation, starting from the landmark research of the boids model by Reynolds [32] for animating flocks of birds and schools of fish, extensive research has been done for crowd animation [7] and close character interactions [12]. Before the deep learning era, researchers have focused on designing hand-tuned controllers or optimization-based approaches [21,37]. Although such methods are forming the foundation of multi-character controllers, the intelligence of the characters is limited due to the simplicity of the rules or the high computational complexity.

Recently, there is growing interest in building more intelligent character control by making use of algorithms such as reinforcement learning [15,16,25,36,43,47]. Especially with the introduction of deep reinforcement learning (DRL), the scale of the learnable systems in terms of generalization and data size has massively grown [33]. The idea of DRL has been applied to terrain running in 2D [26] and 3D [27], tracking human motion capture data [24], video motion tracking [28] and synthesizing an optimal series of motions for achieving a task [29].

Such algorithms are also being applied in the field of simultaneous multi-agent control, which is categorized in the area called multi-agent reinforcement learning (MARL). Amazing results where the agents are intelligently controlled to play football [18], hide-and-seek [1] and autonomous driving [13] have been achieved.

In this paper, we review the basics of MARL, especially in a scenario for controlling characters in a real-time scenario. Starting from the basics of RL, we then extend this to cover MARL in a centralized training and decentralized execution. We then review recent papers that apply MARL for multi-character control and related topics. We finally discuss some future direction of the research.

✉ Taku Komura
T.Komura@ed.ac.uk

Cheng Li
Cheng.Li@ed.ac.uk

Levi Fussell
levi.fussell@ed.ac.uk

1 School of Informatics, University of Edinburgh, 10 Crichton Street, Edinburgh, UK

2 The University of Hong Kong, Pokfulam, Hong Kong

2 Reinforcement learning background

The key idea that makes reinforcement learning different from supervised learning and unsupervised learning is that it learns from interactions with an environment and approximates some return based on these environment interactions to help ‘reinforce’ its decisions in that environment. The entire system can be modelled as a Markov decision process with a discrete-time setup ($t = 0, 1, 2, 3, ...$), where an agent observes the environment state s_t at time step t and takes an action a_t according to the observation. In the next time step $t + 1$, when the agent finishes the action, it receives a numerical reward r_{t+1} from the environment, and the environment state transits from s_t to s_{t+1}. Therefore, a complete interaction can be represented in a sequence, which is called a trajectory:

$$\tau = s_0, a_0, r_1, s_1, a_1, r_2, s_2, a_2, r_3, ... \tag{1}$$

Mathematically, the Markov decision process is written as a tuple $\langle S, A, R, P\rangle$, where S is the set of all possible states, A is the set of all possible actions, $R : S \times A \times S \to \mathbb{R}$ is the reward function with $r_t = R(s_t, a_t, s_{t+1})$, and $P : S \times S \times A \to [0, 1]$ is the transition function of the environment $P(s'|s, a)$ and describes the probability of transition to state s' when action a is taken at state s.

Reinforcement learning’s main target is to maximize the discounted sum of rewards it receives after having taken an action. This is defined as the return:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}, \tag{2}$$

where $\gamma \in [0, 1]$ is the discount factor. An agent’s choices on actions are decided by its policy π, which is a function that outputs the probability for taking action a at state s, normally written as $\pi(a|s)$. In addition to the policy, a value function is used to estimate the return that will be received given a state or state and action. The value function for state s under policy π is:

$$\begin{aligned} V_\pi(s) &= \mathbb{E}_\pi[G_t|S_t = s] \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma V_\pi(S_{t+1})|S_t = s], \end{aligned} \tag{3}$$

The value function for taking action a at state s under policy π is called a Q function that is defined as:

$$\begin{aligned} Q_\pi(s, a) &= \mathbb{E}_\pi[G_t|S_t = s, A_t = a] \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma \mathbb{E}_{a'\sim\pi}[Q_\pi(S_{t+1}, a')]|S_t = s, A_t = a]. \end{aligned} \tag{4}$$

Q-learning [44] is a classic algorithm in reinforcement learning which learns the Q function by approximating with a target Q function that always selects the optimal action:

$$\begin{aligned} Q(S_t, A_t) \leftarrow\ & Q(S_t, A_t) \\ & + \alpha[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)], \end{aligned} \tag{5}$$

where α is the learning rate.

Another kind of algorithm, known as policy gradient or policy optimization, learns a policy function directly with an gradient ascent on an empirical estimate of the true policy gradient. We define $J(\boldsymbol{\theta})$ as the expected return under the policy parameters $\boldsymbol{\theta}$:

$$J(\boldsymbol{\theta}) = \sum_{\tau\sim\pi_\theta} G(\tau)\pi_\theta(\tau), \tag{6}$$

where τ is a trajectory (see Eq. (1)), $\pi_\theta(\tau)$ is the probability that τ is created by policy τ_θ parameterized by θ and $G(\tau)$ is the return (see Eq. (2)) obtained by trajectory τ. So, the goal is to learn the policy parameters $\boldsymbol{\theta}$ that maximize $J(\boldsymbol{\theta})$:

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \eta\nabla J(\boldsymbol{\theta}), \tag{7}$$

where η is the learning rate and the derivative term can be derived in a from that can be estimated with sample trajectories:

$$\begin{aligned} \nabla_\theta J(\boldsymbol{\theta}) &= \mathbb{E}_{\tau\sim\pi_\theta}[\nabla_\theta \log \pi_\theta(\tau)G(\tau)] \\ &= \mathbb{E}_{\tau\sim\pi_\theta}[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t)G(\tau)]. \end{aligned} \tag{8}$$

Combining the two key components, value functions and policy functions, and learning them together is the main idea behind the most commonly used kind of algorithm nowadays, known as Actor-Critic, with a value function acting as the critic and a policy function acting as the actor. The critic learns a function that evaluates the trajectory, which can be the value function $V(s)$, the Q function $Q(s, a)$, the advantage function $V(s) - Q(s, a)$ or other variations, and the actor learns a policy function in the direction indicated by the critic.

With the help of deep learning, reinforcement learning algorithms can solve problems in applications on much larger scales. Specifically, deep reinforcement learning algorithms learn value functions and policy functions using neural networks. For example, deep Q-learning [23], which is the deep learning version of Q-learning, calculates a loss

$$(\underbrace{r + \gamma \max Q(s', a'; \theta_i^-)}_{\text{target}} - \underbrace{Q(s, a; \theta_i)}_{\text{prediction}})^2 \tag{9}$$

to update the Q function instead of updating it in a direct manner shown in Eq. (5). Some other deep reinforcement learning algorithms, such as proximal policy optimization (PPO) [8, 33], deep deterministic policy gradient (DDPG) [17], and asynchronous advantage actor-critic (A3C) [22] are good examples of applying deep learning in actor-critic methods.

3 Multi-agent reinforcement learning

When multiple reinforcement learning agents interact with the same environment, the environment becomes a multi-agent system and multi-agent reinforcement learning algorithms are applied. It can be modelled as a Markov game with the tuple $\langle N, S, A, R, P, O\rangle$, where N is the number of agents, S is the set of the true states, $A = \{A_1, \dots, A_N\}$ is the set of actions for all agents, $P : S \times S \times A \to [0, 1]$ is the transition function $P(s'|s, a_1, \dots, a_N)$ describing the probability of transition to state s' when actions $\{a_1, \dots, a_N\}$ are taken by the N agents, respectively, at state s, R is the reward function, and $O = \{O_1, \dots, O_N\}$ is the individual observations for all agents. Now, the goal is to learn policies for each agent that maximizes their individual returns or the group return, depending on the environment settings, which can be cooperative, competitive or mixed.

3.1 MARL setups for character control

We now discuss some factors and setups that we need to consider when applying MARL to character control.

3.1.1 Non-stationarity

The environment for a single reinforcement learning agent is assumed to be stationary in default, i.e. the state transition function and reward function do not change over time [11]. However, things are different for agents in multi-agent environments. As all agents are learning concurrently, their policies change as time changes, and the environment becomes non-stationary from the agents' perspectives. An agent would possibly perceive a different state s' after taking the same action a at the same state s with other agents having some different policies:

$$P(s'|s, a, \pi_1, \dots, \pi_N) \neq P(s'|s, a, \pi'_1, \dots, \pi'_N), \forall \pi_i \neq \pi'_i, \tag{10}$$

since the transition of the true state involves actions from all agents. This non-stationarity would cause the critic function to be non-stationary as well and, hence, leads to a poorly learnt policy, for any single agent.

3.1.2 Centralized training and decentralized execution

Centralized training and decentralized execution (CTDE) setup is an effective solution in dealing with the non-stationary problem, especially in multi-agent cooperative environments. The decentralized execution part stays the same as the normal setup, where the agents' policies output actions based on their partial observations respectively. The centralized training part "cheats" a little bit, where the critics have access to extra information, for example, the true state of the environment and actions of all agents, which can make the environment stationary, since:

$$\begin{aligned} &P(s'|s, a_1, \dots, a_N, \pi_1, \dots, \pi_N) \\ &\quad = P(s'|s, a_1, \dots, a_N) \\ &\quad = P(s'|s, a_1, \dots, a_N, \pi'_1, \dots, \pi'_N), \forall \pi_i \neq \pi'_i, \end{aligned} \tag{11}$$

Extra information is usually accessible in computer graphics applications, which is an advantage compared to other applications, such as autonomous driving, where simulating the real world is a very difficult task. Hence, we will introduce some multi-agent reinforcement learning algorithms that use the CTDE setup.

3.1.3 Self-play

Self-play is a method frequently used in multi-agent competition environments, such as tennis and chess. Instead of training against an agent with some other policies, the agent learns by competing with itself, i.e. its current best policy or a randomly chosen previous policy. The main idea behind it is to mimic how humans structure competitions. For example, it is more reasonable and efficient for a beginner tennis player to practise against other beginners than practising a world champion or a small child struggling to pick hold the racket. Multi-agent self-play environments can use single-agent RL methods, and some interesting works will be discussed later.

3.2 Algorithms for MARL

We now describe four MARL algorithms that are considered state-of-the-art and suitable for applications in character control.

3.2.1 Value-decomposition networks

VDN [40] is an MARL algorithm that can make agents learn their individual Q value functions when there is only a group reward available. VDN assumes that the joint action-value function for all agents can be decomposed into individual action-value functions for each agent:

$$Q_{\text{tot}}(\mathbf{s}, \mathbf{a}) = \sum Q_i(s_i, a_i), \tag{12}$$

where the individual action-value functions depend on the local observations of each agent. As for the centralized training, Q_{tot} is trained with the deep Q-learning method, using the joint reward to back-propagates gradients into the networks, so the individual Q_i does not need to learn from specifically assigned rewards. And as for the decentralized execution, each agent acts greedily by taking the maximum Q_i value given its local observations. In this scenario, it is equivalent to selecting the joint action greedily, which ensures that the centralized policy and decentralized policy are consistent.

3.2.2 QMIX

QMIX [31] removes the constraint (Eq. (12)) in VDN by introducing a new Q_{tot} function. Instead of just taking the sum of all individual Q_is as Q_{tot}, QMIX uses a mixing network that takes all Q_is and the global state as input to approximate the value of Q_{tot}, which both increase the variety of relation representation between Q_{tot} and individual Q_is, and make better use of the centralized training by considering the global state. More specifically, as shown in Fig. 1, the hyper-networks inside take the global state information as input to generate the weights and biases for the mixing network that computes Q_{tot} using the individual Q_is as inputs. To have the same greedy policy as VDN, QMIX states that the actions which would maximize the Q_{tot} value is just the combination of the actions which would maximize individual Q_i values:

$$\arg\max_{\mathbf{a}} Q_{\text{tot}}(\mathbf{s}, \mathbf{a}) = \begin{pmatrix} \arg\max_{a_1} Q_1(s_1, a_1) \\ \vdots \\ \arg\max_{a_N} Q_N(s_N, a_N) \end{pmatrix} \tag{13}$$

and this can be ensured by setting up a constraint:

$$\frac{\partial Q_{\text{tot}}}{\partial Q_i} \geq 0, \forall i \in [1, N], \tag{14}$$

i.e. Q_{tot} is monotonically increasing with respect to each Q_i. This is achieved by using absolute activation functions for the hyper-networks so that they can output nonnegative weights for the mixing network.

3.2.3 Counterfactual multi-agent policy gradients

When a group of cooperative agents receives a group reward, it is challenging for the individual agents to know their exact contribution. This is known as the multi-agent credit assignment problem and COMA [6] is an Actor-Critic method

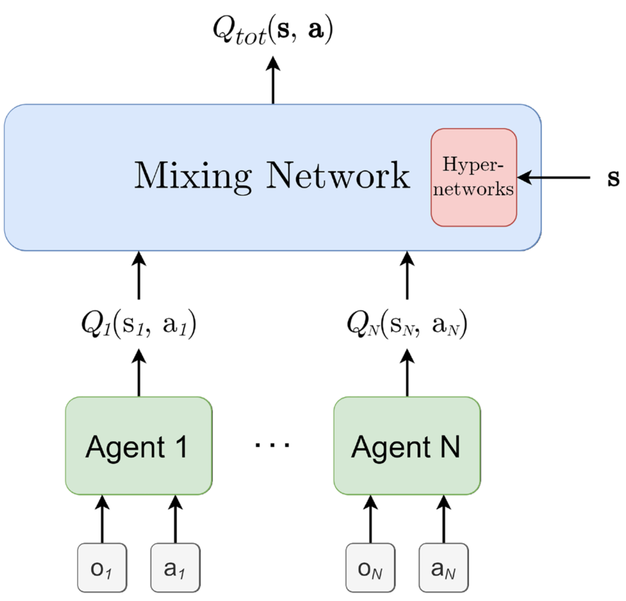

Fig. 1 Network structure for QMIX, graph adapted from [31]

that focuses on it by using a *counterfactual baseline*. It was inspired by the idea of *difference reward* [46]:

$$D_i = r(s, \mathbf{a}) - r(s, (\mathbf{a}^{-i}, c_i)), \tag{15}$$

where $\mathbf{a}^{-i}$ is the joint action of all agents except for the i^{th} agent, and c_i is the default action for the i^{th} agent. This is just the difference of reward when the action taken by the i^{th} agent is replaced by its default action. An action that can make the value of D_i larger would also make the value of $r(s, \mathbf{a})$ larger. However, it is hard to handpick the default action c_i in most scenarios, COMA designed an advantage function to avoid the problem:

$$A_i(s, \mathbf{a}) = Q(s, \mathbf{a}) - \sum_{a_i'} \pi_i(a_i'|\tau_i) Q(s, (\mathbf{a}^{-i}, a_i')), \tag{16}$$

where a_i' is an action different from a_i for the i^{th} agent, and the second term on the right-hand side is the expected Q-value when the agent's current action a_i is marginalized out. This advantage function indicates how good an agent's action is to the whole team compared to its other actions.

3.2.4 Multi-agent deep deterministic policy gradient

MADDPG [20] is the multi-agent version for the algorithm DDPG. It proposes an exceptional CTDE structure, which is widely used nowadays. Unlike the three algorithms mentioned above, where there is only a single value function Q_{tot} or a central critic that considers the joint actions and joint observations or states, MADDPG allocates critics that consider extra information, such as observations and actions

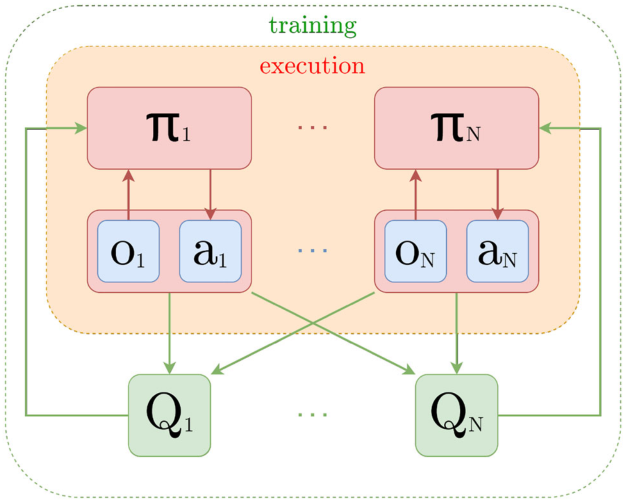

Fig. 2 Network structure for MADDPG, graph adapted from [20]

from other agents, to every single agent, as shown in Fig. 2. Also, each agent has its own policy that only takes its local observations as inputs and generate actions in a deterministic way, which make it capable of dealing with continuous action spaces. Moreover, since each agent learns by maximizing individual return, the environment can be cooperative, competitive or both for MADDPG, which makes it an algorithm with wider range of application.

4 MARL in computer graphics

In this section we review some important research applications for multi-agent reinforcement learning in graphics. Despite being a popular theoretical field for reinforcement learning researchers, we find that MARL is still in its infancy in the graphics community, perhaps due to a lack of clear applications of the framework.

The applications that MARL can be considered most applicable to are crowd simulation, strategy games such as Starcraft II, two-player competitions such as martial arts or fighting, and collective sports such as football. We briefly review how characters have been controlled in such domains and some work where RL/MARL have been applied for solving these problems.

4.1 Crowd simulation

Crowd simulation has been applied to problems such as simulating fire escape scenarios for safety testing [9], large fight simulations for wars in movies [45], simulating the movements of citizens within a city [30] and finding bottlenecks for flow in architecture design [34]. All of these approaches have made use of some interesting heuristics-based navigation that combines local information of an agent's nearby interactions with some global goal information, for example, an agent making it through a target building's exit while avoiding colliding with nearby agents. The results are impressive at scale [45] when a large number of interacting agents result in interesting, emergent behaviours. But the scale of these approaches can hide the fact that these agents are not acting naturally, nor 'intelligently', and any increase in the resolution of the simulation usually shows individual agent movement artefacts.

Chen et al. [4] train a joint collision avoidance model using DRL. The model is trained in a decentralized manner using trajectories produced by ORCA [41]. The model runs much faster than ORCA and runs without any communication between the agents. Their approach assumes the agents have full information about the nearby agents, which is not realistic in real-world applications. To cope with this problem, Fan et al. [5] directly use the sensor information as the state to control mobile robots without colliding with other agents in various complex environments. The system is trained with policy-gradient-based deep reinforcement learning using training data with many agents. Haworth et al. [7] propose a hierarchical controller where the higher level controller plans the footstep patterns towards the goal while the lower level controller computes the PD targets for the full-body character to follow the planned footsteps as accurately as possible. The lower level controller is a task agnostic controller that is trained in a centralized fashion. The innovation of this research is that the characters are controlled by physical simulation based on DeepLoco [27] at the lowest level. Further development can potentially produce interesting effects, such as fighting at the frontlines at war scenes, or physical interactions at bottlenecks such as corridors.

4.2 Self-play in games

Competitive games such as Go, chess, shogi or computer games such as Starcraft have been test-beds for AI systems. One of the keys in such research nowadays is self-play (see Sect. 3.1.3), a scheme to let agents play the game between themselves to improve their skills through exploration. AlphaZero [38] use deep neural networks with Monte Carlo search trees to build an agent and trained it by self-play. Though it consumed tremendous computing resources, the performance was extraordinary on Go, chess and shogi, and it defeated the previous state-of-the-art Stockfish, in chess, and Elmo, in shogi. The success shows great potential in applications of deep reinforcement learning.

Starcraft II, a science fiction real-time strategy video game, is the new milestone for deep reinforcement learning in game playing, due to its large, partially-observable observation space and large action space. Both Sun et al. [39] and Lee et al. [14] demonstrate results in beating the built-in AI in some maps from Starcraft II. Sun et al. use self-play with DDQN and PPO to train the agent policies, while Lee et al. use A3C. They both reduce the action space by using macro actions, which are sequences of unit actions. Also, to decrease the learning complexity for the agent, they both divide the policy into different modules, each handling a category of actions, but with different designs. Vinyals et al. [42] managed to go further, their AlphaStar reached a Grandmaster level, which can beat 99.8% of the human players, including some professional players. They use prioritized fictitious self-play (PFSP) with an RL algorithm similar to A3C, where PFSP picks opponents with probabilities proportional to the win rate against the agent, allowing the agent to compete with the problematic opponents more frequently. Results show that computers have a significant advantage in micromanagement, but they are still inferior to the best professional players in strategies. This gap indicates that there is still a tremendous space for improvement in complicated applications using reinforcement learning.

4.3 Competitive games

For competitive games between two agents, techniques based on game tree expansion [35,37] and reinforcement learning [43] have been proposed. This stream is taken over by DRL-based controllers.

Researchers in OpenAI simulate competitions such as sumo [2] and hide-and-seek [1]; they extend the idea of competitive self-play (see Sect. 4.2) to the domain of continuous control in multi-character games. Bansal et al. [2] design four 1v1 competitive environments with agents using continuous control and use PPO as the reinforcement learning algorithm. When the reward is sparse in a complicated environment, it is hard for the agent to receive the reward; for example as in sumo, if the agent gets a reward only when its opponent is knocked to the ground or pushed out of the ring, the system is hard to train as the agents need to go through a long competition before finding out some actions are useful. To solve this problem, they use an exploration curriculum. In the beginning, the agents would be rewarded for completing relatively simple tasks, such as standing or moving, then gradually reduce the reward for these tasks and focus on the target reward as training goes on. Since self-play is used, it is usual to have two agents have different skill levels during training. To mitigate the effect this would bring, they use opponent sampling. When an agent is training, instead of taking the latest policy as its opponent, it randomly chooses older versions. Experiment results show the importance of the curriculum setup and the choice of sampling strategy, especially for more complex environments.

Baker et al. [1] set up a hide-and-seek environment with two hiders and two seekers, together with some boxes and ramps as tools. They use self-play to train agents with opponents that are at an appropriate level and the CTDE setup with PPO to optimize the policies. Experiment results show that, after millions of steps of training, the agents formed an auto-curricula: the agents move randomly$\rightarrow$ the seekers catch the hiders$\rightarrow$ the hiders hide using boxes$\rightarrow$ the seekers use ramps to climb up$\rightarrow$ the hiders lock up the ramps$\rightarrow$ the seekers climb up the boxes and surf them. Agents have possibilities in learning reasonable actions beyond common knowledge.

4.4 Competitive sports

Competitive sports are a good representation of the complexity of human cooperative and competitive multi-agent coordination. A deep hierarchy of cognition is required to play in a sport, such as low-level subconscious muscle responses all the way up to high-level team planning and coordination. Video game AI in sports requires a large network of state machines [3] that cover the breadth of possible plays, yet the limitation of the system is constrained to how much knowledge of the game the designer puts into the AI. Alternatively, self-learning AI systems for sports could transform the scripted or unnatural sports AI we have in games now into intelligent, surprising and adaptive agents that are as creative to play against as human players. So far, the existing research in the application of MARL to competitive sports games is small but the results are impressive.

Liu et al. [18] demonstrate a self-playing multi-agent system in the game of soccer. They utilize standard actor-critic policy gradients methods and pit agents in 2v2 matches. The selection of agents is from a pool of 32 agents in a population, and agents are randomly sampled from this pool and play a game of soccer to collect experience. They extend beyond the usually population-based sampling for competitive multi-agent environments and also adapt the hyper-parameters of the individual agents, such as the reward coefficients, the discounted return scale and the learning rates, via evolutionary perturbation. They demonstrate some interesting results showing the dynamics of how important various reward components are to the agent over time. Because the agents do not have information about the internal state of other agents in the game, they utilize an recurrent architecture for both actor and critic so that behaviours of opponents and teammates can be incorporated into the decision making. This is one of the first works that presents the idea that MARL algorithms can be used for cooperative and competitive AI for games and simulations.

This work is then extended to a much more difficult scenario where the agents were involved in a physically-based soccer simulation and had to actuate in the environment as a humanoid in order to play the game [19]. The players are first trained to imitate motion capture skills of running and turning. Then, they are trained by RL through mid-level drill training to be able to learn skills such as dribbling, kicking and shooting. Finally, the teams of 2x2 are trained to coordinate by MARL. Results are very impressive as the agents have clear sign of learning simple real-world tactics, both in attack and defence. This work can be extended to a large scale in the future, for example, making the competitions eleven-a-side, or having more modern football rules, such as offside.

5 Discussion

Below, we now provide some discussion on MARL applications that haven't had the attention in the graphics community we believe they should have.

5.1 Variable number of agents

One interesting scenario in multi-agent environments that has not been explored too deeply yet is the variable number of agents. Agents joining or leaving the environment will make other agents' rewards and transition states different after taking the same actions at the same previous state and therefore consequently affect their policies. This can be easily found in training non-playable characters (NPCs) with multi-agent reinforcement learning. For example, in first-person shooting games, NPCs can be killed by the players and re-spawned by the system.

5.2 Heterogeneous versus homogeneous agents

Another interesting topic is heterogeneous agents in multi-agent competitive environments. Homogeneous agents are basically identical, while heterogeneous agents may have different observation ranges, action spaces and learning algorithms. For cooperative environments, the CTDE setup can be used, since it does not require the agents to be homogeneous in theory. However, for competitive environments, self-play would not work since the agents cannot train against themselves. A possible way is to build up a metric system that could evaluate current levels for different agents, to help the agents train against opponents at appropriate levels.

5.3 Future application to computer graphics

Other than all the fascinating applications introduced in the previous section, we suggest another possible MARL application that covers all the elements we discussed: a large-scale war scene. Multiple kinds of soldiers are available for each troop, and they can be immobilized during the war. Agents need to learn to cooperate in order to win the battles. Also, difficulty can be increased by having multiple troops representing different tribes, and the relationship between these tribes can be allies, neutral or rivals. In this scenario, agents need to learn strategies to maximize the interest of their own tribe. Finally, collective sports with a lot of teammates competing with the other team involves a lot of tactics and allocation of players to different locations. Learning such tactics considering the speciality of the players is a very high dimensional and complex problem; handling such problems will not only benefit the game industry but also the sports industries for strategy making.

6 Conclusion

In this paper, we discussed some key ideas in RL and MARL and reviewed some significant algorithms in MARL with CTDE architecture, which are suitable for graphics applications. We also reviewed several exciting MARL applications in the field of computer graphics. MARL is a difficult problem and there is still much research needed to tackle general, real-world problems. This also applies to computer graphics problems as such applications usually have higher complexity. In the foreseeable future, we would like to see more breakthroughs in computer graphics applications with MARL.

Acknowledgements Cheng Li is supported by Microsoft Studios. Levi Fussell is supported by EPSRC DTA program. Taku Komura is partly supported by a start-up fund by The University of Hong Kong (182DRTAKU, 187FRTAKU, 230DRTAKU). Cheng Li and Taku Komura receive research grant from Microsoft.

Declarations

Conflict of interest Levi Fussell declares that he has no conflict of interest.

References

1. Baker, B., Kanitscheider, I., Markov, T.M., Wu, Y., Powell, G., McGrew, B., Mordatch, I.: Emergent tool use from multi-agent autocurricula. In: 8th International Conference on Learning

Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26–30, 2020. OpenReview.net (2020). https://openreview.net/forum?id=SkxpxJBKwS
2. Bansal, T., Pachocki, J., Sidor, S., Sutskever, I., Mordatch, I.: Emergent complexity via multi-agent competition. In: 6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30–May 3, 2018, Conference Track Proceedings. OpenReview.net (2018). https://openreview.net/forum?id=Sy0GnUxCb
3. Brian.: Ai in video games: new promise with red dead redemption 2? https://enlightened-digital.com/ai-in-video-games-new-promise-with-red-dead-redemption-2/. Accessed 01 June 2021
4. Chen, Y.F., Liu, M., Everett, M., How, J.P.: Decentralized non-communicating multiagent collision avoidance with deep reinforcement learning. In: 2017 IEEE International Conference on Robotics and Automation, ICRA 2017, Singapore, Singapore, May 29–June 3, 2017. pp. 285–292. IEEE (2017). https://doi.org/10.1109/ICRA.2017.7989037
5. Fan, T., Long, P., Liu, W., Pan, J.: Distributed multi-robot collision avoidance via deep reinforcement learning for navigation in complex scenarios. Int. J. Robot. Res. **39**(7), 856–892 (2020)
6. Foerster, J.N., Farquhar, G., Afouras, T., Nardelli, N., Whiteson, S.: Counterfactual multi-agent policy gradients. In: McIlraith, S.A., Weinberger, K.Q. (eds.) Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2–7, 2018. pp. 2974–2982. AAAI Press (2018). https://www.aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/17193
7. Haworth, M.B., Berseth, G., Moon, S., Faloutsos, P., Kapadia, M.: Deep integration of physical humanoid control and crowd navigation. In: Guy, S.J., Sueda, S., Karamouzas, I., Zordan, V.B. (eds.) MIG '20: Motion, Interaction and Games, Virtual Event, SC, USA, October 16–18, 2020. pp. 15:1–15:10. ACM (2020). https://doi.org/10.1145/3424636.3426894
8. Heess, N., TB, D., Sriram, S., Lemmon, J., Merel, J., Wayne, G., Tassa, Y., Erez, T., Wang, Z., Eslami, S.M.A., Riedmiller, M.A., Silver, D.: Emergence of locomotion behaviours in rich environments. CoRR (2017). arXiv:1707.02286
9. Helbing, D., Farkas, I., Vicsek, T.: Simulating dynamical features of escape panic. Nature **407**(6803), 487–490 (2000)
10. Hüttenrauch, M., Adrian, S., Neumann, G., et al.: Deep reinforcement learning for swarm systems. J. Mach. Learn. Res. **20**(54), 1–31 (2019)
11. Kaelbling, L.P., Littman, M.L., Moore, A.W.: Reinforcement learning: a survey. J. Artif. Intell. Res. **4**, 237–285 (1996)
12. Kim, M., Hyun, K., Kim, J., Lee, J.: Synchronized multi-character motion editing. ACM Trans. Gr. **28**(3), 1–9 (2009)
13. Kiran, B.R., Sobh, I., Talpaert, V., Mannion, P., Al Sallab, A.A., Yogamani, S., Pérez, P.: Deep reinforcement learning for autonomous driving: a survey. IEEE Trans. Intell. Transp. Syst. (2021)
14. Lee, D., Tang, H., Zhang, J.O., Xu, H., Darrell, T., Abbeel, P.: Modular architecture for starcraft II with deep reinforcement learning. In: Rowe, J.P., Smith, G. (eds.) Proceedings of the Fourteenth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, AIIDE 2018, November 13–17, 2018, Edmonton, Canada. pp. 187–193. AAAI Press (2018). https://aaai.org/ocs/index.php/AIIDE/AIIDE18/paper/view/18084
15. Lee, Y., Wampler, K., Bernstein, G., Popovic, J., Popovic, Z.: Motion fields for interactive character locomotion. ACM Trans. Gr. **29**(6), 138 (2010). https://doi.org/10.1145/1882261.1866160
16. Levine, S., Wang, J.M., Haraux, A., Popović, Z., Koltun, V.: Continuous character control with low-dimensional embeddings. ACM Trans. Gr. **31**(4), 28 (2012)
17. Lillicrap, T.P., Hunt, J.J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., Wierstra, D.: Continuous control with deep reinforcement learning. In: Bengio, Y., LeCun, Y. (eds.) 4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2–4, 2016, Conference Track Proceedings (2016). arXiv:1509.0297
18. Liu, S., Lever, G., Merel, J., Tunyasuvunakool, S., Heess, N., Graepel, T.: Emergent coordination through competition. In: 7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6–9, 2019. OpenReview.net (2019). https://openreview.net/forum?id=BkG8sjR5Km
19. Liu, S., Lever, G., Wang, Z., Merel, J., Eslami, S.M.A., Hennes, D., Czarnecki, W.M., Tassa, Y., Omidshafiei, S., Abdolmaleki, A., Siegel, N.Y., Hasenclever, L., Marris, L., Tunyasuvunakool, S., Song, H.F., Wulfmeier, M., Muller, P., Haarnoja, T., Tracey, B.D., Tuyls, K., Graepel, T., Heess, N.: From motor control to team play in simulated humanoid football. CoRR (2021). arXiv:2105.12196
20. Lowe, R., Wu, Y., Tamar, A., Harb, J., Abbeel, P., Mordatch, I.: Multi-agent actor-critic for mixed cooperative-competitive environments. In: Guyon, I., von Luxburg, U., Bengio, S., Wallach, H.M., Fergus, R., Vishwanathan, S.V.N., Garnett, R. (eds.) Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4–9, 2017, Long Beach, CA, USA. pp. 6379–6390 (2017). https://proceedings.neurips.cc/paper/2017/hash/68a9750337a418a86fe06c1991a1d64c-Abstract.html
21. Min, J., Chai, J.: Motion graphs++ a compact generative model for semantic motion analysis and synthesis. ACM Trans. Gr. **31**(6), 1–12 (2012)
22. Mnih, V., Badia, A.P., Mirza, M., Graves, A., Lillicrap, T.P., Harley, T., Silver, D., Kavukcuoglu, K.: Asynchronous methods for deep reinforcement learning. In: Balcan, M., Weinberger, K.Q. (eds.) Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19–24, 2016. JMLR Workshop and Conference Proceedings, vol. 48, pp. 1928–1937. JMLR.org (2016). http://proceedings.mlr.press/v48/mniha16.html
23. Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., Riedmiller, M.A.: Playing atari with deep reinforcement learning. CoRR (2013). arXiv:1312.5602
24. Peng, X.B., Abbeel, P., Levine, S., van de Panne, M.: Deepmimic: example-guided deep reinforcement learning of physics-based character skills. ACM Trans. Gr. **37**(4), 1–14 (2018)
25. Peng, X.B., Berseth, G., van de Panne, M.: Dynamic terrain traversal skills using reinforcement learning. ACM Trans. Gr. **34**(4), 80:1-80:11 (2015). https://doi.org/10.1145/2766910
26. Peng, X.B., Berseth, G., van de Panne, M.: Terrain-adaptive locomotion skills using deep reinforcement learning. ACM Trans. Gr. **35**(4), 81:1-81:12 (2016). https://doi.org/10.1145/2897824.2925881
27. Peng, X.B., Berseth, G., Yin, K., Van De Panne, M.: Deeploco: dynamic locomotion skills using hierarchical deep reinforcement learning. ACM Trans. Gr. **36**(4), 1–13 (2017)
28. Peng, X.B., Kanazawa, A., Malik, J., Abbeel, P., Levine, S.: Sfv: Reinforcement learning of physical skills from videos. ACM Trans. Gr. **37**(6) (2018)
29. Peng, X.B., Ma, Z., Abbeel, P., Levine, S., Kanazawa, A.: Amp: adversarial motion priors for stylized physics-based character control. ACM Trans. Gr. (2021). https://doi.org/10.1145/3450626.3459670
30. Pettré, J., Ciechomski, P.D.H., Maïm, J., Yersin, B., Laumond, J.P., Thalmann, D.: Real-time navigating crowds: scalable simulation and rendering. Comput. Anim. Virtual Worlds **17**(3–4), 445–455 (2006)
31. Rashid, T., Samvelyan, M., de Witt, C.S., Farquhar, G., Foerster, J.N., Whiteson, S.: QMIX: monotonic value function factorisa-

tion for deep multi-agent reinforcement learning. In: Dy, J.G., Krause, A. (eds.) Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10–15, 2018. Proceedings of Machine Learning Research, vol. 80, pp. 4292–4301. PMLR (2018). http://proceedings.mlr.press/v80/rashid18a.html
32. Reynolds, C.W.: Flocks, herds and schools: a distributed behavioral model. In: Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques, pp. 25–34 (1987)
33. Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O.: Proximal policy optimization algorithms. CoRR (2017). arXiv:1707.06347
34. Shi, X., Ye, Z., Shiwakoti, N., Tang, D., Lin, J.: Examining effect of architectural adjustment on pedestrian crowd flow at bottleneck. Physica A **522**, 350–364 (2019)
35. Shum, H.P., Komura, T., Shiraishi, M., Yamazaki, S.: Interaction patches for multi-character animation. ACM Trans. Gr. **27**(5), 1–8 (2008)
36. Shum, H.P., Komura, T., Yamazaki, S.: Simulating interactions of avatars in high dimensional state space. In: Proceedings of the 2008 Symposium on Interactive 3D Graphics and Games, pp. 131–138 (2008)
37. Shum, H.P., Komura, T., Yamazaki, S.: Simulating multiple character interactions with collaborative and adversarial goals. IEEE Trans. Vis. Comput. Gr. **18**(5), 741–752 (2010)
38. Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., Lillicrap, T., Simonyan, K., Hassabis, D.: A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. Science **362**(6419), 1140–1144 (2018). https://doi.org/10.1126/science.aar6404
39. Sun, P., Sun, X., Han, L., Xiong, J., Wang, Q., Li, B., Zheng, Y., Liu, J., Liu, Y., Liu, H., Zhang, T.: Tstarbots: defeating the cheating level builtin AI in starcraft II in the full game. CoRR (2018). arXiv:1809.07193
40. Sunehag, P., Lever, G., Gruslys, A., Czarnecki, W.M., Zambaldi, V.F., Jaderberg, M., Lanctot, M., Sonnerat, N., Leibo, J.Z., Tuyls, K., Graepel, T.: Value-decomposition networks for cooperative multi-agent learning based on team reward. In: André, E., Koenig, S., Dastani, M., Sukthankar, G. (eds.) Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems, AAMAS 2018, Stockholm, Sweden, July 10-15, 2018. pp. 2085–2087. International Foundation for Autonomous Agents and Multiagent Systems Richland, SC, USA/ACM (2018). http://dl.acm.org/citation.cfm?id=3238080
41. Van Den Berg, J., Snape, J., Guy, S.J., Manocha, D.: Reciprocal collision avoidance with acceleration-velocity obstacles. In: 2011 IEEE International Conference on Robotics and Automation, pp. 3475–3482. IEEE (2011)
42. Vinyals, O., Babuschkin, I., Czarnecki, W.M., Mathieu, M., Dudzik, A., Chung, J., Choi, D.H., Powell, R., Ewalds, T., Georgiev, P., et al.: Grandmaster level in starcraft ii using multi-agent reinforcement learning. Nature **575**(7782), 350–354 (2019)
43. Wampler, K., Andersen, E., Herbst, E., Lee, Y., Popović, Z.: Character animation in two-player adversarial games. ACM Trans. Gr. **29**(3), 1–13 (2010)
44. Watkins, C.J., Dayan, P.: Q-learning. Mac. Learn. **8**(3–4), 279–292 (1992)
45. Wired.: How they created the massive war scenes in the hobbit: Battle of the five armies. https://www.wired.com/2015/01/digital-fx-the-hobbit/. Accessed 01 June 2021
46. Wolpert, D., Tumer, K.: Optimal payoff functions for members of collectives. Adv. Complex Syst. **04**, 355–369 (2002)
47. Won, J., Gopinath, D., Hodgins, J.: Control strategies for physically simulated characters performing two-player competitive sports. ACM Trans. Gr. (2021). https://doi.org/10.1145/3450626.3459761
48. Won, J., Lee, K., Sullivan, C., Hodgins, J.K., Lee, J.: Generating and ranking diverse multi-character interactions. ACM Trans. Gr. **33**(6), 1–12 (2014)

Cheng Li is a PhD student at the University of Edinburgh, supervised by Professor Taku Komura. He received his BSc in Artificial Intelligence and Computer Science from The University of Edinburgh. His interests include multi-agent reinforcement learning, character animation and data-driven character control.

Levi Fussell is a PhD student at the University of Edinburgh, supervised by Professor Taku Komura. His research interests include physically-based character animation, machine learning for video games, and complex systems science.

Taku Komura is a professor in The University of Hong Kong. Before joining HKU 2020, he worked at the University of Edinburgh (2006–2020), City University of Hong Kong (2002–2006) and RIKEN (2000–2002). He received his BSc, MSc and PhD in Information Science from University of Tokyo. His research has focused on data-driven character animation, physically-based character animation, crowd simulation, 3D modelling, cloth animation, anatomy-based modelling and robotics. Recently, his main research interests have been on the application of machine learning techniques for animation synthesis. He received the Royal Society Industry Fellowship (2014) and the Google AR/VR Research Award (2017).