Tunable Dynamics in Agent-Based Simulation using Multi-Objective Reinforcement Learning

Johan Källström
Linköping University
Linköping, Sweden
johan.kallstrom@liu.se

Fredrik Heintz
Linköping University
Linköping, Sweden
fredrik.heintz@liu.se

ABSTRACT

Agent-based simulation is a powerful tool for studying complex systems of interacting agents. To achieve good results, the behavior models used for the agents must be of high quality. Traditionally these models have been handcrafted by domain experts. This is a difficult, expensive and time consuming process. In contrast, reinforcement learning allows agents to learn how to achieve their goals by interacting with the environment. However, after training the behavior of such agents is often static, i.e. it can no longer be affected by a human. This makes it difficult to adapt agent behavior to specific user needs, which may vary among different runs of the simulation. In this paper we address this problem by studying how multi-objective reinforcement learning can be used as a framework for building tunable agents, whose characteristics can be adjusted at runtime to promote adaptiveness and diversity in agent-based simulation. We propose an agent architecture that allows us to adapt popular deep reinforcement learning algorithms to multi-objective environments. We empirically show that our method allows us to train tunable agents that can approximate the policies of multiple species of agents.

KEYWORDS

Modelling for agent based simulation; Reward structures for learning; Learning agent capabilities (agent models, communication, observation)

1 INTRODUCTION

Agent-based simulation can be used for many purposes, including analysis, verification and training. It is a technique that enables simulation of complex systems of autonomous, interacting agents, who may also interact with humans. Popular application domains include social systems, transport systems and ecological systems [5, 13, 20]. By using simulation, complex environments can be created and studied at low cost. It also becomes possible to study scenarios that are not realizable in the real world, e.g. due to safety constraints, for instance simulation of catastrophic events.

An important, but very challenging task in constructing an agent-based simulation is building suitable behavior models for the agents [3, 9, 17]. These models have traditionally been handcrafted by domain experts, which is difficult, expensive and time consuming. Recent advances in reinforcement learning has made it possible to train agents to solve increasingly complicated problems, purely through interaction with the environment. Such agents have achieved impressive performance in classic arcade games [7], the ancient game of Go [16], and complex control tasks [15].

Agents trained through reinforcement learning typically exhibit static behavior at runtime. This may be problematic in some applications of agent-based simulation. In our work we are primarily interested in using agent-based simulation for training humans, by constructing simulations in the form of so called Serious Games. To provide an effective and stimulating training environment it is desirable to adjust the simulation to fit the proficiency and training needs of the student. To create a realistic and interesting simulation, it is also necessary that agents exhibit varied behavior, just like humans would in real-world scenarios.

In this paper we aim to address the need for adaptiveness and diversity in agent-based simulation used for training. Our goal is to train agents that can be tuned at runtime to fit the training needs of a student using a simulation-based training system. Our contributions can be summarized as follows:

- We propose an agent architecture that allows us to adapt existing reinforcement learning algorithms to multi-objective environments and Multi-Objective Reinforcement Learning (MORL) [12]. The proposed architecture makes it possible to train agents that can prioritize among a set of objectives at runtime. The objectives can be related to interactions with the environment or other agents in the system. By adjusting the preferences of such an agent, its behavioral characteristics can be tuned.
- We empirically study the performance of the tunable agent, and show that it has the ability to approximate the policies of multiple species of agents trained with fixed preferences for a set of objectives.

2 RELATED WORK

There are many examples of machine learning, including multi-objective learning, being used for modeling behavior in agent-based simulation. For instance, Bone et al. used reinforcement learning to model how stakeholders with different objectives affect land use change [2], and Rogers et al. used multi-objective optimisation based on genetic algorithms to calibrate the parameters of an agent-based model of a financial market [10]. Within the domain of Serious Games, Sawyer et al. used multi-objective reinforcement learning to adapt game content as to balance between student learning and engagement [14]. Machine learning techniques have also long been used to build game playing agents. For instance, supervised learning and recorded data from Starcraft II matches have been used to build agents that play using similar tactics as humans [6]. Another recent approach used soft Q-learning to constrain an agent's policy to a reference policy, to create an agent whose performance could be adjusted to fit a human opponent [4].

In recent years there has been work on deep methods for multi-objective learning. Mossalam et al. used a single-objective algorithm in combination with a linear scalarization function to calculate the set of optimal policies for convex combinations of the objectives [8]. Roijers et al. presented a policy gradient algorithm for calculating a single, optimal policy for a specific set of user preferences among objectives [11]. Abels et al. proposed to successively learn policies as preferences among objectives are observed, either by constructing a set of policies or by using a single policy [1].

In our work we study applications of multi-objective deep reinforcement learning in agent-based simulation, with a focus on serious games, an area where not much work has been done. In this context we study how a single policy can be pre-trained for a broad range of preference weights, to represent agents with diverse characteristics.

3 BACKGROUND

3.1 Reinforcement Learning

Reinforcement Learning is a Machine Learning technique that allows agents to solve sequential decision making tasks through interaction with the environment. It can be viewed as trial-and-error-learning. When using reinforcement learning, instead of explicitly programming the behavior of agents, a designer or domain expert can focus on specifying the goals of the agents.

The setting of a reinforcement learning problem is often modelled as a Markov Decision Process (MDP). An MDP is a tuple (S, A, T, R, γ), specifying:

- S: The states of the process
- A: The actions of the process
- T: The transition dynamics of the process
- R: The reward received when moving from state s to state s'
- γ: The discount factor indicating the importance of immediate and future rewards respectively

The objective of the agent is to maximize its future expected return

$$E[R] = E[\sum_{t=0}^{\infty} \gamma^t r_t | s_0 = s] \tag{1}$$

One popular algorithm for reinforcement learning is Q-learning. Q-learning is an off-policy, value-based reinforcement learning method that seeks to estimate the state-action value function Q(s,a). While exploring the environment the estimate of Q for a given state is updated according to the update rule

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha(r_t + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)) \tag{2}$$

where α is the desired learning rate.

Traditional Q-learning has difficulties handling a state space of high dimension. Deep Q Networks (DQN) [7] tackle this problem by using a deep neural network to learn an abstract representation of the state space, allowing it to generalize to unseen input. This allows for sample efficient learning in complex domains. To stabilize learning, the network is trained using batches of examples sampled from an experience memory.

3.2 Multi-Objective Reinforcement Learning

Many real world decision making problems deal with multiple, possibly conflicting objectives. While in traditional reinforcement learning all objectives are integrated in one single, scalar reward signal, in multi-objective reinforcement learning each objective is dealt with explicitly [12]. For this purpose, an MDP can be extended with one or more additional objectives, resulting in a Multi-Objective Markov Decision Process (MOMDP). For each time step in the MOMDP, instead of a scalar reward, the agent will observe a vector of rewards, with each item representing one of the objectives, and there will be a vector-valued value function $\mathbf{V}^{\pi}$(s) specifying the expected return for each objective when starting in a given state s and then following policy π. It is possible to define a scalarization function that converts this multi-objective value function to a scalar, representing the overall value of the policy. One option is to use a linear function, giving as output the weighted sum of all values

$$V_{\mathbf{w}}^{\pi}(s) = f(\mathbf{V}^{\pi}(s), \mathbf{w}) = \sum_{n=1}^{n} v_i^{\pi}(s) w_i \tag{3}$$

One downside with the linear scalarization function is that if a deterministic policy is used and the problem has a concave Pareto front all desired policies may not be found [18, 19]. To get better coverage of the solution space, a non-linear scalarization function must then be used.

If the preferences among the objectives, i.e. the desired weights $\mathbf{w}$, are not known at training time, one way of solving the MOMDP is to calculate a set of policies, such that for each choice of weights $\mathbf{w}$ there is an optimal policy in the set. The preferences can then be selected at runtime, allowing for a time dependent prioritization of the objectives. Due to the complexity of the problem, it may not be possible to calculate an exact solution, but instead an approximation must be sought.

4 METHODOLOGY

In this section we present an agent architecture and training scheme that allows us to use standard reinforcement learning algorithms in multi-objective environments. Contrary to the standard reinforcement learning approach, we design our environments in such a way that instead of providing a scalar reward signal in each time step, they provide a vector of events that occurred, with each element in the vector corresponding to one of the objectives. This event vector serves as input to a reward system modelled as part of the agent. Each time step the agent observes which events that occur, and based on its current preferences it calculates a corresponding reward signal using a scalarization function

$$r = f(\mathbf{e}, \mathbf{w}) \tag{4}$$

This scalar is used as the target when training the agent. One consequence of this approach is that different species of agents may have different opinions about which events are desirable and which are not, i.e. diversity among agents is encouraged. Since the scalarization is applied to the rewards directly, we are restricted to using a linear scalarization function [11, 12].

The behavior of the agent is controlled by a policy produced by a learning algorithm. After scalarization, the input to the algorithm is on a format that normal single-objective, value-based as well

as policy-based algorithms can handle. In this paper the policy of the agent is represented by a Deep Q Network (DQN), but other single-objective reinforcement learning algorithms and policy representations could be used. The observed state that we used as input to the learning algorithm is a combination of the environment state and the linear scalarization weights $\mathbf{w}$ corresponding to the agent's preferences among the objectives

$$s_{tot} = [s_1, s_2, ..., s_M, w_1, w_2, ..., w_N] \quad (5)$$

This means that the agent will be trained to select actions according to the current selection of preferences among objectives. The observed state of the environment may be represented by a 1D vector or a multi-dimensional tensor (image or few-hot tensor). In the first case the state observation and the preference vector are simply concatenated and fed through a feedforward network. In the second case the multi-dimensional state representation is processed by a multi-layer convolutional network before being flattened and concatenated with the preference vector, and then fed through a feedforward network. The architecture of the agent is illustrated in Figure 1.

During training the agent must be exposed to a sufficiently large part of the set of possible preferences. The preference weight space is defined by two vectors, $\mathbf{pref}_{high}$ and $\mathbf{pref}_{low}$, specifying the maximum and minimum weight for each objective. At the beginning of each episode the preference weight vector is initialized by sampling from a uniform random distribution according to the specified limits of the weight space for the agent, to promote exploration of the weight space. For each step in the episode, an action is chosen according to the current policy $\pi(s_t, \mathbf{w})$, an event vector $\mathbf{e}_{t+1}$ is observed, and the agent enters a new state s_{t+1}. Examples from interactions with the environment are stored in an experience memory, after calculating a scalar reward using the scalarization function f, together with the current preference weights. Periodically, at the completion of an episode, a batch of examples is sampled from the experience memory, and used to update the DQN. The training scheme is described in Algorithm 1.

Algorithm 1 Training scheme for agent

```
1:  procedure TrainAgent
2:      episodes = 0
3:      while (total.time.steps < max.time.steps) do
4:          obs = env.reset()
5:          pref = preference.space.sample()
6:          done = False
7:          for t = 1 to episode.length do
8:              act = agent.act(obs, pref)
9:              obs.new, e, done = env.step(act)
10:             memory.store(obs, pref, act, f(e, pref), obs.new)
11:             obs = obs.new
12:             if (done) then
13:                 break
14:         episodes = episodes + 1
15:         if (episodes > train.frequency) then
16:             batch = memory.sample()
17:             policy.update(batch)
18:             episodes = 0
```

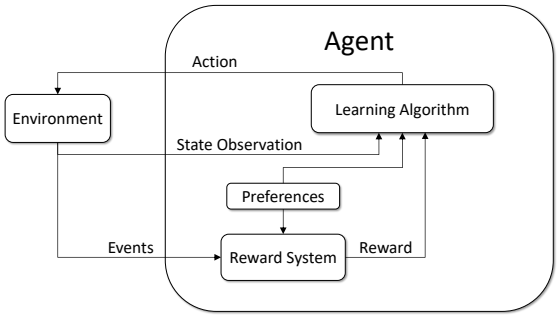

Figure 1: Agent architecture.

5 EXPERIMENTAL EVALUATION

5.1 Simulation Setup

To evaluate the proposed method we conduct experiments in two multi-agent gridworlds. Our intention is to illustrate the basic concepts of the proposed method. The results should generalize to more complex environments, although training times will increase for observation and preference weight spaces of higher dimension.

The size of each environment grid is 8x8. The agent can take the actions Up, Down, Left, Right or Stay. The transition dynamics are deterministic, while parts of the environment contents are randomized for each episode. Agents receive visual observations of the environment consisting of the last three frames as stacked RGB arrays, and the elements of the event vector are either 0 or 1. The agents were configured to use a DQN with two convolutional layers followed by a two layer feedforward network, and a linear scalarization function. The agents are trained with 20M episode steps, a batch size of 32 samples, $\alpha = 10^{-4}$, $\gamma = 0.99$ and the Adam optimizer. The experiments are carried out using 5 runs with different random seeds.

5.2 Experiment 1: Tunable Competitiveness

In this experiment we study how the proposed method can be used to tune agents to act more or less competitively with respect to other agents. Such functionality could be valuable in training and gaming scenarios, to create agents that are suitable for the level of a human opponent.

The environment contains 3 green items, 3 red items and 2 yellow items, which the agent should collect. The positions of the items are randomized at the beginning of each episode, but they always appear in the 16 cells in the center of the grid. In the environment there is also another agent, which has a fixed preference for collecting red items. The tunable agent is trained to be able to prioritize among red, green and yellow items at test time. The agent also has a tunable preference for other agents collecting items. In this way it is possible to tune to which extent this agent should be competitive or cooperative in relation to other agents. In addition to the tunable preferences, there is also a fixed penalty for each step in the environment and for trying to walk outside the grid. An episode ends after 30 steps or when all items preferred by some

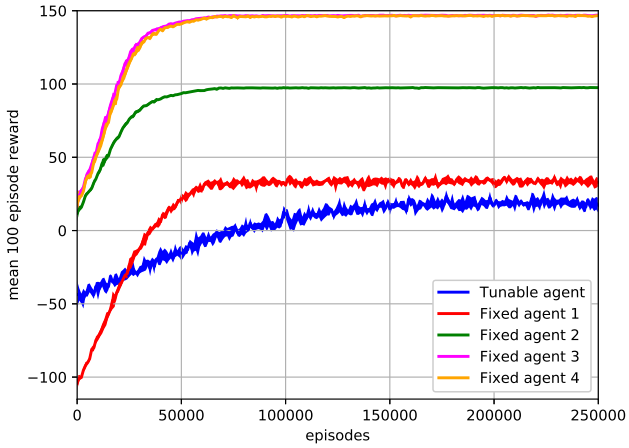

Figure 2: Mean and standard deviation for the training progress in Experiment 1.

agent have been collected. An example of the environment is shown in Figure 4.

For this environment we use a preference weight space defined as ([green item collected, red item collected, yellow item collected, items collected by other agent])

$$pref_{high} = [+20, +20, +20, +20]$$

$$pref_{low} = [-20, -20, -20, -20]$$

The progress of the training of the agent is shown in Figure 2, as mean 100 episode reward per episode. We study this trained agent in four simulation scenarios. The following sets of preferences are used for the objectives:

- Scenario 1, Competitive Agent, **w** = [+10,+20,+10,-20]: Here the agent has a negative preference for other agents collecting items, while red items are valued the most. With these preferences it is expected that the tunable agent will exhibit competitive characteristics, and that the hard-coded agent will struggle to collect items.
- Scenario 2, Cooperative Agent, **w** = [+10,+20,+10,+20]: In contrast to the previous scenario, in this scenario the agent also has a positive preference for other agents collecting items. With these preferences it is expected that the hard-coded agent will have a better chance of collecting items.
- Scenario 3, Fair Agent, **w** = [+20,+15,+20,+20]: Here the agent has a preference for green and yellow items over red ones, and also has a positive preference for other agents collecting items. With these preferences it is expected that the tunable agent will leave items for the other agent to collect.
- Scenario 4, Generous Agent, **w** = [+20,+0,+20,+20]: The agent receives high rewards for collecting green and yellow items, no rewards for red items, and high rewards if other agents collect red items. With these preferences it is expected that the tunable agent will avoid collecting red items, and instead leave them for the other agent.

Each scenario is simulated for 10k episodes. The mean and standard deviation for collected items and number of steps are presented in Table 1. We can see that the qualitative results are roughly as expected and desired. In Scenario 1 the agent prioritizes red items to prevent the other agent from getting them. The other, hard-coded agent still is a little bit better at getting these items. Because the tunable agent is chasing after the other agent it sometimes fails to collect all green and yellow items before the end of the episode. In Scenario 2 the tunable agent will not actively try to prevent the other agent from collecting items, and therefore it has only collected a few red items. In Scenario 3 it would be wise for the tunable agent to leave all red items to the other agent, unless the time penalty for doing so would be too high, and from the results we see that the tunable agent picks up only a few red items. In Scenario 4 the tunable agent has nothing to gain from collecting red items, and we can see that it rarely happens.

5.3 Experiment 2: Tunable Risk Taking

In this experiment we study how the proposed method can be used to tune the risk awareness of agents. Such functionality could be valuable in e.g. traffic simulations.

The environment contains two green items that the agent needs to collect. These items are placed in the two upper corners of the grid. In the center of the grid there is a yellow road segment where scripted red agents, representing cars, are moving vertically. When a car hits a wall or the edge of the grid it changes direction. At the beginning of each episode the initial position and direction, as well as the speed of each car are selected by random. According to the traffic rules the tunable agent is not allowed to pass through the road segment, but can choose to do so if it pleases and if it is willing to risk colliding with a car. The agent must balance rule abidance and the risk of passing the road segment against the time it takes to collect the two items. In addition to the tunable preferences, there is also a fixed penalty for trying to walk outside the grid or into a wall. An episode ends after 50 steps or when both green items have been collected by the agent. An example of the environment is shown in Figure 4.

For this environment we use a preference weight space defined as ([steps, item collected, steps on road, collisions])

$$pref_{high} = [-1, +50, 0, 0]$$

$$pref_{low} = [-10, +50, -20, -50]$$

The progress of the training of the agent is shown in Figure 3, as mean 100 episode reward per episode. We study the behavior of the trained agent in four simulation scenarios. The following sets of preferences are used for the objectives:

- Scenario 1, Always Safe, **w** = [-1,+50,-20,-50]: Here there is a big penalty for passing the road and colliding with cars. This configuration should encourage the agent to take the long route around the road segment, and thus avoid collisions with cars.
- Scenario 2, Always Fast, **w** = [-10,+50,-10,-10]: Here there is a big time penalty, but also penalties for walking on the road segment or colliding with cars. We set the penalty for walking on the road segment to the same as for colliding with cars, so that the agent may deliberately walk in to a car to avoid having to take a costly detour.

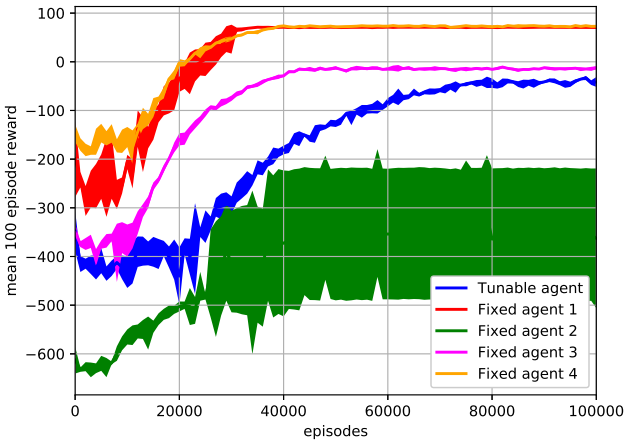

Figure 3: Mean and standard deviation for the training progress in Experiment 2.

- Scenario 3, Fast and Safe, **w** = [-5,+50,0,-50]: Here there is a medium time penalty and a high penalty for colliding with cars. However, there is no penalty for walking on the road segment, so the agent is encouraged to take a shortcut to reach the end of the episode quickly.
- Scenario 4, Slow and Safe, **w** = [-1,+50,0,-50]: Here there is a small time penalty and a high penalty for colliding with cars, but no penalty for walking on the road segment. With this prioritization the agent is expected to try to take a shortcut through the road segment if the traffic conditions are favorable, but otherwise take the long route around the road segment.

Each scenario is simulated for 10k episodes. The mean and standard deviation for total steps, collected objects, steps on the road and collisions are presented in Table 1. We also study the effects of priorities on the agent's selection of route through the environment, and how this affects the likelihood of collisions with other agents. The agent's choice of route through the environment based on the preferences specified for scenarios 1–4 can be seen in Figure 5, displayed as heat maps (brighter areas visited more frequently). We can see that the decisions made by the agent and the resulting routes are roughly as expected. In Scenario 1 the agent always avoids the road segment. In Scenario 2 the agent always takes the shortest route between the two items collected, and therefore collides with cars on several occasions. In Scenario 3 the agent will most of the times try to pass through the center of the road segment, and sometimes along the top of the grid, while maneuvering to avoid collisions. In Scenario 4 the agent will most of the times try to pass through the center of the road segment, while maneuvering to avoid collisions. A few times it has passed along the bottom and top rows of the grid.

5.4 Ability to Approximate Fixed Policies

To study the tunable agent's ability to approximate the policies of different types of agent's with varying characteristics we compared its performance to species of agents trained with fixed preferences

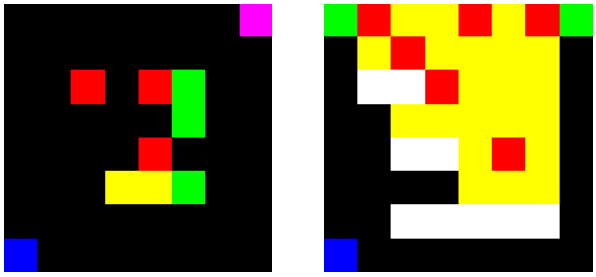

Figure 4: Item gathering environment (left): Tunable agent in blue, agent with hard-coded preferences in pink, and three categories of items in green, red and yellow. Traffic environment (right): Tunable agent in blue, items to collect in green, scripted agents in red, walls in white and road in yellow.

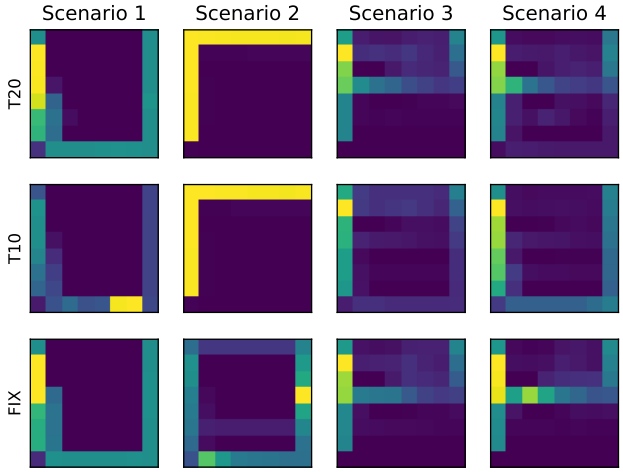

Figure 5: Routes in Experiment 2 for tunable agent trained for 20M steps (T20), tunable agent trained for 10M steps (T10) and agent with fixed preferences (FIX).

for 10M steps. The training progress of these agents is shown in Figure 2 and Figure 3. For further comparison we also trained tunable agents for 10M steps. The studied scenarios are simulated for 10k steps. We conduct this study for the sets of preferences presented for the simulations in Experiment 1 and Experiment 2. The results of the comparison are presented in Table 1. The routes selected by the agents in Experiment 2 are presented in Figure 5.

It can be seen that the tunable agent and the species of fixed agents produce similar results and have the same qualitative behavior, although more variance can be seen for the tunable agent. Scenario 2 in Experiment 2 is an exception. In Scenario 2 agents are encouraged to reach the end of the episode fast, and thus risk colliding with cars. However, it can be seen that the fixed agent often will take the long route around the road segment. When studying this agent's behavior in simulations it can be seen that it makes a bigger

Table 1: Results for experiments 1 (E1) and 2 (E2) for tunable agent trained for 20M (T20) and 10M (T10) steps and for agents trained with fixed preference weights (FIX), presented for scenarios 1-4.

Scn.	Agent	E1-Steps	E1-Green	E1-Red	E1-Yellow	E2-Steps	E2-Objects	E2-Road	E2-Collisions
1	T20	17.5±4.98	2.89±0.46	1.43±0.69	1.92±0.34	28.6±1.17	2.00±0.02	0.02±0.13	0.00±0.07
	T10	17.5±4.92	2.90±0.43	1.42±0.69	1.90±0.38	33.8±8.67	1.82±0.39	0.02±0.18	0.00±0.07
	FIX	14.2±2.06	2.99±0.10	1.46±0.68	2.00±0.07	28.3±0.74	2.00±0.02	0.02±0.13	0.00±0.07
2	T20	13.6±2.23	2.99±0.10	0.69±0.72	2.00±0.08	14.2±0.57	2.00±0.02	6.07±0.32	2.14±1.08
	T10	13.7±2.26	2.99±0.10	0.76±0.74	1.99±0.08	14.2±0.68	2.00±0.03	6.07±0.46	1.98±1.16
	FIX	12.4±1.51	2.99±0.08	0.79±0.68	2.00±0.06	28.7±12.1	1.80±0.40	2.89±2.74	0.86±1.22
3	T20	13.7±2.60	2.99±0.12	0.20±0.43	1.99±0.09	20.9±3.48	2.00±0.02	6.65±1.73	0.44±0.74
	T10	13.7±2.66	2.99±0.12	0.25±0.48	1.99±0.11	21.8±5.51	2.00±0.03	5.02±3.27	0.66±0.91
	FIX	12.7±1.61	3.00±0.08	0.04±0.19	2.00±0.06	21.0±3.19	2.00±0.02	6.55±1.73	0.26±0.52
4	T20	13.8±2.63	2.99±0.13	0.06±0.24	1.99±0.09	25.3±6.49	1.99±0.11	5.80±3.39	0.26±0.64
	T10	13.7±2.41	2.99±0.11	0.09±0.30	1.99±0.09	26.9±4.71	2.00±0.02	1.92±3.04	0.16±0.54
	FIX	12.9±1.74	3.00±0.08	0.02±0.13	2.00±0.06	22.6±3.68	2.00±0.02	7.45±2.83	0.15±0.43

effort than the tunable agents to avoid collisions, and sometimes when the traffic conditions are particularly congested the agent decides to take the long route. This behavior is sub-optimal for gathering reward in this environment, i.e. the agent has got stuck in a local optimum, while the tunable agent trained for 10M steps did not. Figure 3 also shows that the performance of the fixed agent has high variance. For scenarios 2 and 3 in Experiment 2, when the tunable agent is trained for 20M steps instead of 10M steps, its pattern of movement is closer to that of the hard-coded agent.

6 CONCLUSIONS

In this paper we have proposed an agent architecture and training scheme intended for agent-based simulation. The architecture allows us to use standard deep reinforcement learning algorithms in multi-objective environments. The proposed approach can be used to train agents whose behavior can be adjusted at runtime, by specifying the agents' preferences among a set of objectives. Our experiments demonstrate that these tunable agents can approximate the policies of several different species of agents with fixed preferences among objectives. We argue that this functionality is highly valuable for efficient and effective construction of agent-based simulations adapted to user needs, e.g. for application in training systems. The experiments also show that the training time is comparable to that of agents with fixed preferences.

The environments we have studied are relatively simple. For real-world scenarios it could be more challenging to train this type of agent. In future work we would like to study the performance of the proposed method in more complex environments, including environments with continuous action spaces, partial observability, and tasks that require more complex interactions among agents. We would also like to study intelligent exploration strategies, that allow agents to be trained with many objectives and preference spaces with high dimension, and transfer learning to unseen weights. Another interesting topic for future work is development of efficient methods for elicitation of user preferences regarding agent characteristics, to enable construction of simulations that match user needs. Finally, we would also like to study human-agent interaction in e.g. simulation-based training systems.

ACKNOWLEDGMENTS

This work was partially supported by the Swedish Governmental Agency for Innovation Systems (NFFP7/2017-04885), and the Wallenberg Artificial Intelligence, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation.

REFERENCES

[1] Axel Abels, Diederik M. Roijers, Tom Lenaerts, Ann Nowé, and Denis Steckelmacher. 2018. Dynamic Weights in Multi-Objective Deep Reinforcement Learning. (2018). arXiv:cs.LG/1809.07803

[2] Christopher Bone, Suzana Dragicevic, and Roger White. 2011. Modeling-in-the-middle: bridging the gap between agent-based modeling and multi-objective decision-making for land use change. *International Journal of Geographical Information Science* 25, 5 (2011), 717–737.

[3] Chaminda Bulumulla, Lin Padgham, Dhirendra Singh, and Jeffrey Chan. 2017. The Importance of Modelling Realistic Human Behaviour When Planning Evacuation Schedules. In *Proceedings of the 16th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2017)*. São Paulo, Brazil, 446–454.

[4] J. Grau-Moya, F. Leibfried, and H. Bou-Ammar. 2018. Balancing Two-Player Stochastics Games with Soft Q-Learning. In *Proceedings of 27th International Joint Conference on Artificial Intelligence (IJCAI-18)*. Stockholm, Sweden, 268–274.

[5] A. Jindal and S. Rao. 2017. Agent-Based Modeling and Simulation of Mosquito-Borne Disease Transmission. In *Proceedings of the 16th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2017)*. São Paulo, Brazil, 426–435.

[6] N. Justesen and S. Risi. 2017. Learning Macromanagement in StarCraft from Replays using Deep Learning. In *Proceedings of IEEE Conference on Computational Intelligence and Games (CIG 2017)*. New York, USA.

[7] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. 2015. Human-level control through deep reinforcement learning. *Nature* 518, 7540 (26 02 2015), 529–533. http://dx.doi.org/10.1038/nature14236

[8] Hossam Mossalam, Yannis M. Assael, Diederik M. Roijers, and Shimon Whiteson. 2016. Multi-Objective Deep Reinforcement Learning. (2016). arXiv:cs.AI/1610.02707

[9] David V. Pynadath, Heather Rosoff, and Richard S. John. 2016. Semi-Automated Construction of Decision-Theoretic Models of Human Behavior. In *Proceedings of the 15th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2016)*. Singapore, 891–899.

[10] Alex Rogers and Peter Von Tessin. 2004. Multi-objective calibration for agent-based models. In *Proceedings 5th Workshop on Agent-Based Simulation*. Lisbon, Portugal.

[11] Diederik M Roijers, Denis Steckelmacher, and Ann Nowé. 2018. Multi-objective Reinforcement Learning for the Expected Utility of the Return. In *Adaptive Learning Agents (ALA) workshop at AAMAS*, Vol. 18.

[12] Diederik Marijn Roijers, Peter Vamplew, Shimon Whiteson, and Richard Dazeley. 2013. A Survey of Multi-Objective Sequential Decision-Making. *Journal of*

Artificial Intelligence Research 48 (2013), 67–113.

[13] Raquel Rosés, Cristina Kadar, Charlotte Gerritsen, and Chris Rouly. 2018. Agent-Based Simulation of Offender Mobility: Integrating Activity Nodes from Location-Based Social Networks. In *Proceedings of the 17th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2018)*. Stockholm, Sweden, 804–812.

[14] Robert Sawyer, Jonathan Rowe, and James Lester. 2017. Balancing learning and engagement in game-based learning environments with multi-objective reinforcement learning. In *International Conference on Artificial Intelligence in Education*. Springer, 323–334.

[15] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal Policy Optimization Algorithms. *CoRR* abs/1707.06347 (2017). arXiv:1707.06347 http://arxiv.org/abs/1707.06347

[16] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy Lillicrap, Fan Hui, Laurent Sifre, George van den Driessche, Thore Graepel, and Demis Hassabis. 2017. Mastering the game of Go without human knowledge. *Nature* 550 (Oct. 2017), 354–359.

[17] M. Singh, A. Marathe, M. V. Marathe, and S. Swarup. 2018. Behavior Model Calibration for Epidemic Simulations. In *Proceedings of the 17th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2018)*. Stockholm, Sweden, 1640–1648.

[18] Peter Vamplew, Richard Dazeley, Ewan Barker, and Andrei Kelarev. 2009. Constructing stochastic mixture policies for episodic multiobjective reinforcement learning tasks. In *Australasian Joint Conference on Artificial Intelligence*. Springer, 340–349.

[19] Peter Vamplew, John Yearwood, Richard Dazeley, and Adam Berry. 2008. On the limitations of scalarisation for multi-objective reinforcement learning of pareto fronts. In *Australasian Joint Conference on Artificial Intelligence*. Springer, 372–378.

[20] Blake Wulfe, Sunil Chintakindi, Sou-Cheng T. Choi, Rory Hartong-Redden, Anuradha Kodali, and Mykel J. Kochenderfer. 2018. Real-Time Prediction of Intermediate-Horizon Automotive Collision Risk. In *Proceedings of the 17th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2018)*. Stockholm, Sweden, 1087–1096.