Crowd Simulation Via Multi-Agent Reinforcement Learning

Lisa Torrey
St. Lawrence University
ltorrey@stlawu.edu

Abstract

Artificial intelligence is frequently used to control virtual characters in movies and games. When these characters appear in crowds, controlling them is called crowd simulation. In this paper, I suggest that crowd simulation could be accomplished by multi-agent reinforcement learning, a method by which groups of agents can learn to act autonomously in their environment. I present a case study that explores the challenges and benefits of this type of approach and encourages the development of learning techniques for AI in entertainment media.

Introduction

A primary role of artificial intelligence (AI) in entertainment media is to control virtual characters. Some characters are individuals with unique roles, while others are members of large crowds of virtual extras. *Crowd simulation* is the problem of controlling background groups of virtual agents in a believable manner. It arises in several entertainment-media domains, such as computer games and motion pictures.

Rule-based AI has become common in these domains (Johnson and Wiles 2001). In this type of AI, virtual characters follow scripts: if this happens, do that. A recent user study suggests that rule-based agents can be too predictable (Miles and Tashakkori 2010). One potential way to address this issue is to move towards *learning AI*, in which virtual characters adapt their behavior autonomously through machine learning.

Entertainment media has begun to incorporate learning AI in interesting ways. Notable examples come primarily from computer games: genetic algorithms to evolve virtual organisms, reinforcement learning to train virtual companions, and neural networks to train virtual drivers (Johnson and Wiles 2001). However, approaches to crowd simulation have not yet moved in this direction. The most likely reason is that crowd simulation is an inherently multi-agent problem, and is therefore challenging for machine learning.

A range of methods exist for crowd simulation. *Particle modeling* is an approach that models crowds as particles and moves them according to the physics of particle motion (Heigeas et al. 2003). *Strategy-based simulation* has agents follow behavioral rules, which may be probabilistic (Sung, Gleicher, and Chenney 2004). *Cognitive modeling* has agents use planning algorithms to determine the actions necessary to accomplish their goals (Funge, Tu, and Terzopoulos 1999). There is also an approach that imitates the behavior of humans as captured on video (Lerner, Chrysanthou, and Lischinski 2007). These methods cover a wide range of sophisticated AI techniques, but they do not involve agents who learn. To my knowledge, there are currently no crowd-simulation approaches that do.

In a learning approach, a crowd of virtual characters would experiment within their environment. Each character would adapt its behavior in response to rewards it receives from the environment. Over time, each character would develop behaviors that consistently earn high rewards. This approach is an example of *reinforcement learning* (RL).

Single-agent RL is a well-studied method (Sutton and Barto 1998). However, crowd simulation is an inherently multi-agent problem, requiring multi-agent RL (Busoniu, Babuska, and Schutter 2008). Compared to single-agent RL, multi-agent RL poses some additional challenges (Stone and Veloso 2000). To explore these challenges, I present a small case study that applies multi-agent RL to a crowd-simulation problem.

The School Domain

Since this case study requires a simple crowd-simulation domain, I have developed one that I will refer to as the School domain. It represents an environment with classrooms and hallways, and it contains agents who simulate the movement of students through hallways between classes. The desired behavior for agents in this domain is a believable mix of socialization, in which students group together in hallways, and goal-oriented traffic, in which students move towards classrooms.

Figures 1 and 2 show instances of the School domain with six classrooms and one hallway. Agents are represented by colored circles, and they occupy segments of the hallway. Classrooms on either side of the hallway are outlined in black and have doors indicated by gaps.

The domain operates in *episodes*. At the beginning of an episode, agents are placed in random, unique hallway segments and are assigned random goal classrooms. At each step of the episode, each agent observes its environment and

Copyright © 2010, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

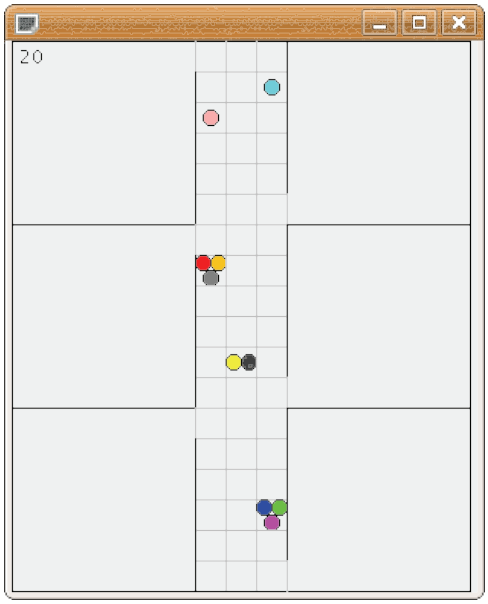

Figure 1: Agents operating in the School domain. In this screen shot, most agents are socializing in the hallway, and there are 20 time steps remaining in the episode.

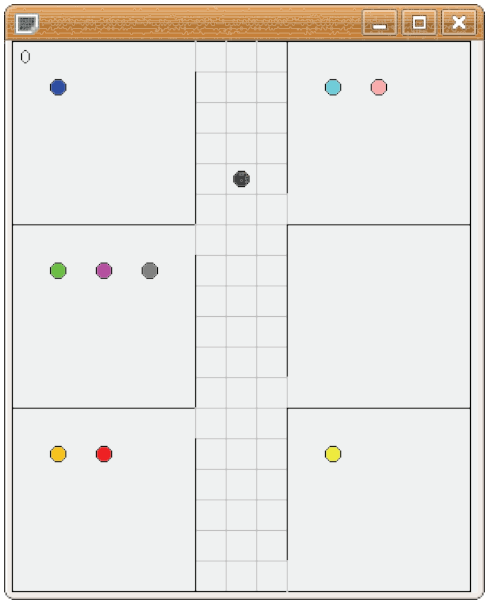

Figure 2: Agents operating in the School domain. In this screen shot, the episode has ended and most agents are in their classrooms, except for one tardy student.

chooses an action to perform. The episode ends after a fixed number of steps. Figure 1 illustrates a good mid-episode configuration and Figure 2 illustrates a good final one.

Agents need to observe their environment at each time step. How much they observe is an important aspect of the domain design. I kept the observations as simple as I could while still providing information that seems necessary for choosing actions. At each time step, each agent observes the following features of its environment:

- The time left in an episode.
- The distance to its goal classroom.
- The distance to the closest other agent.

Agents need to choose an action at each time step. The set of available actions is another important aspect of the domain design. Again, I kept them simple while still allowing for the desired behavior. At each time step, each agent can choose from the following actions:

- Stay in the current segment.
- Move one segment towards the goal classroom.
- Move one segment towards the closest other agent.

There are two additional design details that seemed appropriate in this domain, and that I will mention here for completeness. First, if an agent enters its goal classroom, it remains there regardless of its action choices throughout the rest of the episode. This means that goal states are *absorbing states*, which is a common design decision. Second, if the agent is not in the hallway or is alone in the hallway, the distance to the closest other agent is not well defined. To fix this problem, I define the distance in these cases to be the maximum value in its range.

Note that while this is a simple domain, it is not without potential practical applications. It was inspired by a virtual-reality project to train disabled students to navigate electronic wheelchairs through school hallways (Sonar et al. 2005). Adding virtual students as believable obstacles in the environment would be a crowd simulation problem similar to those found in entertainment media.

Reinforcement Learning

RL is a method by which an agent can learn to act autonomously in its environment. RL algorithms typically perform online learning; agents incrementally update their knowledge as they collect information by trial-and-error. The main challenge in this type of learning is to correctly associate actions with their effects, even though some effects are typically delayed.

Learning with RL often takes place in episodes. At each step of an episode, an agent observes the *state* of its environment and chooses an *action* to perform. It then observes the resulting state and receives a *reward* for its action, and uses this information to update its method of choosing actions, which is called a *policy*. The agent's goal is to learn a policy that maximizes its total episode reward. Figure 3 illustrates this general RL procedure.

One effective type of RL is *Q-learning* (Sutton and Barto 1998). In a Q-learning algorithm, an agent's policy is to take the action that has the highest *Q-value* in the current state. Learning therefore reduces to assigning a Q-value to each state-action pair. A Q-value $Q(s, a)$ estimates the total episode reward achievable by taking action a in state s and continuing to follow the policy thereafter. Typically Q-values begin at zero, and are adjusted after each step:

$$Q(s, a) \longleftarrow r + max_{a'} Q(s', a') \quad (1)$$

where r and s' are the reward and subsequent state after taking action a in state s, and a' ranges over all the actions.

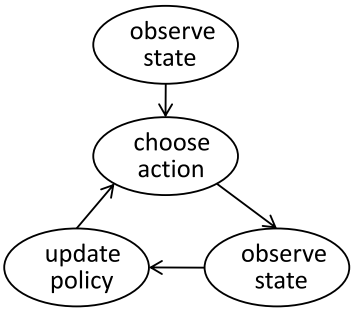

Figure 3: An RL agent observes the initial state of its environment, chooses an action to perform, observes the new state, and uses this information to adjust its policy. Then it chooses another action, and this cycle continues.

Thus the current estimate of a Q-value on the right is used to produce a better estimate on the left, and over time all the Q-values become more accurate.

Under certain conditions, the Q-values are guaranteed to converge to their correct values (Sutton and Barto 1998). One condition is that the environment should not change while the agent is learning. Another is that the agent, if permitted to learn infinitely, would update the Q-value of every possible state-action pair infinitely often. The latter is typically achieved using ϵ-greedy exploration: with probability $\epsilon \in (0, 1)$, an agent chooses a random action instead of the one with the highest Q-value, thus ensuring that every state-action pair has a nonzero probability of occurring.

For this case study, I made two changes to the Q-value update in Equation 1. The first change adjusts it to a popular variant of Q-learning called SARSA, which takes exploration steps into account when updating Q-values (Sutton and Barto 1998). In the SARSA Q-value update, the maximization term in Equation 1 is replaced by the Q-value of the specific action a' that the agent plans to take in state s':

$$Q(s, a) \longleftarrow r + Q(s', a') \tag{2}$$

The second change is to introduce a *learning rate* that controls how quickly Q-values change: $alpha \in (0, 1]$. This new parameter makes the new Q-value into a weighted average between the old estimate and the new one, producing more cautious learning. Using an $\alpha < 1$ is necessary for domains in which the same action may not always have the same effect (Sutton and Barto 1998), which is the case here because there are multiple independent agents. This change turns Equation 2 into the update equation that my RL agents use:

$$Q(s, a) \longleftarrow (1 - \alpha)Q(s, a) + \alpha(r + Q(s', a')) \tag{3}$$

Multi-Agent RL in the School Domain

This section describes how I apply Equation 3 in the School domain. Agents have three observations and three action choices, as described previously. Each possible setting of the three observations corresponds to a state s in the RL algorithm, and each action choice corresponds to an action a.

The reward r, however, poses a challenge in this domain. Designing the reward function is a key problem in any RL domain; since rewards are the primary force driving agent behavior, it is important to reward the desired behavior. In the School domain, the desired behavior characteristic is believability. How can one measure believability in order to reward it? One possibility would be to bring a human into the loop to provide feedback. That approach would be interesting, but it also seems work-intensive and inconvenient.

Instead, I designed the agents in the School domain to construct their own rewards according to internal motivations. Agents have two motivations: to socialize with other agents, and to reach their goal classroom by the end of the episode. They construct rewards for themselves based on whether these events occur. An agent may give itself a goal reward once during an episode, and it may give itself a socialization reward at any time step. Note that it would be possible to create agents with different motivations simply by specifying different intrinsic rewards.

Another challenge in applying the RL algorithm to the School domain involves the number of possible states s. Suppose the length of an episode is 25 and the maximum distance across the domain is 20, as in Figures 1 and 2. Then the number of possible states is $25 \times 20 \times 20 = 10,000$. Since an agent must visit many states many times to learn accurate Q-values, this state space is unpleasantly large, especially for such a small domain.

One way of reducing the size of a state space is to use *tile coding* (Sutton and Barto 1998) to divide observations into intervals. For example, instead of observing the exact time left, an agent could observe that the time is in interval 1 (between 1 and 5 steps left), interval 2 (between 6 and 10 steps left), and so on. Interval 0 must represent exactly 0, since precision there is necessary for reward calculations. If all the features of the environment are tiled this way, the number of possible states becomes $6 \times 5 \times 5 = 150$. The lower precision introduced by tiling may slightly decrease the maximum possible reward, but it will allow learning to progress much more quickly.

A final challenge in the School domain is the application of the single-agent RL algorithm to a multi-agent environment. One straightforward approach is simply to have each agent execute this algorithm independently. However, recall that one of the convergence conditions for Q-learning is that the environment should not change while the agent is learning. Having multiple agents learn in the same environment violates this condition, because changes in each agent's behavior produce changes in the environment for all the other agents.

Violations like this are surprisingly common in successful RL applications. In particular, independent multi-agent Q-learning can sometimes be effective in practice even though theoretical guarantees are lost (Tesauro and Kephart 2002). There is no guarantee that this will be the case here, but it seems worthwhile to try the straightforward approach first, and I do so in this case study.

Experimental Results

This section presents results of experiments in the School domain using the hallway layout of Figures 1 and 2. I set the episode length to 25, which is slightly more than the maximum distance agents might need to travel to their goal. For all agents, I set the goal reward to 30 and the socialization reward per time step to 1, so that reaching the goal could always provide slightly more reward in an episode than constant socializing.

Learning Agents

RL results are typically presented as *learning curves*, which plot the average episode reward against the number of training episodes. To generate these curves, I had the RL agents learn for 10 episodes at a time, and then perform 100 non-learning episodes to estimate their average episode reward at that point. I repeated these stages until the agents had learned for a total of 1000 episodes.

I conducted experiments with $1, 2, 3, 5$, and 10 agents. Since certain parameter settings can be crucial to the speed and success of RL algorithms, I tried a range of learning rates α and exploration rates ϵ. The α settings were $0.01, 0.05, 0.1$, and 0.2 and theϵ settings were $0.1, 0.2, 0.3$, and 0.4. These parameters are often set to decay over time, but for simplicity, I did not do so here.

Since one RL learning curve can vary significantly from another even with the same parameters, I generated and averaged together 10 curves for each set of parameters. This produced a total of 800 learning curves, which took approximately 20 minutes to generate on a single processor.

With only one agent, the challenges of multi-agent RL were absent. At all parameter settings, this agent quickly maximized its rewards. The more interesting cases were those with $2, 3, 5$, or 10 agents. Learning progressed more slowly with more agents, but at their best parameter settings, they all achieved consistently high rewards within 1000 episodes of training.

The α and ϵ parameters had significant effects on both the speed of learning and the rewards achieved by the final policy. The best parameter setting was consistently $\alpha = 0.1$, $\epsilon = 0.4$. Figures 4 and 5 show the effects of varying them in the 10-agent experiments; these results are representative of the $2, 3$, and 5-agent experiments as well.

In this domain, of course, the qualitative appearance of the agents is more important than their quantitative rewards. However, I found that the quantitative performance of an agent was somewhat indicative of its believability. Agents receiving low rewards (below 30) did so because they often did not reach their goal classrooms. Agents receiving high rewards (above 30) did so because they usually did reach their goal classrooms and also socialized on the way.

Agents with high-reward policies typically spent the beginning of an episode socializing with other agents nearby, and moved to their goal classrooms as the episode came to an end, as in Figure 2. In fact, this figure contains actual screenshots of RL agents using a high-reward policy.

Agents with low-reward policies tended to have two undesirable characteristics. First, they often ignored their goal classrooms in favor of the more immediate rewards of socializing. Second, they tended to oscillate between two adjacent hallway segments, as if comically indecisive about where to go.

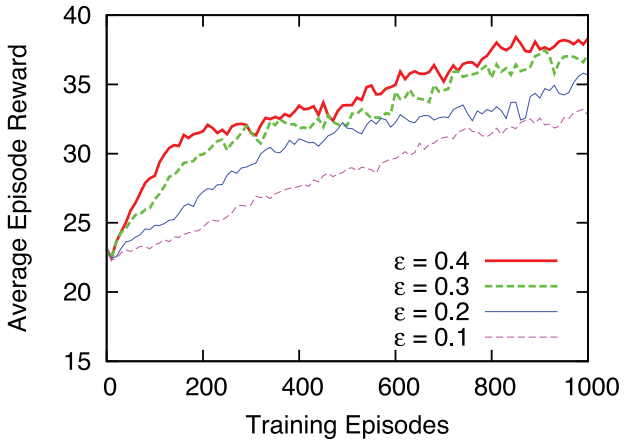

Figure 4: These learning curves show how the average episode reward earned by 10 agents increases as they train. Each curve represents the average of 10 experiments using a learning rate $\alpha = 0.1$ and an exploration rate ϵ as shown.

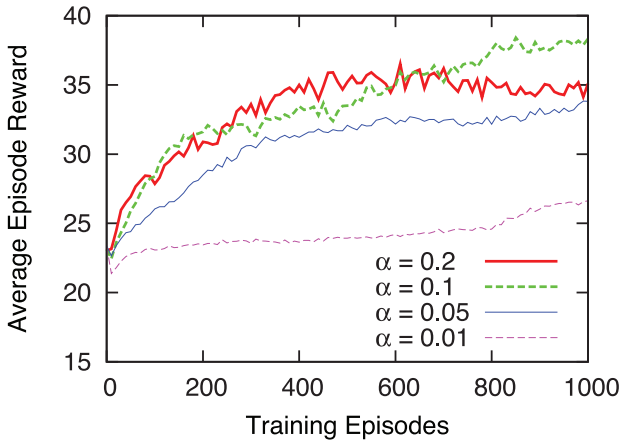

Figure 5: These learning curves show how the average episode reward earned by 10 agents increases as they train. Each curve represents the average of 10 experiments using an exploration rate $\epsilon = 0.4$ and a learning rate α as shown.

Comparison with Rule-Based Agents

These results show that multi-agent reinforcement learning is capable of producing reasonable crowd simulation in the School domain. But how does this approach compare to a non-learning one? To address that question, I also developed agents who follow rules to choose their actions. Based on the behavior of the RL agents, believable behavior in this domain can be described by two simple rules:

1. If the time left is less than or equal to the goal distance, move towards the goal.
2. Otherwise, move towards the closest agent.

Rule-based agents make the same observations that RL agents do, choose from the same actions, and give themselves the same rewards (though the rewards do not affect their behavior). They exhibit behavior similar to the RL agents; in fact, they earn higher rewards. After 1000 episodes of training with $\alpha = 0.1$ and $\epsilon = 0.4$, the ten RL agents earn an average reward of 38 per episode. Ten agents following the rules above earn an average reward of 46 per episode.

The reason for this difference is that rule-based agents always reach their goal. RL agents occasionally fail to do so, as in Figure 2. When they do fail, it is typically a near miss: one or two agents are late by one or two time steps. In these cases, the late agents have not learned an optimal policy because of the chaotic impact of the multi-agent environment.

Recall, however, that qualitative appearance matters more in this domain than quantitative rewards. The real question is: do the higher rewards of rule-based agents produce more believable behavior?

Believable behavior should achieve high rewards, but it should not necessarily be optimal. Real students do not always make it to class on time, as the rule-based agents do. After observing these agents for a while, one begins to see their mechanical, predictable nature. The RL agents, on the other hand, are more lifelike in their slight and systematic sub-optimality. Rule-based agents could be given this quality by putting some design effort into creating probabilistic rules, but RL agents acquire it naturally.

Believable behavior is also diverse. Some students are more social while others are more conscientious; some tend to be late more than others. Again, the RL agents capture this quality without any explicit design, because each agent learns a different policy. They do so despite having the same reward structure, because they influence each others' learning processes. After observing the same 10 agents for a while, one begins to recognize certain agents by their behavior, almost as if they have personalities. Rule-based agents would need different sets of rules to have this quality.

I discovered another difference between the approaches by accident. When I first evaluated the rule-based agents, they did not behave as expected; they alternately moved together and apart. The reason for this behavior was a bug that caused more than two agents in the same location to overlook each other as they searched for nearby agents. The RL agents learned to perform effectively despite the bug, by accepting it as part of their environment and learning around it. The rule-based agents did not have this flexibility.

Challenges

This case study highlights several challenges in applying multi-agent RL to crowd simulation. It also suggests additional challenges that might appear only in a more complex domain. This section explores both types and suggests ways to address them.

Making Design Decisions

Any control problem contains classic design challenges: how should one choose the state description and actions for the agents? Since these decisions are also necessary for rule-based agents, they may not be an additional barrier to the development of learning agents. In fact, designing these elements with rule-based agents in mind is likely to be a successful strategy.

However, there are additional design decisions that are particular to learning agents: choosing a learning algorithm and setting its parameters. Even for single-agent RL there are many algorithms to choose from. While standard implementations of many machine-learning algorithms are available, I am not aware of any comparable RL packages. Furthermore, parameter settings can have a significant impact on agent performance. This may be considered a challenge for the RL community, where theoretical work in classic domains is predominant; we should also address practical design issues in realistic domains.

Improving Multi-Agent RL

The challenges of multi-agent RL arise even in my small case study. By learning independently and simultaneously, some agents do not converge to good policies. They perform well enough to achieve believability in the School domain because the task is simple. However, it is easy to imagine the results becoming unacceptable in a more difficult learning task.

Here we can turn to existing research, because there are more sophisticated approaches to multi-agent RL that could be applied to improve performance. For example, the WoLF algorithm (Bowling and Veloso 2001) has agents adapt their learning rate α in response to their performance. When performing well, they change their policies slowly; when performing poorly, they allow faster changes to learn more quickly. A related approach is Multiple-Timescale Learning (Leslie and Collins 2002), in which some agents are permanently assigned different learning rates than others. This diversity can provide more stability in the overall learning process. A third option is the AWESOME algorithm (Conitzer and Sandholm 2006), in which agents alternate learning, approximating the single-agent case in which the environment is stationary.

All of these algorithms can guarantee convergence to optimal policies in certain limited types of RL domains, such as two-player games with Nash equilibria. Those guarantees may not extend to domains like crowd simulation. However, approaches like these are still likely to provide improvement in performance, and in domains where optimality is not the goal, this could be just what we want.

Scaling Up

My case study involves a relatively small number of agents in a relatively small environment. An important lesson of RL research is that techniques that succeed under these conditions are not guaranteed to scale up. In crowd simulation domains, complexity can increase along three dimensions: the number of agents, the number of actions available to them, and the number of states they can be in.

There are existing ways to address scaling challenges in RL. Tile coding is one that I have used here. State and action spaces can be further reduced by abstracting

away details or by putting them into hierarchies (Dietterich 2000). An approach that may be particularly applicable in the multi-agent setting is *transfer learning*, the use of knowledge gained while learning one task to facilitate learning in another task (Taylor and Stone 2009; Torrey 2009). To reduce the difficulty of training large numbers of agents, one could train a smaller group first and then allow them to adapt to larger crowds.

Managing Predictability

Unpredictable agents can be a source for concern. With rule-based AI, designers have substantial control over their agents' actions. With learning AI, the agents are more autonomous and may act in ways the designers did not expect. While unpredictability can contribute to believability, it is reasonable for designers to want an understanding of their agents.

One way to maintain more control over learning agents may be to use dynamic scripting, a hybrid method that combines rules and RL (Spronck et al. 2006). Within the pure RL framework, one way to make policies more interpretable may be to extract human-readable rules (Torrey 2009).

Whichever approach they use, designers will always need to ask: are there any conditions under which the agents will do something terrible? This question applies to rule-based agents as well as learning agents. Extensive testing of computer games is necessary for this very reason, and it would still be necessary with learning agents.

Conclusion

In this paper, I suggest that multi-agent reinforcement learning could be applied to the problem of crowd simulation. There are many practical challenges involved, but there are also many potential benefits.

Learning can produce more organic behavior that is less predictable and more diverse. Furthermore, learning can provide more resilience to the bugs that are inevitable in any large software project. The initial design effort for learning agents is not likely to be less than for rule-based agents. However, the effort required for future designs may be less, providing benefits for related projects, or even for a single project in which development is a moving target.

I believe that it is worthwhile to work towards incorporating learning techniques into entertainment media. For both the entertainment industry and the machine-learning research community, there are benefits to increasing the sophistication of AI in entertainment. Having impressive AI is a major selling point for entertainment media, and the industry could play a larger role in driving machine-learning research.

References

Bowling, M., and Veloso, M. 2001. Rational and convergent learning in stochastic games. In *The International Joint Conference on Artificial Intelligence (IJCAI'01)*.

Busoniu, L.; Babuska, R.; and Schutter, B. D. 2008. A comprehensive survey of multi-agent reinforcement learning. *IEEE Transactions on Systems, Man, and Cybernetics* 38(2):156–172.

Conitzer, V., and Sandholm, T. 2006. AWESOME: A general multiagent learning algorithm that converges in self-play and learns a best response against stationary opponents. In *The International Conference on Machine Learning (ICML'06)*.

Dietterich, T. 2000. State abstraction in MAXQ hierarchical reinforcement learning. In *Advances in Neural Information Processing Systems (NIPS'00)*.

Funge, J.; Tu, X.; and Terzopoulos, D. 1999. Cognitive modeling: knowledge, reasoning and planning for intelligent characters. In *The International Conference and Exhibition on Computer Graphics and Interactive Techniques (SIGGRAPH'99)*.

Heigeas, L.; Luciani, A.; Thollot, J.; and Castagne, N. 2003. A physically-based particle model of emergent crowd behaviors. In *The International Conference on Computer Graphics and Vision (GRAPHICON'03)*.

Johnson, D., and Wiles, J. 2001. Computer games with intelligence. In *The Australian Journal of Intelligent Information Processing Systems*.

Lerner, A.; Chrysanthou, Y.; and Lischinski, D. 2007. Crowds by example. In *The Conference of the European Association for Computer Graphics (EUROGRAPH'07)*.

Leslie, D., and Collins, E. 2002. Multiple-timescale Q-learning. In *The NIPS Workshop on Multi-Agent Learning*.

Miles, J., and Tashakkori, R. 2010. Improving believability of simulated characters. *Journal of Computing Science in Colleges* 25(3):32–39.

Sonar, A.; Burdick, K.; Begin, R.; Resch, E.; Thompson, E.; Thacher, E.; Searleman, J.; Fulk, G.; and Carroll, J. 2005. Development of a virtual reality-based power wheel chair simulator. In *The International Conference on Mechatronics and Automation*.

Spronck, P.; Ponsen, M.; Sprinkhuizen-Kuyper, I.; and Postma, E. 2006. Adaptive game AI with dynamic scripting. *Machine Learning* 63(3):217–248.

Stone, P., and Veloso, M. 2000. Multiagent systems: a survey from a machine learning perspective. *Autonomous Robots* 8(3):345–383.

Sung, M.; Gleicher, M.; and Chenney, S. 2004. Scalable behaviors for crowd simulation. *Computer Graphics Forum* 23(3):519–528.

Sutton, R., and Barto, A. 1998. *Reinforcement Learning: An Introduction*. Cambridge, Massachusetts: MIT Press.

Taylor, M., and Stone, P. 2009. Transfer learning for reinforcement learning domains: a survey. *Journal of Machine Learning Research* 10(1):1633–1685.

Tesauro, G., and Kephart, J. 2002. Pricing in agent economies using multi-agent Q-learning. In *The International Conference on Autonomous Agents and Multiagent Systems (AAMAS'02)*.

Torrey, L. 2009. *Relational transfer in reinforcement learning*. Ph.D. Dissertation, University of Wisconsin.