ONLINE MARKOV DECISION PROCESSES FOR LEARNING MOVEMENT IN GAMES

Aaron Arvey
Department of Computer Science
Claremont McKenna College
Claremont, California 91711
Email: aarvey@cs.hmc.edu

Eric Aaron
Department of Computer Science
Wesleyan University
Middletown, CT 06459
Email: eaaron@wesleyan.edu

KEYWORDS
Reinforcement learning, Markov decision process, online learning, agent navigation.

ABSTRACT

This paper describes an approach to online learning of navigation intelligence by non-player characters in video games. Using a reinforcement learning framework, specifically Markov decision processes, a non-player character learns navigation strategy by imitating movements of the human player against which it plays. The paper presents three experimental conditions in which our method was applied to a one-on-one dodgeball game, and in two of those conditions, the computer controlled non-player character successfully learned to dodge a player's attacks. The paper also discusses possible extensions of this work to improve agent autonomy and extend online learning beyond navigation strategy.

INTRODUCTION

To provide immersive game experiences for human players competing against computer-controlled *non-player characters* (*NPCs*, for short), NPCs might adapt to the skills and tendencies of the players, even learning from players' expertise. For example, consider a simple one-on-one dodgeball scenario, constructed as a first-person shooter game: Each of two opposing entities (players or NPCs) tries to throw balls to hit its opponent; simultaneously, each entity dodges the balls thrown by its opponent, to avoid being hit. In this context, NPCs might learn strategies for navigating in their domain from observing and imitating the human player's navigation. In particular, a NPC that begins play with only a simplistic framework for intelligence might learn to perform intelligently, perhaps even learning to dodge the balls thrown at it simply by observing the player doing so.

This paper presents an approach to this kind of imitative learning for NPCs in video games. NPCs are trained using a reinforcement learning framework —a *Markov decision process* (*MDP*), in particular— based on observations of human player behavior. Training observations need not occur in an artificial context outside of actual game-play; -instead, our approach trains NPCs online, during play. Indeed, the motivation and purpose of our approach is to learn from human players' actions in actual game conditions.

In this paper we focus on NPC-learning of navigation strategies in a dodgeball game. For simplicity, we consider a grid decomposition of the dodgeball game terrain, and we frame navigation as the straightforward movement from one grid element to another. For instance, a navigation move is simply movement from one grid element to an adjacent one. Our learning approach observes state transitions during game play, where a state is determined by the particular grid elements occupied by the player and the NPC, along with the presence or absence of a ball near the NPC. Our process of NPC-learning can thus be considered to have four components:

1) *Observe* human actions during a training phase. In this phase, a NPC is guided by a simple finite-state machine intelligence, just to provide some foundation on which to base learning.
2) Build a *frequency matrix* from observations, i.e., a table that records how often particular state transitions occur as results of NPC and player actions during game play. In our approach, this step is performed concurrently with the observations in step 1), above.
3) Compute the expected values of navigation moves, based on the frequency matrix. Expected values are computed using the technique of *value iteration*, described more fully in the definitions and review section.
4) Based on the computed expected values, determine a *policy* —a mapping of states to the actions to be performed in those states— for the NPC to carry out during game play.

The NPC then navigates during play by following the computed policy, which is fundamentally based on its observations of player navigation behavior.

In this paper, we present background definitions of MDPs and other related concepts; we then discuss the four above elements of our approach to NPC-learning, leading to experimental results describing how a non-player character in a one-on-one dodgeball game autonomously learned to dodge opponents' attacks. We conclude with a discussion of how our approach could be improved for more effective online learning applications.

RELATED WORK

Video games have been described as the "killer application" of artificial intelligence (Laird and van Lent 2000), and they may prove to be a "killer application" of machine learning as well. In the specific domain of learning navigation, Thurau et al. (2003) showed that a human player's style of movement may be feasibly learned. This was shown through a pattern recognition approach using self organizing maps and multilayer perceptrons based on data gathered from network tournaments of Quake II, an open source video game. Self organizing maps and neural network algorithms, however, appear to be too computationally expensive for online or real-time applications.

Online and real-time reinforcement learning work has been directed mostly towards deterministic methods such as Q learning, the complexity of which is discussed in (Koenig and Simmons 1993). Nondeterministic, pseudo-online and pseudo-real-time approaches to reinforcement learning, such as MDPs, are discussed in (Barto et al. 1995; Bradtke 1994). Such methods are not currently employed for research in real-time environments due to complexity constraints imposed by many non-deterministic environments.

In addition, imitation learning approaches such as (Horman and Kaminka 2004; Alissandrakis et al. 2000; Wood 2004) have been employed to train agents by imitating goal based behaviors. Of particular interest is (Wood 2004), which describes attempts to have agents learn a hierarchical action abstraction model; this is similar to our goal of agents learning a transition model based only on the agents' situated observations.

DEFINITIONS AND REVIEW

In this section, we present MDPs and surrounding concepts, which are central to our NPC-learning process. *A. Markov Decision Process Structure* Given an environment in which an agent will learn, a *Markov decision process* is a 4-tuple (***S***, ***A***, *T*, *R*), where

- ***S*** is a set of states that an agent may be in. S is often derived in part from environmental features, e.g., the grid used to define states in our dodgeball example.
- ***A*** is a set of actions that can be performed by an agent.
- $T : \mathbf{S} \times \mathbf{A} \times \mathbf{S} \rightarrow [0, 1] \subseteq \Re$ is a *transition model*. *T(*s, a, *s')* is the probability that if an agent performs action $a \in \mathbf{A}$ when in state $s \in \mathbf{S}$, it will transition into state $s' \in \mathbf{S}$. *T* is constrained such that $\sum T(s, a, s') = 1$ for any $s \in \mathbf{S}, a \in \mathbf{A}$
- $R : \mathbf{S} \times \mathbf{A} \rightarrow \Re$, where *R(s, a)* is a reward given to the agent for taking action a when in state s. The reward function R represents the "reinforcement" in reinforcement learning.

We also discuss several elements related to MDPs, including the concepts of *policy*, *value function*, and *discount factor*. A *policy* $\pi : \mathbf{S} \rightarrow \mathbf{A}$ assigns an action $a \in \mathbf{A}$ to every state $s \in \mathbf{S}$. The notation π^* is used to refer to the policy that optimizes expected rewards. We discuss how we compute π^* in the next section. A *value function* $V : \mathbf{S} \rightarrow \Re$ based on a policy π is denoted $V^\pi(s)$. The value $V^\pi(s)$ is the expected value that an agent will receive if it follows the policy π in state *s*. A value function $V^\pi(s)$ is computed for each state *s* by

$$V^\pi(s) = R(s,\pi(s)) + \gamma \sum T(s'|s,\pi(s))V^\pi(s')$$

```
$V_0(s) \leftarrow 0 \quad \forall s \in \mathcal{S}$
$V_1(s) \leftarrow R(s,a) \quad \forall s \in \mathcal{S} \quad \forall a \in \mathcal{A}$
$t \leftarrow 1$
while $\exists s \in S$ such that $|V_t(s) - V_{t-1}(s)| > \epsilon$
    forall $s \in \mathcal{S}$
        forall $a \in \mathcal{A}$
            $Q(s,a) \leftarrow R(s,a) + \gamma \sum_{s' \in S} T(s,a,s')V_{t-1}(s'$
        $V_t(s) \leftarrow \max_a Q(s,a)$
        $\pi(s) \leftarrow \arg\max_a Q(s,a)$
    $t \leftarrow t+1$
```

Figure 1. The value iteration algorithm. We employ value iteration to compute (approximate) the value function associated with optimal policy π^*. In this presentation, notation argmax_a *Q(s, a)* refers to an action *a* that maximizes the value of *Q(s, a)*.

where γ is a *discount factor* constant in (0,1]. The discount factor enables a learning method to prefer more immediate rewards over delayed rewards to varying degrees.

Policy Computation for Markov Decision Processes

An optimal policy π^* is defined as the policy which achieves the maximum expected reward as specified by reward function *R*. The value function associated with policy π^* is abbreviated as V^*, and it can be computed (approximated) by *value iteration* (see Figure 1). Value iteration starts by assigning the reward of a state as the value of the state. It then iterates through all possible actions and states. After each iteration, the value of a state has taken into account the value of surrounding states which are progressively further away (i.e., requiring more actions to reach). As the algorithm iterates, the value of each state and the overall policy are updated until they converge to the optimal solution. This algorithm is guaranteed to converge (Bartlette 2003) such that

$$\max_{s \in S} |V_t^\pi(s) - V^*(s)| < 2\varepsilon \frac{\gamma}{1-\gamma}$$

For further clarification on Markov decision processes and corresponding algorithms, see (Kaelbling et al. 1998).

The transition model *T* underlying a MDP may be described as a collection of *Markov chains*, state-transition processes in which the successor state depends solely on the current state. (This contrasts with general probabilistic processes, which may consider all previous states when determining the probability of a successor state.) In a MDP, there exists a corresponding Markov chain for every action *a*, representing the probability of moving from the current state to any other state when action *a* is taken.

Since we are looking at *infinite horizon* problems with theoretically unending executions, value iteration depends on the transition model having several properties (Bartlette 2003). All states must *communicate*, i.e., any state $s \in \mathbf{S}$ must be able to reach any other state $s' \in \mathbf{S}$ in a finite number of actions. Additionally, it must be the case that there is a non-zero probability to return to the current state. A Markov chain with both of these properties is *ergodic*; we refer later to ergodicity, when discussing the construction of the transition model in our MDP. For further theory and applications concerning ergodic Markov chains, see (Bartlette 2003; Ross 2002; Seneta 1981).

METHODS

As mentioned in the previous section, the four parameters to a MDP are a state space ***S***, an action space ***A***, a reward function *R*, and a transition model *T*. In this section, we describe these components in the context of our NPC-learning approach.

Actions, States, and Rewards

In our navigation-learning experiments, the state space, action space, and rewards are built on simple representations and abstractions. The state space is based on the human player's and non-player character's locations in a predetermined grid decomposition of the terrain on which the game is played. In any state, there are four navigation actions for an agent: movement in one of the four cardinal directions. The reward for the agent on any given action is a function of the current state and the distance to the human player; rewards are increased when a NPC maintains a constant, safe distance from the player, and rewards are decreased when a ball is in the state occupied by the NPC.

Transition Model

To clarify how a transition model is used, imagine a robot moving in a grid world, as described by (Russell and Norvig 2003), that will move in one of four cardinal directions altering its position state. Assuming the robot has a semi-reliable control system, we might say that it will arrive at the intended position with probability 0.92. The other 8% of the time, the robot will have moved into one of the three unintended grids or not moved at all. In this situation, the transition probabilities encoded statements about the robot's reliability before it was sent into its environment.

In our approach, an agent constructs a transition model online, while it is playing its game, by incrementally adding to the model as it makes further observations. In particular, the agent straightforwardly tabulates the human player's response to the agent's action. The agent's actions are initially based on a finite state machine, which may later be augmented or replaced by MDP-based intelligence. Once all of the necessary data have been collected, the resulting transition matrices are finalized, and we make sure they are ergodic by using Bayesian data adjustment based on a Laplacian prior (Mitchell 1997). (The Laplacian prior was selected arbitrarily, merely to distribute the probability such that ergodicity can be guaranteed.)

EXPERIMENTS

To apply and evaluate NPC-learning in an experimental context, we employed a dodgeball video game, pictured in Figure 2. The game was developed in C++ using OpenGL and QUAKE III$^{®}$ models. Currently, there are several components that are operating-system-specific to

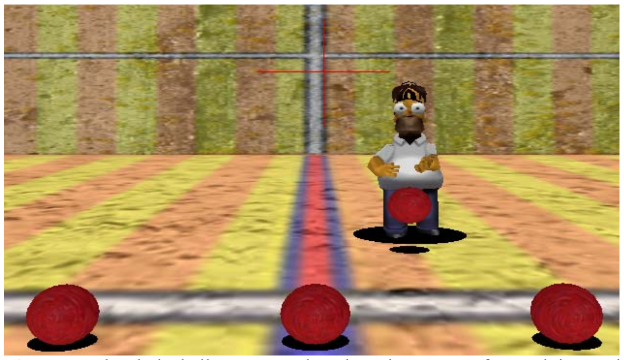
Figure 2. The dodgeball game employed as the context for applying and evaluating our approach to NPC-learning. The human player, not seen, throws balls at and dodges balls from the NPC pictured here.

Windows NT and XP, but the game could be readily ported to other POSIX platforms that support OpenGL. Further information can be found at http://www.cs.hmc.edu/~aarvey/research/cgaims05/.

The human player throws red dodgeballs (seen in the Figure) at the NPC (who, it has been noted, may seem to resemble Homer Simpson). A ball is considered live if it was thrown by a character and has not yet hit the floor, walls, or any obstacle. If a ball is live, a stream of white fire trails the ball, as can be seen in figure 3. The non-human character has perfect knowledge of the entire state space, which includes the position of both players and whether or not a ball is nearby.

Once there is sufficient knowledge of the player's reactions to determine a transition matrix, an optimal policy can be computed. The computation takes approximately 15 seconds for a 5x5 grid decomposition on a 1.2 GHz Intel Pentium Processor, 40 seconds for a 6x6 grid, and 140 seconds for a 7x7 grid. Grid size has little effect on final results and due to the quadratic increase in runtime, results discussed are based on a 5x5 grid.

In our experiments, we consider two components of navigation, *reasoning* processes that decide to which grid element the agent moves, and *steering* processes that guide the agent's step-by-step progress. Our three experiments reported in this section varied in the ways FSMs and MDPs were utilized for these processes. Typically, MDPs are used for higher level reasoning, and control systems of some sort are used for steering; indeed, all of our experiments employ MDPs for reasoning. In two of our experimental conditions, however, a FSM is used as part of the control system, and in the third, the MDP itself is the full control system. In our particular experiments, FSM-based steering made movement seem less rigid, because our FSMs utilized a continuous action set and our MDPs were designed to utilize only a discrete action set. FSM steering goes against the ideal of this study, which is to give the agent as much autonomy as possible and have it rely primarily on learned behavior.

MDP Steering

In this experimental condition, a MDP was the only tool used for steering. At every time step, the state would be inspected and the action derived from the policy would be

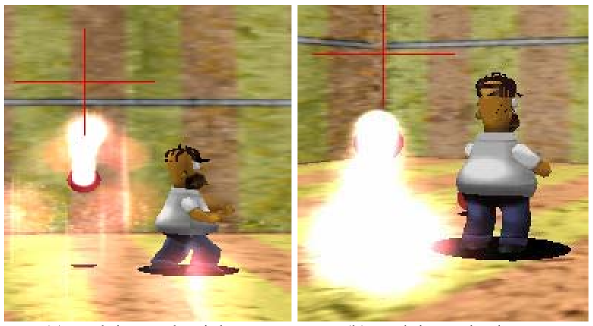

Figure 3. Two images depicting how the non-player character dodges in two different states. In both instances, the NPC had become stuck between two states and thus the only change in state was the incoming ball.

followed. This is in contrast to inspecting the state every 20 time steps and dictating to a control system the direction to take for the next 20 steps, which is similar to the FSM-based steering used in the next experiment discussed.

In our dodgeball video game, this method learned with the least amount of *a priori* knowledge, and the agent's movement became noticeably more like that of the human player. This was most noticeable in the agent's ability to dodge balls. Indeed, the non-player character not only learned how to dodge a ball, but it also did so by moving in a manner very similar to that of how the human player moved. In fact, it appeared to learn dodging behavior better than agents in the other two experimental conditions (described below) did. These results were somewhat encouraging, but the resulting navigation was unrealistic in that the NPC became "stuck" between two grid elements—it moved between two grid elements to dodge attacks, but its reasoning did not enable it to enter any other locations.

MDP Using FSM Steering

In this experimental condition, a MDP was used exclusively for higher level reasoning, and a FSM steered the NPC. Every certain number of time steps, the FSM would ask for guidance from the MDP; after receiving a goal state from the MDP, the FSM would guide the agent in that direction. (Detours, such as dodging balls, were handled by the FSM due to the immediacy of the situation.) Once the FSM reached the goal, it would request a new goal state from the MDP.

Because a FSM was used for steering, movement was smoother and more natural, but the MDP's role in decision making was minimized. The result was very similar to purely FSM based movement; even when parameters were altered to give the MDP more power, little change was noticed. Dodging behavior was not learned, and player navigation style was not well-imitated.

MDP/FSM Hybrid Steering

In this experimental condition, in order to make the movement smoother, a MDP and a FSM were jointly responsible for steering: A MDP controlled steering 90% of the time and a FSM had control the other 10% of the time. As in all experimental conditions, a MDP was employed for reasoning.

The navigation observed using this method was roughly the same as with MDP steering: The agent successfully learned how to dodge a player's attacks. Furthermore, there was some improvement in naturalness: The MDP steered too rigidly when left in complete control, but the occasional FSM continuous action in this hybrid steering made the movement look much smoother.

FUTURE WORK

Autonomy

One goal of our work is to improve autonomy of NPCs, but some simplifications in our present approach are contrary to this goal. For instance, we provide the agent full information concerning the state space, action space, and rewards to be received. In this section, we discuss possibilities for improving autonomy by having agents learn these MDP components using a imitative framework similar to the one currently being employed.

The state space is currently based on a predetermined grid size, positions of the players, and positions of the balls. Instead, however, the agent could learn a continuous state domain, enabling more natural and flexible navigation. A continuous domain would pose new problems for determining the transition model, however, and could also perhaps require considerably more computation power while not completely escaping the need for *a priori* knowledge.

The pre-specified action space, currently representing only movement to adjacent grid spaces, could instead be learned, perhaps by employing a hierarchical framework (e.g., (Wood 2004; Pineau and Thrun 2002)). For example, a hierarchical action "attack" could entail "get a ball," which in turn could require the NPC to "move left" to get a ball; non-hierarchical action representations could lose some of these essential relationships among actions, upon which higher-level reasoning could be based.

The rewards currently being received by the NPC are based on experimenter intuition, but a reward structure could instead be learned via *inverse reinforcement learning* (Ng and Russell 2000). Several promising studies have been conducted in which an agent observes the actions resulting from an optimal policy and derives a close representation of the reward structure being used to generate that policy.

Recomputing Policies

To more effectively remain intelligent in a dynamic game environment, a non-player character could have a threshold level for when the policy needs to be updated due to changes in the transition model underlying its MDP. One approach to this problem would be to use a probability distribution divergence function such as KL-divergence (Kullback and Leibler 1951) in order to determine when the two transition models, old and new, are sufficiently different. Once this threshold is surpassed, re-computation would be necessary. It is instead possible to continually re-compute the policy based on the most recent observation, but transition models frequently remain similar over small changes in time, and such continual computation is likely to be a needless expense.

CONCLUSION

This paper discusses and demonstrates our approach to building a transition model for use in Markov decision processes in real-time video game applications. We applied this technique to the particular goal of learning navigation intelligence for non-player characters in a dodgeball video game: The NPCs observe the navigation styles of a human player, construct the relevant transition model, and compute a policy upon which they base their game play. The paper reports results of our initial experiments, which are encouraging although less than optimal; in particular, NPCs were able to learn how to dodge attacks, and we believe additional research will yield even greater learned intelligence.

ACKNOWLEDGMENTS

We thank Claremont McKenna College for funding support and Matt Reynolds for significant contributions to the code base. We also thank the conference referees for their thoughtful comments.

REFERENCES

Alissandrakis, A., C. L. Nehaniv, and K. Dautenhahn. 2000. “Learning How to Do Things With Imitation Learning.” In *AAAI Fall Symposium on Learning How to Do Things*. 1-6.

Bartlett, P. L. 2003. “An Introduction To Reinforcement Learning Theory: Value Funtion Methods.” In *Advanced Lectures on Machine Learning*. Springer Verlag New York Inc., New York, NY. 184-202.

Barto, A. G., S. J. Bradtke, and S. P. Singh. 1995. “Learning to Act Using Real-Time Dynamic Programming.” *Artificial Intelligence*, vol. 72, no. 1-2: 81-138.

Bradtke, S. J. 1994. “Incremental Dynamic Programming for On-Line Adaptive Optimal Control.” PhD thesis, University of Massachusetts.

Horman, Y., and G. A. Kaminka. 2004. “Improving Sequence Learning For Modelling Other Agents.” In *Proceedings of the AAMAS 2004 Workshop on Learning and Evolution in Agent-Based Systems*.

Kaebling, L. P., M. L. Littman, and A. R. Cassandra. 1998. “Planning and Acting in Partially Observable Stochastic Domains.” *Artificial Intelligence*, vol 101, no 1-2: 99-134.

Koenig, S. and R. G. Simmons. 1993. “Complexity Analysis of Real-Time Reinforcment Learning.” In *National Conference on Artificial Intelligence*. 99-107.

Kullback, S. and R. A. Leibler. 1951. “On Information and Sufficiency.” *Annals of Mathematical Statistics*, vol. 22: 79-86.

Laird, J. E. and M. van Lent. 2000. “Human-Level AI’s Killer Application: Interactive Computer Games.” In *AAAI/IAAI*. AAAI Press/The MIT Press. 1171-1178.

Mitchell, T. M.. 1997. *Machine Learning*. McGraw-Hill Publishers, New York, NY.

Ng, A.Y. and S. Russell. 2000. “Algorithms for Inverse Reinforcement Learning.” In *Proceedings of the 17th International Conference on Machine Learning*. Morgan Kaufmann, San Francisco, CA. 663-670.

Pineau J. and S. Thrun. 2002. “High-Level Robot Behavior Control Using POMDPs.” In *AAAI Workshop on Cognitive Robotics*. AAAI, Menlo Park, CA.

Ross. S. M. 2002. *Introduction to Probability Models*, 8th edition. Academic Press, St. Louis, MO.

Russell, S. and P. Norvig. 2003. *Artificial Intelligence: A Modern Approach*, 2nd edition. Prentice Hall, Englewood Cliffs, NH.

Seneta, E. 1981. *Non-Negative Matrices and Markov Chains*. Springer-Verlag, New York, NY.

Thurau, C., C. Bauckhage, and G. Sagerer. 2003. “Combining Self Organizing Maps and Multilayer Perceptrns to Learn Bot-Behavior for a Commercial Video Game.” In *Proceedings of GAME-ON*. 119-123.

Wood, M. A. 2004. “Agent-Based Imitation Learning Through Hierarchical Behavior Modelling.” Technical Report Department of Computer Science, University of Bath, United Kingdom (Nov.).

BIOGRAPHY

Aaron Arvey is a student at Claremont McKenna College finishing up degrees in both Computer Science and Mathematics. Current research interests include sequential monte carlo methods in computational biology, machine learning, and stochastic processes.

Eric Aaron is an Assistant Professor of Computer Science at Wesleyan University in Middletown, Connecticut. He received his Ph.D. from Cornell University in 2000. His research interests include intelligence modeling for embodied agents and verification methodologies for dynamical navigating actors.