

Improvement in Game Agent Control Using State-Action Value Scaling

Leo Galway, Darryl Charles and Michaela Black

School of Computing & Information Engineering
University of Ulster at Coleraine
Cromore Road, BT52 1SA
United Kingdom

Abstract. The aim of this paper is to enhance the performance of a reinforcement learning game agent controller, within a dynamic game environment, through the retention of learned information over a series of consecutive games. Using a variation of the classic arcade game Pac-Man, the Sarsa algorithm has been utilised for the control of the Pac-Man game agent. The results indicate the use of state-action value scaling between games played as successful in preserving prior knowledge, therefore improving the performance of the game agent when a series of consecutive games are played.

1 Introduction

Digital games provide an interesting test-bed for machine learning research due to the characteristically non-deterministic, dynamic nature of their environments [1]. In particular, the dynamic environments presented by predator/prey style games offer the advantage of being easily decomposed into a finite set of states, each with an associated set of reward values [2]. In order to generate reactive and believable game agent behaviours the use of machine learning techniques is required. However, the effective use of such algorithms is restricted by a number of requirements including the necessity for game agent behaviours to be learned in response to a changing game environment [1], [3]. Subsequently, by incorporating prior knowledge about the learning task into the learning algorithm and knowledge representation used, the performance of the game agent may be improved [4], [5].

Although a large variety of techniques exist within the machine learning domain, reinforcement learning provides an approach to agent-based learning which focuses on an agent's interactions with its environment [6]. As such, reinforcement learning provides a learning methodology appropriate for use within digital game environments and comprises a set of algorithms and techniques which involve learning a sequence of actions in order to maximize an accumulated discounted reward received from the environment over a period of time. Subsequently, a *control policy* can be learned, through an agent's exploration and exploitation of the environment, without requiring explicit training from a domain expert [4], [6], [7], [8]. For a comprehensive discussion on reinforcement learning, please refer to [6].

Within the academic digital games research literature, reinforcement learning techniques have been applied to a variety of games in order to learn control policies for game agents. Research conducted includes the *Sarsa*(λ) generation of a near-optimal control policy for game agents in the fighting game "Tao Feng" [8], and both

Sarsa and *Sarsa*(λ) based game agent controllers within a variation of the game “Pac-Man” [5]. Similarly, *Sarsa*(λ) has been used to generate a control policy for game agent strategies in the game “Settlers of Catan” [4]. By making use of domain specific knowledge within both the value function representation and learning algorithm used, enhanced game agent control has been observed [4], [5].

Based on previous research conducted into reinforcement learning-based control of a game agent within a dynamic, 2D game environment [5], the objective of the research outlined in this paper is to improve the performance of a game agent over a series of games played consecutively by retaining information about the learning task between games. Experiments regarding the retention of state-action values between successive games have been performed, including investigations into the use of linear scaling as a mechanism for retaining learned information over a series of games. Details of the experiments will be presented, along with analysis of the results obtained by the game agent controller, discussed in terms of the game related objectives of the game agent.

2 Methodology

The game environment employed was a variation of the classic arcade game Pac-Man [9], in which the primary objective for the player is to achieve as high a score as possible by manoeuvring the game agent (*Pac-Man*) around a 2D, grid-based environment in order to consume dots while at the same time avoiding being eaten by four opponent agents (*ghosts*). Consisting of a 20x20 grid of game-dependent *features*, including *walls*, *dots*, *energizers*, *tunnels*, *inaccessible spaces* (i.e. grid cells for the starting position of the opponent agents), and *empty spaces* (i.e. grid cells where a dot/energizer has been consumed), the configuration of features establishes a single game *level*. Initially populated with 180 dots (176 dots & 4 energizers), a single level has been used throughout all experiments performed. Within the course of a game, if the game agent consumes an energizer the game state of the opponent agents temporarily changes from the default *Attack* state to the *Evade* state, during which the game agent may eat the opponent agents. The duration of the *Evade* state has been predefined as 300 simulation steps, which may be further reset to a maximum possible duration of 300 simulation steps each time one of the remaining energizers is consumed within the 300 simulation steps. If an opponent agent eats the game agent while in the *Attack* state, the game agent loses 1 out of 5 lives. Eaten agents are regenerated within the inaccessible spaces, with the game state of opponent agents being reset to the *Attack* state. For both the game agent and opponent agents, moves may be made in the North, South, East and West directions. A series of moves were randomly pre-generated and used during game-play for each of the opponent agents, thus preventing the learning algorithm from simply learning a deterministic set of movement patterns for the opponent agents. To allow for direct comparisons of games played, the game agent has been restricted to a maximum of 1000 moves per game. A game ends when the number of game agent’s lives has reached 0, the total set of dots, including energizers, has been consumed, or the game agent has reached the maximum number of moves permitted. For the level used within the experiments presented, a maximum possible score of 2680 may be achieved by the game agent.

During each game played the choice of moves for the game agent was made using the Sarsa control algorithm. By decomposing the game environment into a 20x20 grid, each grid cell was represented by the control algorithm as a state, with actions corresponding to the 4 possible moves. Due to the finite number of resulting state-action pairs, the value function, $Q(s,a)$, was represented as a look-up table. Every time a move was required for the game agent, the control algorithm was run for 100 episodes of learning, with the initial state for each episode of learning corresponding to the current position of the game agent. For each episode of learning performed, 2 steps look-ahead were used and the appropriate state-action values updated. The action corresponding to the state-action pair with the highest value for the state associated with the current position of the game agent was then chosen as the game agent's move. Throughout all episodes of learning a ϵ -greedy action selection policy was used with a low exploration rate ($\epsilon = 0.1$) in order to maintain a high degree of exploitation of learned state-action pairs.

For the experiments presented in this paper two series of 20 consecutive games were played. In the first game played in each series, the initial values for the state-action pairs were generated randomly with values in the range [-0.1, 0.1]. As game-play progressed, the state-action values were updated by the Sarsa algorithm as each movement choice was made for the game agent. In the first series of 20 games played, the set of state-action values at the end of game_n were used as the initial set of state-action values at the start of game_{n+1} , without further re-initialisation. By contrast, in the second series of 20 consecutive games played, the set of state-action values at the end of game_n were linearly scaled within the range [-0.1, 0.1] before being used as the initial set of state-action values for game_{n+1} .

Throughout the experiments performed a number of game related metrics were recorded for each game played, including the score obtained, the number of dots consumed by the game agent, and the entropy of the moves made by the game agent over the course of a game. In particular, previous research has shown that the entropy of the moves made by the game agent may be used to measure the spatial diversity of the agent over the game environment. As such, games in which a high entropy value is measured indicate a more interesting range of moves have been made by the game agent [10]. The normalized entropy of the game agent, E_n , was obtained using the following set of equations:

$$E_r = -\sum \frac{p_i}{P} \log_2 \left(\frac{p_i}{P} \right)$$

$$E_n = \left(\frac{E_r}{\log_2(P)} \right) \quad (1)$$

where p_i is a count of the number of times a specific grid cell has been visited by the game agent and P is the total number of moves made by the game agent during the course of a game [5], [10].

ESANN'2008 proceedings, European Symposium on
Artificial Neural Networks - Advances in Computational Intelligence and Learning
Bruges (Belgium), 23-25 April 2008, d-side publi., ISBN 2-930307-08-0.

