EFFICIENT USE OF REINFORCEMENT LEARNING IN A COMPUTER GAME

Yngvi Björnsson, Vignir Hafsteinsson, Ársæll Jóhannsson, and Einar Jónsson
Reykjavík University, School of Computer Science
Ofanleiti 2
Reykjavik, Iceland
E-mail: {yngvi, vignirh02, arsaelltj02, einarj02}@ru.is

KEYWORDS
Computer game-playing, Mobile games, Artificial intelligence, Reinforcement learning.

ABSTRACT

Reinforcement learning methods are not yet widely used in computer games, at least not for demanding online learning tasks. This is in part because such methods often require excessive number of training samples before converging. This can be particularly troublesome in mobile game devices where both storage and CPU are limited and valuable resources. In this paper we describe a new AI-based game for mobile phones that we are currently developing. We address some of the main challenges of incorporating efficient on-line reinforcement learning methods into such gaming platforms. Furthermore, we introduce two simple methods for interactively incorporating user feed-back into reinforcement learning. These methods not only have the potential of increasing the entertainment value of games, but they also drastically reduce the number of training episodes needed for the learning to converge. This enhancement made it possible for us to use otherwise standard reinforcement learning as the core part of the learning AI in our game.

INTRODUCTION

Reinforcement learning methods have in recent years increased in popularity and are now-a-days used for solving various demanding learning tasks. Although they have been used successfully for years in for example classic board game-playing programs (Tesauro 1994), they have only recently caught the interest of the commercial game community (Evans 2002; Manslow 2004). However, one of the main criticisms these methods have met is their lack of efficiency. That is, many trials are often needed before the learning converges, rendering them practically inapplicable in fast paced game environments where many trials are a luxury one cannot afford. For any on-line learning method to be applicable in practice in games it needs to be *fast*, *effective*, *robust*, and *efficient* (Spronck 2003).

In this paper we address some the challenges of incorporating efficient on-line reinforcement learning techniques into gaming environments. We are using reinforcement learning as the central component in an AI-based game that we are developing for mobile phones — a platform where efficiency is of a paramount importance, in part because of limited CPU power. We describe the main learning method used in our game, and introduce and experiment with two enhancements that allow us to incorporate user feedback interactively into the learning process. Not only does this allow the user to take on a more active role in the development of the game characters, but also drastically reduces the number of training episodes needed for the learning to converge.

These enhancements made it feasible for us to use reinforcement learning as the central learning component in our mobile game. Although the game is still in early stages of development, we have finished prototypes of the components that most heavily rely on learning (the athletic training events). We used them to demonstrate the feasibility of reinforcement learning when augmented with the new learning enhancements.

The remainder of the paper is structured as follows. In the next section we give an overview of the game, and thereafter we explain in detail the learning procedure used in the athletic training event module. We next provide experimental results, and finally conclude and discuss future work.

GAME DESCRIPTION

The game consists of a (pet) creature in a virtual home environment inside a mobile phone or PDA. The objective of the game is to raise a creature with desirable characteristics. The player (user) decides which characteristics are important, and he or she has various ways of interacting with the creature to help shape its character. This is in the spirit of games like *Black & White*. For example, a player might want to raise the creature to be a good athlete, and would therefore have the creature do activities that develop athletic skills, such as swimming or running. Alternatively, the player might prefer more of a thinking creature, and instead have it read books and solve puzzles. In addition to these training activities, the user can show off the creature's skills by having it participate in competitions like athletic or mind-game events. These competitions can be either single-player (e.g. racing against time) or multi-player (e.g. competing against a friend's creature).

Game Architecture

The game is written in Java2 Micro Edition (CLDC 1.1 and MIDP 2.0). Most new phones do or will support these new standards. We have tested the game on a Nokia 6230 mobile phone.

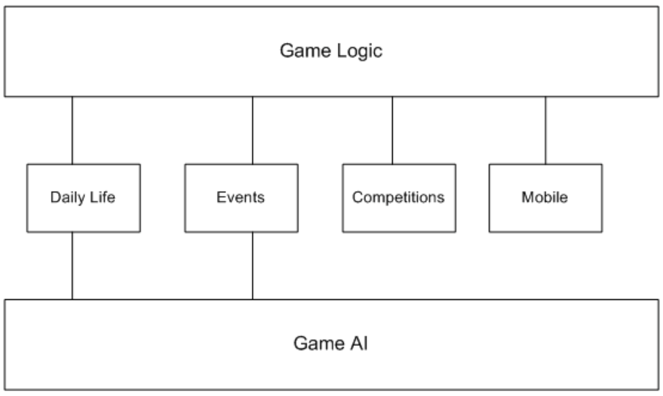

Figure 1. The main modules of the game

The main modules of the game are shown in Figure 1. These modules are loosely coupled and all communications between them are done via interfaces.

Daily-life

This module focuses on the creature's principal needs, such as eating, drinking, sleeping, and social interaction. The food has various attributes; it can be healthy, tasty, fattening etc. Healthy and fattening food types have physical impact on the creature, where as attributes such as taste affect the creature's mental condition (we're not too happy when we need to eat something that tastes bad).

Events

This module contains various sub-games and challenges where the user can train the creature in various skills and activities and benchmark its performance. These include physical activities such as sprinting and swimming events, and simple mind-games like 9-men-morris. The more training the creature gets in an activity the better it becomes. Not only do the physical or mental attributes improve, but the creature also learns the best strategy for the given activity, for example a good strategy in 9-men-morris, or how to pace its speed during a 200 meter sprinting event. During the training the user may play the role of a personal trainer by giving advices, for example by playing a game against it or telling it how to pace its speed during a training run.

Competitions

Players can sign their creatures up for competitions and tournaments, either single or multi-player. The primary purpose is to show-off abilities of the creature, but good results in such competitions (such as 1st prize in a sprinting competition) can provide the player with money and rare valuable items. In a competition the creature's performance is decided by its unique combination of personality, abilities, and skills acquired through training and other previous interactions with the user. Thus it is important to train the creature well before entering a competition.

Mobile

All mobile device specific code, such as graphics, communications, menus and other user interface components are kept in a separate module. This enforces transparent design which will enable the game to be extended to other gaming platforms.

Game AI

Machine learning plays a central role in this game. The main objective of the game AI is to support creatures that evolve and adapt based on the player's preference. There are several attributes that make one creature distinct from the next, both physical (such as strength, dexterity, constitution, fat and health) and mental (such as intelligence, wisdom, patience and social behavior). The way the user interacts with the creature affects how these different attributes develop. The exact learning tasks are somewhat different and can be coarsely divided in two.

In the *Daily-life module* the task of the learning is to allow the various characteristics of the creatures to adapt and develop in response to the user's input. The user is here primarily in a parenting role. The creature, when left unattended, starts exploring and doing things on its own. For example, it could: go hang out around the eating bowl looking for food, go playing, eat your slippers, give you flowers, break your favourite vase, or even poop on the floor. It is up to the user to raise the creature in a way such that it acts responsibly, for example by rewarding it when it does good things and punishing it when it does undesirable things, e.g. eating unhealthy, staying up late, or avoiding exercise and social interactions. These are all activities that negatively affect the creature's health and might possibly eventually lead to its death.

In the *Events module*, on the other hand, the creature learns from experience by repeatedly performing athletic or mind game activities. The more it practices the better it gets at these activities. There is no input necessary from the user. However, if the user wants she can act as a coach and give advice. The creature takes notes of and uses the advice to speed up its learning curve. In this paper we will focus on the learning in this module.

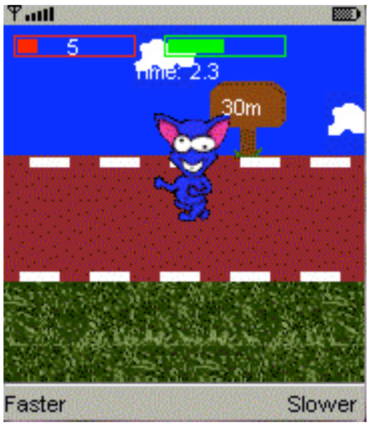

Figure 2. A scene from the Sprinting sub-game

LEARNING

In here we describe the learning method used in the game's athletic training events. All the different sport training events (currently only sprinting and swimming) use the same underlying *reinforcement learning* based methodology. The task is to learn the optimal racing strategy for the given event, possibly with some learning input from the user. The best strategy will differ from one creature to the next because of different physical characteristics (e.g. weight and maximum stamina). For the same reason the optimal racing strategy may differ from one week (or day) to the next, because the physical condition of the creature might have changed in that timeframe.

In the reinforcement learning paradigm an agent learns by interacting with its environment. The agent has a goal that it tries to achieve and while striving towards this goal the agent takes actions that affect the environment. The agent senses the environment and thus knows the consequences of its actions, both in terms of detecting the current state of the world, and also by receiving a numerical *reward* in a response to each action. The strategy that the agent follows for deciding how to act in different situations is called the agent's *policy*. The agent seeks to learn the *optimal policy*, that is, the policy that maximizes the overall reward it receives in the long run.

Reinforcement learning problems are typically formulated as Markov Decision Processes (MDPs). Important world properties are used to represent unique states in the MDP. The underlying assumption is that each state represents all properties of the environment that are important for decision making in that state (Sutton and Barto 1998). The actions the agent takes give the agent an immediate reward and transfer it to a (possibly) new state. These state transitions are not necessarily deterministic.

Formulating the Learning Problem

We can formulate the sprinting game as a MDP. The 200-meter long running course is divided into 20 consecutive 10-meter zones. The creature makes a decision at the beginning of each zone how fast to run. It can choose between four progressively faster speeds: walking, jogging, running, and sprinting. Each running speed affects the creature's endurance differently. The creature's endurance is represented with 21 discrete levels, ranging from 0 (minimum endurance) to 20 (maximum endurance). The endurance decreases 2 levels each zone sprinted, decreases 1 level when running, does not change when jogging, and replenishes by 1 level when walking. The endurance can though never exceed twenty nor become less than zero. If the endurance becomes zero the creature can not run anymore, but must instead walk the next zone to replenishing its endurance. Over all, the decision of how fast to run through a zone depends on which zone the creature is in and its remaining level of endurance. This formulation of the running game allows us to represent it as a (cycle-free) Markov-Decisions Process (MDP), as shown in Figure 3.

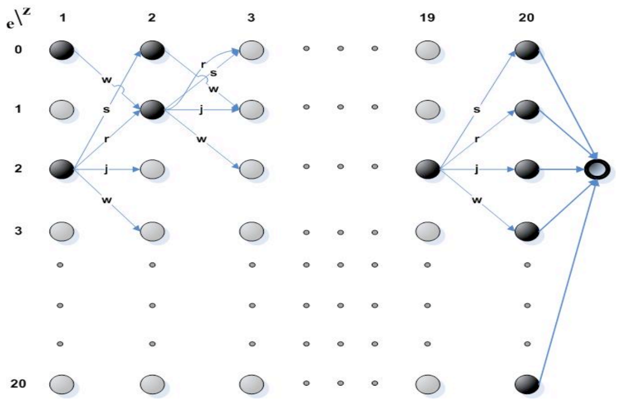

Figure 3. Sprinting game as MDP

Each state is uniquely represented by a *(z, e)* pair, where *z* is the current zone (1-20) and *e* the remaining endurance level (0-20). There are in total 421 different states (21 states for each of the 20 zones plus one final state). The transitions between the states represent the running speed actions, each resulting in us ending in the next zone, although with different remaining endurance. The actions are deterministic and the reward returned by each action is the negation of the time it takes to run the zone at the given speed. The states we draw actions for are shown as dark colored in Figure 3; the light colored states will also have analogous actions although they are not shown. There are 4 possible actions in each state (walk, jog, run, sprint), except in states where endurance is 0 where there is only one possible action (walk).

Depending on the creature characteristics a different MDP might be created. For example, if the creature is overweight the sprinting action might be disabled, or the stamina might drain or replenish at a different phase. Also, the walking or running speed (the rewards) may differ from one creature to the next or by how well the creature is conditioned. The optimal racing strategy may therefore differ from one run to the next, and therefore it may be non-trivial for the user to figure out the correct policy each time.

Q-learning

For learning the optimal policy we use a well-known reinforcement learning algorithm, *Q-learning* (Watkins 1989). The pseudo-code of the algorithm is shown in Fig. 4.

```
Initialize all Q(s,a)
Repeat (for all running episode):
 Set s to be a starting state
 Repeat (for each step in run):
   Choose a from s using agent policy
   Take action a, observe r, s'
   $Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a') - Q(s,a)]$
   $s \leftarrow s'$
 Until s is goal state
End
```

Figure 4. The Q-learning algorithm

The action-value function *Q(s, a)* gives for each state the value of each of its actions. This is the function we must approximate. Once the learning is finished an optimal policy is easily achieved simply by greedily choosing in each state the action with the highest *Q(s, a)* value.

In the beginning the *Q(s, a)* function is initialized with arbitrary values. Next we execute many learning episodes from a given starting state; in our case a 200-meter sprint from a staring position (any zone 1 state can be a start state, depending on the initial endurance of the creature). For each episode a decision is made in each step (zone) what action to take, that is, how fast to run. We use an ε-greedy strategy to pick an action, that is, ε part of the time a random action is taken, but otherwise the best action is taken. The best action is the one with the currently highest *Q(s, a)* value. It is necessary to occasionally take locally non-optimal actions for exploration purposes, otherwise we risk getting stuck in local optima and never finding an improved policy. This is the reason for using the ε-greedy strategy. Typically one gradually decreases the exploration rate as more and more episodes are executed. The update rule

$$Q(s,a) \leftarrow Q(s,a)+\alpha[r+\gamma \max_{a'} Q(s',a')-Q(s,a)]$$

gradually updates the Q(*s*, *a*) action-values until they converge. The constant α is an adjustable step size parameter, and γ is a discount factor of future rewards (not used in our case, that is, set to 1.0). The term *r* is the current reward, *s* and *a* are the current state and action respectively, *s'* is the next state and *a'* an action from that state. The $\max_{a'} Q(s',a')$ function returns the value of the currently best action from state *s'*. The Q-learning algorithm assures convergence to optimal values (in the limit) given that the basic Markov properties hold and the α parameter is set appropriately. For a more detailed discussion of Q-learning see for example (Sutton and Barto 1998).

Improving the learning speed

The standard Q-learning algorithm was able to learn the optimal running strategy in our domain within a couple of thousand episodes. However, there were two problems with this approach:

- The first is that this is far too slow convergence for the algorithm to be practical in our application domain. Even a few hundred episodes would be excessive.
- The second problem is that the player itself has no control over the learning and is simply a passive observer. This could potentially make long learning sessions uninteresting.

To overcome the above problems we designed the game such that the user is allowed to participate in the decision process, effectively taking on the role of a coach by giving feedback. This not only allows the user to take on a more active role in the game, but the player's input can additionally be used to reduce the convergence time of the Q-learning algorithm.

During a run the user observes the creature and can overwrite its decisions. In other words the user can tell it to run either slower or faster. We record the user's (last) preferred action in each MDP state encountered. There are two different ways we can use the user's feedback to speed up the learning.

Imitation Runs

During a many episode training process we periodically rerun a running episode coached by the user, taking the user preferred actions where applicable. These reruns are done in the background and are transparent to the user. Given that the user gave good advice, the creature will get better result sooner and thus starts adapting the good strategy. However, if the advice was not good the creature might get temporarily sidetracked, but then gradually moves away from the user's strategy.

Bonus Reward Points

In this approach extra reward points are awarded to the user-preferred action during regular running episodes. On one hand, this approach has the benefit of not requiring extra reruns. On the other hand, it can be potentially dangerous because we are changing the learning problem. Because of the bonus reward the total reward is not anymore the negation of the total running time. Despite this, given that the feedback is useful, Q-learning can still learn an optimal policy (and that quickly!). Conversely, giving bad user advice can delay the learning process and even possibly prevent optimal policy to be learned. Note, however, that this does reflect a real life scenario where bad advice from a coach is harmful if always followed blindly.

EXPERIMENTS

This section gives the experimental results of a comparison study between standard Q-learning and Q-learning augmented with the enhancements proposed above: *imitation runs* and *bonus reward*. We ran the enhanced Q-learning methods every fifth episode (instead of a regular episode).

At the start of a 200-meter sprint the creature has full endurance (level 20). The time it takes to traverse a zone when walking, jogging, running, and sprinting are 1.5 s., 0.8 s., 0.4s. and 0.1s., respectively. When using an optimal strategy (running all but the last zone where one sprints) the running time for the 200 meters will be 7.7 s. The ε parameter is set to 0.15 and α to 1.0. We ran the three different learning approaches until convergence was reached observing how long it took them to find an optimal policy. The performance of the three methods is shown in the following graphs. Each data point represents the running-time average of 10 independent runs (tie-breaks between equally good actions are broken randomly; therefore we base each data point on several runs).

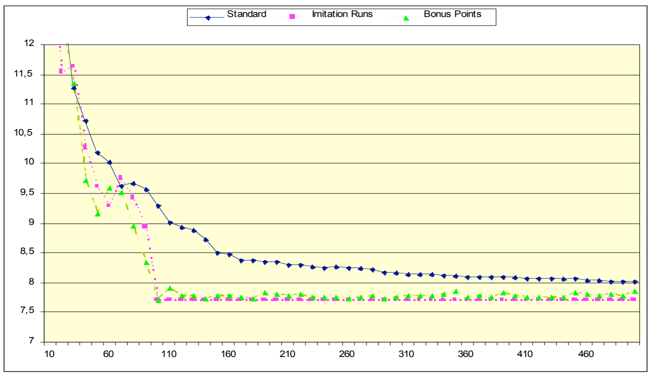

Figure 5. Coach providing helpful advice

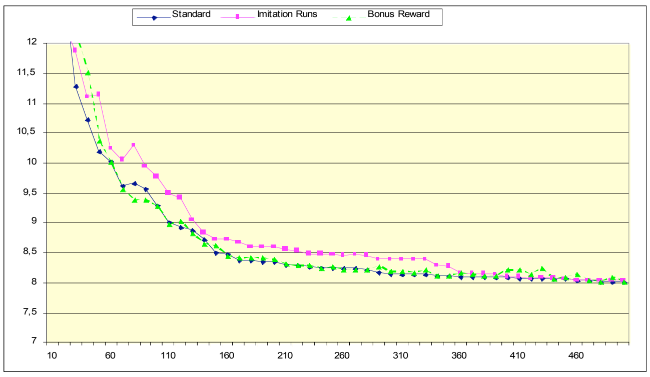

Figure 6. Coach providing random advice

Figure 5 shows how the learning progresses when the user provides useful advice (the user performs the optimal running sequence). Both the enhanced Q-learning methods converge faster than the standard method. Within 100 episodes they have discovered the optimal policy. The standard Q-learning needs about 2000 episodes for that. *This is a twenty fold reduction in the number of episodes*. We ran similar experiments with slightly different values for the ε and α parameter, but all yielded similar results.

We were also curious to know what happens if the coach provides useless advice. We ran the same set of experiments, but now with a random strategy interleaved every fifth episode. The result is shown in Figure 6. Now all methods perform similar, although the *Imitation Run* method is slightly worse to start with. However, within 300 runs they all have converged to an equally good policy and within about 2000 episodes to an optimal one (same as standard Q-learning). The reason why random advice does not seem to hurt is that Q-learning is a so-called *off-policy* algorithm, that is, it can learn a good policy while following an inferior one.

Finally, we experimented with the case where the user deliberately advices a bad policy (walk all the way). In this case both the enhanced algorithms started out really badly but were eventually able to recover, although it took somewhat longer this time. We did not expect beforehand that the *Bonus reward* approach would be able to recover because the learning problem has been changed. However, because the enhanced Q-version is executed only 20% of the time and the reward bonus is relatively small, the change in total reward is small enough not to affect what is an optimal policy. However, this is not necessarily always the case, and one must be careful using this approach in situations where there are several policies similar in quality.

CONCLUSIONS

In this paper we introduced and experimented with techniques for incorporating user-guided feed-back into reinforcement learning. The proposed techniques can drastically reduce the number of training episodes necessary for convergence. They are also robust against the case where the user provides useless feedback. In our game these techniques contributed to the success of the learning by making the learning algorithm converge much faster. This is in our view especially important in mobile games not only because of limited computing resources, but also the fact that typical user game-playing sessions on mobile devices are generally much shorter than on other game platforms

As a future work we are investigating other ways of incorporating and reusing user feedback, as well as measuring how well the techniques scale up to larger and more complex learning tasks.

REFERENCES

Evans, R. 2002. "Varieties of Learning" In *AI Game Programming Wisdom*, Steve Rabin, eds. Charles River Media Inc., Hingham, Massachusetts.

Manslow, J. 2004. "Using Reinforcement Learning to Solve AI Control Problems" In *AI Game Programming Wisdom 2*, Steve Rabin, eds. Charles River Media Inc., Hingham, M.A., 591-601.

Spronck, P.; Sprinkhuizen-Kuyper, I.; and Postma E. 2003. "Online Adaptation of Game Opponent AI in Simulation and in Practice". In *Proceedings of the 4th International Conference on Intelligent Games and Simulation (GAME-ON 2003),* EUROSIS, Belgium, 93-100.

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

Tesauro, G. J. 1994. "TD-Gammon, a self-teaching backgammon program, achieves master-level play." *Neural Computation*, 6(2): 215-219.

Watkins, C. J. C. H. 1989. "Learning from Delayed Rewards." Ph.D. thesis, Cambridge University.

BIOGRAPHY

The first author is an associate professor at the School of Computer Science, Reykjavik University. He holds a Ph.D. degree in computer science from the University of Alberta, Canada, specializing in AI. Prior to moving to Reykjavik University he was for several years a research scientist with the GAMES group at the University of Alberta working on AI in games. The GAMES group has industrial ties with both Electronic Arts and Bioware Inc. The other co-authors are students at the School of Computer Science, Reykjavik University.