A New Perspective to the Keepaway Soccer: The Takers (Short Paper)

Atil Iscen
Middle East Technical University
Ankara, Turkey
atil@ceng.metu.edu.tr

Umut Erogul
Middle East Technical University
Ankara, Turkey
umuero@ceng.metu.edu.tr

ABSTRACT

Keepaway is a sub-problem of RoboCup Soccer Simulator in which 'the keepers' try to maintain the possession of the ball, while 'the takers' try to steal the ball or force it out of bounds. By using Reinforcement Learning as a learning method, a lot of research has been done in this domain. In these works, there has been a remarkable success for the intelligent keepers part, however most of these keepers are trained and tested against simple hand-coded takers. We tried to address this part of the problem by using Sarsa(λ) as a Reinforcement Learning method with linear tile-coding as function approximation and used two different state spaces that we specially designed for the takers. As the results of the experiments confirm, we outperformed the hand-coded taker which results in creating a better trainer and tester for the keepers. Also when designing the new state space, we noticed that smaller state spaces can also be successful for this part of the problem.

Categories and Subject Descriptors

I.2.6 [**Artificial Intelligence**]: Learning; I.2.11 [**Artificial Intelligence**]: Distributed Artificial Intelligence—*Multiagent systems*

General Terms

Experimentation, Performance

Keywords

RoboCup, Keepaway Soccer, Takers

1. INTRODUCTION

Keepaway is a subproblem of the RoboCup Soccer Simulator (RCSS) in which one team, 'the keepers' tries to maintain possession of the ball within a limited region, while the opposing team, 'the takers' tries to gain possession of the ball[2]. This game is commonly preferred in Machine Learning researches [4][5][7], because it can be a good testbed with less agents resulting in a less complex problem rather than two teams with 11 agents playing full team soccer each having different roles. Because of this, there are many researches focusing on the keepaway game, but most of these researches apply machine learning methods to maximize the possession of keeper agents.

In these researches the experiments are made with basic takers, which run towards the ball without considering to cooperate, which does not create a big challenge for the keepers. When developing learning keepers, this type of takers does not really test the potential of the adversary agents. In our project we plan to address this part of the problem by developing learning takers. Another interesting point of developing takers is that in learning keepers problem only the keeper with the ball decides an action whereas in learning takers all the agents have to decide an action in each step. This makes the agents more dependent on each others decision, making the game more suitable for cooperation.

Among learning methods we used Reinforcement Learing[3] which is one of the most preferred learning method in RCSS, because it is well suited to meeting its challenges, like sequential decision making, achieving delayed goals and handling noise. As a Reinforcement Learning algorithm we have chosen Sarsa(λ) learning with tile coding because of its previous success in application of keepers in one of the best known paper in this area.[2]

2. ALGORITHM

2.1 Problem Definition

In keepaway a team of m players faces the task of keeping possession of the ball within a rectangle region of play, resisting attempts of the opposing team of n takers to wrest possession. For research purposes, this problem is embedded into RCSS with the keepaway framework developed by Stone et Al [1]. This framework contains many classes and methods and some high level functions like passing and marking. In addition, although this framework has the main functions for the learning process, we had to modify some of them to make the framework suitable for the takers.

We accepted the task as episodic, each starting with one of the keepers having possession, and finishing when any of the takers gets the ball or the ball goes out of bounds. Apart from the keepers, which decide to an action only when the agent has the ball until its decision to pass, the takers need to decide to an action in each cycle, which can prevent them from reaching any of the keepers if their decided action changes repeatedly. As this makes it impossible to see the effects of their immediate decisions, we decided to make the takers do the selected action n consequent cycles. With

Cite as: A New Perspective to the Keepaway Soccer: The Takers (Short Paper), A. Iscen and U. Erogul, *Proc. of 7th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2008)*, Padgham, Parkes, Müller and Parsons (eds.), May, 12-16., 2008, Estoril, Portugal,pp. 1341-1344.

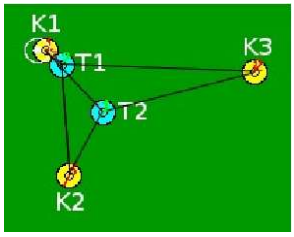

Figure 1: Keepaway scene and labels

using this 'n step same action' trick, we made the learning process easier at start, but this causes a disadvantage when further agility or sensitivity to the states of disregarded cycles is required. We decided the value of n as 15 cycles, which is the duration of a successful pass execution.

2.2 States and Actions

Since we are dealing with the takers, the only possible actions are GotoBall and Mark(n) which means marking the nth keeper, where keepers are ordered by their distance to the ball. When deciding on states, we wanted to minimize the number of states by trying to use the information that would be sufficient. First, to have a dynamic labeling, we sort the keepers according to their distance to the ball. K1 means the keeper with the ball. Then the takers are sorted such that the first taker will be itself. The others are sorted according to their distance to this taker(Fig. 1). For our first taker model (atum) the state variables are constructed as the distances of each taker to each keeper (T1-K1, T1-K2, T1-K3, ..., T2-K1, T2-K2, ...). For increasing number of players, the size of the state space becomes a problem. To overcome this, we minimized the state space by constructing a second taker model. For this model, only the distances for the current taker are taken into consideration, for the other takers only the label of the nearest keeper is considered. This gives for m keepers and n takers $m + (n-1)$ variables, whereas first model has $m * n$ state variables.

2.3 Learning Algorithm

For the learning algorithm, because of its success in learning keepers [2], we have chosen sarsa(λ) which is a commonly used algorithm in RL [3]. For feedback, the rewards are zero until the end of the episode and it becomes 1 when the takers force keepers to end episode. Eventhough we have less state variables, the state space is still too large. To decrease the size, and to generalize the states we used function approximation. For function approximation we used tile coding, which is a linear function approximation scheme that partitions the input space into axis aligned regions called tiles.

3. EXPERIMENTAL RESULTS

3.1 Methodology

For the keepaway problem the common evaluation method is the average episode length of the game. Our aim is to decrease these durations, especially the ones presented in P.Stone's research[2], since we also use the keepers they developed.

After several tries, we chose the learning parameters giving best learning curves. Although we have infinitely many

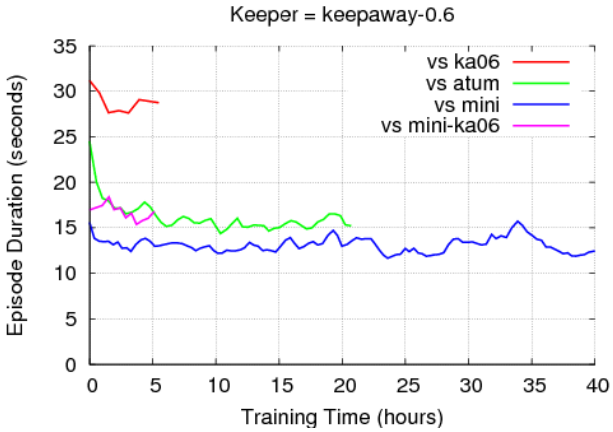

Figure 2: Keeper = ka06

choices, we decided on $\alpha = 0.125, \lambda = 0.5, \epsilon = 0.05, \gamma = 0.8$($\alpha being Learning Rate, \gamma being Discount Factor$).

All of the testing and implementation has been done on 32 bit 2.6.22-14 linux kernel, with rcssbase-10.0.11, rcssserver 10.0.7 and keepaway-0.6 on a 2.20Ghz AMD Athlon PC. The results are converted to graphics using the points constructed by a sliding window containing 300 episodes. All of the experiments are conducted in sync mode(server advances cycle immediately when all clients have responded which allows games to be much faster) with unrestricted vision settings(360 degree view of field).

The first keeper that we used to test is the original hand-coded version that we got from the keepaway framework and is denoted as ka06. For further testing we used the learning keepers provided by M.Taylor et Al[1] which will be denoted as mt07. For mt07 we used weights learned previously, which were saved after a learning process having one of the highest possession durations among learning keepers. For the takers part, we have only one previous taker to compare our work with, which is ka06. Our first taker model is denoted as atum, and the second is denoted as mini in the graphics and the results. The extended versions of the takers like mini-ka06 express takers (in this example mini) which use the previously saved weights against the keepers written after the '-' symbol (ka06 in this example). For atum-l and mini-l, the extension 'l' signifies that they load weights saved during a learning session against learning agents. For durations of experiments we used long sessions for learning agents, and shorter ones for agents loading previously learned weights.

3.2 Results

At first, we compared various takers performances against hand-coded keepers of the keepaway framework[1]. As seen in the Table 1, the hand-coded keepers versus hand-coded takers get an average result of 29.2 seconds, whereas our first taker atum developed to decrease this duration became successfull by getting 16.3 as average. For the third model (mini), although the number of state variables is reduced, it shortens the durations further to 12.9. These statistics clearly show that our expectations in the success of learning takers come true.

For the second test, when we compare atum and mini, there is two important points. First, during the first 5 hours

Table 1: Episode durations against ka06

Takers	Average	Min	Max
ka06	29.2	27.6	31.1
atum	16.3	14.3	24.4
mini	12.9	10.8	15.5

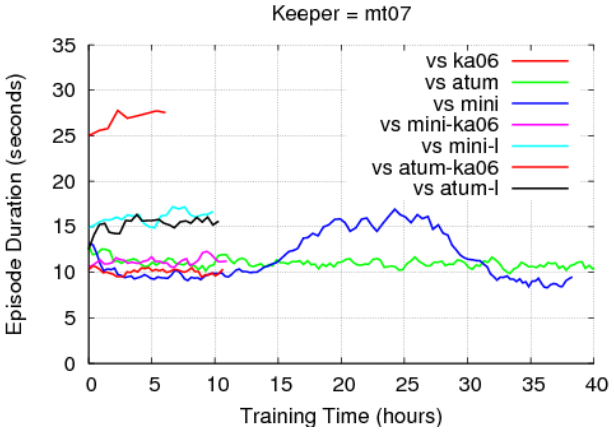

Figure 3: Keeper = mt07

Table 2: Episode durations against mt07

Takers	Average	Min	Max
ka06	26.7	24.9	27.7
atum	10.9	9.9	12.8
mini	11.1	8.3	16.9
mini-ka06	11.1	10.5	12.2
mini-l	15.9	14.8	17.2
atum-ka06	10.1	9.4	10.6
atum-l	15.1	12.4	16.0

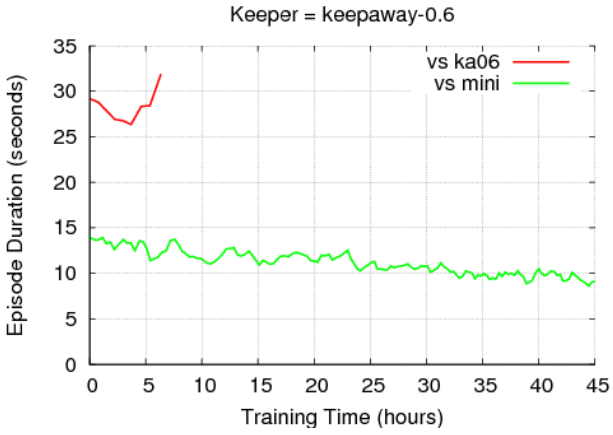

Figure 4: Keeper = ka06 in 4 vs 3

of training(Fig. 3), mini converges more quickly. Secondly, mini has a lower minimum (Table. 2) but its seems more unstable than atum. In our opinion, this is caused by having less state variables not being able to represent the state clearly.

Interestingly there is a big difference between atum,mini and atum-l,mini-l respectively. We believe that the reason of this is the atum-l and mini-l are trained against the keepers which are at the start of the learning process.

Another interesting point for this test is, the takers trained against the hand coded takers are as successful as the takers trained specifically against mt07. This means that the opposing keepers ka06 and mt07 behave similarly, as expected from the statistics given by Stone et Al.[2]

For the last test (Fig. 4), we see that the mini especially developed for keepaway with more agents is successfull at his primary mission with a clear improvement over the hand-coded takers.

4. CONCLUSIONS AND FUTURE WORK

Looking at the results, we can clearly say that we achieved our initial goal which is to develop a successful learning taker which performs better than the hand-coded ones. After testing against various keepers, we have shown that our algorithm is robust to different types of keepers. We also concluded that previous studies on learning keepers can be also applied to the takers with the help of an addition like 'n-step same action'. As a further research, n could be decreased during learning when the agent needs more agility.

Also for the takers part of the keepaway learning problem, using a second model of taker, we saw that we can achieve similar (sometimes better) results in a less stable way. With this new model using less state variables, the same learning process can be used with more than 5 agents with a manageable state space. As a new application area, RoboCup-breakaway can be used to test the general success of this algorithm for the defense team.[6]

In addition to these ones, one of our main contributions is providing a better challenge and a benchmark to the researchers of the keepaway framework. We think that using a better taker for the experiments will help the researchers analyze the true potential of the keepers

5. REFERENCES

[1] P. Stone, G. Kuhlmann, M. E. Taylor, and Y. Liu. Keepaway soccer: From machine learning testbed to benchmark. In I. Noda, A. Jacoff, A. Bredenfeld, and Y. Takahashi, editors, *RoboCup-2005: Robot Soccer World Cup IX*, volume 4020, pages 93–105. Springer Verlag, Berlin, 2006.

[2] P. Stone, R. S. Sutton, and G. Kuhlmann. Reinforcement learning for RoboCup-soccer keepaway. *Adaptive Behavior*, 13(3):165–188, 2005.

[3] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 1998.

[4] M. E. Taylor, S. Whiteson, and P. Stone. Temporal difference and policy search methods for reinforcement learning: An empirical comparison. In *Proceedings of the Twenty-Second Conference on Artificial Intelligence*, pages 1675–1678, July 2007. (Nectar Track).

[5] M. E. Taylor, S. Whiteson, and P. Stone. Transfer via inter-task mappings in policy search reinforcement learning. In *AAMAS '07: Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems*, pages 1–8, New York, NY, USA, 2007. ACM.

[6] L. Torrey, T. Walker, J. W. Shavlik, and R. Maclin. Using advice to transfer knowledge acquired in one reinforcement learning task to another. In *ECML*, pages 412–424, 2005.

[7] S. Whiteson, N. Kohl, R. Miikkulainen, and P. Stone. Evolving keepaway soccer players through task decomposition. *Machine Learning*, 59(1):5–30, May 2005.