Effective and Diverse Adaptive Game AI

István Szita, Marc Ponsen, and Pieter Spronck

Abstract

Adaptive techniques tend to converge to a single optimum. For adaptive game AI, such convergence is often undesirable, as repetitive game AI is considered to be uninteresting for players. In this paper, we propose a method for automatically learning diverse but effective macros that can be used as components of adaptive game AI scripts. Macros are learned by a cross-entropy method. This is a selection-based optimization method that, in our experiments, maximizes an interestingness measure. We demonstrate the approach in a computer role-playing game (CRPG) simulation with two duelling wizards, one of which is controlled by an adaptive game AI technique called 'dynamic scripting.' Our results show that the macros that we learned manage to increase both adaptivity and diversity of the scripts generated by dynamic scripting, while retaining playing strength.

Index Terms

Game, artificial intelligence, reinforcement learning, cross-entropy method, dynamic scripting.

I. Introduction

The main purpose of commercial computer games is to entertain the human player. Most of them do this by posing challenges for the human to overcome, often in the form of combat. The tactics used by the computer-controlled opponents in combat are determined by the game's artificial intelligence (AI). If the game AI manages to keep players motivated to play the game, we call it 'interesting.'

Two key factors in making game AI interesting are *effectiveness* and *diversity*. If game AI is effective, it is believable and a challenge to defeat. If game AI is diverse, it provides a variety of tactics for the player to test his skills against. A major problem is that in most implementations, these two factors are in conflict: effective tactics usually consist of a specific sequence of actions with little room for variety [1].

In theory, adaptive game AI can improve effectiveness against a specific player automatically, while maintaining diversity by constantly trying new tactics. In practice, however, in its quest to improve effectiveness adaptive game AI tends to converge to a very small number of strong tactics, thereby losing diversity. Adaptive AI in actual

During this research, I. Szita held a postdoc position at Maastricht University, The Netherlands (e-mail: szityu@gmail.com).

M. Ponsen is affiliated with the Department of Knowledge Engineering, Maastricht University, The Netherlands (e-mail: m.ponsen@micc.unimaas.nl).

P. Spronck is affiliated with the Tilburg centre for Creative Computing (TiCC), Tilburg University, The Netherlands, and with the Dutch Open University (e-mail: p.spronck@uvt.nl).

Manuscript received October, 2008; revised January 2009, February 2009.

commercial games is also hampered by the fact that games can afford only a few ineffective opponents before the player loses interest. To counteract this issue, adaptive game AI must be based on a-priori knowledge.

In this article we pose the following problem statement: To what extent is it possible to automatically create a-priori knowledge, that can be used to generate game AI that is interesting, i.e., both effective and diverse. We will investigate this question in a simple role-playing game (RPG) scenario, and present some arguments that our approach scales up to more complex tasks.

In Section II we give an overview of related work and literature about concepts that are discussed in this paper. In Section III we provide a general description of our proposed algorithm for automatic macro generation, and discuss the details of applying it to a computer RPG combat environment. In section IV we test the properties of dynamic scripting augmented with macros experimentally. We discuss our results and draw conclusions in Section V.

II. Background

In this section we discuss background information on the work presented in this paper. We provide information on scripting of game AI (II-A), dynamic scripting (II-B), macro actions (II-C), diversity (II-D), and cross-entropy learning (II-E).

A. Scripting

For a long time, the dominant approach to programming game AI was scripting [2], [3], [4], [5]. Even though nowadays some games use approaches to AI that are more advanced than straightforward scripts (such as Goal-Oriented Behavior, which is further discussed in Subsection V-B), often these approaches still use scripts to implement basic action sequences to accomplish tasks. Scripts have the advantage that they are easy to implement, interpret, and modify. However, they also have numerous disadvantages:

Labor-intensiveness. Scripts have to be reasonably complex because they will be used in a complex game environment. They are therefore time-consuming to implement by hand, and must be tested in many different situations [6].

Weaknesses. Because of their complexity, it is likely that scripts contain undetected weaknesses. Thus, supposedly tough opponents can be defeated with simple 'exploits' [3].

Predictability. A human player swiftly recognizes patterns in the behavior generated by scripts. This makes gameplay relatively monotonous, and lowers the entertainment value of the game considerably [1].

Non-adaptivity. Scripts are static and therefore unable to adapt to the human player's style. If the player overuses a certain tactic, the game AI should be able to switch to a corresponding counter-tactic [1].

A simple way to increase the diversity of game AI is to use multiple different scripts. However, such diversity comes at the price of an increased amount of programming and testing needed. Furthermore, even with several scripts the AI remains non-adaptive and leaves the possibility of easy-to-exploit weaknesses.

Random actions are also often applied to make AI more diverse and unpredictable. For example, in a computer role-playing game (CRPG) environment such as BioWare's *Baldur's Gate* series, scripts may contain such commands as "cast a randomly selected, strong defensive spell," instead of specifying a single spell. However, the style of game AI still remains predictable because it is rarely affected by this kind of randomness. Furthermore, randomly chosen actions may seem out of place, and thus inversely affect the illusion of intelligent behavior.

B. Dynamic scripting

Dynamic scripting [1], [7], [8] is a reinforcement learning technique that is able to learn effective game AI scripts automatically. It is computationally fast, effective, robust, and efficient. It maintains several rulebases, one for each opponent type in the game. The rules in the rulebases are manually designed using domain-specific knowledge. When a new opponent is generated, the rules that comprise the script controlling the opponent are extracted from the rulebase corresponding to the opponent type. The probability that a rule is selected for a script is proportional to the value of the weight associated with the rule. The rulebase adapts by changing the weight values to reflect the success or failure rate of the associated rules in scripts. We discuss details of the weight-adjustment procedure in Subsection IV-A. From a theoretical point of view, dynamic scripting belongs to the family of reinforcement learning methods (as it learns from evaluative feedback [9]), but its formalism (i.e., assigning weights/credits to individual rules) is also closely connected to *learning classifier systems* [10].

Dynamic scripting cannot guarantee convergence. This actually is essential for its successful use in games. The learning task in a game constantly changes (e.g., an opponent player may choose to switch tactics), thus aiming for an optimal script may result in overfitting to a specific strategy. Dynamic scripting is capable of generating a variety of behaviors, and to respond quickly to changing game dynamics.

Dynamic scripting was successfully applied to the role-playing game *Neverwinter Nights* [8]. Ponsen et al. [11] applied dynamic scripting to the real-time strategy game *Wargus*. They used evolutionary learning to learn macros, and showed that the learned rules improve dynamic scripting's performance. In these applications, dynamic scripting adapted quickly to a number of static tactics and learned effective counter-tactics. It could not ensure, however, that its generated scripts represent diverse playing styles. Because of the incremental nature of dynamic scripting's updates, it is unlikely that it can switch between two strong tactics without trying several suboptimal ones in-between. While dynamic scripting can learn *effective* tactics, it has no built-in mechanism to improve *diversity*, so it tends to converge to static, predictable scripts.

C. Macro actions

In their simplest form, macro actions are sequences of actions that are handled as a single unit. Macro actions offer a straightforward way to augment dynamic scripting. With larger building blocks, dynamic scripting should be able to switch between different playing styles more quickly. However, without a careful selection of the actions to be grouped, additional macros can even decrease learning performance. In this paper we discuss how macro actions can be automatically designed that allow dynamic scripting to generate interesting tactics.

Grouping basic actions into higher-level abstract actions is an elemental idea in the focus of much current research. There exist several different approaches under many different names (e.g. macros, options, behaviors, and skills). The definition of macros also varies: some researchers define them as closed-loop control policies, while others define them as open-loop action sequences. A full overview is well beyond the scope of this paper (Barto and Mahadevan [12] and McGovern [13] provide such an overview). We only mention several results about automatic macro construction.

McGovern [13] searches for subgoals in a gridworld, and learns macros that reach these subgoals. A state is designated as a subgoal, if it is a "bottleneck", i.e., the number of successful episodes passing through that state is higher than the corresponding value of the neighboring states and also higher than some threshold level. In the HASSLE architecture of Bakker and Schmidhuber [14], abstract states are created automatically by clustering. For each abstract state, a new macro is learned. The work of Singh et al. [15] (described in more detail Subsection II-D) deals with semi-automatically constructed macros: one macro is learned for reaching each "salient event."

D. Diversity and Interestingness

Searching for diversity is a central issue in reinforcement learning: in an unknown environment, the agent must explore new, unknown situations so that it gains knowledge of the system. At the same time, the already obtained knowledge should be used for collecting as much reward as possible, and the agent must find a balance between the two kinds of activities.

There is a rich literature of exploration methods in reinforcement learning (Wiering [16] provides an overview). Most exploration methods, however, are specific to (finite) Markov decision processes. For example, they assume that we can maintain a visit count for each distinct state. Furthermore, these methods perform exploration in order to find a single optimal policy (or value function). In contrast, our aim is to maintain diversity, with policies that are not necessarily optimal, but still effective.

Schmidhuber [17] proposes that exploration should be concentrated to *interesting areas* that are defined as follows. An area is "boring" either if it is well known to the agent and not much is left to learn, or if the area is poorly understood and it is hard to make any progress in learning. The area in-between is where the agent can learn quickest, and is therefore most interesting to him. Naturally, the area of interestingness is continually changing. In the present research, we apply this principle for autonomous learning of interesting behaviors that are given in an explicit form. Note that this machine learning-centered definition of interestingness differs considerably from the traditional human-centered definition, though, according to Schmidhuber, a connection does exist (which motivates the name).

There exist several instantiations of this general concept. Schmidhuber [18] uses two competing agents: the agents can make bets on the outcomes of future observations. An agent gets rewarded if he can correctly guess the answer but his opponent cannot. The underlying idea is that (a) the agents should not bet on outcomes which are known to both of them or unknown to both of them; and (b) they are motivated to explore new areas about which they can ask test questions.

Oudeyer and Kaplan [19] investigate knowledge acquisition in an Aibo robot. They define the area of interestingness in a more straightforward way: they train a predictive dynamical model of the environment, and Aibo is motivated to perform actions that cause the predictive model to improve quickly.

Singh et al. [15] perform similar experiments in a gridworld. The gridworld is enriched with various objects, such as a light switch, a ball, and a bell. A set of "salient events" is hand-coded, such as ringing the bell, and switching the light on. The agent is considered to be in an interesting situation when it will trigger some salient event, but its model is unable to predict that will do so. Technically, the reward of the agent is proportional to the error in predicting salient events.

E. Optimization with the cross-entropy method

The cross-entropy method (CEM) of Rubinstein [20] is a general algorithm for global optimization tasks, bearing close resemblance to estimation-of-distribution evolutionary methods [21]. A short introduction to the algorithm is given in Subsection III-F (with emphasis on its application to macro learning). For a fuller and more general description see the tutorial by De Boer et al. [22].

For sufficiently large population size, CEM is known to converge to the global optimum on combinatorial optimization problems [23]. The areas of its successful application range from combinatorial optimization problems such as the optimal buffer allocation problem [24] and DNA sequence alignment [25] to independent process analysis [26] and reinforcement learning [27], [28], [29]. Recently, the cross-entropy method has also been applied successfully to learning behaviors for the games Tetris [30] and Pac-Man [31].

III. SCHEME FOR MACRO LEARNING

To fulfill their purpose, macro actions have to meet at least three requirements:

Effectiveness. Macros should be effective in the sense that effective tactics can be assembled from them.

Diversity. Macros should differ considerably from each other and should represent different playing styles.

Appropriate size. The size of a macro should be balanced between the two extremes: a single rule or a complete script. In the first case, macros are essentially useless, while in the second case, we lose the possibility of combining multiple macro actions, which is a powerful way of creating diversity.

Our goal is to learn macro actions that satisfy these requirements. The rules constituting the macro actions will be selected from a rulebase. We maintain a probability distribution over the rules of the rulebase, and update probabilities so that macros with higher fitness become more probable. The fitness function depends on two factors: the first rewards strong macros (in terms of combat effectiveness), while the second rewards scripts that differ considerably from previously learned macros. Macros are learned incrementally as a separate optimization task. The reason is that the fitness function depends on previously learned macros.

We test our approach in a simulation environment called MiniGate, which is described in Subsection III-A. The macro-generation algorithm is summarized in Listing 1. It starts with an empty macro list. For each prospective macro (defined in Subsection III-B) we start a new learning epoch and initialize the probability vector. In the main

Listing 1

```
% K: number of macros to be learned
% T: number of battles in a training epoch
L := {}; % start with empty list of macros
for k := 1 to K do {
    phase := k div (K/2); % 0=opening, 1=midgame
    p := InitProbabilities();
    for i := 1 to T do {
        % draw random script according to p
        S_i := GenerateScript(p, phase);
        % play battle using S_i, get game record
        G_i := EvaluateScript(S_i);
        % calculate fitness of script
        F_i := GetFitness(S_i, G_i, L);
        p := UpdateProbabilities(p, i,
                {S_1, ..., S_i}, {F_1, ..., F_i}); }
    M := ExtractMacro(p);
    L := L ∪ {M}; } % add new macro to the list
```

learning loop, a script is generated according to the actual probability vector (III-C). It is evaluated in the game environment and the results are recorded (III-D). Then, its overall fitness is calculated, considering both playing strength and distance from previous macros (III-E). Finally, the probability vector is updated, according to the fitness of the current script (III-F). At the end of the learning epoch, a new macro is extracted (III-G) and added to the list. Subsection III-H presents the experimental results of the procedure.

A. Game environment: MiniGate

We use the MiniGate environment, an open-source combat simulation of the *Baldur's Gate* games. All of our experiments are carried out on a single test problem, namely the duel of two wizards. This test problem was chosen because (a) there is a large variety of possible tactics; (b) there can be multiple different playing styles that are effective; and (c) the task is simple enough so that the results can be interpreted by humans relatively easily.

Both wizards are controlled by scripts. One of the wizards has a static, fixed script, while the other one is adaptive, and its script is updated by the macro-learning procedure. In all other respects, the two wizards' capabilities are equal. The behavior of a wizard is determined completely by its script: each time a decision needs to be made, the

rules are checked in order, and the first applicable rule is executed. If there are no applicable rules, the wizard uses his sling as default attack.

[Fig. 1 about here.]

B. Macros

Our aim is to create macros, action sequences of several steps long, that satisfy the requirements described in Subsection II-C. To make this task well-defined, we need to specify how the length of macros is determined and how we decide their starting and finishing conditions.

In the case of this specific computer RPG environment we can settle these questions relatively easily. In a typical battle, both wizards take about 6 to 9 decisions, after which they have no spells left and can fight only with slings until one of them dies. We decided that macro actions will be uniformly 3 actions long, so the adaptive agent can execute two macros, rounding off with several other actions at the end of the script. This decision is arbitrary but (a) it is reasonable according to our observations of the game; and (b) makes the segmentation problem trivial.

To simplify explanation, we define *opening* and *midgame phases* of a battle. The opening phase of a battle lasts until the third decision of the adaptive agent, while the midgame phase lasts from action 4 through 6. A total of $K = 30$ macros are to be learned, 15 for the opening and 15 for the midgame phase. An opening macro can only be used in the beginning of a script, and a midgame macro can be used only after an opening macro. Therefore, a valid script contains at most one of each.

Note that the procedure is extensible to games where there are more than two phases. However, to improve readability, we will not describe the algorithm in its full generality, but restrict ourselves to the two-phase case.

C. Script generation

The script generation routine is slightly different for the opening and the midgame macros. In both cases, full scripts are generated, from which we extract the corresponding macro using the method described in Subsection III-G. However, for the learning of the opening macros scripts are assembled from single rules, while for the learning of midgame macros, both opening macros and single rules may be used.

For the first case, our script generation method draws the rules from a fixed rulebase (shown in Appendix A) according to the actual probability distribution, and assembles a script from them.

Let the number of rules in the rulebase be M (in our case, $M = 24$). The probability vector $\mathbf{p}$ is an M-dimensional vector

$$\mathbf{p} = (p_1, \ldots, p_M)$$

where $p_i \in [0, 1]$ for all $i \in [1, \ldots, M]$, and $\sum_{i=1}^{M} p_i = 1$. The script generation procedure selects rule i with probability p_i. Different rules are drawn independently from each other. Script generation only determines whether a rule is included in the script or not, their order is fixed and pre-determined: the resulting script contains the rules in the same order as they appear in the rulebase (see [32] for an automatized approach for rule ordering).

Script generation for midgame macros is similar to the previous case, but the first three rules of the script will not be chosen independently but as the rules of an opening macro. The specific opening macro will be drawn randomly according to probability distribution

$$\mathbf{p}^o = (p_1^o, \ldots, p_{M^o}^o)$$

where M^o is the number of opening macros, $p_i^o \geq 0$ for all i, and $\sum_{i=1}^{M^o} p_i^o = 1$. Exactly one opening macro is selected; the probability of choosing the ith one is p_i^o. After the opening macro has been selected, we let the new script begin with the opening macro, then the consecutive rules are determined by the previous procedure. An example is shown in Figure 2.

[Fig. 2 about here.]

D. Script execution and generation of game records

To evaluate a script, the wizards play one battle. The adaptive wizard is controlled by the script to be tested, while the static wizard uses a fixed script. During training, this fixed script was always the "summoning tactic," (listed in Appendix B, Subsection B-A), which is highly effective against a single wizard. Note that the outcome of the battle is stochastic, even if both wizards' tactics are fixed: according to the game rules, all spell effects and inflicted damage have random factors.

At the end of battle, we gather data about the battle, and record

- the script of the adaptive wizard,
- the winner,
- the remaining hit points of the wizards at the end of battle,
- the duration of the battle (in game rounds),
- the ordered list of rules that were used by the adaptive wizard (note that this may be different from his script: some of the rules may never be selected, for instance because its conditions were not fulfilled, or the battle ended quickly).

The recorded data is used for fitness calculation, updating the parameters and extracting macros.

E. Fitness calculation

There are two sources of reward for the agent: he gains rewards for the diversity of his script, and for being an effective combatant. This corresponds to Schmidhuber's interestingness principle: firstly, we penalize "boring" scripts that lead to well-known areas (by giving a low reward for diversity); and secondly, we penalize "boring" scripts that lead to areas where it is hard to learn a good policy (by giving a low reward for playing strength). The "interesting" scripts in the intermediate area should get the highest fitness.

When calculating the reward for diversity, we compare the current script to all the previously extracted macros for the same phase. Let the number of such macros be K. Let the characteristic vector $\mathbf{v}_k$ of macro k ($1 \leq k \leq K$)

be an M-dimensional 0/1-vector, its jth component defined as

$$v_{k,j} := \begin{cases} 1, & \text{if macro } k \text{ contains rule } j; \\ 0, & \text{otherwise.} \end{cases}$$

Furthermore, consider the macro that we could extract from the current script, that is, either the first three rules applied (opening phase) or the three rules applied after finishing an opening macro (midgame phase). Denote its characteristic vector by $\mathbf{v}_0$. The reward of the script for diversity will be proportional to the difference of $\mathbf{v}_0$ from all the other characteristic vectors:

$$F_{div} := \frac{1}{K} \sum_{k=1}^{K} \|\mathbf{v}_k - \mathbf{v}_0\|.$$

Thus, if the rules of a macro are used in very few or none of the already established macros, it gets a high reward for diversity.

The second part of the reward comes from playing well. In order to evaluate how well the adaptive wizard fought, we use the fitness function of Spronck et al. [8] without modifications. The agent receives rewards for (i) winning the fight, (ii) in case of winning, for remaining as healthy as possible, (iii) in case of losing, for causing as much damage before dying as possible, and (iv) in case of losing, for staying alive as long as possible. We introduce the following notations: $h_0(A)$ and $h_T(A)$ denote the hitpoints of the adaptive wizard at the beginning and at the end of battle, respectively; $h_0(S)$ and $h_T(S)$ denote the same for the static wizard; D denotes the timestep when the adaptive wizard died (if ever), and D_{max} is a constant corresponding to 10 turns. The score of the adaptive wizard is defined as:

$$F_{str} := \begin{cases} 0.55 + 0.35 \frac{h_T(A)}{h_0(A)}, & h_T(A) > 0; \\ 0.1 \min\left(\frac{D}{D_{max}}, 1\right) + 0.1\left(1 - \frac{h_T(S)}{h_0(S)}\right) & \text{otherwise.} \end{cases}$$

We define the *overall fitness* of the script as a weighted sum[1] of the two rewards:

$$F := F_{str} + c \cdot F_{div}$$

We found experimentally that $c = 0.25$ is a suitable value for balancing the two terms. We use the measure F to guide the search in the space of possible scripts. Note that the fitness is highly stochastic, because the outcome of a battle depends on many random factors.

F. Parameter update: cross-entropy learning

Our goal is to update the probability vector $\mathbf{p}$ so that the chance of drawing high-fitness scripts increases. We start out with a uniform distribution. For updating we utilize the *cross-entropy method* (CEM), an efficient global optimization algorithm. Rubinstein [20] gives a detailed description of the algorithm, explaining its name, derivation

[1]Combining multiple optimization objectives is the subject of extensive research. However, if one wants to get an "optimum solution" in some sense, instead of the Pareto-front of non-dominated solution, then (as shown by Gábor et al. [33]) one must define a total ordering on the objective vectors, such as linear combination, lexicographic ordering, or an arbitrary scalar function of the objective vector. For the sake of simplicity, we chose linear combination.

and mechanism (see also Subsection II-E for an overview of its applications relevant to our topic). Here we resort to a short algorithmic description in the context of script-learning.

The cross-entropy method is a population-based algorithm. After evaluating a generation of samples, the best few percent of them are selected (the "elite samples") and used to update the probabilities. The new probability distribution is selected from a parametrized distribution family in such a way that the *cross-entropy distance* is minimized from the empirical distribution defined by the elite samples. For many distribution families, the minimum cross-entropy distribution can be expressed as a simple formula of the elite samples. Such "well-formed" families include all members of the *natural exponential family*, e.g., the Gaussian, Bernoulli, multinomial, exponential, gamma, Dirichlet, Poisson, and geometric distributions. For the sake of brevity, we shall only present the special case for the multinomial distributions (as this will be used in the paper). Note that the calculation of the cross-entropy distance appears only in the derivation of the learning rule, but not in the final algorithm, making the name somewhat misleading.

The pseudocode of CEM for script optimization is shown in Listing 2. The algorithm bears similarity to other global optimization methods (simulated annealing, ant colony optimization [34], evolution strategies [35], CMA-ES [36]), and the family of *estimation of distribution algorithms* including the compact genetic algorithm [37], population-based incremental learning[2] [38] and the univariate marginal distribution algorithm [39]. Our choice of algorithm was motivated by the conceptual simplicity of CEM, its theoretical foundations, and its robustness to meta-parameter choice [20]. However, we wish to point out that the choice of CEM is not crucial in the working of our macro learning scheme: in principle, any global optimization algorithms could be used that are able to maximize F over the space of scripts. The specific algorithm we used is provided here for the sake of reproducibility.

The cross-entropy method has three parameters: the population size N, the selection ratio ρ and the step-size α. Our settings were $N = 100$, $\rho = 0.1$ and $\alpha = 0.7$ (note that α is the per-episode learning rate; this corresponds to a per-instance learning rate of $\alpha' = \alpha/(\rho \cdot N) = 0.07$ for an on-line learning algorithm). CEM is quite insensitive to the choice of ρ and α: several preliminary experiments indicated that its performance was fairly uniform in the interval $0.5 \leq \alpha \leq 0.8$ and $0.05 \leq \rho \leq 0.25$. The running time of the algorithm is directly proportional to the population size N. However, a too low value of N may lead to sub-optimal solutions, so, in general, it should be set as high as the computational budget permits. In preliminary experiments, population sizes $N > 100$ did not visibly improve on the quality of solutions, so we settled for $N = 100$. We generate a total of 1500 samples, which means that 15 update iterations are carried out. We found this choice sufficient for converging to near-deterministic solutions.

G. Macro extraction

As the result of the script optimization procedure, we obtain a vector of probabilities containing one entry for each rule in the rulebase. For obtaining macros, we need to decide the probability that a particular rule was used

[2]Interesingly, PBIL has identical update rules to the special case of CEM we are considering in this paper.

Listing 2

```
procedure UpdateProbabilities(p, n, ScriptList, FitnessList);
% p: probability vector of rules
% n: number of trials so far
% ScriptList: list of scripts in previous trials
% FitnessList: list of fitnesses in previous trials

% CEM−specific parameters:
% N: population size
% ρ: selection ratio
% α: step size

M := length(p); % number of rules in the rulebase
if n mod N = 0 then begin % update with full population
   % Sort last N samples according to fitness, best ones first
   ScriptList := SortLastN( ScriptList, FitnessList, N);

   N_e := ρ · N; % number of elite samples
   % calculate frequencies of rules in the elite samples
   for j := 1 to M do
      p'_j := 0; % p': new probability vector
      for i := 1 to N_e do
         if ScriptList_i contains rule j then
            p'_j := p'_j + 1;
      p'_j := p'_j/N_e;

   % Update probability vector
   for j := 1 to M do
      p_j := (1 − α) · p_j + α · p'_j;
```

in the opening (or midgame) phase, and probabilities need to be discretized to 0 or 1. We proceed as follows.

Let the number of the macro to be extracted be k. Let $\mathbf{w}_1, \ldots, \mathbf{w}_N$ be the list of characteristic vectors of samples in the last iteration of CEM; that is, component j of vector $\mathbf{w}_i$ is 1 if rule j was fired during the corresponding phase of battle i, and 0 otherwise. Let the vector corresponding to macro k be

$$\mathbf{v}_k = \frac{1}{N} \sum_{i=1}^{N} \mathbf{w}_i$$

From this vector, we can easily extract a macro: we select the three rules with the largest values. Note that for the calculation of diversity (III-E) we use the non-integer vector $\mathbf{v}_k$, because it can be more informative than just the list of the three most-often-used rules.

H. Results

We ran the above-described algorithm for learning 15 opening-phase macros, and consecutively 15 midgame-phase ones. We repeated the experiment 3 times with different random number seeds. As a result, we obtained three 30-element macro sets.

An example of a learned opening-phase macro is

```
cast( "Monster Summoning I", closestenemy )
cast( "Deafness", closestenemy )
cast( "Blindness", closestenemy ),
```

This macro starts with summoning monsters around the enemy wizard, then deafening the enemy (so that his spells will fail with 50% probability) and blinding (so that his physical attacks will fail with high probability and his defense is lowered). Upon success, the combination of these spells makes the enemy wizard completely harmless and easy to kill. The full list of learned macros can be found in Appendix C.

Analyzing the obtained macros, we can discover several trivial dependencies based on the limited number of spells per level. For example, it is totally useless to put both "Monster Summoning I" and "Fireball" in a macro, as the wizard is able to cast only one level-3 spell. Furthermore, in the new search space of macros, the selection probabilities of individual rules are drastically altered: several spells are downweighted (for example, "Potion of Free Action" cannot be found in any of the macros, "Grease" is found only in 3 out of 30), while several of them are considerably upweighted (like "Monster Summoning I" in the opening macros and "Magic Missile" in the midgame macros). It is hard to point out strong dependencies like "spell A should be always followed by spell B" because the learned macro set contains the components of many different styles of tactics (for example, we can find fully defensive openings like #2, #5 and #15, and also fully offensive ones like #10). Nevertheless, we can find several interesting solutions, like the totally incapacitating opening macro #3 (the combination of "Monster Summoning I", "Blindness" and "Deafness" make very unlikely that the opponent wizard can ever finish a spell), and macro #11, which gives strong protection against all offensive spells (including "Monster Summoning I" and "Fireball"), then starts a strong counterattack.

Qualitatively, the learned macros have high diversity, although there are some very similar ones, too (for example, opening macros #8 and #14 are quite similar). The rule "`cast( Monster Summoning I, closestenemy)`"

seems to be the single most powerful spell against a lonely low-level wizard, as it occurs in 9 out of 15 opening macros. The damage that is caused by the summoned monsters is relatively low, but happens often, and makes spellcasting nearly impossible (a spell fizzles harmlessly if the wizard is hit during casting). It is notable that this rule does not occur in *all* of the macros, which is most likely due to the diversity-rewarding learning procedure. It is also notable that the frequency of the other rules is relatively balanced (though having significant fluctuations, as noted above).

It still remains to be seen whether it is possible to construct a diverse set of effective policies from these macros. We investigate this question in the next section.

IV. Applying Interesting macros for adaptive AI

In this section, we set out to actually use the macros that were learned in the previous section. The macros are added to the rulebase of dynamic scripting as new atomic actions. The section describes the experiments to investigate how the addition of these macros affects the performance of dynamic scripting. Specifically, we measure the changes in the speed of adaptation, playing strength and diversity. First, we give some details of the dynamic-scripting implementation (IV-A), then describe our experimental setup (IV-B) and the performance measures used (IV-C), and discuss the experimental results (IV-D).

A. Dynamic scripting implementation

Throughout the experiments, the behavior of the adaptive wizard was adapted by dynamic scripting. Spronck et al. [8] describe the algorithm in detail; here we recapitulate it only shortly. Dynamic scripting assigns weights to each rule in the rulebase. In the beginning of a battle, 10 rules are drawn randomly and assembled into a script. The probability that a rule is included in a script is proportional to its weight. Initially, all weights are set uniformly to 100. After each battle, weights are adjusted, but always kept within the range $[0, 1000]$. The adjustment is calculated as follows.

Let F_{str} (defined in Subsection III-E) be the score obtained by the adaptive player. Let $b = 0.3$ be the baseline score. This is greater than the score of any lost battle but lower than the score of any won battle. Let $P_{max} = 20$ and $R_{max} = 100$ be the maximum penalty and maximum reward, respectively (the ratio of P_{max} and R_{max} was selected in preliminary experiments). For each rule that has been executed, the weights are modified by

$$\Delta W = \begin{cases} -\left\lfloor P_{max} \dfrac{b - F_{str}}{b} \right\rfloor, & \text{if adaptive player lost;} \\ \left\lfloor R_{max} \dfrac{F_{str} - b}{1 - b} \right\rfloor, & \text{if adaptive player won.} \end{cases}$$

The weights of the remaining rules are raised or lowered evenly so that the sum of all weights remains constant.

Dynamic scripting can be easily extended to utilize the macros that were learned for the opening and midgame phases. To this end, we add the macros to the rulebase as extra rules, also assigning weights to them. A script will consist of one opening macro, one midgame macro and ten simple rules. All of them are chosen with probability proportional to their weights.

Note that most of the added simple rules can never be executed (and thus, cannot be rewarded or penalized) because the wizard will have reached his spell limit by the time these rules come into play. Therefore, the actual number does not really matter, we have chosen ten for the sake of simplicity.

B. Description of experiments

We compared two systems: dynamic scripting with the basic rulebase (DS-B) and dynamic scripting with the extended rulebase that also contains macros for the opening and midgame (DS-M).

The adaptive players were tested against three different static tactics:

Summoning tactic. The wizard summons monsters around his opponent. The monsters cause direct damage and interrupt the spells cast by the adaptive wizard with high probability. After that, the wizard throws various offensive spells. This tactic is the same that was used against the adaptive wizard during macro learning.

Offensive tactic. The wizard throws a fireball at its opponent and continues with various direct damage spells and disabling spells. This tactic represented strong play in earlier experiments with the MiniGate environment [8].

Optimized tactic. This is a script that was learned by dynamic scripting (DS-B), when trained against the Summoning tactic. This tactic is similar to the Summoning tactic, but it is much stronger: when the two tactics play against each other, the Optimized tactic wins over 65% of the time.

Novice tactic. This tactic tries to simulate a novice player's tactic (and is derived from Red Wizard A's behavior in the Novice tactic of Spronck [1]). This tactic was added to test the behavior of our learning algorithms against a fairly weak opponent.

All of these tactics are highly efficient except the last one, and in fact, they are hard to defeat even for a human player. Their description can be found in Appendix B.

For each of the 2×4 combinations of the adaptive and static opponents, 50 parallel runs were performed. Each run consisted of 500 battles.

C. Performance measures

We let the adaptive player fight against some static player. We wish to measure how many games are needed for the adaptive player to become consistently better. To this end, we use the *average turning point*, computed as follows. We record the scores of both players, averaged over 10 steps. If the average score of the adaptive player becomes higher than the static player's, and remains so for 10 steps, then we conclude that the turning point has been reached and the adaptive player is better. Low values for the average turning point are indicative for high efficiency of the algorithm. Note that this decision procedure implies that the lowest turning point we are able to measure is 20.

We also measured how strong the adaptive player became at the end of training. We quantify this by counting the battles won by the adaptive player during the last 100 battles out of 500 (the strategies typically stabilized long before the 400th battle). High values for the number of wins are indicative for high effectiveness of the algorithm.

At the end of a 500-battle long training epoch, dynamic scripting usually stabilizes. However, typically it does not converge to a deterministic script but rather to a distribution of scripts (because the randomness of rule selection is maintained). We measure the difference between the final distributions of different training epochs (note that from this point of view it is not relevant whether the distribution of a single epoch is sharply peaked or spread out, it is the difference between different epochs that matters). This quantifies whether there are many equally good optima to converge to, or just a few of them.

The number of different training epochs is denoted by N. For each epoch $i \in \{1, \ldots, N\}$, consider the last 100 battles (out of the 500). For each rule $k \in \{1, \ldots, M\}$, its frequency of application by $p_{i,k}$ is noted. Furthermore, let

$$\mathbf{p}_i := (p_{i,1}, \ldots, p_{i,M}).$$

We define the diversity D as the average of pairwise distances between the $\mathbf{p}_i$ vectors:

$$D := \frac{\sum_{1 \leq i < i' \leq N} \|\mathbf{p}_i - \mathbf{p}_{i'}\|}{N(N-1)/2}$$

High values for D are indicative for a high diversity.

D. Experimental results

The results of our experiments are summarized in Tables I to III, and visualized in Fig 3.[3] The DS-B method learns to defeat the Offensive tactic quickly and consistently, and performs reasonably well against the Summoning tactic. It is much less effective against the Optimized tactic. This result may seem surprising: in principle, the method should be able to learn a tactic with a win ratio of at least 50% (by selecting exactly the same rules that constitute the Optimized tactic). However, in the beginning of the learning process a positive reinforcement comes too rarely, therefore the chance is low that the appropriate rules get reinforced.

[TABLE 1 about here.]

[TABLE 2 about here.]

[TABLE 3 about here.]

[Fig. 3 about here.]

The trends are the same for DS with macros, but the results are uniformly better regarding both the time needed for adaptation and the quality of the learned tactic. DS-M is able to reach a win ratio close to 50% even against the Optimized tactic. Furthermore, DS-M is able to increase the efficiency of adaptation and the win ratio parallel to an increase in the diversity of learned policies. The results against the Novice tactic are particularly interesting:

[3]According to the Mann-Whitney-Wilcoxon rank-sum test, all the differences between turning points are significant on a 5% level, except for the Offensive tactics. The differences in winning ratios are significant in all cases (in fact, they are significant even on a 0.1% error level).

apart from the slightly worse average turning point and slightly better win ratio, DS-M reached approximately the same diversity level as against the other, stronger tactics. This is in sharp contrast with the diversity loss of DS-B.

To get an interpretation of the diversity results, consider the two extremes: if convergence was always to the same solution, diversity was 0. On the other hand, the diversity of a population that is drawn according to the initial distribution, i.e., each rule is included with equal probability, the diversity measure is 5.82 (note that the rules are not necessarily *applied* with equal probability, because there may be rules that are included in the script but are not executed). In light of these values, we can conclude that DS-M learns considerably more diverse tactics than DS-B.

V. Discussion

We have proposed a method for learning diverse but effective macros that can be used as components of game AI scripts. We demonstrated that the macros learned this way can increase adaptivity: the dynamic scripting technique that uses these macros is able to learn scripts that are both more effective and more diverse than dynamic scripting that uses rulebases consisting of singular rules. A likely reason is that macro actions are constructed in such a way that they can take large steps in various (but sensible) directions. This reduction in online adaptation time is gained at the expense of offline training time.

Our demonstrations were performed in a CRPG simulation with two wizards duelling, but our approach is readily applicable for other script-controlled game AIs. The learned macros can be used either by an adaptive system such as dynamic scripting, or by game developers to speed up the construction of new AI scripts.

A. Scalability

For practical applications, scalability is a critical issue. First of all, note that primitive commands need not be "primitive", they can be arbitrarily complex actions. A good example of this is Neverwinter Nights [1], where primitive commands like "*use offensive magic at an enemy that attacks from a distance, preferably a spellcaster*" are actually functions having several dozen lines of NWNscript. A script of 6-10 primitive actions per agent was sufficient to construct strong group strategies.

It is clear that macros reduce the search space considerably. To get an idea about the extent of reduction, we make some rough calculations: assume there are k primitive actions, of which $m << k$ are used to construct a script. Then, dynamic scripting has to choose from $\binom{k}{m} \approx k^m$ possible scripts. Let us create M groups of macros, and for the purposes of our rough calculations, assume that in each group, there are approximately k macros. This gives a search space of roughly k^M macros, so the reduction factor is $\approx k^{m-M}$. Note that the extent of search space reduction is independent on the quality of the macros, it depends only on their numbers.

Of course, if the macros have insufficient quality (for example, they are too similar, or are composed of bad moves), then the reduced search space will not contain any strong strategies. As our main contribution, we proposed a method for learning high-quality macros, that are both useful and diverse.

B. Relationship to other decision-making mechanisms

The macro generation technique demonstrated in this paper is closely tied to dynamic scripting, and consequently, to scripting. There are several other classic decision making mechanisms, like finite state machines and decision trees/decision lists, that are equivalent to scripts in expressive power. While the use of macros within these mechanisms has a rich literature, generalizing our interestingness-based macro generation method to these domains is far from trivial, and we are not aware of any work in similar vein.

Recently, Goal-Oriented Behavior (GOB) and Goal-Oriented Action Planning (GOAP) gained popularity amongst game developers. These terms cover a wide area of techniques, some of which are quite similar to regular scripting, in which characters in games choose actions based on their immediate needs [2]. Examples of games which use such techniques are *The Sims*, *F.E.A.R.*, *S.T.A.L.K.E.R.: Shadow of Chernobyl*, and *Empire: Total War* [40], [41].

Plans generated by a GOAP system are equivalent in expressive ability to scripts. However, there is an important difference: the system can generate new plans before every combat (or even many times during a single combat). A GOAP system will therefore, in general, generate more diverse behavior than a straightforwardly scripted system. However, the actions used to achieve the goals by a regular GOAP system are still static.

It might be interesting to combine a GOAP system with dynamic scripting. A possible approach is to consider an action sequence needed to achieve a certain goal as a macro, and apply the techniques discussed in the present paper to learn a variety of diverse but effective macros for each goal. Agents could then adapt automatically, maintaining diversity and effectiveness, by using a dynamic scripting approach to select an action sequence macro for each of an agent's goals.

C. Conclusions

In this paper we described a method for automatic macro generation. A defining criterion for macro generation was that the resulting macros should lead to interesting behavior. In accordance with previous literature, we defined interestingness as a behavior that is both effective and sufficiently different from previously tried behaviors. The obtained macros were plugged into the dynamic scripting algorithm.

We performed experiments in a simple RPG combat simulation. Macros reduced the search space considerably, speeding up the adaptation rate of dynamic scripting. As the main result of the paper, we showed that the use of interesting macros in dynamic scripting is able to raise both the eventual winning ratio and the variety of applied tactics. We believe this was possible because macro learning was driven by an interestingness measure that takes into account both effectiveness and diverse playing style.

APPENDIX A
RULEBASE

This section provides the complete rulebase, consisting of 24 rules, used by the adaptive wizard. The scripting language is fully described by Spronck [1].

```
if healthpercentage < 50 then
   drink( "Potion of Healing" );
if locatedin( "Nauseating Fumes" ) then
   drink( "Potion of Free Action" );
drink( "Potion of Fire Resistance" );
cast( "Monster Summoning I", closestenemy );
cast( "Mirror Image" );
cast( "Hold Person", closestenemy );
cast( "Fireball", closestenemy );
cast( "Blindness", closestenemy );
cast( "Deafness", closestenemy );
cast( "Strength" );
cast( "Luck" );
cast( "Shield" );
cast( "Blur" );
cast( "Ray of Enfeeblement", closestenemy );
cast( "Stinking Cloud", closestenemy );
cast( "Grease", closestenemy );
cast( "Chromatic Orb", closestenemy );
cast( "Flame Arrow", closestenemy );
cast( "Magic Missile", closestenemy );
cast( "Melf's Acid Arrow", closestenemy );
cast( "Larloch''s Minor Drain", closestenemy );
cast( "Shocking Grasp", closestenemy );
cast( "Charm Person", closestenemy );
rangedattack( closestenemy );
```

APPENDIX B
TACTICS FOR THE STATIC WIZARD

This section provides details of the four different tactics used by the static wizard, functionally described in Subsection IV-B. The are the Summoning tactic (B-A), the Offensive tactic (B-B), the Optimized tactic (B-C), and the Novice tactic (B-D).

A. Summoning tactic

```
if healthpercentage < 50 then
   drink( "Potion of Healing" );
cast( "Mirror Image" );
cast( "Monster Summoning I", centreenemy );
cast( "Shield" );
cast( "Larloch's Minor Drain", closestenemy );
rangedattack( closestenemy );
```

B. Offensive tactic

```
if healthpercentage < 50 then
   drink( "Potion of Healing" );
cast( "Mirror Image" );
if not closestenemy.influence("Mirrored") then
   cast( "Fireball", closestenemy );
```

```
cast( "Chromatic Orb", closestenemy );
cast( "Magic Missile", closestenemy );
cast( "Shield" );
cast( "Blindness", closestenemy );
cast( "Melf's Acid Arrow", closestenemy );
rangedattack( closestenemy );
```

C. Optimized tactic

```
if healthpercentage < 50 then
   drink( "Potion of Healing" );
cast( "Mirror Image" );
cast( "Monster Summoning I", closestenemy );
cast( "Blur" );
cast( "Shield" );
cast( "Chromatic Orb", closestenemy );
cast( "Larloch's Minor Drain", closestenemy );
cast( "Charm Person", closestenemy );
rangedattack( closestenemy );
```

D. Novice tactic

```
if healthpercentage < 50 then
   drink( "Potion of Healing" );
cast( "Hold Person", closestenemy );
cast( "Mirror Image" );
if not closestenemy.influence( freezinginfluence ) then
   cast( "Stinking Cloud", defaultenemy );
cast( "Magic Missile", closestenemy( "Wizard" ) );
cast( randomoffensive, randomenemy );
rangedattack( closestenemy );
```

APPENDIX C
MACROS LEARNED BY OUR ALGORITHM

We list here the macros learned during one of the training runs. For the other two training runs, results look similar, so they are not shown here (naturally, we used each of the three macro sets for the quantitative evaluation). The opening-phase macros learned by our algorithm were as follows.

```
[macro #1]
 cast( "Monster Summoning I", closestenemy );
 cast( "Melf's Acid Arrow", closestenemy );
 cast( "Charm Person", closestenemy );

[macro #2]
 cast( "Mirror Image" );
 cast( "Hold Person", closestenemy );
 cast( "Shield" );

[macro #3]
 cast( "Monster Summoning I", closestenemy );
 cast( "Deafness", closestenemy );
 cast( "Blindness", closestenemy );
```

```
[macro #4]
 cast( "Monster Summoning I", closestenemy );
 cast( "Grease", closestenemy );
 cast( "Stinking Cloud", closestenemy );

[macro #5]
 if healthpercentage < 50 then drink( "Potion of Healing" );
 drink( "Potion of Fire Resistance" );
 cast( "Charm Person", closestenemy );

[macro #6]
 cast( "Monster Summoning I", closestenemy );
 cast( "Blur" );
 cast( "Magic Missile", closestenemy );

[macro #7]
 cast( "Fireball", closestenemy );
 cast( "Luck" );
 cast( "Larloch's Minor Drain", closestenemy );

[macro #8]
 cast( "Monster Summoning I", closestenemy );
 cast( "Ray of Enfeeblement", closestenemy );
 cast( "Strength" );

[macro #9]
 cast( "Stinking Cloud", closestenemy );
 cast( "Shield" );
 cast( "Deafness", closestenemy );

[macro #10]
 cast( "Monster Summoning I", closestenemy );
 cast( "Chromatic Orb", closestenemy );
 cast( "Shocking Grasp", closestenemy );

[macro #11]
 drink( "Potion of Fire Resistance" );
 cast( "Mirror Image" );
 cast( "Monster Summoning I", closestenemy );

[macro #12]
 if healthpercentage < 50 then drink( "Potion of Healing" );
 cast( "Fireball", closestenemy );
 cast( "Charm Person", closestenemy );

[macro #13]
 cast( "Monster Summoning I", closestenemy );
 cast( "Deafness", closestenemy );
 cast( "Melf's Acid Arrow", closestenemy );

[macro #14]
 cast( "Monster Summoning I", closestenemy );
```

```
 cast( "Strength" );
 cast( "Blindness", closestenemy );

[macro #15]
 cast( "Hold Person", closestenemy );
 cast( "Grease", closestenemy );
 cast( "Blur" );
```

The midgame-phase macros learned were as follows.

```
[macro #1]
 cast( "Mirror Image" );
 cast( "Grease", closestenemy );
 cast( "Strength" );

[macro #2]
 cast( "Ray of Enfeeblement", closestenemy );
 cast( "Magic Missile", closestenemy );
 cast( "Shocking Grasp", closestenemy );

[macro #3]
 cast( "Hold Person", closestenemy );
 cast( "Blindness", closestenemy );
 cast( "Magic Missile", closestenemy );

[macro #4]
 cast( "Ray of Enfeeblement", closestenemy );
 cast( "Larloch's Minor Drain", closestenemy );
 cast( "Shocking Grasp", closestenemy );

[macro #5]
 cast( "Flame Arrow", closestenemy );
 cast( "Magic Missile", closestenemy );
 cast( "Larloch's Minor Drain", closestenemy );

[macro #6]
 cast( "Hold Person", closestenemy );
 cast( "Chromatic Orb", closestenemy );
 cast( "Shocking Grasp", closestenemy );

[macro #7]
 cast( "Ray of Enfeeblement", closestenemy );
 cast( "Blindness", closestenemy );
 cast( "Chromatic Orb", closestenemy );

[macro #8]
 cast( "Blindness", closestenemy );
 cast( "Magic Missile", closestenemy );
 cast( "Charm Person", closestenemy );

[macro #9]
 cast( "Flame Arrow", closestenemy );
 cast( "Magic Missile", closestenemy );
 cast( "Charm Person", closestenemy );
```

```
[macro #10]
 cast( "Larloch's Minor Drain", closestenemy );
 cast( "Shocking Grasp", closestenemy );
 cast( "Charm Person", closestenemy );

[macro #11]
 cast( "Blindness", closestenemy );
 cast( "Larloch's Minor Drain", closestenemy );
 cast( "Charm Person", closestenemy );

[macro #12]
 cast( "Ray of Enfeeblement", closestenemy );
 cast( "Magic Missile", closestenemy );
 cast( "Shocking Grasp", closestenemy );

[macro #13]
 cast( "Ray of Enfeeblement", closestenemy );
 cast( "Magic Missile", closestenemy );
 cast( "Shocking Grasp", closestenemy );

[macro #14]
 cast( "Hold Person", closestenemy );
 cast( "Blindness", closestenemy );
 cast( "Magic Missile", closestenemy );

[macro #15]
 cast( "Luck" );
 cast( "Blindness", closestenemy );
 cast( "Magic Missile", closestenemy );
```

ACKNOWLEDGMENTS

The first author is supported by the Eötvös grant of the Hungarian State. The second author is sponsored by the Interactive Collaborative Information Systems (ICIS) project, supported by the Dutch Ministry of Economic Affairs, grant nr: BSIK03024. The third author is supported by a grant from the Dutch Organisation for Scientific Research (NWO grant 612.066.406).

REFERENCES

[1] P. Spronck, *Adaptive Game AI*. Maastricht University Press, 2005.

[2] I. Millington, *Artificial Intelligence for Games*. Morgan Kaufmann, 2006.

[3] A. Nareyek, "Intelligent agents for computer games," in *Computers and Games, Second International Conference, CG 2000*, ser. Lecture Notes in Computer Science, T. Marsland and I. Frank, Eds., vol. 2063. Heidelberg, Germany: Springer-Verlag, 2002, pp. 414–422.

[4] S. L. Tomlinson, "Working at thinking about playing or a year in the life of a games AI programmer," in *Proceedings of the 4th International Conference on Intelligent Games and Simulation (GAME-ON 2003)*, Q. Mehdi, N. Gough, and S. Natkin, Eds. Ghent, Belgium: EUROSIS, 2003, pp. 5–12.

[5] P. Tozour, "The perils of AI scripting," in *AI Game Programming Wisdom*, S. Rabin, Ed. Hingham, MA: Charles River Media, Inc., 2002, pp. 541–547.

[6] M. Brockington and M. Darrah, "How *not* to implement a basic scripting language," in *AI Game Programming Wisdom*, S. Rabin, Ed. Hingham, MA: Charles River Media, Inc., 2002, pp. 548–554.

[7] P. Spronck, I. Sprinkhuizen-Kuyper, and E. Postma, "Online adaptation of computer game opponent AI," in *Proceedings of the 15th Belgium-Netherlands Conference on Artificial Intelligence*, 2003, pp. 291–298.

[8] P. Spronck, M. Ponsen, I. Sprinkhuizen-Kuyper, and E. Postma, "Adaptive game AI with dynamic scripting," *Machine Learning*, vol. 63, no. 3, pp. 217–248, 2006.

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

[10] L. Bull and T. Kovacs, *Foundations of Learning Classifier Systems*. Springer, 2005, ch. Foundations of Learning Classifier Systems: An Introduction, pp. 3–14.

[11] M. Ponsen, H. Muñoz-Avila, P. Spronck, and D. W. Aha, "Automatically acquiring adaptive real-time strategy game opponents using evolutionary learning," in *Proceedings of the 17th Innovative Applications of Artificial Intelligence Conference*, 2005.

[12] A. G. Barto and S. Mahadevan, "Recent advances in hierarchical reinforcement learning," *Discrete Event Dynamic Systems*, vol. 13, no. 4, pp. 341–379, 2003.

[13] A. McGovern, "Autonomous discovery of temporal abstractions from interaction with an environment," Ph.D. dissertation, University of Massachusetts, 2002.

[14] B. Bakker and J. Schmidhuber, "Hierarchical reinforcement learning based on subgoal discovery and subpolicy specialization," in *Proceedings of the 8-th Conference on Intelligent Autonomous Systems*, 2004, pp. 438–445.

[15] S. P. Singh, A. G. Barto, and N. Chentanez, "Intrinsically motivated reinforcement learning," in *Advances in Neural Information Processing Systems 17*, 2005.

[16] M. A. Wiering, "Explorations in efficient reinforcement learning," Ph.D. dissertation, Universiteit van Amsterdam, 1999.

[17] J. Schmidhuber, "Curious model-building control systems," in *Proceedings of the International Joint Conference on Neural Networks*, 1991, pp. 1458–1463.

[18] ——, "Exploring the predictable," in *Advances in Evolutionary Computing*, S. Ghosh and S. Tsutsui, Eds. Springer, 2002, pp. 579–612.

[19] P.-Y. Oudeyer and F. Kaplan, "The discovery of communication," *Connection Science*, vol. 18, no. 2, 2006.

[20] R. Y. Rubinstein, "The cross-entropy method for combinatorial and continuous optimization," *Methodology and Computing in Applied Probability*, vol. 1, pp. 127–190, 1999.

[21] H. Muehlenbein, "The equation for response to selection and its use for prediction," *Evolutionary Computation*, vol. 5, pp. 303–346, 1998.

[22] P.-T. de Boer, D. P. Kroese, S. Mannor, and R. Y. Rubinstein, "A tutorial on the cross-entropy method," *Annals of Operations Research*, vol. 134, pp. 19–67, 2004.

[23] A. Costa, O. D. Jones, and D. P. Kroese, "Convergence properties of the cross-entropy method for discrete optimization," *Operations Research Letters*, 2007, to appear.

[24] G. Allon, D. P. Kroese, T. Raviv, and R. Y. Rubinstein, "Application of the cross-entropy method to the buffer allocation problem in a simulation-based environment," *Annals of Operations Research*, vol. 134, pp. 137–151, 2005.

[25] J. Keith and D. P. Kroese, "Sequence alignment by rare event simulation," in *Proceedings of the 2002 Winter Simulation Conference*, 2002, pp. 320–327.

[26] Z. Szabó, B. Póczos, and A. Lőrincz, "Cross-entropy optimization for independent process analysis," in *ICA*, 2006, pp. 909–916.

[27] F. Dambreville, "Cross-entropic learning of a machine for the decision in a partially observable universe," *Journal of Global Optimization*, 2006, to appear.

[28] I. Menache, S. Mannor, and N. Shimkin, "Basis function adaptation in temporal difference reinforcement learning," *Annals of Operations Research*, vol. 134, no. 1, pp. 215–238, 2005.

[29] S. Mannor, R. Y. Rubinstein, and Y. Gat, "The cross-entropy method for fast policy search," in *20th International Conference on Machine Learning*, 2003.

[30] I. Szita and A. Lőrincz, "Learning Tetris using the noisy cross-entropy method," *Neural Computation*, vol. 18, no. 12, pp. 2936–2941, 2006.

[31] ——, "Learning to play using low-complexity rule-based policies: Illustrations through Ms. Pac-Man," *Journal of Articial Intelligence Research*, vol. 30, pp. 659–684, 2006.

[32] T. Timuri, P. Spronck, and J. van den Herik, "Automatic rule ordering for dynamic scripting," in *The Third Artificial Intelligence and Interactive Digital Entertainment Conference*, 2007, pp. 49–54.

[33] Z. Gábor, Zsolt Kalmár, and Csaba Szepesvári, "Multi-criteria reinforcement learning," in *Proceedings of the 15th International Conference on Machine Learning*, 1998, pp. 197–205.

[34] M. Dorigo and L. M. Gambardella, "Ant colony optimization: a new meta-heuristic," in *Congress on Evolutionary Computation*, vol. 2, 1999.

[35] H.-G. Beyer and H.-P. Schwefel, "Evolution strategies–a comprehensive introduction," *Natural Computing*, vol. 1, no. 1, pp. 3–52, 2002.

[36] N. Hansen and A. Ostermeier, "Completely derandomized self-adaptation in evolution strategies," *Evolutionary Computation*, vol. 9, no. 2, pp. 159–195, 2001.

[37] G. R. Harik, F. G. Lobo, and D. E. Goldberg, "The compact genetic algorithm," *IEEE Transactions on Evolutionary Computation*, vol. 3, no. 4, pp. 287–297, 1999.

[38] S. Baluja, "Population-based incremental learning: A method for integrating genetic search based function optimization and competitive learning," Carnegie Mellon University Pittsburgh, PA, USA, Tech. Rep., 1994.

[39] Q. Zhang, "On stability of fixed points of limit models of univariate marginal distribution algorithm and factorized distribution algorithm," *IEEE Transactions on Evolutionary Computation*, vol. 8, no. 1, pp. 80–93, 2004.

[40] J. Orkin, "3 states and a plan: The A.I. of F.E.A.R." in *Game Developers Conference*, 2006.

[41] E. Long, "Enhanced NPC behaviour using goal oriented action planning," Master's thesis, University of Abertay Dundee, 2007.

LIST OF FIGURES

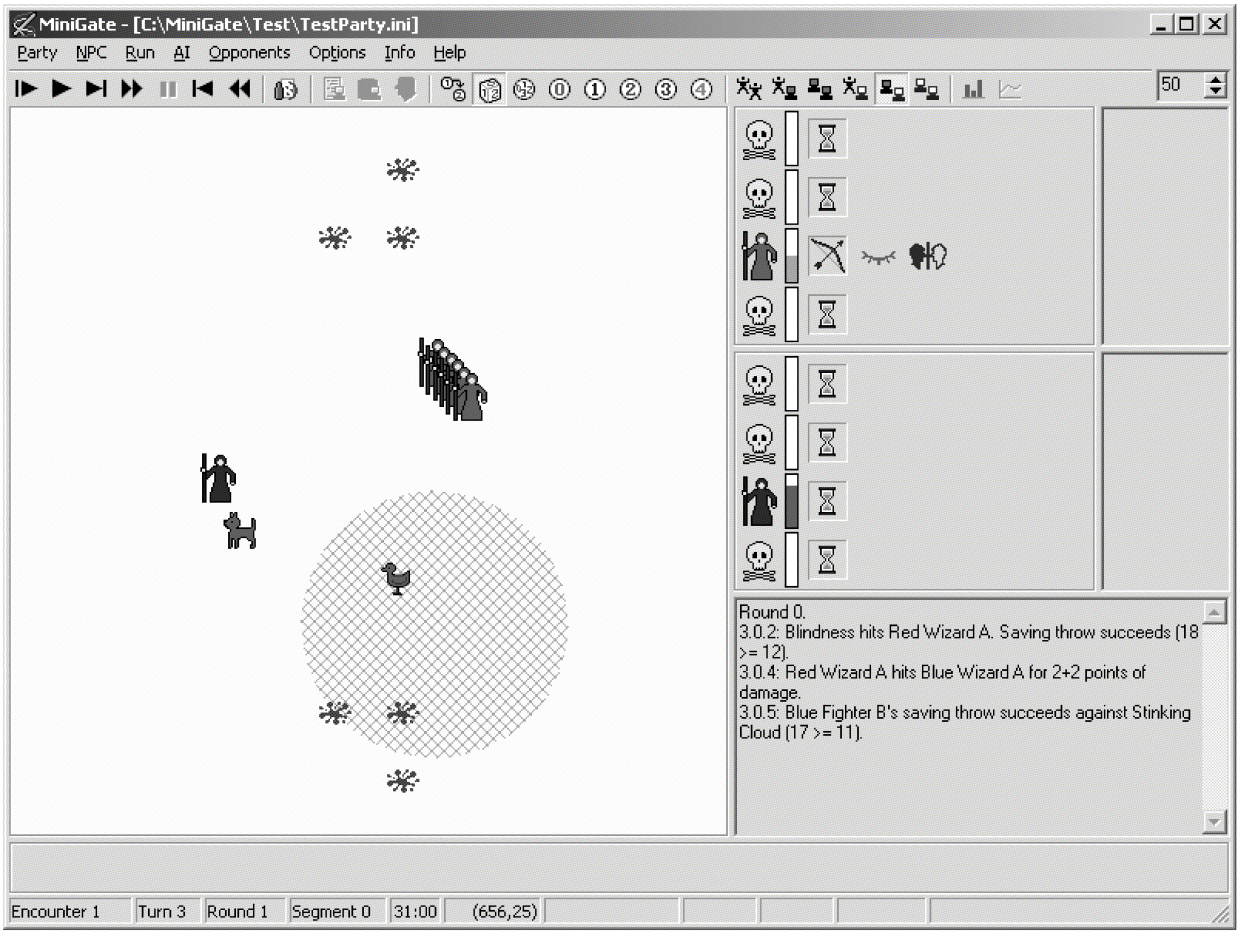
MiniGate - [C:\MiniGate\Test\TestParty.ini]
Party NPC Run AI Opponents Options Info Help
50
Round 0.
3.0.2: Blindness hits Red Wizard A. Saving throw succeeds (18 >= 12).
3.0.4: Red Wizard A hits Blue Wizard A for 2+2 points of damage.
3.0.5: Blue Fighter B's saving throw succeeds against Stinking Cloud (17 >= 11).
Encounter 1
Turn 3
Round 1
Segment 0
31:00
(656,25)

Fig. 1

```
[ opening macro #7 ]
 cast( "Monster Summoning I", closestenemy );
 cast( "Magic Missile", closestenemy );
 cast( "Chromatic Orb", closestenemy );
[ simple rules ]
 if healthpercentage < 50 then
    drink( "Potion of Healing" );
 cast( "Fireball", closestenemy );
 cast( "Luck" );
 cast( "Stinking Cloud", closestenemy );
 cast( "Grease", closestenemy );
 cast( "Flame Arrow", closestenemy );
 cast( "Magic Missile", closestenemy );
```

Fig. 2

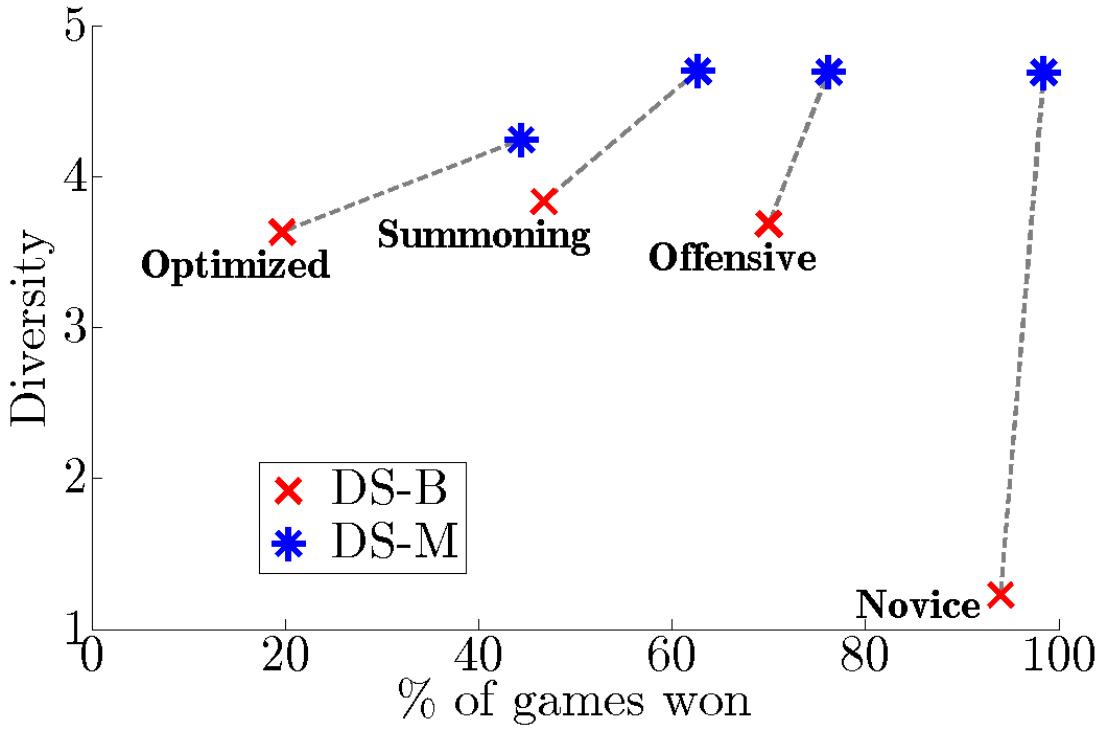
Diversity
% of games won
Optimized
Summoning
Offensive
Novice
DS-B
DS-M
1
2
3
4
5
0
20
40
60
80
100

Fig. 3

List of Tables

TABLE I

	DS-B	DS-M
Summoning tactic	132.3 (0)	105.0 (0)
Optimized tactic	>427.1 (31)	>265.6 (2)
Offensive tactic	42.1 (0)	35.7 (0)
Novice tactic	20.2 (0)	23.4 (0)

TABLE II

	DS-B	DS-M
Summoning tactic	46.7	62.7
Optimized tactic	19.7	44.4
Offensive tactic	70.0	76.2
Novice tactic	94.0	98.4

TABLE III

	DS-B	DS-M
Summoning tactic	3.84	4.70
Optimized tactic	3.63	4.24
Offensive tactic	3.68	4.69
Novice tactic	1.22	4.69