Goal-Directed Hierarchical Dynamic Scripting for RTS Games

Anders Dahlbom & Lars Niklasson

School of Humanities and Informatics
University of Skövde, Box 408, SE-541 28 Skövde, Sweden
anders.dahlbom@his.se & lars.niklasson@his.se

Abstract

Learning how to defeat human players is a challenging task in today's commercial computer games. This paper suggests a goal-directed hierarchical dynamic scripting approach for incorporating learning into real-time strategy games. Two alternatives for shortening the re-adaptation time when using dynamic scripting are also presented. Finally, this paper presents an effective way of throttling the performance of the adaptive artificial intelligence system. Put together, the approach entails the possibility of an artificial intelligence opponent to be challenging for a human player, but not too challenging.

Introduction

The quality of the opponent intelligence in modern computer games primarily comes from the ability of artificial intelligence opponents (AIOs) to exert human-like behavior (Spronck, et al. 2003). A cornerstone of human behavior is learning; humans are able to quickly adapt to and cater for changing situations. This is valid for players of computer games too; they are able to quickly identify and exploit the behavior of the opponent intelligence. We argue that the entertainment value of a computer game can be increased by allowing AIOs to adapt to the opponent behavior, i.e. to the behavior of the human player.

Learning how to *defeat* human players might however raise issues when incorporated into commercial computer games, as the goal is for the player to win (Lidén 2003), but not too easily (Woodcock, et al. 2000). Hence, an AIO needs to be carefully balanced.

Balancing the performance of an AIO is however not a trivial task, as the expertise amongst game players varies. Learning paradigms also usually aim at reaching the best available performance. Therefore, methods for balancing AIOs might be of even more importance, as a game should be challenging for both novice and expert players.

This paper extends the approach of dynamic scripting (Spronck, et al. 2003), by adding a goal-directed ability as a means for enabling fast learning in RTS games. Two alternatives for shortening re-adaptation times are also presented. Finally, this paper presents an efficient approach for throttling the performance of adaptive AIOs.

RTS games

In real-time strategy games two or more players fight each other on a battlefield, where each player is in control of an army. An army usually consists of various combat units and structures for managing the war, such as: training facilities, in-game research facilities, stationary defenses, and resource gathering centers. A vital component in many RTS games is resources such as gold and supplies. These need to be gathered and allocated in order to construct a base to operate from and in order to create combat units.

The key to victory in an RTS game often depends on two factors: good resource management and strategic thinking. Everything comes to a certain cost and resources can be of limited amounts. Therefore, an efficient flow and good allocation of resources is needed. Furthermore, tactical and strategic decisions are needed for how to defeat the opponents. Advantages in the terrain need to be found and weaknesses of enemies need to be spotted. Together, these advantages and weaknesses can be used to implement a good strategy for victory.

AI in RTS games

An AIO in an RTS game faces similar tasks as a human player. In order to appear intelligent it might need to create one cohesive strategy for victory. The AI system in an RTS game can be compared with how real-world armies operate. At the top, the commander-in-chief decides on a grand plan based on doctrines, reports etc. This plan is propagated through the chain of command down to regiments or similar units, which execute different parts of the plan. Eventually, orders reach the lower levels where individual soldiers contribute with their part to the plan.

Similarly, an AIO can be structured in a hierarchical fashion stretching from strategic and tactical warfare to individual unit combat. At the top, resources need to be collected and managed efficiently. The order in which to produce buildings and units also needs to be scheduled efficiently. Furthermore, Buro & Furtak (2004) state that both spatial- and temporal reasoning is of great importance. Temporal reasoning is concerned with how actions relate to each other over time, and spatial reasoning is concerned with analyzing the environment. Forbus et al. (2001), point out the importance of exploiting the terrain in

war games. Key positions need to be found for where to put defenses and for where to attack. Moreover, it can be of importance to detect movement patterns of enemies in order to place defenses strategically and to appear intelligent.

The battlefield in an RTS game is usually unknown from the beginning, and players need to explore it to find resource locations and key positions in the environment. Even though the world has been explored, or if its structure is known in advance, regions not seen by friendly units are usually covered by a fog of war (FOW). Considering that an RTS game is a dynamic environment the view of the world for one player might not be completely true, as other players might have changed it. Therefore, an AIO needs to be capable of making decisions under uncertainty. A model might need to be established for how the opponents play and what their intentions are.

Several players are also allowed to team up against common enemies. AIOs might therefore need to be able to collaborate with each other, as well as with human players.

It can be important to combine many of these aspects and create a plan which also considers future situations that might occur. In order to achieve the longer-term goal of victory, a plan might also need to include objectives that are not directly profitable, or even unprofitable, in the near future. In the end, everything however needs to be executed through issuing low-level commands that control the behavior of each individual unit.

At the lower levels, the main task for the AI system is pathfinding. It might however also need to possess the capabilities of unit coordination and situation analysis, in order for the units to appear intelligent. Even though the AI system can be quite complex, shortcuts are allowed as it is, in the end, the entertainment value that counts.

Dynamic scripting

Dynamic scripting (Spronck, et al. 2003) is a technique for achieving online adaptation of computer game opponents. In dynamic scripting, scripts are created online, i.e. during game-play, based on rules extracted from a rulebase. The technique is based on reinforcement learning and adaptation proceeds by rewarding or punishing certain rules according to their influence on the outcome.

Originally, dynamic scripting was used to create scripts for opponent parties in computer role-playing games (CRPGs). Before an encounter between the party of a human player and an opponent party, controlled by the AI, rules are extracted to govern the behavior of the opponent party. All rules in a rulebase are associated with weights which determine the probability that they are extracted and used in a script. Rules are evaluated when an encounter has ended and their weights are updated according to the outcome of the encounter.

A fitness function is used to calculate fitness values for all rules during the adaptation process. The fitness values are based on the contribution to the outcome and they are used to calculate new weights. This is handled by a weight-update function which maps fitness values to weight changes. Finally, a weight redistribution function is applied so that the total weight-sum remains constant. Hence, if one weight is increased, then other weights are decreased.

A cornerstone of dynamic scripting is that it is based on domain knowledge. Domain knowledge is used when rules are created, as the designer has knowledge of the domain. Domain knowledge is also used to separate rules during run-time; rules for controlling a warrior are different from rules for controlling a wizard. Different rulebases are therefore created for each character type in a CRPG.

The fact that the rules are manually designed is very attractive from a game developer's perspective, as the quality assurance phase becomes easier. Moreover, the behavior of AIOs in an RTS game is often determined by scripts (Spronck, et al. 2002). Spronck (2005) also states that dynamic scripting achieves eight demands that can be needed to successfully implement online learning in computer games: speed, effectiveness, robustness, efficiency, clarity, variety, consistency, and scalability. Therefore, dynamic scripting should be suitable for achieving adaptive behavior in RTS games.

Dynamic Scripting in RTS games

According to Ponsen & Spronck (2004), dynamic scripting is not directly applicable to RTS games due to the differences between scripts for CRPGs and RTS games. Ponsen & Spronck (2004) has however applied a modified dynamic scripting algorithm to an RTS game, which mainly differs with regard to two aspects:

1. Instead of separating rules with respect to different opponent types (warrior, wizard, etc.), rules are separated with respect to different game states.
2. Rules for an AIO are adapted when a state change occurs and rules are evaluated with respect to the fitness for the previous state and the fitness for the whole game. In the original dynamic scripting algorithm (Spronck, et al. 2003), rules are evaluated after each encounter between opponent parties.

Ponsen & Spronck (2004) separate states based on what type of buildings that are available in the game at any point in time, since each building allows for various kinds of in-game actions. Therefore, a state change occurs every time a new building is constructed. For example, if a heavy weapons factory is constructed, then tanks and artillery can be built. If the factory is destroyed, then heavy weapons cannot be constructed any more and rules associated with these are useless. On top of this, Ponsen & Spronck implemented a loop which was used to continuously launch attacks against the opponent player.

Extending dynamic scripting

In this section, a goal-directed hierarchical approach for extending the dynamic scripting algorithm (Spronck, et al. 2003) is presented. We argue that two main advantages can be gained by introducing a goal-directed component:

1. The illusion of intelligence can be strengthened given that: (1) it is important that agents in computer games seem to possess some intelligence (Laird 2000), and (2) the most important aspect of an agent's intelligence is its goal-directed component (Nareyek 2002).
2. The complex domain knowledge possessed by human designers can easily be translated to individual goals and prerequisites. These can be used to dictate the behavior of AIOs whilst the structure is kept simple and allows for fast learning through a smaller learning space.

The approach also extends the dynamic scripting algorithm by utilizing a hierarchical structure which allows for emergent planning and resource allocation. We argue that AIOs in RTS games are preferably built in a hierarchical fashion as the tasks for an AIO in an RTS game span from strategic decisions and tactical command, all the way down to individual unit behavior. A hierarchy should thus constitute good mapping from tasks to behavior.

Goal-directed rule hierarchy

Similarly to dynamic scripting, goal-directed hierarchical dynamic scripting (GoHDS) maintains several rulebases, one for each basic player type in a game. Each rule in a rulebase has a purpose to fill and several rules can have the same purpose, e.g. to attack an enemy but in different ways. We extend the amount of domain knowledge by grouping rules with the same purpose, and say that these rules have a common goal. Hence, goals are introduced and put in several goalbases, one for each player type. A rule is seen as a strategy for achieving a goal, which can be seen as domain knowledge used to direct the behavior.

The learning mechanism in GoHDS operates on the probability that a specific rule is selected as strategy for achieving a specific goal. In order to allow for reusability of rules, so that many goals can share individual rules, weights are detached from rules and instead attached to the relationships between goals and rules, see Figure 1. By assigning weights to each goal-rule relationship, adaptation can occur in a separate learning space for each goal. This can allow for higher flexibility and reuse.

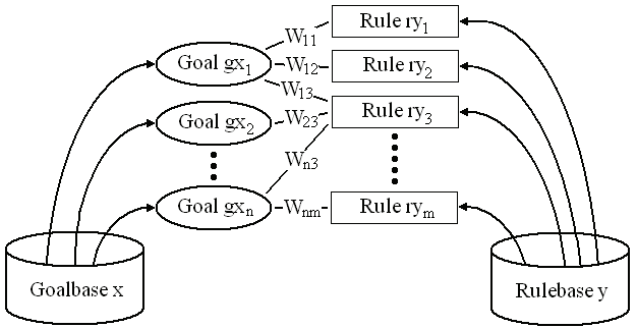

Figure 1: Illustration of the goal-rule layout.

Rules in GoHDS are divided in two distinct states: init and active. The init state has the purpose of asserting that the global tactical/strategic state is suitable for a given rule, e.g. preconditions are checked to see if the rule is at all applicable. If the rule is not applicable, then goals are started with the purpose of fulfilling the global tactical/strategic state that is needed for the rule. For example, if an assault squad of tanks is to be built, then a heavy weapons factory is needed. In the case where a heavy weapons factory does not exist, then it is not necessary to check if there is enough cash to build tanks, and instead, a goal to create a heavy weapons factory can be started. Rules change to the active state when their preconditions are fulfilled. The active state has the purpose of executing the main action of rules if their optional condition(s) is (are) true, e.g. to build an assault squad in the previous example.

An advantage of using dynamic scripting is that rules are designed in a clear and understandable fashion. This might pose a problem if rules are to give the illusion of intelligence at the tactical/strategic level. For example, a single rule for ordering a blitzkrieg assault might neither be simple nor reusable if a single rule handles the complete behavior. Hence, rules are broken down into smaller rules and sub-goals which are connected to form a hierarchy of goals and rules. This is illustrated in Figure 2. By dividing rules into many small rules and goals, the simplicity and understandability can more easily be maintained.

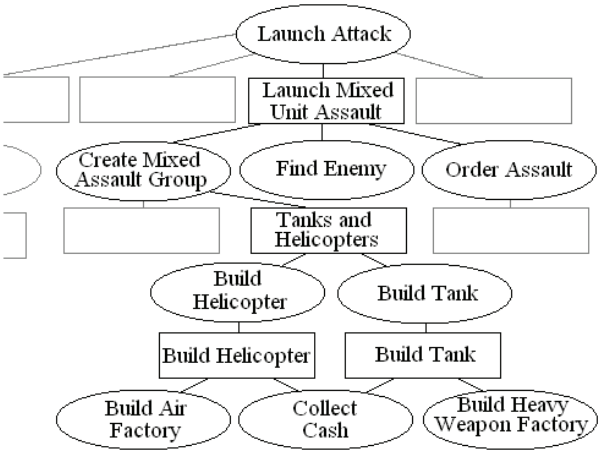

Figure 2: Illustration of a simple goal-rule hierarchy.

GoHDS in an RTS game

Is the GoHDS method enough for creating an AIO which gives the illusion of intelligence in an RTS game? At the tactical/strategic level an AIO faces tasks such as resource allocation, modeling, spatial- and temporal reasoning, planning, and decision making. These tasks can all be important when creating AIOs in RTS games, but which tasks can GoHDS handle and how can it be combined with other systems in order to achieve such a system?

The GoHDS method does not contain a communication system, it is not a spatial reasoning system, nor is it a temporal reasoning system; hence, collaboration and spatial- and temporal reasoning are excluded. Explicit modeling is ruled out as dynamic scripting is not a system for making plausible hypotheses concerning enemy intent. Dynamic scripting is however a machine learning technique and its weights implicitly model the behavior

previously expressed by its enemies. Some degree of resource allocation and planning is also managed in the hierarchy and by the preconditions.

We argue that the GoHDS method might need to be complemented with other systems in order to be applicable in practice. A collection of many sub-systems can in combination be used to form an overall strategy for victory, and GoHDS can be used as one such sub-system. The introduction of goals through GoHDS can be exploited further by using goals as an interface between the different systems. For example, GoHDS might need to be fed with goals to be efficient. It might also need to retrieve information concerning vantage points, paths, and avenues of approach. A simple example of how to combine a set of systems with GoHDS is now presented.

First, a perception system is needed in order to act. This can for example be achieved through a 2D map containing all vital objects that are seen by friendly units. Furthermore, the perception system can be complemented with influence maps for detecting movement patterns of enemies. The perception system can be used by a modeling system which, for instance, keeps a state vector of the world. Each state can then be matched against a target state and for each state that is not fulfilled a goal to fulfill it can be fed to GoHDS. Furthermore, GoHDS can communicate with the perception system on its own in order to retrieve state information. The modeling system and GoHDS could also communicate with some form of resource management system that prioritizes and performs production scheduling. In addition, a pathfinding system could be used by GoHDS, the modeling system, and an object system. The pathfinding system could also use some form of terrain analysis system for input. Figure 3 illustrates a simple example of the described system.

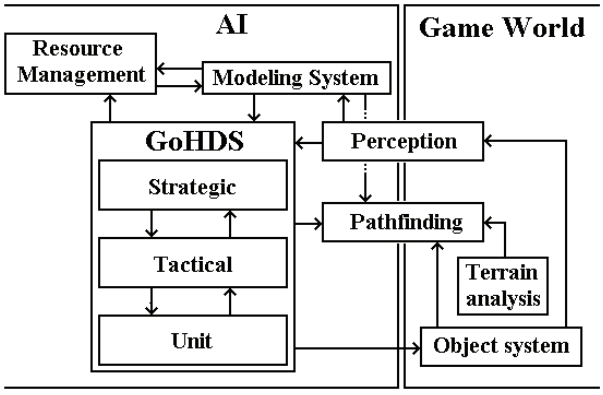

Figure 3: Example of how GoHDS can be combined with other systems.

Learning in RTS games

Exploiting the feedback

At the tactical/strategic level in an RTS game, the number of occasions when feedback is available can be rather few. It is therefore important to exploit the feedback as much as possible, when it actually exists. In other words, there is a need for rapid learning. In many learning paradigms, a distinct learning rate (LR) factor usually controls the speed at which learning proceeds. In dynamic scripting the LR factor consists of a reward and a punishment factor. In order to actually exploit the feedback from the environment, it is important to understand how these factors affect the learning process, both individually, as well as when combined with a distinct LR factor. It is also interesting to investigate if the time to achieve adaptation can be lowered by manipulating these factors.

In order to compare how the punishment and reward factors affect the learning process, this paper has investigated three different settings of these factors: (1) higher rewards, (2) higher punishments, and (3) equal rewards and punishments. It is also interesting to investigate if the adaptation time can be shortened by increasing both factors proportionally at the same time. Hence, this paper has also investigated if a larger LR yields shorter adaptation times. It is however important to remember that having too large a LR factor could introduce predictability, which eliminates one of the benefits of using dynamic scripting – unpredictability (Spronck, et al. 2003).

In methods based on reinforcement learning, the punishment and reward factors are usually proportional to the fitness relative to some pre-defined break-even point i.e. the point where good and bad behaviors join. Temporal aspects are however usually neglected. In case of dynamic scripting, considering temporal aspects of results could however be applicable. For example, if a rule achieves low fitness for a number of consecutive evaluations, then that rule is potentially no good and its weight can be drastically decreased. Similarly, in the case of consecutive good results the weight for a rule can be drastically increased. A potential realization of this could be to track the trend of change over time in fitness results, i.e. to introduce the derivative of the fitness results.

Using the derivative of the results is however not directly applicable as the fitness results do not constitute a continuous function. The point-wise derivative could be used, but with the potential problem of introducing oscillating results. A third approach for exploiting the derivative is to use some form of smoothing function, such as a non-uniform rational b-spline (NURB[I]), or similar function. Fitness results can be inserted into a NURB which can be used to find the derivative. By using a NURB, the influence of historical fitness results can be weighted so that recent results have a higher impact on the derivative. Historical results can however help to minimize the impact of the derivative in case of uncertain direction.

This paper has investigated if adaptation time can be lowered by including the derivative in the weight-update function.

[I] For more information regarding NURB curves and their derivative, see for example Piegl and Tiller (1995).

Performance throttling

Computer games need to be fun for both novice and expert players. This can be a problem for many adaptation algorithms since they usually aim at reaching the best available performance. This problem could however possibly be solved by designing fitness criteria that do not promote the best performance, but which promote high entertainment value. Entertainment value is a complex term, but we argue that it can be increased if the performance of an AIO matches that of the human player. This means that the performance of an AIO might need to be throttled to match the expertise exerted by its human opponent, i.e. to put up a good fight, but to lose.

One approach for limiting the performance of an AIO is to investigate the fitness and weight-update functions. The fitness function determines the score achieved for each rule and the weight-update function translates the fitness scores into weight changes. Establishing design criteria for a fitness function that does not promote the best available behavior can be considered a difficult problem and hence we focus on the weight-update function. The weight-update function used by Spronck, et al. (2003) proportionally maps fitness values into weight changes so that the best available fitness gives the largest weight increase and vice versa. We suggest that a fitness-mapping function can be used in between the fitness and weight-update functions, which maps fitness scores into a fitness space that promotes behaviors relative to a difficulty level.

We have investigated if a fitness-mapping function, based on the sine function, can be used to throttle the performance of an AIO. One revolution of the sine function has been used and its amplitude and frequency has been translated and scaled to fit the fitness space. Further, the function is phase-shifted to center its peak on a fitness value that corresponds to the currently active difficulty level. The following function has been used:

$$f^* = \begin{cases} \dfrac{\sin\left(2\pi\left(f - f_T + 0.5\right) - \dfrac{\pi}{2}\right)}{2} + 0.5 & \left|f - f_T\right| \le 0.5 \\ 0 & \textit{otherwise} \end{cases}$$

where f^* denotes the fitness after mapping, f the original fitness value, and f_T the target fitness.

Simulation experiments

The results in this section are based on Dahlbom (2004).

Test environment

The aim of the test environment is to: (1) compare various settings of the punishment and rewards factors, (2) measure the adaptation time over varying learning rates, (3) measure the adaptation time when including the derivative, and (4) measure the performance when applying varying fitness-mapping targets.

A simulation involves two artificial players launching assault raids against each other. One of the players is a dynamically scripted player and the other is a manually designed player, referred to as opponent player. At the start of each simulation both players are given 1500 cash to symbolize some form of resources. A simulation proceeds by ordering the players to attack each other in an alternating fashion, which starts an encounter. For each encounter both players select a rule, either for attacking or for defending, and for each rule, a group of ten units are created to a cost of ten. Consequently ten are withdrawn from each player's stash of cash for each encounter.

During an encounter, the two opposing groups fire at each other in an alternating fashion. Each group has a predetermined probability of hitting each other. This probability depends on the rules that are applied. One unit is withdrawn from a group when hit by the other and an encounter ends when one of the two groups has run out of units. The remaining units for the victorious player are transferred back to the stash of cash. Finally, a simulation ends when a player has run out of cash.

Goal-rule hierarchy

The structure of GoHDS has been delimited to cover only two levels in the goal-rule hierarchy. By limiting the size of the structure, simulations can be carried out under similar circumstances. Disadvantages of limiting the structure are however that: (1) the usefulness GoHDS is not tested and (2) game specific implications are ignored.

Two goals have been created: attack and defend. Each of these goals has eight rules for accomplishing the goal, see Figure 4. In order for adaptation to be possible in the environment, some rules are stronger and some are weaker, according to a predefined scheme. By using a predefined scheme it is known in advance that reaching convergence is always possible, and hence, the time to reach convergence can always be measured. The environment can also be seen as a variant of the prisoner's dilemma.

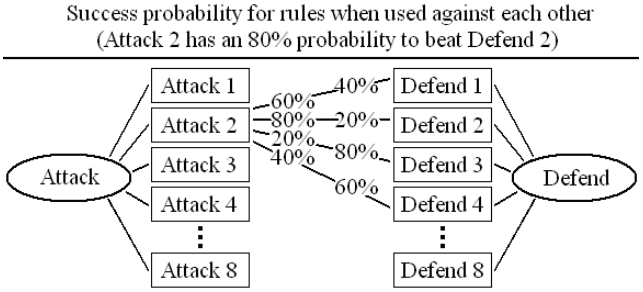

Figure 4: Illustration of the goal-rule setup used.

During a simulation both players always have the defend goal active, which yields that they are always prepared to defend themselves. An attack goal is then given to each player in an alternating fashion to start an encounter.

When an opponent player is assigned a goal it selects rules according to a predefined scheme. A dynamically scripted player selects rules according to the dynamic scripting procedure.

Weight-update functions

Everything in the test environment is based on groups of units fighting each other, and hence, the fitness function is based on the number of surviving units in a group, u_S, and the number of units killed by the group, u_K. As each group initially consists of ten units, the fitness value, f, will be in the range of 0 to 1, and it is calculated as:

$$f = \begin{cases} 0.5 + 0.05u_S & u_S > 0 \\ 0.05u_K & otherwise \end{cases}$$

Informally this means that if a group has lost an encounter, its fitness is below 0.5 and proportional to the number of opposing units destroyed. If a group has won an encounter, then its fitness is above 0.5 and relative to the number of surviving units in the group.

The fitness for a rule is used to calculate a weight change. Two weight-update functions have been used: (1) fitness proportional weight function and (2) the fitness proportional function combined with a fitness derivative function. A similar weight update function to that used by Spronck, et al. (2003), has been used as the proportional function, and a new weight W_P, is calculated as follows:

$$W_P = \begin{cases} \max\left(0, W_O - M_P \cdot \frac{b-f}{b}\right) & f < b \\ \min\left(W_O + M_R \cdot \frac{f-b}{1-b}\right) & otherwise \end{cases}$$

where W_O denotes the old weight, f the fitness, M_P the maximum punishment, M_R the maximum reward, and b the break-even point. A break-even point of 0.5 has been used in all simulations.

When including the derivative, a new weight W_{PD} is calculated as a sum of the proportional function, W_P, and the derivative function, W_D, multiplied by the maximum weight, M_W. The derivative of the fitness results, W_D, has been calculated by inserting historical fitness results into a NURB curve of degree four with evenly distributed knots, and then extracting the derivative from it as follows:

$$W_D = \begin{cases} d(n-1, fv, wv_T) & \text{sgn}(d(n-1,...)) \neq \text{sgn}(d(n,...)) \\ d(n-1, fv, wv_N) & otherwise \end{cases}$$

where n is the degree of the NURB, i.e. 4, $d(t, fv, wv)$ is the derivative at point t on a NURB curve based on a fitness vector fv, and a weight vector wv. Observe that wv_T and wv_N are not to be confused with rule weights; they describe weights for pulling the NURB curve towards its control points, here constituted of the fitness results.

The motivation for using two different weight vectors is: if the point in which the derivative is calculated resides on a local maxima or minima, then the derivative will point in the wrong direction. Hence we use wv_T which pulls the curve towards the most recent fitness result in order to avoid bad behaviors. wv_N and wv_T are defined as:

$$wv_N = \left\{\frac{1}{n}, \frac{2}{n}, ..., \frac{n}{n}\right\}, wv_T = \left\{\frac{1}{n}, ..., \frac{n-1}{n}, 10\right\}$$

Experiments

Four opponent players have been used during simulation: two statically designed and two dynamically designed. The static opponents have a predefined scheme for exactly which rules to deploy during run-time and their purpose is to allow for clear measurements on adaptation and re-adaptation times. To assess the performance against more human-like behavior the dynamic opponents dynamically select rules, but according to a predefined scheme.

Constant: This opponent always deploys the first available rule for each assigned goal.

Changing: This opponent deploys the first rule for each assigned goal during the first 80 encounter pairs (attack and defend), after which it deploys the second rule for each goal. The second rule has an 80% probability of beating the rule that is strong against the first rule.

Consecutive: This opponent deploys a rule until the average fitness for that rule, over the last five encounters, is below 0.5, then the next available rule is selected which in turn has an 80% probability against the rule that is strong against the previously deployed rule. The purpose is to simulate some form of human-like domain knowledge.

Best: An average of ten previous fitness results are calculated for each rule and the rule with the highest average is used at each selection point. This opponent has the purpose of simulating human-like short-term memory.

During and after simulation we have used three different measures to quantify the quality of the dynamically scripted opponents: (1) turning point, (2) re-adaptation point, and (3) average fitness. The turning point is a combination of the average and absolute turning point calculations used by Spronck, et al. (2003), and it is calculated as the first encounter: (1) followed by at least ten consecutive successful encounters and (2) after which the number of consecutive successful encounters is never followed by a longer run of consecutive unsuccessful encounters. The re-adaptation point is calculated as the length of the longest interval of unsuccessful encounters occurring after the turning point has been reached.

Results

Table 1 presents average turning point results achieved during simulation. For each opponent, seven different settings have been used: higher rewards (R+), higher punishments (P+), equal punishments and rewards over three different learning rates (1x, 2x, and 4x), and inclusion of the derivative (df/dt). A 95% confidence is also included which determines the average to a 95% certainty given that the results follow a normal distribution. All simulations have been carried out 20 times in order to calculate the confidence intervals. All values have also been rounded off to the nearest integer.

Table 1: Average turning point results.

Opponent	Constant		Changing		Consec.		Best	
Setting	μ	±	μ	±	μ	±	μ	±
R+	12	3	16	4	116	33	132	32
P+	10	3	12	4	105	33	76	31
1x	11	2	11	3	110	32	96	34
2x	8	2	10	4	98	34	68	30
4x	8	2	10	2	37	20	41	21
df/dt	12	3	10	3	86	36	74	29

It is clear that the consecutive and best opponents are much harder to defeat than the two static opponents, since the average number of encounters before a turning point could be reached is much higher. It can also be observed that the adaptation time is significantly shorter against the dynamic opponents when using a learning rate of four instead of one. A learning rate of four is also significantly better than a rate of two against the consecutive opponent. It is obvious that a higher learning rate has a potential of shortening the adaptation time.

Figure 5 illustrates the adaptation time against the tactic changing opponent when investigating the punishment and reward factors. We see that it can be slightly more efficient to have equal factors, or to have higher punishments than rewards, but not to a significant extent.

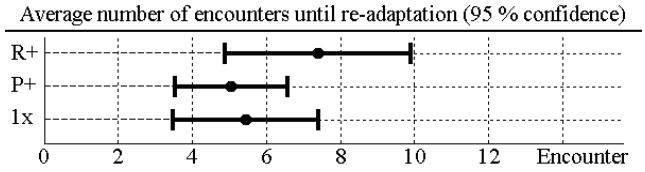

Figure 5: Diagram showing re-adaptation point over three different settings of punishments and rewards.

Table 2 shows the average performance against the four opponents. Again, we observe that the consecutive and best opponents are much harder to defeat. It can also be observed that fitness increases against the tactic changing opponent when increasing the learning rate and when including the derivative.

Table 2: Average performance results.

Opponent	Constant	Changing	Consec.	Best
Setting	μ	M	μ	μ
R+	0.69	0.63	0.49	0.50
P+	0.69	0.64	0.50	0.51
1x	0.70	0.66	0.49	0.50
2x	0.71	0.68	0.50	0.51
4x	0.72	0.68	0.52	0.53
df/dt	0.69	0.68	0.50	0.51

Figure 6 illustrates the re-adaptation time against the tactic changing opponent over three different learning rates and when including the derivative. It can be observed that increasing the learning rate significantly shortens the re-adaptation time. The re-adaptation time is however also significantly shortened when including the derivative.

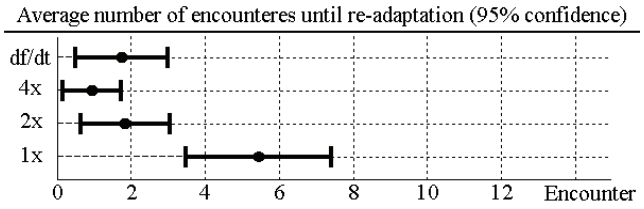

Figure 6: Re-adaptation point over varying learning rate and when including the derivative.

Figure 7 shows results regarding performance throttling when applying a fitness-mapping function. In all four diagrams it can be observed that the fitness centers on 0.5 against the consecutive and best opponents. This complies with earlier results, which points out that the GoHDS algorithm clearly has problems easily defeating the dynamic opponent types. It can however be seen that when applying fitness-mapping targets between 0.2 and 0.5, the average fitness increase similarly against all four opponent types. This means that the performance can be throttled.

In Figure 7, it can also be observed that by applying a fitness-mapping function, the performance can be throttled to negative levels against all four opponents. This means that an AIO can be designed to lose against a human player. It can however also be designed to play even.

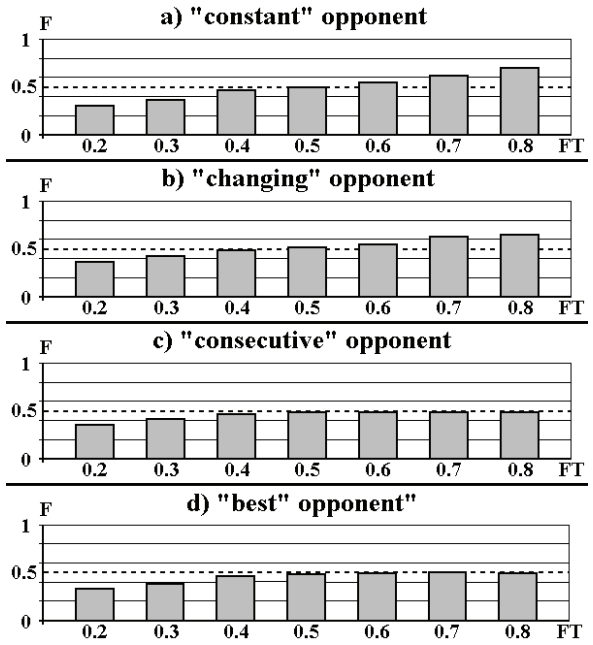

Figure 7: Average fitness (F) over varying fitness-mapping target (FT), against (a) constant opponent, (b) changing opponent, (c) consecutive opponent, and (d) best opponent. The dotted line at 0.5 separates victories from losses and results below 0.5 mean that the AIO using GoHDS lost on average.

Conclusion and discussion

A goal-directed hierarchical approach for extending dynamic scripting has been proposed, GoHDS. In GoHDS, goals are used as domain knowledge for selecting rules, and a rule is seen as a strategy for achieving a goal. A goal can in turn be realized through an arbitrary number of rules and the adaptation process operates on the probability that a specific rule is used as strategy for achieving the purpose of the goal. Rules are divided into sub-goals which put together forms a hierarchical structure. Some degree of planning is introduced by allowing rules to have preconditions, which if false initiate goals with the purpose of fulfilling them.

Simulation results have shown that by increasing the learning rate, or by including the derivative, re-adaptation times can be significantly shortened. Increasing the learning rate too much could however result in predictable behavior. This could lower the entertainment value, and hence, it could possibly be preferred to include the derivative. An approach for effectively throttling the performance of AIOs has also been presented, fitness-mapping, which provides the ability for throttling performance to negative levels, i.e. to lose.

The simulation results might however be dependent on the test environment, and hence, investigations conducted in real games are of great interest in order to verify the results. We however argue that fitness-mapping should be applicable elsewhere too.

Even though the goal-rule hierarchy proposed in this paper has not been thoroughly evaluated, it should still provide a good platform for constructing an RTS game AIO system. The system covers not only the strategic level, but also all levels of the AIO down to every single unit. Hence, the system also serves as an interface between different levels of the AIO. Given that AIOs in an RTS game are preferably built in a hierarchical fashion, the goal-rule hierarchy provides a good structure for achieving a goal directed behavior, which includes adaptation.

Future work

In future work we will investigate the applicability of GoHDS in practice, both in RTS games as well as in other simulated environments and when applied in other domains. We will also investigate the surrounding systems for achieving an illusion of intelligence. The complete picture is considered of high importance.

Acknowledgments

This work was supported by the Information Fusion Research Profile (University of Skövde, Sweden) in partnership with the Swedish Knowledge Foundation under grant 2003/0104.

References

Buro, M., & Furtak, T. 2004. RTS Games and Real-Time AI Research. In proceedings of the Behavior Representation in Modeling and Simulation Conference (BRIMS). Arlington VA 2004.

Dahlbom, A. 2004. An adaptive AI for real-time strategy games. M.Sc. diss., University of Skövde.

Forbus, K.D., Mahoney, J.V., & Dill, K. 2001. How qualitative spatial reasoning can improve strategy game AIs. In proceedings of the AAAI Spring Symposium on AI and Interactive Entertainment, March, 2001.

Laird, J.E. 2000. An Exploration into Computer Games and Computer Generated Forces. In proceedings of The Eight Conference on Computer Generated Forces and Behavior Representation. Orlando, FL.

Lidén, L. 2003. Artificial Stupidity: The Art of Intentional Mistakes. In *Ai game programming wisdom 2* (ed. S. Rabin), 41-48. Charles River Media.

Nareyek, A. 2002. Intelligent Agents for Computer Games. In proceedings of the Second International Conference on Computers and Games (CG 2000).

Piegl, L., and Tiller, W. 1995. *The nurbs book, 2nd edition*. Springer.

Ponsen, M.J.V., & Spronck, P. 2004. Improving Adaptive Game AI with Evolutionary Learning. In proceedings of Computer Games: Artificial Intelligence, Design and Education (CGAIDE 2004), 389-396. University of Wolverhampton.

Spronck, P. 2005. Adaptive Game AI. Ph.D. thesis, Maastricht University Press, Maastricht, The Netherlands.

Spronck, P., Sprinkhuizen-Kuyper, I., & 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), (eds. Quasim Mehdi and Norman Gough), 93-100. EUROSIS, Belgium.

Spronck, P., Sprinkhuizen-Kuyper, I., & Postma, E. 2002. EVOLVING IMPROVED OPPONENT INTELLIGENCE. In proceedings of the 3rd International Conference on Intelligent Games and Simulation (GAME-ON 2002), (eds. Quasim Mehdi, Norman Gough, and Marc Cavazza), 94-98. Europe Bvba.

Woodcock, S., Pottinger, D., and Laird, J.E. 2000. Game AI: The State of the Industry. *Game Developer Magazine* (August), 24-39. CMP Media LLC.