Proceedings of the First
Artificial Intelligence and Interactive
Digital Entertainment Conference

Hierarchical Plan Representations for Encoding Strategic Game AI

Hai Hoang, Stephen Lee-Urban, Héctor Muñoz-Avila

Department of Computer Science & Engineering, Lehigh University, Bethlehem, PA 18015-3084 USA

Abstract

In this paper we explore the use of Hierarchical-Task-Network (HTN) representations to model strategic game AI. We will present two case studies. The first one reports on an experiment using HTNs to model strategies for Unreal Tournament® (UT) bots. We will argue that it is possible to encode strategies that coordinate teams of bots in first-person shooter games using HTNs. The second one compares an alternative to HTNs called Task-Method-Knowledge (TMK) process models. TMK models are of interest to game AI because, as we will show, they are as expressive as HTNs but have more convenient syntax. Therefore, HTN planners can be used to generate correct plans for coordinated team AI behavior modeled with TMK representations.

Introduction

Goal-Oriented Action Planning (GOAP) is a new and promising paradigm for encoding game AI (Orkin, 2003). GOAP is motivated by the need for a decision making architecture that allows characters to decide not only what to do, but how to do it. The advantage over Finite State Machines (FSM) is that characters encoded with GOAP can find alternate solutions to situations encountered in the game, and can handle dependencies that may not have been thought of at development time.

The logic behind GOAP is STRIPS planning (Fikes & Nilsson, 1971), a form of planning that represents actions (i.e., the possible actions that a character can execute), states of the world (i.e., the situation in the game at some point of time), goals (i.e., in-game objectives such as patrolling a location) and plans (i.e., sequence of actions achieving a goal such as undertaken by a character to patrol a location).

Hierarchical Task-Network (HTN) planning is another form of planning that advocates reasoning on the level of high-level tasks rather than on the level of the actions (Erol *et al.*, 1994). HTN planning decomposes high-level tasks into simpler ones, until eventually all tasks have been decomposed into actions. HTN planning has two main advantages over STRIPS planning. First, it is provably more expressive than STRIPS planning. That is, there are problems that can be expressed as an HTN planning problem that cannot be expressed as a STRIPS planning problem. Second, several authors have pointed out that HTNs can encode strategic knowledge naturally.

In this paper we explore the use of HTN representations to model strategic game AI. We will present two case studies. The first one shows that HTNs can be used to model team-based strategies for Unreal Tournament® (UT) bots. We will present experiments with UT bots supporting this claim. The second one discusses an alternative to HTNs called TMK models. TMK modeled processes and are of interest for game AI because TMK models are used by TIELT to model AI behavior. TIELT is a project funded by DARPA to create a testbed for integrating machine learning algorithms with computer game engines. The goal of TIELT is to bridge decision systems and computer games, allowing researchers to more easily test novel algorithms in sophisticated games while at the same time demonstrating the potential practical utility of these algorithms to game developers. Our case study shows that TMK models are equally expressive as HTNs and, therefore, TMK models share the well-defined properties of HTNs.

Related Work

Goal-Oriented Action Planning

GOAP represents actions that a character can execute (Orkin, 2003). Based on a given in-game situation, it determines which actions to execute and the appropriate sequencing of these actions (i.e., a plan). This allows a modular development of AI behavior. Rather than explicitly specifying the interrelations between actions as done when encoding FSMs, these interrelations are determined at run time. A terminology clarification is needed; the states in the FSMs correspond to the actions in planning and vice versa. Figure 1 contrasts FSMs with GOAP with a simple example. The FSM specifies two states in which the AI character can be: patrolling and fighting (taken from (Houlette & Fu, 2003)). When a monster is in sight, it changes to the fight state. When the fight is over and there is no monster (i.e., it has been killed or fled), the character resumes patrolling.

Actions in GOAP use a STRIPS representation (Fikes & Nilsson, 1971). In STRIPS, ***actions*** are instances of generic schemata called operators. An ***operator*** has preconditions and effects. The preconditions indicate the conditions that must be valid for the operator to be applicable. The effects indicate how the current situation changes as a result of applying the operator. Figure 1 also shows the two operators defining the two actions (states) in the FSM and a possible resulting plan after applying these operators to a game situation. Although simplistic, the example illustrates the dynamic nature of the generation of AI behavior. Characters do not have to use all possible actions and their sequencing is not predefined. This reduces the difficulty of having to predict every possible situation when encoding the FSM.

.

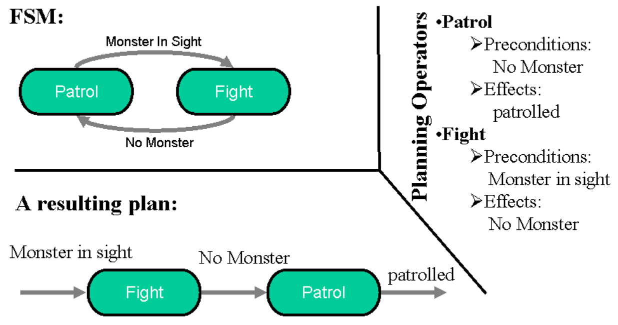

Figure 1. Contrasting FSMs and GOAP

A ***planning problem*** is defined as a collection of goals to achieve, an initial situation or state of the world, and a collection of operators. A well known difficulty of using planning algorithms in real-world problems is in solving planning problems efficiently. A ***solution*** to a planning problem is a sequence of actions, called a ***plan***, that fulfill the goals of the problem relative to the state of the world. GOAP advocates the use of domain-specific heuristics to guide the plan generation process and highly efficient data structures to represent the information needed during plan generation (e.g.., the current situation, the operators, etc.). GOAP has been successfully applied to control the behavior of the AI opponents in the upcoming first-person shooter game (FPS) F.E.A.R..

HTN Planning in Games

Hierarchical planning has been shown to be a promising means to build computer opponents. For example, Bridge Baron® 8 won the 1997 world-championship competition for computer programs using HTN planning techniques to plan its declarer play (Smith *et al*., 1998). Other authors have acknowledged the richness of HTN planning for building game AI, although no specific applications of HTNs in games were provided. In this paper we explore the use of HTN representations to encode strategic team-based behavior of game AI in modern FPS games.

Client-Server Architectures for Unreal Tournament

The UT server provides sensory information about events in the UT world and controls all gameplay and interaction between the bots and players. A client program uses this information to decide commands controlling the behavior of a bot and passes them to the server. TIELT could have been used as a client, but TIELT currently only supports the controlling of one bot at a time, which means we would have to run multiple instances of TIELT to control multiple bots. The client program that we used was Javabot, developed at Carnegie Mellon University (http://utbot.sourceforge.net/), because it supports running multiple bots at the same time by default. Javabot uses a FSM to implement the behavior of the bot based on the sensory information provided by the UT server. Javabot uses the Gamebot API, developed at the University of Southern California, to communicate with the UT server. Event handlers are used to detect relevant events that may require a change in the course of action. For example while exploring, if the bot detects an enemy in the surrounding area, it may decide to start hunting. Our first case study encodes team-based strategies for UT bots using HTNs.

Synthetic Adversaries and Realism

The virtual training of soldiers for Military Operations on Urbanized Terrain (MOUT) is a system developed and used for actual training of military personnel (Laird & Duchim, 2000). The system is based on Quake bots. The emphasis of MOUT is realism; bots can explicitly communicate with each other while accomplishing their goals. In our first case study we will describe an application of HTNs to coordinate teams of bots. But our focus is not realism; control of the various bots is centralized in the HTN, and therefore coordination and communication is implicit (rather than explicit) . Our case study focus is on challenging game AI.

Hierarchical FSMs

Hierarchical FSMs are an extension of FSMs in which states can expand into their own sub-FSMs (Houlette & Fu, 2003). Events can either change the state at the same level in the FSM hierarchy or make a transition at a higher level in the FSM hierarchy. When entering a state, the program chooses a state for its child in the hierarchy. Therefore, hierarchical FSMs allow the definition of stratified AI behavior and can encode strategic AI behavior. Our motivation for using HTN representations instead of Hierarchical FSMs is analogous to the motivation for using GOAP instead of FSMs: alternative strategies can be found according to situations encountered in the game, and can handle dependencies between parts of a strategy that may not have been thought of at development time.

Encoding Strategic Game AI in HTNs

HTNs decompose high-level tasks into simpler tasks. There are two kinds of tasks: compound and primitive. ***Compound tasks*** can be further decomposed into subtasks whereas ***primitive tasks*** cannot. The primitive tasks indicate concrete actions. Each level in an HTN brings more details on how to achieve the high-level tasks. The sequencing of the leaves in a fully expanded HTN indicate the plan achieving the high-level tasks. In the context of game AI the decompositions can be used to encode game strategies and the leaves to actual in-game actions such as patrol, attack, etc.

The main knowledge artifacts in HTN planning are called ***methods***. A method encodes how to achieve a compound task. Methods consists of 3 elements: (1) The task being achieved, called the ***head*** of the method, (2) the set of preconditions indicating the conditions that must be fulfilled for the method to be applicable, and (3) the subtasks needed to achieve the head. The second knowledge artifacts are the operators. Operators in HTN planning have the same purpose as in STRIPS planning, namely, they represent action schemes. However, operators in HTN planning consist of the primitive tasks to achieve, and the effects, indicating how the world changes when the operator is applied. They have no preconditions because applicability conditions are determined in the methods.

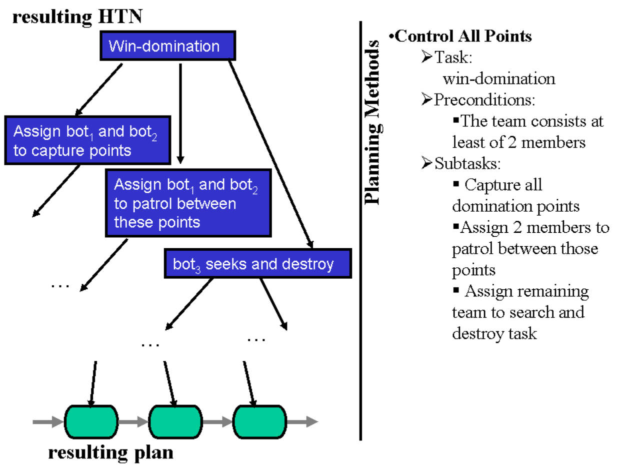
resulting HTN
Win-domination
Assign bot$_1$ and bot$_2$ to capture points
Assign bot$_1$ and bot$_2$ to patrol between these points
bot$_3$ seeks and destroy
resulting plan
Planning Methods
•Control All Points
➢Task:
win-domination
➢Preconditions:
▪The team consists at least of 2 members
➢Subtasks:
▪ Capture all domination points
▪Assign 2 members to patrol between those points
▪ Assign remaining team to search and destroy task

Figure 2. A Method and an HTN for UT bots

The crucial difference between STRIPS and HTN planning is that in the former, the reasoning process takes place at the level of the actions whereas in the latter the reasoning process takes place at the level of the tasks. This is precisely our motivation to extend the GOAP framework by introducing HTNs; in the context of game AI, this difference means that HTN planning reasons on what strategy to select and how to accomplish this strategy rather than directly on what actions to execute. Even though it is generally possible to encode strategies in STRIPS representations, HTNs capture strategies naturally because of the explicit representation of stratified interrelations between tasks. Furthermore, representing HTNs in STRIPS operators is very cumbersome in general (Lotem & Nau, 2000) and sometimes even impossible (Erol *et al.*, 1994). In the next section we will present concrete examples of methods encoding strategies.

Case Study: Strategic Planning for UT Bots

Our first case study reports on an experiment we developed to build strategic game AI to control a team of Unreal Tournament® (UT) bots in a domination game. In domination games, there are fixed locations in the game world that are called domination locations. When a team member steps into one of these locations, the status of the location changes to be under the control of his/her team. The team gets a point for every five seconds that each domination location remains under the control of that team. The game is won by the first team that gets a pre-specified amount of points.

The purpose of our first case study is threefold. First, we wanted to demonstrate the capabilities of HTNs to encode strategic AI behavior. For this purpose, we encoded domination game strategies in HTN methods. Second, we wanted to support our claim that HTNs can extend the GOAP framework. Currently, GOAP has been demonstrated controlling single F.E.A.R. bots. In our experiment we control a team of UT bots. Third, we wanted to contrast our work with previous work with UT bots. As discussed in the Related Work section, the behavior of the standard UT bots is defined by FSMs controlling single bots.

Figure 2 shows an example of the method *Control All Points* encoding a strategy for the task *win-domination*, to win a domination game. This strategy requires that the team consists of at least 2 members. The strategy calls for two members to capture all domination points, and patrol between them. The remaining team members are assigned to search and destroy tasks (provided that there are more than 2 team members). Achieving these tasks require sub-strategies defined by other methods. For example, when needed (e.g., for search and destroy tasks), we group bots together that move from waypoint to waypoint. A waypoint is a predefined location and is used by the bots for navigation purposes. This strategy increases the chances of killing enemy bots due to numeric superiority.

Figure 2 also sketches a resulting HTN when *Control All Points* is used in a game. In this situation there are 3 domination points and 3 team members. The first 2 team members are assigned to the domination points and patrol between them, and the third is assigned to search and destroy tasks. The resulting plan is the sequence of all leaves (i.e., primitive tasks) in the HTN.

For plan generation, we used the HTN planner SHOP. However, to be able to use a planning system like SHOP to generate HTNs controlling teams of UT bots in actual domination games we needed to address two technical challenges: (1) Information about the world is maintained by the UT server, and (2) the world is dynamic; e.g., when a bot is accomplishing a task, it might get attacked

The first challenge affects how the method's preconditions are evaluated and how actions are executed. Planning systems like SHOP assume that the situation in the world is maintained in an internal data structure and actions are executed by modifying this structure directly. The second challenge affects how actions are executed. The assumption in SHOP is that the state of the world only changes by executing actions. This does not hold in games like UT, where other factors (like opposing team) also change the state of the world.

To address these problems we updated the internal structure of SHOP as the UT server messages were received indicating changes in the game world. The most important extension, however, refers to the actions. We use standard event-driven UT bots encoded in Java to execute the actions, but we extend them so they can also perform the primitive tasks assigned by the HTN, such as going to a certain waypoint. As a result, a grand strategy is laid out by the HTNs and event-driven programming allows the bots to react in this highly dynamic environment while contributing to the grand task. The event-driven program encoded in the Javabot FSMs allows the bot to react if, for example, an enemy bot is shooting at it.

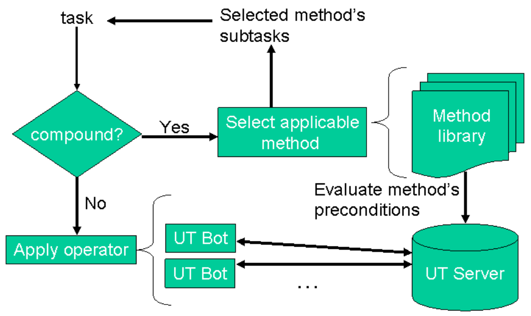

Figure 3. Dataflow of the HTN Planning

Figure 3 shows the dataflow of the system. Given a task to achieve (e.g., *win-domination*), there are two possible cases:

- If the task is compound, applicable methods are found by processing the updates from the UT server. That is, the method's preconditions are evaluated based on the information provided by the UT server. When an applicable method is found, it is decomposed into its subtasks and the process is repeated.
- If the task is primitive, a UT bot performs this task. Which UT bot gets activated is decided as part of the HTN decomposition process. For example, the subtask *assign* bot_2 *to* dom_2 in Figure 2 will eventually be decomposed into a concrete action, whereby bot_2 will move to dom_2.

We encoded two different strategies in the HTNs. The first strategy is called *Control Half Plus One Points*. This strategy selects half plus one of the domination points and sets bots to capture these points. After capturing these points the bots will patrol between these places to defend them. The second one is the *Control All Points* strategy.

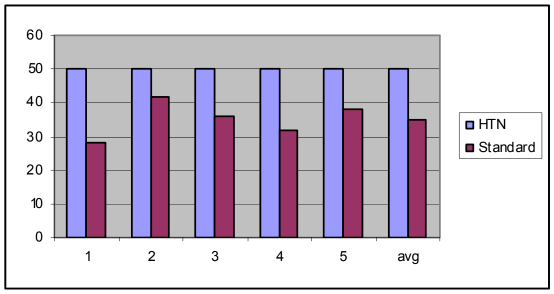

Figure 4. *Control Half Plus One* vs. standard

For the experiments, we had two opponent teams. The first team consisted of the standard UT bots that came with Javabot. We refer to the first team as *standard* team. For the second team we did several improvements to the code of the standard UT bots. In particular, we improved navigation issues and domination tactics. We refer to this team as the *improved* team. The team that uses the HTNs uses the same improved code. The only difference is that the domination strategies are dictated by the HTNs. We refer to this team as the *HTN* team.

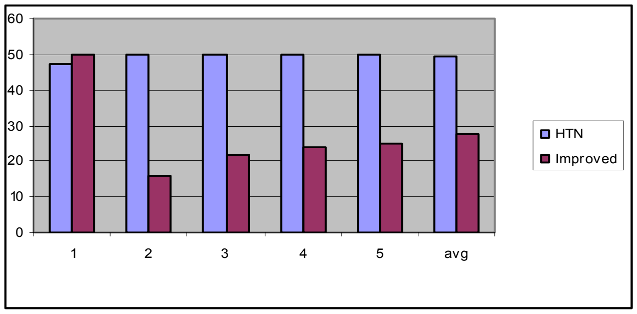

Figure 5. *Control Half Plus One vs.* Improved

We ran the experiments on the domination map Dom-Stalwart, that came with Gamebot. We counted results only when a match terminated where no bot from either team got disconnected from the server. Since the positions of the bots are determined by the UT server randomly and these positions improve the chances that either team will win, we ran the experiments 5 times and averaged their results. These results are shown in Figures 4 and 5 for the *Control Half Plus One Points* strategy versus the standard and the improved teams respectively. Figures 6 and 7 show the results for the *Control All Points* strategy versus the standard and the improved teams respectively. The number of points to win a match was set to 50.

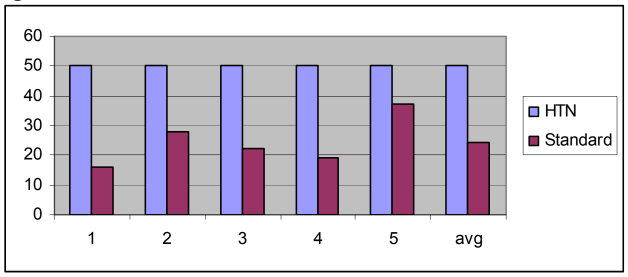

Figure 6. *Control All Points* vs. Standard

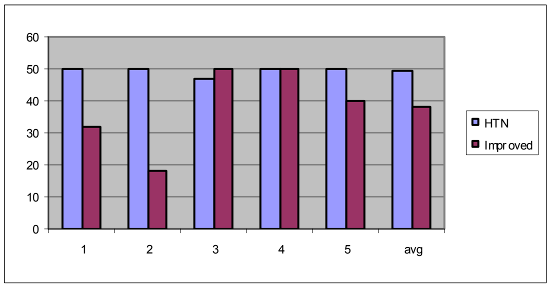

Figure 7. *Control All Points* vs. Improved

These results show a clear dominance by the HTN team over the two other teams. This is not surprising since the HTNs allow the bots to coordinate their tasks cohesively.

Our main result is that it is possible to encode strategies that coordinate teams of bots in FPS games using HTNs and run them effectively, using standard FSMs to encode individual bots behavior.

Case Study: TMK Process Representations

Having shown the capabilities of HTNs for coordinating strategic team-based behavior, we will now discuss TMK models, an alternative hierarchical representation to HTNs. TMK models are used by the TIELT testbed software to encode AI game behavior. The goal of TIELT is to bridge decision systems and computer games, helping researchers to test novel algorithms in sophisticated games while at the same time demonstrating the potential practical utility of these algorithms to game developers.

Description

The Task-Method-Knowledge (TMK) formalism was developed for modeling processes (Murdock, 2000). Like HTNs, TMK models describe a system in terms of the manipulation of domain knowledge via a task-method hierarchical architecture, and allow reasoning at a strategic level rather than on the level of actions. Tasks are the basis of the model, and transform the input knowledge-state to an output knowledge-state. Methods decompose a task into subtasks in a recursive fashion until leaf tasks are reached. Leaf tasks are defined as procedures accomplished via the internal manipulation of knowledge.

TIELT

TIELT (http://nrlsat.ittid.com), the Testbed for Integrating and Evaluating Learning Techniques, is a free software tool created to ease the evaluation of decision systems in simulators (Aha & Molineaux, 2004). The simulators can be of several different types of game genres such as real time strategy, first-person shooter, team sports games, or even a simulator not related to gaming. One key way that TIELT makes the evaluation of decision systems easier is by reducing the number of integrations between simulators and decision systems from (m * n) to (m + n), where m is the number of investigated simulators and n is the number of decision systems being evaluated.

TIELT decomposes the problem of decision system evaluation on performance tasks in simulators into various components, tied together via a GUI. One of these components is the Agent Description. This component provides the ability to richly define complex actions by describing them using a slightly modified TMK formalism. The language used to represent TIELT's TMK model is based on XML and is called the TIELT Script Extended Markup Language (TSXML). TSXML provides a clear and uniform syntax that straightforwardly captures the TMK created via TIELT's interface. Because this XML based syntax might be unclear for some readers, we use a pseudo-code style format in our examples.

Comparison with HTNs

TMK models at first appear to be more expressive than HTNs since the TMK language explicitly provides constructs for looping, conditional execution, assignment, functions with return values, and other features not found in HTNs. However, we found that HTNs implicitly provide support for the same features, albeit in a less obvious fashion, and a translation from TMK models to HTNs is always possible. For the sake of clarity, we will use pseudo-code for describing the HTNs instead of the LISP-based syntax used in HTN planners like SHOP.

Table 1 shows a synopsis of 3 TMK constructs and how they can be mapped into HTNs. We omit a complete and formal proof due to the lack of space.

Table 1: Mapping of TMK models and HTNs

TMK Models	**HTNs**
Return values of functions	Use unbound variable as parameter in caller's invocation; set same variable in callee's preconditions
If-then-else	Use HTN method syntax
Iterations (while)	Recursion
Iterations (for)	Change to while, recursion
Assignment (set)	Split into new method and pass in evaluated value
Tasks have preconditions	Add preconditions to methods

Returning values from functions can be simulated by adding unbound variables in the methods. This is illustrated in Figure 8. The TMK enemyOwnsDOM method returns a Boolean value indicating if our team owns less domination points than the number of available domination points. In the HTN method we add the returned value explicitly as a parameter of the task in the head of the method. The subtask dummySubtask() is always fulfilled. This subtask is needed for full adherence with HTN formalisms that require all subtasks to be atoms.

Figure 8: Returning values of functions

```
TMK Method Task: boolean enemyOwnsDOM( )
   If
      totalDominationPoints(td)
      totalDominationPointsOwnedbyTeam(tdTeam)
   Then
      return ( td > tdTeam )
HTN Method  Task: enemyOwnsDOM (ret)
   Preconditions:
         totalDominationPoints(td)
         totalDominationPointsOwnedbyTeam(tdTeam)
         ret = (td > tdTeam)
   Subtasks: dummySubtask()
```

The translation for the TMK "if" statement is straightforward because HTNs represented in systems like SHOP allow sequences of preconditions-subtasks pairs with a similar meaning as if-then-else statements. SHOP evaluates these sequences by checking the preconditions of the first pair; if these are true, then the SHOP continues with the subtasks of the first pair (Figure 9). If the preconditions of the first pair are not fulfilled, then SHOP checks the preconditions of the second pair. If these are satisfied, SHOP continues with the subtasks of the second pair, and so forth.

Figure 9: Representing If-Then-Else statements

```
TMK Method  Task: void doSmartTactic( )
  If ( numEnemies == 0 ) Then
      celebrate();
 else If ( numEnemies == 1 )
      Then  hunt();
 else  runaway();
```

```
HTN Method Task: doSmartTactic( )
   Preconditions: numEnemies == 0
    Subtasks: celebrate()
   Preconditions: numEnemies == 1
    Subtasks: hunt ()
   Preconditions:  true
   Subtasks: runaway()
```

It is well known from basic programming principles that "while" loops can be represented using recursion. We once again take advantage of the sequences of preconditions-subtasks pairs for this purpose (Figure 10). The precondition of the first pair is the condition, ownAllDOMPoints(), used to continue iterating in the loop. The first subtask, patrol(), is executed in the loop body. The second subtask, patrolling(), is a recursive call. When the condition ownAllDOMPoints() no longer holds, then the precondition of the second preconditions-subtasks pair is evaluated and found to be true, which always holds. As before, the subtask dummySubtask() is always fulfilled.

Figure 10: Representing while statements

```
TMK Method  Task: patrolling( )
While  (ownAllDOMPoints())
   patrol( )
```

```
HTN Method  Task: patrolling( )
   Preconditions: ownAllDOMPoints( )
   Subtasks:
        patrol()
        patrolling()
   Preconditions: true
   Subtasks: dummySubtask()
```

Other, more compact translations are possible in HTN planners that support lists of objects in the parameters of the tasks and deferred evaluation, but the mappings described here are the most direct and suffice to illustrate the equivalency between TMK models and HTNs.

TMK models represent an attractive alternative to HTNs, and are in fact equivalent. This equivalence means that TMK models have the same well-defined properties as HTNs. In particular, HTN planners can be used to generate correct coordinated team plans for AI behavior modeled with TMK representations. TIELT provides a convenient environment to experiment with hierarchical plan representations for encoding AI gaming strategies.

Final Remarks

In this paper we explored the use of HTN representations to model strategic game AI. We discussed two case studies. The first one shows that HTNs can be used to model effective team strategies for Unreal Tournament® (UT) bots. HTNs were used to encode strategies that coordinate teams of bots in FPS games and run them effectively using standard FSMs to encode individual bots behavior. As a result, a grand strategy is laid out by the HTNs and event-driven programming allows the bots to react in this highly dynamic environment while contributing to the grand task. The second case study discussed the TMK model representations used by TIELT to model game AI behavior. Our case study shows that TMK models are equally expressive as HTNs and, consequently, TMK models share the well-defined properties of HTNs. Therefore, HTN planners can be used to generate correct plans from AI behavior modeled with TMK representations.

Acknowledgements

These research was supported in part by the Naval Research Laboratory and DARPA.

References

Aha, D.W., & Molineaux, M. Integrating learning in interactive gaming simulators. *Challenges of Game AI: AAAI'04 Workshop Proceedings* (Technical Report WS-04-04). San Jose, CA: AAAI Press, 2004

Erol, K., Nau, D., & Hendler, J. HTN planning: Complexity and expressivity. *AAAI-94 Proceedings.* AAAI Press, 1994.

Fikes, R., & Nilsson, N., Strips: a new approach to the application of theorem proving. *AI*, 1971.

Houlette, R., & Fu, D. The Ultimate Guide to FSMs in Games. In: *AI Game Programming Wisdom 2*. Charles River Media, 2003.

Laird, J. E., Duchim J.C. Creating Human-like Synthetic Characters with Multiple Skill Levels: A Case Study using the Soar Quakebot.. *AAAI 2000 Fall Symposium Series: Simulating Human Agents*, AAAI Press. November 2000.

Lotem, A., & Nau, D. S.. New advances in GraphHTN: Identifying independent subproblems in large HTN domains. *AIPS-2000 Proceedings*, AAAI Press, 2000.

Murdock, J.W. *Semi-Formal Functional Software Modeling with TMK*. Technical Report GIT-CC-00-05, Georgia Institute of Technology, 2000.

Orkin, J. Applying Goal-Oriented Action Planning to Games. In: *AI Game Programming Wisdom 2*. Charles River Media, 2003.

Smith, S. J. J., Nau, D. S., & Throop, T. . Success in spades: Using AI planning techniques to win the world championship of computer bridge. *IAAI Proceedings*, 1998