Proceedings of the Seventeenth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment
(AIIDE 2021)

Advanced Real-Time Hierarchical Task Network: Long-Term Behavior in Real-Time Games

Kousuke Namiki, Tomohiro Mori, Youichiro Miyake, Shinpei Sakata, Gustavo Martins
SQUARE ENIX CO., LTD.
namikous@square-enix.com, moritomo@square-enix.com, miyakey@square-enix.com, s-sakata@square-enix.com, martgust@square-enix.com

Abstract

HTN (Hierarchical Task Network) is a widespread technique in the game industry to achieve long-term behavior and smart intelligence in NPCs. However, in the more real-time games of recent years, a situation has emerged where the system is not functioning effectively. We understand the re-planning process is the fundamental problem of HTN in real-time games and propose a new method to solve this problem.

Introduction

In recent years, HTN (Hierarchical Task Network) has been known as a method to achieve long-term behavior with AI Characters in game. (Sterren 2013) (Humphreys 2013) (Straatman et. al 2013) The general HTN system allows for long-term thinking and consistent behavior, which in theory improves the player's gaming experience. However, HTN systems have several vulnerabilities: it can't respond to changes in the environment; it can't respond to events with stochastic behavior; and it can't anticipate the behavior of opponents. (Soemers and Winands 2016) In a real-time game, HTN has failed to improve the quality of the game experience beyond what game developers expected because of these vulnerabilities. In order to overcome these problems, we propose a method called ART-HTN (Advanced Real-Time Hierarchical Task Network) to improve long-term behavior for real-time games.

Background

HTN allows NPCs to make smart long-term decisions. For example, in a sneaking game, the NPC can plan the safest route to take when entering a stage with many enemies or make smart long-term plans on how to kill enemies and which items to pick up. Such long-term NPC behavior can improve the player's experience and make the NPCs seem intelligent if the game progresses as planned. However, in many real-time games, the environment is constantly changing and, over time, the situation can become completely different from what was originally planned, and HTN responds by remaking the plan in a process called re-planning. However, the more frequently re-planning is performed, the more the NPCs lose their ability of long-term thinking, and the closer they get to the kind of reactive behavior described in the FSM or BehaviorTree. In order to address this issue, ART-HTN will make several improvements.

Functions of ART-HTN

The ART-HTN has three main functions: Simulation Planner, Multi Scenario Plan and Plan Executor.

Simulation Planner

In the planning process, we use the simulation-based plan generation method instead of the traditional symbolic planner. Since it is difficult to run a fast simulation with all the variables of a 3D game, we create a simple game model that extracts the main parameters that affect the game's outcome and use this model for simulation-based planning. (Sailer, Buro and Lanctot 2007) Thus, the simulation planner can consider multiple candidate tasks in a specific situation, and these multiple tasks become branches of actions to form a multi-scenario plan.

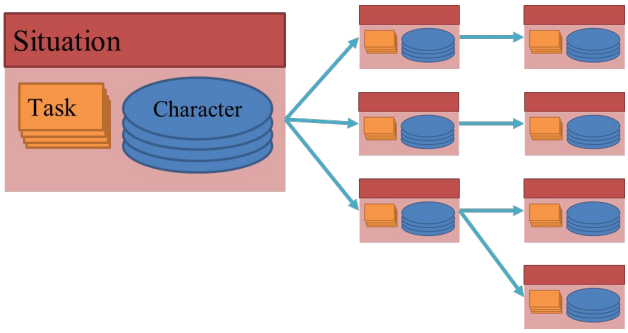

Figure. 1: ST-Network

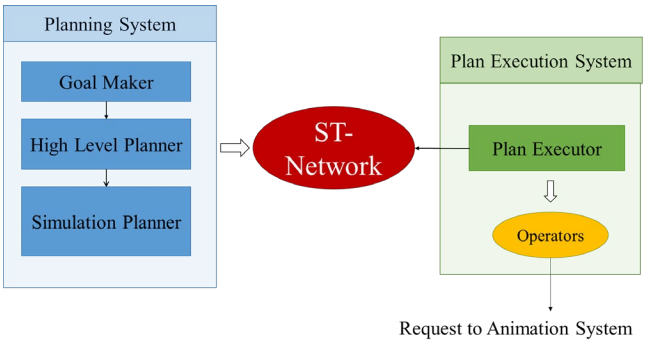

Figure. 2: Process of ART-HTN Systems

Multi Scenario Plan

In existing planning methods, plans are output as a single chain of tasks connected by Precondition and Effect. ART-HTN replaces this single-chain planning structure with a planning structure that includes multiple branches. It functions similarly to a game tree in a turn-based game, outputting paired data of situations and actions that show the outcome if the AI character takes specific action in a particular situation. We call this data Situation Task Network or "ST-Network" for short (Fig. 1). By exploring the network, the AI can calculate what actions would be effective in a given situation.

Plan Executor

The Plan Executor calculates the appropriate situation in the plan and enumerates nodes close to the current situation as transition candidates. Among them, it chooses the most effective action and decides on it. When it comes to the stage of actually executing actions, the game situation becomes even more granular. The Plan Executor is also responsible for resolving the details that are not included in the plan. It adds actions to the plan during the execution phase, adjusts the plan's details to ground it in the actual game situation, and keeps the plan running.

Usecase of ART-HTN

In this chapter, we introduce a concrete example of how the main functions of ART-HTN work together, using a robot game as the subject. The ART-HTN consists of two modules, the Planning System and the Plan Execution System. The Planning System generates the planning data, ST-Network, and passes it to the Plan Execution System. The Plan Execution System modifies the received ST-Network partially if necessary, makes a decision on the optimal move, and decomposes the task registered in the ST- Network (which is data that indicates what action a particular character will take in a specific situation) into Operators that can be executed in the game environment. The task is the simple behavioral data for running the character in the simulator, while the Operator is the behavioral data for running the character in the actual game environment. The task contains only the essential processing related to victory and defeat, while the Operator contains many controls related to the expression of the game, such as the direction of the character's head and posture.

The Planning System consists of several subsystems, including Goal Maker, High-Level Planner, and Simulation Planner. At the beginning of the planning process, the Goal Maker determines the plan's direction. The Planning System specifies the main goal and sub-goals to be achieved as much as possible and their importance. This goal acts as a success condition for the planning and as an evaluation function for the simulation.

The High-Level Planner is responsible for rough march plans for units and logistics plans such as fuel and ammunition supply. At points where the enemy is likely to be hiding, the simulation planner can be used to estimate the damage that would be sustained if the enemy were to engage at a certain point.

The Simulation Planner is primarily responsible for battle planning with the enemy. When it enters into an engagement with the enemy, or when the High-Level Planner wants to predict an engagement scenario, the Simulation Planner runs a battle simulation and generates a battle plan. The plan data is output as ST-Network and passed to the Plan Executor (Fig. 2).

After receiving the plan data, the Plan Executor's role is actually to move the game characters. The Plan Executor decomposes the tasks contained in the Situation and generates Operators. The contents of the Operators are written with functions supported by the game engine and can be run with standard AI systems such as BehaviorTree and visual scripts. The Situation node of the ST-Network has multiple branches, and the Plan Executor can respond to changes in

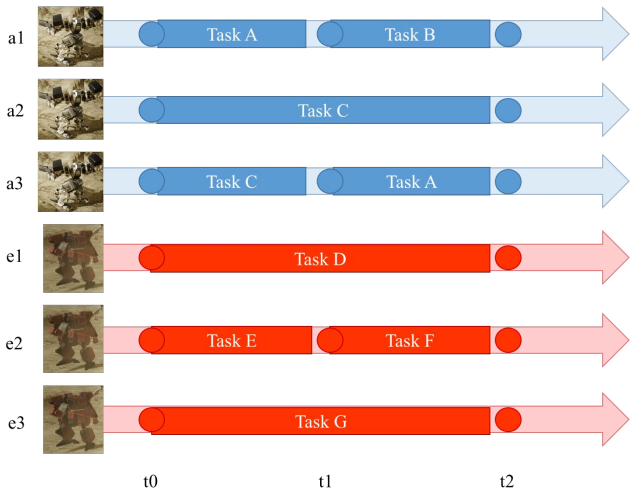

Figure. 3: An example of Task Assigning

the Situation without replanning, as long as the actions taken by the opposing players are included in the ST-Network branches. Due to the robustness of the Plan Executor, the Planning System can be executed somewhat independently of the actual game progress. This separation allows for a longer-term search for superior controls, such as spending a few seconds in a separate thread to search for advanced moves.

Battle Planning

Battle Planning performed by Simulation Planner does not use symbolic planning but rather a state-space exploration using a simulator. The simulation is executed at a particular time step, and the actions of agents participating in the battle are represented by data called to task. It is a kind of combinatorial search problem: what task ***w*** should be assigned to friendly agent ***a*** at a specific time ***t***, and what task ***w*** could be executed by enemy ***e*** at that time. We show an example of Task Assigning (Fig. 3). If the problem is solved by brute force, the number of combinations would be too large to be executed on a consumer game console. For this reason, we used three approaches to reduce the number of states to be searched: meaningful action selection, using opponent models with Composite Task, and symbol extraction.

Meaningful Action Selection

Given the nature of the game, actions such as moving a few meters to the left or right are not very effective in the Action or RTS games we are targeting. When developing enemies with scripts or BehaviorTree, game developers create character AI behaviors in large granular actions such as "approaching the opponent," "moving away from the opponent," and "searching for obstacles and hiding." In the state-space search performed by our Simulation Planner, the granularity of the task to be searched is aligned to a semantically adequate size to prevent the search for meaningless states. In random action search, the number of states increases with meaningless actions such as stepping left or right, but since the tasks we search actions with meaningful granularity such as "approaching the opponent" or "moving away from the opponent," we can prevent unnecessary increases in the number of states.

Using Opponent Models with Composite Task

The Simulation Planner can handle two types of tasks: Primitive Task, simple behavioral task, and Composite Task, hierarchical task that can have other tasks as child task. If the Composite Task performs probabilistic state transitions, the Simulation Planner will copy the Composite Task for each state transition. It is possible to assign a new Task to an agent running a Composite Task, but to avoid a pointless increase in the number of states, the Composite Task can reject the assignment. This authority of Composite Task prevents the problem of a "hide behind a cover and attack" task being replaced to a "get closer to the enemy" task while the character is moving and being canceled by another task before the Composite Task can have any beneficial effect.

Symbol Extraction

In symbolic planning systems used in the game industry, the state of the game is often represented by multiple tuples, called WorldState, and although there is no restriction on the type of tuple values in principle, bool-valued flags are often used for convenience. Our Simulation Planner uses state-space search and works without symbolic planning control in principle. However, for advanced tactical actions such as "surround the enemy and then attack them all at once" or "separate the enemy forces by diversion and then destroy the main enemy force," symbol-based plan generation has excellent advantages. In order to introduce symbol control, we have prepared a helper class called Symbol Analyzer. The Symbol Analyzer determines whether or not a particular symbol is established in a Situation. For example, in the case of the encirclement analyzer, it judges whether or not the enemy is encircled based on the relative positions of friend and enemy. The engagement state analyzer determines whether a battle has started or not. The variables extracted by the Symbol Analyzer are saved as part of the Situation data and can be used when assigning tasks to each agent or determining the Composite Task conditions.

Simulation Planner uses these functions to perform a state-space search to find a state that satisfies the goal condition and a path to get there. Once a sufficient amount of paths have been obtained, the evaluation value is calculated from the terminal node of the ST-Network, and the number of enemies destroyed and the number of friends destroyed is propagated backward from the terminal node to the starting node and included in the evaluation value of the intermediate node.

Game Development with Assistant AI

ART-HTN does not use Machine Learning directly, but it is designed to work with Machine Learning to improve development efficiency. In general, the bottleneck of machine learning using Neural Networks in consumer game development is the slow update time of the game. Since a single frame update of a game includes various processes such as physics calculation, graphics, and animation, even if drawing processes are disabled to increase the frame rate, it is difficult to disable model asset loading and animation updates. Even when machine learning using the game application itself is accelerated by disabling unnecessary functions such as graphics, the acceleration of FramePerSecond is limited to a few dozen times. Learning efficiency can be improved by installing more expensive hardware than a typical development PC, but the development cost will be higher in this case.

In the ART-HTN system, the Planning System is usually linked to the game engine as a static library, but it can be run as a C++ program independent of the game engine if necessary, so the Planning System by itself does not necessarily require the game engine. In addition, since the Simulation Planner does not have a drawing system and the Physics Simulation performed in the planner is extremely simple, the simulation can be performed much faster than running a game engine. Using Neural Network as the evaluation function of the Situation, the results of hundreds of hours of simulations can be used as the evaluation value. It is currently challenging to provide Neural Network-based decision-making systems as a game feature because a general-purpose Neural Network inference engine works on all gaming platforms such as PS5, XBOX Series X, Switch is not standard yet. However, it is becoming technically and cost feasible to utilize Neural Networks in the development process. The system design allows for Machine Learning with Neural Networks in the development process for level design, enemy AI adjustment, and balancing of weapon and armor parameters.

Figure. 4: a screen shot of the original ART-HTN demo

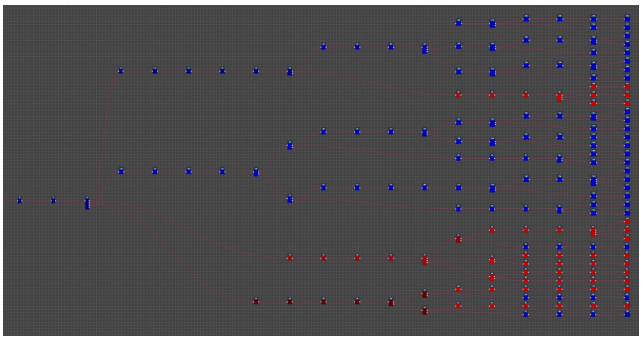

Figure. 5: a generated ST-Network on ART-HTN demo

Demonstration

To verify ART-HTN, we made a 3D action game with robots. The robot, which has an ART-HTN system, fights with the other enemies (Fig. 4). In this demo, ART-HTN generates offensive and defensive plans and chooses defensive plan to survive. By firing smoke grenades to block the enemy's view, the robot attempts to reduce the hit rate of the enemy's attacks. ART-HTN generates ST-Network(Fig. 5) and predicts the wining-rate(blue-color) and defeat-rate(red-color) of each Situation. The Executor chooses a better task.

Conclusion and Future Work

It has been more than 15 years since F.E.A.R. (Orkin, J. 2006) introduced the planning method to games in 2005. Today, the main memory of game consoles has increased more than tenfold, and CPU resources have increased as well. As a result, many of the memory constraints that existed then are no longer an issue now. Our system is still only able to compute one-on-one situations, but in the future, we would like to achieve higher-order tactical in situations where multiple agents are fighting and improve player's game experience.

Biographies of Authors

Kousuke Namiki
Senior AI Engineer: Square-Enix Advanced Technology Division. He has been involved in the game industry since 2008 and joined SquareEnix in 2012. He worked in the development of titles such as
- Final Fantasy VII Remake
- Kingdom Hearts III
- Final Fantasy XV
- Bloodborne
- Steel Battalion: Heavy Armor
- Monster Hunter Diary: Poka Poka Airou Village

as Game AI engineer and QA automation AI engineer. Currently, he is engaged in research and development of next-generation AI systems in Advanced Technology Division.

Tomohiro Mori
AI Engineer: Square-Enix Advanced Technology Division. He researched about CharacterAI and Animation in Future University Hakodate. He joined SquareEnix in 2019. He is researching AI and Animation
- Full Procedural Animation
- Character AI using Hierarchical Task Network

Youichiro Miyake
Lead AI researcher: Square-Enix Advanced Technology Division. Graduated from Kyoto University. He has been engaged in the development and research of artificial intelligence in digital games since 2004. He is also the chair of the Japan Game AI Specialty Group of the International Game Developers Association, a board member of the Digital Game Society of Japan, a board member of the Japan Society for Arts and Sciences, an editorial board member of the Japanese Society for Artificial Intelligence, and a CEDEC committee member.

Gustavo Martins
R&D Engineer: Square-Enix Advanced Technology Division. He has been involved in the game industry since 2012, starting his career at EA Canada as a Technical Artist. He has worked in the development of titles such as:
- FIFA 14
- Final Fantasy XV
- Kingdom Hearts III
- Resident Evil 2

Currently, he's working as a R&D Engineer at Square Enix, researching and developing new technologies for games.

Shinpei Sakata
Technical Artist: Square-Enix Advanced Technology Division
- Final Fantasy XIV

References

Sterren, W.v.d. 2013 “Hierarchical Plan-Space Planning for Multi-unit Combat Maneuvers”, GAME AI PRO, chapter 13, pp.169-183, A K Peters/CRC Press.

Humphreys, T. 2013 “Exploring HTN Planners through Example”, GAME AI PRO, chapter 12, pp.149-167, A K Peters/CRC Press.

Straatman, R; Verweij, T; Champandard, A; Morcus, R; Kleve, H. 2013 “Hierarchical AI for Multiplayer Bots in Killzone 3”, GAME AI PRO, chapter 29, pp.377-390, A K Peters/CRC Press,

Sailer, F; Buro, M and Lanctot, M. 2007 “Adversarial Planning Through Strategy Simulation”, IEEE Conference on computational intelligence and games 2007

Soemers, D.J.N. and Winands, M.H.M. 2016 “Hierarchical Task Network Plan Reuse for Video Games”, IEEE Conference on computational intelligence and games 2016

Orkin, J. 2006 “Three States and a Plan: The AI of F.E.A.R” Game Developers Conference 2006