11

A Character Decision-Making System for *FINAL FANTASY XV* by Combining Behavior Trees and State Machines

Youichiro Miyake, Youji Shirakami, Kazuya Shimokawa, Kousuke Namiki, Tomoki Komatsu, Joudan Tatsuhiro, Prasert Prasertvithyakarn, and Takanori Yokoyama

11.1 Introduction

Behavior trees and state machines were originally separate techniques—each with their own positive and negative points. The intent behind behavior trees is to make a series of character behaviors, whereas the intent behind finite-state machines (FSMs) is to make a stable cycle of character actions (Miyake 2015a, Miyake 2015b). For *FINAL FANTASY XV* shown in Figure 11.1, we have developed a new decision-making system that combines behavior trees and state machines into a single structure using the LUMINOUS STUDIO (SQUARE ENIX's game engine). This system has both the flexibility of behavior trees and the strict control of state machines as well as giving scalability to the development of a character decision-making system (Figure 11.2). This new decision-making system, which we call the AI Graph, extends the node formalism to enable sharing nodes between FSMs and behavior trees, provides advanced techniques for code reuse using trays that organize code reuse and behavior blackboards, and also provides many features for integrating with detailed low-level character behavior (Miyake 2016a).

Level designers can make a multilayered decision-making system for each character by using a visual node graph tool called the AI Graph. For example, for the first step, a level designer makes a top-layer state machine with several states by setting and connecting state machine nodes. Then the level designer can make a new state machine as a substate of one or more of the top-level states, or the designer can also make a new behavior tree inside any of the top-level states. Furthermore, the level designer can then make new state machines or behavior trees inside each subsequent substates. In this way, the level designer can make a hierarchical structure of state machines and behavior trees by simply editing nodes on the tool.

Each layer of the AI Graph also has a blackboard system by which the designer can register variables used in the game. By connecting the blackboards of separate nodes, the different layers can share and use these variables.

Figure 11.1

FINAL FANTASY XV screenshot.

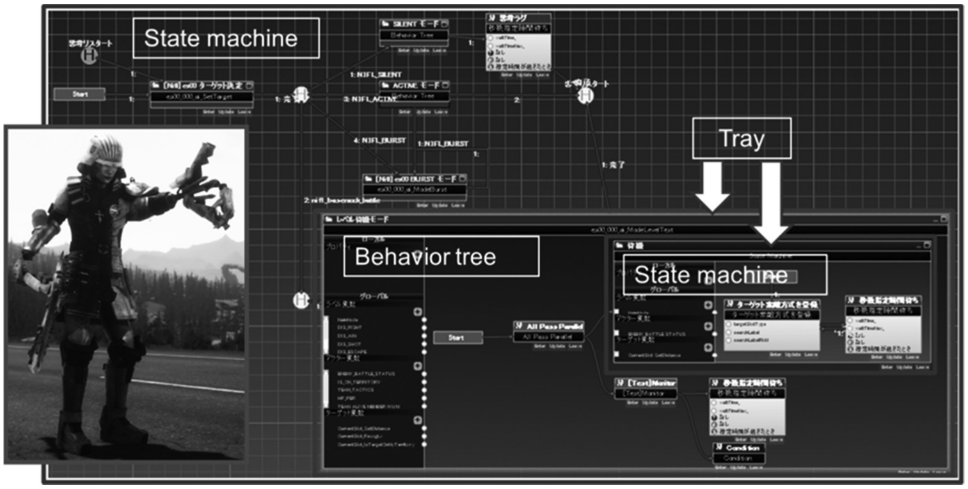


Figure 11.2

AI Graph image.

The AI Graph system has a real-time debug system that connects to and communicates with the game's run-time. Active nodes are highlighted on the decision graph tool as they are executed. During development, this makes finding any problems in the decision graph much easier. AI Graph maintains scalability, variation, and diversity in character AI design through the course of development because of its data-driven approach. In this chapter, we will explain the AI Graph structure, operation principle, and examples from *FINAL FANTASY XV*.

FINAL FANTASY XV is an RPG game in which a player travels in a large open world with three buddies while they fight with monsters and enemies in real time (Figure 11.1). All characters have intelligence to make their decisions by themselves. Also for the player character, AI supports the player character's behaviors.

11.2 AI Graph Structure and Operation Principles

AI Graph is a node-based graph system in which it is possible to make a hierarchical structure with a GUI-based node graph tool. The tool works both offline and while the game is running.

By using the AI Graph tool, a user can make a state machine or behavior tree for each layer (Figure 11.3). To make the next layer, a user can select one node and make a state machine or behavior tree in it. In this way, AI Graph makes a hierarchical nested structure. As requirements change, the hierarchical nested structure allows developers to make as many layers as they want. Finally, the AI Graph generates the data to be executed by the AI program.

The hierarchy executes in the following manner. When a node in the state machine or behavior tree contains another layer, it immediately executes that next layer. The process continues executing nodes until it cannot go to a deeper layer. It then returns to a higher layer after finishing a lower layer.

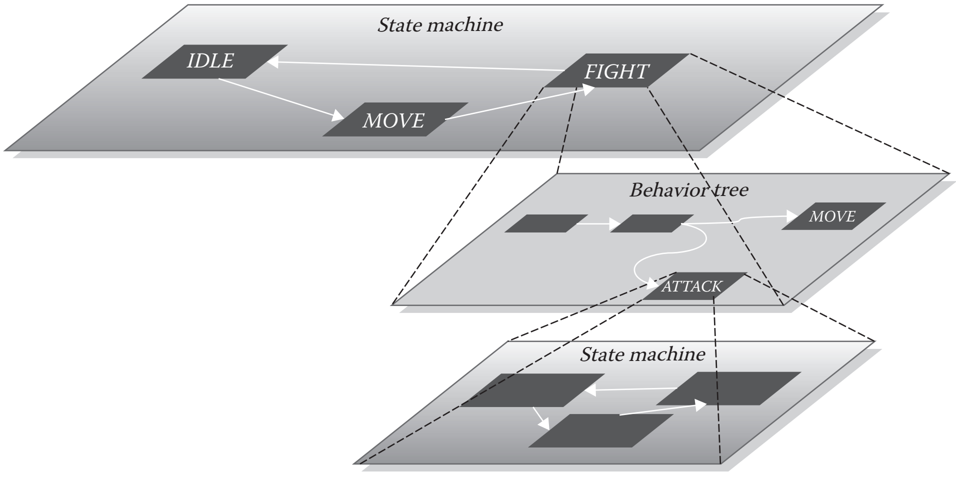


Figure 11.3

AI Graph model.

When a state machine's transition happens in an upper layer, the state currently executing lower layers must be finished. In this case, after all processing of lower layers has finished, the transition occurs. (See Section 11.5.3 for dealing with interruptions.)

11.3 AI Graph Tool

AI Graph tool is one part of the SQUARE ENIX game engine used to make a character's AI. It has three regions (Figure 11.4). The center of the screen is a field to build a state machine and behavior tree graph by connecting nodes. The left vertically long window

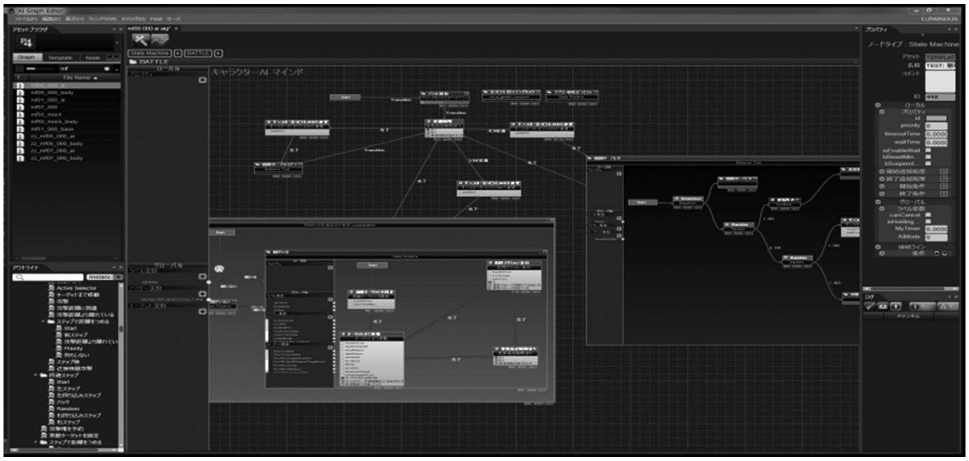


Figure 11.4

AI Graph tool screenshot.

shows variables and nodes that are already made and can be reused. The right vertically long window shows properties for customizing a node and is called the property window. A node can be connected with another node by an arc. In a state machine, a node denotes a state, and an arc indicates transition of the state. In a behavior tree, a node denotes a behavior or operator of the behavior tree, and an arc is used to express the behavior tree structure.

A tray is used to enclose a state machine or a behavior tree. This enables a user to move one entire state machine or behavior tree by moving the tray, and it is also easy to see the layered architecture through the tray hierarchy.

11.4 Implementation Techniques of the AI Graph Node

In the AI Graph, all nodes are reused. For example, a node that can be used in a state machine can also be used in a behavior tree. But ordinarily, the execution method of state machines and behavior trees is different. To make it possible for an AI node to be executed in both a state machine and behavior tree, each AI Graph node has four methods:

1. Start process (when a node is called)
2. Update process (when a node is executed)
3. Finalizing process (when a node is terminated)
4. A condition to signal termination

For both a behavior tree and state machine, the start process, the finalizing process, and the update process are necessary to begin to execute, finalize, and execute a node. The difference between them is what causes stopping a node. For a behavior tree, a node terminates itself by judging an internal terminate condition, whereas a state machine node is terminated by an external transition condition. Thus if a node has these four components, it can be executed in both behavior trees and state machines.

11.5 AI Graph Features

The following describes features of the AI Graph.

11.5.1 The Blackboard within the AI Graph

Variables can be shared via a blackboard consisting of two types (Figure 11.5). One is a local blackboard, which belongs to a tray. Variables of a local blackboard can be shared only in that local blackboard. The other is the global blackboard. Variables of the global blackboard can be shared with the game and all characters' individual AIs. In the AI Graph tool, both blackboards are shown on the left side. In Figure 11.5, you can see there are several variables listed. In the tool, two connected blackboards can share variables.

These variables are used to describe the properties of a node, the transition conditions of a state machine, and so on. For example, the global variable "IS_IN_CAMERA" means whether an actor is in camera or not, and this variable can be used to describe a transition condition inside a state machine contained in a tray.

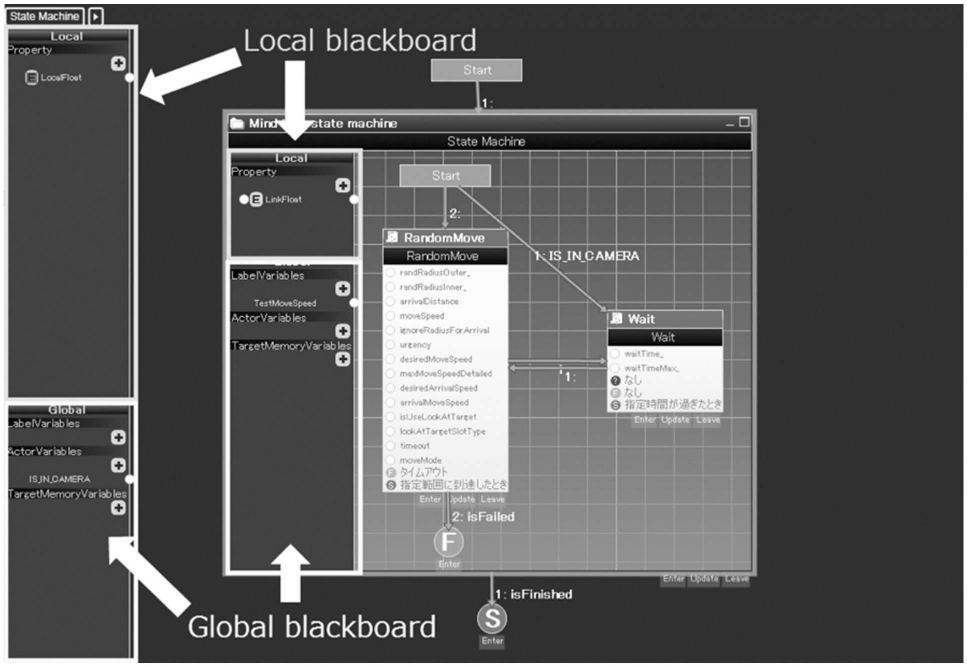


Figure 11.5

Blackboard architecture.

11.5.2 Parallel Thinking within the AI Graph

For some situations, a character must think about two things at a time. The AI Graph allows a character to have two concurrent thinking processes, and it is better to make two simple graphs rather than one big complex graph (Figure 11.6).

For example, one thinking process is a simple state machine to set a character behavior, and the other is a simple state to cause the character to look at a target that suddenly appears. The one state machine begins from a "START" node, and the other state machine begins from "PSTART" node.

The two state machines are executed concurrently. So the character can look around and search for a new target while it keeps attacking. Further, a behavior tree can execute two processes by a parallel node. For example, one behavior is to decide on a target and the other is to approach and attack.

11.5.3 Interrupting the Thinking Process

Often, a character will need to interrupt its current execution and execute another specific action. An interrupt node interrupts a process in the AI Graph when an interrupting condition is satisfied, and it executes the node linked to the interrupt node. For example, when a new game mission starts, monsters must rush to a player. After rushing toward a

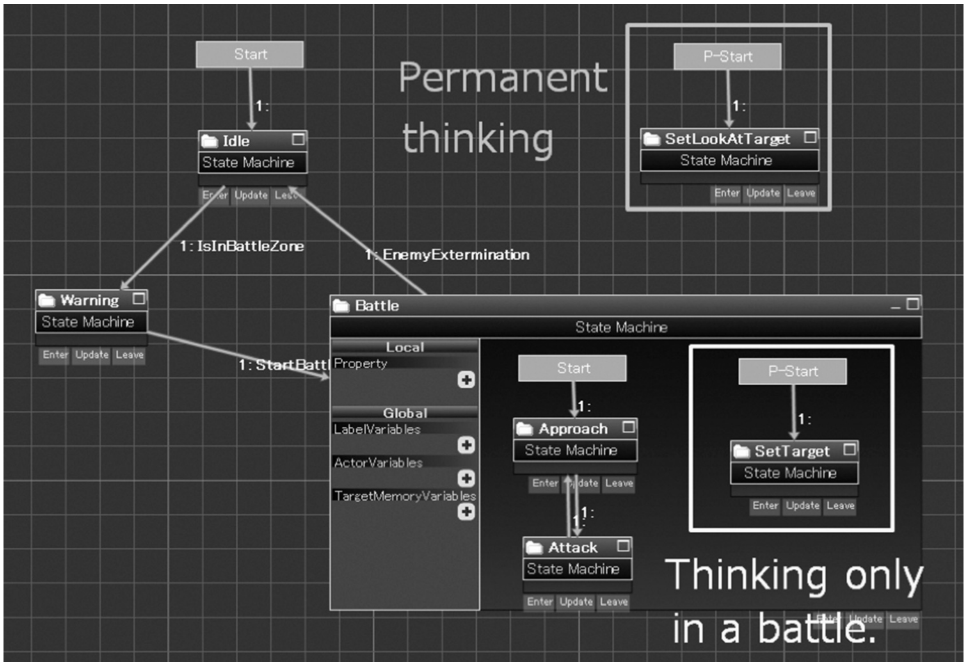


Figure 11.6

Parallel thinking.

player's position, they begin their original thinking process. In this case, two AI Graphs are prepared. One AI Graph includes an interrupt node (Figure 11.7). It causes the current tray to stop and the other tray process to start when the transition condition connected to the interrupt node is satisfied. And after the tray process finishes, the process returns to the original process.

11.5.4 Data and Overrides

An AI Graph can be saved as an asset file. If an AI Graph is fundamental for a character, it is repeatedly called and used. But an AI Graph often requires changes, because it needs to be specialized for each character. For example, when a state machine is saved as an asset file, a user might change a state of the state machine to customize the behavior. In this case, a function to change a node is called an "override," much like in C++.

In *FINAL FANTASY XV*, there are many different types of monsters. For all monsters, the top layer is set through a common template, but their fighting states are different. Therefore, the fighting state is overridden for each monster. Furthermore, a monster's AI Graph can be created by overriding the graph repeatedly from the common logic to monster battle logic (Figure 11.8). In this way, overriding methods make the AI Graph development easier and more effective.

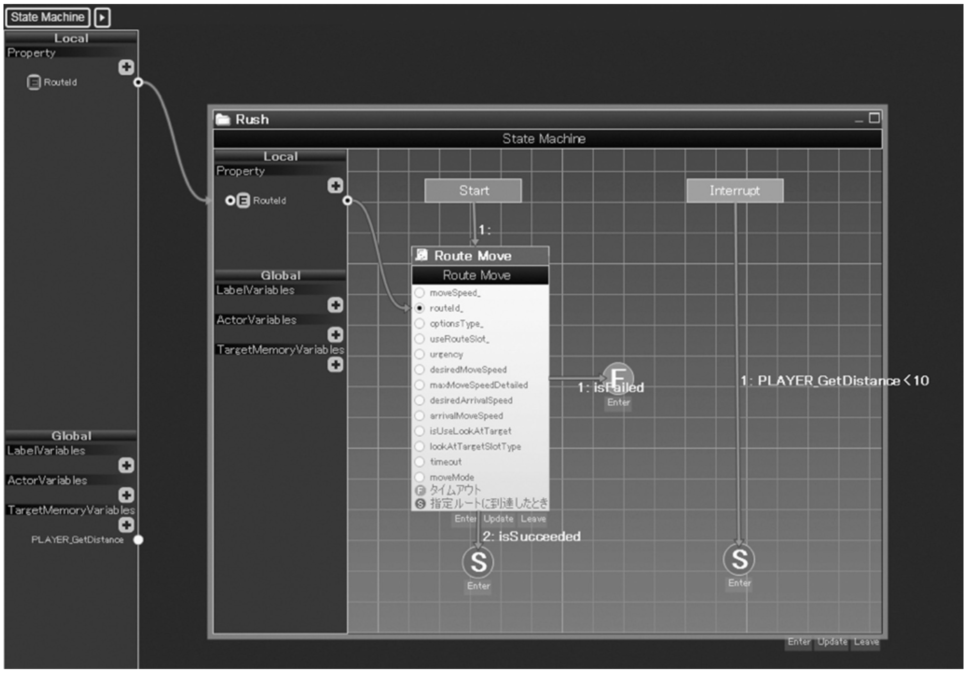


Figure 11.7

Interrupting an AI Graph.

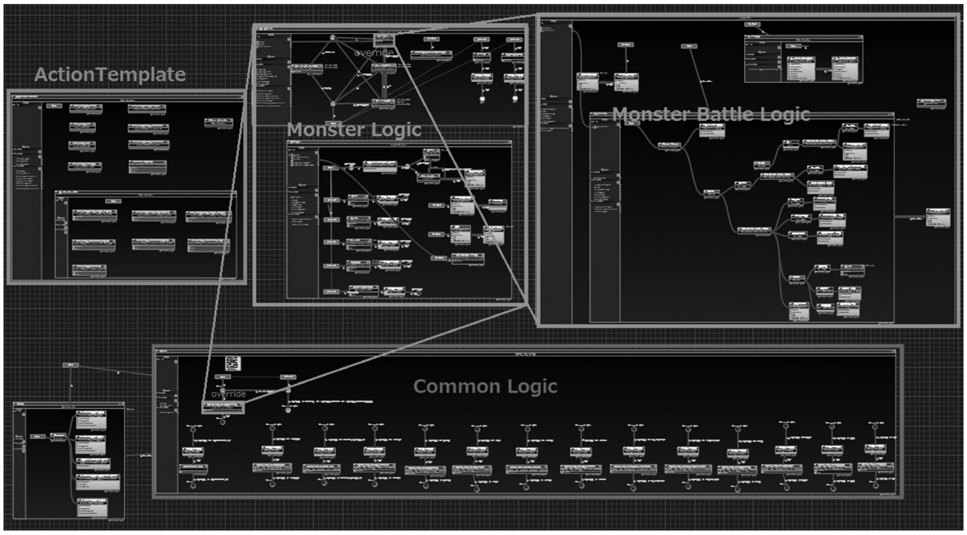


Figure 11.8

Overriding a monster's AI Graph.

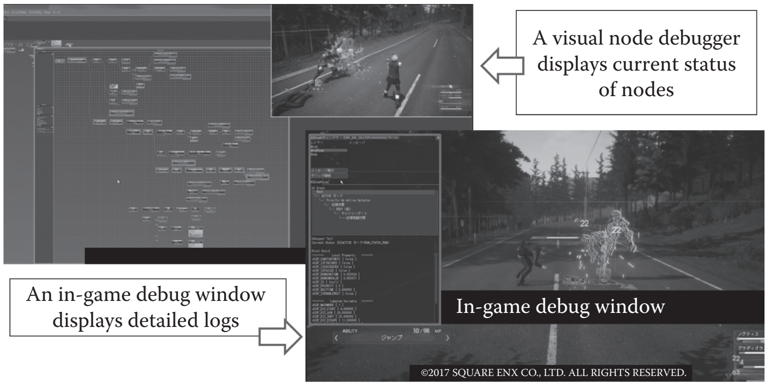

A visual node debugger displays current status of nodes
An in-game debug window displays detailed logs
In-game debug window
©2017 SQUARE ENX CO., LTD. ALL RIGHTS RESERVED.

Figure 11.9

Visual node debugger (a) and in-game debug window (b).

11.6 Debugging with the AI Graph

For AI development, fast iteration is one of the most important features to keep the AI improving until the end of development. As such, a user should be able to reload an AI Graph without compiling when they want to make a change. In our AI Graph Editor, an AI Graph can be compiled in the Editor independently from other systems' code. This is an example of a data-driven system.

There are two debug windows (Figure 11.9). While a game program runs, an AI Graph keeps a connection with the program. This is called the visual node debugger. In this debugger, the active node currently being executed is highlighted in green. This enables a user to trace the active node in real time.

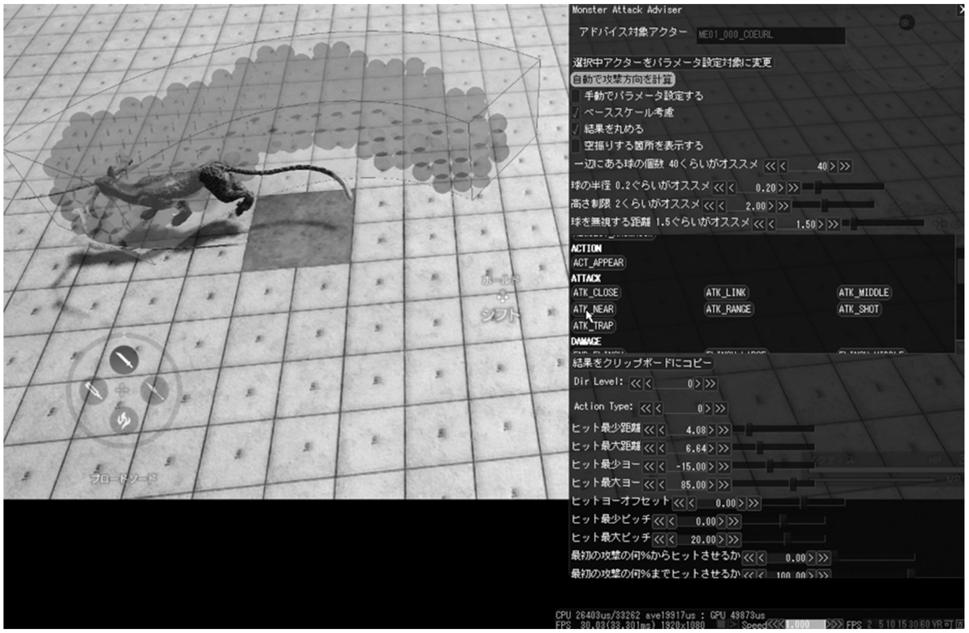
The other debug window is in a game window. The window displays detailed logs that are generated from a character's AI Graph and AI Graph variables.

11.7 Extracting Animation Parameters through Simulation

In the early stages of development, some monsters' attacks could not reach a player because the attack distance was not large enough. Our solution was to simulate a monster's attack motion, measuring and storing the exact attack distance for each move.

During development, many spheres were distributed around a monster to find the orbit of the monster's attack motion (Figure 11.10). If the motion hits a sphere, the sphere is marked. All marked spheres show the orbit region of the monster's motion. Then the region can be approximated by simple solid figures such as a sphere and sector, and parameters such as the attack distance and attack angles are extracted. These parameters are assigned to an attack node of the AI Graph as the attack parameters. When the node is executed, these new parameters are used when a monster attacks a player.

Manually adjusting AI parameters would have taken too much time during development. This method of analyzing the motion through simulation adjusts the AI parameters automatically and reduces development time.



```
Monster Attack Adviser
アドバイス対象アクター ME01_000_COEURL
選択中アクターをパラメータ設定対象に変更
自動で攻撃方向を計算
[ ] 手動でパラメータ設定する
[✓] ベーススケール考慮
[✓] 結果を丸める
[ ] 空振りする箇所を表示する
一辺にある球の個数 40くらいがオススメ << <   40 > >>
球の半径 0.2ぐらいがオススメ << <   0.20 > >>
高さ制限 2くらいがオススメ << <   2.00 > >>
球を無視する距離 1.5ぐらいがオススメ << <   1.50 > >>
ACTION
ACT_APPEAR
ATTACK
ATK_CLOSE        ATK_LINK         ATK_MIDDLE
ATK_NEAR         ATK_RANGE        ATK_SHOT
ATK_TRAP
DAMAGE
結果をクリップボードにコピー
Dir Level: << <   0 > >>
Action Type: << <   0 > >>
ヒット最少距離 << <   4.08 > >>
ヒット最大距離 << <   6.64 > >>
ヒット最少ヨー << <   -15.00 > >>
ヒット最大ヨー << <   85.00 > >>
ヒットヨーオフセット << <   0.00 > >>
ヒット最少ピッチ << <   0.00 > >>
ヒット最大ピッチ << <   20.00 > >>
最初の攻撃の何%からヒットさせるか << <   0.00 > >>
最初の攻撃の何%までヒットさせるか << <   100.00 > >>
CPU 26403us/33262 ave19917us : GPU 49873us
FPS  30.03(33.301ms) 1920x1080   Speed<<< 1.000 >>> FPS 2 5 10 15 30 60 VR 可
```

Figure 11.10

Attack motion analysis in simulation.

11.8 Cooperation of Characters via "Meta-AI"

Our meta-AI (more commonly called an AI Director) is an AI that monitors the game and dynamically changes the situation by giving characters orders (Miyake 2016b).

In *FINAL FANTASY XV*, the meta-AI arranges battle sequences. It monitors a battle situation and each character's behavior. When a player or the buddies get into danger, the meta-AI will select one of the buddies who is most appropriate to help (e.g., the nearest buddy who is not attacking). The meta-AI gives the selected character an order to go help the character in danger (Figure 11.11). Buddies' decision-making always depends on the AI Graph. But when a buddy receives an order from the meta-AI, it must stop its AI Graph and obey the meta-AI's order.

In a battle, the meta-AI can give four kinds of orders as follows:

1. Save a player or a buddy in danger.
2. When a player is surrounded by enemies, allow a player to escape.
3. Follow an escaping player.
4. Obey the team tactics.

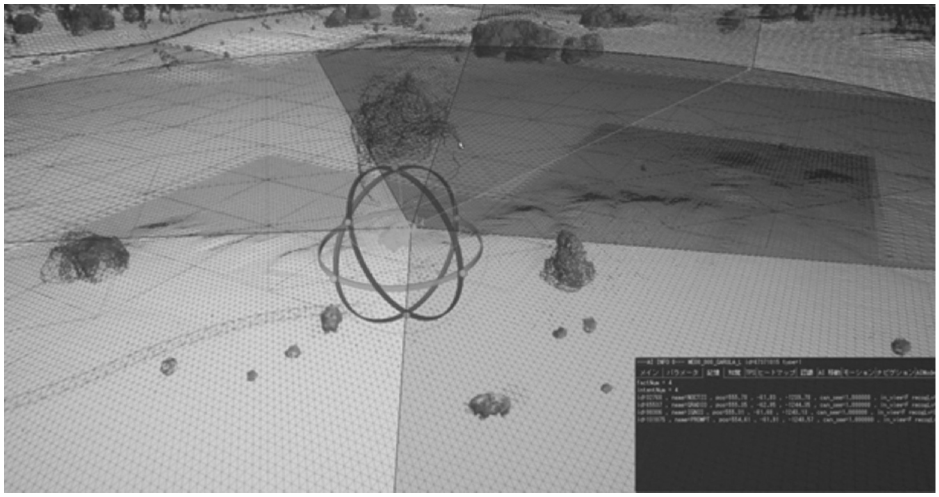
By using these orders, a meta-AI can tighten a battle flow and can control a player's tension and relaxation.

Figure 11.11

Meta-AI gives an order to save a player to a buddy.

11.9 Sensors

A monster's visual sensors consist of two fan-shaped regions (Figure 11.12). One is a wide fan-shaped region, and the other is a narrow fan-shaped region to detect enemies more precisely. When enemies are in these regions, they will be assigned to a target list.



```
factNum = 4
intentNum = 4
[illegible]
```

Figure 11.12

Monster sensor system consisting of two fan-shaped regions.

There is a node to select a target in the AI Graph. In this node, a user can select a targeting mode which is a way to select one target from a target list. Such a target mode can be customized by parameters such as min distance, max distance, min angle, max angle, and priority setting. A priority setting is the parameter type used to decide an enemy's priority in a target list.

11.10 Rule-Based AI System and the AI Graph

For some monsters, a rule-based system and AI Graph are combined. For these monsters, an AI Graph for the top layer is fixed. But there is an independent rule-based system, which includes many rules. It always checks which rule can be fired. Then it selects one of the rules, which calls a corresponding AI Graph template. This is a very simple system with the benefit that one rule condition perfectly corresponds to a single AI Graph to be executed. Although the degree of freedom is partly limited, simplicity of data and the ease of maintenance are clear benefits in our case.

11.11 Body State Machine and the AI Graph

In our game, a character system consists of three layers: an AI layer, a body layer, and an animation layer. These three modules send messages to each other and share variables via blackboards. The AI Graph does not directly initiate animation data. The AI Graph sends a message to the animation layer via a body layer, which consists of a state machine. Especially for shooting and damage behavior, the AI Graph calls the special control nodes within the body layer.

This three-layered architecture separates the roles to control a character, separating concerns between intelligence and animation. It also avoids increasing the size of an AI Graph (Figure 11.13).

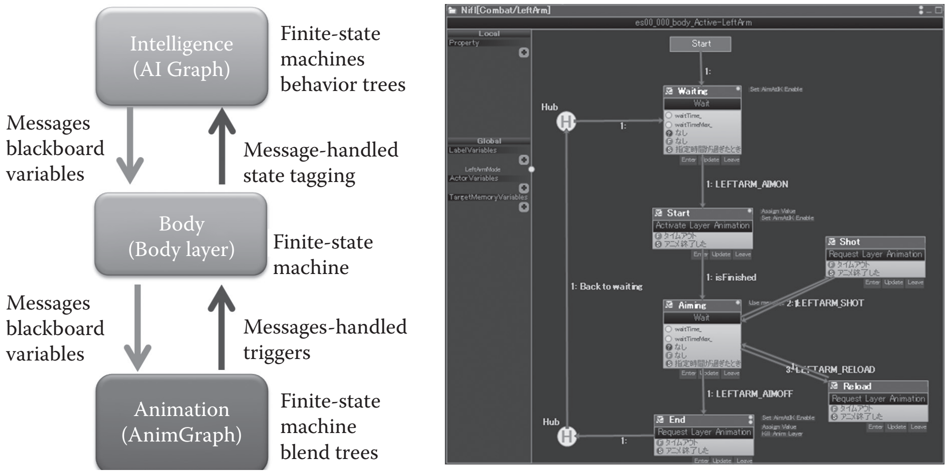


Figure 11.13

Three-layered character system.

A body layer represents a character's body as a node of a state machine. For example, a character's body state is expressed as running, jumping, or climbing a ladder.

A body layer has two roles:

1. Restricting character's actions, a body state can prohibit some actions in this state. For example, while a character is climbing a ladder, it cannot shoot using its hands.
2. Informing a change of body state to the AI layer. Sometimes a character body takes a reactive action to external force or damage. When a reactive action happens, only the body layer knows the information. Then it must send the information by a message to the AI layer. The AI layer will use it for decision-making.

11.12 Conclusion

As game environments and rules become more complex, a character is required to behave more smoothly and intelligently. When development for a next-gen AI began, we realized that it was critical to improve our AI tools. After many discussions within the team, the idea to combine state machines and behavior trees was agreed upon. This allows our developers to leverage both techniques in a nested hierarchical node structure, enabling a very flexible architecture. We called this tool the AI Graph Editor, and it was critical to completing *FINAL FANTASY XV*, which was released in 2016. Additional videos for each technical topic are available in PDF form (Shirakami et al. 2015).

All figures ©2016 SQUARE ENIX CO., LTD. All Rights Reserved. Main character design by TETSUYA NOMURA. All other trademarks are the property of their respective owners.

References

Miyake, Y., 2016a. A multilayered model for artificial intelligence of game characters as agent architecture, in *Mathematical Progress in Expressive Image Synthesis III*, *Volume 24 of the series Mathematics for Industry*, pp. 57–60. http://link.springer.com/chapter/10.1007/978-981-10-1076-7_7.

Miyake, Y., 2016b. Current status of applying artificial intelligence in digital games, in *Handbook of Digital Games and Entertainment Technologies*, Springer, 2016, http://link.springer.com/referenceworkentry/10.1007/978-981-4560-52-8_70-1.

Miyake, Y., 2015a. Current status of applying artificial intelligence for digital games, *The Japanese Society of Artificial Intelligence*, 30(1), 45–64. doi:10.1007/978-981-4560-52-8_70-1.

Miyake, Y., 2015b. AI techniques for contemporary digital games, *SA '15: SIGGRAPH Asia 2015 Courses*, November 2015, http://dl.acm.org/citation.cfm?id=2818164.

Shirakami, Y., 2015. Miyake, Y., Namiki, K., and Yokoyama, T., Character Decision Making System for FINAL FANTASY XV -EPISODE DUSCAE-, CEDEC 2015, In SQUARE ENIX PUBLICATIAONS, http://www.jp.square-enix.com/tech/publications.html, http://www.jp.square-enix.com/tech/library/pdf/2015cedec_FFXV_AI_English_part1.pdf, http://www.jp.square-enix.com/tech/library/pdf/2015cedec_FFXV_AI_English_part2.pdf