Chapter 3
Behavior Trees:
Introduction and Memory-Compact Implementation

Björn Knafla
Bjoern Knafla Parallelization + AI + Gamedev, Germany

Alex J. Champandard
AiGameDev.com, Austria

ABSTRACT

Behavior trees (BTs) are increasingly deployed in the games industry for decision making and control of non-player characters (NPCs, also named agents or actors) and gameplay. Behavior trees are incrementally created from reactive and goal-oriented behaviors by hierarchically compositing purposeful sub-behaviors. Composite behaviors (branches in the tree) decide which child behavior to run when and react to changing behavior execution states. Leaf nodes of the behavior tree check conditions, or control the execution of game state affecting actions. This chapter introduces a basic behavior tree concept, describes how behavior tree traversal drives NPC decision making via an example, and sketches a memory-compact implementation.

INTRODUCTION

Games are about entertainment and a great player experience. According to Chris Butcher from Bungie (Valdes, 2004), non-player characters (NPCs), often called agents or actors, should behave in ways that the player believes them to be intelligent. Artificial intelligence in games is more concerned with believability of the computed behavior than with simulating intelligence itself. Big and rich behavior repertoires enable agents to react to many situations in a variety of ways so they seem to have common sense as Isla puts it (2005).

Creating such huge sets of behaviors, organizing the decision-making process when to behave

DOI: 10.4018/978-1-4666-1634-9.ch003

in which way, and maintaining and iterating on the collection of decisions and actions to tune them for fun, is challenging. Part of the challenge stems from the balancing act between authoring autonomous and reactive agents while not losing control over the gameplay experience, which should adhere to the intended game design.

Behavior trees (BTs) are increasingly employed throughout the games industry to meet these challenges (Champandard, 2011). An agent's behavior is modularly assembled and structured by hierarchically organizing sub-behaviors - decisions, actions, or (sub-)trees formed out of decisions and actions - which are successively connected to form larger, more expressive behavior trees. These connections between decision nodes and their children are the sole way to model relationships between behaviors.

To update or tick the behavior of an agent during a simulation step, its behavior tree is searched best-first to find the most appropriate action(s) to run in the current game world situation which is (are) then executed. Traversal is steered by the semantics of decider nodes and the behaviors execution states.

The following topic background is briefly mentioned before explaining a concept of behavior trees based on a small example. Then a memory-compact implementation approach is described. Afterwards, future study and research directions are noted and the chapter is concluded.

BACKGROUND

Scripting and finite-state machines (FSMs) have been the primary tools to model decision making, and action selection and execution for computational behavior of agents in games (Champandard, 2007).

Scripting offers the most direct control over the gameplay in a scene. However, it also complicates reuse, analyzation, and maintenance of behaviors, especially if the controlled entities should remain reactive to the actions of the player (Champandard, 2007).

Finite-state machines provide a simple and intuitive mechanism to model reactive behavior (Champandard, 2007). In games, finite-state machines aren't used to accept languages, but to process and react to continuous streams of events or inputs (Fu & Houlette, 2004). A (finite) set of states corresponds to behaviors while (agent external or internal) events trigger transitions between states.

Diagrams of finite-state machines look conceptually simple but they hide many details necessary to implement them as Isla (2005) observes, e.g., when to follow a transition? Some transitions are voluntary, e.g., when a guard decides to idle instead of searching on for an intruder, while others are forced by an event, e.g., when a trespasser enters a guard's viewing field and the guard transitions from idle to the attack state.

The more states and transitions are used, the more labor intensive it gets to maintain and understand the flexible transition wiring. Hierarchical finite-state machines (HFSMs) lower this complexity by enabling reuse of smaller HFSMs inside of states of higher-level HFSMs and by allowing transitions between states in the different levels of the hierarchy (Millington, 2006). Though even with hierarchical finite-state machines Isla's critique and the complexity due to the ad-hoc transition wiring remains, e.g., there is no way to express sequences or explicit selection of behaviors in a straightforward manner with HFSMs.

Isla's article from 2005 about the artificial intelligence (AI) in Halo 2 describes how behavior trees can be used to handle the complexity of behavior modeling in games. Since then, different flavors of behavior trees have been deployed in addition to finite-state machines by increasing numbers of game developers (Champandard, 2011; Hecker, 2009; Pillosu, 2009).

Champandard broke down the different concepts and extended them with his own and thereby