The Integration of Autonomous and Scripted Behavior through Task Management

Lars Lidén

Presto Studios Inc.
5414 Oberlin Drive, Suite 200
San Diego, CA 92121
lars@presto.com

Abstract

The behavior of computer controlled agents in video games is generally determined by one of two methods. The actions of agents can be *scripted*; in which case every move and action an agent takes is pre-specified before run-time. Alternatively agents can be programmed with *autonomous behaviors*; in which case actions are determined procedurally during run-time. Each method has advantages and disadvantages. The two methods can be seamlessly integrated using a task management system in which agents reconcile both scripted and autonomous behaviors in the form of assigned tasks.

Introduction

One of the principal tools for creating intelligent agents in computer games is scripting. A script is a hard coded description of how an agent behaves during specific predetermined situations that arise during a game. Scripts can encompass an agent's navigation and movement through the environment, what behaviors it expresses or the sounds and speech it makes at a given time. Scripting is of particular importance for adventure games where the actions of agents are essential for both developing a game's story and for interactive puzzles in which agents are required to behave in a predetermined fashion to aid the human player. Scripting has the disadvantage that an agent's behavior isn't by definition dynamic. Each agent is programmed to perform a specific action in a particular situation. Although some complexity can be added by randomly picking from a number of different scripts, players quickly pick up on the repetition of actions and the illusion of intelligent behavior is lost.

An alternative to scripting behaviors is to program agents to express autonomous behaviors using techniques such as hierarchical finite state machines. Based on the current state of the game, an autonomous agent generates its own actions based on a set of internal rules. Autonomous behavior is of particular importance for action and strategy games in which agents must react to a given situation in an intelligent and dynamic fashion. A major disadvantage with autonomously behaving agents is that they can not express unique and predetermined behaviors for special circumstances. This makes autonomous agents inappropriate for generating story or reacting to planed events in the game in a predetermined manner. As a result, games with autonomous agents often resort to cut scenes to build story.

The following paper describes a technique that fully integrates both the scripting and autonomous behavior approaches in a singular system. It has the advantage of scripting behavior, in that story building elements are easily incorporated into game play, while at the same time allowing for the dynamics possible with autonomous behavior. In addition, by integrating the two methods agents can dynamically alter their scripts in the middle of performing actions to respond to changes in game state that might otherwise break hard coded scripts.

Task Management

At their core, both autonomous and scripting behaviors can be broken down into smaller sets of atomic sub-tasks. Autonomous behaviors tend to be comprised of individual tasks, such as firing of a weapon, or expressing a bored behavior. Scripts, on the other hand, tend to be comprised of a large number of sub-tasks. A script, for example, might consist of directions for walking over to a particular location, chopping down a tree, getting firewood, walking back to a house, opening a door, placing wood in a fireplace and lighting it.

Autonomous and scripting behaviors can be integrated by breaking them into their sub-tasks and assigning them to agents. Tasks tell agents to do things such as attack an enemy, go to a specific location, or turn a hand-crank. The agents then act as task managers, negotiating their current list of assigned tasks. The behavior of an agent at any moment in time is determined by selecting a winning task.

Agents can then behave both autonomously or through scripted behaviors as particular situations may require. For autonomous behavior, agents follow of a set of tasks known as a *personality*. Switching between tasks in the personality is determined by internal state variables, such as the agent's *fear*, *aggression* or *boredom* level. One can change the personality of a given agent by choosing a different set of tasks. Scripting behavior is more complicated. A script assigns tasks to specific situations that arise in the world, such as an agent reaching a particular place, or an enemy triggering an event.

A given agent can switch back and forth between autonomous behavior and scripting behavior as the need arises, based on events in the game. Although games will vary, agents usually exhibit autonomous behavior almost all of the time, with scripting used on a limited basis.

Tasks

Each agent has a set of basic behaviors that are composed of *tasks*. At any point in time an agent exhibits the behavior given by its current active task. There are two fundamentally different types of tasks:

<u>*Behavioral tasks*</u> are tasks that directly control the behavior of the agent. They consist of actions such as "Wander Around" or "Flee".

<u>*Procedural tasks*</u> on, the other hand, are not used for dictating the behavior of an agent directly. Rather, they indirectly influence an agent's behavior by changing the script or personality that the agent is using or other properties of the agent.

Personalities and scripts are organized into *task lists.* A task list is composed of a set of *behavioral tasks* (along with the probability of each behavior being expressed) and any number of *procedural tasks.*

There are three types of task lists. Autonomous behaviors are governed by a *personality* task list**.** Scripted behaviors are governed by *parent* and *target script* task lists.

The behavior expressed by an agent is chosen from the behavioral tasks in the active task list based on its probability. The probabilities of any task list must summate to 100%. Any and all procedural tasks in the list are performed regardless of which behavioral task was chosen from the list. Task lists don't have to contain a procedural task, and they can contain multiple procedural tasks. During the course of its lifetime, an agent may changes its behavior by changing task lists.

A typical task list might be composed of:

ATTACK	50%
FLANK	30%
SET AMBUSH	20%
USE PARENT SCRIPT	"Collect Wood"

In the given example, the agent would chose to attack, flank or set an ambush with the given probabilities. Additionally, the agent would change its Parent Script to a new script called "Collect Wood".

When the same task list is used repetitively, the agent switches from one probabilistic task to another based on the probabilities of the task. For example, if a task list contains 50% *Wander Near* and 50% *Wander Around*, the agent will express *Wander Near* half of the time and *Wander Around* half of the time.

How does the agent know when to alternate between the two tasks? For most tasks, the length of time a selected behavior is expressed is determined by a predetermined timeout period. Agents express the chosen behavior for a certain amount of time, before randomly choosing from the task list again.

Other tasks, such as *Attack* and *Flank*, require that the action is completed before the agent randomly chooses from the active task list again, regardless of the amount of time it takes to complete that task.

Imagine a task list containing 50% *Attack* and 50% *Flank.* If the agent selects the *Attack* task, it completes one attack before randomly selecting between *Attack* and *Flank* again, and vice versa.

Personality

The autonomous behavior of an agent is represented by a 3x3 matrix of task lists known as its personality. The axes of the matrix are composed of two of the internal states of the agent, such as *Health***,** *Aggression***,** *Fear* or *Boredom***.** When expressing autonomous behavior, the agent chooses from the set of task lists based on the current values of the chosen internal states.

The values of the internal states range from 0-100 and are determined during runtime using parameters such as the distance to the nearest enemy and events such as being attacked.

For an aggressive creature, one might want to choose a Personality with *Aggression* and *Health* as the two relevant internal states. For a friendly NPC, on the other hand, one might want to choose a Personality based on *Fear* and *Boredom*. Each of these internal states is sub-divided into three levels: Low, Medium and High and have a range from 0 to 100.

The two selected internal states form a 3x3 matrix with the levels (low, medium and high) in each dimension. Each position in the matrix contains a task list that governs the agent's behavior when its internal states correspond to that position in the matrix. An example is depicted in Table 1.

	-	A G G R E S S I O N	+
+ H E	Wander Around 50% Face Target 40% Set Ambush 10%	Attack 20% Flank 30% Set Ambush 50%	Attack 100%
A L	Wander Around 60% Flee 40%	Attack 40% Set Ambush 50% Flank 10%	Attack 50% Flank 50%
T H -	Flee 100%	Set Ambush 50% Flee 50%	Attack 10% Flee 90%

Table 1: An Example Personality

In the example, *aggression* and *health* were chosen as the relevant internal states. When in autonomous behavior mode, the agent will attack 100% of the time when its aggression and health are both high. On the other hand, when aggression is medium and health is low, the agent has an equal chance of setting an ambush or fleeing.

Scripting Behaviors

As discussed previously, the use of autonomous behaviors will often not suffice. Game designers may want an agent to express a particular behavior at a particular time. For example, maybe a designer wants an agent to guard a particular area and only get aggressive when an enemy enters this region. Alternatively, maybe the designer wants an agent to lead an enemy down a predetermined path to a location where another agent has set a trap.

Such scripting tasks are usually tied to particular events or locations in the environment. In games, agents often employ a set of connected *nodes* known as a *network* to navigate the environment. Such a network can be appended to also contain scripts associated with particular nodes of the network. Two types of scripts can be associated with nodes: *parent scripts* and *target scripts.* Each node contains its own sets of parent and target scripts.

By default, when a node is created on a network, both the parent and target scripts are composed a special task called "*NoTask*". *NoTask* indicates that there is no scripting behavior associated with this node.

Parent Scripts

Parent scripts are used to initiate scripting events when an agent nears their vicinity. A parent script is a set of behaviors expressed by an agent when its nearest reachable node contains that script. For example, imagine that Node 7 contains the following parent script:

GO NODE 18	50%
TURN CRANK	50%

When the agent is in the vicinity of Node 7, the behavior associated with this task list will be expressed, rather than the autonomous behavior dictated by the agent's personality.

Target Scripts

Other agents in its environment, as well as the human player can also influence the behavior of an agent. For example, tasks such as *attack* or *defend* require that an agent know whom its enemies and friends are. It is also helpful for an agent know about other characters on its network so it can avoid bumping into them during navigation. Each agent is given a list of targets that influence its behavior. When an agent is in scripting mode, the location of other agents can be used to assign particular scripting tasks.

A target script, is similar to a parent script, however, the agent expresses the behavior of the target script when one of its targets is in the vicinity of the node with that target script. For example, imagine that Node15 contains that following target script:

ATTACK	100%

In this example, whatever the agent's position, when its current target is in the vicinity of Node15, the agent will express the attack behavior. One might use this, for example, to make the agent seem like it is protecting a certain region.

AllNode Script

In some cases one might want to create a scripting behavior which is universal to an entire network. For example, imagine that based on some event, we want an agent to go to a particular location and pull a switch, regardless of its position on the network. However, we don't know which node will be nearest to the agent when this event happens. If an *AllNode* script exits, the behavior of the *AllNode* script will be expressed when the parent script of the agent's nearest node is set to *NoTask.* For example, if an *AllNode* script of "PULL SWITCH" exists, an agent will go to and pull the switch whatever the current position on its network as long as its nearest node has no parent script.

Behavior Resolution

The behavior of an agent is resolved in the following manner. First the agent checks the nearest node to see if there is a parent script associated with that node. If there is one, it expresses the behavior of the parent script. If the parent script contains "*NoTask*", the agent then checks for an *AllNode* script. If none exists, the target script of the node that is nearest to its current most important target is checked. If there is a target script associated with that node, the agent expresses the behavior of that target script. If the target script also contains "*NoTask*", then the agent expresses the autonomous behavior associated with its personality.

Switching Scripts

It is also possible to have an agent switch between scripts. This can be useful when an agent is required to express different behaviors within the same network at different times. For example, initially, an agent may be assigned to script "Chop Wood", however, after some event may switch to parent script "Go to Store".

There are two ways in which an agent can switch the parent and/or target script to which it pays attention. First, a procedural task can change the target or parent script of an agent. As previously discussed, procedural tasks are tasks in a task list that control behavior indirectly. Secondly, events in the environment can trigger a change in behavior. Agents can receive messages or events from other objects that cause a change in their parent or target script.

Script switching can also be used as a way switch between autonomous and scripted behavior. For example, imagine a designer wants to set up an initial script (such as walking to a particular location and setting up an ambush). Once the agent has finished expressing the behavior associated with the script (the ambush has taken place), the designer wants the agent to return to expressing autonomous behavior. This can be accomplished by first assigning the agent to a set of parent scripts with tasks that express the initial desired behavior. Then, the designer can switch the agent back to expressing autonomous behavior, by changing the agent's parent script to another set of parent scripts that have *NoTask*s assigned to them. As all nodes are assigned *NoTask*, the agent will then express the autonomous behavior defined by his personality. In a similar fashion scripts can be interrupted or adjusted midstream as changes in game state require.

A Simple Example

Figure 1 and Table 2 give a cartoon version of how a simple script might be employed to make an agent collect wood until it is attacked, at which point it retrieves a weapon and attacks its enemy.

Initially the agent follows the "Get Wood" script that sends the agent to the woodpile. When the agent is done collecting wood an event changes it to the "Go Home" script that sends it to drop off the wood. Once the wood has been dropped, another event changes the agent back to the "Get Wood" script.

If the agent is attacked at any time during wood collecting, the agent switches to the "Get Weapon" script. Once the weapon has been retrieved, the agent switches to a "NoTask" script. The agent then defaults to its assigned personality that details how it will attack its enemy in a dynamic fashion. When the enemy is dead, the agent returns to collecting wood.

Although this example is a gross over simplification, it provides a rough idea of how multiple scripts can be combined with autonomous behavior. An actual game would employ path finding algorithm, rather than node-to-node scripting for navigation and the scripts would be composed of multiple tasks with assigned probabilities.

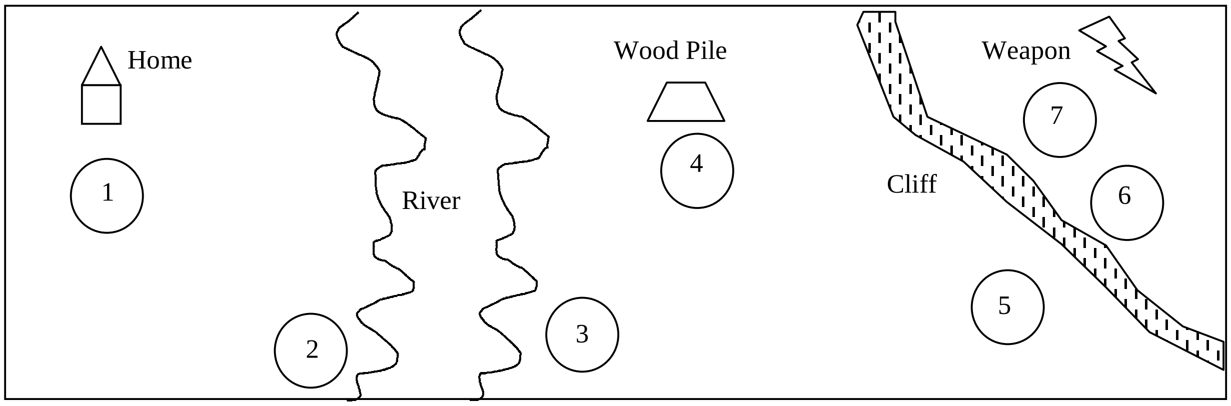
Conclusion

By representing autonomous and scripted behaviors as sets of atomic tasks, one can cleanly integrate both types of behavior. Game agents can respond dynamically during scripted events to changes in the game state while at the same time avoiding the repetition that often makes scripting seem non-realistic. While in autonomous mode, agents use internal state variables to select which behavioral tasks are expressed. When in scripting mode, agents use environmental tasks (associated with nodes) to guide behavior.

A variant of the techniques described in this paper is incorporated into Presto Studio's proprietary Sprint engine. Two action/adventure games, *Star Trek: Hidden Evil* and *Beneath*, have successfully employed the model.

	PARENT SCRIPT		
NODE NUMBER	*"Go Home"*	*"Get Wood"*	*"Get Weapon"*
Node 1	Drop (Wood)	Walk To (Node 2)	Walk To (Node 2)
Node 2	Walk To (Node 1)	Swim To (Node 3)	Swim To (Node 3)
Node 3	Swim To (Node 2)	Walk To (Node 4)	Walk To (Node 5)
Node 4	Walk To (Node 3)	Get (Wood)	Walk To (Node 5)
Node 5	Walk To (Node 3)	Walk To (Node 4)	Climb To (Node 6)
Node 6	Jump To (Node 5)	Jump To (Node 5)	Walk To (Node 7)
Node 7	Walk To (Node 6)	Walk To (Node 6)	Get (Weapon)

Table 2: A Cartoon Script


Home
1
2
River
3
Wood Pile
4
Cliff
Weapon
7
6
5