Chapter 6

High-Level Behaviors

6.1 Introduction

For the purposes of creating interesting animations for autonomous characters, some degree of high-level scripting of a character's behavior is needed. The general idea is to aggregate several lower-level task commands using a logical scripting language, in order to create a script (program) that defines a higher-level behavior.

This chapter describes how more complex high-level behaviors for animated characters can be constructed. The concepts are illustrated with simple scripted examples that have been implemented and integrated with the task-level navigation and manipulation planners presented in Chapter 3 and Chapter 5 respectively. The examples are discussed along with a summary discussion.

6.2 Programming Behaviors

Scripting Languages

High-level behaviors for animated characters are typically defined via scripts written in a scripting language. Many commercial applications (such as video games) employ some kind of scripting system for animating complex behaviors. Various researchers have also proposed different kinds of scripting systems for designing animated agents (for example, the PAT-nets and SCA loops of U. Penn [BWB$^+$95], Funge's CML scripts [Fun98], and the Improv system at NYU [PG96] which uses non-deterministic scripts). Although some systems make complex behaviors easier to program than other, all generally offer the same functionality. This is evident by the fact that any given scripting system can typically be programmed to emulate (simulate the behavior of) any other scripting system.

At a minimum, a scripting language should provide the ability to *loop*, perform *conditional branches*, and invoke commands for executing *atomic tasks*. The ability to *invoke other scripts* (possibly passing parameters to them) is not necessary, but very convenient for modularizing behaviors.

Atomic tasks are either low-level motor commands, high-level task commands (such as "moveTo", "getObject", "touchObject", etc), or commands to initiate the playback of a pre-defined motion, such as waving hello to another character (see Section 2.4). A script can define not only a sequence of atomic tasks that make up a more complex behavior, but also the logical relationships and conditions under which the atomic tasks should be performed.

System Integration

Communication between a scripting language and an animation system is accomplished via what we shall refer to here as *sensors* ("inputs") and *actions* ("outputs"). Sensors serve as feedback channels, providing abstract representations of what the character perceives in the environment. Actions correspond to atomic actions that can be performed by the character.

Using the data provided by sensors in conditional branches facilitates the ability to alter a character's actions depending upon what the character perceives in its environment. As a consequence, even very simple scripts with sensing feedback can yield quite complex behaviors and interactions between characters. This feedback loop corresponds to the practical implementation of the control loop of the "virtual robot".

The Generality of Scripting

With the necessary sensors and actions accessible through a scripting language, scripts are a fully-general method of programming a character. There is often conceptually no limit to the complexity of a script that may be defined (though limitations to their ability to be practically implemented may exist). Thus, even a very complicated behavior-based model can generally be simulated using a simple scripting language.

Some researchers have argued that parallel computation will fundamentally alter the abilities of software agents. In fact, parallelism itself does not provide any computational advantage, since any parallel machine can be simulated by a single-processor serial machine. However, the improved performance due to parallelism may enable algorithms that are too slow for serial machines to become practical.

6.3 Following and Pursuit Behaviors

Using a navigation algorithm as a scriptable action, interesting animations involving multiple characters can be created. For example, simply setting the navigation goal of one character to be the sensed location of another character immediately yields following or pursuit behavior.

We have created several example scripted behaviors using the navigation strategy presented in Chapter 3. Figure 6.1 shows an example involving two characters that have been scripted to follow another character that is under user control (in the foreground). The actual script executed is shown in Figure 6.2. Because path planning is used for navigation, the character will naturally circumvent obstacles while following the target character.

There are several possible strategies for avoiding collisions between characters. One simple idea is have each character "plan around" the others. For example, we can project the geometry of the other characters at their current locations prior to obstacle growth. Since each character may change its location over time, replanning may be necessary periodically in order to avoid the possibility of a collision between two characters. One simple coordination scheme might replan whenever other moving characters cross a character's current path.

More sophisticated schemes that account for character velocities are also possible. Instead of planning around the other characters at their *current positions*, each character could plan around the other characters' *predicted future positions*. A character's future position could be calculated by a simple extrapolation of its current actual or estimated velocity. The examples shown in Figure 3.12, Figure 3.13, and those in Figure 6.1 and Figure 6.4 utilize this technique in order to avoid potential inter-character collisions.

Increasingly complex animations can be constructed by building upon existing behaviors. For example, given the following (pursuit) behavior, it is quite simple to script several characters to play a game of tag, where one character who is "it" pursues nearby characters,

Figure 6.1: Two human characters scripted to follow another human character under user control.

```
while (TRUE) {
 if (! nearCharacter(target, 1.7))
  moveTo(target, 1.5);
 else
  stop();
}
```

Figure 6.2: An example following (pursuit) script.

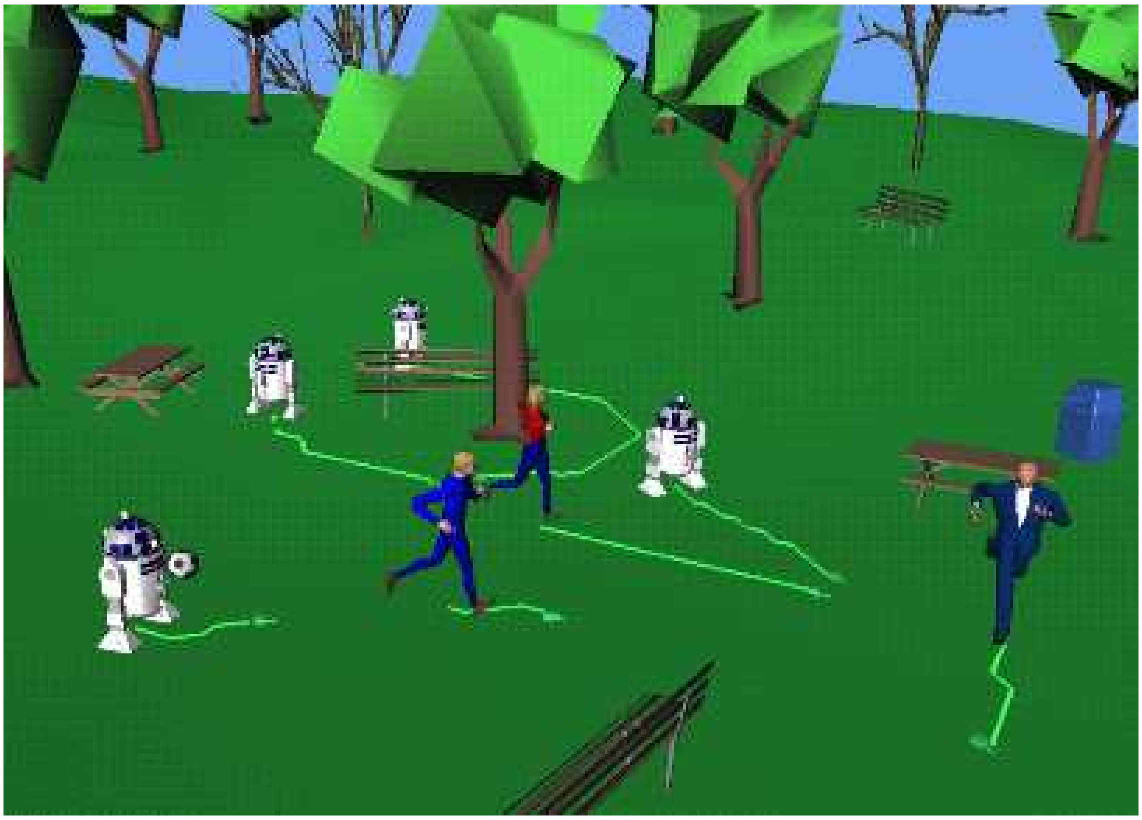

Figure 6.3: An scenario involving wandering and following behaviors for human and robot characters.

while the others attempt to flee.

Figure 6.3 and Figure 6.4 show examples involving seven characters (three humans and four robots) in different scenarios. In Figure 6.3, two human character are scripted to follow the other human character (which is under user control), while the robots are instructed to wander and follow any human that passes nearby. In Figure 6.4, all of the characters are instructed to pursue a single character under user control. Besides being applicable as a general pursuit behavior for video games, it can be used to animate a crowd following a leader (e.g. a virtual tour guide leading a group of virtual tourists).

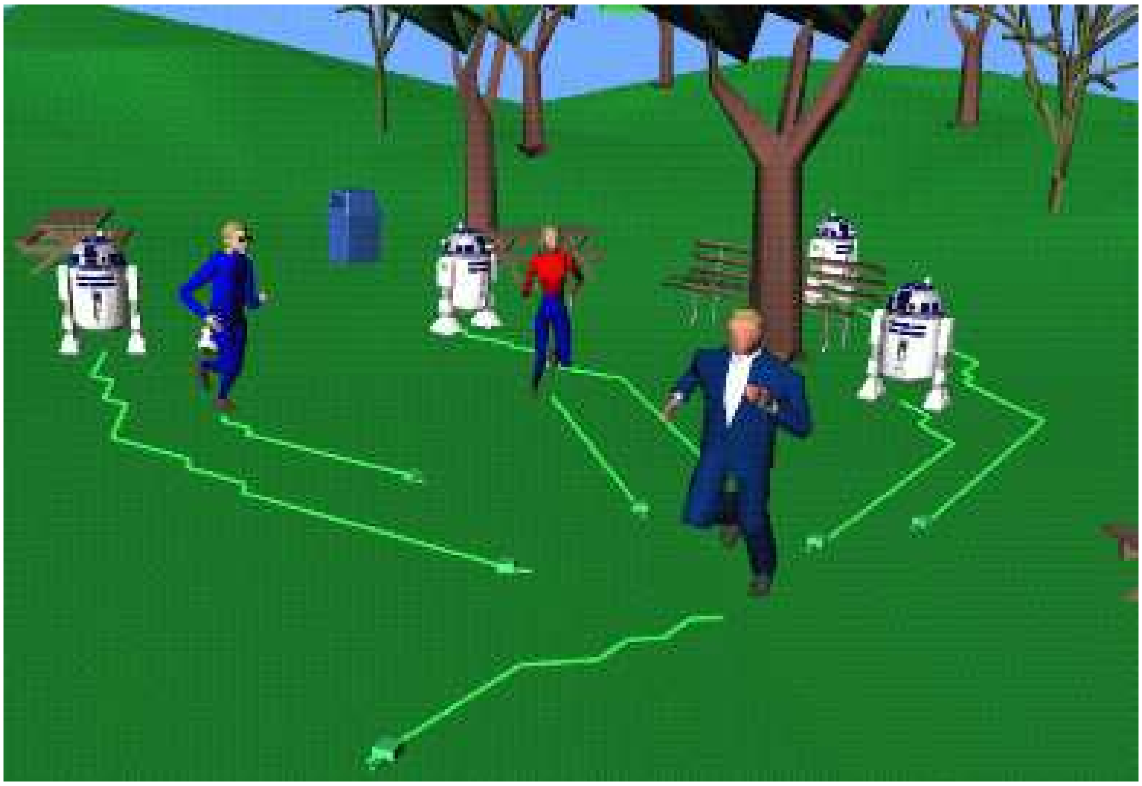

Figure 6.4: A group of characters scripted to pursue a user-controlled character.

6.4 Integrating Navigation and Manipulation

Given the two fundamental atomic task commands "moveTo" (navigation) and "getObject" (manipulation), one can script high-level behaviors that involve picking up and moving objects around. First, the navigation planner can be used to plan a motion for the character to move near a desired object. Then, the manipulation planner can be invoked to grasp and pick up the object.

For example, one could create a *cleaning behavior* for a virtual park employee in which the character wanders around picking up any litter it walks near. By utilizing the synthetic vision module of Chapter 4, the basic cleaning behavior could be modified to only pick up objects that the character *actually sees*. Synthetic vision could also be used to create a script that controls the character to actively look for litter instead of pure wandering. Modularizing behaviors (such a cleaning behaviors), allows us to integrate the behavior as a subgoal for more complex actions. For example, an autonomous waiter in a virtual cafe could roam around tables filling the patrons' cups of coffee, but if he happens to see some litter, the cleaning behavior can be invoked.

In the same way that a vast library of clip motions could potentially be used to animate specific tasks, a vast *library of behaviors* could be developed over time and provided as a resource to animators. As the number and complexity of the behaviors increases, the perceived "intelligence" of the the animated agents will likely also increase accordingly.