5
Playing Back the Animation

Because our algorithm does not store the animation's keyframes, the character's pose[1] needs to be be generated for each frame of animation. We chose to do this on-the-fly, calculating the pose before each frame is displayed on the screen.

The calculations that determine the pose depend on a solid physics simulation system. However, physics simulation systems are notoriously fickle when it comes to accuracy and consistency. It is often the case that running the same simulation several times will result in different outcomes.

Numeric methods (such as those used in real-time physics simulation) are inherently inaccurate, but we can at least ensure they are *consistent* by employing what [Valente et al. 2005] describe as the "fixed-frequency deterministic game loop." This technique is based on the fact that if we always update the game state at a fixed number of times per second (with the same delta-time, or dt), then the simulation will consistently produce the same result. See Algorithm 1, where the function **GameStateStep** contains the time-dependent logic that updates the character's pose. We will see two different implementations of this function, which correspond to the two animation representations discussed in Chapter 4.

Algorithm 1 This game loop ensures consistent results, even if the time between frames is variable. The value $FIXED\_DT$ is a constant number that represents the time interval between updates.

```
accumDT ← 0
function GameTick()
  dt ← time elapsed since last frame
  accumDT ← accumDT + dt
  while accumDT > FIXED_DT do
    GameStateStep(FIXED_DT)
    accumDT ← accumDT − FIXED_DT
  end while
end function
```

[1]Pose: The position and orientation of the character's bones.

5.1
Playing Back a Sequence of Commands

To play back the animation, one must define a time interval between op-code executions ($OPCODE\_DT$), and then run the animation program side-by-side with the physics simulation (see Algorithm 2). If the constant value $OPCODE\_DT$ is equal to $FIXED\_DT$, then the animation will advance one op-code per frame.

Algorithm 2 The following algorithm advances one frame of the animation using an op-code sequence representation.

```
accumOpDT ← 0
function GameStateStep(dt)
  {Run animation program:}
  accumOpDT ← accumOpDT + dt
  while accumOpDT > OPCODE_DT do
    Take next op-code from the program.
    Interpret its command.
    Modify active body components according to the command.
    accumOpDT ← accumOpDT − OPCODE_DT
  end while
  {Run physics simulation:}
  PhysicsStep(dt)
end function
```

5.2
Playing Back an Expression Tree

This representation is more sophisticated, simulating the character's sense of touch and proprioception[2]. We decided to evaluate the controllers' expression trees every frame, as if the character's response to external stimuli were immediate. Algorithm 3 is the resulting implementation.

Algorithm 3 The following algorithm advances one frame of the animation using an expression tree representation.

```
function GameStateStep(dt)
  {Run animation program:}
  for each controller do
    Evaluate the controller's expression tree.
    Modify active body components according to the evaluation results.
  end for
  {Run physics simulation:}
  PhysicsStep(dt)
end function
```

[2]Proprioception: A person's awareness of their own limbs' position relative to each other.