been any user since the last tick, send the appropriate messages to all objects requesting notification of that input.

(2) **Detonate alarms.** Send messages to any objects that have waited the requested number of ticks.

(3) **Detect collisions.** Either of the two previous steps may have resulted in objects appearing or moving on the screen. Now detect whether the boundaries of any pair of objects overlap or have overlapped since the last tick. This need not be an isolated step in the underlying system but, depending on hardware and efficiency considerations, might be performed in conjunction with primitives encountered in the previous two steps. In any case, by the time we get to the next step, we must have a complete list of all colliding pairs of objects.

(4) **Send collision messages.** For any pair of objects identified in the previous step, if either of the objects has declared that it cares about collisions with instances of the other object's class, then send it the appropriate message. New collisions resulting from activity during this step do not generate their own messages until the next tick.

(5) **Compose display buffer.** All processing involving programmer code is finished for this tick, so all objects have assumed their end-of-tick state. Now we can create the next display frame, based on the current state of the objects, and show it on the screen.

**Game development environment**

Gambit's syntax and semantics were designed to accommodate a highly interactive programming environment. A Gambit game consists of several modules, each of which is either a globals definition module or a class definition module. A class definition module consists only of the definitions of Gambit classes, and a globals definition module consists of variable, type, and constant declarations that the programmer would like to make visible within other modules.

Globals definition modules and class specification modules are Gambit's fundamental units of compila-