GPars - Groovy Parallel Systems

Jon Kerridge

Version 1.0, 2015-10-29

Table of Contents

Barriers and Buckets: Hand-Eye Co-ordination Test

In this chapter, three shared memory synchronisation techniques are combined to provide control of a highly dynamic environment.

Three Choices

A **Barrier** provides a means whereby a known number of processes collectively control their operation so they all wait at the barrier until all of them have synchronised with the barrier, at which time they are all released to run in parallel.

An **AltingBarrier** is a specialisation of the **Barrier** that allows it to act also as a guard in an **Alternative**. Finally, a **Bucket** provides a flexible refinement of a barrier.

Typically, there will be a collection of **Buckets** into which processes are placed depending upon some criterion. Another process then, subsequently, causes a **Bucket** to flush all its processes so they are executed in parallel.

These processes will, in due course, become idle, whereupon they place themselves in other buckets. The next **Bucket** in sequence is then flushed and so the cycle is repeated. **Buckets** can be used to control discrete event simulations in a very simple manner. The process that undertakes the flushing of the buckets must not be one of the processes that can reside in a **Bucket**.

The Aim

The aim of this example is to present a user with a randomly chosen set of targets that each appear for a different random time. During the time the targets are available, the user clicks the mouse over each of the targets in an attempt to hit as many of the targets as possible. The display includes information of how many targets have been hit and the total number of targets that have been displayed. The targets are represented by different coloured squares on a black background and a hit target is coloured white. A target, that is not 'hit' before its self-determined random time has elapsed, is coloured grey. There is a gap between the end of one set of targets and the display of the next set during which time the screen is made all black. The minimum time for which a target is displayed is set by the user; obviously the longer this time the easier it is to hit the targets. Targets will be available for a period between the shortest time and twice that time. `Figure 14-1` shows the screen, at the point when six targets have been displayed, and none have yet been hit. The system has displayed a total of 88 targets of which 15 targets have been hit. The minimum target delay was 900 milliseconds. It can be deduced there are 16 targets in a 4 x 4 matrix.

The Solution

The solution presumes that each target is managed by its own process and that it is these processes that are held in a **Bucket** until it is the turn of that **Bucket** to be flushed. When a target is enabled, it displays itself until either it is 'hit' by a mouse-click, in which case it turns white, or the time for which it appears elapses and it is coloured grey. It is obvious that each of these target processes will finish at a different time and because the number of targets is not predetermined, a barrier is used to establish when all the enabled target processes have finished.

After this, the target process determines into which bucket it is going fall and thereby remains inactive until that bucket is flushed. The other processes used in the solution are shown in `Figure 14-2`.

Want to read more of this chapter? Download this chapter's PDF here.