Concurrent Programming Report

Fergus Strangways-Dixon STRFER001 25/08/2016

The code was full of concurrency errors. The most notable being the unsynchronised first block. This resulted in a mutual exclusion error, as only one ‘person’ is able to stand on a block at a time. I fixed this by creating a Reentrant lock on the first block, which threads would block on in order to enter the area. The lock unlocks after a successful move away from the first block has been made. I preferred an explicit lock object over a synchronised block of code because I could control exactly when (in the code) the lock released. The counter also had to be synchronised to avoid check-write interleaving errors. This was done with a simple synchronise(counter) statement.

The pause button works by interrupting all the active threads, which leads to them finishing their current move, then waiting. When resumed, the GUI class calls .notify() on each thread, which then proceeds moving. The interrupt exceptions are handled inside while loops, so they throw the exception and continue until they hit the .wait() calling if statement in the beginning of the loop. I had to fiddle with the sleep timers of the threads otherwise pausing/resuming resulted in them all moving immediately, which jarred the visual aspect of the project.

The main way I dealt with the concurrency errors – mainly identified by watching the animation run and then analysing the code – was by using synchronised blocks of code. I did however run into deadlocks with pausing and resuming code, so had to refine my blocks down to absolute essential operations to avoid deadlocks.