Introduction

Concurrency in programming is the management of work to a resource to ensure proper distribution and effective use of data. This stops errors based on data being accessed in the wrong order. The assignment shows the intricacies of this by fixing and completing a program. The party/bar shown in the code has multiple threads for each person(represented by a dot and share information) and counters for information displayed.

The requirements were to add a pause button, synchronize, the start of the threads, ensure one person enters at a time from the right place and negate all deadlocks.

Method

The entering one at a time problem was fixed by synchronizing the wait method and checking if a person is on the block. If it is occupied then then the person wait, if it isn't then they move to the block

Synchronizing the start was done using a barrier object which allows the threads to continue once all the threads have reached the same point.

A pause button was put in. A global variable with the pause status was used. The variable is checked to see if the simulation is paused.

To avoid a race condition with people trying to get to a square an AtomicBoolean was used and the getBlock method was synchronized.

Code Edits

I added a setPause() mutator method and a getPause() assessor method to help track and change whether or not the program is paused. A CyclicBarrier was added to the PartyApp and the PersonMover constructor was modified to allow it to be passed through it.

An AtomicBoolean was added to GridBlock in place of th eexisting occupied boolean. All uses of the occupied variable were changed to accommodate this. If the block is occupied the thread using it call notify and getBlock. If not it gets the block. The get and release block methods are synchronized to avoid a race condition.

A method was added to RoomGrid which notify all threads if the program is unpaused.

A CyclicBarrier was added to the PersonMover (along with the perviously mentioned addition to the constructor). The barrier was used to make sure the threads are created and run.

Debug

In the initial code there were people not moving because their threads were in deadlock. Multiple race conditions were seen because of the shared data. Trace variables were used to see the thread usage.