**Rumbidzai Mashumba**

**MSHRUM006**

**CSC2002S Practical : PCP 2**

1. **Introduction:**

MonteCarloMinimization is an extensive search for a global minimum of a function. We adopted a paralyzation approach to improve the speed of the search. This method was chosen for its ability to be separated into smaller tasks that can be executed in parallel.

1. Start and Pause button

To make sure the simulation on starts at command I implemented a count down latch. This allowed multiple threads to wait until the simulation start command was actitvated, ensuring synchronized and controlled initiation of the simulation.

To enable pause functionality I integrated an AtomicBoolean with the Count Down Latch allowing seamless pausing amd resuming of simulation. The Count Down Latch stopped new threads from starting while the toggled flag paused thread execution.

1. Exit and Entrance can be accessed one patron at a time

To ensure that only one patron could use the entrance and exit I employed a synchrinizing mechanism. By synchronizing patrons where forced to wait until the door was available before they could use it, gauranting that only one patron could enter and exit at a give time.

1. Max people

To ensure that the number of people in the club did not exceed the maximum I made use of a busy-waiting loop. It checked if the maximum number of patrons inside the club had reached the specified limit and if the limit was reached, the loop continues to wait until the number of patrons inside the club decreases and a space becomes available.

1. Realistic distance

To ensure that a realistic distance was maintained, I synchronized access to the grid blocks to ensure that only one patron can occupy a block at a time. This made it possible to have only one patron occupying one block at a time.

1. Synchronized movement

This was achieved by synchronizing the Move method.

Patrons' simultaneous movement is ensured through synchronization mechanisms. If the new block is available for occupancy by the patron (indicated by **newBlock.get(myLocation.getID())**), the patron is allowed to move to the new block. This movement occurs in a synchronized manner to prevent conflicts and maintain consistency.

To ensure liveness, the **release** method is used to release the current block, allowing other patrons to occupy it. This enables multiple patrons to move concurrently, contributing to the dynamic and active atmosphere of the simulation.