4.4 Discuss in the report section of your project a strategy for detecting a deadlock in the simulation

To detect deadlocks, we can create a separate monitoring thread that keeps track of each thread's state (active or blocked) and the resources it currently holds.
In order to differentiate between a blocked thread and one that is just checking if the square it wants to move to is free, we can use a timeout mechanism. So that, when a thread attempts to acquire a mutex lock, it uses the try_lock_for() function with a specified timeout duration. If the lock is not acquired within the timeout, the thread communicates this information to the monitoring thread, indicating that it is blocked.

Our monitoring thread can periodically check the states of all threads and the resources they hold. If there is a cycle in the resource allocation graph like a set of threads waiting for resources held by each other, then there is a deadlock

Once a deadlock is detected, we can choose a victim thread and terminate it early to release the resources it holds. We can also attempt to resolve the deadlock by rolling back some of the transactions or reordering the resource requests by changing the path taken by the robot to accomplish its task. To make sure that our deadlock detection is accurate and not just identifying temporary resource contention, we can periodically check for deadlocks and only do something if the same deadlock is detected in multiple consecutive checks.