My Locking and Game Play Function Algorithm as a Pseudo Code

While (True)

1. First check current turn, if the turn of the game is not match with the thread's turn, do not let it obtain to lock. In other words, thread will spin until its turn. However, check the game status. If the game is finished, break the loop and leave the function to terminate the loser thread properly. Therefore, either winner or loser thread will not be alive after game ending.

2. If the turn matches with the thread's turn, acquire the lock.

3. Check if the game is over at previous turn. If it ended, unlock the lock, and terminate.

4. Take random coordinates until finding an empty location. Then, mark it with thread's mark and change the turn of the game.

5. Check the table, if it is finished, announce the result, and unlock the lock and terminate.

6. If game continues, unlock the lock, and go back to first point and repeat this process.

As an abstract view of the above Pseudo Code

```
While (True)
    While (game of turn is not equal to thread's turn) spin it but also check if the game ended or not.
    Acquire Lock
    Do Operations
    Unlock the Lock
```

Conclusion: Since there is a **turn based fair** locking, I think my locking algorithm is like Fetch-and-add Ticket Lock without the Tickets.

Correctness of my locking algorithm: Since I locked in a proper location, all critical operations on the table are being done in an isolated way. There is no race condition of threads. Therefore, this locking algorithm is correct.

Since I always update game status, the game turn and unlock the lock, there is **no deadlock** in the algorithm.

Fairness of my locking algorithm: Since it is **turn** based, one thread cannot acquire the lock consecutively in above algorithm. Therefore, this locking algorithm is **fair.**

Performance: Since threads spin until their turn, some CPU is wasted. However, we have only two threads spinning consecutively, so it is not a big problem.

Main Thread

Firstly, program takes size (N) argument to create N x N game table. Then, main thread creates Thread X and wait for turn change. Therefore, Player X always start as a first player in the game. Then, it creates Thread Y. Threads have arguments as struct to hold their marks (‘X’, ‘O’) and their turn number (0, 1). These arguments will passed to their function to have proper locking mechanism. Main Thread also “joins” both threads. Therefore, main thread always “waits” for the termination of both threads. Then, it terminates itself.

Game Status Checking Function/Algorithm

1. Check if there is Row winning
2. Check if there is Column winning
3. Check if there is Diagonal-Anti Diagonal winning
4. Check if the table is full or not