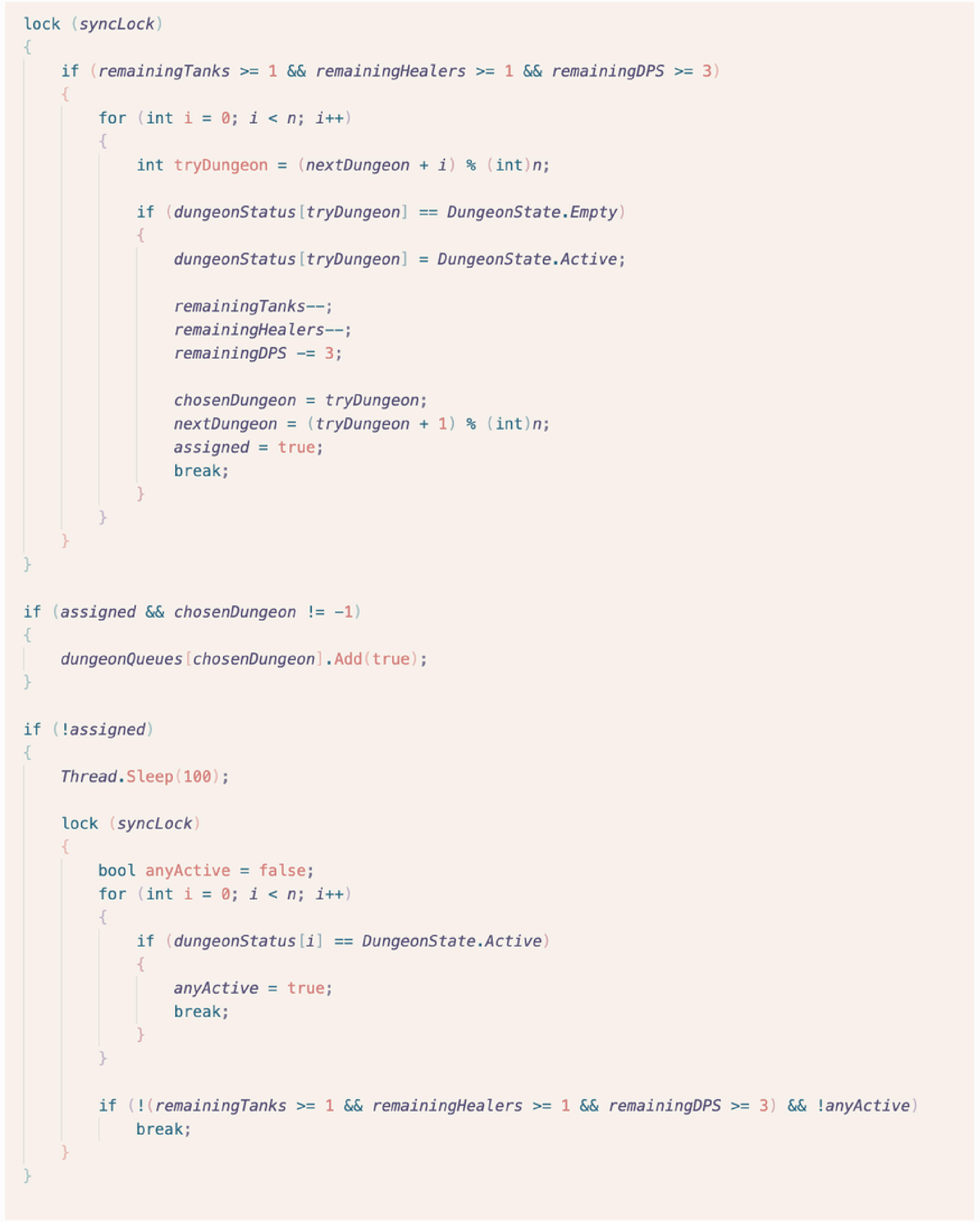
STDISCM

# Problem Set 2

Github Repository

# Synchronization Mechanism

- The program uses one shared lock to prevent multiple threads from trying to update shared variables (player counts and dungeon status) at the same time.
- Each dungeon runs in its own thread and waits for a signal when a party is ready. A queue is used to prevent threads from interfering with each other when the dispatcher is assigning parties to dungeon threads.

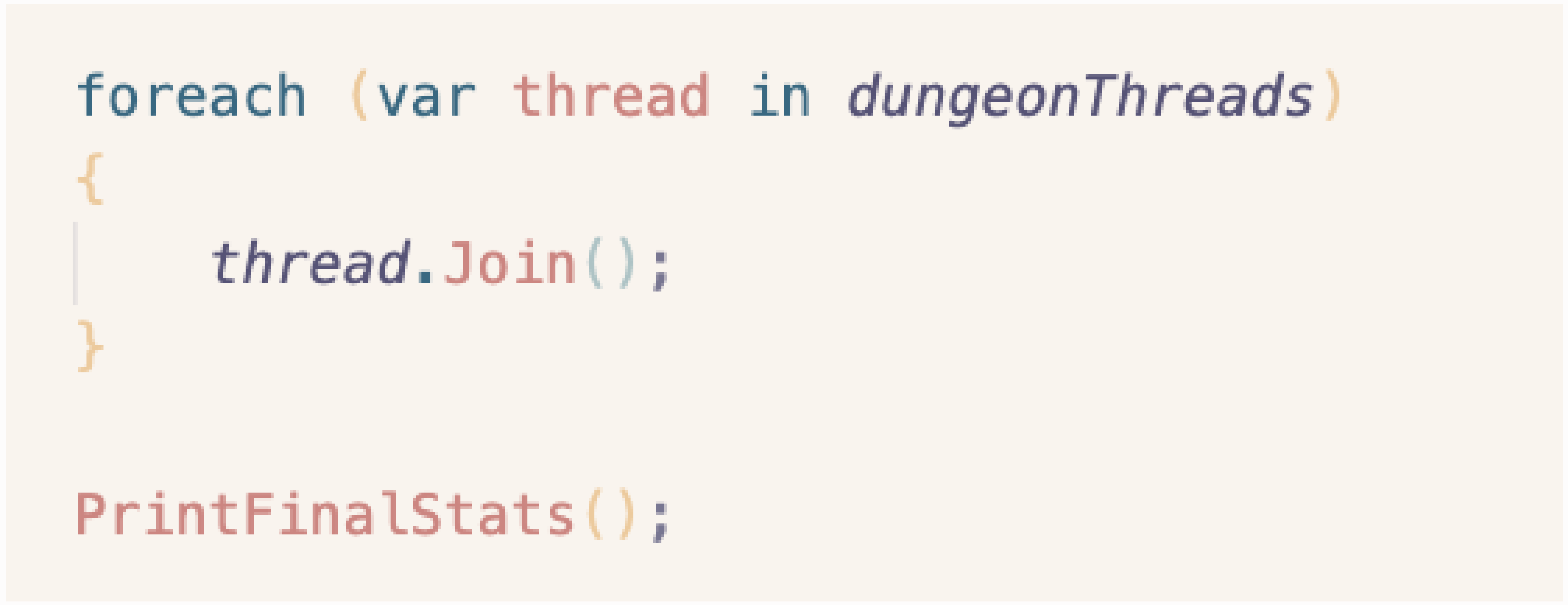


```
lock (syncLock)
{
    if (remainingTanks >= 1 && remainingHealers >= 1 && remainingDPS >= 3)
    {
        for (int i = 0; i < n; i++)
        {
            int tryDungeon = (nextDungeon + i) % (int)n;

            if (dungeonStatus[tryDungeon] == DungeonState.Empty)
            {
                dungeonStatus[tryDungeon] = DungeonState.Active;

                remainingTanks--;
                remainingHealers--;
                remainingDPS -= 3;

                chosenDungeon = tryDungeon;
                nextDungeon = (tryDungeon + 1) % (int)n;
                assigned = true;
                break;
            }
        }
    }
}

if (assigned && chosenDungeon != -1)
{
    dungeonQueues[chosenDungeon].Add(true);
}

if (!assigned)
{
    Thread.Sleep(100);

    lock (syncLock)
    {
        bool anyActive = false;
        for (int i = 0; i < n; i++)
        {
            if (dungeonStatus[i] == DungeonState.Active)
            {
                anyActive = true;
                break;
            }
        }

        if (!(remainingTanks >= 1 && remainingHealers >= 1 && remainingDPS >= 3) && !anyActive)
            break;
    }
}
```

With the lock, only one thread at a time updates shared player counts

# Synchronization Mechanism

- Once all players are used, or no more parties can be formed, the dispatcher signals that no more work is coming. The final results are printed after all dungeons are cleared by their last parties.



```
foreach (var thread in dungeonThreads)
{
    thread.Join();
}

PrintFinalStats();
```

Program waits for all dungeon threads to finish before printing the final results.

## Deadlock

**Deadlock is not as only one lock (syncLock) is used, and blocking operations are performed outside of it.**

- Circular waiting is avoided by never nesting locks or waiting for multiple locks.
- Blocking operations like BlockingCollection.Add() are performed after releasing the lock.
- Threads do not wait on multiple resources at the same time. A dungeon thread can wait on only one of these: a BlockingCollection<bool>, or access to the lock. No thread ever holds one resource and waits for another/
- All dungeon threads run independently — no thread ever blocks another indefinitely.

## Starvation

**Each dungeon is sure to be visited by the round-robin approach.**

- The dispatcher checks each dungeon in order and assigns parties only to those that are empty.
- No dungeon is permanently skipped or given priority.
- As long as there are enough resources to form a party, the traversal continues across cycles, so each dungeon eventually receives an assignment, preventing starvation.