Possible Deadlock

- Mutexes were used to protect shared resources in the code
- Two mutexes were used:
  - partyMutex
  - statusMutex
- There shouldn’t be any deadlocks as they are independent from each other.

```
instances.push_back(thread([&, i]() {
    while (true) {
        {
            lock_guard<mutex> lock(partyMutex);
            if (partyCount == 0) {
                break;
            }
            partyCount--;
        }
        {
            lock_guard<mutex> lock(statusMutex);
            dungeonStatus[dungeons[i].getId()] = 1;
            logOutput("Party entered Dungeon " + to_string(dungeons[i].getId()) + "\n\n");
            logDungeonStatus(dungeonStatus);
        }
        int time = dis(gen);
        dungeons[i].RunDungeon(time);
        {
            lock_guard<mutex> lock(statusMutex);
            dungeonStatus[dungeons[i].getId()] = 0;
            logOutput("Party finished Dungeon " + to_string(dungeons[i].getId()) + "\n\n");
            logDungeonStatus(dungeonStatus);
        }
    }
}));
```

Possible Starvation

- In allocating parties to dungeons, my code does not implement a queueing system and just allows the first available thread to take in a party.
- However, this can lead to starvation if the parties theoretically finish instantly, as the first thread may take most of the parties.

What can be done?

- A queueing system like round robin can be implemented to ensure that tasks are equally distributed among dungeons.