P2 - Looking for Group Synchronization

MOJICA, Harold C.
STDISCM S11

March 28, 2025

Possible Deadlock

Waiting for Players but No Availability

- If there are not enough players to form a full party, the *runDungeon* function might end up waiting indefinitely.
- However, this issue is avoided by checking player availability before assigning them to instances.

```
// Wait for a party or shutdown signal
cv.wait(lock, [&]() {
    return shutdown ||
        (instances[instanceId].tank == 1 &&
            instances[instanceId].healer == 1 &&
            instances[instanceId].dps == 3);
});
```

```
// Wait for party members, THEN Enter dungeon, THEN Finish
void runDungeon(int instanceId) {
    while (true) {
        std::unique_lock<std::mutex> lock(mtx);
```

```
// Distribute players to instances
int z = 0;
while (tankPlayers >= 1 && healerPlayers >= 1 && dpsPlayers >= 3) {
```

Possible Starvation

Uneven Distribution of Players

- If certain instances always get assigned new parties first, others might remain idle.

- However, the *“z = (z + 1) % maxInstances”* ensures round-robin assignment, preventing one instance from monopolizing players.

```
// Distribute players to instances
int z = 0;
while (tankPlayers >= 1 && healerPlayers >= 1 && dpsPlayers >= 3) {

    // Find an inactive instance
    if (!instances[z].active) {
        instances[z].tank = 1;
        instances[z].healer = 1;
        instances[z].dps = 3;
        instances[z].active = true;
        tankPlayers--;
        healerPlayers--;
        dpsPlayers -= 3;

        // Notify that a party is available
        cv.notify_all();
    }

    // Loop through instances
    z = (z + 1) % maxInstances;
}
```

Synchronization Mechanisms

- **Mutexes** and **condition variables** were used for proper synchronization.
- **Round-robin scheduling** was used to prevent starvation.
- The **cv.wait()** and **notify_all()** mechanisms were used prevent deadlocks.

```
class DungeonManager {
    private:
        // Synchronization
        bool shutdown = false;
        std::mutex mtx;
        std::condition_variable cv;
```

```
// Shutdown
{
    std::lock_guard<std::mutex> lock(mtx);
    shutdown = true;
    cv.notify_all();
}
```