STDISCM PS2

By Bentley Lu

POSSIBLE DEADLOCK

- Issue: Deadlock can occur if all dungeon instance slots are full, causing multiple threads to wait indefinitely for an open slot while holding locks.
- Impact: This prevents queue progress, leaving threads stuck and blocking other necessary operations.
- Cause: Lack of proper synchronization when managing instance availability.

AVOIDING DEADLOCKS

```
if (availableSlot == -1) {
    instanceAvailable.wait(lock, [this]() {
        return std::any_of(instanceStatus.begin(), instanceStatus.end(),
            [](const InstanceStatus& inst) { return !inst.active; });
        });
    continue;
}
```

- Releases the lock while waiting to avoid blocking other threads.
- Ensures the thread resumes only when an instance becomes available.
- Prevents circular waiting, avoiding deadlock.

POSSIBLE STARVATION

- Issue: Uneven workload distribution among instances leads to bottlenecks and inefficient dungeon queue processing.
- Impact: Some instances remain underutilized while others are overloaded, leading to starvation for certain instances.
- Cause: Instances are assigned without considering varying completion times, preventing optimal distribution of parties across available slots.

AVOIDING STARVATION

```
int availableSlot = -1;
for (uint32_t i = 0; i < maxInstances; i++) {
    uint32_t index = (nextInstanceIndex + i) % maxInstances;
    if (!instanceStatus[index].active) {
        availableSlot = static_cast<int>(index);
        nextInstanceIndex = (index + 1) % maxInstances;
        break;
    }
}
```

- The nextInstanceIndex keeps track of the last assigned instance, ensuring the next available one is chosen.
- No single instance gets overloaded while others remain idle.
- Each instance gets a fair chance to be assigned a new party, preventing bottlenecks.

GITHUB REPO:
HTTPS://GITHUB.COM/BOONTLY/STDISCM_P2

THANK YOU