Задачи комплексные, сочетающие темы разных модулей. 10 задач по возрастающей сложности, каждая рассчитана на 2-3 часа работы.

## Комплексные задачи по многопоточному программированию в мире RPG

### Задача 1: "Магический артефакт с ограниченной мощностью"

**Сложность:** 1/5
**Модули:** 3 (async/await), 4 (SemaphoreSlim)

**Задача:** Создайте класс MagicOrb с ограниченным запасом маны (например, 100 единиц). Несколько потоков-магов (Wizard) асинхронно пытаются кастовать заклинания, которые тратят ману. Используйте SemaphoreSlim для ограничения одновременного доступа к орбу.

**Требования:**

- Маги могут кастовать заклинания разной мощности (5-20 маны)
- Если маны недостаточно - ждать восстановления
- Орб восстанавливает ману со временем
- Использовать асинхронные методы с await

```
// Пример интерфейса
public class MagicOrb
{
   public async Task<bool> TryCastSpell(string wizard, int manaCost);
}
```

### Задача 2: "Система квестов с унаследованным кодом"

**Сложность:** 2/5
**Модули:** 1 (Refactoring), 3 (Task), 4 (lock)

**Задача:** Дан унаследованный класс LegacyQuestManager с жесткими зависимостями. Нужно:

1. Проанализировать и сделать код тестируемым
2. Добавить систему уведомлений о завершении квестов
3. Реализовать потокобезопасное обновление прогресса

**Исходный код имеет проблемы:**

- Статические зависимости

- Прямое создание объектов в методах
- Отсутствие интерфейсов
- **Пример legacy кода можно взять в дополнении или придумать самостоятельно**

### Задача 3: "Менеджер заклинаний с кэшированием"

**Сложность:** 2/5
**Модули:** 3 (async/await), 4 (ReaderWriterLockSlim)

**Задача:** Создайте SpellManager, который загружает описания заклинаний из файлов. Используйте ReaderWriterLockSlim для безопасного доступа к кэшу.

**Требования:**

- Множественное чтение описаний заклинаний
- Эксклюзивная блокировка при добавлении новых заклинаний
- Асинхронная загрузка из файлов
- Кэширование результатов

```
public class SpellManager
{
   public async Task<string> GetSpellDescriptionAsync(string spellName);
   public void AddSpell(string spellName, string description);
}
```

### Задача 4: "Гильдия магов с межпроцессным взаимодействием"

**Сложность:** 3/5
**Модули:** 2 (Process), 4 (Mutex)

**Задача:** Создайте систему гильдии, где несколько экземпляров приложения (процессы) могут регистрировать магов. Используйте Mutex для межпроцессной синхронизации доступа к общему файлу с данными гильдии.

**Требования:**

- Запуск нескольких экземпляров приложения
- Синхронизация через именованный Mutex
- Чтение/запись в общий файл JSON
- Обработка конфликтов при регистрации

**Задача 5: "Асинхронный сбор ресурсов в подземелье"**

**Сложность:** 3/5
**Модули:** 3 (Task, async/await), 4 (SemaphoreSlim, CancellationToken)

**Задача:** Создайте систему сбора ресурсов в подземелье. Несколько искателей приключений одновременно собирают ресурсы, но количество узлов сбора ограничено.

**Требования:**

- Ограничение одновременного сбора с помощью SemaphoreSlim
- Возможность отмены операции через CancellationToken
- Асинхронное ожидание освобождения узлов
- Сбор статистики по каждому искателю

**Задача 6: "Распределенные вычисления заклинаний"**

**Сложность:** 4/5
**Модули:** 2 (Process), 3 (Task), 4 (синхронизация)

**Задача:** Создайте систему, где главный процесс запускает дочерние процессы для вычисления сложных магических формул. Координируйте работу через межпроцессное взаимодействие.

**Требования:**

- Запуск внешних процессов-вычислителей
- Распределение задач между процессами
- Сбор результатов
- Обработка падений дочерних процессов

**Задача 7: "Рефакторинг боевой системы с тестами"**

**Сложность:** 4/5
**Модули:** 1 (Legacy Code), 3 (многопоточность), 4 (синхронизация)

**Задача:** Дан унаследованный класс CombatSystem с проблемами потокобезопасности. Нужно:

1. Добавить модульные тесты
2. Выделить чистые методы (Sprout Method)
3. Обеспечить потокобезопасность
4. Добавить асинхронные версии методов

**Проблемы в коде:**

- Состояние гонки при расчете урона
- Блокировки UI потока
- Смесь логики и представления
- **Пример legacy кода можно взять в дополнении или придумать самостоятельно**

### Задача 8: "Система аукциона предметов"

**Сложность:** 5/5
**Модули:** 3 (Task), 4 (все примитивы синхронизации)

**Задача:** Создайте многопользовательскую систему аукциона с реальным временем.

**Требования:**

- Множественные читатели просматривают лоты (ReaderWriterLockSlim)
- Эксклюзивные ставки (lock)
- Таймеры завершения торгов (Timer + ManualResetEvent)
- Уведомления участников (async/await)
- Предотвращение взаимных блокировок

```
public class AuctionHouse
{
   public Task<BidResult> PlaceBidAsync(string itemId, string player, decimal amount);
   public IReadOnlyList<AuctionItem> GetActiveAuctions();
}
```

### Задача 9: "Распределенный мир с сервером процессов"

**Сложность:** 5/5
**Модули:** 2 (Process), 3 (async), 4 (Mutex, Semaphore)

**Задача:** Создайте систему, где главный серверный процесс управляет несколькими дочерними процессами-зонами.

**Требования:**

- Главный процесс - координатор
- Дочерние процессы - игровые зоны

- Межпроцессная синхронизация перемещения игроков
- Балансировка нагрузки между зонами
- Асинхронная коммуникация

**Задача 10: "Полная RPG система с Legacy рефакторингом"**

**Сложность:** 5/5
**Модули:** 1, 2, 3, 4 (все модули)

**Задача:** Создайте полноценную RPG систему, начиная с унаследованного кода и постепенно применяя все изученные техники.

**Пример legacy кода можно взять в дополнении или придумать самостоятельно**

**Этапы:**

1. **Анализ** унаследованной системы персонажей и инвентаря
2. **Рефакторинг** с добавлением тестов и интерфейсов
3. **Многопоточность** - асинхронные сохранения, параллельные вычисления
4. **Синхронизация** - безопасные транзакции, торговля между игроками
5. **Процессы** - отдельный процесс для логирования и аналитики

**Критерии оценки для студентов:**

**Для простых задач (1-3 сложность):**

- Корректная работа в однопоточном режиме

- Базовая потокобезопасность
- Отсутствие deadlock

**Для сложных задач (4-5 сложность):**

- Полная обработка ошибок
- Оптимальное использование ресурсов
- Модульные тесты
- Чистая архитектура
- Документация ключевых решений

**Дополнительные рекомендации:**

- Начинайте с анализа требований и проектирования архитектуры
- Пишите тесты параллельно с разработкой
- Используйте CancellationToken для отмены операций
- Документируйте выбор примитивов синхронизации
- Проверяйте производительность под нагрузкой

Каждая задача позволяет выбрать свой уровень сложности - можно реализовать базовую функциональность или добавить расширенные возможности.

Удачи в создании магических многопоточных миров!