Привет. Подскажи пожалуйста, как лучше выполнить это задание на языке python?

Задание: "Космический шахтёр: Гонка за ресурсами" Сюжет Вы управляете космической добывающей станцией на астероиде. Вокруг вас 4 планеты с разными ресурсами. Ваша задача — добывать, торговать и защищаться от угроз, используя разные методы параллелизма. Три уровня управления Уровень 1: Асинхронный трейдер (asyncio) • Ваш корабль летает между планетами • На каждой планете можно купить/продать ресурсы • Цены меняются каждые 3 секунды (асинхронный таймер) • События происходят случайно: метеоритный дождь, солнечные вспышки • Корабль может летать и торговать одновременно Уровень 2: Многопоточные шахты (threading) • У вас есть 3 типа шахт на станции: 1. Энерго-шахта (быстрая, но мало ресурсов) 2. Глубинная шахта (медленная, много ресурсов) 3. Экспериментальная шахта (нестабильная, может сломаться) • Каждая шахта работает в отдельном потоке • Можно ремонтировать и улучшать шахты во время работы Уровень 3: Процессы для тяжёлых задач (multiprocessing) • Навигатор: рассчитывает лучший маршрут между планетами • Аналитик: предсказывает, где цена вырастет (простой алгоритм) • Защитник: если нападают пираты — запускает симуляцию боя Игровые объекты Ресурсы (всего 4 вида): Энергия (Energy) - для работы шахт Металл (Metal) - для улучшений Кристаллы (Crystals) - дорогой товар Запчасти (Parts) - для ремонта Планеты (4 штуки): • Лёд-9: много Энергии, дешёвый Металл • Огневик: дорогие Кристаллы, нет Запчастей • Механикус: дешёвые Запчасти, средние цены • Нейтральная: средние цены на всё Ваша станция: • Склад: 1000 единиц каждого ресурса • Деньги: 5000 кредитов • Здоровье: 100 HP (пираты атакуют) • Шахты: 3 штуки (потоки) Как играть Ход 1: Проверяем шахты (потоки) # Каждая шахта 1 раз в секунду даёт ресурсы # В главном меню видим: [Шахта 1] Энергия: +5/сек (работает) [Шахта 2] Металл: +2/сек (требует ремонта!) [Шахта 3] Кристаллы: +1/сек (работает) Ход 2: Смотрим рынок (асинхронно) # Цены обновляются каждые 3 секунды Планета Лёд-9: Энергия: 10 кредитов (падает) Металл: 25 кредитов (растёт!) Планета Огневик: Кристаллы: 100 кредитов (стабильно) Ход 3: Отправляем корабль (асинхронно) # Выбираем планету и что везём Корабль летит на Лёд-9 (5 секунд) Везу: 50 Энергии (купил здесь раньше дешевле) Хочу купить: Металл (здесь дёшево) Ход 4: Запускаем анализ (процесс) # Пока корабль летит (5 секунд) # Запускаем процесс "Аналитик": "Через 10 сек цена на Металл вырастет на 30%!" События (все асинхронные) 1. Пиратская атака (раз в 30-60 сек) [ТРЕВОГА!] Пираты атакуют станцию! Выберите действие: 1. Сражаться (запустит процесс-симулятор боя) 2. Заплатить выкуп (500 кредитов) 3. Спрятаться (станция не работает 10 сек) 2. Космическая погода [СОБЫТИЕ] Солнечная вспышка! - Шахты работают на 50% скорости 15 сек - Цены на Энергию +20% 3. Поломка оборудования [АВАРИЯ] Шахта #2 сломалась! Нужно: 10 Запчастей и 5 секунд на ремонт Ремонтировать? (Да/Нет) Цели игры • Заработать 10,000 кредитов Что нужно сделать Часть 1: Шахты (Threading) # Класс

Mine(Thread): # - Добывает ресурс каждую секунду # - Может сломаться (случайно) # - Имеет уровень (1-3) # - Можно улучшать (блокирует шахту на время) Часть 2: Корабль и рынок (Asyncio) # Асинхронные функции: # - fly_to(planet) # Летит 3-7 секунд # - update_prices() # Меняет цены каждые 3 сек # - handle_events() # Случайные события # - trade() # Покупка/продажа Часть 3: Тяжёлые задачи (Multiprocessing) # Функции в отдельных процессах: # - calculate_profit() # Где выгоднее торговать? # - battle_simulation() # Симуляция боя с пиратами # - find_best_route() # Оптимальный маршрут Часть 4: Игровой цикл (Всё вместе) # Главная функция: while money < target and health > 0: # 1. Проверяем шахты (потоки) # 2. Проверяем корабль (асинхронно) # 3. Обрабатываем события (асинхронно) # 4. Показываем меню игроку # 5. Если пираты - запускаем процесс боя Конкретика: Задание A: Создайте шахты (Threading) 1. Сделайте 3 класса шахт, наследующих Thread 2. Каждая должна добывать свой ресурс 3. Реализуйте поломки (раз в 30-90 сек) 4. Сделайте улучшение шахт (блокировка на 5 сек) Задание B: Создайте рынок (Asyncio) 1. Цены 4 ресурсов на 4 планетах (матрица 4x4) 2. Асинхронная функция меняет цены каждые 3 сек 3. Корабль летит асинхронно с прогресс-баром 4. Случайные события каждые 10-20 сек Задание C: Система боя (Multiprocessing) 1. При атаке пиратов запускается процесс 2. Процесс считает исход боя (простая математика) 3. Возвращает результат: победа/поражение/ничья 4. Пока процесс работает — игра ждёт Пример игровой сессии [СТАРТ] Деньги: 5000, Шахты: 3 базовые > Улучшаю шахту #1 (5 сек, 1000 кредитов) ... Жду 5 секунд (асинхронный sleep) [ГОТОВО] Шахта #1 теперь даёт +8 вместо +5! > Отправляю корабль на Огневик [ПОЛЁТ] 5... 4... 3... (асинхронный счётчик) [СОБЫТИЕ] Метеоритный дождь! Цены -10% [ПРИЛЕТЕЛИ] На Огневике, продаю Энергию > Пираты атакуют! [БОЙ] Запускаю симуляцию... (процесс) [РЕЗУЛЬТАТ] Победа! Потерял 15 HP, но защитил груз > Анализирую рынок [АНАЛИТИК] Через 20 сек Металл подорожает на 50% > Скупаю Металл, жду, продаю дорого [ПРИБЫЛЬ] +3000 кредитов!

Привет! Лучший подход к заданию — модульная структура с классами для шахт (threading), asyncio-циклом для рынка/корабля/событий и multiprocessing для тяжёлых задач, плюс общий игровой цикл с ANSI-обновлением терминала без prompt_toolkit (учитывая твои прошлые проблемы с потоками и вводом). Это обеспечивает параллелизм, стабильность и кросс-платформенность.[1] [2]

Шахты (Threading)

Создай базовый класс `Mine`, наследующий `threading.Thread`. Каждая шахта (энергия, металл, кристаллы) работает в daemon-потоке, добывает ресурсы каждую секунду, имеет шанс поломки (random 1/60), блокировку на ремонт/улучшение (Lock).

```
import threading
import time
import random
from collections import defaultdict

class Mine(threading.Thread):
    def __init__(self, name, resource, base_rate=5, daemon=True):
        super().__init__(daemon=daemon)
        self.name = name
        self.resource = resource
```

```
        self.rate = base_rate
        self.level = 1
        self.running = True
        self.broken = False
        self.lock = threading.Lock()
        self.storage = defaultdict(int)  # Глобальный склад через shared dict

    def run(self):
        while self.running:
            with self.lock:
                if self.broken:
                    time.sleep(1)
                    continue
                self.storage[self.resource] += self.rate
                if random.random() < 0.01:  # Поломка ~1/100 сек
                    self.broken = True
            time.sleep(1)

    def upgrade(self, cost=1000):
        with self.lock:
            if not self.broken and self.storage['money'] >= cost:
                self.storage['money'] -= cost
                self.rate += 3
                self.level += 1
                time.sleep(5)  # Блокировка

    def repair(self, parts=10):
        with self.lock:
            if self.broken and self.storage['Parts'] >= parts:
                self.storage['Parts'] -= parts
                self.broken = False
```

Запуск: `mines = [Mine('Энерго', 'Energy'), Mine('Глубинная', 'Metal', 2), Mine('Эксперимент', 'Crystals', 1)]; [m.start() for m in mines]`.[1] [3]

Рынок и корабль (Asyncio)

Матрица цен (4x4), обновление каждые 3 сек в фоне. Корабль — async задача с sleep для полёта + прогресс. События (пираты, вспышки) через asyncio.create_task.

```
import asyncio
import random

PLANETS = ['Лёд-9', 'Огневик', 'Механикус', 'Нейтральная']
RESOURCES = ['Energy', 'Metal', 'Crystals', 'Parts']
prices = {p: {r: random.randint(10, 100) for r in RESOURCES} for p in PLANETS}

async def update_prices():
    while True:
        for p in prices:
            for r in prices[p]:
                prices[p][r] += random.randint(-5, 5)
        await asyncio.sleep(3)
```

```
async def fly_to(planet, cargo):
    print(f"Полёт на {planet}...")
    for i in range(5, 0, -1):
        print(f"\r{i}...", end='')
        await asyncio.sleep(1)
    print(f"\nПрилетели! Торговля: {cargo}")

async def handle_events():
    while True:
        if random.random() < 0.05:  # Пират ~каждые 20 сек
            print("[ТРЕВОГА!] Пираты!")
        await asyncio.sleep(random.randint(10, 20))
```

В цикле: `asyncio.create_task(update_prices())`; `await fly_to('Лёд-9', '50 Energy')`.[4] [5]

Бой и анализ (Multiprocessing)

Для пиратов/анализа — отдельный процесс с Queue для результатов. Симуляция: простая математика (HP vs пираты).

```
import multiprocessing as mp

def battle_simulation(health, pirates=10):
    time.sleep(3)  # Тяжёлая симуляция
    damage = random.randint(5, 20)
    win = health > damage * 2
    return f"{'Победа' if win else 'Поражение'}! Урон: {damage}"

def analyst(prices):
    time.sleep(2)
    best = max((prices[p]['Metal'] for p in PLANETS))
    return f"Металл вырастет на 30% на планете с {best} кр!"

# Запуск
if __name__ == "__main__":
    with mp.Pool(1) as pool:
        result = pool.apply(battle_simulation, (100,))
        print(result)
```

В asyncio: `loop.run_in_executor(None, battle_simulation, station_health)`.[6] [7]

Игровой цикл

Объедини в while money < 10000: статус шахт (`print(mine.storage)`), await asyncio.sleep(0.1), меню ввода (input() с ANSI \033[H для обновления). Mines daemon=True — не блокируют выход. Тестируй поэтапно: сначала шахты, потом +asyncio.run(main_loop()).

```
storage = defaultdict(int, {'money': 5000, 'health': 100})
# ... запуск mines, asyncio loop
```

Этот дизайн масштабируем, без внешних lib (кроме std), учитывает твои проблемы с потоками/ANSI. Запусти части — увидишь параллелизм! [8] [9]

⁂

1. work.programming_issue
2. https://www.stratascratch.com/blog/python-threading-like-a-pro/
3. https://realpython.com/intro-to-python-threading/
4. https://towardsdatascience.com/level-up-your-coding-skills-with-python-threading-8f1bd06b9476/
5. https://www.python4data.science/en/latest/performance/asyncio-example.html
6. https://betterstack.com/community/guides/scaling-python/python-async-programming/
7. https://www.linkedin.com/posts/pragya-sah-950712222_im-pleased-to-share-a-project-i-recently-activity-7329783689205166080--IYH
8. https://www.dataleadsfuture.com/combining-multiprocessing-and-asyncio-in-python-for-performance-boosts/
9. https://www.perplexity.ai/search/d4506c16-a729-44d4-9c02-78c1dbab618e
10. https://dev.to/imsushant12/multithreading-in-python-lifecycle-locks-and-thread-pools-3pg3
11. https://github.com/AsyncAlgoTrading/aat
12. https://www.youtube.com/watch?v=sG64yeX3Nh4
13. https://cduser.com/when-ai-goes-to-war-building-a-strategic-combat-simulator-in-python/
14. https://www.youtube.com/watch?v=VoDTVOJR3Og
15. https://stackoverflow.com/questions/78649612/asteroid-mining-dynamic-programming-problem
16. https://stackoverflow.com/questions/28492103/how-to-combine-python-asyncio-with-threads
17. https://www.pyquantnews.com/the-pyquant-newsletter/how-to-simulate-stock-prices-with-python
18. https://www.youtube.com/watch?v=K-cprFKHJ8I
19. https://www.reddit.com/r/learnpython/comments/188m06s/combination_of_multiprocessing_threading_and/
20. https://pypi.org/project/SpacePyTraders/
21. https://www.youtube.com/watch?v=kJBzihJhkqs
22. https://stackoverflow.com/questions/27435284/multiprocessing-vs-multithreading-vs-asyncio