تمرین اول

نام درس: سیستم های چندعاملی

استاد درس: دکتر ناصر مزینی

نام: محمد حقیقت

شماره دانشجویی: 403722042

گرایش: هوش مصنوعی

دانشکده: مهندسی کامپیوتر

نیم سال دوم 1403-1404

```
# --- AI Studio API Placeholder ---
# IMPORTANT: Replace these with your actual AI Studio API credentials and endpoint.
# This is a mock implementation. You will need to adjust the `AIStudioChatBot.chat`
# method to match the exact request format (payload, headers) of your API.
AI_STUDIO_API_URL = "https://generativelanguage.googleapis.com/v1beta/models/gemini-2.0-flash:generateContent"
AI_STUDIO_API_KEY = "AIzaSyAlma1gIlqBd4TfGh5wXhQNhxDLeExMj5U"

random.seed(42)
```

اطلاعات API: متغیرهای AI_STUDIO_API_URL و AI_STUDIO_API_KEY آدرس و کلید دسترسی به سرویس هوش مصنوعی گوگل هستند. این کد به درستی اشاره می‌کند که این مقادیر باید با اطلاعات واقعی جایگزین شوند.

```
# --- Constants ---
GRID_SIZE = 15
NUM_BUILDINGS = 7
NUM_CIVILIANS = 8
NUM_TYPE1_FIRES = 5
NUM_TYPE2_FIRES = 4
NUM_AGENTS_PER_TEAM = 2
MAX_STEPS = 50
SENSOR_RANGE = 2 # This results in a 5x5 perception grid (center + 2 on each side)

# --- Cell Types ---
EMPTY = 0
BUILDING = 1
CIVILIAN = 2
FIRE = 3
STATION = 6
LARGE_FIRE = 7
# New Agent Types for Teams
AGENT_A1 = 4
AGENT_A2 = 5
AGENT_B1 = 8
AGENT_B2 = 9
```

ثابت‌های شبیه‌سازی: متغیرهایی مانند GRID_SIZE, NUM_BUILDINGS, NUM_CIVILIANS و غیره، پارامترهای یک محیط شبیه‌سازی شده را تعریف می‌کنند. به نظر می‌رسد این کد، بخشی از یک پروژه بزرگتر است که در آن "عامل‌ها" (Agents) در یک محیط شهری با آتش‌سوزی، ساختمان‌ها و غیرنظامیان حرکت می‌کنند.

```python
# --- Custom Color Mapping ---
COLORS = {
    EMPTY: "lightgray",
    BUILDING: "yellow",
    CIVILIAN: "blue",
    FIRE: "red",
    STATION: "purple",
    LARGE_FIRE: "orange",
    AGENT_A1: "green",
    AGENT_A2: "darkgreen",
    AGENT_B1: "cyan",
    AGENT_B2: "darkcyan",
    -1: "black"  # Unknown
}
```

انواع سلول‌ها و رنگ‌ها: این بخش، کدهای عددی (مثل EMPTY = 0, FIRE = 3) را به عناصر مختلف محیط و رنگ‌های مشخص برای نمایش گرافیکی آن‌ها اختصاص می‌دهد.

```python
class AIStudioChatBot:
    """
    A class to interact with Google's Generative Language API (AI Studio),
    now with built-in retry logic for handling API rate limits.
    """
    def __init__(self, api_url, api_key):
        if "YOUR_API_KEY" in api_key or not api_key:
            print("WARNING: AI Studio API Key is not set. LLM calls will fail.")
            print("Please update the AI_STUDIO_API_KEY variable.")

        self.api_url_with_key = f"{api_url}?key={api_key}"
        self.headers = {
            "Content-Type": "application/json"
        }
```

کلاس AIStudioChatBot

این کلاس، مسئول اصلی تمام فرآیند ارتباط با API است.

متد __init__ (سازنده کلاس)

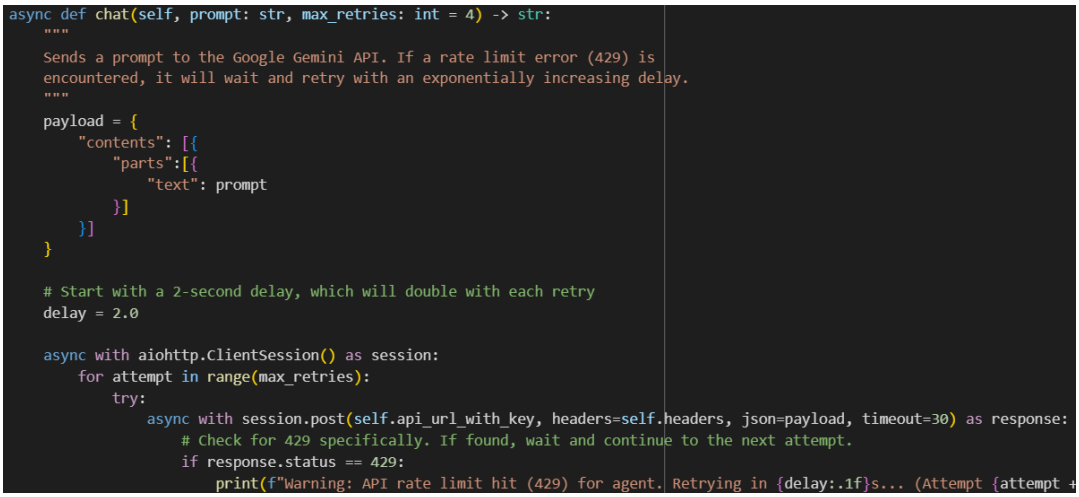
وقتی یک شیء از این کلاس ساخته می‌شود، این متد به طور خودکار اجرا می‌شود.

کار اصلی آن، آماده‌سازی اولیه است:

آدرس نهایی API را با چسباندن کلید به انتهای آن می‌سازد.

یک "هشدار" چاپ می‌کند اگر کلید API تنظیم نشده باشد.

هدرهای (Headers) استاندارد لازم برای یک درخواست وب (از نوع JSON) را تعریف می‌کند.



```python
async def chat(self, prompt: str, max_retries: int = 4) -> str:
    """
    Sends a prompt to the Google Gemini API. If a rate limit error (429) is
    encountered, it will wait and retry with an exponentially increasing delay.
    """
    payload = {
        "contents": [{
            "parts":[{
                "text": prompt
            }]
        }]
    }

    # Start with a 2-second delay, which will double with each retry
    delay = 2.0

    async with aiohttp.ClientSession() as session:
        for attempt in range(max_retries):
            try:
                async with session.post(self.api_url_with_key, headers=self.headers, json=payload, timeout=30) as response:
                    # Check for 429 specifically. If found, wait and continue to the next attempt.
                    if response.status == 429:
                        print(f"Warning: API rate limit hit (429) for agent. Retrying in {delay:.1f}s... (Attempt {attempt +
```

متد async def chat (ارسال پیام و دریافت پاسخ)

این متد، مهم‌ترین و هوشمندترین بخش کد است و وظیفه ارسال درخواست به هوش مصنوعی را بر عهده دارد.

کلمه کلیدی async: این کلمه نشان می‌دهد که تابع به صورت ناهمزمان (Asynchronous) کار می‌کند. به زبان ساده، یعنی وقتی برنامه منتظر پاسخ از سرور گوگل است، "قفل" نمی‌شود و می‌تواند در همان حین به کارهای دیگر خود رسیدگی کند. این ویژگی برای کارایی بالا در برنامه‌های مدرن ضروری است.

آماده‌سازی درخواست (payload): پیامی که شما می‌خواهید از هوش مصنوعی بپرسید (ورودی prompt) را در یک ساختار استاندارد JSON قرار می‌دهد که برای API گوگل قابل فهم باشد.

منطق تلاش مجدد (Retry Logic): این هوشمندانه‌ترین قسمت کد است.

حلقه تلاش: کد درخواست را در یک حلقه تا max_retries بار (مثلاً ۴ بار) ارسال می‌کند.

مدیریت خطای "شلوغی سرور" (Rate Limit): گاهی اگر درخواست‌های زیادی به یک API ارسال کنید، سرور با خطای 429 پاسخ می‌دهد که یعنی: «لطفاً کمی صبر کنید، الان سرم شلوغ است». این کد به جای اینکه تسلیم شود، این خطا را تشخیص می‌دهد، یک پیام هشدار چاپ می‌کند، چند ثانیه صبر می‌کند و دوباره تلاش می‌کند.

انتظار هوشمند (Exponential Backoff): برای اینکه به سرور فشار نیاورد، بعد از هر تلاش ناموفق، زمان انتظار را دو برابر می‌کند (delay *= 2). مثلاً بار اول ۲ ثانیه، بار دوم ۴ ثانیه، بار سوم ۸ ثانیه و... صبر می‌کند.

مدیریت خطاهای دیگر: کد همچنین خطاهای دیگری مانند مشکلات شبکه (TimeoutError) یا خطاهای مربوط به درخواست (مثلاً اشتباه بودن کلید API) را تشخیص می‌دهد و در این موارد، پیام خطای مناسبی چاپ کرده و از تلاش مجدد بی‌دلیل خودداری می‌کند.

دریافت و پردازش پاسخ: اگر درخواست موفقیت‌آمیز بود، پاسخ را که در فرمت JSON است دریافت کرده، متن اصلی جواب هوش مصنوعی را از دل آن ساختار تو در تو استخراج کرده و به عنوان خروجی برمی‌گرداند.

```
class Environment:
    def __init__(self):
        self.grid = np.zeros((GRID_SIZE, GRID_SIZE), dtype=int)
        self.scores = {'A': 0, 'B': 0}
        self.civilians_rescued = {'A': 0, 'B': 0}
        self.type1_fires_extinguished = {'A': 0, 'B': 0}
        self.type2_fires_extinguished = {'A': 0, 'B': 0}
        self.steps = 0
        self.setup_environment()
```

کلاس Environment

این کلاس، مسئولیت مدیریت تمام جنبه‌های محیط شبیه‌سازی را بر عهده دارد. از ایجاد نقشه اولیه گرفته تا گسترش آتش، اعمال اقدامات عامل‌ها (Agents)، و محاسبه امتیازات. به طور خلاصه، این کلاس "خدای" دنیای مجازی شماست که وضعیت آن را در هر لحظه کنترل می‌کند.

متد __init__ (سازنده کلاس)

وقتی یک شیء از این کلاس ساخته می‌شود، این متد به‌طور خودکار اجرا شده و دنیای بازی را برای شروع آماده می‌کند:

self.grid: یک "نقشه" یا "شبکه" (grid) به ابعاد 15x15 ایجاد می‌کند که در ابتدا تمام خانه‌های آن خالی است (مقدار 0).

self.scores و سایر دیکشنری‌ها: متغیرهایی برای نگهداری و پیگیری امتیازات دو تیم (A و B) و آمار عملکرد آن‌ها (مانند تعداد آتش‌های خاموش‌شده و غیرنظامیان نجات‌یافته) ایجاد می‌کند.

self.steps: یک شمارنده برای ردیابی تعداد مراحل یا "نوبت‌های" بازی است که از صفر شروع می‌شود.

self.setup_environment(): بلافاصله پس از ایجاد متغیرهای اولیه، این متد را فراخوانی می‌کند تا نقشه خالی را با عناصر بازی پر کند.

```python
def setup_environment(self):
    self.grid[0, 0] = STATION
    self.grid[14, 14] = STATION
    for _ in range(NUM_BUILDINGS): self.place_random_entity(BUILDING)
    for _ in range(NUM_CIVILIANS): self.place_random_entity(CIVILIAN)
    for _ in range(NUM_TYPE1_FIRES): self.place_random_entity(FIRE)
    for _ in range(NUM_TYPE2_FIRES): self.place_random_entity(LARGE_FIRE)
```

متد setup_environment (آماده‌سازی محیط)

این متد، دنیای بازی را با عناصر اولیه پر می‌کند:

دو "پایگاه" یا "ایستگاه" (STATION) در گوشه‌های بالا-چپ و پایین-راست نقشه قرار می‌دهد.

سپس به تعداد مشخصی، "ساختمان" (BUILDING)، "غیرنظامی" (CIVILIAN) و دو نوع "آتش" (FIRE و LARGE_FIRE) را در خانه‌های تصادفی و خالی نقشه قرار می‌دهد.

```python
def place_random_entity(self, entity):
    while True:
        x, y = random.randint(0, GRID_SIZE - 1), random.randint(0, GRID_SIZE - 1)
        if self.grid[x, y] == EMPTY:
            self.grid[x, y] = entity
            break
```

متد place_random_entity (قرار دادن یک عنصر به صورت تصادفی)

این یک متد کمکی است که کارش بسیار ساده و مهم است:

یک نوع عنصر (مثلاً ساختمان) را به عنوان ورودی می‌گیرد.

آنقدر مختصات تصادفی روی نقشه تولید می‌کند تا یک خانه خالی پیدا کند.

سپس عنصر مورد نظر را در آن خانه خالی قرار می‌دهد. این کار تضمین می‌کند که هیچ دو عنصری روی هم قرار نگیرند.

```python
def spread_fire(self):
    fire_positions = list(zip(*np.where((self.grid == FIRE) | (self.grid == LARGE_FIRE))))
    if not fire_positions:
        return

    new_fires = []
    for x, y in fire_positions:
        directions = [(0, 1), (0, -1), (1, 0), (-1, 0)]
        random.shuffle(directions)
        for dx, dy in directions:
            new_x, new_y = x + dx, y + dy
            if 0 <= new_x < GRID_SIZE and 0 <= new_y < GRID_SIZE:
                cell_content = self.grid[new_x, new_y]
                if cell_content not in [FIRE, LARGE_FIRE, BUILDING, STATION, AGENT_A1, AGENT_A2, AGENT_B1, AGENT_B2]:
                    if cell_content == CIVILIAN:
                        self.scores['A'] -= 100
                        self.scores['B'] -= 100
                        print(f"CIVILIAN DIED IN FIRE! Both teams lose 100 points. Scores: A={self.scores['A']}, B={self.scores['B']}")
```

متد spread_fire (گسترش آتش)

این متد یکی از قوانین پویای دنیای شما را پیاده‌سازی می‌کند:

ابتدا تمام خانه‌هایی که در حال حاضر آتش‌سوزی (از هر دو نوع) دارند را پیدا می‌کند.

برای هر خانه آتش‌گرفته، به صورت تصادفی تلاش می‌کند تا آتش را به یکی از خانه‌های مجاور (بالا، پایین، چپ، راست) گسترش دهد.

قوانین گسترش: آتش به خانه‌هایی که خودشان آتش، ساختمان، پایگاه یا یک عامل هستند، سرایت نمی‌کند.

نکته بسیار مهم: اگر آتش به خانه‌ای سرایت کند که یک "غیرنظامی" در آن حضور دارد، آن غیرنظامی از بین می‌رود و هر دو تیم ۱۰۰ امتیاز منفی می‌گیرند! این قانون، تیم‌ها را برای نجات سریع غیرنظامیان تحت فشار قرار می‌دهد.

```python
def get_grid(self):
    return self.grid

def update_cell(self, x, y, value):
    self.grid[x, y] = value

def increment_step(self):
    self.steps += 1

def is_game_over(self):
    return self.steps >= MAX_STEPS
```

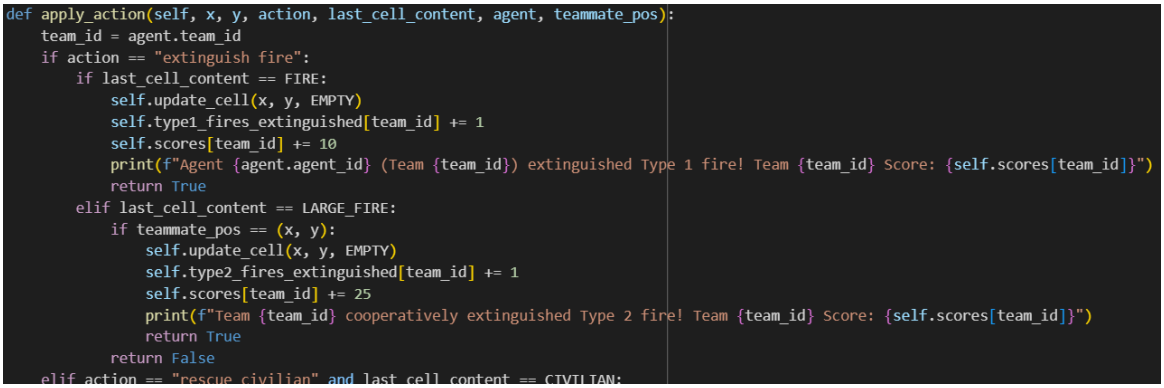
متدهای کمکی و مدیریتی

get_grid(): وضعیت فعلی نقشه را برمی‌گرداند.

update_cell(x, y, value): مقدار یک خانه مشخص از نقشه را تغییر می‌دهد.

increment_step(): شمارنده مراحل بازی را یکی اضافه می‌کند.

is_game_over(): چک می‌کند که آیا بازی به حداکثر مراحل تعریف‌شده (MAX_STEPS) رسیده است یا نه.



```
def apply_action(self, x, y, action, last_cell_content, agent, teammate_pos):
    team_id = agent.team_id
    if action == "extinguish fire":
        if last_cell_content == FIRE:
            self.update_cell(x, y, EMPTY)
            self.type1_fires_extinguished[team_id] += 1
            self.scores[team_id] += 10
            print(f"Agent {agent.agent_id} (Team {team_id}) extinguished Type 1 fire! Team {team_id} Score: {self.scores[team_id]}")
            return True
        elif last_cell_content == LARGE_FIRE:
            if teammate_pos == (x, y):
                self.update_cell(x, y, EMPTY)
                self.type2_fires_extinguished[team_id] += 1
                self.scores[team_id] += 25
                print(f"Team {team_id} cooperatively extinguished Type 2 fire! Team {team_id} Score: {self.scores[team_id]}")
                return True
            return False
    elif action == "rescue civilian" and last_cell_content == CIVILIAN:
```

متد apply_action (اعمال کردن یک اقدام)

این متد، قلب تعامل بین عامل‌ها و محیط است. وقتی یک عامل تصمیم به انجام کاری می‌گیرد، این متد آن تصمیم را در دنیای بازی اعمال کرده و نتایجش را محاسبه می‌کند:

ورودی‌ها: موقعیت عامل، نوع اقدام (مثلاً "خاموش کردن آتش")، و اطلاعات دیگر مانند محتوای قبلی خانه و موقعیت هم‌تیمی را دریافت می‌کند.

منطق تصمیم‌گیری:

extinguish fire (خاموش کردن آتش):

اگر آتش از نوع ۱ باشد، آن را خاموش کرده، به تیم مربوطه ۱۰ امتیاز می‌دهد.

اگر آتش از نوع ۲ (بزرگ) باشد، فقط در صورتی خاموش می‌شود که هم‌تیمی نیز در همان خانه حضور داشته باشد (اقدام تیمی). در این صورت، تیم ۲۵ امتیاز می‌گیرد.

rescue civilian (نجات غیرنظامی): غیرنظامی را از نقشه برمی‌دارد (به این معنی که عامل او را "حمل" می‌کند).

deliver civilian (تحویل غیرنظامی): اگر عامل یک غیرنظامی را به "پایگاه" برساند، تیم ۵۰ امتیاز کسب می‌کند.

```python
class LLMAgent:
    def __init__(self, environment, chatbot, team_id, agent_num, shared_map):
        self.env = environment
        self.team_id = team_id
        self.agent_num = agent_num
        self.agent_id = f"{team_id}{agent_num}"
        self.agent_type = (AGENT_A1 if agent_num == 1 else AGENT_A2) if team_id == 'A' else (AGENT_B1 if agent_num == 1 else AGENT_B2)

        self.pos = None
        self.shared_map = shared_map
        self.carrying_civilian = False
        self.stations = [(0, 0), (14, 14)]
        self.llm_responses = []
        self.chatbot = chatbot
        self.last_cell_content = EMPTY

        self.place_agent()
        self.update_discovered_map()
```

کلاس LLMAgent

این کلاس، یک عامل هوشمند را در شبیه‌سازی تعریف می‌کند. هر عامل، یک شخصیت مستقل در بازی است که می‌تواند دنیا را "ببیند"، با هم‌تیمی خود "ارتباط" داشته باشد و مهم‌تر از همه، با استفاده از هوش مصنوعی (که در کلاس AIStudioChatBot تعریف شد) برای حرکت و اقدام بعدی خود تصمیم‌گیری کند.

این کلاس، ترکیبی از "بدن" (موقعیت، وضعیت) و "مغز" (منطق تصمیم‌گیری) عامل است.

متد __init__ (سازنده کلاس)

این متد، شناسنامه و تجهیزات اولیه عامل را مشخص می‌کند:

وابستگی‌ها: در بدو تولد، عامل به محیط بازی (environment)، ابزار گفتگو با هوش مصنوعی (chatbot) و نقشه مشترک تیم (shared_map) متصل می‌شود.

هویت: برای عامل یک شناسه منحصر به فرد (مثلاً A1)، شماره تیم (A یا B) و نوع عددی آن (که روی نقشه نمایش داده می‌شود) تعیین می‌شود.

وضعیت اولیه: وضعیت‌های اولیه عامل مانند "حمل نکردن غیرنظامی" (carrying_civilian = False) و موقعیت پایگاه‌ها (stations) مشخص می‌شود.

ورود به بازی: بلافاصله پس از ایجاد، متد place_agent را فراخوانی می‌کند تا عامل در یک نقطه از نقشه ظاهر شود و سپس با update_discovered_map اطراف خود را "نگاه" می‌کند.

```python
def place_agent(self):
    while True:
        x, y = random.randint(0, GRID_SIZE - 1), random.randint(0, GRID_SIZE - 1)
        if self.env.get_grid()[x, y] == EMPTY:
            self.env.update_cell(x, y, self.agent_type)
            self.pos = (x, y)
            self.last_cell_content = EMPTY
            break
    self.shared_map[0, 0] = STATION
    self.shared_map[14, 14] = STATION
```

```python
def update_discovered_map(self):
    x, y = self.pos
    for i in range(max(0, x - SENSOR_RANGE), min(GRID_SIZE, x + SENSOR_RANGE + 1)):
        for j in range(max(0, y - SENSOR_RANGE), min(GRID_SIZE, y + SENSOR_RANGE + 1)):
            cell_content = self.env.get_grid()[i, j]
            self.shared_map[i, j] = cell_content
```

متدهای place_agent و update_discovered_map

این دو متد، قابلیت‌های فیزیکی و حسی عامل را شبیه‌سازی می‌کنند:

place_agent: عامل را در یک نقطه خالی تصادفی روی نقشه قرار می‌دهد.

update_discovered_map (قابلیت دید): این متد، "دید" عامل را شبیه‌سازی می‌کند. عامل به اطراف خود در یک محدوده مشخص (SENSOR_RANGE) نگاه کرده و هرچه را که در دنیای واقعی بازی (self.env.grid) می‌بیند، روی "نقشه مشترک تیم" (self.shared_map) به‌روزرسانی می‌کند. این کار، مفهوم "مه جنگ" (Fog of War) در بازی‌های استراتژیک را پیاده‌سازی می‌کند؛ یعنی تیم فقط از بخش‌هایی از نقشه خبر دارد که عامل‌هایش دیده‌اند.

```python
    def generate_prompt(self, teammate_pos, teammate_carrying, enemy_positions, current_cell_content):
        # Translate the numeric cell content into a human-readable string
        cell_content_map = {
            EMPTY: "Empty Ground", CIVILIAN: "a Civilian", FIRE: "a Type 1 Fire",
            LARGE_FIRE: "a Type 2 Fire", STATION: "a Station", BUILDING: "a Building"
        }
        your_current_situation = cell_content_map.get(current_cell_content, "Unknown")
        shared_map_str = np.array2string(self.shared_map, separator=', ')

        prompt = f"""
You are an emergency response agent ({self.agent_id}) for Team {self.team_id}. Your goal is to get a hi

[STATUS & SCORES]
- Your Team's Score: {self.env.scores[self.team_id]}
```

متد generate_prompt (آماده‌سازی گزارش برای هوش مصنوعی)

این متد، مهم‌ترین بخش ارتباط با هوش مصنوعی است. وظیفه آن، تهیه یک گزارش کامل، دقیق و ساختاریافته از وضعیت فعلی برای ارسال به LLM است. این گزارش (prompt) شامل همه اطلاعاتی است که هوش مصنوعی برای یک تصمیم‌گیری خوب نیاز دارد:

وضعیت کلی: امتیاز تیم خودی و حریف.

وضعیت شخصی: موقعیت فعلی، اینکه آیا غیرنظامی حمل می‌کند یا نه.

اطلاعات تیمی: موقعیت هم‌تیمی و دشمنان دیده‌شده.

اطلاعات حیاتی: اینکه دقیقاً روی چه چیزی ایستاده است (آتش، غیرنظامی، زمین خالی).

نقشه تیمی: تصویری متنی از تمام دانش تیم درباره نقشه.

قوانین دستوری (بسیار مهم): یک مجموعه قوانین بسیار صریح و اولویت‌بندی‌شده به هوش مصنوعی داده می‌شود. این قوانین به LLM می‌گویند که تحت چه شرایطی کدام اقدام را انتخاب کند (مثلاً: "اگر غیرنظامی حمل می‌کنی و روی پایگاه هستی، حتماً اقدام 'تحویل غیرنظامی' را انتخاب کن").

فرمت خروجی: از هوش مصنوعی خواسته می‌شود که پاسخ خود را فقط در قالب یک ساختار مشخص JSON برگرداند تا خواندن آن برای کامپیوتر آسان باشد.

```python
def validate_action(self, action, teammate_pos):
    x, y = self.pos
    if action == "extinguish fire":
        return self.last_cell_content == FIRE or \
               (self.last_cell_content == LARGE_FIRE and teammate_pos == self.pos)
    elif action == "rescue civilian":
        return self.last_cell_content == CIVILIAN and not self.carrying_civilian
    elif action == "deliver civilian":
        return self.last_cell_content == STATION and self.carrying_civilian
    elif action in ["move up", "move down", "move left", "move right"]:
        new_x, new_y = x, y
        if action == "move up": new_x -= 1
        elif action == "move down": new_x += 1
        elif action == "move left": new_y -= 1
        elif action == "move right": new_y += 1
```

```python
def get_fallback_action(self, teammate_pos):
    if self.validate_action("deliver civilian", teammate_pos): return "deliver civilian"
    if self.validate_action("rescue civilian", teammate_pos): return "rescue civilian"
    if self.validate_action("extinguish fire", teammate_pos): return "extinguish fire"

    moves = ["move up", "move down", "move left", "move right"]
    random.shuffle(moves)
    for move in moves:
        if self.validate_action(move, teammate_pos):
            return move
    return "none" # No valid move available
```

متدهای validate_action و get_fallback_action

این دو متد، سیستم‌های ایمنی و پشتیبان عامل هستند:

validate_action (بررسی اعتبار اقدام): این متد چک می‌کند که آیا دستوری که از LLM آمده، مطابق قوانین بازی "مجاز" است یا نه. برای مثال، آیا واقعاً آتشی برای خاموش کردن وجود دارد یا آیا خانه‌ای که می‌خواهد به آن حرکت کند، دیوار (ساختمان) نیست؟ این کار از خطاهای احتمالی یا "توهمات" هوش مصنوعی جلوگیری می‌کند.

get_fallback_action (اقدام جایگزین): اگر ارتباط با LLM برقرار نشد، یا پاسخی نامفهوم داد، یا دستوری غیرمجاز صادر کرد، این متد وارد عمل می‌شود. این متد یک اقدام "ایمن" و از پیش تعریف‌شده (مثل حرکت تصادفی به یک خانه مجاز) را انتخاب می‌کند تا عامل در بازی متوقف نشود.

```python
async def move(self, teammate_pos, teammate_carrying, enemy_positions):
    self.update_discovered_map()
    # *** IMPORTANT CHANGE: Pass self.last_cell_content to the prompt generator ***
    prompt = self.generate_prompt(teammate_pos, teammate_carrying, enemy_positions, self.last_cell_content)
    response_entry = {"step": self.env.steps, "agent_id": self.agent_id, "prompt": prompt, "action": "none", "reasoning": "N/A", "fa

    action = None
    try:
        response_text = await asyncio.wait_for(
            self.chatbot.chat(prompt),
            timeout=25
        )

        if not response_text or not response_text.strip():
            raise ValueError("Empty response from LLM")

        json_match = re.search(r'```json\n([\s\S]*?)\n```', response_text, re.DOTALL)
        if json_match:
```

متد async def move (حلقه تصمیم‌گیری در هر نوبت)

این متد، فرآیند کامل "فکر کردن" و "تصمیم گرفتن" عامل در هر نوبت بازی است:

اطراف خود را نگاه کرده و نقشه را به‌روز می‌کند.

گزارش کامل (prompt) را با generate_prompt آماده می‌کند.

سعی می‌کند با chatbot.chat از LLM دستور بگیرد.

پاسخ LLM را تحلیل کرده و دستور را از داخل JSON استخراج می‌کند.

با validate_action، دستور را بررسی می‌کند.

اگر دستور نامعتبر بود، با get_fallback_action یک دستور جایگزین انتخاب می‌کند.

تمام این فرآیند (سوال، جواب، انتخاب نهایی) را برای تحلیل‌های بعدی ثبت و لاگ می‌کند.

در نهایت، دستور نهایی و معتبر را به متد execute_action می‌دهد تا اجرا شود.

```python
def execute_action(self, action, teammate_pos):
    x, y = self.pos
    if action in ["extinguish fire", "rescue civilian", "deliver civilian"]:
        success = self.env.apply_action(x, y, action, self.last_cell_content, self, teammate_pos)
        if success:
            if action == "rescue civilian":
                self.carrying_civilian = True
                print(f"Agent {self.agent_id} picked up civilian!")
            elif action == "deliver civilian":
                self.carrying_civilian = False

            if action == "deliver civilian":
                self.last_cell_content = STATION
            else:
                self.last_cell_content = EMPTY
            self.shared_map[x, y] = self.last_cell_content
```

متد execute_action (اجرای دستور)

این متد، "دست و پا"ی عامل است. دستوری که از "مغز" (متد move) آمده را در محیط بازی پیاده می‌کند:

اگر دستور، یک اقدام باشد (مثل خاموش کردن آتش یا نجات غیرنظامی)، متد apply_action از کلاس Environment را فراخوانی می‌کند تا تغییرات در دنیای بازی اعمال شود و وضعیت داخلی خود را (مثلاً self.carrying_civilian) به‌روز می‌کند.

اگر دستور، یک حرکت باشد، موقعیت خود را روی نقشه تغییر می‌دهد. نکته کلیدی اینجاست که ابتدا خانه قبلی خود را به وضعیت اصلی‌اش برمی‌گرداند و سپس در خانه جدید مستقر می‌شود.

```python
def display_competitive_maps(env, team_a_agents, team_b_agents, step):
    fig, (ax1, ax2, ax3) = plt.subplots(1, 3, figsize=(21, 6))

    ax1.imshow(env.get_grid(), cmap=FULL_CMAP, interpolation='nearest', vmin=0, vmax=max(COLORS.keys()))
    ax1.set_title(f"Full City Map (Step {step})")
    ax1.axis('off')

    ax2.imshow(team_a_agents[0].shared_map, cmap=DISCOVERED_CMAP, interpolation='nearest', vmin=-1, vmax=max(COLORS.keys()))
    ax2.set_title(f"Team A Discovered Map | Score: {env.scores['A']}")
    ax2.axis('off')

    ax3.imshow(team_b_agents[0].shared_map, cmap=DISCOVERED_CMAP, interpolation='nearest', vmin=-1, vmax=max(COLORS.keys()))
    ax3.set_title(f"Team B Discovered Map | Score: {env.scores['B']}")
    ax3.axis('off')

    legend_elements = [plt.Rectangle((0,0),1,1, color=color, label=label) for label, color in COLOR_LEGEND.items()]
    fig.legend(handles=legend_elements, loc='lower center', ncol=3, bbox_to_anchor=(0.5, -0.05))

    plt.tight_layout(rect=[0, 0, 1, 0.95])

    if not os.path.exists('game_steps'):
        os.makedirs('game_steps')
```

تابع display_competitive_maps (نقشه‌کش و گزارشگر تصویری)

این تابع وظیفه به تصویر کشیدن وضعیت بازی در هر لحظه را بر عهده دارد. می‌توان آن را مانند یک "گزارشگر تصویری" یا "دوربین ناظر" در نظر گرفت.

ورودی‌ها: وضعیت فعلی محیط (env)، لیست عامل‌های تیم A و B، و شماره مرحله (step).

عملکرد:

یک پنجره با سه نقشه در کنار هم ایجاد می‌کند:

نقشه سمت چپ (Full City Map): این نقشه، دید کامل و واقعی از کل شهر است. همه چیز (آتش، غیرنظامیان، عامل‌ها) در موقعیت واقعی‌شان نمایش داده می‌شود. این "دید خداگونه" (God's-eye view) است.

نقشه وسط (Team A Discovered Map): این نقشه، دانش تیم A از محیط را نشان می‌دهد. فقط چیزهایی که عامل‌های تیم A تا آن لحظه با "سنسور" خود دیده‌اند در آن مشخص است و بقیه نقشه "ناشناخته" (سیاه) باقی می‌ماند. امتیاز تیم A نیز در عنوان این نقشه نمایش داده می‌شود.

نقشه سمت راست (Team B Discovered Map): این نقشه نیز مشابه نقشه وسط، دانش و دید تیم B را نشان می‌دهد.

در پایین تصویر، یک راهنمای رنگ‌ها (Legend) نمایش می‌دهد تا مشخص باشد هر رنگ نماد چه عنصری است.

نکته بسیار جالب: این تابع هر تصویر را با نام مرحله فعلی (مثلاً map_step_01.png) در یک پوشه به نام game_steps ذخیره می‌کند. این کار به شما اجازه می‌دهد که بعد از اتمام شبیه‌سازی، یک فیلم یا انیمیشن از روند کامل بازی بسازید!

```python
async def run_competitive_simulation():
    env = Environment()
    chatbot = AIStudioChatBot(AI_STUDIO_API_URL, AI_STUDIO_API_KEY)

    shared_map_A = np.full((GRID_SIZE, GRID_SIZE), -1, dtype=int)
    shared_map_B = np.full((GRID_SIZE, GRID_SIZE), -1, dtype=int)

    agent_A1 = LLMAgent(env, chatbot, 'A', 1, shared_map_A)
    agent_A2 = LLMAgent(env, chatbot, 'A', 2, shared_map_A)
    agent_B1 = LLMAgent(env, chatbot, 'B', 1, shared_map_B)
    agent_B2 = LLMAgent(env, chatbot, 'B', 2, shared_map_B)

    team_A = [agent_A1, agent_A2]
    team_B = [agent_B1, agent_B2]
    all_agents = [agent_A1, agent_A2, agent_B1, agent_B2]

    display_competitive_maps(env, team_A, team_B, env.steps)
```

تابع async def run_competitive_simulation (موتور اصلی شبیه‌سازی)

این تابع، قلب تپنده و کارگردان اصلی کل پروژه است. این تابع همه چیز را از ابتدا تا انتها مدیریت می‌کند.

الف) بخش آماده‌سازی و راه‌اندازی (Setup)

یک Environment (محیط بازی) و یک AIStudioChatBot (ابزار ارتباط با هوش مصنوعی) می‌سازد.

دو "نقشه مشترک" خالی (shared_map_A و shared_map_B) برای هر تیم ایجاد می‌کند. این نقشه‌ها در ابتدا کاملاً "ناشناخته" هستند.

چهار عامل LLMAgent (دو عامل برای تیم A و دو عامل برای تیم B) را ایجاد می‌کند. نکته کلیدی این است که هر دو عامل تیم A به یک نقشه مشترک متصل می‌شوند. اینگونه است که دانش آن‌ها با هم به اشتراک گذاشته می‌شود.

در همان ابتدا، با display_competitive_maps وضعیت اولیه بازی (مرحله ۰) را نمایش می‌دهد.

```
while not env.is_game_over():
    env.increment_step()
    print(f"\n--- Starting Step {env.steps}/{MAX_STEPS} ---")

    random.shuffle(all_agents)

    for agent in all_agents:
        if agent.team_id == 'A':
            teammate = agent_A2 if agent is agent_A1 else agent_A1
            enemies = [agent_B1.pos, agent_B2.pos]
        else: # Team B
            teammate = agent_B2 if agent is agent_B1 else agent_B1
            enemies = [agent_A1.pos, agent_A2.pos]

        await agent.move(teammate.pos, teammate.carrying_civilian, enemies)

        await asyncio.sleep(5)

    if env.steps % 5 == 0:
```

حلقه اصلی بازی (Main Game Loop)

این حلقه while تا زمانی که بازی تمام نشده، به طور مداوم تکرار می‌شود و اتفاقات هر "نوبت" یا "مرحله" در آن رخ می‌دهد:

افزایش مرحله: شماره مرحله بازی را یکی بالا می‌برد.

بر زدن عامل‌ها (random.shuffle): قبل از اینکه عامل‌ها حرکت کنند، ترتیب آن‌ها را به صورت تصادفی به هم می‌ریزد. این کار بسیار هوشمندانه است و باعث می‌شود که هیچ تیمی همیشه برتری "حرکت اول" را در هر نوبت نداشته باشد و بازی عادلانه‌تر شود.

نوبت هر عامل: در یک حلقه for، به ترتیب (که اکنون تصادفی شده) به هر عامل اجازه حرکت می‌دهد:

برای هر عامل، هم‌تیمی و دشمنانش را مشخص می‌کند.

مهم‌ترین بخش: متد await agent.move(...) را فراخوانی می‌کند. در این لحظه، عامل با هوش مصنوعی مشورت کرده، تصمیم گرفته و حرکت یا اقدام خود را انجام می‌دهد.

await asyncio.sleep(5): بعد از حرکت هر عامل، برنامه برای ۵ ثانیه مکث می‌کند. این کار یک دلیل کاملاً عملی دارد: جلوگیری از ارسال درخواست‌های بیش از حد سریع به API گوگل و رد شدن درخواست‌ها به دلیل شلوغی (Rate Limit).

گسترش آتش: هر ۵ مرحله یک‌بار، متد env.spread_fire() فراخوانی می‌شود تا آتش در محیط گسترش پیدا کند. این کار به بازی پویایی و هیجان می‌بخشد.

نمایش وضعیت جدید: پس از اینکه همه عامل‌ها حرکت کردند و آتش گسترش یافت، دوباره display_competitive_maps فراخوانی می‌شود تا وضعیت جدید نقشه نمایش داده شود.

```
# --- FINAL REPORT (No changes here) ---
print("\n\n" + "="*30)
print("      SIMULATION COMPLETE")
print("="*30 + "\n")

# ... (rest of the function remains the same)
score_A = env.scores['A']
score_B = env.scores['B']

winner = "DRAW"
if score_A > score_B:
    winner = "Team A"
elif score_B > score_A:
    winner = "Team B"
else:
    fires_A = env.type1_fires_extinguished['A'] + env.type2_fires_extinguished['A']
    fires_B = env.type1_fires_extinguished['B'] + env.type2_fires_extinguished['B']
    if fires_A > fires_B:
        winner = "Team A (by tie-breaker)"
    elif fires_B > fires_A:
        winner = "Team B (by tie-breaker)"
```

بخش گزارش نهایی و تحلیل (Final Report)

وقتی حلقه بازی تمام می‌شود، این بخش اجرا می‌شود تا نتایج را اعلام کند:

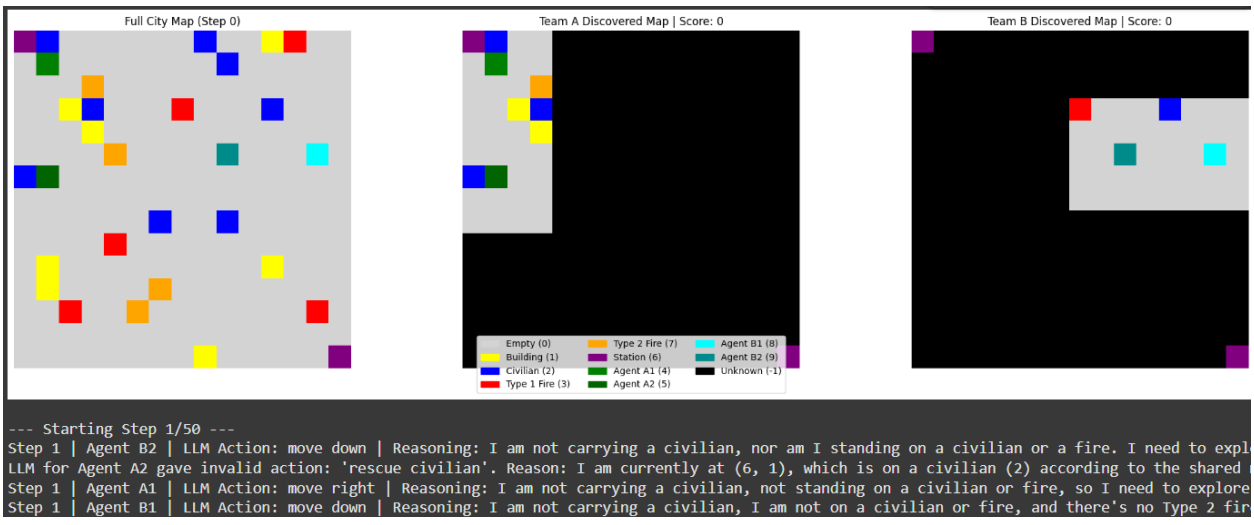
یک پیام "SIMULATION COMPLETE" چاپ می‌کند.

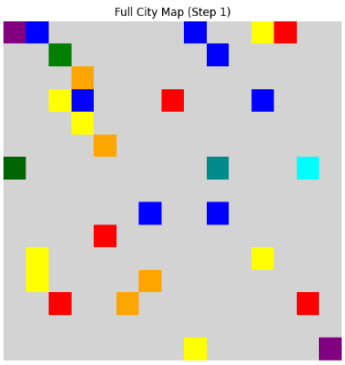
برنده را اعلام می‌کند: بر اساس امتیازات، برنده را مشخص می‌کند. حتی یک قانون تساوی‌شکن (tie-breaker) هم دارد: اگر امتیازها مساوی بود، تیمی که آتش بیشتری خاموش کرده برنده است.

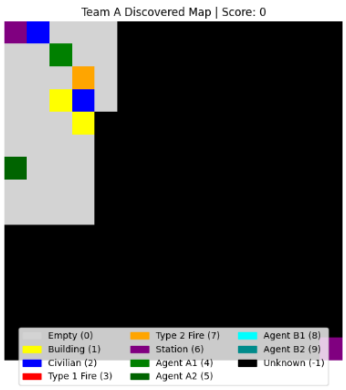
گزارش عملکرد تیم‌ها: برای هر تیم، آمار دقیقی از عملکردش ارائه می‌دهد: تعداد غیرنظامیان نجات‌یافته، آتش‌های خاموش‌شده، و یک معیار بسیار جالب به نام "نسبت اقدامات معتبر LLM". این درصد نشان می‌دهد که هوش مصنوعی چند بار دستورات صحیح و قابل اجرا صادر کرده و چند بار سیستم به "اقدام جایگزین" (fallback) پناه برده است. این یک معیار عالی برای سنجش کیفیت prompt شماست.

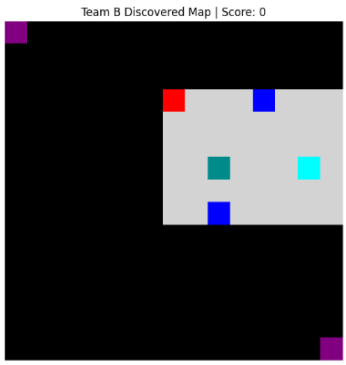
ثبت کامل تصمیمات: تمام مکالمات (promptها)، استدلال‌ها و اقدامات نهایی هر عامل در هر مرحله را در یک فایل متنی به نام decision_log.txt ذخیره می‌کند. این فایل برای تحلیل و اشکال‌زدایی رفتار هوش مصنوعی فوق‌العاده ارزشمند است.

بخشی از خروجی:

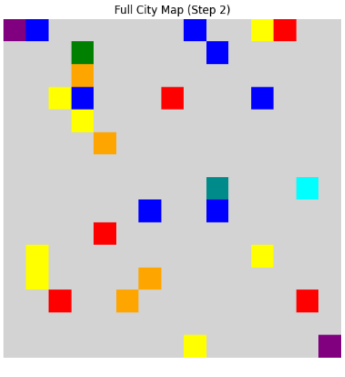


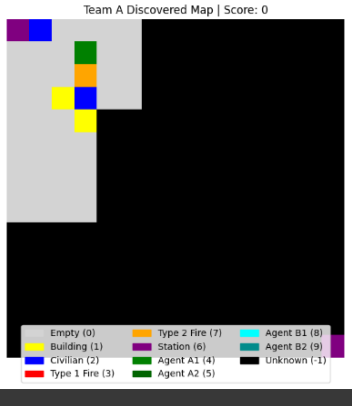


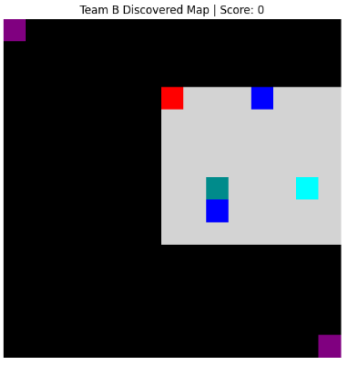




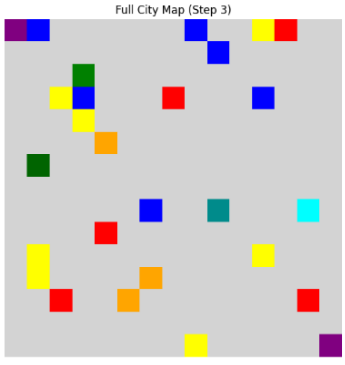

```
--- Starting Step 2/50 ---
Step 2 | Agent B2 | LLM Action: move down | Reasoning: I am not carrying a civilian, not standing on a civilian or a fire. Therefore, I should
Step 2 | Agent A2 | LLM Action: rescue civilian | Reasoning: I am currently standing on a civilian and not carrying one, so I should rescue the
Agent A2 picked up civilian!
Step 2 | Agent B1 | LLM Action: move down | Reasoning: I am not carrying a civilian, not standing on a civilian, and not standing on a fire. I
Step 2 | Agent A1 | LLM Action: move right | Reasoning: I am not carrying a civilian, not standing on a civilian, and not standing on a fire.
```

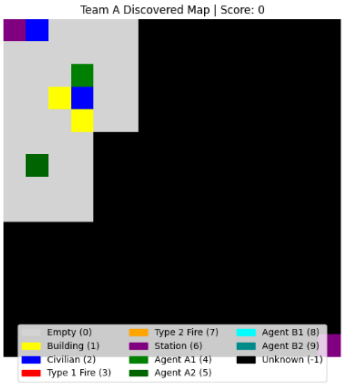


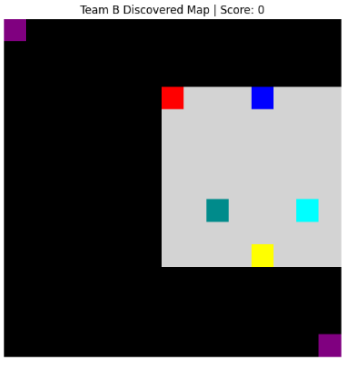




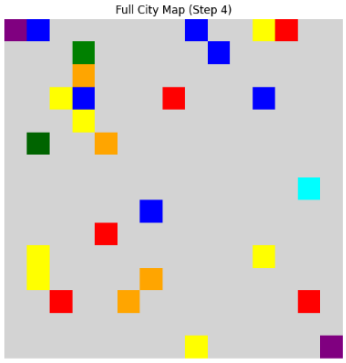

```
--- Starting Step 3/50 ---
LLM for Agent A1 gave invalid action: 'rescue civilian'. Reason: I am at (1, 3) and not carrying a civilian. The map shows a civilian (2) at (1
LLM for Agent B2 gave invalid action: 'rescue civilian'. Reason: I am not carrying a civilian and I am standing on a civilian (2) at (7, 9), so
Step 3 | Agent B1 | LLM Action: move down | Reasoning: I am not carrying a civilian, not on a station, and not on a fire. There is no type 2 fi
LLM for Agent A2 gave invalid action: 'deliver civilian'. Reason: I am carrying a civilian and standing on a station (6), so the highest priori
```

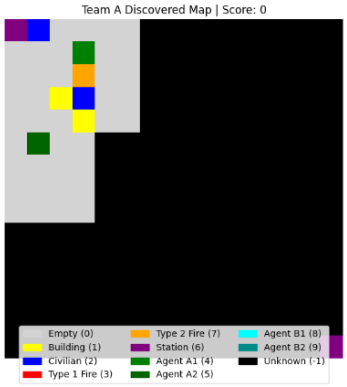


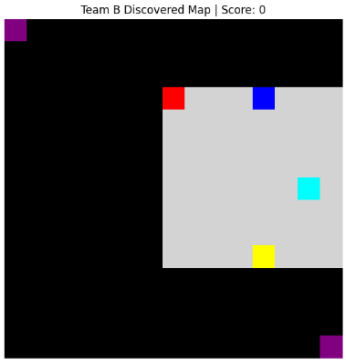




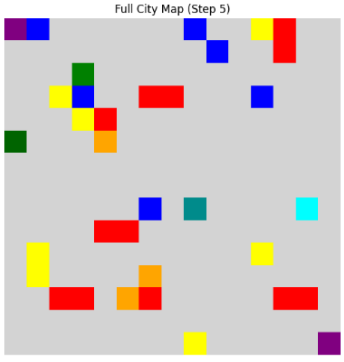

```
--- Starting Step 4/50 ---
LLM for Agent A1 gave invalid action: 'extinguish fire'. Reason: I am currently standing on a Type 2 Fire (7). My teammate is at position (6, 1
Step 4 | Agent B2 | LLM Action: rescue civilian | Reasoning: I am currently standing on a civilian and I am not carrying a civilian, so I shoul
Agent B2 picked up civilian!
Step 4 | Agent B1 | LLM Action: move up | Reasoning: I am not carrying a civilian, nor am I standing on a civilian or fire. I need to explore t
LLM for Agent A2 gave invalid action: 'deliver civilian'. Reason: I am carrying a civilian and I am standing on a station (6), so I must delive
```

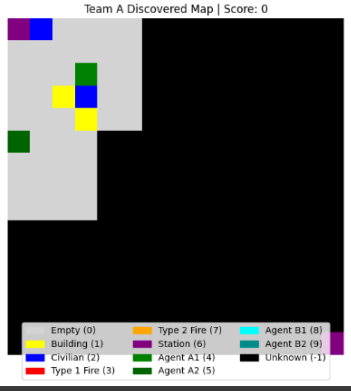


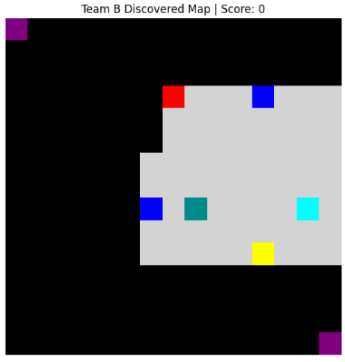




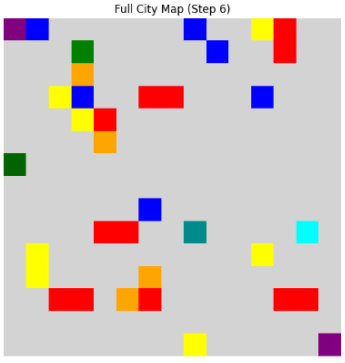

```
--- Starting Step 5/50 ---
LLM for Agent B2 gave invalid action: 'move to deliver'. Reason: I am carrying a civilian and I am not at a station. I need to move towards th
LLM for Agent A2 gave invalid action: 'move to deliver'. Reason: I am carrying a civilian and am not at a station. The nearest station is at (
Step 5 | Agent B1 | LLM Action: move down | Reasoning: I am not carrying a civilian. I am not on a station, a civilian, or a fire. Therefore,
LLM for Agent A1 gave invalid action: 'extinguish fire'. Reason: I am at (1,3) and standing on Empty Ground, but the shared map shows a Type 2

!!! FIRE IS SPREADING !!!
```

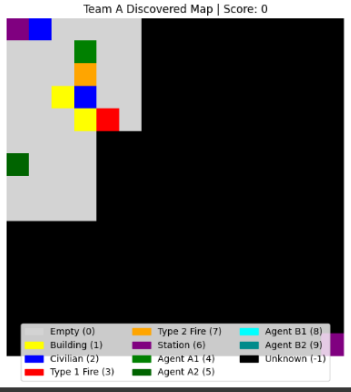


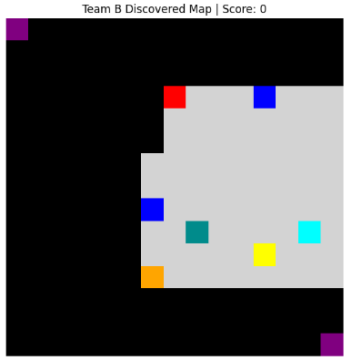




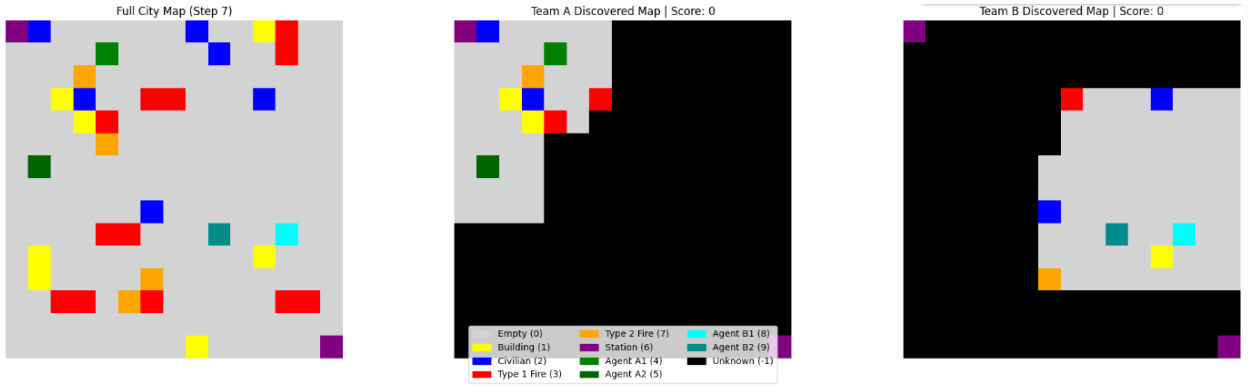

```
--- Starting Step 6/50 ---
LLM for Agent A2 gave invalid action: 'move to deliver'. Reason: I am carrying a civilian and I am not at a station. The nearest station is at
Step 6 | Agent B1 | LLM Action: move down | Reasoning: I am not carrying a civilian, not standing on a civilian, and not standing on a fire. Th
LLM for Agent B2 gave invalid action: 'move to deliver'. Reason: I am carrying a civilian and not at a station. The nearest station is at (0,0)
LLM for Agent A1 gave invalid action: 'extinguish fire'. Reason: I am standing on a Type 2 Fire (7). According to the condition for extinguishi
```



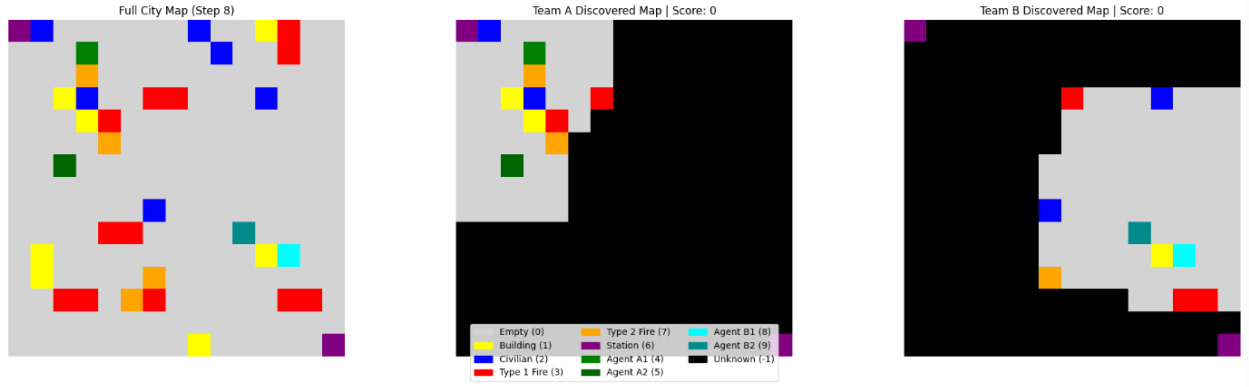





```
--- Starting Step 7/50 ---
Step 7 | Agent B1 | LLM Action: move left | Reasoning: I am not carrying a civilian, not standing on a civilian or fire, and there is no type 2
LLM for Agent A2 gave invalid action: 'deliver civilian'. Reason: I am carrying a civilian and currently standing on a station (6). Therefore,
Step 7 | Agent A1 | LLM Action: move right | Reasoning: I am not carrying a civilian, nor am I standing on a civilian, fire, or station. There
LLM for Agent B2 gave invalid action: 'move to deliver'. Reason: I am carrying a civilian and I am not at a station. I need to move towards the
```




```
--- Starting Step 8/50 ---
Step 8 | Agent A1 | LLM Action: move left | Reasoning: I am not carrying a civilian. There is a civilian at (1, 1). Moving left will bring me c
LLM for Agent A2 gave invalid action: 'move to deliver'. Reason: I am carrying a civilian and not at a station. The nearest station is (0,0) wh
Step 8 | Agent B1 | LLM Action: move down | Reasoning: I am not carrying a civilian, not on a fire, and not on a station. My teammate is at (9,
LLM for Agent B2 gave invalid action: 'move to deliver'. Reason: I am carrying a civilian and I am not at a station. Therefore, I must move tow
```




```
--- Starting Step 9/50 ---
LLM for Agent A2 gave invalid action: 'move to deliver'. Reason: I am carrying a civilian and not at a station. Therefore, I need to move towar
Step 9 | Agent A1 | LLM Action: move right | Reasoning: I am not carrying a civilian. I am not standing on a civilian, fire, or station. Theref
LLM for Agent B2 gave invalid action: 'move to deliver'. Reason: I am carrying a civilian and I am not at a station. The nearest station is at
LLM for Agent B1 gave invalid action: 'move left'. Reason: I am not carrying a civilian, not standing on a civilian or fire. I need to explore
```

نتیجه نهایی:

```
==============================
      SIMULATION COMPLETE
==============================

WINNER: Team B

--- Final Scores ---
Team A: -410
Team B: -360

--- Team A Performance ---
Civilians Rescued: 0/8
Type 1 Fires Extinguished: 19/5
Type 2 Fires Extinguished: 0/4
Valid LLM Action Ratio: 49/100 (49.00%)

--- Team B Performance ---
Civilians Rescued: 0/8
Type 1 Fires Extinguished: 24/5
Type 2 Fires Extinguished: 0/4
Valid LLM Action Ratio: 58/100 (58.00%)
```

فایل لاگ و تمامی عکس ها نیز در فایل تمرین قرار داده شده است.

برای رفع برخی ایرادات و ابهامات از AI استفاده شده است.