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

```python
class Environment:
    def __init__(self):
        self.grid = np.zeros((GRID_SIZE, GRID_SIZE), dtype=int)
        self.civilians_rescued = 0
        self.fires_extinguished = 0
        self.score = 0
        self.steps = 0
        self.setup_environment()

    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_FIRES):
            self.place_random_entity(FIRE)
```

کلاس Environment شامل یک شبکه (grid) دوبعدی است که در آن عناصری مانند آتش، غیرنظامیان، ساختمان‌ها، ایستگاه‌ها و (agent) قرار دارند.

متد __init__ : این متد هنگام ایجاد یک نمونه از کلاس Environment اجرا می‌شود و وظیفه مقداردهی اولیه به ویژگی‌های محیط را بر عهده دارد :

یک شبکه دوبعدی با ابعاد GRID_SIZE × GRID_SIZE ایجاد می‌شود که تمام خانه‌های آن در ابتدا خالی (صفر) هستند.

متغیرهایی برای شمارش تعداد غیرنظامیان نجات‌یافته(civilians_rescued) ، آتش‌های خاموش‌شده (fires_extinguished) ، امتیاز (score) و تعداد گام‌ها (steps) تعریف می‌شوند.

در نهایت، متد setup_environment فراخوانی می‌شود تا محیط اولیه را آماده کند

متد setup_environment: این متد وظیفه پیکربندی اولیه شبکه را بر عهده دارد :

دو ایستگاه (STATION) در موقعیت‌های ثابت [0, 0] و [14, 14] قرار می‌گیرند.

تعدادی ساختمان(NUM_BUILDINGS) ، غیرنظامی (NUM_CIVILIANS) و آتش (NUM_FIRES) به‌صورت تصادفی در شبکه قرار داده می‌شوند.

قرارگیری این عناصر از طریق متد place_random_entity انجام می‌شود

```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: این متد یک عنصر (مانند آتش، غیرنظامی یا ساختمان) را به‌صورت تصادفی در شبکه قرار می‌دهد :

مختصات تصادفی (x, y)انتخاب می‌شود.

اگر خانه موردنظر خالی (EMPTY) باشد، عنصر موردنظر در آن قرار می‌گیرد. در غیر این صورت، این فرآیند تکرار می‌شود تا یک خانه خالی پیدا شود.

```python
def spread_fire(self):
    fire_positions = list(zip(*np.where(self.grid == FIRE)))
    if not fire_positions:
        return
    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 and
                self.grid[new_x, new_y] not in [FIRE, BUILDING, STATION, AGENT]):
                if self.grid[new_x, new_y] == CIVILIAN:
                    self.score -= 100
                    print(f"Civilian died in fire! Score/MQ: {self.score}")
                self.grid[new_x, new_y] = FIRE
                break
```

متد spread_fire: این متد رفتار گسترش آتش در شبکه را شبیه‌سازی می‌کند :

تمام موقعیت‌های حاوی آتش در شبکه شناسایی می‌شوند.

برای هر آتش، یکی از چهار جهت (بالا، پایین، چپ، راست) به‌صورت تصادفی انتخاب می‌شود.

اگر خانه مجاور در محدوده شبکه باشد و حاوی آتش، ساختمان، ایستگاه یا مأمور نباشد، آتش به آن خانه گسترش می‌یابد.

اگر خانه مجاور حاوی غیرنظامی باشد، امتیاز 100 واحد کم می‌شود و پیام مرگ غیرنظامی نمایش داده می‌شود.

این فرآیند برای هر آتش فقط یک‌بار انجام می‌شود به دلیل استفاده از break

```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 or (self.fires_extinguished == NUM_FIRES and self.civilians_rescued == NUM_CIVILIANS)
```

متد get_grid: این متد شبکه فعلی را به‌عنوان خروجی برمی‌گرداند تا وضعیت محیط قابل بررسی باشد.

متد update_cell: این متد محتوای یک خانه خاص (x, y) در شبکه را به مقدار مشخص‌شده تغییر می‌دهد.

متد increment_step: این متد تعداد گام‌های سپری‌شده (steps) را یک واحد افزایش می‌دهد.

متد is_game_over: این متد بررسی می‌کند که آیا بازی به پایان رسیده است یا خیر.

بازی در دو حالت پایان می‌یابد :

- تعداد گام‌ها به حداکثر مقدار مجاز (MAX_STEPS) رسیده باشد.
- تمام آتش‌ها خاموش شده و تمام غیرنظامیان نجات یافته باشند..

```python
def apply_action(self, x, y, action, last_cell_content):
    """
    Apply the specified action at position (x, y) based on the last known cell content.
    Returns True if the action was successful, False otherwise.
    """
    if action == "extinguish" and last_cell_content == FIRE:
        self.update_cell(x, y, EMPTY)  # Update grid to EMPTY
        self.fires_extinguished += 1
        self.score += 10
        print(f"Fire extinguished! Score: {self.score}")
        return True
    elif action == "rescue" and last_cell_content == CIVILIAN:
        self.update_cell(x, y, EMPTY)  # Update grid to EMPTY
        return True  # Note: civilians_rescued is incremented in deliver civilian
    elif action == "drop" and last_cell_content == STATION:
        self.civilians_rescued += 1
        self.score += 50
        print(f"Civilian rescued! Score: {self.score}")
        return True
    return False
```

متد apply_action: این متد اقداماتی که مأمور می‌تواند در یک موقعیت خاص (x, y) انجام دهد را مدیریت می‌کند. ورودی‌های این متد شامل مختصات، نوع اقدام (action) و محتوای قبلی خانه (last_cell_content) است. اقدامات ممکن عبارت‌اند از :

- **خاموش کردن آتش(extinguish)** : اگر خانه حاوی آتش باشد، آن را به حالت خالی تغییر می‌دهد، تعداد آتش‌های خاموش‌شده را افزایش می‌دهد و 10 امتیاز اضافه می‌کند.
- **نجات غیرنظامی(rescue)** : اگر خانه حاوی غیرنظامی باشد، آن را به حالت خالی تغییر می‌دهد (شمارش نجات غیرنظامی در مرحله تحویل به ایستگاه انجام می‌شود).
- **تحویل به ایستگاه(drop)** : اگر خانه یک ایستگاه باشد، تعداد غیرنظامیان نجات‌یافته افزایش می‌یابد و 50 امتیاز اضافه می‌شود.
- اگر اقدام نامعتبر باشد، متد False برمی‌گرداند.

```python
class LLMAgent:
    def __init__(self, environment, email, password):
        self.env = environment
        self.pos = None
        self.discovered_map = np.full((GRID_SIZE, GRID_SIZE), -1)
        self.carrying_civilian = False
        self.known_fires = []
        self.known_civilians = []
        self.known_buildings = []
        self.stations = [(0, 0), (14, 14)]
        self.llm_responses = []
        self.chatbot = self.initialize_chatbot(email, password)
        self.last_cell_content = EMPTY  # Track the cell's content before AGENT overwrite
        self.place_agent()
        self.update_discovered_map()
```

متد __init__:

این متد هنگام ایجاد یک نمونه از کلاس LLMAgent اجرا می‌شود و وظیفه مقداردهی اولیه به ویژگی‌های مأمور (agent) را بر عهده دارد. عملکردهای اصلی این متد عبارت‌اند از:

دریافت ورودی‌ها :متد سه ورودی شامل شیء محیط(environment) ، ایمیل (email) و رمز عبور (password) برای اتصال به چت‌بات دریافت می‌کند.

تنظیم ویژگی‌های مأمور :

self.env: ذخیره شیء محیط برای تعامل با شبکه شبیه‌سازی.

self.pos: موقعیت مأمور که در ابتدا مقدار ندارد و بعداً تعیین می‌شود.

self.discovered_map: یک شبکه دوبعدی به اندازه GRID_SIZE × GRID_SIZE که در ابتدا تمام خانه‌های آن به مقدار 1-(ناشناخته) تنظیم می‌شوند.

self.carrying_civilian: نشان‌دهنده این است که آیا مأمور در حال حمل غیرنظامی است یا خیر (در ابتدا خیر).

self.known_fires، self.known_civilians، self.known_buildings: لیست‌هایی برای ذخیره موقعیت‌های شناخته‌شده آتش‌ها، غیرنظامیان و ساختمان‌ها.

self.stations: لیست ایستگاه‌ها با مختصات ثابت (0, 0) و (14, 14).

self.llm_responses: لیستی برای ذخیره پاسخ‌های چت‌بات.

self.last_cell_content: برای ردیابی محتوای خانه‌ای که مأمور در آن قرار دارد (در ابتدا EMPTY).

راه‌اندازی چت‌بات :متد initialize_chatbot با ایمیل و رمز عبور فراخوانی می‌شود تا چت‌بات آماده شود.

قرار دادن مأمور :متد place_agent فراخوانی می‌شود تا مأمور در یک موقعیت تصادفی خالی قرار گیرد.

به‌روزرسانی نقشه :متد update_discovered_map فراخوانی می‌شود تا اطلاعات اولیه محیط اطراف مأمور ثبت شود.

```
def initialize_chatbot(self, email, password):
    sign = Login(email, password)
    cookies = sign.login()
    chatbot = ChatBot(cookies=cookies.get_dict())
    return chatbot
```

متد initialize_chatbot

این متد وظیفه ایجاد و راه‌اندازی چت‌بات را دارد که احتمالاً برای تصمیم‌گیری‌های مأمور استفاده می‌شود. عملکردهای اصلی آن عبارت‌اند از:

ورود به سیستم :با استفاده از ایمیل و رمز عبور، یک شیء Login ایجاد شده و فرآیند ورود (login) انجام می‌شود.

دریافت کوکی‌ها :پس از ورود موفق، کوکی‌های جلسه دریافت می‌شوند که برای احراز هویت چت‌بات لازم هستند.

ایجاد چت‌بات :یک شیء ChatBot با استفاده از کوکی‌های دریافت‌شده ساخته می‌شود.

بازگرداندن چت‌بات :شیء چت‌بات به‌عنوان خروجی برگردانده می‌شود تا در سایر متدها برای تعامل با مدل زبانی استفاده شود.

این متد اتصال به یک سرویس خارجی (مانند مدل زبانی) را برقرار می‌کند تا مأمور بتواند از هوش مصنوعی برای انتخاب اقدامات استفاده کند.

```
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, AGENT)
            self.pos = [x, y]
            self.last_cell_content = EMPTY
            break
    self.discovered_map[0, 0] = STATION
    self.discovered_map[14, 14] = STATION
```

متد place_agent

این متد مأمور را در یک موقعیت تصادفی در شبکه محیط قرار می‌دهد. عملکردهای اصلی آن عبارت‌اند از:

انتخاب موقعیت تصادفی :مختصات (x, y) به‌صورت تصادفی در شبکه انتخاب می‌شوند.

بررسی خالی بودن خانه :اگر خانه انتخاب‌شده در شبکه محیط خالی (EMPTY) باشد، مأمور در آن قرار می‌گیرد. در غیر این صورت، این فرآیند تکرار می‌شود تا یک خانه خالی پیدا شود.

به‌روزرسانی محیط :با استفاده از متد update_cell از کلاس Environment، خانه انتخاب‌شده به‌عنوان حاوی مأمور (AGENT) علامت‌گذاری می‌شود.

ثبت موقعیت مأمور :مختصات (x, y) در self.pos ذخیره می‌شوند.

تنظیم محتوای اولیه خانه :متغیر self.last_cell_content به EMPTY تنظیم می‌شود، زیرا مأمور در خانه خالی قرار گرفته است.

ثبت ایستگاه‌ها :مختصات ایستگاه‌ها (0, 0) و (14, 14) به‌عنوان ایستگاه (STATION) در نقشه کشف‌شده (self.discovered_map) ثبت می‌شوند.

```
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 = self.env.get_grid()[i, j]
            self.discovered_map[i, j] = cell
            pos = (i, j)
            if cell == FIRE and pos not in self.known_fires:
                self.known_fires.append(pos)
            elif cell == CIVILIAN and pos not in self.known_civilians:
                self.known_civilians.append(pos)
            elif cell == BUILDING and pos not in self.known_buildings:
                self.known_buildings.append(pos)
            if cell != FIRE and pos in self.known_fires:
                self.known_fires.remove(pos)
            if cell != CIVILIAN and pos in self.known_civilians:
                self.known_civilians.remove(pos)
```

متد update_discovered_map

این متد نقشه کشف‌شده (self.discovered_map) را بر اساس موقعیت فعلی مأمور و محدوده حسگر (SENSOR_RANGE) به‌روزرسانی می‌کند. عملکردهای اصلی آن عبارت‌اند از:

به‌روزرسانی نقشه کشف‌شده :محتوای هر خانه از شبکه محیط (self.env.get_grid()) خوانده شده و در self.discovered_map ثبت می‌شود.

مدیریت لیست‌های شناخته‌شده :

- اگر خانه حاوی آتش (FIRE) باشد و در self.known_fires نباشد، به لیست اضافه می‌شود.
- اگر خانه حاوی غیرنظامی (CIVILIAN) باشد و در self.known_civilians نباشد، به لیست اضافه می‌شود.
- اگر خانه حاوی ساختمان (BUILDING) باشد و در self.known_buildings نباشد، به لیست اضافه می‌شود.
- اگر خانه‌ای قبلاً به‌عنوان آتش یا غیرنظامی ثبت شده باشد اما دیگر حاوی آن‌ها نباشد (مثلاً آتش خاموش شده یا غیرنظامی نجات یافته)، از لیست مربوطه حذف می‌شود.

این متد اطلاعات محیطی مأمور را به‌روز نگه می‌دارد تا تصمیم‌گیری‌های دقیق‌تری انجام شود.

```python
def get_perception_view(self):
    x, y = self.pos
    view = np.full((5, 5), -1, dtype=int)
    for i in range(-SENSOR_RANGE, SENSOR_RANGE + 1):
        for j in range(-SENSOR_RANGE, SENSOR_RANGE + 1):
            grid_x, grid_y = x + i, y + j
            view_i, view_j = i + SENSOR_RANGE, j + SENSOR_RANGE
            if 0 <= grid_x < GRID_SIZE and 0 <= grid_y < GRID_SIZE:
                view[view_i, view_j] = self.discovered_map[grid_x, grid_y]
    return view
```

متد get_perception_view

این متد نمای ادراکی مأمور را در یک محدوده ۵×۵ (بر اساس SENSOR_RANGE ارائه می‌دهد. عملکردهای اصلی آن عبارت‌اند از:

ایجاد شبکه نمای ادراکی :یک شبکه ۵×۵ ایجاد می‌شود که در ابتدا تمام خانه‌های آن به مقدار -1 (ناشناخته) مقداردهی می‌شوند.

پر کردن شبکه :برای هر خانه در محدوده حسگر (از -SENSOR_RANGE تا +SENSOR_RANGE نسبت به موقعیت مأمور)، اگر مختصات در محدوده شبکه محیط (GRID_SIZE) باشد، محتوای آن خانه از self.discovered_map خوانده شده و در شبکه نمای ادراکی ثبت می‌شود.

بازگرداندن نما :شبکه ۵×۵ به‌عنوان خروجی برگردانده می‌شود.

این متد به مأمور امکان می‌دهد تا دید محدودی از محیط اطراف خود داشته باشد که برای تصمیم‌گیری در مورد اقدامات بعدی استفاده می‌شود.

```python
def get_nearest_station(self):
    x, y = self.pos
    distances = [abs(x - sx) + abs(y - sy) for sx, sy in self.stations]
    min_distance = min(distances)
    nearest_station = self.stations[distances.index(min_distance)]
    return nearest_station, min_distance
```

متد get_nearest_station

این متد نزدیک‌ترین ایستگاه به موقعیت فعلی مأمور را پیدا می‌کند. عملکردهای اصلی آن عبارت‌اند از:

محاسبه فاصله‌ها :برای هر ایستگاه در لیست self.stations یعنی (0, 0)و (14, 14) ، فاصله منهتن از موقعیت فعلی مأمور محاسبه می‌شود.

یافتن نزدیک‌ترین ایستگاه :ایستگاهی که کمترین فاصله را دارد انتخاب می‌شود.

بازگرداندن نتیجه :مختصات نزدیک‌ترین ایستگاه و فاصله آن به‌عنوان یک تاپل برگردانده می‌شود.

این متد به مأمور کمک می‌کند تا در صورت نیاز به تحویل غیرنظامی، کوتاه‌ترین مسیر به ایستگاه را شناسایی کند.

```
    def generate_prompt(self):
        perception_view = self.get_perception_view()
        nearest_station, distance = self.get_nearest_station()
        prompt = f"""
[AGENT STATE]
Gird size:({GRID_SIZE}, {GRID_SIZE})
Position: ({self.pos[0]}, {self.pos[1]})
Sensor View: {perception_view.tolist()}
Known Fires: {self.known_fires}
Known Civilians: {self.known_civilians}
Carrying Civilian: {'Yes' if self.carrying_civilian else 'No'}
Last Cell Content: {self.last_cell_content} (0=Empty, 2=Civilian, 3=Fire, 5=Station)
Nearest Station: {nearest_station} (Distance: {distance})
Score: {self.env.score}
Time Left: {MAX_STEPS - self.env.steps} steps

[GOAL AND PRIORITY]
Extinguish fires, rescue civilians, and avoid penalties.
```

متد generate_prompt

این متد یک درخواست (prompt) متنی تولید می‌کند که به چت‌بات (مدل زبانی) ارسال می‌شود تا تصمیم بعدی مأمور را مشخص کند. عملکردهای اصلی آن عبارت‌اند از:

جمع‌آوری اطلاعات محیطی :

نمای ادراکی مأمور با استفاده از متد get_perception_view دریافت می‌شود که یک شبکه ۵×۵ از محیط اطراف مأمور را نشان می‌دهد.

نزدیک‌ترین ایستگاه و فاصله آن با استفاده از متد get_nearest_station محاسبه می‌شود.

ساخت درخواست :یک رشته متنی ساختارمند ایجاد می‌شود که شامل بخش‌های زیر است :

وضعیت مأمور :اطلاعاتی مانند اندازه شبکه(GRID_SIZE)، موقعیت فعلی مأمور، نمای ادراکی، لیست آتش‌ها و غیرنظامیان شناخته‌شده، وضعیت حمل غیرنظامی، محتوای آخرین خانه، نزدیک‌ترین ایستگاه، امتیاز فعلی محیط و تعداد گام‌های باقی‌مانده.

هدف و اولویت :توضیح مختصری درباره هدف مأمور (خاموش کردن آتش‌ها، نجات غیرنظامیان و اجتناب از جریمه).

اقدامات ممکن :فهرستی از اقدامات معتبر شامل حرکت (بالا، پایین، چپ، راست)، خاموش کردن آتش، نجات غیرنظامی و تحویل غیرنظامی به ایستگاه، همراه با شرایط انجام هر اقدام.

فرمت پاسخ :درخواست می‌شود که چت‌بات پاسخ را در قالب JSON با دو کلید action (اقدام انتخاب‌شده) و reasoning (دلیل انتخاب اقدام) ارائه دهد.

بازگرداندن درخواست :رشته درخواست به‌عنوان خروجی برگردانده می‌شود تا به چت‌بات ارسال شود.

این متد اطلاعات لازم را به‌صورت ساختارمند به مدل زبانی ارائه می‌دهد تا تصمیم‌گیری هوشمندانه‌ای برای اقدام بعدی مأمور انجام شود.

```python
def validate_action(self, action):
    x, y = self.pos
    if action == "extinguish fire":
        return self.last_cell_content == FIRE
    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
        return (0 <= new_x < GRID_SIZE and 0 <= new_y < GRID_SIZE and
                (self.discovered_map[new_x, new_y] == -1 or self.discovered_map[new_x, new_y] != BUILDING))
    return False
```

متد validate_action

این متد بررسی می‌کند که آیا اقدام پیشنهادی (مانند حرکت یا عملیاتی مانند خاموش کردن آتش) معتبر و قابل اجرا در موقعیت فعلی مأمور است یا خیر. عملکردهای اصلی آن عبارت‌اند از:

بررسی اقدامات عملیاتی :

خاموش کردن آتش(extinguish fire) : معتبر است اگر محتوای آخرین خانه (self.last_cell_content) آتش (FIRE) باشد.

نجات غیرنظامی(rescue civilian) : معتبر است اگر محتوای آخرین خانه غیرنظامی (CIVILIAN) باشد و مأمور در حال حاضر غیرنظامی حمل نکند.

تحویل غیرنظامی(deliver civilian) : معتبر است اگر محتوای آخرین خانه ایستگاه (STATION) باشد و مأمور در حال حمل غیرنظامی باشد.

بررسی اقدامات حرکتی :

برای حرکت (بالا، پایین، چپ، راست)، مختصات جدید (new_x, new_y) بر اساس اقدام محاسبه می‌شود.

حرکت معتبر است اگر :

مختصات جدید در محدوده شبکه (GRID_SIZE) باشد.

خانه مقصد در نقشه کشف‌شده (self.discovered_map) ناشناخته باشد (-1) یا حاوی ساختمان (BUILDING) نباشد.

بازگرداندن نتیجه :اگر اقدام معتبر باشد، True و در غیر این صورت False برگردانده می‌شود.

این متد تضمین می‌کند که مأمور تنها اقداماتی را انجام دهد که با قوانین محیط و وضعیت فعلی سازگار هستند.

```
def get_fallback_action(self):
    valid_actions = []
    for action in ["move up", "move down", "move left", "move right", "extinguish fire", "rescue civilian", "deliver civilian"]:
        if self.validate_action(action):
            valid_actions.append(action)
    return random.choice(valid_actions) if valid_actions else None
```

متد get_fallback_action

این متد یک اقدام جایگزین (fallback) را انتخاب می‌کند زمانی که اقدام پیشنهادی توسط چت‌بات نامعتبر باشد یا پاسخی از چت‌بات دریافت نشود. عملکردهای اصلی آن عبارت‌اند از:

جمع‌آوری اقدامات معتبر :تمام اقدامات ممکن (حرکت به بالا، پایین، چپ، راست، خاموش کردن آتش، نجات غیرنظامی، تحویل غیرنظامی) بررسی می‌شوند و با استفاده از متد validate_action فهرستی از اقدامات معتبر ایجاد می‌شود.

انتخاب تصادفی :اگر فهرست اقدامات معتبر خالی نباشد، یک اقدام به‌صورت تصادفی از میان آن‌ها انتخاب می‌شود.

مدیریت حالت بدون اقدام :اگر هیچ اقدام معتبری وجود نداشته باشد، مقدار None برگردانده می‌شود.

این متد به‌عنوان یک مکانیزم ایمنی عمل می‌کند تا در صورت بروز مشکل در تصمیم‌گیری چت‌بات، مأمور همچنان بتواند اقدامی منطقی انجام دهد یا از توقف جلوگیری شود.

```python
async def move(self):
    self.update_discovered_map()
    prompt = self.generate_prompt()
    response_entry = {"step": self.env.steps, "prompt": prompt, "action": None, "reasoning": None, "fallback": False}
    try:
        response = await asyncio.wait_for(
            asyncio.to_thread(self.chatbot.chat, prompt),
            timeout=10
        )
        response_text = response.text if hasattr(response, 'text') else str(response)

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

        json_match = re.search(r'```json\n([\s\S]*?)\n```', response_text)
        if json_match:
            json_str = json_match.group(1)
        else:
            json_start = response_text.find('{')
            json_end = response_text.rfind('}') + 1
            if json_start != -1 and json_end != 0:
                json_str = response_text[json_start:json_end]
            else:
                raise ValueError("No valid JSON found in response")
```

متد move

این متد وظیفه اصلی مدیریت فرآیند تصمیم‌گیری و اجرای اقدامات مأمور را در هر گام از شبیه‌سازی بر عهده دارد. این متد به‌صورت ناهمگام (asynchronous) پیاده‌سازی شده است تا با چت‌بات تعامل کند و پاسخ‌های آن را پردازش نماید. عملکردهای اصلی آن عبارت‌اند از:

به‌روزرسانی نقشه کشف‌شده :ابتدا متد update_discovered_map فراخوانی می‌شود تا اطلاعات محیط اطراف مأمور به‌روز شود.

تولید درخواست :متد generate_prompt فراخوانی می‌شود تا یک درخواست متنی برای چت‌بات تولید شود که شامل اطلاعات محیطی و گزینه‌های اقدام است.

ایجاد ثبت پاسخ :یک دیکشنری برای ثبت اطلاعات گام جاری (شماره گام، درخواست، اقدام، دلیل و وضعیت اقدام جایگزین) ایجاد می‌شود.

تعامل با چت‌بات :

درخواست به چت‌بات ارسال می‌شود و پاسخ در یک بازه زمانی محدود (10 ثانیه) دریافت می‌شود.

پاسخ چت‌بات پردازش می‌شود تا یک شیء JSON استخراج شود که شامل اقدام (action) و دلیل انتخاب آن (reasoning) است.

اعتبارسنجی و اجرای اقدام :

اگر پاسخ معتبر باشد و اقدام پیشنهادی با استفاده از متد validate_action تأیید شود، اقدام با متد execute_action اجرا می‌شود.

اگر پاسخ نامعتبر باشد مثلاً JSON نادرست، پاسخ خالی یا اقدام غیرمجاز متد get_fallback_action فراخوانی می‌شود تا یک اقدام جایگزین انتخاب شود.

در صورت عدم وجود اقدام جایگزین، هیچ اقدامی انجام نمی‌شود و این موضوع ثبت می‌شود.

مدیریت خطاها :

خطاهایی مانند Timeout (عدم دریافت پاسخ در زمان تعیین‌شده)، خطای پردازش JSON ، یا خطاهای غیرمنتظره مدیریت می‌شوند.

در هر حالت خطا، سیستم به سراغ اقدام جایگزین می‌رود یا عدم اجرای اقدام را ثبت می‌کند.

ثبت پاسخ :اطلاعات پاسخ (اقدام، دلیل، و اینکه آیا اقدام جایگزین بوده یا خیر) در لیست self.llm_responses ذخیره می‌شود.

این متد هسته اصلی تعامل مأمور با محیط و چت‌بات را تشکیل می‌دهد و تضمین می‌کند که مأمور در هر گام اقدامی منطقی انجام دهد یا به‌طور مناسب به خطاها پاسخ دهد.

```python
def execute_action(self, action):
    x, y = self.pos
    if action == "extinguish fire":
        if self.last_cell_content == FIRE:
            success = self.env.apply_action(x, y, "extinguish", self.last_cell_content)
            if success:
                self.discovered_map[x, y] = EMPTY
                if (x, y) in self.known_fires:
                    self.known_fires.remove((x, y))
                self.last_cell_content = EMPTY
                self.env.update_cell(x, y, AGENT)  # Ensure environment grid shows AGENT
                self.discovered_map[x, y] = AGENT  # Ensure discovered map shows AGENT
    elif action == "rescue civilian":
        if self.last_cell_content == CIVILIAN:
            success = self.env.apply_action(x, y, "rescue", self.last_cell_content)
            if success:
                self.carrying_civilian = True
                self.discovered_map[x, y] = EMPTY
                if (x, y) in self.known_civilians:
                    self.known_civilians.remove((x, y))
                self.last_cell_content = EMPTY
                self.env.update_cell(x, y, AGENT)  # Ensure environment grid shows AGENT
                self.discovered_map[x, y] = AGENT  # Ensure discovered map shows AGENT
                print(f"Civilian picked up!")
```

متد execute_action

این متد وظیفه اجرای اقدام انتخاب‌شده (چه از چت‌بات و چه اقدام جایگزین) را بر عهده دارد و وضعیت مأمور و محیط را بر اساس آن به‌روزرسانی می‌کند. عملکردهای اصلی آن عبارت‌اند از:

مدیریت اقدامات عملیاتی :

خاموش کردن آتش(extinguish fire) : اگر محتوای آخرین خانه (self.last_cell_content) آتش باشد، متد apply_action از کلاس Environment فراخوانی می‌شود تا آتش خاموش شود. در صورت موفقیت :

نقشه کشف‌شده و شبکه محیط به حالت خالی (EMPTY) به‌روزرسانی می‌شوند.

موقعیت آتش از لیست self.known_fires حذف می‌شود.

محتوای آخرین خانه به EMPTY تغییر می‌کند.

مأمور در شبکه محیط و نقشه کشف‌شده به‌عنوان AGENT ثبت می‌شود.

نجات غیرنظامی :(rescue civilian) اگر محتوای آخرین خانه غیرنظامی باشد، متد apply_action فراخوانی می‌شود. در صورت موفقیت :

وضعیت self.carrying_civilian به True تغییر می‌کند.

نقشه کشف‌شده و شبکه محیط به حالت خالی به‌روزرسانی می‌شوند.

موقعیت غیرنظامی از لیست self.known_civilians حذف می‌شود.

محتوای آخرین خانه به EMPTY تغییر می‌کند.

مأمور در شبکه و نقشه به‌عنوان AGENT ثبت می‌شود.

پیامی مبنی بر برداشتن غیرنظامی چاپ می‌شود.

تحویل غیرنظامی(deliver civilian) : اگر محتوای آخرین خانه ایستگاه باشد، متد apply_action فراخوانی می‌شود. در صورت موفقیت :

وضعیت self.carrying_civilian به False تغییر می‌کند.

شبکه محیط و نقشه کشف‌شده به حالت ایستگاه (STATION) به‌روزرسانی می‌شوند تا ایستگاه حفظ شود.

مدیریت اقدامات حرکتی :

برای حرکت (بالا، پایین، چپ، راست)، مختصات جدید (new_x, new_y) بر اساس اقدام محاسبه می‌شود.

محتوای خانه مقصد به‌عنوان self.last_cell_content ذخیره می‌شود.

خانه فعلی مأمور در شبکه محیط به حالت خالی (EMPTY) تغییر می‌کند.

موقعیت مأمور (self.pos) به مختصات جدید به‌روزرسانی می‌شود.

خانه جدید در شبکه محیط به‌عنوان حاوی مأمور (AGENT) علامت‌گذاری می‌شود.

متد update_discovered_map فراخوانی می‌شود تا نقشه کشف‌شده به‌روز شود.

این متد اطمینان می‌دهد که هر اقدام به‌درستی در محیط اعمال شده و وضعیت مأمور و محیط به‌صورت هماهنگ به‌روزرسانی شود.

```python
def display_maps(env, agent, step):
    fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(14, 5))

    ax1.imshow(env.get_grid(), cmap=FULL_CMAP, interpolation='nearest', vmin=0, vmax=5)
    ax1.set_title(f"Full City Map (Step {step})")
    ax1.axis('off')

    discovered_display = agent.discovered_map
    ax2.imshow(discovered_display, cmap=DISCOVERED_CMAP, interpolation='nearest', vmin=-1, vmax=5)
    ax2.set_title("Agent's Discovered Map")
    ax2.axis('off')

    from matplotlib.patches import Patch
    legend_elements = [Patch(facecolor=color, label=label) for label, color in COLOR_LEGEND.items()]
    fig.legend(handles=legend_elements, loc='upper center', ncol=4, bbox_to_anchor=(0.5, -0.05), title="Color Key")

    plt.tight_layout()

    if sys.platform != "emscripten":
        try:
            plt.savefig(f'map_step_{step}.png', bbox_inches='tight')
        except Exception as e:
            print(f"Could not save plot for step {step}: {e}")

    plt.show()
```

تابع display_maps

این تابع وظیفه نمایش بصری نقشه کامل شهر (محیط شبیه‌سازی) و نقشه کشف‌شده توسط مأمور را در هر گام از شبیه‌سازی بر عهده دارد. عملکردهای اصلی آن عبارت‌اند از:

ایجاد پنجره نمایش :با استفاده از کتابخانه matplotlib، یک پنجره با دو زیرپنجره (subplot) ایجاد می‌شود که ابعاد کلی آن ۱۴×۵ است.

نمایش نقشه کامل شهر :

شبکه محیط (env.get_grid()) با استفاده از متد imshow به‌صورت تصویری نمایش داده می‌شود.

از یک نقشه رنگی (FULL_CMAP) برای نمایش عناصر مختلف (مانند آتش، غیرنظامی، ایستگاه) استفاده می‌شود.

عنوان زیرپنجره شامل شماره گام فعلی (Step {step}) است.

محورهای مختصات خاموش می‌شوند تا نمایش ساده‌تر باشد.

نمایش نقشه کشف‌شده مأمور :

نقشه کشف‌شده مأمور (agent.discovered_map) نیز با استفاده از imshow و نقشه رنگی متفاوت (DISCOVERED_CMAP) نمایش داده می‌شود.

عنوان این زیرپنجره «نقشه کشف‌شده مأمور» است.

محورهای مختصات خاموش می‌شوند.

ایجاد راهنمای رنگی :یک راهنما (legend) در پایین پنجره ایجاد می‌شود که رنگ‌های مختلف و معانی آن‌ها بر اساس COLOR_LEGEND را نشان می‌دهد.

تنظیم چیدمان :با استفاده از tight_layout، چیدمان پنجره به‌گونه‌ای تنظیم می‌شود که عناصر به‌خوبی نمایش داده شوند.

ذخیره تصویر :

اگر پلتفرم در حال اجرا emscripten نباشد، تصویر نقشه با نام map_step_{step}.png ذخیره می‌شود.

در صورت بروز خطا هنگام ذخیره، پیام خطا چاپ می‌شود.

نمایش تصویر :پنجره نهایی با استفاده از plt.show() نمایش داده می‌شود.

این تابع به کاربر امکان می‌دهد تا به‌صورت بصری وضعیت محیط و دانش مأمور از محیط را در هر گام مشاهده کند.

```python
async def run_simulation():
    EMAIL = "Haghighat.Mohammad.2020@gmail.com"
    PASSWORD = "Aa@12345678@Zz"

    env = Environment()
    agent = LLMAgent(env, EMAIL, PASSWORD)

    while not env.is_game_over():
        await agent.move()
        env.increment_step()

        if env.steps % 5 == 0:
            env.spread_fire()

        display_maps(env, agent, env.steps)
        await asyncio.sleep(0.5)

        if env.fires_extinguished == NUM_FIRES and env.civilians_rescued == NUM_CIVILIANS:
            print(f"Success! All fires extinguished and civilians rescued in {env.steps} steps.")
            break
```

تابع run_simulation

این تابع ناهمگام (asynchronous) وظیفه اجرای کل فرآیند شبیه‌سازی را بر عهده دارد و اجزای مختلف محیط و مأمور را هماهنگ می‌کند. عملکردهای اصلی آن عبارت‌اند از:

مقداردهی اولیه :

ایمیل و رمز عبور برای ورود به چت‌بات تعریف می‌شوند.

یک نمونه از کلاس Environment برای ایجاد محیط شبیه‌سازی ساخته می‌شود.

یک نمونه از کلاس LLMAgent با استفاده از محیط و اطلاعات ورود ایجاد می‌شود.

حلقه اصلی شبیه‌سازی :

تا زمانی که بازی پایان نیابد (env.is_game_over() برابر Falseباشد حلقه ادامه می‌یابد.

در هر گام :

متد moveمأمور به‌صورت ناهمگام فراخوانی می‌شود تا مأمور اقدامی انجام دهد.

تعداد گام‌ها با env.increment_step افزایش می‌یابد.

هر ۵ گام، متد env.spread_fire فراخوانی می‌شود تا آتش در محیط گسترش یابد.

تابع display_maps فراخوانی می‌شود تا نقشه‌ها نمایش داده شوند.

یک تأخیر ۰.۵ ثانیه‌ای (asyncio.sleep) برای مشاهده بهتر شبیه‌سازی اعمال می‌شود.

بررسی موفقیت :

اگر تمام آتش‌ها خاموش شوند (env.fires_extinguished == NUM_FIRES) و تمام غیرنظامیان نجات یابند(env.civilians_rescued == NUM_CIVILIANS) ، پیام موفقیت همراه با تعداد گام‌های صرف‌شده چاپ می‌شود و حلقه پایان می‌یابد.

پایان بازی :

اگر تعداد گام‌ها به حداکثر (MAX_STEPS) برسد، بازی پایان می‌یابد و اطلاعاتی مانند امتیاز نهایی، تعداد آتش‌های خاموش‌شده و تعداد غیرنظامیان نجات‌یافته چاپ می‌شود.

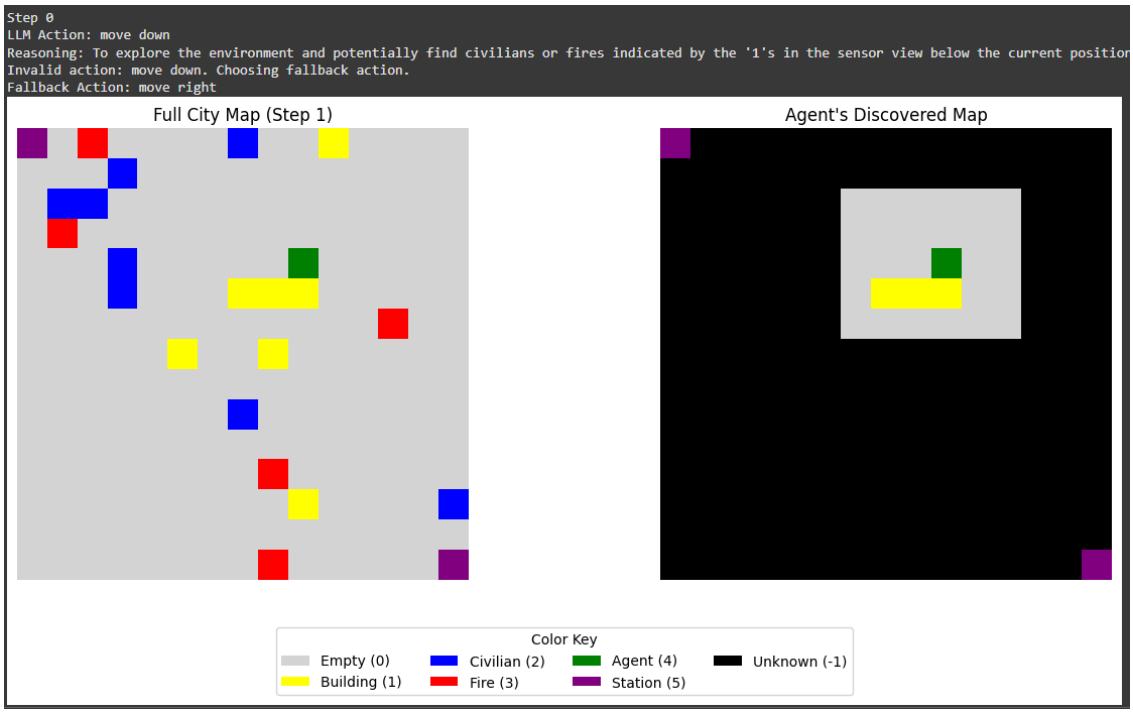
نمایش پاسخ‌های چت‌بات :

در پایان شبیه‌سازی، تمام پاسخ‌های چت‌بات ذخیره‌شده در agent.llm_responses چاپ می‌شوند.
برای هر گام :

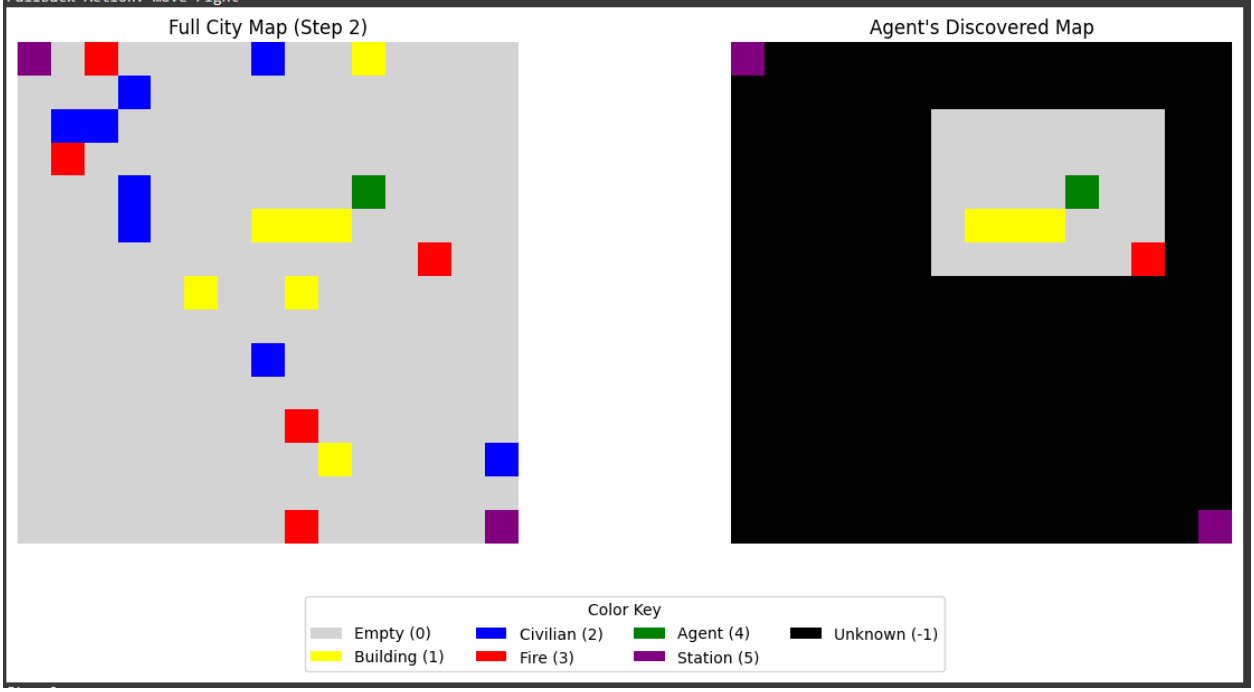
شماره گام، درخواست ارسالی، اقدام انتخاب‌شده، دلیل اقدام و اینکه آیا اقدام جایگزین بوده یا خیر نمایش داده می‌شود.

این تابع کل شبیه‌سازی را مدیریت می‌کند و اطمینان می‌دهد که محیط، مأمور و نمایش بصری به‌درستی با یکدیگر هماهنگ شوند.

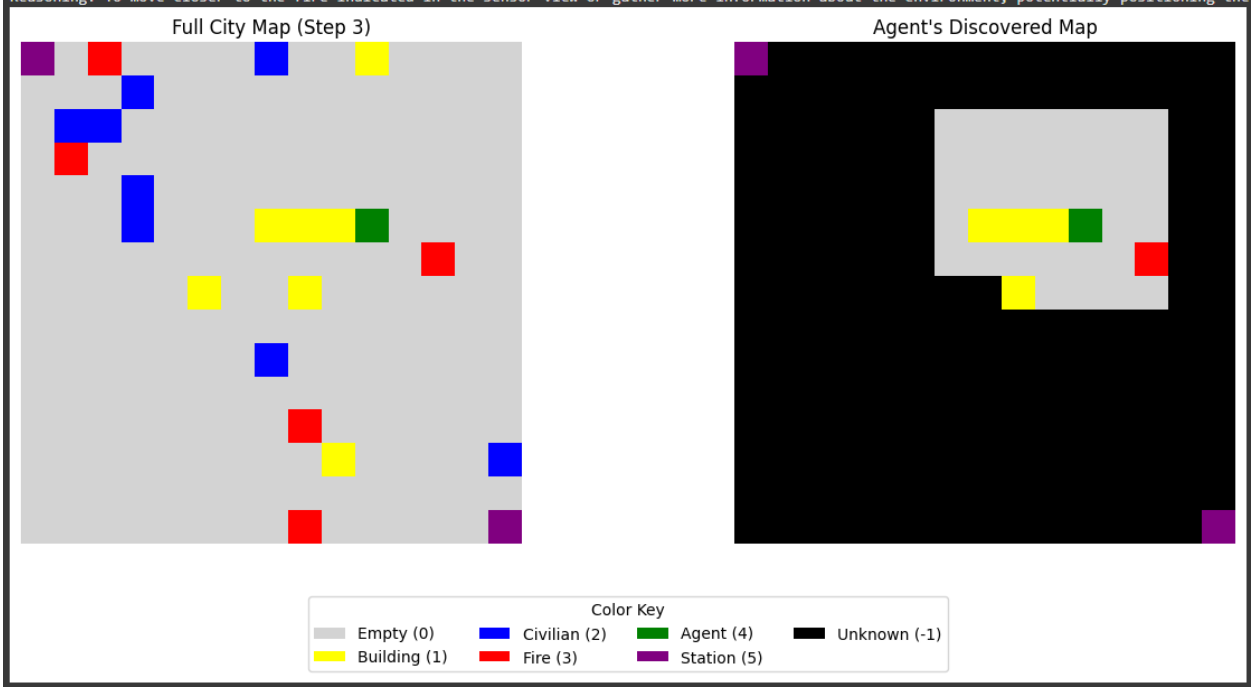
بخشی از خروجی :



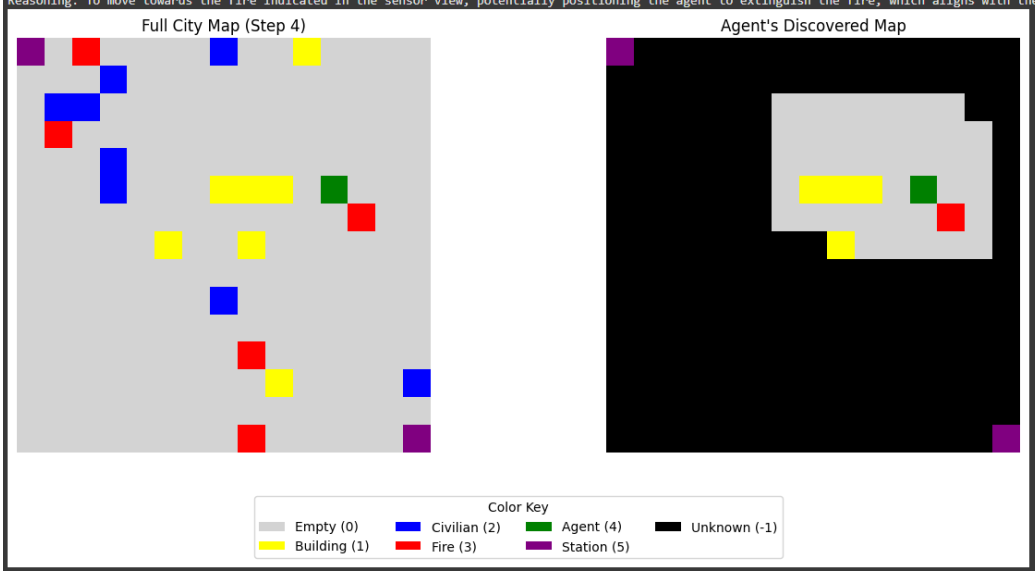
```
Step 1
LLM Action: move down
Reasoning: To continue exploring downwards and investigate the '1's in the sensor view, which could indicate the presence of civilians or fires.
Invalid action: move down. Choosing fallback action.
Fallback Action: move right
```



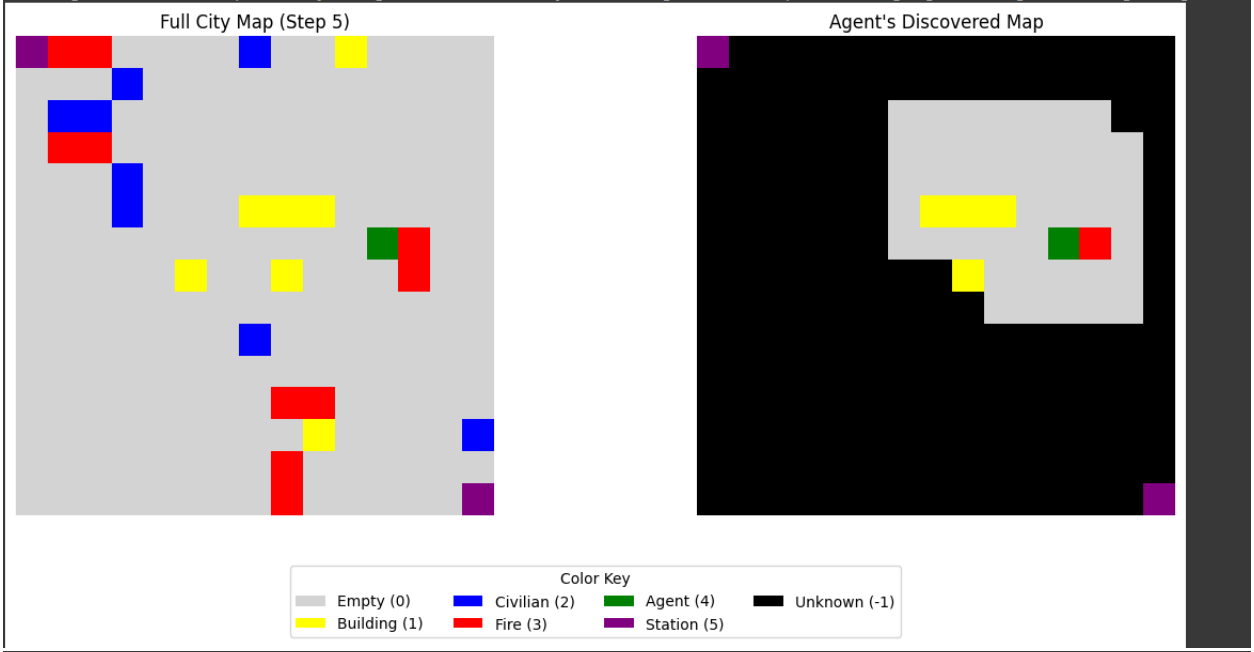

```
Step 2
LLM Action: move down
Reasoning: To move closer to the fire indicated in the sensor view or gather more information about the environment, potentially positioning the
```



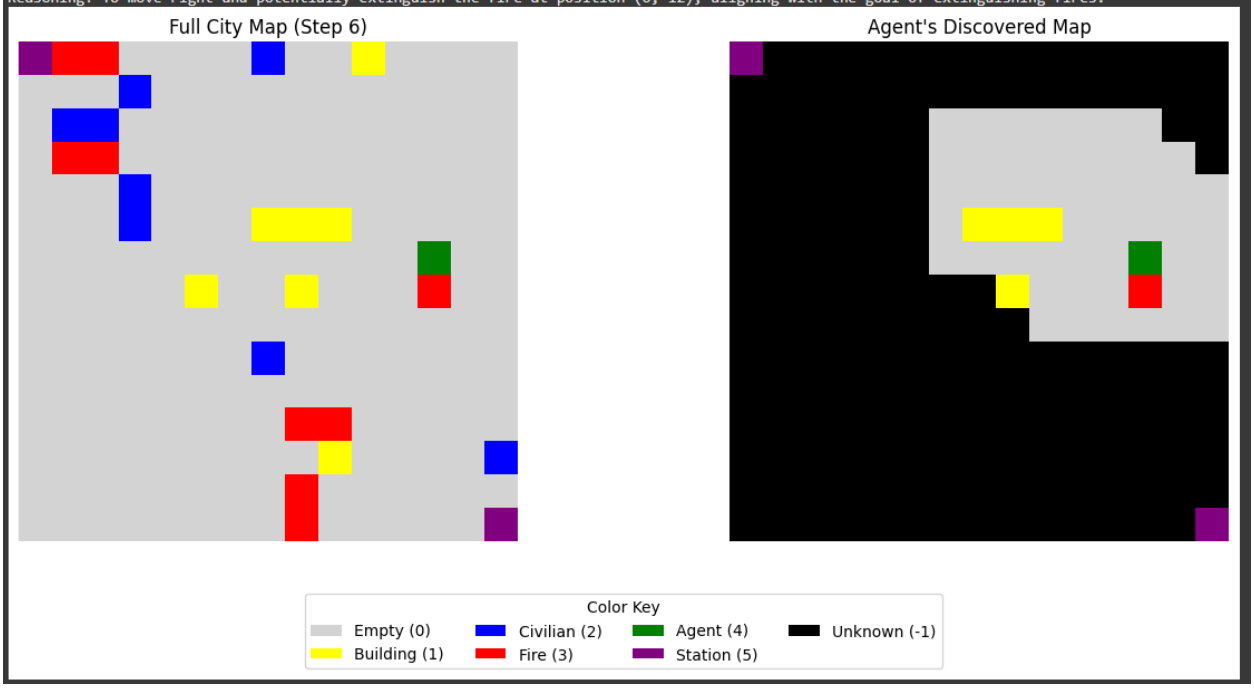
Step 3
LLM Action: move right
Reasoning: To move towards the fire indicated in the sensor view, potentially positioning the agent to extinguish the fire, which aligns with the goal of extinguishing fires.



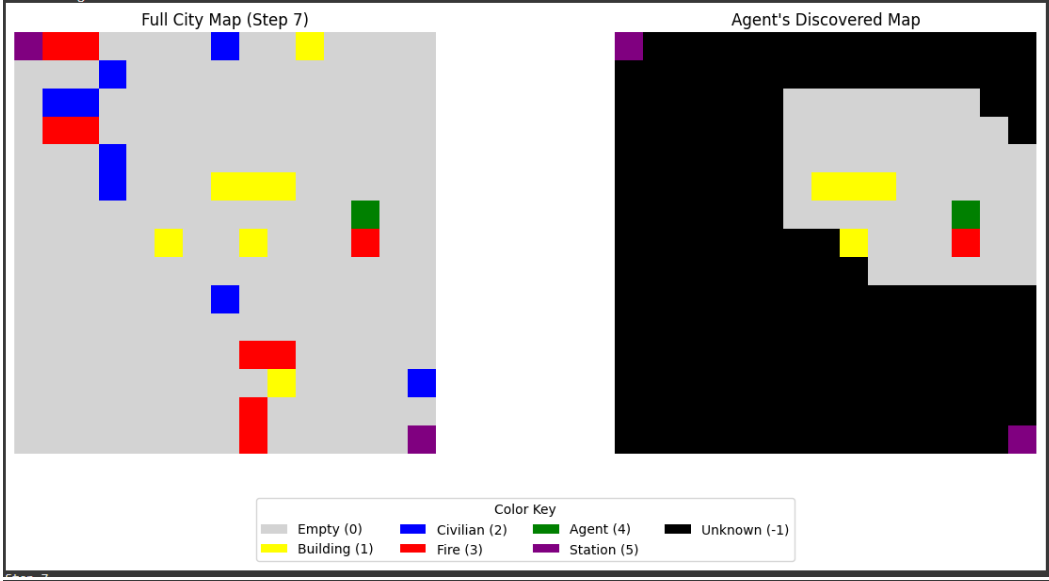
Step 4
LLM Action: move down
Reasoning: To move down and potentially extinguish the fire directly below the agent's current position, aligning with the goal of extinguishing fires.



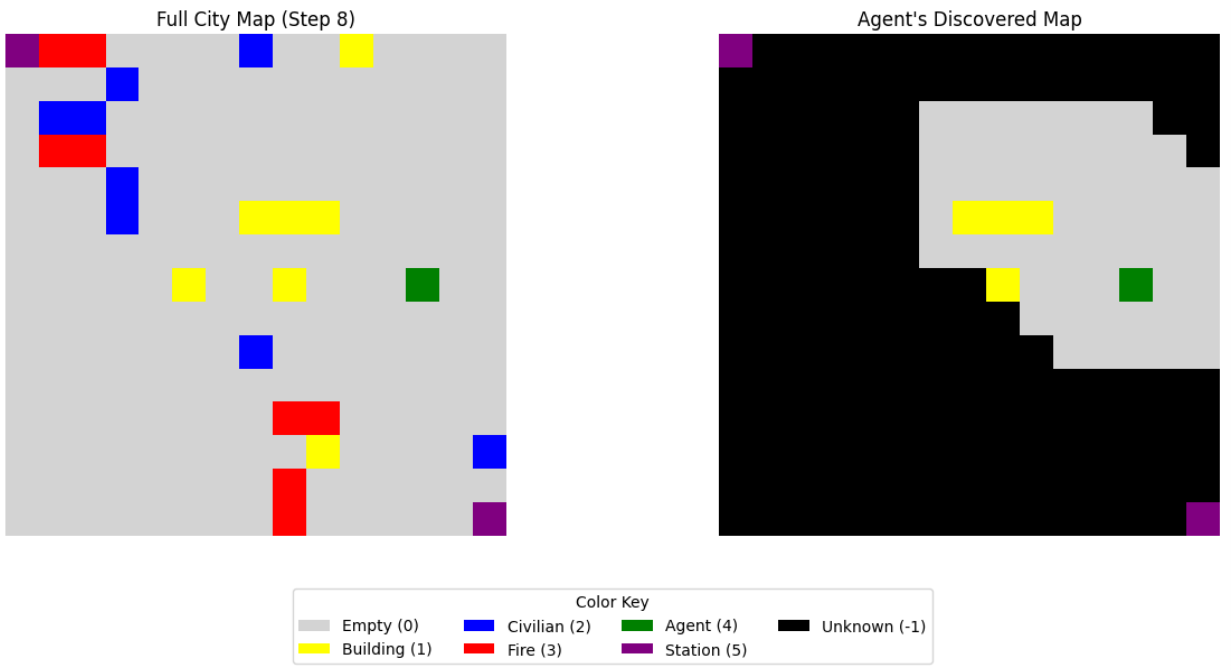
Step 5
LLM Action: move right
Reasoning: To move right and potentially extinguish the fire at position (6, 12), aligning with the goal of extinguishing fires.



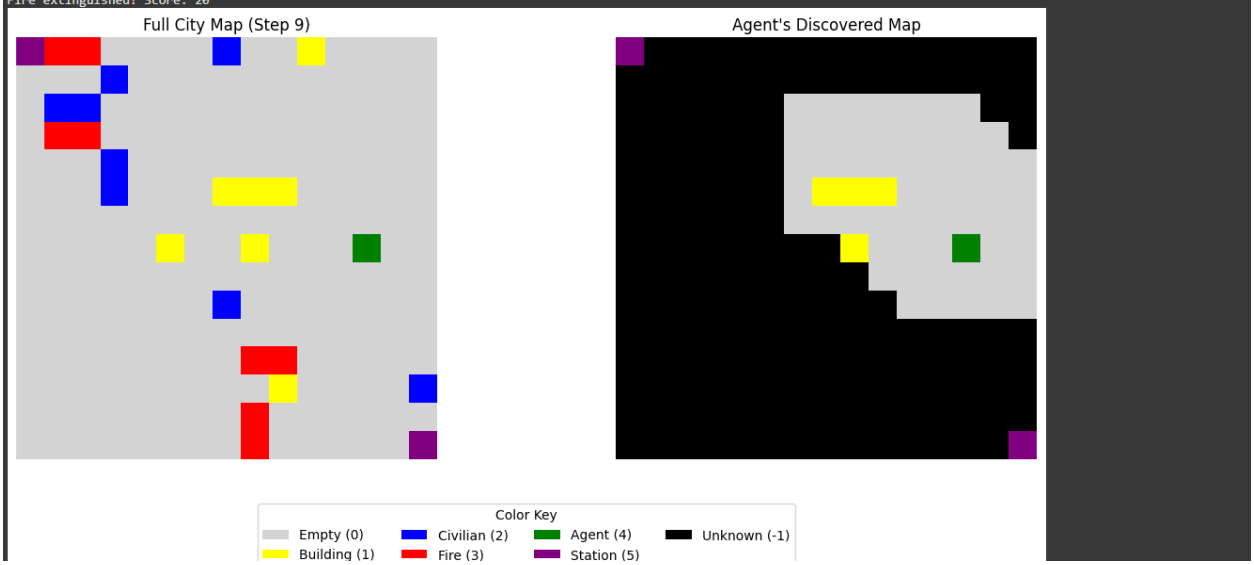
Step 6
LLM Action: extinguish fire
Reasoning: To extinguish the fire at the current position (6, 12), as the last cell content was a fire and this action directly aligns with the goal of extinguishing fires.
Fire extinguished! Score: 10



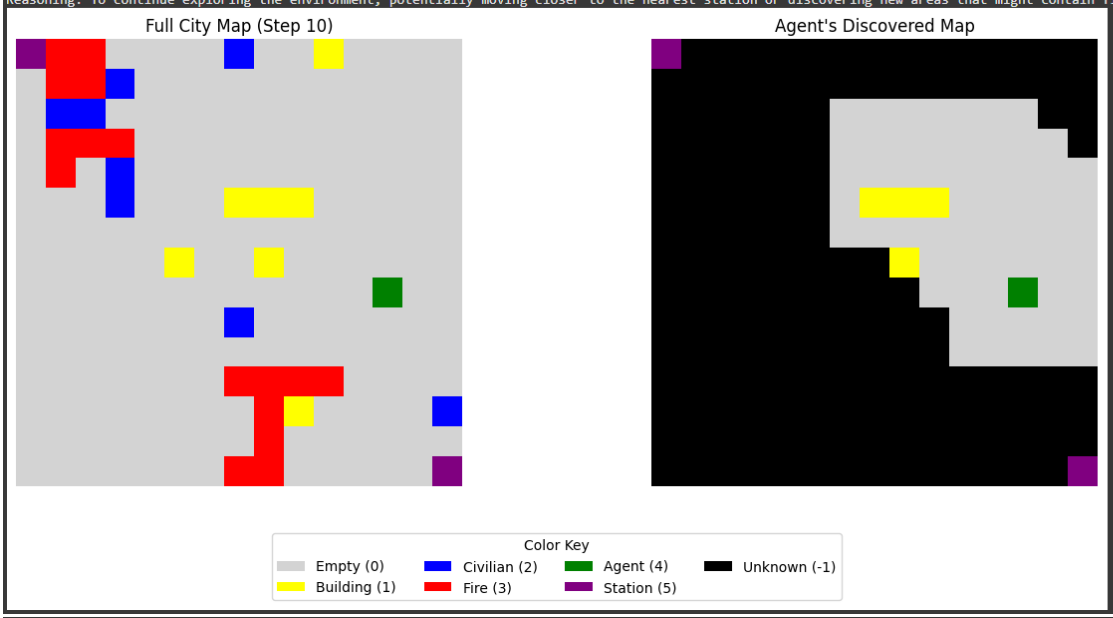
Step 7
LLM Action: move down
Reasoning: To move down towards the known fire at position (7, 12), aligning with the goal of extinguishing fires.



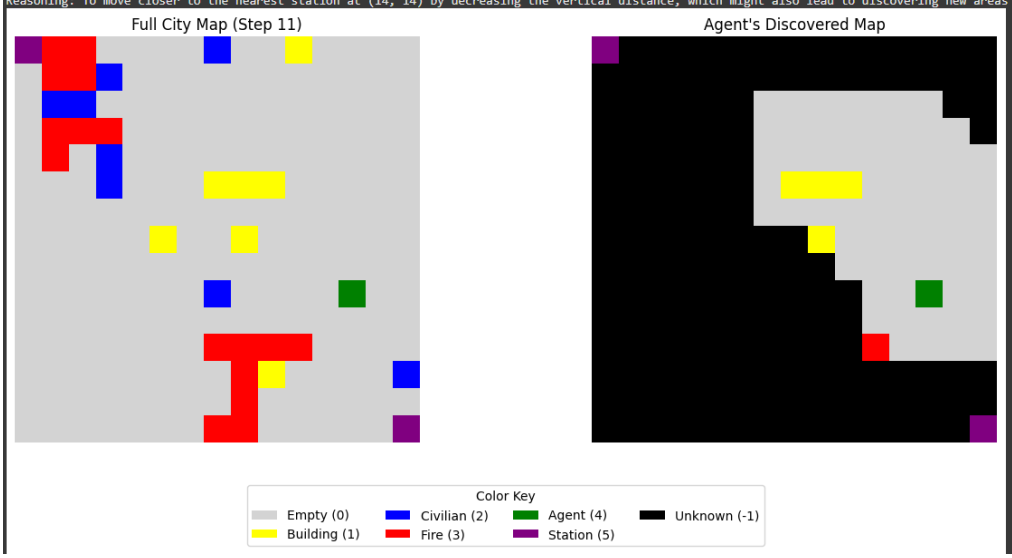
```
Step 8
LLM Action: extinguish fire
Reasoning: To extinguish the fire at the current position (7, 12), as the last cell content was a fire and this action directly aligns with the goal of extinguishing fires.
Fire extinguished! Score: 20
```



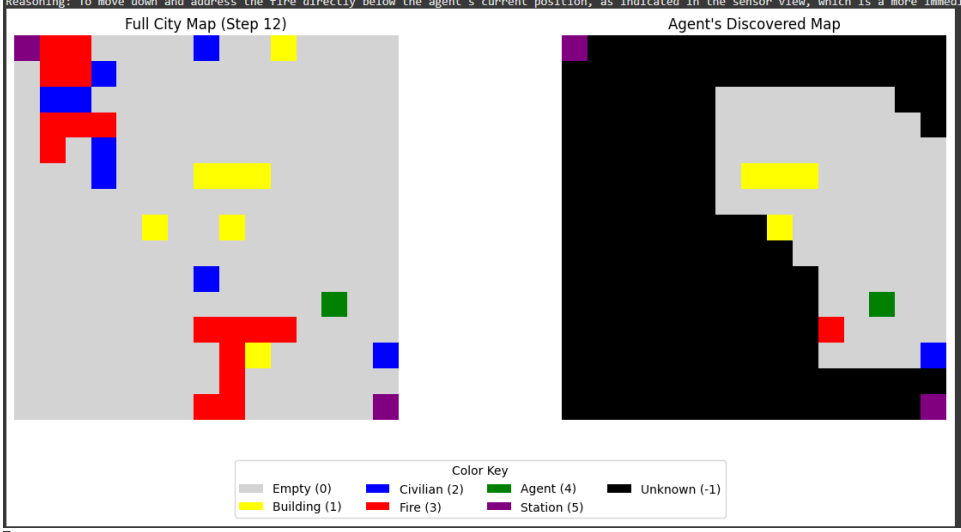
Step 9
LLM Action: move down
Reasoning: To continue exploring the environment, potentially moving closer to the nearest station or discovering new areas that might contain fires or civilians.



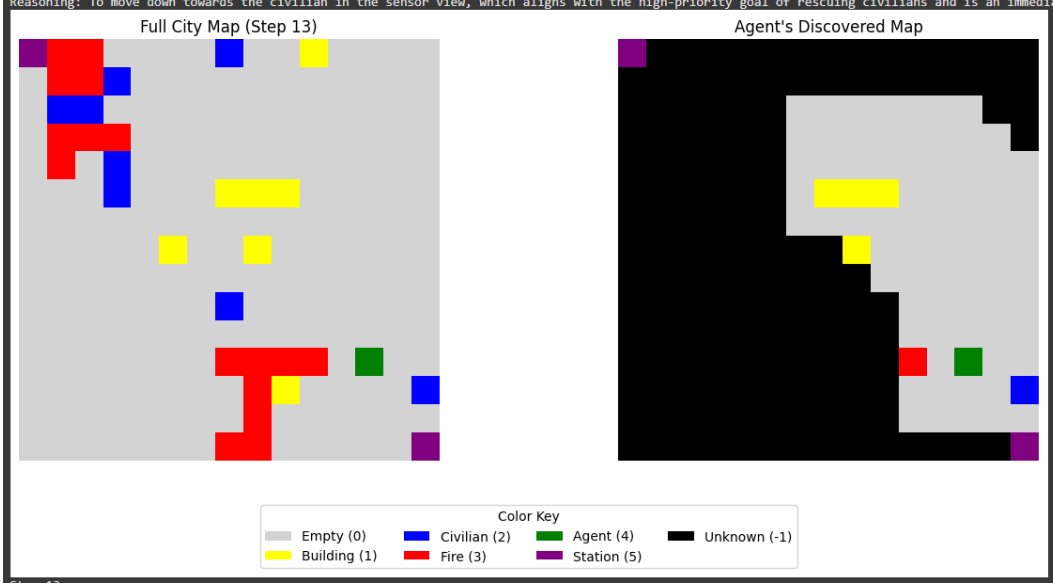
Step 10
LLM Action: move down
Reasoning: To move closer to the nearest station at (14, 14) by decreasing the vertical distance, which might also lead to discovering new areas or potential fires and civilians.



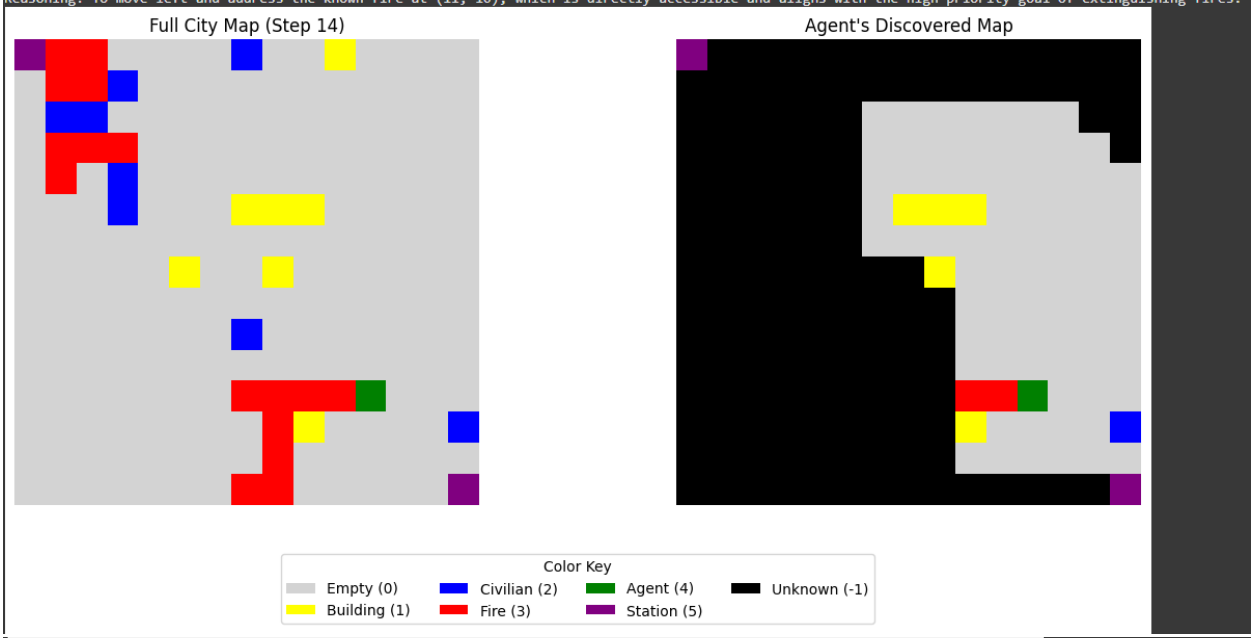
Step 11
LLM Action: move down
Reasoning: To move down and address the fire directly below the agent's current position, as indicated in the sensor view, which is a more immediate concern than the known fire at (11, 10).



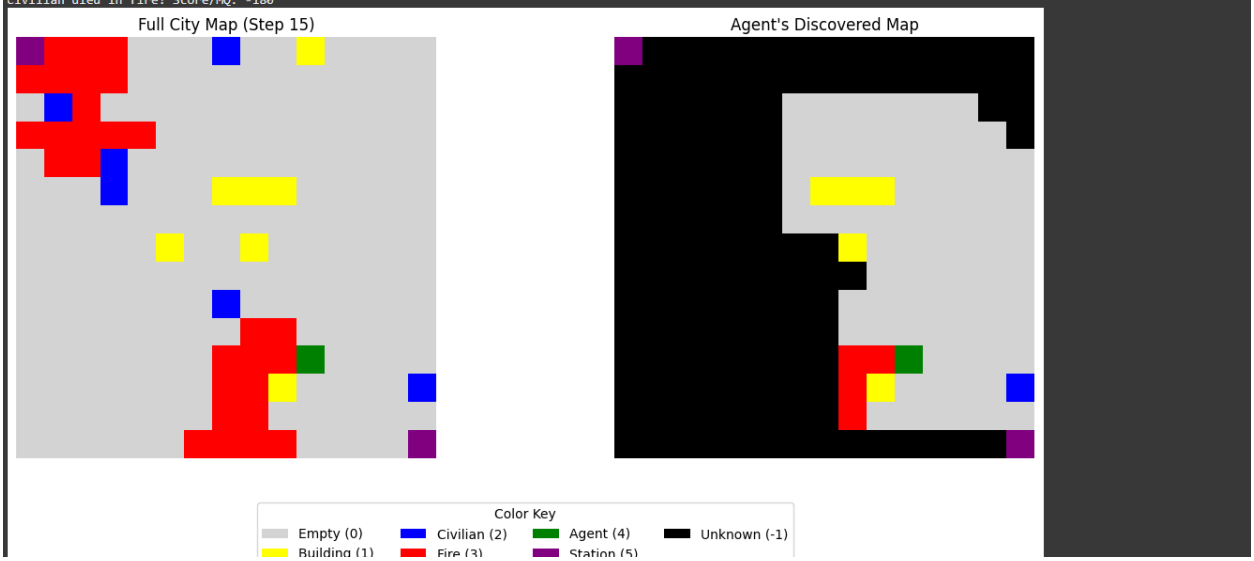
Step 12
LLM Action: move down
Reasoning: To move down towards the civilian in the sensor view, which aligns with the high-priority goal of rescuing civilians and is an immediate opportunity for action.

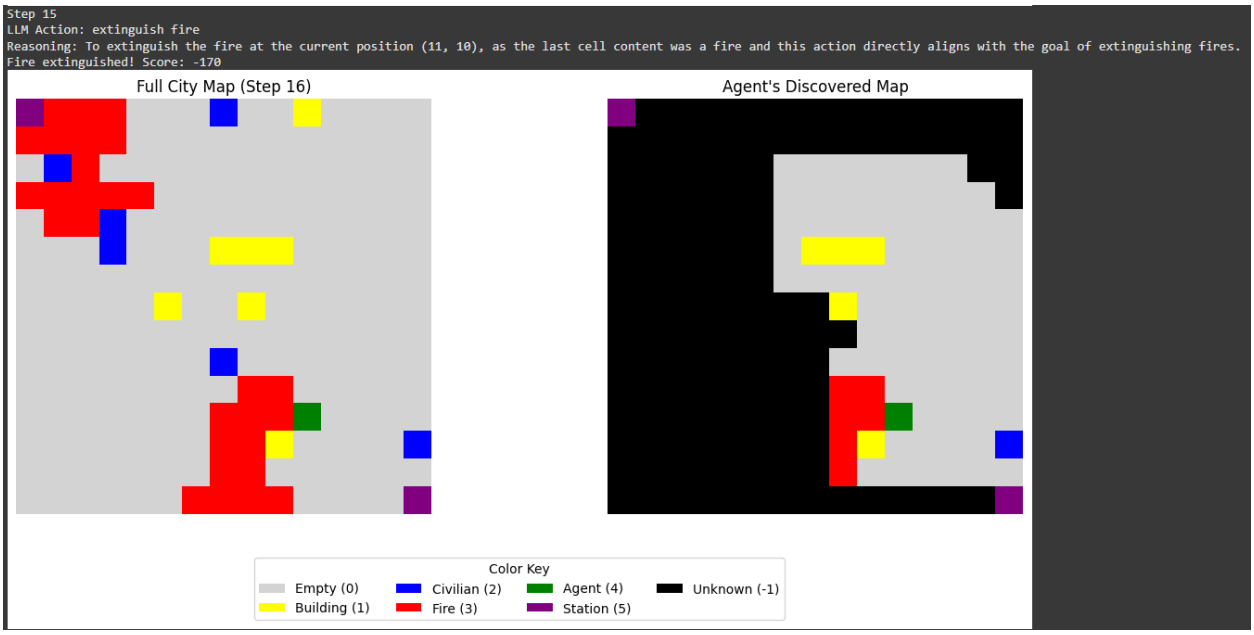


Step 13
LLM Action: move left
Reasoning: To move left and address the known fire at (11, 10), which is directly accessible and aligns with the high-priority goal of extinguishing fires.



Step 14
LLM Action: move left
Reasoning: To move left and position the agent to extinguish the fire at (11, 10), which is directly adjacent and aligns with the high-priority goal of extinguishing fires.
Civilian died in fire! Score/MQ: -80
Civilian died in fire! Score/MQ: -180





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