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 1404-1403

```
# Constants
GRID_SIZE = 15
NUM_BUILDINGS = 7
NUM_CIVILIANS = 8
NUM_FIRES = 5
MAX_STEPS = 50
SENSOR_RANGE = 2  # For 5x5 sensor area centered on agent
```

این ثابت ها قوانین و ساختار اولیه بازی رو تعیین می کنن. اندازه شبکه و تعداد موجودیت ها تعادل خوبی بین پیچیدگی و سادگی ایجاد می کنه. محدوده حسگر 2 به ایجنت دید محدودی میده که شبیه سازی رو واقعی تر می کنه.

GRID_SIZE: اندازه شبکه 15 در 15 هست، یعنی 225 سلول.

NUM_BUILDINGS: تعداد ساختمان ها (7).

NUM_CIVILIANS: تعداد غیرنظامی ها (8).

NUM_FIRES: تعداد آتش سوزی ها (5).

MAX_STEPS: حداکثر تعداد قدم ها قبل از پایان بازی (50).

SENSOR_RANGE: محدوده حسگر ایجنت (2)، یعنی ایجنت می تونه یه مربع 5 در 5 اطراف خودش رو ببینه (2 سلول در هر جهت + خودش).

```
# Cell types
EMPTY = 0
BUILDING = 1
CIVILIAN = 2
FIRE = 3
AGENT = 4
STATION = 5
```

انواع سلول ها: اعداد ثابت برای نشان دادن نوع هر سلول در شبکه:

EMPTY: خالی

BUILDING: ساختمان

CIVILIAN: غیرنظامی

FIRE: آتش

AGENT: ایجنت

STATION: ایستگاه

تعریف رنگ ها و نقشه های رنگی

```python
# Custom color mapping
COLORS = {
    EMPTY: "lightgray",  # 0
    BUILDING: "brown",   # 1
    CIVILIAN: "blue",    # 2
    FIRE: "red",         # 3
    AGENT: "green",      # 4
    STATION: "purple"    # 5
}
DISCOVERED_COLORS = COLORS.copy()
DISCOVERED_COLORS[-1] = "black"  # Unknown for agent's map

# Create custom colormaps
FULL_CMAP = ListedColormap([COLORS[i] for i in range(6)])  # 0 to 5
DISCOVERED_CMAP = ListedColormap([DISCOVERED_COLORS[i] for i in sorted(DISCOVERED_COLORS.keys())])

# Legend labels
COLOR_LEGEND = {
    "Empty (0)": COLORS[EMPTY],
    "Building (1)": COLORS[BUILDING],
    "Civilian (2)": COLORS[CIVILIAN],
    "Fire (3)": COLORS[FIRE],
    "Agent (4)": COLORS[AGENT],
    "Station (5)": COLORS[STATION],
    "Unknown (-1)": DISCOVERED_COLORS[-1]
}
```

COLORS: دیکشنری ای که به هر نوع سلول یه رنگ اختصاص میده.

DISCOVERED_COLORS: کپی از COLORS که یه رنگ اضافه برای سلول های ناشناخته (-1) داره (سیاه).

FULL_CMAP: نقشه رنگی برای نمایش کل محیط (فقط 0 تا 5).

DISCOVERED_CMAP: نقشه رنگی برای نقشه کشف شده ایجنت (شامل -1 تا 5).

COLOR_LEGEND: راهنمای رنگ ها برای نمایش در نمودار.

این بخش برای نمایش بصری خیلی مهمه. استفاده از رنگ های متمایز باعث می شه کاربر به راحتی بتونه موجودیت ها رو تشخیص بده. اضافه کردن -1 برای ناشناخته ها نشون دهنده دید محدود ایجنته.

کلاس Environment

```
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()
```

این کلاس هسته شبیه سازی رو تشکیل میده و محیط رو آماده می کنه. متغیرهای شمارشگر نشون میدن که هدف بازی، نجات غیرنظامی ها و خاموش کردن آتش است.

grid: یه آرایه 15 * 15 پر از صفر (خالی) می سازه.

متغیرهای شمارشگر برای غیرنظامی های نجات یافته، آتش های خاموش شده، امتیاز و قدم ها تعریف می کنه.

تابع setup_environment رو فراخوانی می کنه.

```
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)
```

دو ایستگاه در گوشه های بالا-چپ و پایین-راست قرار می گیرن.

به ترتیب ساختمان ها، غیرنظامی ها و آتش ها به صورت تصادفی در سلول های خالی قرار داده می شن.

قرار دادن تصادفی موجودیت ها باعث می شه هر بار بازی متفاوت باشه، که تنوع و قابلیت تکرار رو بالا می بره.

```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
```

این تابع یه موجودیت (مثل ساختمان یا آتش) رو در یه مکان تصادفی خالی قرار میده.

حلقه while تضمین می کنه که موجودیت ها روی هم قرار نگیرن، که منطقی و ضروریه.

```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: {self.score}")
                self.grid[new_x, new_y] = FIRE
                break
```

آتش به صورت تصادفی به یکی از چهار جهت (بالا، پایین، چپ، راست) پخش می شه.

اگه به غیرنظامی برسه، امتیاز 100- می شه و غیرنظامی از بین می ره.

آتش فقط به سلول های مجاز (نه ساختمان، ایستگاه یا ایجنت) پخش می شه.

این مکانیزم پویایی و چالش به بازی اضافه می کنه. جریمه سنگین (-100) برای مرگ غیرنظامی نشون دهنده اولویت نجات اون هاست.

```python
def apply_action(self, x, y, action):
    cell = self.grid[x, y]
    if action == "extinguish" and cell == FIRE:
        self.update_cell(x, y, EMPTY)
        self.fires_extinguished += 1
        self.score += 10
        print(f"Fire extinguished! Score: {self.score}")
        return True
    elif action == "rescue" and cell == CIVILIAN:
        self.update_cell(x, y, EMPTY)
        return True
    elif action == "drop" and cell == STATION:
        self.civilians_rescued += 1
        self.score += 50
        print(f"Civilian rescued! Score: {self.score}")
        return True
    return False
```

سه نوع عمل:

extinguish: آتش رو خاموش می کنه (+10 امتیاز).

rescue: غیرنظامی رو برمی داره (بدون امتیاز فوری).

drop: غیرنظامی رو در ایستگاه تحویل میده (+50 امتیاز).

کلاس Agent

```python
class Agent:
    def __init__(self, environment):
        self.env = environment
        self.pos = None
        self.discovered_map = np.full((GRID_SIZE, GRID_SIZE), -1)
        self.carrying_civilian = False  # Agent can carry only 1 civilian at a time
        self.place_agent()
        self.update_discovered_map()
```

این بخش ایجنت رو آماده می کنه تا تو محیط کار کنه. نقشه کشف شده و محدودیت حمل یه غیرنظامی، دید و توانایی ایجنت رو محدود می کنه و بازی رو واقعی تر می کنه.

پارامتر ورودی:

environment: یه نمونه از کلاس Environment که ایجنت قراره توش عمل کنه.

متغیرهای نمونه (Instance Variables):

self.env: محیطی که ایجنت بهش وصل می شه و باهاش تعامل داره.

self.pos: موقعیت فعلی ایجنت در شبکه (به صورت [x, y]) که ابتدا None هست و بعداً مقداردهی می شه.

self.discovered_map: یه آرایه 15*15 که دید ایجنت رو از محیط نشون میده. در ابتدا همه سلول ها -1 (ناشناخته) هستن و فقط چیزایی که ایجنت می بینه به روز می شن.

self.carrying_civilian: یه متغیر بولین که نشون میده آیا ایجنت الان یه غیرنظامی رو حمل می کنه یا نه (در ابتدا False).

فراخوانی توابع:

self.place_agent(): ایجنت رو در یه مکان تصادفی خالی در شبکه قرار میده.

self.update_discovered_map(): نقشه کشف شده ایجنت رو با توجه به موقعیت اولیه ش به روز می کنه.

```
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]
            break
    self.discovered_map[0, 0] = STATION
    self.discovered_map[14, 14] = STATION
```

این تابع تضمین می کنه که ایجنت در شروع بازی تو یه جای خالی قرار بگیره و از موقعیت ایستگاه ها آگاه باشه. انتخاب تصادفی موقعیت باعث تنوع در شروع هر بازی می شه.

قرار دادن ایجنت:

یه حلقه while اجرا می شه که به صورت تصادفی مختصات x و y بین 0 تا 14 (اندازه شبکه منهای 1) انتخاب می کنه.

اگه سلول انتخاب شده خالی (EMPTY) باشه:

با self.env.update_cell(x, y, AGENT) اون سلول به AGENT (عدد 4) تغییر می کنه.

موقعیت ایجنت (self.pos) به [x, y] تنظیم می شه.

تنظیم اولیه نقشه کشف شده:

دو ایستگاه در مختصات [0, 0] و [14, 14] از قبل معلومن، پس این سلول ها در discovered_map به STATION (عدد 5) تغییر می کنن

```
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)):
            self.discovered_map[i, j] = self.env.get_grid()[i, j]
```

این تابع دید ایجنت رو شبیه سازی می کنه. محدوده حسگر 2 به ایجنت یه دید محلی میده، نه دید کامل، که باعث می شه بازی استراتژیک تر بشه.

موقعیت فعلی:

x, y = self.pos: مختصات فعلی ایجنت رو می گیره.

محدوده حسگر:

SENSOR_RANGE که 2 هست، یعنی ایجنت می تونه یه مربع 5*5 (از x-2 تا x+2 و y-2 تا y+2) رو ببینه.

max(0, ...) و min(GRID_SIZE, ...) مطمئن می شن که حلقه فقط تو محدوده شبکه (0 تا 14) اجرا بشه و از مرزها بیرون نزنه.

به روزرسانی نقشه:

برای هر سلول در محدوده حسگر، مقدار اون سلول از شبکه اصلی (self.env.get_grid()) به discovered_map کپی می شه.

```python
def move(self):
    directions = [(0, 1), (0, -1), (1, 0), (-1, 0)]
    dx, dy = random.choice(directions)
    new_x = self.pos[0] + dx
    new_y = self.pos[1] + dy
    if (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)):
        self.env.update_cell(self.pos[0], self.pos[1], EMPTY)
        self.pos = [new_x, new_y]
        self.handle_interaction()
        self.env.update_cell(new_x, new_y, AGENT)
        self.update_discovered_map()
```

جهت حرکت:

directions: چهار جهت ممکن (راست، چپ، پایین، بالا).

dx, dy = random.choice(directions): یه جهت تصادفی انتخاب می شه.

موقعیت جدید:

new_x و new_y: مختصات جدید با اضافه کردن dx و dy به موقعیت فعلی.

شرط حرکت:

حرکت فقط اگه این شرط ها برقرار باشه انجام می شه:

مختصات جدید تو شبکه باشه (0 <= new_x < GRID_SIZE و 0 <= new_y < GRID_SIZE).

سلول مقصد یا ناشناخته باشه (-1) یا ساختمان نباشه (!= BUILDING).

اجرای حرکت:

سلول فعلی خالی می شه (EMPTY).

موقعیت ایجنت به روزرسانی می شه (self.pos = [new_x, new_y]).

handle_interaction(): تعامل با سلول جدید بررسی می شه.

سلول جدید به AGENT تغییر می کنه.

نقشه کشف شده به روز می شه.

```python
def handle_interaction(self):
    x, y = self.pos
    observed_cell = self.discovered_map[x, y]
    if observed_cell == FIRE and random.choice([True, False]):
        success = self.env.apply_action(x, y, "extinguish")
        if success:
            self.discovered_map[x, y] = EMPTY
    elif observed_cell == CIVILIAN and not self.carrying_civilian and random.choice([True, False]):
        success = self.env.apply_action(x, y, "rescue")
        if success:
            self.carrying_civilian = True  # Agent now carries 1 civilian
            self.discovered_map[x, y] = EMPTY
            print(f"Civilian picked up! Score: {self.env.score}")
    elif observed_cell == STATION and self.carrying_civilian:
        success = self.env.apply_action(x, y, "drop")
        if success:
            self.carrying_civilian = False  # Civilian dropped, agent can carry another
```

این تابع هسته رفتار ایجنت رو تشکیل میده. تصمیم گیری تصادفی (50% شانس) برای آتش و غیرنظامی ساده ست، ولی می تونه با منطق هدفمندتر جایگزین بشه. مدیریت وضعیت حمل غیرنظامی هم به خوبی پیاده سازی شده.

x, y = self.pos: مختصات فعلی ایجنت.

observed_cell: نوع سلول فعلی از دید ایجنت (بر اساس discovered_map).

تعامل ها:

خاموش کردن آتش:

اگه سلول آتش (FIRE) باشه و با 50% شانس تصمیم به خاموش کردن بگیره:

("apply_action("extinguish فراخوانی می شه.

اگه موفق باشه، سلول تو نقشه ایجنت خالی می شه.

نجات غیرنظامی:

اگه سلول غیرنظامی (CIVILIAN) باشه، ایجنت غیرنظامی حمل نکنه و با 50% شانس تصمیم به نجات بگیره:

("apply_action("rescue فراخوانی می شه.

اگه موفق باشه:

self.carrying_civilian = True: ایجنت حالا یه غیرنظامی حمل می کنه.

سلول خالی می شه و پیام چاپ می شه.

تحویل به ایستگاه:

اگه سلول ایستگاه (STATION) باشه و ایجنت غیرنظامی حمل کنه:

("drop")apply_action فراخوانی می شه.

اگه موفق باشه، self.carrying_civilian = False می شه و ایجنت آزاد می شه.

جمع بندی کلاس Agent:

کلاس Agent یه ایجنت ساده اما کاربردی رو تعریف می کنه که:

تو محیط حرکت تصادفی می کنه.

یه دید محدود 5*5 داره.

می تونه آتش خاموش کنه، غیرنظامی نجات بده و به ایستگاه تحویل بده.

فقط یه غیرنظامی رو می تونه حمل کنه.

نقاط قوت:

دید محدود و نقشه کشف شده، شبیه سازی رو واقعی تر می کنه.

تعاملات اصلی (آتش، غیرنظامی، ایستگاه) به خوبی پوشش داده شده.

نقاط ضعف:

حرکت و تصمیم گیری تصادفی باعث می شه ایجنت بهینه عمل نکنه. اضافه کردن الگوریتم هایی مثل مسیر یابی (A*) یا یادگیری تقویتی می تونه عملکرد رو بهتر کنه.

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

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

    # Discovered map
    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')

    # Add a legend
    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()
    plt.show()
```

این تابع مثل یه دوربین عمل می کنه که دو تا نمای متفاوت از دنیای بازی رو به ما نشون میده:

نقشه کامل شهر (Full Map): یه تصویر از کل محیط که همه چیز رو همون طور که واقعاً هست نشون میده مثل یه نمای خداگونه که همه ساختمان ها، آتش ها، غیرنظامی ها، ایستگاه ها و ایجنت رو می بینیم. این به ما میگه تو دنیای واقعی چه خبره.

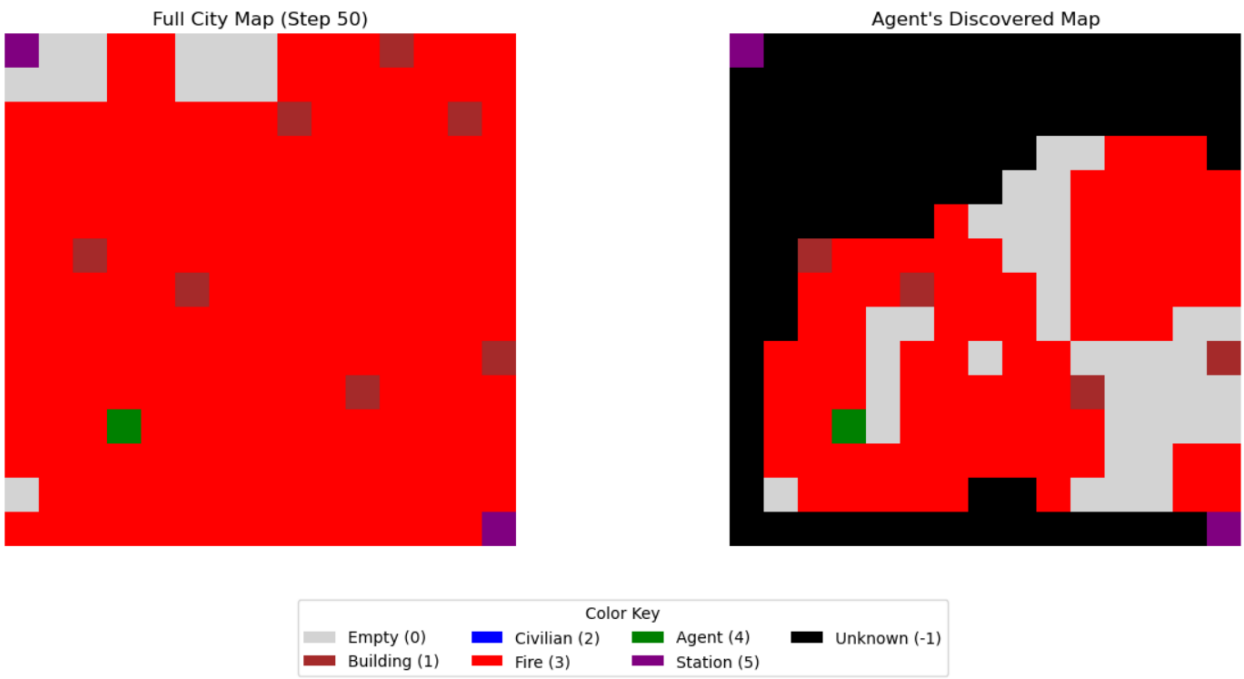
نقشه ایجنت (Discovered Map): فقط چیزایی که ایجنت تا حالا دیده رو نشون میده. جاهایی که هنوز نرفته سیاه (ناشناخته) هستن، و فقط محدوده اطرافش که حسگرش بهش دسترسی داره معلومه.

علاوه بر این، یه "راهنما" (legend) هم اضافه می کنه که به ما می گه هر رنگ چه معنایی داره (مثلاً قرمز یعنی آتش، آبی یعنی غیرنظامی). این تابع هر بار که فراخوانی می شه، این دو نقشه رو کنار هم میگذاره تا بتونیم هم کار ایجنت رو دنبال کنیم و هم ببینیم چقدر از محیط رو کشف کرده.

```python
def run_simulation():
    env = Environment()
    agent = Agent(env)

    while not env.is_game_over():
        clear_output(wait=True)
        agent.move()
        env.increment_step()

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

        display_maps(env, agent)
        time.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

    if env.steps >= MAX_STEPS:
        print(f"Game Over! Final Score: {env.score}")
        print(f"Fires extinguished: {env.fires_extinguished}")
        print(f"Civilians rescued: {env.civilians_rescued}/{NUM_CIVILIANS}")
```

این تابع مثل "موتور" کل شبیه سازی عمل می کنه. کارش اینه که بازی رو از صفر شروع کنه و تا آخر پیش ببره. مراحل کارش به این شکله:

1. **راه اندازی :**اول یه محیط (شهر) می سازه و ایجنت رو توش قرار میده.
2. **حلقه بازی :**بعد وارد یه چرخه می شه که توش :
    - ایجنت هر بار یه قدم حرکت می کنه.
    - تعداد قدم ها رو می شماره.
    - هر ۵ قدم، آتش رو پخش می کنه تا بازی سخت تر بشه.
    - نقشه ها رو نشون میده (با فراخوانی (display_mapsو یه کم صبر می کنه تا ما بتونیم ببینیم چی داره می شه.
3. **پایان بازی :**دو تا حالت داره که بازی تموم بشه :
    - **موفقیت :**اگه ایجنت همه آتش ها رو خاموش کنه و همه غیرنظامی ها رو نجات بده، پیغام برنده شدن میده.
    - **شکست :**اگه قدم ها به حد مشخص (۵۰) برسه و کار تموم نشده باشه، بازی رو قطع می کنه و نتیجه نهایی (امتیاز، تعداد نجات ها و آتش های خاموش شده) رو گزارش میده.




```
Game Over! Final Score: -680
Fires extinguished: 12
Civilians rescued: 0/8
```