به نام خدا

گزارش کار پروژه چهارم درس مبانی هوش مصنوعی

FOL & Knowlede Base

استاد:

حسین کارشناس

دستیار آموزش (TA):

پوریا صامتی

اعضای گروه:

پویا اسفندانی – یونس ایوبی راد

این پروژه با هدف پیاده‌سازی یک پایگاه دانش برای مسیریابی در محیطی شبکه‌ای و مبتنی بر گراف طراحی شده است. در این پایگاه دانش، توانایی مدیریت اندازه شبکه، تعیین موانع و موقعیت‌های عامل، و یافتن مسیر بهینه بین نقاط مختلف وجود دارد.

پایگاه دانش :

۱. قواعد پویا (Dynamic Rules)

```
:- dynamic(grid_size/2).
:- dynamic(agent/2).
:- dynamic(obstacle/2).
```

سه نوع قاعده پویا در پایگاه دانش تعریف شده است:

- grid_size/2 : اندازه محیط را با استفاده از مختصات دو بعدی (MaxX, MaxY) تعیین می‌کند.
- agent/2 : موقعیت فعلی عامل را در شبکه مشخص می‌کند.
- obstacle/2 : مکان‌های موانع را در شبکه تعریف می‌کند.

تعریف قواعد پویا به این دلیل انجام شده است که امکان تغییر و بروزرسانی اندازه شبکه، موقعیت عامل و موانع در زمان اجرا وجود داشته باشد.

۲. بررسی موقعیت در شبکه

تابع inside_grid(X, Y) بررسی می‌کند که آیا مختصات داده شده (X, Y) داخل محدوده تعریف‌شده شبکه قرار دارد یا خیر. این تابع از قاعده زیر پیروی می‌کند:

```
inside_grid(X, Y) :-
    grid_size(MaxX, MaxY),
    X ⩾ 1, X =< MaxX,
    Y ⩾ 1, Y =< MaxY.
```

این قاعده از محدودیت‌های شبکه استفاده کرده و اطمینان حاصل می‌کند که مختصات ورودی خارج از مرزهای شبکه نیست.

۳. بررسی حرکت معتبر

قاعده valid_move(X, Y) بررسی می‌کند که آیا موقعیت (X, Y) یک حرکت معتبر است یا خیر. برای این منظور، علاوه بر اطمینان از قرار داشتن در داخل شبکه، بررسی می‌کند که مکان مورد نظر مانع نباشد:

```
valid_move(X, Y) :-
    inside_grid(X, Y),
    \+ obstacle(X, Y).
```

۴. تعریف حرکات

چهار حرکت ممکن در شبکه تعریف شده‌اند:

- حرکت به راست (X+1, Y)
- حرکت به چپ (X-1, Y)
- حرکت به بالا (X, Y+1)
- حرکت به پایین (X, Y-1)

تابع move/2 این حرکات را با استفاده از قاعده‌های زیر پیاده‌سازی کرده است:

```
move((X, Y), (NX, Y)) :- NX is X + 1, valid_move(NX, Y).
move((X, Y), (NX, Y)) :- NX is X - 1, valid_move(NX, Y).
move((X, Y), (X, NY)) :- NY is Y + 1, valid_move(X, NY).
move((X, Y), (X, NY)) :- NY is Y - 1, valid_move(X, NY).
```

این قواعد تنها در صورتی حرکت را مجاز می‌دانند که مکان بعدی یک حرکت معتبر باشد.

۵. یافتن کوتاه‌ترین مسیر

یکی از بخش‌های مهم این پروژه، الگوریتم جستجوی **BFS** برای یافتن کوتاه‌ترین مسیر است. این الگوریتم در قاعده shortest_path/4پیاده‌سازی شده است. ورودی‌های این قاعده شامل موارد زیر است:

- Start: موقعیت شروع
- Goal: موقعیت هدف
- Path: مسیری که عامل باید طی کند
- MaxDepth: حداکثر عمقی که جستجو می‌تواند ادامه یابد

تابع bfs/5 برای اجرای الگوریتم جستجوی عرض‌اول (Breadth-First Search) استفاده می‌شود:

- اگر گره فعلی با گره هدف یکی باشد، مسیر برگردانده می‌شود.
- در غیر این صورت، با بررسی تمام حرکات ممکن و افزودن گره‌های جدید به صف جستجو، جستجو ادامه پیدا می‌کند :

```
bfs([[Goal | Rest] | _], Goal, Path, _, _) :-
    reverse([Goal | Rest], Path).

bfs([CurrentPath | OtherPaths], Goal, Path, MaxDepth, Depth) :-
    Depth < MaxDepth,
    CurrentPath = [CurrentNode | _],
    findall([NextNode | CurrentPath],
            (move(CurrentNode, NextNode),
             \+ member(NextNode, CurrentPath)),
            NewPaths),
    append(OtherPaths, NewPaths, UpdatedPaths),
    NewDepth is Depth + 1,
    bfs(UpdatedPaths, Goal, Path, MaxDepth, NewDepth).
```

ماژول main

تابع initialize_prolog

```python
def initialize_prolog(prolog, agent, obstacle):
    prolog.consult("knowledgebase.pl")

    prolog.retractall("grid_size(_, _)")
    prolog.retractall("agent(_, _)")
    prolog.retractall("obstacle(_, _)")

    prolog.assertz(f"grid_size({8}, {8})")

    prolog.assertz(f"agent({agent[0]}, {agent[1]})")

    for obstacle in obstacle:
        prolog.assertz(f"obstacle({obstacle[0]}, {obstacle[1]})")
```

این تابع مسئول مقداردهی اولیه محیط در Prolog است.

- ابتدا پایگاه دانش knowledgebase.pl به Prolog معرفی می‌شود.
- تمامی داده‌های قبلی مربوط به اندازه شبکه، موقعیت عامل، و موانع از پایگاه دانش حذف می‌شوند.
- اندازه شبکه با استفاده از قاعده grid_size/2 مشخص می‌شود (در اینجا به‌طور پیش‌فرض ۸×۸).
- موقعیت عامل با قاعده agent/2 اضافه می‌شود.
- تمام مختصات موانع با استفاده از قاعده obstacle/2 در پایگاه دانش ثبت می‌شوند.

2. تابع extract_environment_data

```python
def extract_environment_data(grid):
    agent = None
    obstacle = []

    for r, row in enumerate(grid):
        for c, cell in enumerate(row):
            if cell == 'B':
                agent = (r + 1, c + 1)
            elif cell == 'R':
                obstacle.append((r + 1, c + 1))

    return agent, obstacle
```

این تابع اطلاعات مربوط به محیط را از یک ماتریس دوبعدی استخراج می‌کند.

- ورودی این تابع ماتریسی است که هر خانه آن نشان‌دهنده وضعیت خاصی از محیط است:
  - 'B' نمایانگر موقعیت عامل.
  - 'R' نشان‌دهنده موقعیت موانع.
- با پیمایش ماتریس:
  - موقعیت عامل در متغیر agentذخیره می‌شود.
  - موقعیت تمام موانع به لیست obstacleاضافه می‌شود.
- خروجی تابع یک زوج شامل موقعیت عامل و لیست مختصات موانع است که می‌تواند برای مقداردهی Prologاستفاده شود.

تابع get_pig

```python
def get_pig(arr):
    p_indexes = []
    for i in range(8):
        for j in range(8):
            if arr[i][j] == 'P':
                p_indexes.append((i, j))
    return p_indexes
```

این تابع برای یافتن موقعیت تمام خانه‌هایی که دارای مقدار 'P' هستند در یک آرایه دوبعدی (۸×۸) استفاده می‌شود.

arr یک ماتریس ۸×۸ که نشان‌دهنده محیط یا شبکه است.

با دو حلقه تو در تو، تمام خانه‌های ماتریس بررسی می‌شود.

اگر مقدار خانه‌ای برابر با 'P' باشد، مختصات آن خانه به لیست p_indexes اضافه می‌شود.

لیستی از تمامی مختصات (به‌صورت (i, j))که مقدار 'P' دارند.

تابع find_path

```python
def find_path(start, goal):
    depth = [64, 128, 256, 512, 1024, 2048, 4096, 10000, 15000, 20000, 50000]
    for i in depth:
        query = f"shortest_path(({start[0]}, {start[1]}), ({goal[0]}, {goal[1]}), Path,{i})"
        result = list(prolog.query(query))
        if result:
            return result[0]["Path"]
```

این تابع مسئول یافتن کوتاه‌ترین مسیر از موقعیت شروع به مقصد در شبکه است.

- ورودی‌ها:
  - Start مختصات نقطه شروع.

- Goal مختصات نقطه هدف.

- عملکرد:
  - با استفاده از چندین عمق جستجو (که در لیست depth تعریف شده‌اند)، سعی می‌کند مسیر کوتاه‌ترین فاصله را با استفاده از کوئری Prolog پیدا کند.
  - در هر عمق، کوئری shortest_path به Prolog ارسال می‌شود.
  - اگر نتیجه‌ای پیدا شود، مسیر (Path) به عنوان خروجی بازگردانده می‌شود.

- خروجی:
  - کوتاه‌ترین مسیر بین نقطه شروع و هدف (در قالب لیستی از مختصات)، یا None اگر مسیری وجود نداشته باشد.

تابع choose_path

```
def choose_path(env):
    bird = env.get_bird_position()

    pigs = get_pig(env.grid)
    nbird = [bird[0] + 1, bird[1] + 1]
    shortest_path = [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                     1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                     1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
    for pig in pigs:
        next = [pig[0] + 1, pig[1] + 1]
        path = find_path(nbird, next)
        if path is not None and len(shortest_path) > len(path):
            shortest_path = path
    return shortest_path
```

این تابع کوتاه‌ترین مسیر از موقعیت فعلی پرنده (Bird) به نزدیک‌ترین خوک (Pig) را پیدا می‌کند.

- ورودی:
  - Env شیءای که شامل اطلاعات مربوط به محیط (مانند موقعیت پرنده و شبکه) است.

- عملکرد:

- ابتدا موقعیت پرنده از محیط گرفته می‌شود.
- سپس، لیستی از مختصات تمام خوک‌ها در شبکه با استفاده از تابع get_pig بدست می‌آید.
- برای هر خوک، مسیر بین پرنده و خوک محاسبه می‌شود (با استفاده از find_path)
- کوتاه‌ترین مسیر ممکن انتخاب شده و به‌روزرسانی می‌شود.

- خروجی:
  - کوتاه‌ترین مسیر از پرنده به نزدیک‌ترین خوک.

حلقه اصلی برنامه :

```
env = FirstOrderAngry(template='simple')

screen, clock = PygameInit.initialization()
FPS = 8
env.reset()

agent, obstacle = extract_environment_data(env.grid)
initialize_prolog(prolog, agent, obstacle)

running = True
while running:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            pygame.quit()
    if len(steps) == 0:
        steps = list(choose_path(env))
        steps.pop(0)
    print(steps)
    action = get_action(env.get_bird_position(), steps.pop(0))
    bird_pos, is_win = env.bird_step(action)
    env.render(screen)
    if is_win:
        print(f'Win')
        running = False
    pygame.display.flip()
    clock.tick(FPS)
pygame.quit()
```

- حلقه while تا زمانی که بازی ادامه دارد (running = True) اجرا می‌شود.
- مدیریت رویدادها : اگر کاربر پنجره بازی را ببندد (pygame.QUIT)، برنامه به درستی بسته می‌شود.
- تعیین مسیر: اگر لیست گام‌ها (steps) خالی باشد، مسیر جدید از پرنده به نزدیک‌ترین خوک محاسبه می‌شود (choose_path(env)) و اولین گام مسیر حذف می‌شود.
- گام بعدی:
  - گام بعدی از لیست stepsبرداشته می‌شود و اقدام مناسب با استفاده از get_action تعیین می‌شود.
  - موقعیت جدید پرنده و وضعیت بازی (برد یا ادامه) با اجرای تابع env.bird_step(action) به‌روزرسانی می‌شود.