دانشگاه اصفهان

دانشکده مهندسی کامپیوتر

گزارش پروژه مبانی هوش ـ فاز ۴

استاد:

دکتر حسین کارشناس

دستیار آموزشی:

پوریا صامتی

اعضا گروه:

سپهر فاطمی

پوریا اردستانی

شیما مغزی

بهمن ۱۴۰۳

پیاده‌سازی پایگاه دانش:

پیاده سازی پایگاه دانش به صورت زیر می‌باشد. در این پیاده‌سازی، کد پایتون پس از انجام هر کنشی، وضعیت فعلی محیط (grid) را در قالب Query به پایگاه دانش ارسال می‌کند و کنشی که باید در آن وضعیت انجام دهد را دریافت می‌کند.

```
 :- dynamic position/3.
:- dynamic visited/2.
```

دو predicates به صورت dynamic تعریف می‌کنیم؛ مقادیر این predicates ها می‌تواند در زمان اجرا تغییر کند.

position(Type, X, Y) نوع عنصر واقع در موقعیت X,Y گرید را ذخیره نگه می‌دارد.

visited(X, Y) موقعیت‌هایی که پرنده آنها را طی کرده است نگه می‌دارد.

```
init_state(Grid) :-
    retractall(position(_, _, _)),
    grid_to_positions(Grid, 0).
```

موقعیت‌های قبلی را ریست می‌کنیم و وضعیت جدید grid را initialize می‌کنیم. هدف این است که عناصر محیط و موقعیت‌های آنها را به فکت‌های position تبدیل کنیم؛ برای اینکار گرید را به صورت سطری و ستونی پیمایش می‌کنیم.

```
grid_to_positions([], _).
grid_to_positions([Row|Rest], RowNum) :-
    process_row(Row, RowNum, 0),
    NextRow is RowNum + 1,
    grid_to_positions(Rest, NextRow).
process_row([], _, _).
process_row([Cell|Rest], RowNum, ColNum) :-
    atom_string(CellAtom, Cell),
    add_position(CellAtom, RowNum, ColNum),
    NextCol is ColNum + 1,
    process_row(Rest, RowNum, NextCol).
```

در این قسمت هر یک از سلول‌های grid را تبدیل به یک atom میکنیم و عنصر آن خانه به همراه موقعیتش را به پایگاه دانش اضافه می‌کنیم. (add_position)

```
add_position('B', X, Y) :- assertz(position(bird, X, Y)).
add_position('P', X, Y) :- assertz(position(pig, X, Y)).
add_position('R', X, Y) :- assertz(position(rock, X, Y)).
add_position('T', _, _).
```

فکت‌های موقعیت‌های پرنده، خوک‌ها و سنگ‌ها به صورت داینامیک به پایگاه دانش اضافه می‌شوند.

```
move(right, 0, 1, 3).
move(up, -1, 0, 0).
move(down, 1, 0, 1).
move(left, 0, -1, 2).
```

۴ کنش ممکن در محیط را تعریف می‌کنیم؛ برای مثال کنش راست که شناسه آن ۳ می‌باشد، در سطر (Row) هیچ تغییری نمی‌دهد، ولی ستون پرنده را (Column) یک واحد افزایش می‌دهد.

```
valid_position(X, Y) :-
    X >= 0, X < 8,
    Y >= 0, Y < 8.
```

از این predicate در valid_move استفاده می‌شود تا بررسی کند که کنش عامل منجر به خروج از مرز گرید محیط نشود.

```
manhattan_distance(X1, Y1, X2, Y2, Distance) :-
    DX is abs(X2 - X1),
    DY is abs(Y2 - Y1),
    Distance is DX + DY.
```

از manhattan_distance برای محاسبه فاصله منهتن بین دو position در گرید استفاده می‌کنیم. این فاصله در Distance ذخیره می‌شود. با استفاده از فاصله منهتن به پرنده اولویت میدهیم تا کنشی را انتخاب کند که آنرا به یک خوک نزدیکتر می‌کند.

```
valid_move(X, Y, NewX, NewY) :-
    valid_position(NewX, NewY),
    \+ position(rock, NewX, NewY).
```

در صورتی که پرنده بخواهد از X,Y به موقعیت جدید NewX, NewY برود، باید بررسی شود که این موقعیت خارج از مرزهای گرید بازی نیست و اینکه شامل rock نیز نمی‌باشد.

```
choose_best_action(X, Y, Action) :-
    findall(
        (Dist, A, NewX, NewY),
        (position(pig, PigX, PigY),
         move(_, DX, DY, A),
         NewX is X + DX,
         NewY is Y + DY,
         valid_move(X, Y, NewX, NewY),
         manhattan_distance(NewX, NewY, PigX, PigY, Dist)),
        Moves
    ),
    sort(Moves, SortedMoves),
    remove_visited(SortedMoves, FilteredMoves),
    FilteredMoves = [(_, Action, _, _)|_],
    !.
```

هدف از این predicate این است که کنشی را انتخاب کند که فاصله عامل را تا نزدیکترین خوک، کم کند. ورودی X,Y که موقعیت فعلی پرنده است را می‌گیرد و Action را برمیگرداند.

یک لیست از تمامی کنش های عامل که valid هستند را به همراه فاصله منهتن آنها در قالب (Dist, A, NewX, NewY) به دست می‌آوریم. (findall).

شناسه کنش ها به همراه جهت‌های حرکتی آنها را با استفاده از فکت‌های move بازیابی می‌کنیم و برای هر یک از move ها، NewX و NewY را به دست می‌آوریم. سپس valid بودن موقعیت جدید را بررسی می‌کنیم و سپس فاصله منهتن را محاسبه کرده و در Dist ذخیره میکنیم. این حالات مختلف را در لیست Moves ذخیره می‌کنیم. پس هر المان از لیست ما شامل یک تاپل به صورت (Dist, A, NewX, NewY) می‌باشد.

سپس این لیست را براساس Dist ، Sort میکنیم. پس از آن، با remove_visited، خانه‌هایی را که عامل قبلا آنها را پیمایش کرده است را از لیست حذف می‌کنیم.

```
remove_visited([], []).
remove_visited([(Dist, A, X, Y)|Rest], Filtered) :-
    (visited(X, Y) ->
        remove_visited(Rest, Filtered)
    ;
        assertz(visited(X, Y)),
        Filtered = [(Dist, A, X, Y)|FilteredRest],
        remove_visited(Rest, FilteredRest)
    ).
```

با استفاده از یک ساختار شرطی، بررسی می‌کنیم که آیا موقعیت X,Y قبلا پیمایش شده است یا خیر (visited(X,Y) is true of flase) در صورتی که true باشد، برای بقیه المان های لیست مجددا remove_visited را فراخوانی می‌کنیم؛ در صورتی که false باشد، ابتدا به صورت dynamic، فکت visited(X,Y) را به پایگاه دانش اضافه می‌کنیم و سپس آن کنش را به لیست Filtered اضافه می‌کنیم.

در نهایت، اولین Action از لیست FilteredMoves به عنوان کنش عامل انتخاب می‌کنیم.

```
get_next_action(Grid, Action) :-
    init_state(Grid),
    position(bird, BirdX, BirdY),
    choose_best_action(BirdX, BirdY, Action),
    !.
```

get_next_action ، Query اصلی ما خواهد بود که grid فعلی را ورودی می‌گیرد و پس از initialize کردن آن و اضافه کردن موقعیت پرنده به پایگاه دانش و سپس اجرای choose_best_action، کنش مناسب را برای عامل انتخاب می‌کند.

پیاده‌سازی کد پایتون:

کد پایتون به صورت زیر می‌باشد:

```
prolog = Prolog()
prolog.consult("KB2.pl")

env = FirstOrderAngry(template='simple')
screen, clock = PygameInit.initialization()
FPS = 8
env.reset()

def get_prolog_action(env):

    grid_list = env.grid

    # print("Current Grid State:")
    # for row in grid_list:
    #     print(row)

    grid_str = str(grid_list)
    query = f"get_next_action({grid_str}, Action)"

    print("\nProlog Query:", query)

    result = list(prolog.query(query))
    print("\nProlog Result:", result)

    if not result:
        raise RuntimeError("Error no valid path")

    return result[0]['Action']


def print_grid_state(env):
    print("\nCurrent Grid State:")
    for row in env.grid:
        print(' '.join(row))
    print()


running = True
while running:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            pygame.quit()

    #print_grid_state(env)
    action = get_prolog_action(env)
    print(f"action: {action}")

    bird_pos, is_win = env.bird_step(action)
    print(f"Bird moved to : {bird_pos}")

    env.render(screen)
```

```
    if is_win:
        print('Win!!!!!')
        running = False

    pygame.display.flip()
    clock.tick(FPS)

pygame.quit()
```

در تابع get_prolog_action، یک get_next_action Query به همراه grid فعلی به پایگاه دانش فرستاده می‌شود و یک Action از آن دریافت می‌شود. عامل این کنش را انجام می‌دهد و سپس برای کنش بعدی، مجددا یک Query ارسال می‌کند و تا زمانی که بازی خاتمه یابد، این فرآیند تکرار می‌شود.