Proyecto de Simulación de Agentes. Programación Declarativa. Prolog.

Masiel Villalba Carmenate villalbamasiel@gmail.com

Facultad de Matemática y Computación (MATCOM),
Universidad de la Habana (UH), Cuba.

1. Definición del problema

Dado un tablero donde en cada casilla puede aparecer alguno de los siguientes elementos:

- suciedad
- niños
- corrales
- obstáculos

Existen las siguientes leyes:

- Los niños se mueven y ensucian arbitrariamente.
- Las casillas de tipo obstáculo no pueden ser ocupadas por ningún agente.
- Los niños pueden mover los obstáculos.
- Si un niño está en una casilla de tipo corral, entonces no puede moverse ni ensuciar.
- Existe un intervalo de tiempo t según el cual el tablero cambia aleatoriamente.

Se busca implementar un agente que consiga limpiar el tablero y ubicar a todos los niños en una casilla corral antes de que se rebase un umbral de suciedad predeterminado, en caso contrario se considera que queda despedido. Este agente no puede mover los obstáculos de lugar pero sí puede cargar o dejar a un niño en una posición determinada.

2. Principales ideas

Para la implementación del robot se modelaron dos comportamientos: reactivo y proactivo.

2.1. Agente proactivo

El robot proactivo se caracteriza por tener un plan muy bien definido, e "inquebrantable": recoger primero a todos los niños. Puesto que ellos son los responsables de la suciedad, una vez que estén tranquilos se puede comenzar con la limpieza y que no sea en vano. Esta planificación para cuando el intervalo de cambio del tablero es grande, es verdaderamente ideal, pues sería suficiente para que el robot recoja y limpie todo antes de que se desordene el ambiente de nuevo. Pero para cuando el t es pequeño, se trata entonces de un ambiente dinámico, en el que el objetivo de ubicar a los niños nunca será cumplido, y por tanto el de la limpieza tampoco. Hemos escuchado algunas veces que el comportamiento reactivo figura en realizar acciones random en cada turno.

2.2. Agente reactivo

El nuestro es aparentemente similar al proactivo, al menos en el inicio del código, pero en realidad, en lugar de enfocarse ciegamente en recoger primero a todos niños siempre realiza la acción más inmediata. Lo más inmediato realmente sería limpiar siempre primero, pero en ninguno de los dos casos fue considerada esta estrategia, puesto que el robot es el primero en actuar en cada turno y justo después de su actuación algún niño puede ensuciar la casilla que acaba de limpiar. Por eso para los dos casos tenemos que siempre que se pueda deje a un niño en el corral si carga alguno y que recoja a cualquiera que tenga cerca si no carga ninguno. Entonces, regresando a la reactividad, este además de hacer la acciõn más inmediata, se mantiene recogiendo niños, o al menos intentándolo, mientras el porciento de suciedad este por debajo de 40, y cuando se alcance este tope, ya empieza a preocuparse por priorizar la limpieza. Tambien hemos asumido que el robot puede limpiar aunque lleve un niño cargado. En resumen, creemos que un agente reactivo no es aquel que no se plantee objetivos, claro que no, sino que es sensible al ambiente, y sabe reordenar sus prioridades según como sea afectado por las condiciones en que esta. Está consciente de que sus planes pueden frustrarse por razones externas y por eso siempre hace lo que es más seguro, más urgente.

3. Características de la implementación

3.1. Función de arranque

Los argumentos que recibe la función principal para correr el programa son:

- N: cantidad de filas del tablero a generar.
- M: cantidad de columnas del tablero a generar.
- Ps: porciento de suciedad inicial respecto a la cantidad de casillas del tablero.
- Po: porciento de obstáculos respecto a la cantidad de casillas del tablero.
- Niños: cantidad de niños a ubicar.
- T: periodo de cambio del ambiente.

```
// proyecto.pl
main(N,M,Ps,Po,Ninnos,T) :-
    retractall(tablero(X)), retractall(no_ninnos(X)),
    tablero(N,M,Tablero), assert(tablero(Tablero)),
        assert(no_ninnos(Ninnos)),
    Mult is N*M,
    parte(Mult,Ps,Cs),
    parte(Mult,Po,Co),
    generar_tablero(N,M,Cs,Co),
    simulacion(1,T,N,M).
```

3.2. Lógica y hechos

La representación del tablero se consiguió a través de predicados dinámicos. Se tiene:

- corral/1
- niño/1
- robot/1
- sucio/1
- obstaculo/1

Cada uno recibe una tupla que representa una casilla y retornan `true` si en esa posicón del tablero hay un elemento de ese tipo. Por ejemplo, `sucio((4,5))` es `true` si la casilla (4,5) está sucia. Para ubicar o eliminar un elemento en una casilla se emplean los predicados `assert` y `retracall` respectivamente. La forma en que se diseño el tablero garantiza que nunca se haga `assert` a una posición que no existe en el tablero.

3.3. Predicados principales

Algunos predicados han sido imprescindibles para modelar la naturaleza del problema:

- `carga_niño/0`: es verdadero si el robot lleva cargado algún niño. Cuando el robot carga un niño, este "deja de existir"temporalmente, puesto que niño(Casilla_del_robot) es falso.
- `tablero/1`, `no_niños/1`: no serían tan necesarios pero se han agregado para tener más a mano dos datos importantes: la cantidad de niños y las casillas del tablero $N * M$, que en una misma ejecución no cambian. `no_niños(X)` es verdader si hay `X` niños en el tablero.

4. Generando tableros aleatorios

```
// proyecto.pl
generar_tablero(N,M,Ns,No):-
    limpiar_todo, tablero(Tablero), no_ni\~nos(Nc),
    mi_write(['Corrales: ',Nc,' Sucias: ',Ns,' Obstaculos: ',No]),
    lograr_K_corrales(Tablero,Nc,N,M,0),
    findall(C,(member(C,Tablero),not(ni\~no(C))),Noboy),
    action_random(1,Noboy,poner_robot),
    findall(C,(member(C,Tablero), not(corral(C))),Nocorral),
    action_random(Ns,Nocorral,poner_suciedad),
    findall(G,vacia(G),Para_obs),
    action_random(No,Para_obs,poner_obstaculo).
```

5. Flujo de la simulación

```
// proyecto.pl
simulacion(Tiempo,_,N,M) :-
    porciento_suciedad(P), 60=<P,
    mi_write(['la suciedad a alcanzado el ',P,' pociento.']),
    writeln('El robot queda despedido, termina la simulacion'),
    informe(N,M,Tiempo,'Despedido'),!.
simulacion(Tiempo,_,_,_) :-
    findall(X,sucio(X),[]), findall(X,(ni\~no(X),not(corral(X))),[]),
    write('todo esta limpio y ordenado, termina la simulacion'),
    informe(N,M,Tiempo,'OK'), !.
simulacion(Tiempo,Interval,_,_) :-
    Tiempo=:=100*Interval, write('se ha alcanzado 100 veces t, termina
        la simulacion'),
    informe(N,M,Tiempo,'100 veces t'), !.
simulacion(Tiempo,Interval,N,M) :-
    mi_write(['\n\n\nESTAMOS EN EL MINUTO ',Tiempo]),
    robot(Robot_pos), cuadricula(Robot_pos,Cuadro),
    writeln('ACTUACION DEL ROBOT'),!,
    actua_robot_reactivo(Robot_pos,Cuadro),
    ((findall(X, ni\~no(X), ListNi\~nos), mover_Ni\~nos(ListNi\~nos)) ;
        true),
    Modulo is Tiempo mod Interval, try_change_enviroment(Modulo,N,M),
    T is Tiempo+1, ver_tablero,
    simulacion(T,Interval,N,M).
```

6. Lista de predicados de Prolog más utilizados en el proyecto

- assert/1
- findall/3

- `retractall/1`
- `numlist/3`
- `member/2`

Ver funcionamiento de cada uno en `https://www.swi-prolog.org/`.

Referencias

1. Bramer, Max: Logic programming with Prolog, Springer, 2005