Artificial intelligence - Project 3
- Limbaje de planificare-

Galis Laurentiu

14/1/2021

1 Introducere

Pentru realizarea acestui proiect au fost nevoie de cunostinte legate de planificarea unei probleme si gasirea pasilor care duc la un scop bine stablit. Acest lucru se poate realiza din punct de vedere software cu ajutorul diferitelor lombaje de planificare ge genereaza toate STARILE unei probleme ce are un DOMENIU unic. Un exemplu de acest limbaj este PDDL.

Proiectul este format din doua probleme de ce contin doua domenii diferite si fiecare avand cate doua probleme atasate domeniului.

1.1 Cerintele problemelor

1.1.1 Hanoi Towers

In aceasta problema este vorba despre 3 bare in care se afla un diferit numar de discuri de diferite dimensiuni. pentru inceput aceste bare sunt puse in ordine crescatoare din punctul de vedere al dimensiunii pe prima bara. Scopul final al acestui puzzle este de a muta aceste discuri pe alta bara sub aceeasi forma. Mai exact SE MUTA O SINGURA DATA cate un disc pe alta bara pana cand se ajunge la solutia finala. Cu cat se afla mai multe discuri pe bara, cu atat creste si complexitatea problemei.

1.1.2 Labirint

In aceasta problema este vorba despre un agent care este situat la o anumita pozitie pe o MATRICE. Aceasta matrice poate contine sau nu ziduri aflate la diferite pozitii pe matrice. Agentul nu poate sa se afle pe un zid sau sa poata trece de un zid acesta trebuind sa-l ocoleasca. Scopul final al agentului este sa ajunga la POARTA care il va elibera din labirint. Aceasta poarta poate fi situata oriunde in labirint.

2 Solutie conceptuala

Pentru rezolvarea acestor probleme este nevoie de a defini domeniul si diferite probleme. Domeniul descrie cum arata MEDIUL in care se desfasoara actiunile, OBIECTELE(agentii) care fac actiunile si ACTIUNILE care au loc in acest mediu. Avand un domeniu se pot genera diferite probleme pe baza unui domeniu. In probleme se folosesc obiectele si actiunile definite in domeniu si se pot crea diferite cazuri. Asadar va fi nevoie de un fisier pentru a defini domeniul si alt fisier pentru a defini problemele.

2.1 Definirea domeniilor

Domeniul descrie cum arata mediul. Asadar avem nevoie de predicate(agenti) si actiuni la care sunt supuse predicatele.

2.1.1 Hanoi Towers

- Predicate: discul NU SE MAI AFLA pe pozitia initiala, discul SE AFLA pe alt obiect si un disc este MAI MIC decat alt disc
- Actiuni: MUTAREA discului de pe o pozitie la alta

2.1.2 Labirint

- Predicate: se INCREMENTEAZA agentul cu o pozitie in matrice, se DECREMENTEAZA agentul cu o pozitie, agentul SE AFLA la o anumita pozitite in matrice si SE AFLA UN ZID la o anumita pozitie
- Actiuni: mutarea agentului in SUS, mutarea agentului in JOS, mutarea agentului in STANGA si mutarea agentului in DREAPTA.

2.2 Definirea problemelor

Problemele folosesc diferite domenii si supun agentii la diferite cazuri si diferite actiuni avand ca scop GENERAREA UNUI SCOP(goal). Mai jos sunt prezentate niste imagini despre cum ar trebui sa arate problemele.

2.2.1 Hanoi Towers

- Mai jos este prezentat puzzle-ul Hanoi Towers cu 3 discuri (Problema 1)

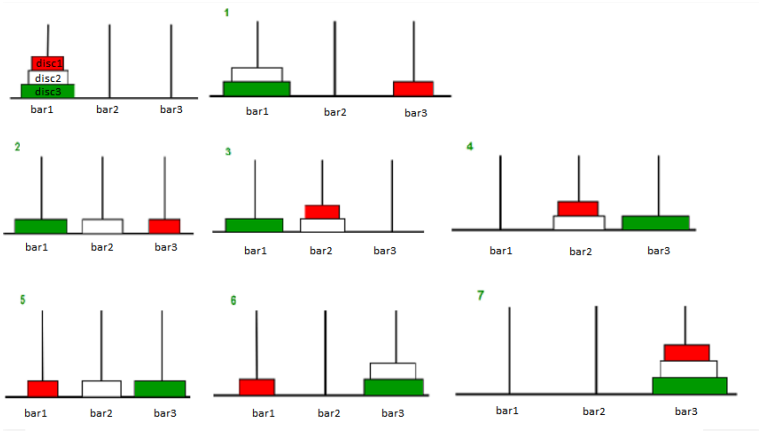

Figure 1: Problema 1.

- Mai jos este prezentat puzzle-ul Hanoi Towers cu 4 discuri (Problema 2)

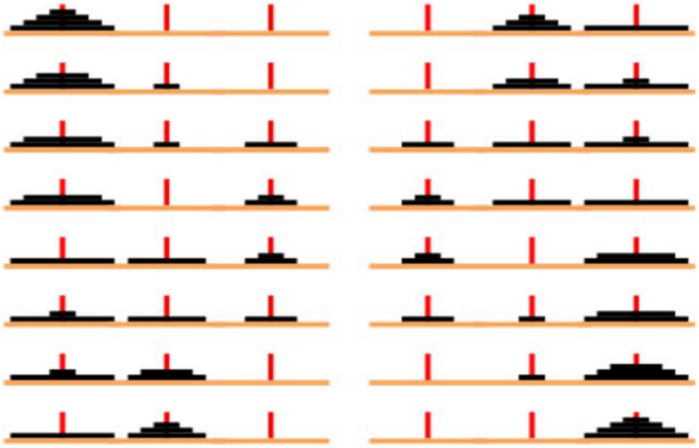

Figure 2: Problema 2.

2.2.2 Labirint

- Mai jos este prezentat labirintul fara ziduri (Problema 1)

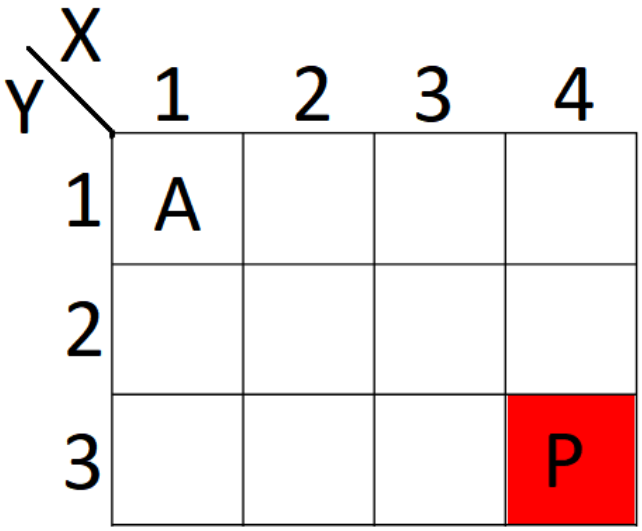

Figure 3: Problema 3.

- Mai jos este prezentat labirintul cu ziduri (Problema 2)

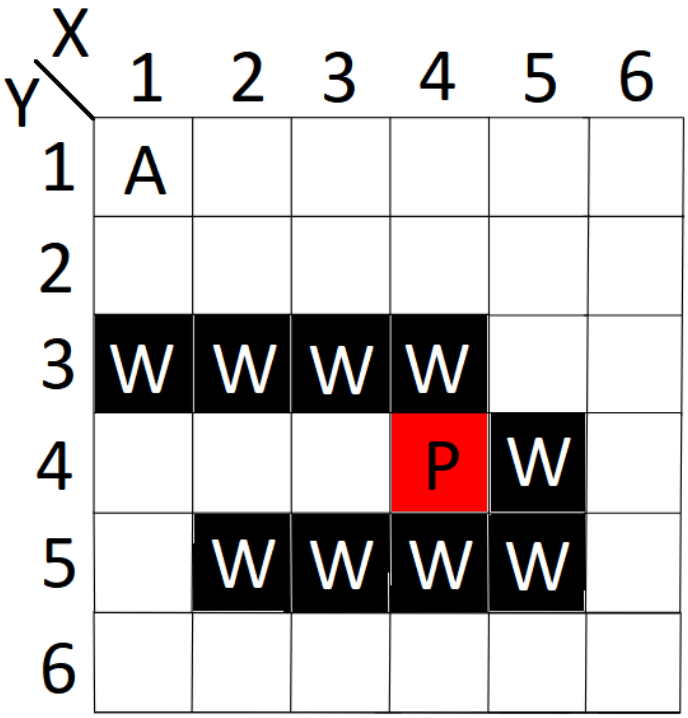

Figure 4: Problema 4.

2.3 PDDL

PDDL este un limbaj axat pe actiune, inspirat de binecunoscutele formulari STRIPS ale problemelor de planificare. La baza sa se află o simpla standardizare a sintaxei pentru a exprima familiar semantica actiunilor, folosind pre- si post- conditii pentru a descrie aplicabilitatea si efectele unei actiuni. Sintaxa este inspirată de Lisp, deci o mare parte din structura unei descrieri de domeniu este o listă de expresii cu paranteze asemănătoare cu Lisp. O problema de planificare este creata prin asocierea unei descrieri de domeniu cu o descriere a problemei. Aceeasi descriere a domeniului poate fi asociata cu multe descrieri de probleme diferite pentru a produce diferite probleme de planificare in acelasi domeniu. Conditiile pre si post actiuni sunt exprimate ca propozitii logice construite din predicate si termeni de argumente (obiecte dintr-o instanta problema) si conectivitati logice. .

3 Implementare

Ambele probleme au aproape aceeasi idee de implementare diferenta dintre ele fiind domeniile si parametri din acestea. Cu cat numarul parametrilor este mai mare sau complexitatea acestora este mai mare, cu atat si problema va fi mult mai complexa.

3.1 Hanoi Towers

3.1.1 Reprezentare DOMENIU in PDDL

Predicatele folosite la aceasta problema sunt:

- (clear ?x) - discul nu se mai afla unde era
- (on ?x ?y) - x se afla pe y
- (smaller ?x ?y)) - x este mai mic decat y

Code:

```
(define (domain hanoi)
  (:requirements :strips)
  (:predicates
                 (clear ?x) ;discul nu se mai afla unde era
                 (on ?x ?y) ; x se afla pe y
                 (smaller ?x ?y)) ; x este mai mic decat y

  ;este doar o singura actiune in joc(muti un disc doar o singura data)
  (:action move
                   :parameters (?disc ?from ?to)
            :precondition (and (smaller ?to ?disc)
                         (on ?disc ?from)
                         (clear ?disc)
                         (clear ?to))
            :effect (and (clear ?from)
                                (on ?disc ?to)
                                (not (on ?disc ?from))
                                      (not (clear ?to)) ) )
  )
```

3.1.2 Reprezentare PROBLEMA 1 in PDDL

Problema 1 este reprezentata in imaginea de mai sus(Problema 1):

Code:

```
(define (problem hanoi3Disc)
  (:domain hanoi)
  (:objects bar1 bar2 bar3 disc1 disc2 disc3)
  (:init
   (smaller bar1 disc1) (smaller bar1 disc2) (smaller bar1 disc3)
   (smaller bar2 disc1) (smaller bar2 disc2) (smaller bar2 disc3)
   (smaller bar3 disc1) (smaller bar3 disc2) (smaller bar3 disc3)
   (smaller disc2 disc1) (smaller disc3 disc1) (smaller disc3 disc2)

   (clear bar2) (clear bar3) (clear disc1)

   (on disc3 bar1) (on disc2 disc3) (on disc1 disc2))
```

```
13
14    (:goal (and (on disc3 bar3)
15                (on disc2 disc3)
16                (on disc1 disc2)))
17    )
```

3.1.3 Reprezentare PROBLEMA 2 in PDDL

Problema 2 este reprezentata in imaginea de mai sus(Problema 2):

Code:

```
1  (define (problem hanoi4Disc)
2    (:domain hanoi)
3    (:objects bar1 bar2 bar3 disc1 disc2 disc3 disc4)
4    (:init
5     (smaller bar1 disc1) (smaller bar1 disc2) (smaller bar1 disc3) (smaller bar1 disc4)
6     (smaller bar2 disc1) (smaller bar2 disc2) (smaller bar2 disc3) (smaller bar2 disc4)
7     (smaller bar3 disc1) (smaller bar3 disc2) (smaller bar3 disc3) (smaller bar3 disc4)
8     (smaller disc2 disc1) (smaller disc3 disc1) (smaller disc3 disc2) (smaller disc4 disc1)
9     (smaller disc4 disc2) (smaller disc4 disc3)
10
11    (clear bar2) (clear bar3) (clear disc1)
12
13    (on disc4 bar1) (on disc3 disc4) (on disc2 disc3) (on disc1 disc2))
14
15   (:goal (and (on disc4 bar3)
16               (on disc3 disc4)
17               (on disc2 disc3)
18               (on disc1 disc2)))
19   )
```

3.2 Labirint

Un lucru in plus adaugat la aceasta problema fata ce cealalta este ca aici am definit doua tipuri ale predicatelor de tipul "agent" si "position" pentru a face compilatorul sa inteleaga ce tipuri de date folosesc. Aceste tipuri se pot declara oricum vrea programatorul. SINTAXA: (:types type1 - object).
Mai mult de atat, la realizarea actiunilor am folosit si operatori conditionali formati din operatorul FORALL (SINTAXA: (forall (?v1 ... ?vn)<effect>)) si operatorul WHEN (SINTAXA: (when <condition> <effect>)). Acest lucru face descrierea domeniului mult mai usoara si mult mai usor de inteles codul.

3.2.1 Reprezentare DOMENIU in PDDL

Predicatele folosite la aceasta problema sunt:

- (inc ?x ?y - position) - incrementez cu o pozitie agentul
- (dec ?x ?y - position) - decrementez cu o pozitie agentul
- (at ?a - agent ?x ?y - position) - agentul se afla la o anumita pozitie
- (wall ?x ?y) - la o anumita pozitie se afla un perete

Code:

```
1  (define (domain labirint)
2    (:requirements :strips)
3    (:types agent position)
4    (:predicates
```

```
    (inc ?x ?y - position)
    (dec ?x ?y - position)
    (at ?a - agent ?x ?y - position)
    (wall ?x ?y))

  (:action move-up
    :parameters (?a - agent)
    :effect (forall (?x ?y ?yNew - position)
                    (when
                      (and (at ?a ?x ?y)
                           (dec ?y ?yNew)
                           (not (wall ?x ?yNew)))
                      (and (not (at ?a ?x ?y))
                           (at ?a ?x ?yNew)))
                    ))

  (:action move-down
    :parameters (?a - agent)
    :effect (forall (?x ?y ?yNew - position)
                    (when
                      (and (at ?a ?x ?y)
                           (inc ?y ?yNew)
                           (not (wall ?x ?yNew)))
                      (and (not (at ?a ?x ?y))
                           (at ?a ?x ?yNew)))
                    ))

  (:action move-right
    :parameters (?a - agent)
    :effect (forall (?x ?y ?xNew - position)
                    (when
                      (and (at ?a ?x ?y)
                           (inc ?x ?xNew)
                           (not (wall ?xNew ?y)))
                      (and (not (at ?a ?x ?y))
                           (at ?a ?xNew ?y)))
                    ))

  (:action move-left
    :parameters (?a - agent)
    :effect (forall (?x ?y ?xNew - position)
                    (when
                      (and (at ?a ?x ?y)
                           (dec ?x ?xNew)
                           (not (wall ?xNew ?y)))
                      (and (not (at ?a ?x ?y))
                           (at ?a ?xNew ?y)))
                    ))
)
```

3.2.2 Reprezentare PROBLEMA 3 in PDDL

Problema 3 este reprezentata in imaginea de mai sus(Problema 3):
Code:

```
(define (problem labirintP1)
  (:domain labirint)
  (:objects x1 x2 x3 x4 y1 y2 y3 y4 - position A - agent)        ;4 linii si 4 coloane
  (:init
    (inc x1 x2) (inc x2 x3) (inc x3 x4)
    (inc y1 y2) (inc y2 y3) (inc y3 y4)
    (dec x4 x3) (dec x3 x2) (dec x2 x1)
    (dec y4 y3) (dec y3 y2) (dec y2 y1)
    (at A x1 y1))
  (:goal (at A x4 y4)))
```

3.2.3 Reprezentare PROBLEMA 4 in PDDL

Problema 4 este reprezentata in imaginea de mai sus(Problema 4):

Code:

```
(define (problem labirintP2)
  (:domain labirint)
  (:objects x1 x2 x3 x4 x5 x6 y1 y2 y3 y4 y5 y6 - position A - agent) ;6 linii si 6 coloane
  (:init
    (inc x1 x2) (inc x2 x3) (inc x3 x4) (inc x4 x5) (inc x5 x6)
    (inc y1 y2) (inc y2 y3) (inc y3 y4) (inc y4 y5) (inc y5 y6)
    (dec x6 x5) (dec x5 x4) (dec x4 x3) (dec x3 x2) (dec x2 x1)
    (dec y6 y5) (dec y5 y4) (dec y4 y3) (dec y3 y2) (dec y2 y1)

    (wall x1 y3) (wall x2 y3) (wall x3 y3) (wall x4 y3) (wall x5 y4)
    (wall x2 y5) (wall x3 y5) (wall x4 y5) (wall x5 y5)

    (at A x1 y1))
  (:goal (at A x3 y4)))
```

4 Rezultate

Rezultatele au fost generate de catre un translator in sistemul de operare Linux. Fiecare problema a fost rulata cu doua euristici diferite care au la baza Algoritmul de cautare A*.
Output-urile fiecarei probleme sunt descrise mai jos.

4.1 Problema 1

- Euristica h=ff()
  Comanda: /fast-downward.py hanoi.pddl hanoiP1.pddl –heuristic "h=ff()" – search "astar(h)"
  Output:
  move disc1 disc2 bar3 (1)
  move disc2 disc3 bar2 (1)
  move disc1 bar3 disc2 (1)
  move disc3 bar1 bar3 (1)
  move disc1 disc2 bar1 (1)
  move disc2 bar2 disc3 (1)
  move disc1 bar1 disc2 (1)

- Euristica h=cg()
  Comanda: /fast-downward.py hanoi.pddl hanoiP1.pddl –heuristic "h=cg()" – search "astar(h)" move disc1 disc2 bar3 (1)
  move disc2 disc3 bar2 (1)
  move disc1 bar3 disc2 (1)
  move disc3 bar1 bar3 (1)
  move disc1 disc2 bar1 (1)
  move disc2 bar2 disc3 (1)
  move disc1 bar1 disc2 (1)

4.2 Problema 2

- Euristica h=ff()
  Comanda: /fast-downward.py hanoi.pddl hanoiP2.pddl –heuristic "h=ff()" – search "astar(h)"
  Output:
  move disc1 disc2 bar2 (1)
  move disc2 disc3 bar3 (1)
  move disc1 bar2 disc2 (1)
  move disc3 disc4 bar2 (1)
  move disc1 disc2 disc4 (1)
  move disc2 bar3 disc3 (1)
  move disc1 disc4 disc2 (1)
  move disc4 bar1 bar3 (1)
  move disc1 disc2 disc4 (1)
  move disc2 disc3 bar1 (1)
  move disc1 disc4 disc2 (1)
  move disc3 bar2 disc4 (1)
  move disc1 disc2 bar2 (1)
  move disc2 bar1 disc3 (1)
  move disc1 bar2 disc2 (1)

- Euristica h=cg()
  Comanda: /fast-downward.py hanoi.pddl hanoiP2.pddl –heuristic "h=cg()" – search "astar(h)" **Output:**

move disc1 disc2 bar2 (1)
move disc2 disc3 bar3 (1)
move disc1 bar2 disc2 (1)
move disc3 disc4 bar2 (1)
move disc1 disc2 disc4 (1)
move disc2 bar3 disc3 (1)
move disc1 disc4 disc2 (1)
move disc4 bar1 bar3 (1)
move disc1 disc2 disc4 (1)
move disc2 disc3 bar1 (1)
move disc1 disc4 disc2 (1)
move disc3 bar2 disc4 (1)
move disc1 disc2 bar2 (1)
move disc2 bar1 disc3 (1)
move disc1 bar2 disc2 (1)

4.3 Problema 3

- Euristica h=ff()
  Comanda: /fast-downward.py labirint.pddl labirintP1.pddl –heuristic "h=ff()" – search "astar(h)"
  Output:
  move-down a (1)
  move-down a (1)
  move-right a (1)
  move-right a (1)
  move-right a (1)

4.4 Problema 4

- Euristica h=cg()
  Comanda: /fast-downward.py labirint.pddl labirintP2.pddl –heuristic "h=cg()" – search "astar(h)"
  Output:
  move-down a (1)
  move-right a (1)
  move-right a (1)
  move-right a (1)
  move-right a (1)
  move-down a (1)
  move-right a (1)
  move-down a (1)
  move-down a (1)
  move-down a (1)
  move-left a (1)
  move-left a (1)
  move-left a (1)
  move-left a (1)
  move-left a (1)
  move-up a (1)
  move-up a (1)
  move-right a (1)
  move-right a (1)

5 Concluzie

In concluzie, acest proect a a avut scopul de a aprofunda dezvoltarea si planificarea oricarei probleme ce necesita o parte logica. Mai mult de atat m-a ajutat sa dezvolt o noua gandire de progtamare si de vizualizare a unei probleme cu ajutorul limbajului PDDL.