Inteligenta Artificiala - Tema 2
Planificare in jocul "Game About Squares"

Teodor-Stefan Dutu

Universitatea Politehnica Bucuresti
Facultatea de Automatica si Calculatoare
Grupa 341C3

Abstract. Tema propune crearea unui plan de rezolvarea unor niveluri ale jocului *Game About Squares*, pe care il trateaza atat ca pe o problema de cautare in spatiul starilor, cat si ca pe o problema de planificare modelata in `STRIPS` si `ADL`.

Cuprins

1 Cerinta 1 - Operatorii STRIPS

Pentru a putea crea planuri pentru rezolvarea nivelurilor acelori niveluri din `Game About Squares` care contin maximum 2 patrate, am definit urmatorii operatori.

1.1 Operatorul `move_changer`

Acest operator muta un patrat de la pozita sa la o pozitie adiacenta la care se gaseste un *schimbator de directie*, modificand si directia in care se va misca, pe viitor, patratul.

```
move_changer(P, DirP, Xp, Yp, Xs, Ys, DirS)
LP: pos(P, Xp, Yp) & schimbator(Xs, Ys, DirS) & delta(DirP, Xp, Yp, Xs, Ys)
    & empty(Xs, Ys) & dir(P, DirP)
LE: empty(Xs, Ys) & dir(P, DirP) & pos(P, Xp, Yp)
LA: empty(Xp, Yp) & dir(P, DirS) & pos(P, Xs, Ys)
```

1.2 Operatorul `move_empty`

Acest operator muta un patrat la o pozitie adiacenta daca pe aceasta nu se gasesc niciun alt patrat si niciun schimbator de directie.

```
move_empty(P, DirP, Xi, Yi, Xf, Yf)
LP: pos(P, Xi, Yi) & fara_schimbator(Xf, Yf) & delta(DirP, Xi, Yi, Xf, Yf)
    & empty(Xf, Yf) & dir(P, DirP)
LE: empty(Xf, Yf) & pos(P, Xi, Yi)
LA: empty(Xi, Yi) & pos(P, Xf, Yf)
```

1.3 Operatorul `move_square_empty_empty`

Acest operator muta un patrat la o pozitie unde se afla deja un alt patrat, ceea ce va duce la "impingerea" acestuia din urma, si trateaza cazul in care in niciuna dintre noile pozitii ale patratelor nu se gaseste un schimbator de directie.

```
move_square_empty_empty(P1, X1, Y1, DirP1, P2, X2, Y2, X3, Y3)
LP: pos(P1, X1, Y1) & pos(P2, X2, Y2) & fara_schimbator(X2, Y2)
    & fara_schimbator(X2, Y2) & delta(DirP1, X1, Y1, X2, Y2)
    & delta(DirP1, X2, Y2, X3, Y3) & empty(X3, Y3) & dir(P1, DirP1)
LE: pos(P1, X1, Y1) & pos(P2, X2, Y2) & empty(X3, Y3)
LA: pos(P1, X2, Y2) & pos(P2, X3, Y3) & empty(X1, Y1)
```

1.4 Operatorul `move_square_changer_empty`

Acest operator muta un patrat la o pozitie unde se afla deja un alt patrat, ceea ce va duce la "impingerea" acestuia din urma, si trateaza cazul in care pe pozitia patratului "impins" se gaseste un schimbator de directie.

```
move_square_changer_empty(P1, X1, Y1, DirP1, P2, X2, Y2, X3, Y3, DirP2)
LP: pos(P1, X1, Y1) & pos(P2, X2, Y2) & schimbator(X2, Y2)
    & fara_schimbator(X3, Y3) & delta(DirP1, X1, Y1, X2, Y2)
    & delta(DirP1, X2, Y2, X3, Y3) & empty(X3, Y3) & dir(P1, DirP1)
LE: pos(P1, X1, Y1) & pos(P2, X2, Y2) & empty(X3, Y3) & dir(P1, DirP1)
LA: pos(P1, X2, Y2) & pos(P2, X3, Y3) & empty(X1, Y1) & dir(P1, DirP2)
```

1.5 Operatorul `move_square_empty_changer`

Acest operator muta un patrat la o pozitie unde se afla deja un alt patrat, ceea ce va duce la "impingerea" acestuia din urma, si trateaza cazul in care pe pozitia finala a patratului "impins" se gaseste un schimbator de directie.

```
move_square_empty_changer(P1, X1, Y1, DirP1, P2, X2, Y2, DirP2, X3, Y3, DirS)
LP: pos(P1, X1, Y1) & pos(P2, X2, Y2) & schimbator(X3, Y3) & dir(P1, DirP1)
    & fara_schimbator(X2, Y2) & delta(DirP1, X1, Y1, X2, Y2)
    & delta(DirP1, X2, Y2, X3, Y3) & empty(X3, Y3) & dir(P2, DirP2)
LE: pos(P1, X1, Y1) & pos(P2, X2, Y2) & empty(X3, Y3) & dir(P2, DirP2)
LA: pos(P1, X2, Y2) & pos(P2, X3, Y3) & empty(X1, Y1) & dir(P2, DirS)
```

1.6 Operatorul `move_square_changer_changer`

Acest operator muta un patrat la o pozitie unde se afla deja un alt patrat, ceea ce va duce la "impingerea" acestuia din urma, si trateaza cazul in care pe pozitiile finale ale ambelor patrate se gasesc schimbatoare de directie.

```
move_square_changer_changer(P1, X1, Y1, DirP1, P2, X2, Y2, DirP2, X3, Y3, DirS3)
LP: pos(P1, X1, Y1) & pos(P2, X2, Y2) & schimbator(X3, Y3)
    & schimbator(X2, Y2) & delta(DirP1, X1, Y1, X2, Y2)
    & delta(DirP1, X2, Y2, X3, Y3) & empty(X3, Y3) & dir(P2, DirP2) & dir(P1, DirP1)
LE: pos(P1, X1, Y1) & pos(P2, X2, Y2) & empty(X3, Y3) & dir(P2, DirP2)
    & dir(P1, DirP1)
LA: pos(P1, X2, Y2) & pos(P2, X3, Y3) & empty(X1, Y1) & dir(P1, DirP2)
    & dir(P2, DirS3)
```

2 Cerinta 2 - Operatorul ADL

Am definiti operatorul de mai jos atat in ADL, cat si in codul Python care rezolva tema, cu mentiunea ca in Python a fost nevoie ca functia sa mai primeasca un parametru, state, prin care sa pot afla date despre mediu (patrate, pozitii, schimbatoare).

```
(:action move_square
    :parameters (?x ?y ?direction)
    :precondition (or (empty ?x ?y) (pos ?square ?x ?y))
    :effect (when (not (empty ?x ?y))
        (or
            (when (eq ?direction east)
                (and
                    (not (pos ?square ?x ?y))
                    (eq ?new_x (+ ?x 1))
                    (when (schimbator ?new_x ?y ?new_direction)
                        (dir ?square ?new_direction))
                    (move_square ?new_x ?y ?direction)
                    (pos ?square ?new_x ?y)))
            (when (eq ?direction north)
                (and
                    (not (pos ?square ?x ?y))
                    (eq ?new_y (+ ?y 1))
                    (when (schimbator ?x ?new_y ?new_direction)
                        (dir ?square ?new_direction))
                    (move_square ?x ?new_y ?direction)
                    (pos ?square ?x ?new_y)))
            (when (eq ?direction west)
                (and
                    (not (pos ?square ?x ?y))
                    (eq ?new_x (- ?x 1))
                    (when (schimbator ?new_x ?y ?new_direction)
                        (dir ?square ?new_direction))
                    (move_square ?new_x ?y ?direction)
                    (pos ?square ?new_x ?y)))
            (when (eq ?direction south)
                (and
                    (not (pos ?square ?x ?y))
                    (eq ?new_y (- ?y 1))
                    (when (schimbator ?x ?new_y ?new_direction)
                        (dir ?square ?new_direction))
                    (move_square ?x ?new_y ?direction)
                    (pos ?square ?x ?new_y))))))
```

3 Bonus - Micsorarea numarului de stari descoperite

Pentru a reduce numarul de stari descoperite am actionat pe doua planuri: pe de o parte am eliminat unele stari redundante sau din care nu se putea ajunge la solutie inca de la inceput, fara a le mai procesa, si pe de alta parte, am folosit o cautare in spatiul starilor de tip `Best-First`, deoarece nu conta calea de cost minim pana la solutie, ci gasirea cat mai rapida a acesteia.

Astfel, am utilizat urmatoarele 3 functii, doua pentru a decide daca o stare trebuie explorata sau nu, iar a treia fiind functia euristica aplicata in cadrul algoritmului `Best-First`.

3.1 Functia euristica

Functia folosita este **suma distantelor euclidiene de la fiecare patrat la scopul acestuia**. Distanta `Manhattan` pare mai potrivita in acest context, dat fiind ca miscarea patratelor se face exclusiv paralel cu *Ox* si *Oy*. Totusi, avand in vedere ca din cauza posibilitatii de a misca mai multe patrate apasand pe unul singur, distanta `Manhattan` nu este admisibila, deoarece poate supraestima numarul real de operatii necesare pentru a ajunge intr-o stare finala. Nici distanta `Euclid` nu este o euristica admisibila, dar dat fiind ca este dominata de cea `Manhattan`, supraestimeaza mai putin numarul real de operatii necesare pentru ca toate patratele sa ajunga in scopurile lor. Din acest motiv, euristica `Euclid` este mai precisa decat `Manhattan` si a fost aleasa astfel incat algoritmul `Best-First` folosit pentru gasirea solutiei sa prefere intotdeauna starile care au patratele cat mai aproape de scopurile lor.

Un contraexemplu care demonstreaza ca niciuna dintre euristicile antementionate nu este admisibila in cazul acestei probleme se gaseste chiar in primul nivel al jocului. Fie starea descrisa de predicatele:

```
pos(blue, 0, 2)
goal(blue, 0, 1)
dir(blue, south)
pos(red, 0, 1)
goal(red, 0, 0)
dir(red, north)
```

In acest caz, atat `h_euclid(state)` cat si `h_manhattan(state)` intorc valoarea 2, dar este suficienta o singura mutare pentru a plasa ambele patrate in starile lor-scop, si anume:
`move_square_empty_empty(blue, 0, 2, south, red, 0, 1)`.

3.2 Eliminarea starilor redundante

Am identificat doua tipuri de stari redundante si am creat functii pentru detectarea acestora pentru ca mai apoi sa le pot elimina. Aceste tipuri de stari sunt cele in care un patrat ajunge intr-o pozitie din care nu mai poate ajunge in pozitia sa scop si cele in care patratele se departeaza prea mult de *bounding box-ul* care cuprinde initial elementele din stare (patrate, schimbatoare de directie, scopuri).

In primul caz, elimin toate starile in care exista cel putin un patrat astfel incat in semiplanul inchis determinat de directia sa nu se gaseste nici scopul acestuia, nici un alt patrat, nici un schimbator cu o alta directie decat cea a patratului in cauza. In cel de-al doilea caz, elimin toate starile in care exista cel putin un patrat care iese in afara dreptunghiului de arie minima care contine toate elementele starii intiale.

Bibliografie

1. *Cursul de Inteligenta Artficiala: Cursul 7 - Planificare*
`https://curs.upb.ro/pluginfile.php/441622/mod_resource/content/1/IA_Lect_7_Planif.pdf`
Data ultimei accesari: 20 Dec 2020
2. *Planning Domain Definition Language - CSCI 431*
`http://csci431.artifice.cc/notes/pddl.html#conditional-preconditions-and-effects`
Data ultimei accesari: 20 Dec 2020