Department of Computer Science
Technical University of Cluj-Napoca

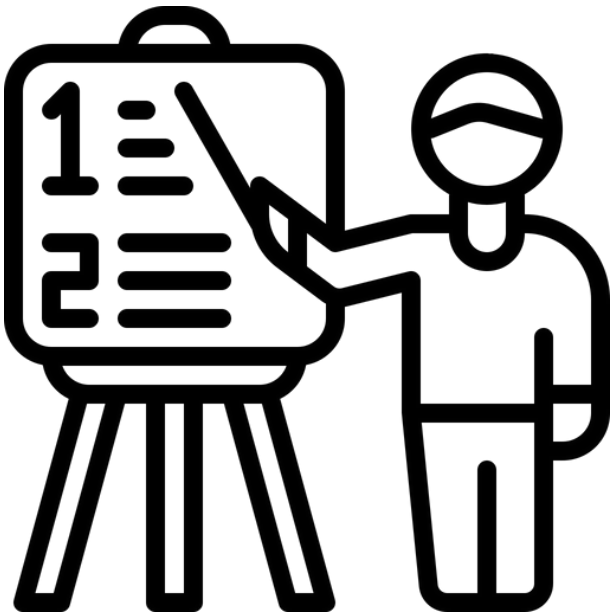
Artificial Intelligence

Laboratory activity

Human vs. AI – cine face planuri mai bune?

Nume: Cozma Victoria
Grupa: 30235
Email: victorya.cozma@gmail.com

Nume: Bouaru Radu
Grupa: 30233
Email: bouaru_radu@yahoo.com



Teaching Assistant: Roxana Ramona Szabo
szomiur@gmail.com

Cuprins

Acest document prezintă cum utilizarea unor unelte de planning precum PDDL şi diferite forme de a face planuri – care se apropie din ce în ce mai mult de planurile pe care le fac oamenii – conduc AI-ul în a imita acţiunile pe care le face un agent uman, şi în acelaşi timp a oferi probabil nişte rezolvări la unele probleme care nu sunt evidente pentru un agent uman.

$* * *$

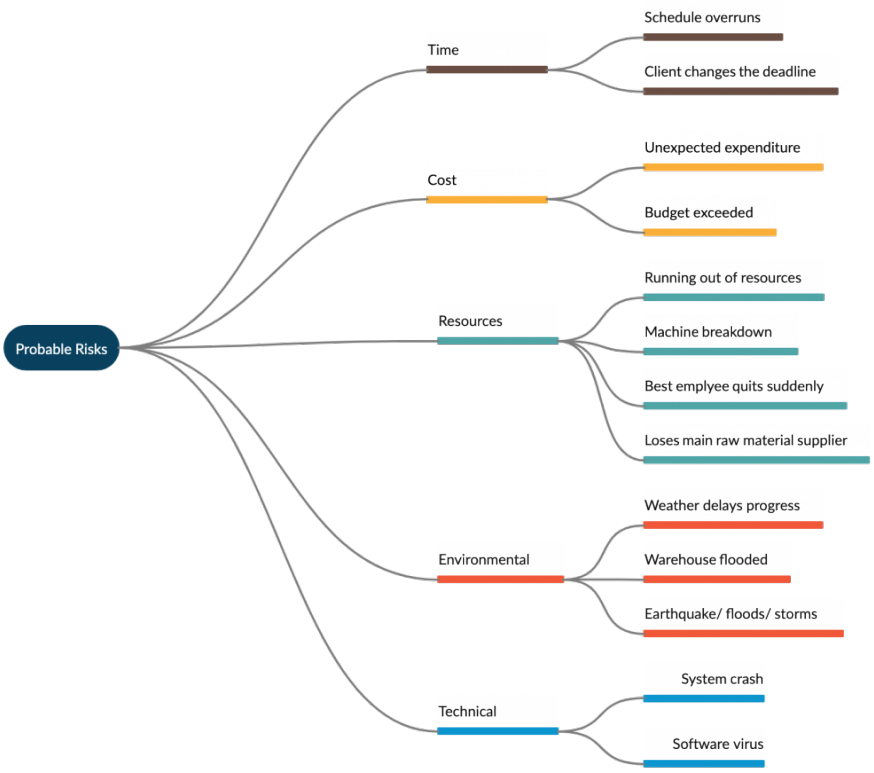
Warm-Up Party

Pentru a începe aventura noastră în lumea planningului, vom clarifica iniţial nişte termeni cu care ne vom întâlni pe tot parcursul acestui document.

Planningul este un domeniu al inteligenţei artificiale care utilizează tehnici automate pentru a rezolva probleme de programare şi crearea planurilor. O astfel de problemă implică o stare iniţială pe care dorim s-o transformăm în aplicaţie într-o stare finală, pe baza unui set de acţiuni alese automat.

Cele mai simple astfel de probleme au loc într-o lume **complet observabilă**, însă în lumea reală (care evident, nu poate avea acest atribut niciodată), planningul trebuie să mai ia în calcul şi alţi parametri pentru a ajunge la un rezultat finit conform aşteptărilor. Domeniile din lumea reală trebuie să ia în calcul observabilitatea parţială (situaţia iniţială poate fi specificată parţial) şi nondeterminismul (acţiunile pot avea anumite consecinţe, care duc la căi diferite de rezolvare a problemei). Astfel apare noţiunea de **conformant planning**, care ia în calcul atât incertitudinea în starea iniţială, cât şi faptul că acţiunile pot fi nondeterministe. Pe lângă acest conformant planning, apare şi noţiunea de **contingent planning** atunci când starea iniţială poate fi totuşi observată (spre exemplu, se primesc informaţii de la nişte senzori care ulterior sunt folosite pentru a determina cum evoluează crearea planului).

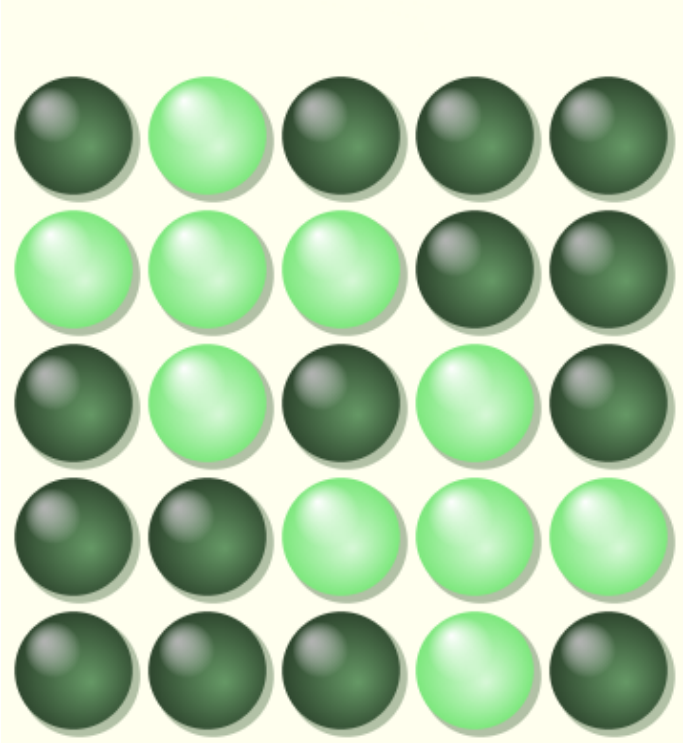
$$contingent\ planning\ = conformant\ planning\ +\ observations$$



Lights Out!

Jocul Lights Out! este un joc electronic care are un grid 5x5 de lumini. Când începe jocul, se aprind un anumit număr de lumini, bazându-se pe un pattern predefinit. Când se apasă pe o lumină, atât aceasta cât şi cele din jurul ei se sting, şi invers. Scopul final al puzzle-ului este să se stingă toate luminile de pe acest grid folosind cât mai puţine apăsări.

Tabla de joc arată astfel:



Această primă tablă este foarte uşor de rezolvat pentru un agent uman, deoarece se observă direct care sunt locurile unde trebuie să apăsăm ca să terminăm puzzle-ul. Însă pe o tablă mai complexă, acest lucru poate fi mai complicat de observat direct, deci domeniul în acest caz este parţial observabil. Desigur, putem să rezolvăm problema în câte click-uri dorim, dar implementând-o corespunzător, tool-ul ne va oferi un plan de a stinge aceste lumini în cât mai puţine click-uri de fiecare dată, indiferent de configuraţia iniţială a tablei.

Pentru a formaliza jocul în limbajul PDDL, trebuie să definim următoarele predicate cu care se va lucra mai departe:

```
is-light ?r ?c -> ne va spune daca campul de pe pozi ia (r,c) este sau nu aprins
on-line-(up, down, left, right) ?r ?c  -> ne va spune daca campul de pe pozitia (r,c)
                                          se afla pe prima linie sau ultima
linie, prima coloana sau ultima coloana    deoarece aceste campuri impreuna cu
                                         colturile vor avea nevoie de tratament
                                         special
```

Ulterior, se definesc acţiunile şi efectele specifice fiecărui câmp, iar aici identificăm 3 cazuri. După cum am menţionat anterior, trebuie să tratăm anumite câmpuri special, şi anume:
- dacă poziţia (r,c) de pe teren se află într-unul din colţuri, atunci aceasta va avea doar doi vecini cărora trebuie să li se schimbe starea
O astfel de acţiune are următorul template:

```
   (:action light-down-corner-up-right
    :parameters (?row ?col ?next-row ?prev-col)
    :precondition (and (is-light ?row ?col)
          (on-line-up ?row ?col)
          (on-line-right ?row ?col)
          (not (on-line-left ?row ?col))
          (not (on-line-down ?row ?col))
          (next-row ?row ?next-row)
          (next-col ?prev-col ?col))
    :effect (and (not (is-light ?row ?col))
        (when (is-light ?next-row ?col) (not (is-light ?next-row ?col)))
        (when (is-light ?row ?prev-col) (not (is-light ?row ?prev-col)))
        (when (not (is-light ?next-row ?col)) (is-light ?next-row ?col))
        (when (not (is-light ?row ?prev-col)) (is-light ?row ?prev-col))
    )
  )
```

- dacă poziţia (r,c) de pe teren se află pe una dintre liniile sau coloanele de pe margini exceptând colţurile, atunci aceasta va avea doar 3 vecini cărora să li se schimbe colţurile
O astfel de acţiune are următorul template:

```
  (:action light-down-line-up
    :parameters (?row ?col ?prev-col ?next-row ?next-col)
    :precondition (and (is-light ?row ?col)
          (on-line-up ?row ?col)
          (not (on-line-down ?row ?col))
          (not (on-line-left ?row ?col))
          (not (on-line-right ?row ?col))
          (next-row ?row ?next-row)
          (next-col ?prev-col ?col)
          (next-col ?col ?next-col))
    :effect (and (not (is-light ?row ?col))
        (when (is-light ?next-row ?col) (not (is-light ?next-row ?col)))
        (when (is-light ?row ?prev-col) (not (is-light ?row ?prev-col)))
        (when (is-light ?row ?next-col) (not (is-light ?row ?next-col)))
        (when (not (is-light ?next-row ?col)) (is-light ?next-row ?col))
        (when (not (is-light ?row ?prev-col)) (is-light ?row ?prev-col))
        (when (not (is-light ?row ?next-col)) (is-light ?row ?next-col))
    )
  )
```

- altfel, se folosesc acţiunile normale de light-up şi light-down, pentru restul poziţiilor din teren care au toţi 4 vecinii în jurul lor
O astfel de acţiune are următorul template:

```
(:action light-down
  :parameters (?row ?col ?prev-row ?next-row ?prev-col ?next-col)
  :precondition (and (is-light ?row ?col)
            (not (on-line-up ?row ?col))
        (not (on-line-down ?row ?col))
        (not (on-line-right ?row ?col))
        (not (on-line-left ?row ?col))
        (next-row ?prev-row ?row)
        (next-row ?row ?next-row)
        (next-col ?prev-col ?col)
        (next-col ?col ?next-col))
  :effect (and (not (is-light ?row ?col))
      (when (is-light ?prev-row ?col) (not (is-light ?prev-row ?col)))
      (when (is-light ?next-row ?col) (not (is-light ?next-row ?col)))
      (when (is-light ?row ?prev-col) (not (is-light ?row ?prev-col)))
      (when (is-light ?row ?next-col) (not (is-light ?row ?next-col)))
      (when (not (is-light ?prev-row ?col)) (is-light ?prev-row ?col))
      (when (not (is-light ?next-row ?col)) (is-light ?next-row ?col))
      (when (not (is-light ?row ?prev-col)) (is-light ?row ?prev-col))
      (when (not (is-light ?row ?next-col)) (is-light ?row ?next-col))
  )
)
```

Pentru diferitele configuraţii ale tablei am creat diferite fişiere cu stări pe care ni se va genera planul de ”stingere” a luminilor. Avem 6 astfel de fişiere. Comanda în terminalul Linux pentru obținerea unicului model al problemei este următoarea:

```
./Contingent-FF -o allOut/allOut_domain.pddl -f allOut/allOut_statesX.pddl
```

unde X = 1..6.

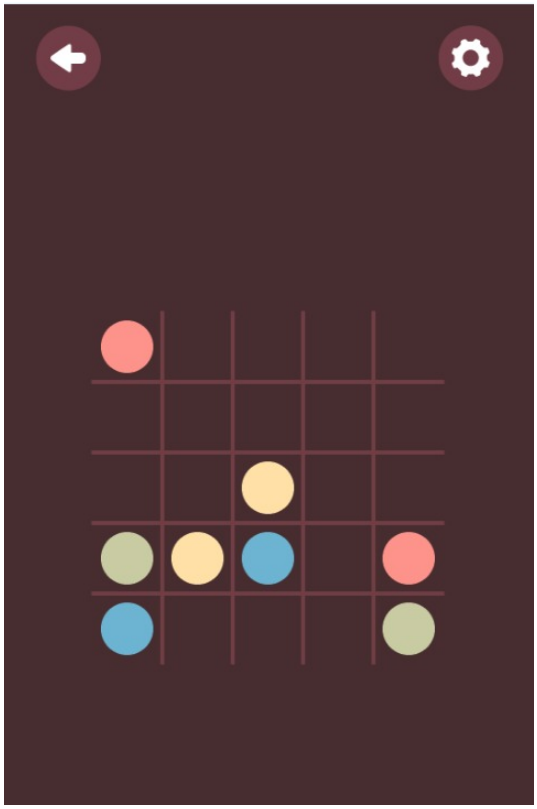
Soluţia pentru tabla prezentată mai sus arată în felul urmator

```
ff: found plan as follows
-------------------------------------------------
  0||0 --- LIGHT-DOWN ROW2 COL2 ROW1 ROW3 COL1 COL3 --- SON: 1||0
-------------------------------------------------
  1||0 --- LIGHT-DOWN ROW4 COL4 ROW3 ROW5 COL3 COL5 --- SON: 2||-1
-------------------------------------------------
```

Color Pipes

Color Pipes este un joc apărut pe piaţă în anul 2012. Acest joc implică un puzzle de tip numberlink, care este un tip de joc logic ce presupune identificarea unor căi care să conecteze numere într-un grid. Jucătorul trebuie să găsească perechi între toate numerele din grid, perechi desemnate printr-o linie continuă. Aceste linii nu pot să se intersecteze între ele, iar numerele trebuie să fie obligatoriu unul la un capăt al liniei şi celălalt număr la celălalt capăt al liniei. Se consideră că o problemă de acest tip este bine definită dacă are o soluţie unică şi toate celulele gridului sunt umplute. Jocul în cauză, Color Pipes, implementează ideea de numberlink, dar în loc de numere acesta foloseşte culori. În rest, ideea este exact aceeaşi: de a conecta culorile între ele corect, fără a lăsa spaţii libere pe teren.

Tabla de joc arată astfel:



Ca şi la jocul anterior, probabil primele nivele concepute sunt mai simple, însă pe măsură ce evoluăm în joc, dificultatea creşte din ce în ce mai mult. Pentru agentul uman, acest lucru înseamnă încercări repetate de a rezolva un nivel, ceea ce duce mai degrabă la ideea de machine-learning când o transpunem în lumea AI-ului. Însă bazându-ne pe planning, putem obţine din prima rezultatele corecte dacă avem

setul de predicate şi acţiuni complet şi corect definite. Astfel, vom implementa nivele cu cate 4 culori, pentru a prezenta puterea tool-ului de planning. Predicatele de care avem nevoie sunt următoarele:

```
(is-red ?r ?c) -> gasim culoarea rosu pe pozitia (r,c) pe grid
(is-green ?r ?c) -> gasim culoarea verde pe pozitia (r,c) pe grid
(is-blue ?r ?c) -> gasim culoarea albastru pe pozitia (r,c) pe grid
(is-yellow ?r ?c) -> gasim culoarea gablen pe pozitia (r,c) pe grid
```

Acţiunile care trebuie definite e să mutăm culorile pe cele 4 direcţii (nord-sud-est-vest). Vom exemplifica acum acţiunea pentru o singură culoare:

```
(:action move-red-down
  :parameters (?row ?col ?next-row)
  :precondition (and (is-red ?row ?col)
        (next-row ?row ?next-row)
                    (not (is-green ?next-row ?col))
                    (not (is-blue ?next-row ?col))
                    (not (is-yellow ?next-row ?col)))
  :effect (is-red ?next-row ?col)
)
(:action move-red-up
  :parameters (?row ?col ?prev-row)
  :precondition (and (is-red ?row ?col)
        (next-row ?prev-row ?row)
                    (not (is-green ?prev-row ?col))
                    (not (is-blue ?prev-row ?col))
                    (not (is-yellow ?prev-row ?col)))
  :effect (is-red ?prev-row ?col)
)
  (:action move-red-right
  :parameters (?row ?col ?next-col)
  :precondition (and (is-red ?row ?col)
        (next-col ?col ?next-col)
                    (not (is-green ?row ?next-col))
                    (not (is-blue ?row ?next-col))
                    (not (is-yellow ?row ?next-col)))
  :effect (is-red ?row ?next-col)
)
  (:action move-red-left
  :parameters (?row ?col ?prev-col)
  :precondition (and (is-red ?row ?col)
        (next-col ?prev-col ?col)
                    (not (is-green ?row ?prev-col))
                    (not (is-blue ?row ?prev-col))
                    (not (is-yellow ?row ?prev-col)))
  :effect (is-red ?row ?prev-col)
)
```

Comenzile în terminalul Linux pentru obținerea unicului model al problemei pentru fiecare stare sunt următoarele:

```
./fast-downward.py examples/colorPipes_domain.pddl examples/colorPipes_state1.pddl
    --heuristic "h=ff()" --search "astar(h)"
./fast-downward.py examples/colorPipes_domain.pddl examples/colorPipes_state2.pddl
    --heuristic "h=ff()" --search "astar(h)"
```

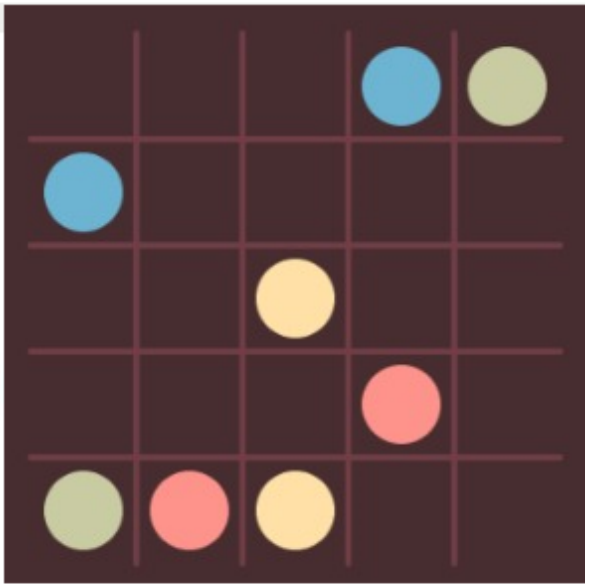
Soluţia pentru tabla prezentată mai sus arată în felul urmator

```
 1 (move-blue-right row5 col1 col2)
 2 (move-blue-right row5 col2 col3)
 3 (move-red-right row1 col1 col2)
 4 (move-red-right row1 col2 col3)
 5 (move-green-up row4 col1 row3)
 6 (move-red-right row1 col3 col4)
 7 (move-red-right row1 col4 col5)
 8 (move-red-down row1 col5 row2)
 9 (move-red-down row2 col5 row3)
10 (move-red-down row3 col5 row4)
11 (move-blue-up row5 col3 row4)
12 (move-green-up row3 col1 row2)
13 (move-green-right row2 col1 col2)
14 (move-green-right row2 col2 col3)
15 (move-yellow-up row4 col2 row3)
16 (move-yellow-right row3 col2 col3)
17 (move-green-right row2 col3 col4)
18 (move-green-down row2 col4 row3)
19 (move-green-down row3 col4 row4)
20 (move-green-down row4 col4 row5)
21 (move-green-right row5 col4 col5)
22 ; cost = 21 (unit cost)
```

,

Soluţia pentru o altă tablă:

```
 1 (move-blue-up row2 col1 row1)
 2 (move-blue-right row1 col1 col2)
 3 (move-green-up row5 col1 row4)
 4 (move-green-up row4 col1 row3)
 5 (move-green-right row3 col1 col2)
 6 (move-blue-right row1 col2 col3)
 7 (move-blue-right row1 col3 col4)
 8 (move-green-up row3 col2 row2)
 9 (move-green-right row2 col2 col3)
10 (move-red-up row5 col2 row4)
11 (move-green-right row2 col3 col4)
12 (move-green-right row2 col4 col5)
13 (move-green-up row2 col5 row1)
14 (move-yellow-right row3 col3 col4)
15 (move-yellow-right row3 col4 col5)
16 (move-yellow-down row3 col5 row4)
17 (move-yellow-down row4 col5 row5)
18 (move-yellow-left row5 col5 col4)
19 (move-yellow-left row5 col4 col3)
20 (move-red-right row4 col2 col3)
21 (move-red-right row4 col3 col4)
22 ; cost = 21 (unit cost)
```



Bomb It

Bomberman este un joc strategic bazat pe un grid, care presupune un roboţel care trebuie să navigheze printr-un labirint spre o locaţie finală şi să se ferească de bombele lăsate de alţi roboţei. De la versiunea de bază lansată în 1983, s-au dezvoltat ulterior o multitudine de jocuri care preiau această idee de bază. Unul dintre ele e şi Bomb It, unul dintre jocurile copilăriei noastre. Interfaţa acestui joculeţ aduce multe amintiri bune înapoi:

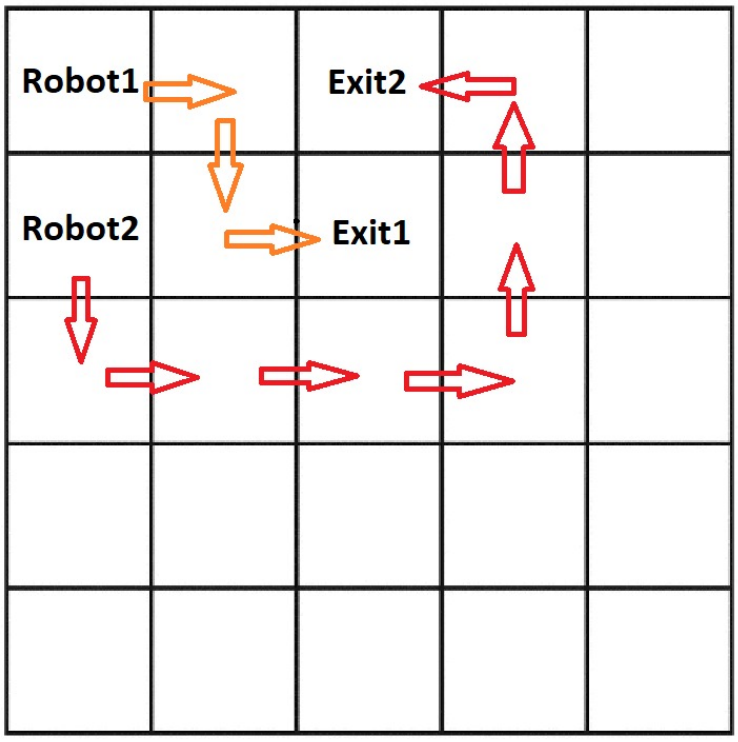
Această problemă deschide calea spre problemele nondeterminis. Vom implementa o idee de Bomb It care implică doi agenţi care pot să lase sau nu în urmă bombe pe teren pe unde se deplasează. Scopul acestor doi agenţi este să ajungă la ieşire, fără să calce într-o bombă lăsată de celălalt agent. Predicatele folosite sunt următoarele:

```
(agent ?a) -> Arata existenta unui agent in jocul nostru
(is-agent ?a ?r ?c) -> Verifica daca pe pozitia (r,c) din tabela exista sau nu
                                                agent
(bomb-it ?r ?c) -> Plaseaza o bomba pe pozitia (r,c) din tabela
```

Singurele acţiuni care sunt implementate în această secţiune sunt mişcările agentului pe cele 4 direcţii în grid (nord-sud-est-vest). Fiecare acţiune are câte un efect nondeterministic, şi anume de a lăsa o bombă prin locurile pe unde a trecut agentul. Agenţii îşi calculează traiectoriile pentru a ajunge la final ţinând cont de acest efect.

```
(:action move-agent-down
  :parameters (?a ?row ?col ?next-row)
  :precondition (and (is-agent ?a ?row ?col)
        (agent ?a)
        (next-row ?row ?next-row)
                    (not (bomb-it ?next-row ?col)))
  :effect (and (is-agent ?a  ?next-row ?col)
        (not (is-agent ?a ?row ?col))
        (nondet (bomb-it ?next-row ?col)))
)
```

Starea finală şi solutia a problemei arată în felul următor, iar traiectoriile rezultante au legătură cu efectul nondeterministic.



Comanda în terminalul Linux pentru obținerea unicului model de rezolvare al problemei este următoarea:

```
./Contingent-FF -o cff-tests/bomb_it/bomb_it_domain.pddl -f
    cff-tests/bomb_it/bomb_it_state.pddl
```

Poisoned cocktails problem

Această problemă este una dezvoltată după o idee personală. Problema spune că un agent trebuie să creeze cocktailuri. Fiecare cocktail va conţine rom şi apă, iar apa pentru fiecare cocktail va fi luată dintr-o sticlă care poate sau nu să conţină otravă. Agentul trebuie să se asigure că apa nu este otrăvită înainte să o pună în cocktail, însă el nu ştie care sticlă de apă este otrăvită. Acesta însă deţine antidotul pentru otrava respectivă. Aceasta problema are si ea efect nondeterministic, deoarece el nu stie in care sticla este otrava, si aceasta poate fi sau in sticla a doua, sau in sticla a treia.

Aceasta problema va avea trei predicate:

```
(is-water ?w) -> inseamna ca in pahar se afla apa
(is-rom ?r) -> inseamna ca in pahar se afla rom
(is-poisoned ?p) -> inseamna ca paharul este otravit sau nu
```

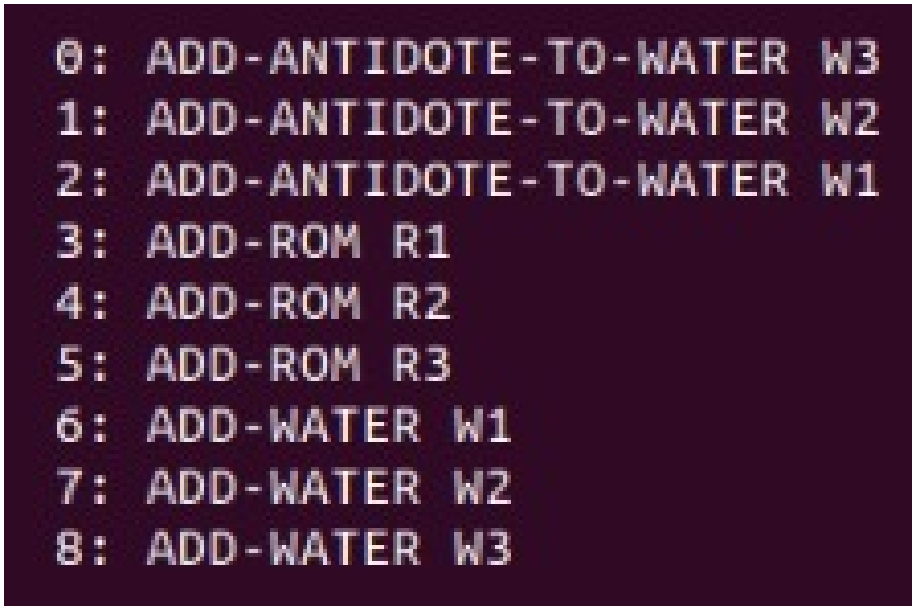
Actiunile pe care le poate face agentul de planning sunt, desigur, de a adauga apa intr-un pahar, de a adauga rom intr-un pahar si de a adauga antidotul in paharul in care s-a adaugat apa otravita. Singura actiune cu efect nondeterministic este adaugarea apei in pahar, care poate sau nu sa duca si la adaugarea otravei in pahar in acelasi timp.

Totodata, actiunea de a decontamina apa are loc doar atunci cand apa este otravita.

```
  (:action add-water
     :parameters  (?water)
     :precondition (and (is-water ?water)
                 (not (is-poisoned ?water)))
     :effect ( and (not (is-water ?water)))
  )

  (:action add-antidote-to-water
     :parameters  (?water)
     :precondition (is-water ?water)
     :effect (when (is-poisoned ?water) (not (is-poisoned ?water)))
 )
```

Plannerul va merge pe cea mai safe varianta dintre toate, astfel incat acesta va adauga la inceput antidotul tuturor paharelor, iar mai apoi va crea cocktailurile.

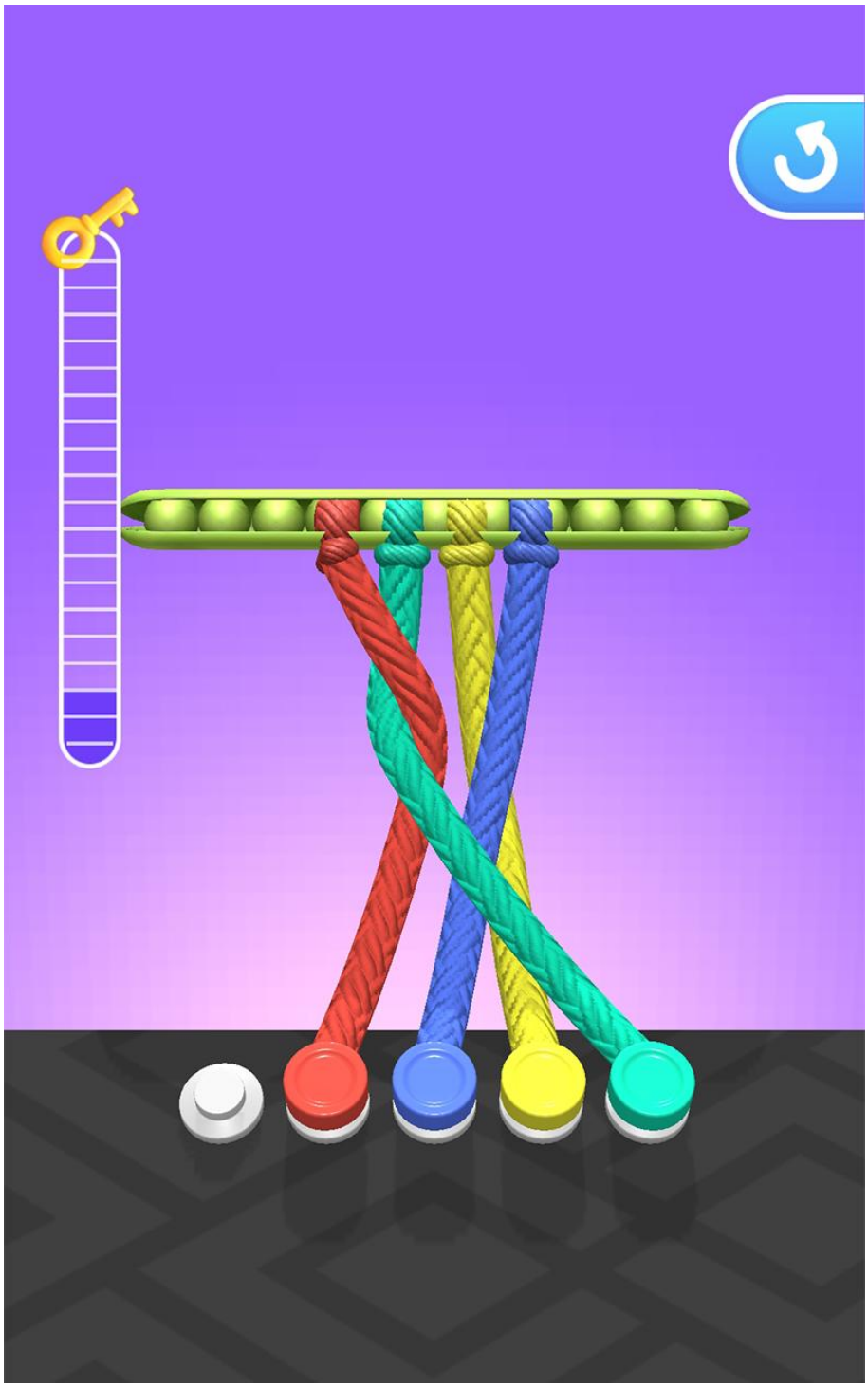


Comanda în terminalul Linux pentru obținerea unicului model de rezolvare al problemei este următoarea:

```
./Conformant-FF -o cff-tests/cocktails/cocktails_domain.pddl
                -f cff-tests/cocktails/cocktails_state.pddl
```

Unfinished business

Am mai incercat sa implementam o problema, Color ropes, care are un comportament mai dinamic, insa rezolvarea ei este inca in lucru. Scopul acestei probleme este de a dezlega toate cablurile ca sa nu contina noduri.



Singurele idei pe care le avem momentan pentru domeniu le vedeti in continuare:

```
(define (domain ropes)
   (:predicates (is-rope ?r)
                (is-table ?t)
                (empty-table ?t)
                (clear-rope ?r)
                (on ?r1 ?r2)
                ;(on-table ?r ?t)
                )

(:action move-from-rope
  :parameters (?table1 ?table2 ?rope1 ?rope2)
  :precondition (and   (is-table ?table1)
        (is-table ?table2)
        (is-rope ?rope1)
        (is-rope ?rope2)
        ;(on-table ?rope1 ?table1)
        ;(on-table ?rope2 ?table2)
        (not (empty-table ?table1))
        (empty-table ?table2)
        (clear-rope ?rope1)
        (not (clear-rope ?rope2))
        (on ?rope1 ?rope2))
  :effect (and  (not (empty-table ?table2))
           (empty-table ?table1)
              (clear-rope ?rope2)
              (not (on ?rope1 ?rope2)))
)
)
```

Bibliografie

https://jocuri.clopotel.ro/Jocuri$_{P}uzzle-16/color-pipes--HI7uFJZs$

$https://www.mathsisfun.com/games/allout.html$

$https://www.mathsisfun.com/numbers/number-block-freeplay.html$

$https://www.crazygames.ro/joc/bomb-it-7$

Anexă

Fișierul allOut_domain.pddl

```
(define (domain allOut)
(:predicates
  ( is-light ?r ?c )
  ( on-line-up ?r ?c)
  ( on-line-down ?r ?c)
  ( on-line-left ?r ?c)
  ( on-line-right ?r ?c)
  ( next-row ?r1 ?r2 )
  ( next-col ?c1 ?c2 )
)


  (:action light-down-line-up
    :parameters (?row ?col ?prev-col ?next-row ?next-col)
    :precondition (and (is-light ?row ?col)
          (on-line-up ?row ?col)
          (not (on-line-down ?row ?col))
          (not (on-line-left ?row ?col))
          (not (on-line-right ?row ?col))
          (next-row ?row ?next-row)
          (next-col ?prev-col ?col)
          (next-col ?col ?next-col))
    :effect (and (not (is-light ?row ?col))
```

```
        (when (is-light ?next-row ?col) (not (is-light ?next-row ?col)))
        (when (is-light ?row ?prev-col) (not (is-light ?row ?prev-col)))
        (when (is-light ?row ?next-col) (not (is-light ?row ?next-col)))
        (when (not (is-light ?next-row ?col)) (is-light ?next-row ?col))
        (when (not (is-light ?row ?prev-col)) (is-light ?row ?prev-col))
        (when (not (is-light ?row ?next-col)) (is-light ?row ?next-col))
    )
)

    (:action light-up-line-up
    :parameters (?row ?col ?prev-col ?next-row ?next-col)
    :precondition (and (not (is-light ?row ?col))
          (on-line-up ?row ?col)
          (not (on-line-down ?row ?col))
          (not (on-line-left ?row ?col))
          (not (on-line-right ?row ?col))
          (next-row ?row ?next-row)
          (next-col ?prev-col ?col)
          (next-col ?col ?next-col))
    :effect (and (is-light ?row ?col)
        (when (is-light ?next-row ?col) (not (is-light ?next-row ?col)))
        (when (is-light ?row ?prev-col) (not (is-light ?row ?prev-col)))
        (when (is-light ?row ?next-col) (not (is-light ?row ?next-col)))
        (when (not (is-light ?next-row ?col)) (is-light ?next-row ?col))
        (when (not (is-light ?row ?prev-col)) (is-light ?row ?prev-col))
        (when (not (is-light ?row ?next-col)) (is-light ?row ?next-col))
    )
)

    (:action light-down-line-down
    :parameters (?row ?col ?prev-col ?prev-row ?next-col)
```

```
  :precondition (and (is-light ?row ?col)
        (on-line-down ?row ?col)
        (not (on-line-up ?row ?col))
        (not (on-line-left ?row ?col))
        (not (on-line-right ?row ?col))
        (next-row ?prev-row ?row)
        (next-col ?prev-col ?col)
        (next-col ?col ?next-col))
  :effect (and (not (is-light ?row ?col))
      (when (is-light ?prev-row ?col) (not (is-light ?prev-row ?col)))
      (when (is-light ?row ?prev-col) (not (is-light ?row ?prev-col)))
      (when (is-light ?row ?next-col) (not (is-light ?row ?next-col)))
      (when (not (is-light ?prev-row ?col)) (is-light ?prev-row ?col))
      (when (not (is-light ?row ?prev-col)) (is-light ?row ?prev-col))
      (when (not (is-light ?row ?next-col)) (is-light ?row ?next-col))
  )
)

(:action light-up-line-down
  :parameters (?row ?col ?prev-col ?prev-row ?next-col)
  :precondition (and (not (is-light ?row ?col))
        (on-line-down ?row ?col)
        (not (on-line-up ?row ?col))
        (not (on-line-left ?row ?col))
        (not (on-line-right ?row ?col))
        (next-row ?prev-row ?row)
        (next-col ?prev-col ?col)
        (next-col ?col ?next-col))
  :effect (and (is-light ?row ?col)
      (when (is-light ?prev-row ?col) (not (is-light ?prev-row ?col)))
      (when (is-light ?row ?prev-col) (not (is-light ?row ?prev-col)))
```

```
      (when (is-light ?row ?next-col) (not (is-light ?row ?next-col)))
      (when (not (is-light ?prev-row ?col)) (is-light ?prev-row ?col))
      (when (not (is-light ?row ?prev-col)) (is-light ?row ?prev-col))
      (when (not (is-light ?row ?next-col)) (is-light ?row ?next-col))
  )
)

(:action light-down-line-left
  :parameters (?row ?col ?next-row ?prev-row ?next-col)
  :precondition (and (is-light ?row ?col)
        (on-line-left ?row ?col)
        (not (on-line-up ?row ?col))
        (not (on-line-down ?row ?col))
        (not (on-line-right ?row ?col))
        (next-row ?prev-row ?row)
        (next-col ?col ?next-col)
        (next-row ?row ?next-row))
  :effect (and (not (is-light ?row ?col))
      (when (is-light ?prev-row ?col) (not (is-light ?prev-row ?col)))
      (when (is-light ?row ?next-col) (not (is-light ?row ?next-col)))
      (when (is-light ?next-row ?col) (not (is-light ?next-row ?col)))
      (when (not (is-light ?prev-row ?col)) (is-light ?prev-row ?col))
      (when (not (is-light ?row ?next-col)) (is-light ?row ?next-col))
      (when (not (is-light ?next-row ?col)) (is-light ?next-row ?col))
  )
)

(:action light-up-line-left
  :parameters (?row ?col ?next-row ?prev-row ?next-col)
  :precondition (and (not (is-light ?row ?col))
        (on-line-left ?row ?col)
```

```
        (not (on-line-up ?row ?col))
        (not (on-line-down ?row ?col))
        (not (on-line-right ?row ?col))
        (next-row ?prev-row ?row)
        (next-col ?col ?next-col)
        (next-row ?row ?next-row))
  :effect (and (is-light ?row ?col)
      (when (is-light ?prev-row ?col) (not (is-light ?prev-row ?col)))
      (when (is-light ?row ?next-col) (not (is-light ?row ?next-col)))
      (when (is-light ?next-row ?col) (not (is-light ?next-row ?col)))
      (when (not (is-light ?prev-row ?col)) (is-light ?prev-row ?col))
      (when (not (is-light ?row ?next-col)) (is-light ?row ?next-col))
      (when (not (is-light ?next-row ?col)) (is-light ?next-row ?col))
  )
)

(:action light-down-line-right
  :parameters (?row ?col ?next-row ?prev-row ?prev-col)
  :precondition (and (is-light ?row ?col)
        (on-line-right ?row ?col)
        (not (on-line-up ?row ?col))
        (not (on-line-down ?row ?col))
        (not (on-line-left ?row ?col))
        (next-row ?prev-row ?row)
        (next-row ?row ?next-row)
        (next-col ?prev-col ?col))
  :effect (and (not (is-light ?row ?col))
      (when (is-light ?prev-row ?col) (not (is-light ?prev-row ?col)))
      (when (is-light ?row ?prev-col) (not (is-light ?row ?prev-col)))
      (when (is-light ?next-row ?col) (not (is-light ?next-row ?col)))
      (when (not (is-light ?prev-row ?col)) (is-light ?prev-row ?col))
```

```
        (when (not (is-light ?row ?prev-col)) (is-light ?row ?prev-col))
        (when (not (is-light ?next-row ?col)) (is-light ?next-row ?col))
    )
)

(:action light-up-line-right
    :parameters (?row ?col ?next-row ?prev-row ?prev-col)
    :precondition (and (not (is-light ?row ?col))
          (on-line-right ?row ?col)
          (not (on-line-up ?row ?col))
          (not (on-line-down ?row ?col))
          (not (on-line-left ?row ?col))
          (next-row ?prev-row ?row)
          (next-row ?row ?next-row)
          (next-col ?prev-col ?col))
    :effect (and (is-light ?row ?col)
        (when (is-light ?prev-row ?col) (not (is-light ?prev-row ?col)))
        (when (is-light ?row ?prev-col) (not (is-light ?row ?prev-col)))
        (when (is-light ?next-row ?col) (not (is-light ?next-row ?col)))
        (when (not (is-light ?prev-row ?col)) (is-light ?prev-row ?col))
        (when (not (is-light ?row ?prev-col)) (is-light ?row ?prev-col))
        (when (not (is-light ?next-row ?col)) (is-light ?next-row ?col))
    )
)

(:action light-down-corner-up-left
    :parameters (?row ?col ?next-row ?next-col)
    :precondition (and (is-light ?row ?col)
          (on-line-up ?row ?col)
          (on-line-left ?row ?col)
          (not (on-line-right ?row ?col))
```

```
        (not (on-line-down ?row ?col))
        (next-row ?row ?next-row)
        (next-col ?col ?next-col))
  :effect (and (not (is-light ?row ?col))
      (when (is-light ?next-row ?col) (not (is-light ?next-row ?col)))
      (when (is-light ?row ?next-col) (not (is-light ?row ?next-col)))
      (when (not (is-light ?next-row ?col)) (is-light ?next-row ?col))
      (when (not (is-light ?row ?next-col)) (is-light ?row ?next-col))
  )
)

 (:action light-up-corner-up-left
  :parameters (?row ?col ?next-row ?next-col)
  :precondition (and (not (is-light ?row ?col))
        (on-line-up ?row ?col)
        (on-line-left ?row ?col)
        (not (on-line-right ?row ?col))
        (not (on-line-down ?row ?col))
        (next-row ?row ?next-row)
        (next-col ?col ?next-col))
  :effect (and (is-light ?row ?col)
      (when (is-light ?next-row ?col) (not (is-light ?next-row ?col)))
      (when (is-light ?row ?next-col) (not (is-light ?row ?next-col)))
      (when (not (is-light ?next-row ?col)) (is-light ?next-row ?col))
      (when (not (is-light ?row ?next-col)) (is-light ?row ?next-col))
  )
)

 (:action light-down-corner-up-right
  :parameters (?row ?col ?next-row ?prev-col)
  :precondition (and (is-light ?row ?col)
```

```
        (on-line-up ?row ?col)
        (on-line-right ?row ?col)
        (not (on-line-left ?row ?col))
        (not (on-line-down ?row ?col))
        (next-row ?row ?next-row)
        (next-col ?prev-col ?col))
  :effect (and (not (is-light ?row ?col))
      (when (is-light ?next-row ?col) (not (is-light ?next-row ?col)))
      (when (is-light ?row ?prev-col) (not (is-light ?row ?prev-col)))
      (when (not (is-light ?next-row ?col)) (is-light ?next-row ?col))
      (when (not (is-light ?row ?prev-col)) (is-light ?row ?prev-col))
  )
)


 (:action light-up-corner-up-right
  :parameters (?row ?col ?next-row ?prev-col)
  :precondition (and (not (is-light ?row ?col))
        (on-line-up ?row ?col)
        (on-line-right ?row ?col)
        (not (on-line-left ?row ?col))
        (not (on-line-down ?row ?col))
        (next-row ?row ?next-row)
        (next-col ?prev-col ?col))
  :effect (and (is-light ?row ?col)
      (when (is-light ?next-row ?col) (not (is-light ?next-row ?col)))
      (when (is-light ?row ?prev-col) (not (is-light ?row ?prev-col)))
      (when (not (is-light ?next-row ?col)) (is-light ?next-row ?col))
      (when (not (is-light ?row ?prev-col)) (is-light ?row ?prev-col))
  )
)
```

```
(:action light-down-corner-down-right
  :parameters (?row ?col ?prev-row ?prev-col)
  :precondition (and (is-light ?row ?col)
        (on-line-down ?row ?col)
        (on-line-right ?row ?col)
        (not (on-line-left ?row ?col))
        (not (on-line-up ?row ?col))
        (next-row ?prev-row ?row)
        (next-col ?prev-col ?col))
  :effect (and (not (is-light ?row ?col))
      (when (is-light ?prev-row ?col) (not (is-light ?prev-row ?col)))
      (when (is-light ?row ?prev-col) (not (is-light ?row ?prev-col)))
      (when (not (is-light ?prev-row ?col)) (is-light ?prev-row ?col))
      (when (not (is-light ?row ?prev-col)) (is-light ?row ?prev-col))
  )
)

(:action light-up-corner-down-right
  :parameters (?row ?col ?prev-row ?prev-col)
  :precondition (and (not (is-light ?row ?col))
        (on-line-down ?row ?col)
        (on-line-right ?row ?col)
        (not (on-line-left ?row ?col))
        (not (on-line-up ?row ?col))
        (next-row ?prev-row ?row)
        (next-col ?prev-col ?col))
  :effect (and (is-light ?row ?col)
      (when (is-light ?prev-row ?col) (not (is-light ?prev-row ?col)))
      (when (is-light ?row ?prev-col) (not (is-light ?row ?prev-col)))
      (when (not (is-light ?prev-row ?col)) (is-light ?prev-row ?col))
      (when (not (is-light ?row ?prev-col)) (is-light ?row ?prev-col))
```

```
  )
)

(:action light-down-corner-down-left
  :parameters (?row ?col ?prev-row ?next-col)
  :precondition (and (is-light ?row ?col)
        (on-line-down ?row ?col)
        (on-line-left ?row ?col)
        (not (on-line-right ?row ?col))
        (not (on-line-up ?row ?col))
        (next-row ?prev-row ?row)
        (next-col ?col ?next-col))
  :effect (and (not (is-light ?row ?col))
      (when (is-light ?prev-row ?col) (not (is-light ?prev-row ?col)))
      (when (is-light ?row ?next-col) (not (is-light ?row ?next-col)))
      (when (not (is-light ?prev-row ?col)) (is-light ?prev-row ?col))
      (when (not (is-light ?row ?next-col)) (is-light ?row ?next-col))
  )
)

(:action light-up-corner-down-left
  :parameters (?row ?col ?prev-row ?next-col)
  :precondition (and (not (is-light ?row ?col))
        (on-line-down ?row ?col)
        (on-line-left ?row ?col)
        (not (on-line-right ?row ?col))
        (not (on-line-up ?row ?col))
        (next-row ?prev-row ?row)
        (next-col ?col ?next-col))
  :effect (and (is-light ?row ?col)
      (when (is-light ?prev-row ?col) (not (is-light ?prev-row ?col)))
```

```
      (when (is-light ?row ?next-col) (not (is-light ?row ?next-col)))
      (when (not (is-light ?prev-row ?col)) (is-light ?prev-row ?col))
      (when (not (is-light ?row ?next-col)) (is-light ?row ?next-col))
  )
)

(:action light-up
  :parameters (?row ?col ?prev-row ?next-row ?prev-col ?next-col)
  :precondition (and (not (is-light ?row ?col))
        (not (on-line-up ?row ?col))
        (not (on-line-down ?row ?col))
        (not (on-line-right ?row ?col))
        (not (on-line-left ?row ?col))
        (next-row ?prev-row ?row)
        (next-row ?row ?next-row)
        (next-col ?prev-col ?col)
        (next-col ?col ?next-col))
  :effect (and (is-light ?row ?col)
      (when (is-light ?prev-row ?col) (not (is-light ?prev-row ?col)))
      (when (is-light ?next-row ?col) (not (is-light ?next-row ?col)))
      (when (is-light ?row ?prev-col) (not (is-light ?row ?prev-col)))
      (when (is-light ?row ?next-col) (not (is-light ?row ?next-col)))
      (when (not (is-light ?prev-row ?col)) (is-light ?prev-row ?col))
      (when (not (is-light ?next-row ?col)) (is-light ?next-row ?col))
      (when (not (is-light ?row ?prev-col)) (is-light ?row ?prev-col))
      (when (not (is-light ?row ?next-col)) (is-light ?row ?next-col))
  )
)

(:action light-down
  :parameters (?row ?col ?prev-row ?next-row ?prev-col ?next-col)
```

```
    :precondition (and (is-light ?row ?col)
                (not (on-line-up ?row ?col))
           (not (on-line-down ?row ?col))
           (not (on-line-right ?row ?col))
           (not (on-line-left ?row ?col))
           (next-row ?prev-row ?row)
           (next-row ?row ?next-row)
           (next-col ?prev-col ?col)
           (next-col ?col ?next-col))
    :effect (and (not (is-light ?row ?col))
        (when (is-light ?prev-row ?col) (not (is-light ?prev-row ?col)))
        (when (is-light ?next-row ?col) (not (is-light ?next-row ?col)))
        (when (is-light ?row ?prev-col) (not (is-light ?row ?prev-col)))
        (when (is-light ?row ?next-col) (not (is-light ?row ?next-col)))
        (when (not (is-light ?prev-row ?col)) (is-light ?prev-row ?col))
        (when (not (is-light ?next-row ?col)) (is-light ?next-row ?col))
        (when (not (is-light ?row ?prev-col)) (is-light ?row ?prev-col))
        (when (not (is-light ?row ?next-col)) (is-light ?row ?next-col))
    )
  )

  (:action senseLIGHT
    :parameters (?row ?col)
    :observe (is-light ?row ?col))

)
```

Fișierele cu cele 6 nivele implementate pentru problema Lights Out!

```
(define (problem allOut)
  (:domain allOut)
  (:objects
```

```
    row1 row2 row3 row4 row5
    col1 col2 col3 col4 col5)
  (:init
    (next-row row1 row2)          (next-col col1 col2)
    (next-row row2 row3)          (next-col col2 col3)
    (next-row row3 row4)          (next-col col3 col4)
    (next-row row4 row5)          (next-col col4 col5)
    (is-light row1 col2)          (is-light row2 col1)
    (is-light row2 col2)          (is-light row2 col3)
    (is-light row3 col2)          (is-light row3 col4)
    (is-light row4 col3)          (is-light row4 col4)
    (is-light row4 col5)          (is-light row5 col4))

  (:goal
   (and
      (not(is-light row1 col1)) (not(is-light row3 col4))
      (not(is-light row1 col2)) (not(is-light row3 col5))
      (not(is-light row1 col3)) (not(is-light row4 col1))
      (not(is-light row1 col4)) (not(is-light row4 col2))
      (not(is-light row1 col5)) (not(is-light row4 col3))
      (not(is-light row2 col1)) (not(is-light row4 col4))
      (not(is-light row2 col1)) (not(is-light row4 col5))
      (not(is-light row2 col2)) (not(is-light row5 col1))
      (not(is-light row2 col3)) (not(is-light row5 col2))
      (not(is-light row2 col5)) (not(is-light row5 col3))
      (not(is-light row3 col1)) (not(is-light row5 col4))
      (not(is-light row3 col2)) (not(is-light row5 col5))
      (not(is-light row3 col3))
      )))
```

```
(define (problem allOut)
  (:domain allOut)
```

```
(:objects
   row1 row2 row3 row4 row5
   col1 col2 col3 col4 col5)
(:init
   (next-row row1 row2)          (next-col col1 col2)
   (next-row row2 row3)          (next-col col2 col3)
   (next-row row3 row4)          (next-col col3 col4)
   (next-row row4 row5)          (next-col col4 col5)

   (on-line-up row1 col1)   (on-line-down row5 col1)
   (on-line-up row1 col2)   (on-line-down row5 col2)
   (on-line-up row1 col3)   (on-line-down row5 col3)
   (on-line-up row1 col4)   (on-line-down row5 col4)
   (on-line-up row1 col5)   (on-line-down row5 col5)

   (on-line-right row1 col5)  (on-line-left row1 col1)
   (on-line-right row2 col5)  (on-line-left row2 col1)
   (on-line-right row3 col5)  (on-line-left row4 col1)
   (on-line-right row4 col5)  (on-line-left row4 col1)
   (on-line-right row5 col5)  (on-line-left row5 col1)



   (is-light row4 col3)
   (is-light row4 col4)
   (is-light row4 col5)
   (is-light row5 col2)
   (is-light row5 col4)
 )



(:goal
```

```
   (and
      (not(is-light row1 col1)) (not(is-light row3 col4))
      (not(is-light row1 col2)) (not(is-light row3 col5))
      (not(is-light row1 col3)) (not(is-light row4 col1))
      (not(is-light row1 col4)) (not(is-light row4 col2))
      (not(is-light row1 col5)) (not(is-light row4 col3))
      (not(is-light row2 col1)) (not(is-light row4 col4))
      (not(is-light row2 col2)) (not(is-light row4 col5))
      (not(is-light row2 col3)) (not(is-light row5 col1))
      (not(is-light row2 col4)) (not(is-light row5 col2))
      (not(is-light row2 col5)) (not(is-light row5 col3))
      (not(is-light row3 col1)) (not(is-light row5 col4))
      (not(is-light row3 col2)) (not(is-light row5 col5))
      (not(is-light row3 col3))
      )))
```

```
(define (problem allOut)
  (:domain allOut)
  (:objects
     row1 row2 row3 row4 row5
     col1 col2 col3 col4 col5)
  (:init
     (next-row row1 row2)          (next-col col1 col2)
     (next-row row2 row3)          (next-col col2 col3)
     (next-row row3 row4)          (next-col col3 col4)
     (next-row row4 row5)          (next-col col4 col5)

     (on-line-up row1 col1)   (on-line-down row5 col1)
     (on-line-up row1 col2)   (on-line-down row5 col2)
     (on-line-up row1 col3)   (on-line-down row5 col3)
     (on-line-up row1 col4)   (on-line-down row5 col4)
     (on-line-up row1 col5)   (on-line-down row5 col5)
```

```
   (on-line-right row1 col5)  (on-line-left row1 col1)
   (on-line-right row2 col5)  (on-line-left row2 col1)
   (on-line-right row3 col5)  (on-line-left row4 col1)
   (on-line-right row4 col5)  (on-line-left row4 col1)
   (on-line-right row5 col5)  (on-line-left row5 col1)

   (is-light row1 col1)
   (is-light row1 col3)
   (is-light row1 col5)

   (is-light row2 col1)
   (is-light row2 col3)
   (is-light row2 col5)

   (is-light row4 col1)
   (is-light row4 col3)
   (is-light row4 col5)

   (is-light row5 col1)
   (is-light row5 col3)
   (is-light row5 col5)
 )


(:goal
 (and
    (not(is-light row1 col1)) (not(is-light row3 col4))
    (not(is-light row1 col2)) (not(is-light row3 col5))
    (not(is-light row1 col3)) (not(is-light row4 col1))
    (not(is-light row1 col4)) (not(is-light row4 col2))
```

```
     (not(is-light row1 col5)) (not(is-light row4 col3))
     (not(is-light row2 col1)) (not(is-light row4 col4))
     (not(is-light row2 col2)) (not(is-light row4 col5))
     (not(is-light row2 col3)) (not(is-light row5 col1))
     (not(is-light row2 col4)) (not(is-light row5 col2))
     (not(is-light row2 col5)) (not(is-light row5 col3))
     (not(is-light row3 col1)) (not(is-light row5 col4))
     (not(is-light row3 col2)) (not(is-light row5 col5))
     (not(is-light row3 col3))
     )))
```

```
(define (problem allOut)
  (:domain allOut)
  (:objects
     row1 row2 row3 row4 row5
     col1 col2 col3 col4 col5)
  (:init
     (next-row row1 row2)          (next-col col1 col2)
     (next-row row2 row3)          (next-col col2 col3)
     (next-row row3 row4)          (next-col col3 col4)
     (next-row row4 row5)          (next-col col4 col5)

     (on-line-up row1 col1)   (on-line-down row5 col1)
     (on-line-up row1 col2)   (on-line-down row5 col2)
     (on-line-up row1 col3)   (on-line-down row5 col3)
     (on-line-up row1 col4)   (on-line-down row5 col4)
     (on-line-up row1 col5)   (on-line-down row5 col5)

     (on-line-right row1 col5)  (on-line-left row1 col1)
     (on-line-right row2 col5)  (on-line-left row2 col1)
     (on-line-right row3 col5)  (on-line-left row4 col1)
     (on-line-right row4 col5)  (on-line-left row4 col1)
```

```
   (on-line-right row5 col5)  (on-line-left row5 col1)


   (is-light row1 col1)
   (is-light row1 col3)
   (is-light row1 col5)

   (is-light row2 col1)
   (is-light row2 col3)
   (is-light row2 col5)

   (is-light row4 col1)
   (is-light row4 col3)
   (is-light row4 col5)

   (is-light row5 col1)
   (is-light row5 col3)
   (is-light row5 col5)
 )



(:goal
 (and
    (not(is-light row1 col1)) (not(is-light row3 col4))
    (not(is-light row1 col2)) (not(is-light row3 col5))
    (not(is-light row1 col3)) (not(is-light row4 col1))
    (not(is-light row1 col4)) (not(is-light row4 col2))
    (not(is-light row1 col5)) (not(is-light row4 col3))
    (not(is-light row2 col1)) (not(is-light row4 col4))
    (not(is-light row2 col2)) (not(is-light row4 col5))
    (not(is-light row2 col3)) (not(is-light row5 col1))
    (not(is-light row2 col4)) (not(is-light row5 col2))
```

```
     (not(is-light row2 col5)) (not(is-light row5 col3))
     (not(is-light row3 col1)) (not(is-light row5 col4))
     (not(is-light row3 col2)) (not(is-light row5 col5))
     (not(is-light row3 col3))
     )))
```

```
(define (problem allOut)
  (:domain allOut)
  (:objects
     row1 row2 row3 row4 row5
     col1 col2 col3 col4 col5)
  (:init
     (next-row row1 row2)          (next-col col1 col2)
     (next-row row2 row3)          (next-col col2 col3)
     (next-row row3 row4)          (next-col col3 col4)
     (next-row row4 row5)          (next-col col4 col5)

     (on-line-up row1 col1)   (on-line-down row5 col1)
     (on-line-up row1 col2)   (on-line-down row5 col2)
     (on-line-up row1 col3)   (on-line-down row5 col3)
     (on-line-up row1 col4)   (on-line-down row5 col4)
     (on-line-up row1 col5)   (on-line-down row5 col5)

     (on-line-right row1 col5)  (on-line-left row1 col1)
     (on-line-right row2 col5)  (on-line-left row2 col1)
     (on-line-right row3 col5)  (on-line-left row4 col1)
     (on-line-right row4 col5)  (on-line-left row4 col1)
     (on-line-right row5 col5)  (on-line-left row5 col1)

     (is-light row2 col1)
     (is-light row3 col1)
     (is-light row4 col1)
```

```
   (is-light row1 col2)
   (is-light row2 col2)
   (is-light row3 col2)
   (is-light row4 col2)
   (is-light row5 col2)

   (is-light row1 col4)
   (is-light row2 col4)
   (is-light row3 col4)
   (is-light row4 col4)
   (is-light row5 col4)

   (is-light row2 col5)
   (is-light row3 col5)
   (is-light row4 col5)
 )


(:goal
 (and
    (not(is-light row1 col1)) (not(is-light row3 col4))
    (not(is-light row1 col2)) (not(is-light row3 col5))
    (not(is-light row1 col3)) (not(is-light row4 col1))
    (not(is-light row1 col4)) (not(is-light row4 col2))
    (not(is-light row1 col5)) (not(is-light row4 col3))
    (not(is-light row2 col1)) (not(is-light row4 col4))
    (not(is-light row2 col2)) (not(is-light row4 col5))
    (not(is-light row2 col3)) (not(is-light row5 col1))
    (not(is-light row2 col4)) (not(is-light row5 col2))
    (not(is-light row2 col5)) (not(is-light row5 col3))
```

```
     (not(is-light row3 col1)) (not(is-light row5 col4))
     (not(is-light row3 col2)) (not(is-light row5 col5))
     (not(is-light row3 col3))
     )))
```

```
(define (problem allOut)
  (:domain allOut)
  (:objects
     row1 row2 row3 row4 row5
     col1 col2 col3 col4 col5)
  (:init
     (next-row row1 row2)          (next-col col1 col2)
     (next-row row2 row3)          (next-col col2 col3)
     (next-row row3 row4)          (next-col col3 col4)
     (next-row row4 row5)          (next-col col4 col5)

     (on-line-up row1 col1)   (on-line-down row5 col1)
     (on-line-up row1 col2)   (on-line-down row5 col2)
     (on-line-up row1 col3)   (on-line-down row5 col3)
     (on-line-up row1 col4)   (on-line-down row5 col4)
     (on-line-up row1 col5)   (on-line-down row5 col5)

     (on-line-right row1 col5)  (on-line-left row1 col1)
     (on-line-right row2 col5)  (on-line-left row2 col1)
     (on-line-right row3 col5)  (on-line-left row4 col1)
     (on-line-right row4 col5)  (on-line-left row4 col1)
     (on-line-right row5 col5)  (on-line-left row5 col1)

     (is-light row2 col1)
     (is-light row2 col2)
     (is-light row2 col4)
     (is-light row2 col5)
```

```
    (is-light row4 col1)
    (is-light row4 col5)
    (is-light row5 col1)
    (is-light row5 col2)
    (is-light row5 col4)
    (is-light row5 col5)
  )


 (:goal
  (and
     (not(is-light row1 col1)) (not(is-light row3 col4))
     (not(is-light row1 col2)) (not(is-light row3 col5))
     (not(is-light row1 col3)) (not(is-light row4 col1))
     (not(is-light row1 col4)) (not(is-light row4 col2))
     (not(is-light row1 col5)) (not(is-light row4 col3))
     (not(is-light row2 col1)) (not(is-light row4 col4))
     (not(is-light row2 col2)) (not(is-light row4 col5))
     (not(is-light row2 col3)) (not(is-light row5 col1))
     (not(is-light row2 col4)) (not(is-light row5 col2))
     (not(is-light row2 col5)) (not(is-light row5 col3))
     (not(is-light row3 col1)) (not(is-light row5 col4))
     (not(is-light row3 col2)) (not(is-light row5 col5))
     (not(is-light row3 col3))
     )))
```

Fișierul colorPipes_domain.pddl

```
(define (domain colorPipes)


 (:predicates
```

```
 (is-red ?r ?c)
 (is-green ?r ?c)
 (is-blue ?r ?c)
 (is-yellow ?r ?c)
 (next-row ?r1 ?r2)
 (next-col ?c1 ?c2)
)

(:action move-red-down
  :parameters (?row ?col ?next-row)
  :precondition (and (is-red ?row ?col)
        (next-row ?row ?next-row)
                    (not (is-green ?next-row ?col))
                    (not (is-blue ?next-row ?col))
                    (not (is-yellow ?next-row ?col)))
  :effect (is-red ?next-row ?col)
)

(:action move-green-down
  :parameters (?row ?col ?next-row)
  :precondition (and (is-green ?row ?col)
        (next-row ?row ?next-row)
                    (not (is-red ?next-row ?col))
                    (not (is-blue ?next-row ?col))
                    (not (is-yellow ?next-row ?col)))
  :effect (is-green ?next-row ?col)
)

(:action move-blue-down
  :parameters (?row ?col ?next-row)
  :precondition (and (is-blue ?row ?col)
```

```
        (next-row ?row ?next-row)
                    (not (is-green ?next-row ?col))
                    (not (is-red ?next-row ?col))
                    (not (is-yellow ?next-row ?col)))
  :effect (is-blue ?next-row ?col)
)

(:action move-yellow-down
  :parameters (?row ?col ?next-row)
  :precondition (and (is-yellow ?row ?col)
        (next-row ?row ?next-row)
                    (not (is-green ?next-row ?col))
                    (not (is-blue ?next-row ?col))
                    (not (is-red ?next-row ?col)))
  :effect (is-yellow ?next-row ?col)
)

(:action move-red-up
  :parameters (?row ?col ?prev-row)
  :precondition (and (is-red ?row ?col)
        (next-row ?prev-row ?row)
                    (not (is-green ?prev-row ?col))
                    (not (is-blue ?prev-row ?col))
                    (not (is-yellow ?prev-row ?col)))
  :effect (is-red ?prev-row ?col)
)

(:action move-green-up
  :parameters (?row ?col ?prev-row)
  :precondition (and (is-green ?row ?col)
        (next-row ?prev-row ?row)
```

```
                    (not (is-red ?prev-row ?col))
                    (not (is-blue ?prev-row ?col))
                    (not (is-yellow ?prev-row ?col)))
  :effect (is-green ?prev-row ?col)
)

(:action move-blue-up
  :parameters (?row ?col ?prev-row)
  :precondition (and (is-blue ?row ?col)
        (next-row ?prev-row ?row)
                    (not (is-green ?prev-row ?col))
                    (not (is-red ?prev-row ?col))
                    (not (is-yellow ?prev-row ?col)))
  :effect (is-blue ?prev-row ?col)
)

(:action move-yellow-up
  :parameters (?row ?col ?prev-row)
  :precondition (and (is-yellow ?row ?col)
        (next-row ?prev-row ?row)
                    (not (is-green ?prev-row ?col))
                    (not (is-blue ?prev-row ?col))
                    (not (is-red ?prev-row ?col)))
  :effect (is-yellow ?prev-row ?col)
)

(:action move-red-right
  :parameters (?row ?col ?next-col)
  :precondition (and (is-red ?row ?col)
        (next-col ?col ?next-col)
                    (not (is-green ?row ?next-col))
```

```
                    (not (is-blue ?row ?next-col))
                    (not (is-yellow ?row ?next-col)))
  :effect (is-red ?row ?next-col)
)

(:action move-green-right
  :parameters (?row ?col ?next-col)
  :precondition (and (is-green ?row ?col)
        (next-col ?col ?next-col)
                    (not (is-red ?row ?next-col))
                    (not (is-blue ?row ?next-col))
                    (not (is-yellow ?row ?next-col)))
  :effect (is-green ?row ?next-col)
)

(:action move-blue-right
  :parameters (?row ?col ?next-col)
  :precondition (and (is-blue ?row ?col)
        (next-col ?col ?next-col)
                    (not (is-green ?row ?next-col))
                    (not (is-red ?row ?next-col))
                    (not (is-yellow ?row ?next-col)))
  :effect (is-blue ?row ?next-col)
)

(:action move-yellow-right
  :parameters (?row ?col ?next-col)
  :precondition (and (is-yellow ?row ?col)
        (next-col ?col ?next-col)
                    (not (is-green ?row ?next-col))
                    (not (is-blue ?row ?next-col))
```

```
                    (not (is-red ?row ?next-col)))
  :effect (is-yellow ?row ?next-col)
)

(:action move-red-left
  :parameters (?row ?col ?prev-col)
  :precondition (and (is-red ?row ?col)
        (next-col ?prev-col ?col)
                    (not (is-green ?row ?prev-col))
                    (not (is-blue ?row ?prev-col))
                    (not (is-yellow ?row ?prev-col)))
  :effect (is-red ?row ?prev-col)
)

(:action move-green-left
  :parameters (?row ?col ?prev-col)
  :precondition (and (is-green ?row ?col)
        (next-col ?prev-col ?col)
                    (not (is-red ?row ?prev-col))
                    (not (is-blue ?row ?prev-col))
                    (not (is-yellow ?row ?prev-col)))
  :effect (is-green ?row ?prev-col)
)

  (:action move-blue-left
  :parameters (?row ?col ?prev-col)
  :precondition (and (is-blue ?row ?col)
        (next-col ?prev-col ?col)
                    (not (is-green ?row ?prev-col))
                    (not (is-red ?row ?prev-col))
                    (not (is-yellow ?row ?prev-col)))
```

```
    :effect (is-blue ?row ?prev-col)
  )


    (:action move-yellow-left
    :parameters (?row ?col ?prev-col)
    :precondition (and (is-yellow ?row ?col)
          (next-col ?prev-col ?col)
                        (not (is-green ?row ?prev-col))
                        (not (is-blue ?row ?prev-col))
                        (not (is-red ?row ?prev-col)))
    :effect (is-yellow ?row ?prev-col)
  )
)
```

Fisierele pentru starile la Color Pipes

```
(define (problem colorPipes)
  (:domain colorPipes)
  (:objects
     row1 row2 row3 row4 row5
     col1 col2 col3 col4 col5)
  (:init
     (next-row row1 row2)          (next-col col1 col2)
     (next-row row2 row3)          (next-col col2 col3)
     (next-row row3 row4)          (next-col col3 col4)
     (next-row row4 row5)          (next-col col4 col5)


     (is-red row1 col1)            (is-green row4 col1)
     (is-blue row5 col1)          (is-yellow row4 col2))
  (:goal
    ( and
      (is-red row4 col5)
```

```
      (is-green row5 col5)
      (is-blue row4 col3)
      (is-yellow row3 col3)
    )
  )
)
```

```
(define (problem colorPipes)
  (:domain colorPipes)
  (:objects
     row1 row2 row3 row4 row5
     col1 col2 col3 col4 col5)
  (:init
     (next-row row1 row2)          (next-col col1 col2)
     (next-row row2 row3)          (next-col col2 col3)
     (next-row row3 row4)          (next-col col3 col4)
     (next-row row4 row5)          (next-col col4 col5)

     (is-red row5 col2)            (is-green row5 col1)
     (is-blue row2 col1)          (is-yellow row3 col3))
  (:goal
    ( and
      (is-red row4 col4)
      (is-green row1 col5)
      (is-blue row1 col4)
      (is-yellow row5 col3)
    )
  )
)
```

Fisierul bomb_it_domain.pddl

```
(define (domain BombIt)
```

```
(:predicates
 (agent ?a)
 (is-agent ?a ?r ?c)
 (bomb-it ?r ?c)
 (next-row ?r1 ?r2)
 (next-col ?c1 ?c2)
)

(:action move-agent-down
  :parameters (?a ?row ?col ?next-row)
  :precondition (and (is-agent ?a ?row ?col)
        (agent ?a)
        (next-row ?row ?next-row)
                    (not (bomb-it ?next-row ?col)))
  :effect (and (is-agent ?a  ?next-row ?col)
        (not (is-agent ?a ?row ?col))
        (nondet (bomb-it ?next-row ?col)))
)



(:action move-agent-up
  :parameters (?a ?row ?col ?prev-row)
  :precondition (and (is-agent ?a ?row ?col)
        (agent ?a)
        (next-row ?prev-row ?row)
                    (not (bomb-it ?prev-row ?col)))
  :effect (and (is-agent ?a ?prev-row ?col)
        (not (is-agent ?a ?row ?col))
        (nondet (bomb-it ?prev-row ?col)))
)
```

```
  (:action move-agent-right
    :parameters (?a ?row ?col ?next-col)
    :precondition (and (is-agent ?a ?row ?col)
          (agent ?a)
          (next-col ?col ?next-col)
                      (not (bomb-it ?row ?next-col)))
    :effect (and (is-agent ?a ?row ?next-col)
          (not (is-agent ?a ?row ?col))
          (nondet (bomb-it ?row ?next-col)))
  )

  (:action move-agent-left
    :parameters (?a ?row ?col ?prev-col)
    :precondition (and (is-agent ?a ?row ?col)
          (agent ?a)
          (next-col ?prev-col ?col)
                      (not (bomb-it ?row ?prev-col)))
    :effect (and (is-agent ?a ?row ?prev-col)
          (not (is-agent ?a ?row ?col))
      (nondet (bomb-it ?row ?prev-col)))
  )
)
```

Fisierul bomb_it_state.pddl

```
(define (problem bombIt)
  (:domain BombIt)
  (:objects
     row1 row2 row3 row4 row5
     col1 col2 col3 col4 col5
     agent1
```

```
    agent2)
  (:init
    (next-row row1 row2)          (next-col col1 col2)
    (next-row row2 row3)          (next-col col2 col3)
    (next-row row3 row4)          (next-col col3 col4)
    (next-row row4 row5)          (next-col col4 col5)

    (agent agent1)
    (agent agent2)
    (is-agent agent1 row1 col1)            (is-agent agent2 row2 col1)
  )
  (:goal
    ( and
      (is-agent agent1 row2 col5)
      (is-agent agent2 row5 col5)
    )
  )
)
```

Fisierul cocktails_domain.pddl

```
(define (domain cocktails)
   (:predicates (is-water ?w)
                (is-rom ?r)
                (is-poisoned ?p)
   )

   (:action add-water
       :parameters  (?water)
       :precondition (and (is-water ?water)
                   (not (is-poisoned ?water)))
       :effect ( and (not (is-water ?water)))
```

```
    )

    (:action add-antidote-to-water
        :parameters  (?water)
        :precondition (is-water ?water)
        :effect (when (is-poisoned ?water) (not (is-poisoned ?water)))
    )

    (:action add-rom
        :parameters  (?rom)
        :precondition (is-rom ?rom)
        :effect (not (is-rom ?rom)))
)
```

Fisierul cocktails_state.pddl

```
(define (problem cocktails1)

(:domain cocktails)

(:objects w1 w2 w3 r1 r2 r3)

(:init
  (is-water w1)
  (is-water w2)
  (is-water w3)

  (is-rom r1)
  (is-rom r2)
  (is-rom r3)

  (is-poisoned w1)
```

```
  (unknown (is-poisoned w2))
  (unknown (is-poisoned w3))
  (or (is-poisoned w2) (is-poisoned w3))
)
(:goal
      (and
  (not (is-rom r1))
  (not (is-rom r2))
  (not (is-rom r3))
  (not (is-water w1))
  (not (is-water w2))
  (not (is-water w3)))
)
)
```

* * *

Intelligent Systems Group