Tema 2: **Planificare**

Tudor Berariu
Laboratorul AI-MAS
Facultatea de Automatică și Calculatoare
Universitatea "Politehnica" București

13 decembrie 2012

1 Scopul temei

Scopul acestei teme îl reprezintă familiarizarea cu conceptul de planificare și implementarea unor algoritmi de planificare (unul clasic: căutare neghidată înapoi în spațiul stărilor; și unul care să îmbunătățească performanțele primului printr-o strategie la alegere). Problema presupune planificare într-un mediu dinamic.

2 Descrierea problemei

Pe un etaj al unei clădiri se află un *labirint* de camere cu uși de acces între ele. Accesul nu este obligatoriu bidirecțional. Două dintre acestea au o destinație specială, fiind depozite: camera roșie și camera albastră. Restul camerelor sunt albe.

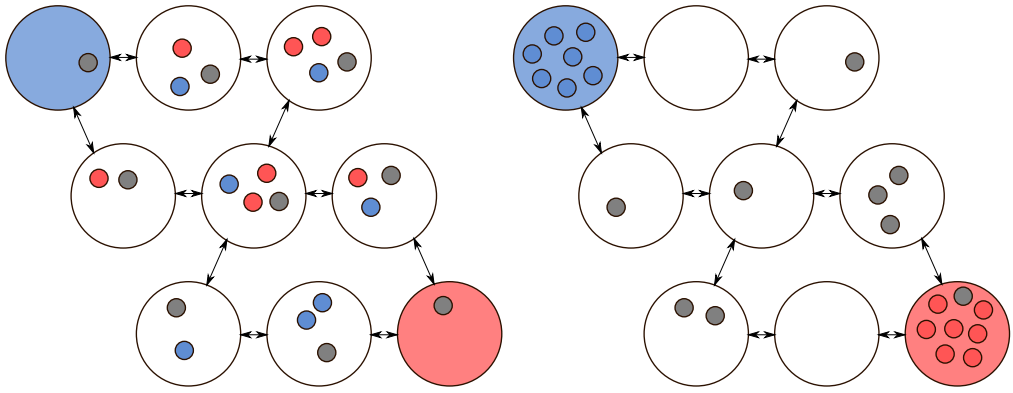
În camere se află un număr de sfere de culoare roșie, albastră sau gri.

Atât în camera roșie, cât și în camera albastră, se află câte un robot de aceeași culoare (roșu sau albastru) care are misiunea de a aduce în depozitul respectiv toate sferele de aceeași culoare (vezi Figura 1).

Fiecare dintre roboți se poate deplasa liber dintr-o cameră în oricare altă cameră vecină cu prima. Fiecare dintre roboți are 2 spații interne de depozitare în care poate încărca câte o sferă indiferent de culoare. Deoarece aceste sfere sunt destul de grele, iar compartimentele se află în părțile laterale

ale robotului, acesta nu se poate deplasa dintr-o cameră în alta cu un singur compariment încărcat. Așadar, robotul se poate muta, fie cu ambele spații interne de stocare goale, fie cu ambele încărcate cu câte o sferă. Un robot poate încărca și descărca orice sferă din și în orice cameră cu o singură excepție: nu se pot încărca sferele aflate deja în depozitul lor (sferele duse în depozitul lor nu mai pot fi mișcate).

Pe etaj sunt N camere (depozitul roșu, depozitul albastru și $N-2$ camere albe), M sfere roșii, M sfere albastre și N sfere gri (inițial câte o sferă gri în fiecare cameră, vezi Figura 1a). Un agent își încheie misiunea atunci când toate cele M sfere de aceeași culoare se află în depozitul corespunzător. Poziția finală a sferelor gri nu este importantă.



(a) Configurația de start (b) O posibilă stare finală

Figura 1: Exemplu de scenariu (camerele și ușile sunt reprezentate prin noduri și arce în graf)

Mutarea dintr-o cameră într-o altă cameră vecină, încărcarea unei sfere și descărcarea unei sfere se fac într-o unitate de timp. De asemenea, procesul care constă în actualizarea informațiilor despre pozițiile tuturor sferelor din toate camerele și conceperea unui plan (o secvență de acțiuni) necesită o unitate de timp. Cum robotul acționează într-un mediu dinamic, se poate întâmpla ca până la momentul aplicării unei operator, condițiile acestuia să nu mai fie adevărate. Dacă robotul încearcă să execute o acțiune ce nu se poate aplica (de exemplu: încărcarea unei sfere care nu se găsește în camera curentă), atunci acesta se blochează și mai are nevoie de o unitate de timp pentru a-și reveni. Aplicarea unui operator eronat (de exemplu: încărcarea unei a treia sfere, mutarea dintr-o cameră în alta care nu este vecină cu

prima, etc.) se penalizează de asemenea prin blocarea robotului pentru o unitate de timp. După ce se deblochează, reintră în rutina de planificare.

Se caută ingineri care să programeze un planificator pentru acești doi roboți astfel încât aceștia să rezolve cât mai repede misiunea pe care au.

2.1 Reprezentarea cunoștințelor

Pentru a reprezenta cunoștințele despre mediu se vor folosi predicatele:

- `Location(room)` - cu semnificația că robotul se află în camera `room`;
- `Spheres(color, room, n)` - cu semnificația că în camera `room` se găsesc `n` sfere de culoarea `color`;
- `Color(room, color)` - cu semnificația că încăperea `room` are culoarea `color`;
- `Color(color)` - cu semnificația că robotul are culoarea `color`;
- `Door(room1, room2)` - cu semnificația că se poate trece din camera `room1` în camera `room2`;
- `Carries(color, n)` - cu semnificația că robotul are încărcate în compartimentele interne `n` sfere de culoarea `color`.
- `Succ(n1, n2)` - predicat ce va fi adevărat pentru orice numere naturale `n1` și `n2` consecutive (`n2 = n1 + 1`);
- `Greater(n1, n2)` - predicat ce va fi adevărat pentru orice numere naturale `n1` și `n2` pentru care (`n1` $>$ `n2`);
- `Positive(n)` - predicate ce va fi adevărat pentru orice număr natural strict pozitiv `n`.

Intern, robotul poate folosi oricâte alte predicate pentru a reprezenta complet starea sa sau pe cea a mediului, însă doar cele enumerate mai sus vor fi folosite pentru a transmite starea curentă planificatorului atât la începutul programului, cât și pe parcurs când este necesară replanificarea.

2.2 Operatori

Planurile conțin următoarii operatori:

- `Move(room1, room2)` care reprezintă acțiunea prin care agentul se mută din camera `room1` în camera `room2`. Acțiunea `Move` reușește doar dacă există o ușă între `room1` și `room2` și, fie ambele compartimente ale robotului sunt pline, fie ambele compartimente sunt goale (robotul cară zero sau două sfere).

- `Load(color)` care reprezintă acțiunea prin care agentul culege o sferă de culoarea `color` din camera în care se află și-o încarcă într-un spațiu intern de stocare liber. Acțiunea reușește întotdeauna dacă în camera în care se află agentul există cel puțin o sferă de culoare `color`, camera nu are aceeași culoare cu sfera și robotul nu cară deja două sfere.

- `Unload(color)` care reprezintă acțiunea prin care agentul descarcă o sferă de culoare `color` în camera în care se află. Acțiunea reușește numai dacă robotul are în compartimentele interne cel puțin o sferă de culoare `color`.

- `Test(condition)` a cărui aplicare constă în verificarea conjuncției de predicate din `condition`. În cazul în care condițiile sunt adevărate, planul este continuat prin aplicarea următoarei acțiuni, altfel, se abandonează planul curent pentru replanificare.
  `condition` poate conține doar predicatele `Spheres`, `Succ`, `Greater` și `Positive` (restul informațiilor despre lume nu se schimbă pe parcurs). De exemplu, pentru a verifica faptul că în `Room1` se găsesc cel puțin două bile roșii:

  $$\texttt{Spheres(Red, Room1, n)} \wedge \texttt{Succ(n1, n)} \wedge \texttt{Positive(n)}$$

  sau, pentru a verifica simultan că în camera `Room1` se află cu cel puțin două sfere roșii mai multe decât în camera `Room2`, iar în camera `Room3` se află o singură sferă gri (de data aceasta în sintaxă Scheme):

Listing 1: Exemplu condiție pentru operatorul Test

```
((Spheres Red Room1 n1) (Spheres Red Room2 n2) (Succ n n1)↩
    (Greater n n2) (Spheres Grey Room3 1))
```

Fiecare dintre acești operatori se execută într-o unitate de timp.

2.3 Planificatorul

Planificatorul robotului nu are memorie internă. El primește o listă cu 4 elemente:

- obiectivul său, un predicat de forma `spheres(color,warehouse,m)` (de exemplu, pentru robotul roșu: `spheres(Red, RedWarehouse, M)`);
- starea lumii: culoarea lui, camera în care se află el, numărul de sfere de culoare gri, roșie și albastră din fiecare cameră, perechile de camere vecine și culorile camerelor;
- restul de acțiuni ce nu au fost executate, dacă a eșuat aplicarea unui plan sau verificarea condiției unui operator `Test`;
- o listă cu informații suplimentare, pe care robotul a reîntors-o odată cu ultimul plan (permite salvarea unui context de calcul și simulează memoria internă).

Rezultatul planificării conține 2 elemente:

- planul efectiv: o secvență de operatori dintre `Move`, `Load`, `Unload`, `Test`;
- o listă conținând orice informații; aceasta îi va fi retrimisă planificatorului dacă planul eșuează sau verificarea condiției unui operator `Test` eșuează.

3 Cerințe

3.1 [0.2p] Cerința 1: Descriere STRIPS

Folosind predicatele enumerate în secțiunea anterioară, dar și alte predicate suplimentare pe care le considerați necesare, descrieți următorii operatori folosind STRIPS: `Move`, `Load` și `Unload`.

Operatorul `Test` are un statut special și nu trebuie descris.

3.2 [0.6p] Cerința 2: Planificare simplă

Să se implementeze un agent `memoryless-agent` care aduce toate sferele de aceeași culoare cu el în depozitul corespunzător după cum urmează. Atât

timp cât mai există bile care nu se află în depozit, agentul execută ciclul următor:

1. Agentul își face un plan pentru a aduce o sferă de aceeași culoare cu el în depozit.

2. Agentul execută pe rând acțiunile planului.

   (a) Dacă o acțiune nu se poate aplica (condițiile nu sunt îndeplinite), atunci agentul rămâne blocat pentru o unitate de timp și apoi se întoarce la pasul 1.

   (b) Dacă acțiunile s-au terminat, ciclul curent se încheie.

Agentul `memoryless-agent` nu poate include operatorul `Test` în planul lui și nici nu-și poate salva informații pe care să le utilizeze în momentul replanificării.

Planificarea trebuie făcută prin căutare înapoi în spațiul stărilor (backward state-space search / regression planning).

Planificatorul agentului `memoryless-agent` va fi apelat de fiecare dată astfel:

```
(memoryless-agent goal world-state () ()).
```

Unde `goal` va fi `(Spheres color warehouse (+ 1 m))` unde `color` este culoarea robotului, `warehouse` reprezintă depozitul, iar `m` este numărul de bile duse deja acolo.

Agentul nu va face diferența între o planificare de la zero și o replanificare după eșecul aplicării unui operator.

Testarea se va face apelând planificatorul `memoryless-agent` atât pentru agentul albastru, cât și pentru agentul roșu.

3.3 [0.4p] Cerința 3: Tehnici de planificare avansată

Să se implementeze un agent `advanced-agent` care este mai eficient decât `memoryless-agent`. Se poate folosi orice strategie pentru căutarea / construirea planului. Mai mult, agentul poate include în acțiunile din plan operatorul `Test(condition)`, unde `condition` este o listă predicate dintre `Spheres`, `Succ`, `Greater` și `Positive`. Dacă acestea nu sunt adevărate în starea curentă, se apelează funcția:

`(advanced-agent goal world-state rest-of-actions saved-info).`

Sarcina acestei cerințe este de a construi un agent cât mai eficient, îmbunătățindu-se `memoryless-agent`. Se recomandă exploatarea următoarelor direcții:

- optimizarea căutării înapoi prin adăugarea unei euristici care să ghideze căutarea;
- repararea unui plan eșuat fără a reface replanificarea de la zero (vezi Figura 2);
- gestionarea simultană a mai multor planuri.

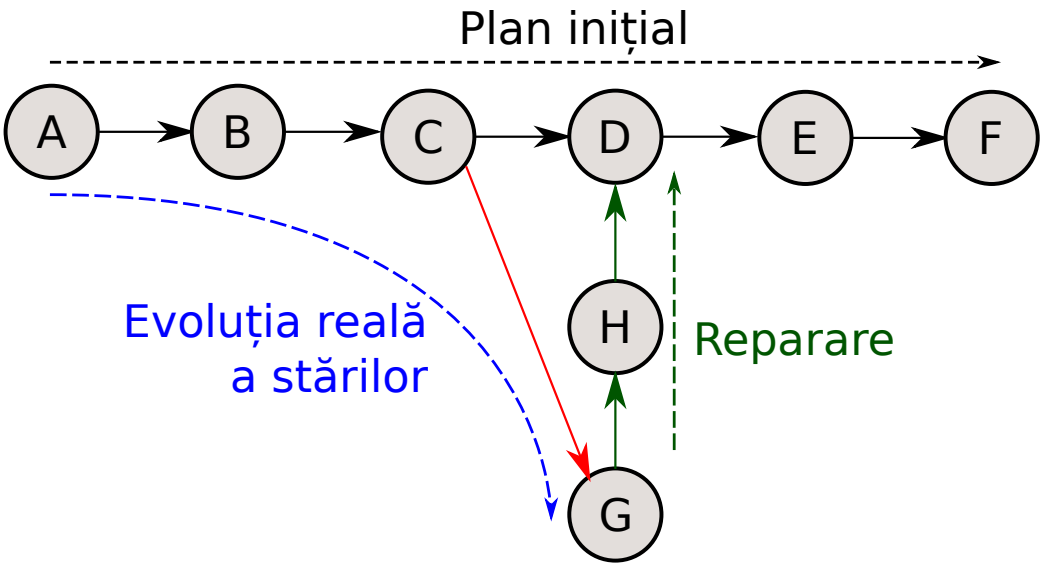


Figura 2: Replanificare: repararea unui plan (adaptată din [RN03])

În Figura 2 este reprezentat un plan inițial care într-un mediu static ar fi dus la secvența de stări: $A \to B \to C \to D \to E \to F$. Cum însă mediul este unul dinamic, din starea C s-a ajuns în starea G (tranziția marcată cu roșu). Repararea unui plan presupune găsirea unei secvențe de acțiuni care să ducă mediul înapoi în starea D de unde să se reia restul de acțiuni din planul inițial. Practic replanificarea va produce un plan $G \to H \to D \to E \to F$ realizat prin concatenarea planului de reparare cu restul planului precedent.

Testarea `advanced-agent` se va face prin plasarea lui în același scenariu cu `memoryless-agent` (unul va fi planificatorul robotului roșu, iar celălalt planificatorul robotului albastru).

4 Trimiterea temei

Cerința 1 se trimite într-un fișier pdf:

`Nume_Prenume_Grupa_T2_C1.pdf`

Cerințele 2 și 3 se rezolvă într-un fișier Scheme (`agents.scm`) unde trebuie implementate funcțiile `memoryless-agent` și `advanced-agent`. Fiecare dintre acestea primește 4 parametri:

1. `goals` - listă de predicate reprezentând obiectivele robotului;
2. `world-state` - listă de predicate care descrie starea lumii (culoarea robotului, camera în care se află, culorile tuturor camerelor, ușile dintre camere, numărul de sfere din fiecare culoare din fiecare cameră);
3. `left-actions` - listă de acțiuni ce au rămas neexecutate din planul precedent;
4. `info` - valoarea pe care a reîntors-o aceeași funcție odată cu planul precedent;

și reîntorc o listă `(plan info)` unde `plan` este o listă de acțiuni, iar `info` poate lua orice valoare.

Acțiunile vor fi liste în care pe prima poziție se regăsește numele operatorului, urmat de argumentele acestuia (obiecte din lumea problemei).

5 Testare

În arhiva atașată acestui document se găsesc 6 scenarii numerotate de la zero la cinci (în fișierele `scenarioX.scm`) ce pot fi folosite pentru testarea agenților.

Fișierul `play.scm` conține funcții pentru rularea scenariilor și testarea agenților. Pentru rularea unui scenariu se folosește funcția `run` care primește 2 argumente:

- `scenario-file` - fișierul din care să se încarce scenariul
- `level` - 1 pentru `memoryless-agent` vs. `memoryless-agent` și 2 pentru `advanced-agent` vs. `memoryless-agent`

.

5.1 Scenariul 0

Starea inițială a scenariului 0 (`scenario0.scm`) este reprezentată în Figura 3.

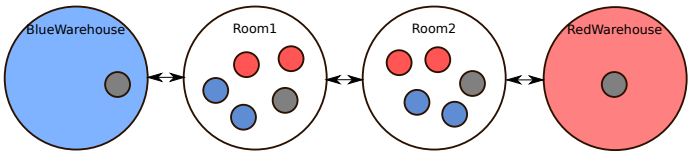


Figura 3: Starea inițială pentru scenariul 0

Pentru scenariul 0, planificatorul va fi apelat inițial cu:

Listing 2: Apelul inițial către planificatorul agentului roșu

```
(memoryless-agent
 '(Spheres Red RedWarehouse 1)
 '((Color BlueWarehouse Blue) (Color RedWarehouse Red)
   (Color Room1 White) (Color Room2 White)
   (Door BlueWarehouse Room1) (Door Room1 BlueWarehouse)
   (Door Room1 Room2) (Door Room2 Room1) (Door Room2 ↩
      RedWarehouse)
   (Door RedWarehouse Room2) (Spheres Grey BlueWarehouse 1)
   (Spheres Red BlueWarehouse 0) (Spheres Blue BlueWarehouse 0)
   (Spheres Grey Room1 1) (Spheres Red Room1 2)
   (Spheres Blue Room1 2) (Spheres Grey Room2 1)
   (Spheres Red Room2 2) (Spheres Blue Room2 2)
   (Spheres Grey RedWarehouse 1) (Spheres Red RedWarehouse 0)
   (Spheres Blue RedWarehouse 0) (Carries Red 0)
   (Carries Blue 0) (Carries Grey 0)
   (Location RedWarehouse) (Color Red)
   )
 '()
 '()
 )
```

5.2 Scenariul 1

Starea inițială a scenariului 1 (`scenario1.scm`) este reprezentată în Figura 4.

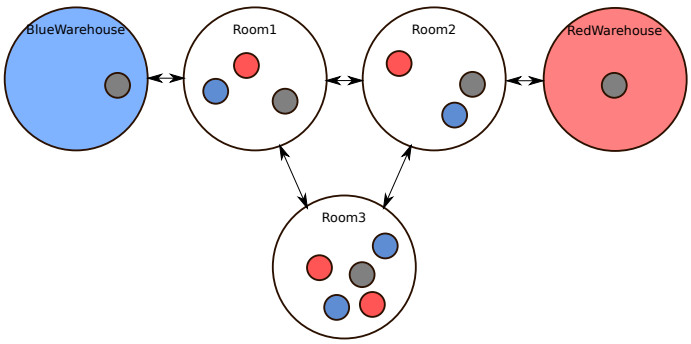


Figura 4: Starea inițială pentru scenariul 1

5.3 Scenariul 2

Starea inițială a scenariului 2 (`scenario2.scm`) este reprezentată în Figura 5.

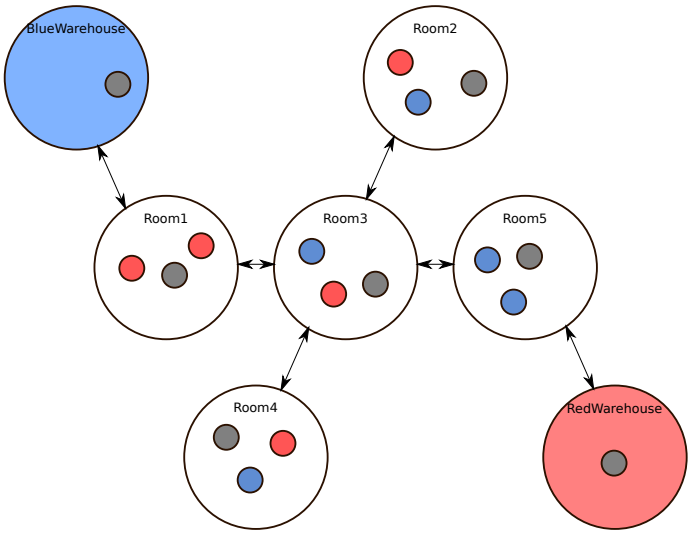


Figura 5: Starea inițială pentru scenariul 2

Inițial obiectivul agentului `memoryless-agent` pentru robotul albastru este:

```
(Spheres Blue BlueWarehouse 1)
```

După ce obiectivul este atins, planificatorul va fi apelat cu:

```
(Spheres Blue BlueWarehouse 2)
```

5.4 Scenariul 3

Starea inițială a scenariului 3 (`scenario3.scm`) este reprezentat în Figura 6.

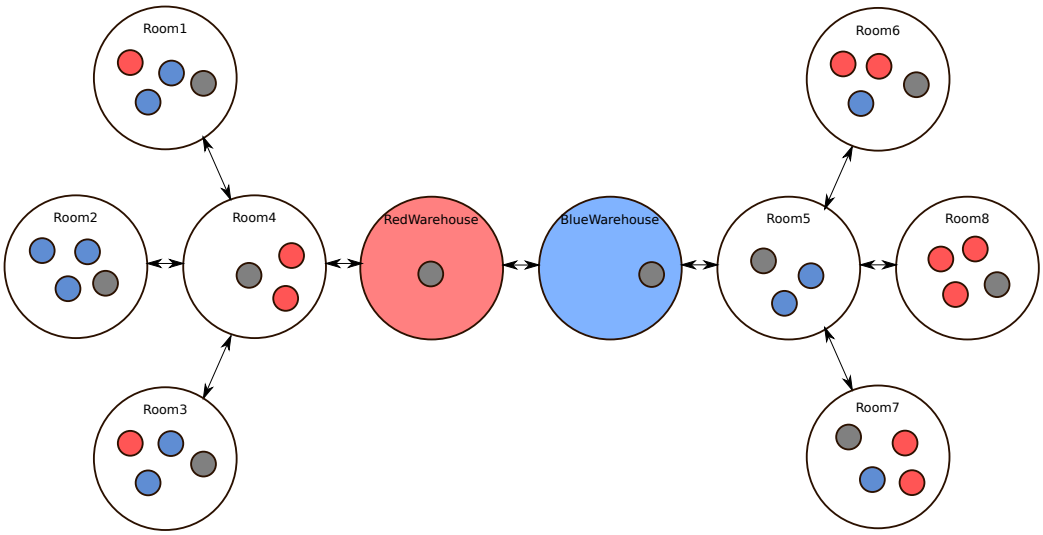


Figura 6: Starea inițială pentru scenariul 3

5.5 Scenariul 4

Starea inițială a scenariului 4 (`scenario4.scm`) este reprezentat în Figura 7.

5.6 Scenariul 5

Starea inițială a scenariului 5 (`scenario5.scm`) este reprezentat în Figura 8.

Bibliografie

[RN03] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Pearson Education, 2 edition, 2003.

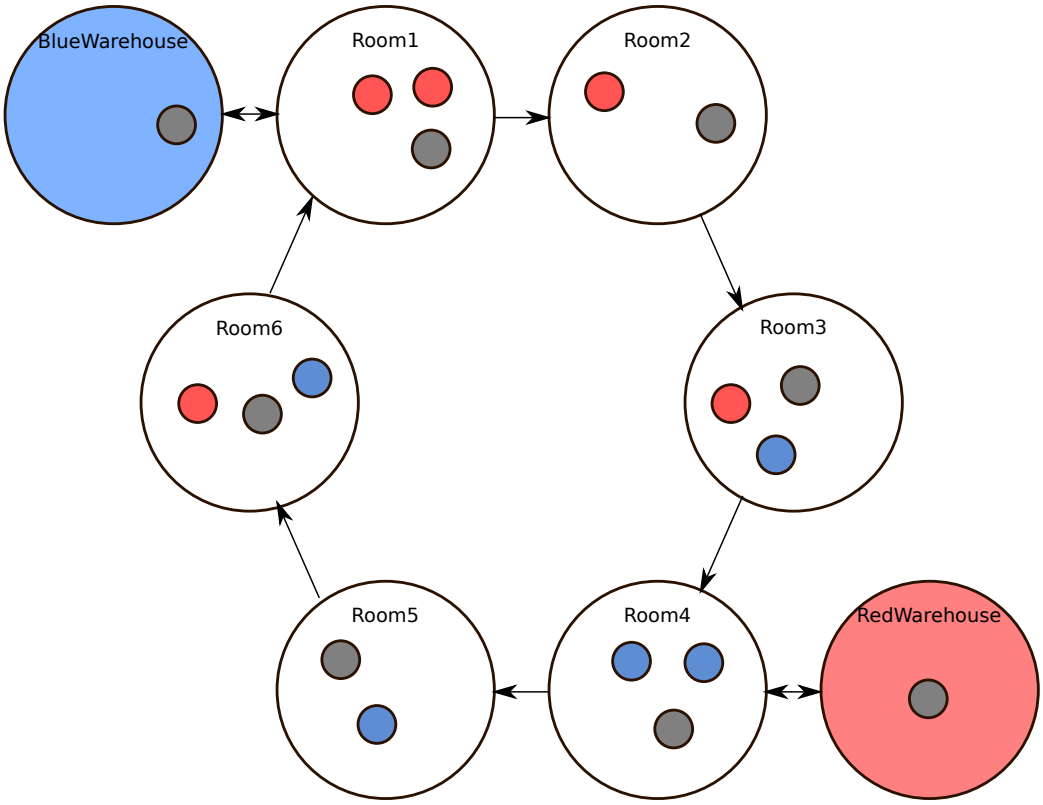


Figura 7: Starea inițială pentru scenariul 4

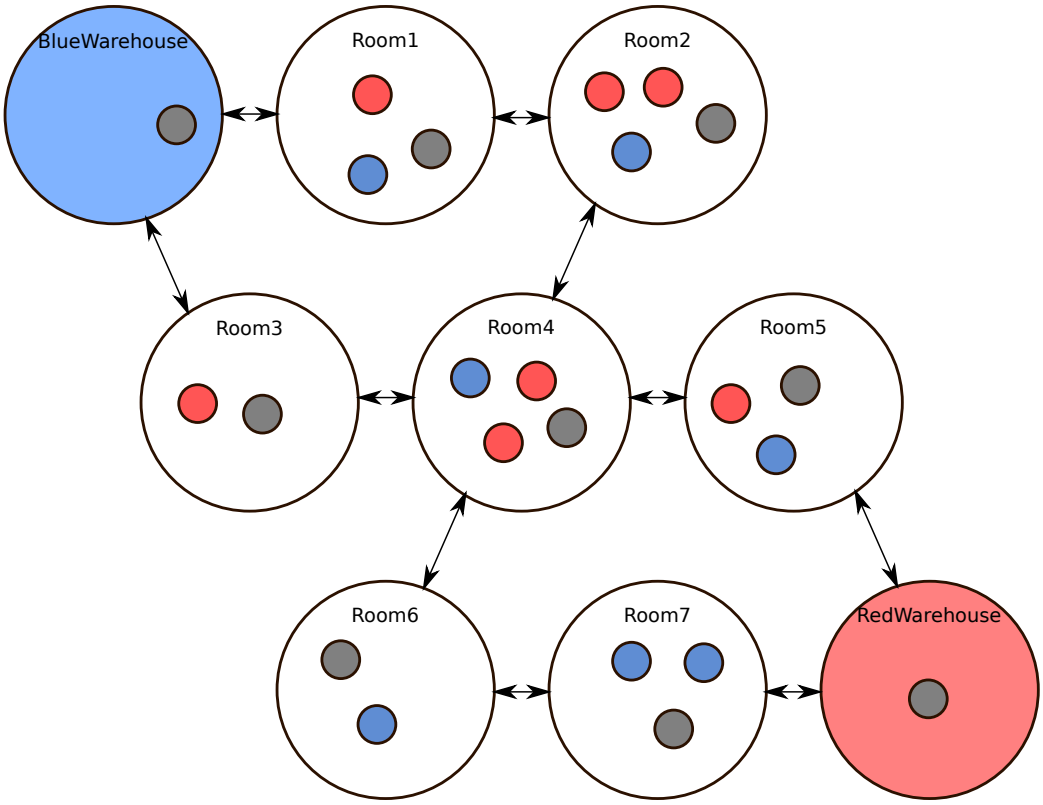


Figura 8: Starea inițială pentru scenariul 5