Inteligență Artificială
Tema 1: **Algoritmul A***

Tudor Berariu
tudor.berariu@gmail.com
Laboratorul AI-MAS
Facultatea de Automatică și Calculatoare

2 noiembrie 2014

1 Scopul temei

Scopul acestei teme îl reprezintă înțelegerea și implementarea algoritmului **A***. Testarea se va face pe un joc simplu pentru care se cere și găsirea unei euristici cât mai bune pentru aplicarea algoritmului **A***.

2 Descrierea jocului

Jocul ales pentru demonstrarea utilității algoritmului **A*** este *Unblock* [1]. Acesta se desfășoară pe o tablă de dimensiune $height \times width$ pe care se găsesc N blocuri de dimensiune $length_i, 1 \leq i \leq N$. Acestea sunt dispuse fie orizontal, fie vertical, așa cum se poate observa în Figura 1. Fiecare bloc poate glisa pe direcția pe care este poziționat dacă nu se lovește de alte blocuri și nu depășește marginile tablei (vezi Figura /reffig:2).

Scopul jocului este acela ca prin mutări succesive să se aducă blocul roșu în extremitatea dreaptă, precum în Figura 3.

În cadrul acestei teme se vor genera planuri (secvențe de mișcări) care să ghideze blocul roșu către marginea din dreapta a tablei.

[1] `http://www.quickflashgames.com/games/unblock/`

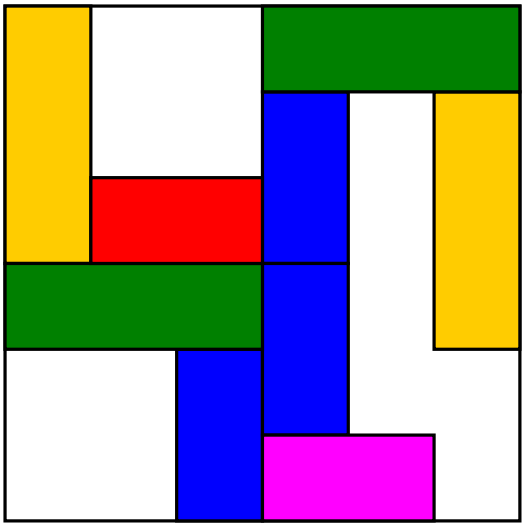

Figura 1: Stare posibilă în jocul *Unblock*. Obiectivul este acela de a duce blocul roșu în extremitatea dreaptă.

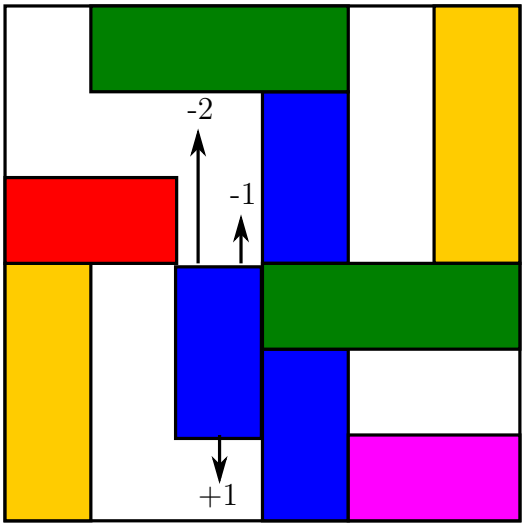

Figura 2: Blocul albastru se poate muta în trei poziții: două linii mai sus, o linie mai sus sau o linie mai jos.

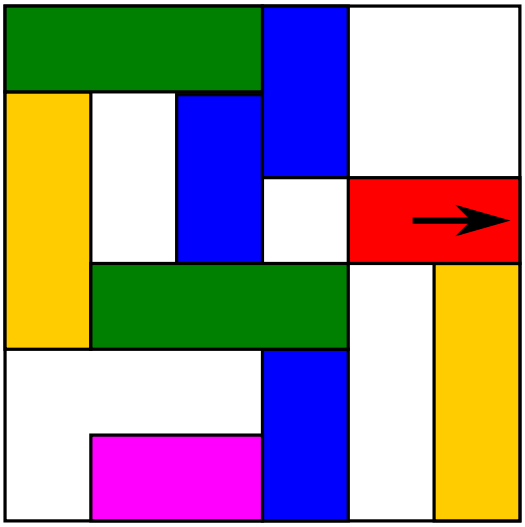

Figura 3: Stare finală posibilă a jocului (blocul roșu a ajuns la limita din dreapta)

3 Cerințe

În fișierul `unblock.rkt` sunt implementate funcția ce întoarce lista tuturor stărilor următoare posibile pentru una data (`get_reachable_states(s)`) și o funcție ce verifică dacă o stare este finală (`is_final?(s)`). De asemenea, sunt implementate căutările în adâncime și în lățime pe care le puteți folosi ca punct de plecare pentru rezolvarea primei sarcini.

Cerințele aceste teme sunt următoarele:

a) `[0.4 puncte]` Să se implementeze algoritmul **A*** folosind orice euristică admisibilă. Să se compare rezultatele algoritmului cu cele ale căutării în adâncime și lățime folosind funcția `compare_algorithms`.

b) `[0.1 puncte]` Să se găsească o euristică cât mai informată, care să reducă semnificativ stările explorate față de căutarea în adâncime sau în lățime.

c) `[0.1 puncte]` Să se demonstreze admisiblitatea euristicilor folosite.

Euristicile folosite, precum și demonstrația de la bonus, trebuie descrise într-un document care să fie inclus în arhiva temei.

4 Detalii despre fișierul `unblock.rkt`

4.1 Algoritmii de căutare

Abstractizarea problemei propuse

Pentru rezolvarea primei cerințe (implementarea **A***) folosiți funcțiile:

- `get-reachable-states(current-state)` - care întoarce o listă de perechi `(next-state . action)` ce conțin stări următoare cu acțiunile asociate cu care s-a produs tranziția.
- `is-final?(state)` - predicat ce întoarce `#t` dacă argumentul este o stare finală.

și implementați funcția `a_star(state)`. Vă puteți inspira din funcțiile `dfs(state)` și `bfs(state)`, deja implementate.

Evaluarea soluțiilor

Pentru a compara algoritmii, rulați funcția `compare-algorithms` care primește trei argumente

- lista algoritmilor de comparat (funcții ce primesc o tablă : starea inițială) și întorc un plan;
- lista numelor acestor algoritmi;
- lista tablelor ce trebuie rezolvate (sunt definite în fișier câteva scenarii de joc: `board1`, `board2`, ...).

De exemplu, expresia de mai jos

```
(compare-algorithms
    `(,dfs ,bfs ,a-star)
    `("DFS" "BFS" "A*")
    `(,board1 ,board2 ,board3 ,board4 ,board5
      ,board6 ,board7 ,board8 ,board9 ,board10))
```

produce următorul rezultat care sumarizează numărul de stări explorate de fiecare algoritm (desigur, algoritmul **A*** nu este încă implementat, dar cu siguranță veți schimba asta):

board	DFS	BFS	A*
1	861	867	GREȘIT
2	374	1069	GREȘIT
3	55	85	GREȘIT
4	1116	2231	GREȘIT
5	812	848	GREȘIT
6	1357	517	GREȘIT
7	468	601	GREȘIT
8	781	791	GREȘIT
9	7373	7935	GREȘIT
10	5060	7080	GREȘIT

Testele

În fișier există câteva scenarii reprezentate:

- `board0a`, `board0b`, `board0c` - table de joc simple pentru testare rapidă.
- `board1` - `board5` - primele cinci niveluri de aici: `http://www.quickflashgames.com/games/unblock/`
- `board6` - `board10` - primele cinci niveluri din setul *expert* de aici: `http://www.quickflashgames.com/games/unblock-2/`
- `board11` - scenariu pentru care DFS și BFS nu reușesc să rezolve în timp decent

Vizualizare

Pentru a vizualiza o singură stare puteți folosi funcția `display-board(board)`. De exemplu, `display-board(board8)` produce imaginea din Figura 4.

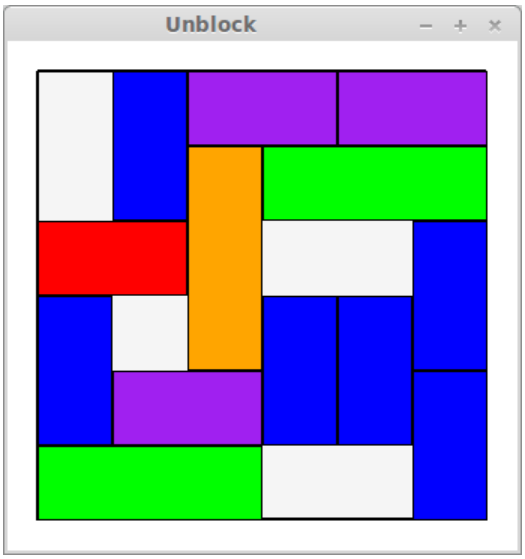

Figura 4: Scenariul 6

Pentru a vizualiza un plan aveți la dispoziție funcția `display-plan(state0, plan)`, unde `plan` este o secvență de mișcări întoarsă de unul dintre algoritmii implementați.

Euristici

Pentru construirea unei euristici cât mai bune, citiți în continuare detaliile despre structurile de date ce descriu tabla și blocurile.

4.2 Structurile de date specifice jocului

În fișierul `unblock.rkt` se găsesc definițiile a trei structuri:

block - structură cu trei câmpuri ce reprezintă definiția unui bloc:

name numele asociat blocului;

orientation variabilă ce indică dacă un bloc este orientat orizontal sau vertical (are valoarea `HORIZONTAL` sau `VERTICAL`;

length lungimea blocului.

```
(struct block (name orientation length))
```

block-on-board - structură ce extinde un bloc cu poziția pe tablă:

row linia pe tablă

column coloana pe tablă

```
(struct block-on-board block (row column))
```

board - structură ce reprezintă starea tablei la un moment dat:

height - înălțimea (numărul de linii ale) tablei;

width - lățimea (numărul de coloane ale) tablei;

blocks - tabelă de dispersie având chei numele blocurilor și valori structuri de tip `block` (descrise mai sus).

```
(struct board (height width blocks))
```

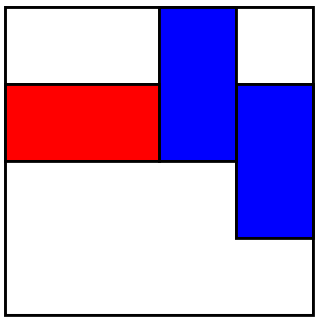

Figura 5: Scenariul `board0b`.

Exemplu de configurație inițială de joc (corespunzătoare Figurii 5):

```
(define board0b
  (board 4 4
         (hash "red" (block-on-board "red" HORIZONTAL 2 1 0)
               "blue1" (block-on-board "blue1" VERTICAL 2 0 2)
               "blue2" (block-on-board "blue2" VERTICAL 2 1 3)
              )))
```

Un plan generat prin căutare în adâncime este:

```
> (dfs board0b)
'(("blue2" -1 0) ("blue2" 2 0) ("blue1" 1 0) ("blue2" -1 0)
  ("blue2" -1 0) ("blue1" 1 0) ("red" 0 1) ("blue2" 1 0)
  ("red" 0 -1) ("blue2" 1 0) ("red" 0 1) ("red" 0 1))
```

Funcțiile `print-plan(plan)` și `display-plan(board, plan)` ajută la descifrarea planului:

```
> (print-plan (dfs board0b))
  1. Move block 'blue2' -1 rows, 0 columns.
  2. Move block 'blue2' 2 rows, 0 columns.
  3. Move block 'blue1' 1 rows, 0 columns.
  4. Move block 'blue2' -1 rows, 0 columns.
  5. Move block 'blue2' -1 rows, 0 columns.
  6. Move block 'blue1' 1 rows, 0 columns.
  7. Move block 'red' 0 rows, 1 columns.
  8. Move block 'blue2' 1 rows, 0 columns.
  9. Move block 'red' 0 rows, -1 columns.
  10. Move block 'blue2' 1 rows, 0 columns.
  11. Move block 'red' 0 rows, 1 columns.
  12. Move block 'red' 0 rows, 1 columns.
```

Funcția care întoarce toate stările în care se poate ajunge dintr-o stare curentă este `get-reachable-states(s)`. De exemplu, pentru starea inițială `board0b`:

```
> (get-reachable-states board0b)
(((board 4 4
         (hash "red" (block-on-board "red" 2049 2 1 0)
               "blue2" (block-on-board "blue2" 2048 2 0 3)
               "blue1" (block-on-board "blue1" 2048 2 0 2)))
```

```
  . ("blue2" -1 0))
 ((board 4 4
         (hash "red" (block-on-board "red" 2049 2 1 0)
    "blue2" (block-on-board "blue2" 2048 2 2 3)
    "blue1" (block-on-board "blue1" 2048 2 0 2)))
  . ("blue2" 1 0))
 ((board 4 4
         (hash "red" (block-on-board "red" 2049 2 1 0)
               "blue2" (block-on-board "blue2" 2048 2 1 3)
               "blue1" (block-on-board "blue1" 2048 2 1 2)))
  . ("blue1" 1 0))
 ((board 4 4
         (hash "red" (block-on-board "red" 2049 2 1 0)
               "blue2" (block-on-board "blue2" 2048 2 1 3)
               "blue1" (block-on-board "blue1" 2048 2 2 2)))
  . ("blue1" 2 0)))
```

Rezultatul conține patru perechi (*stare.acțiune*) unde *stare* reprezintă o nouă configurație a blocurilor, iar *acțiune* este o listă compusă din numele blocului mutat, numărul de linii și numărul de coloane pe care s-a deplasat.

Succes!