Sistem de parking

Golban Ovidiu

January 6, 20221

# 1.Introducere

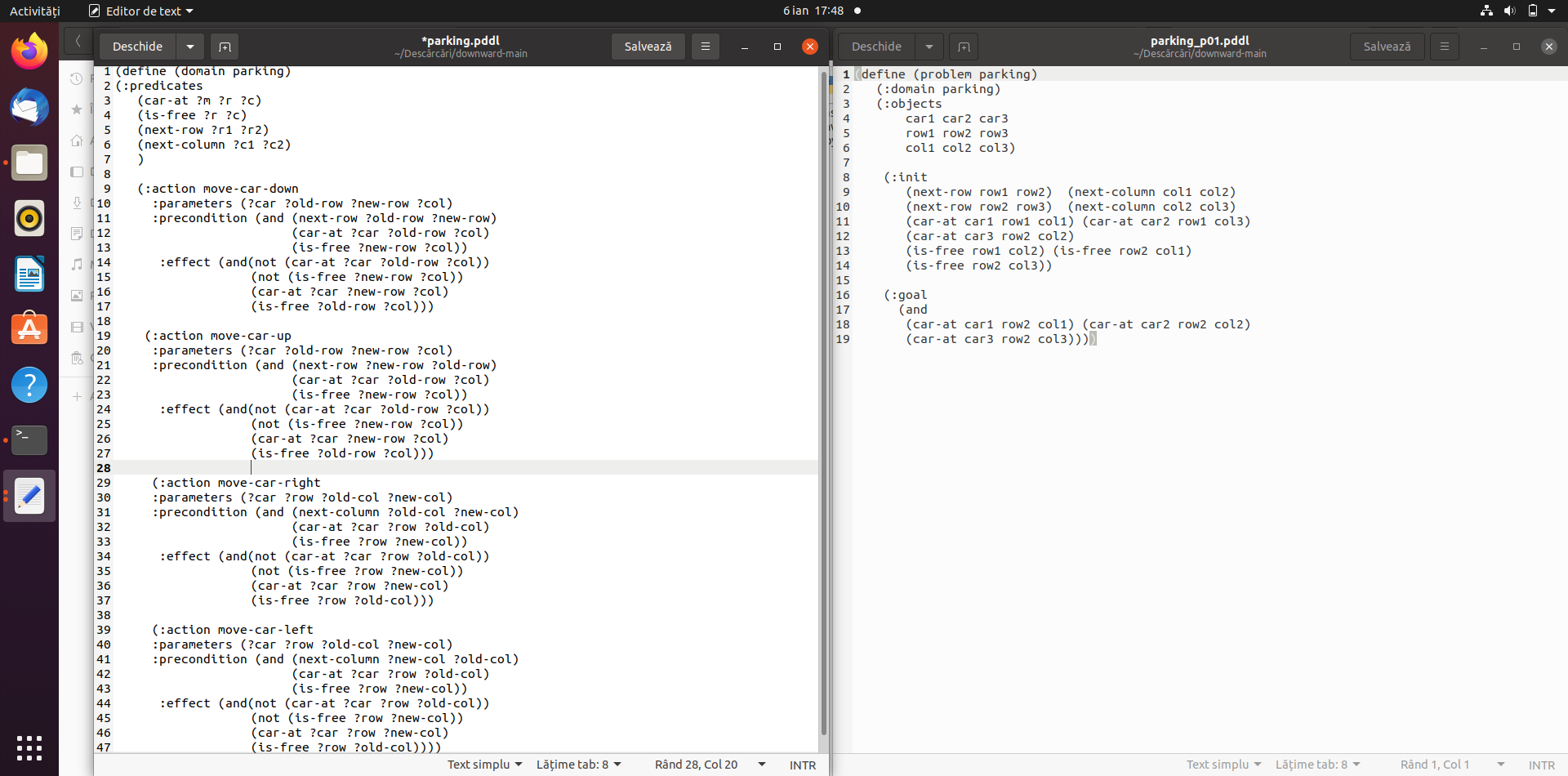
Sistemul implementat pentru acest proiect vine iınspre a face un pas mic pentru un oras,tot mai smarts sii astfel, are scop automatizarea unui parking prin plasarea masinilor pe anumite locuri cerute deadministratorul parkingului.2

# 2.Pasii pentru realizarea obiectivelor proiectului



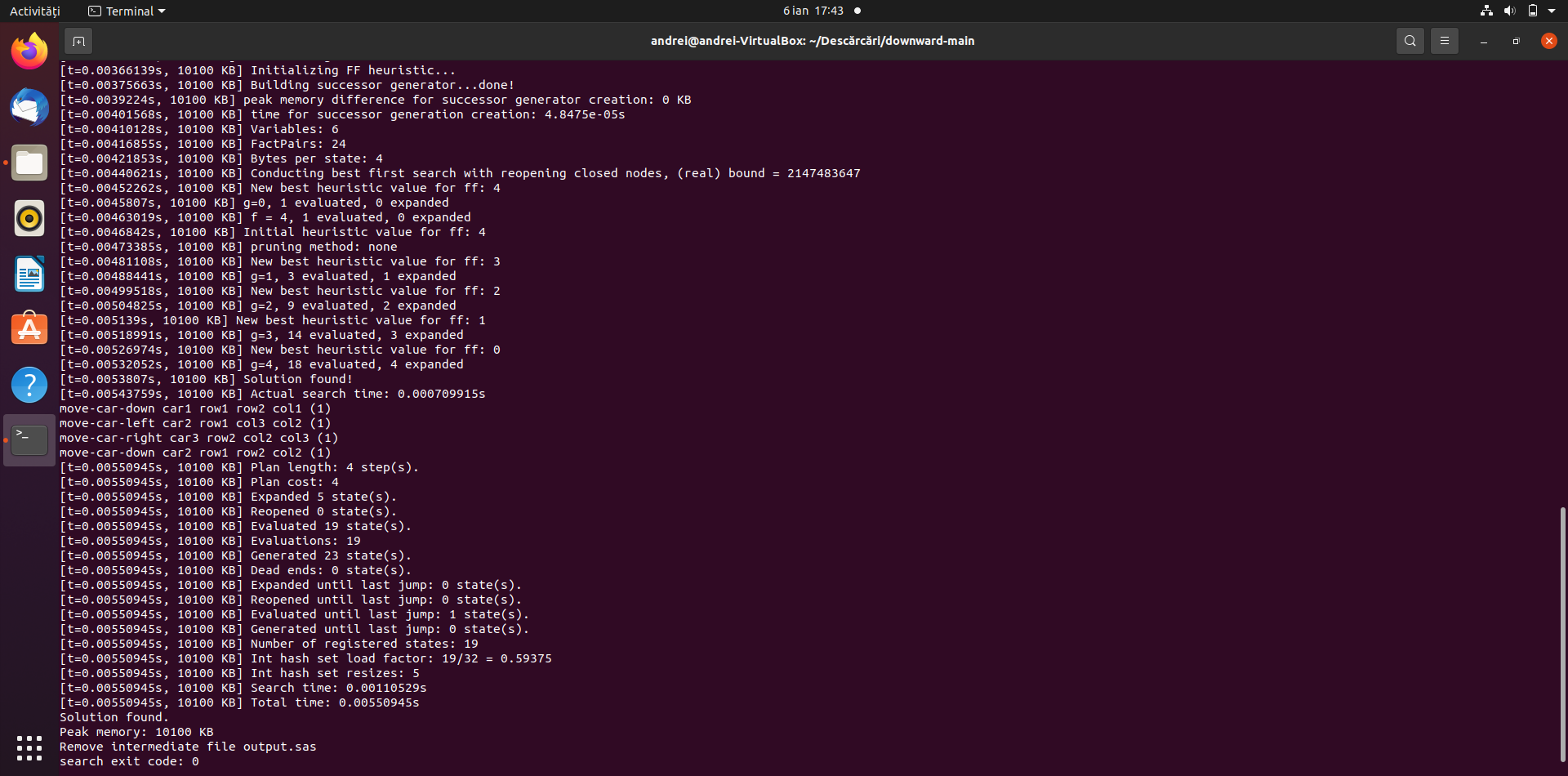
Fig. 0 – Cum arată parkingul?

# Cum am ajuns la aceasta varianta de proiect - Etapa intermediara

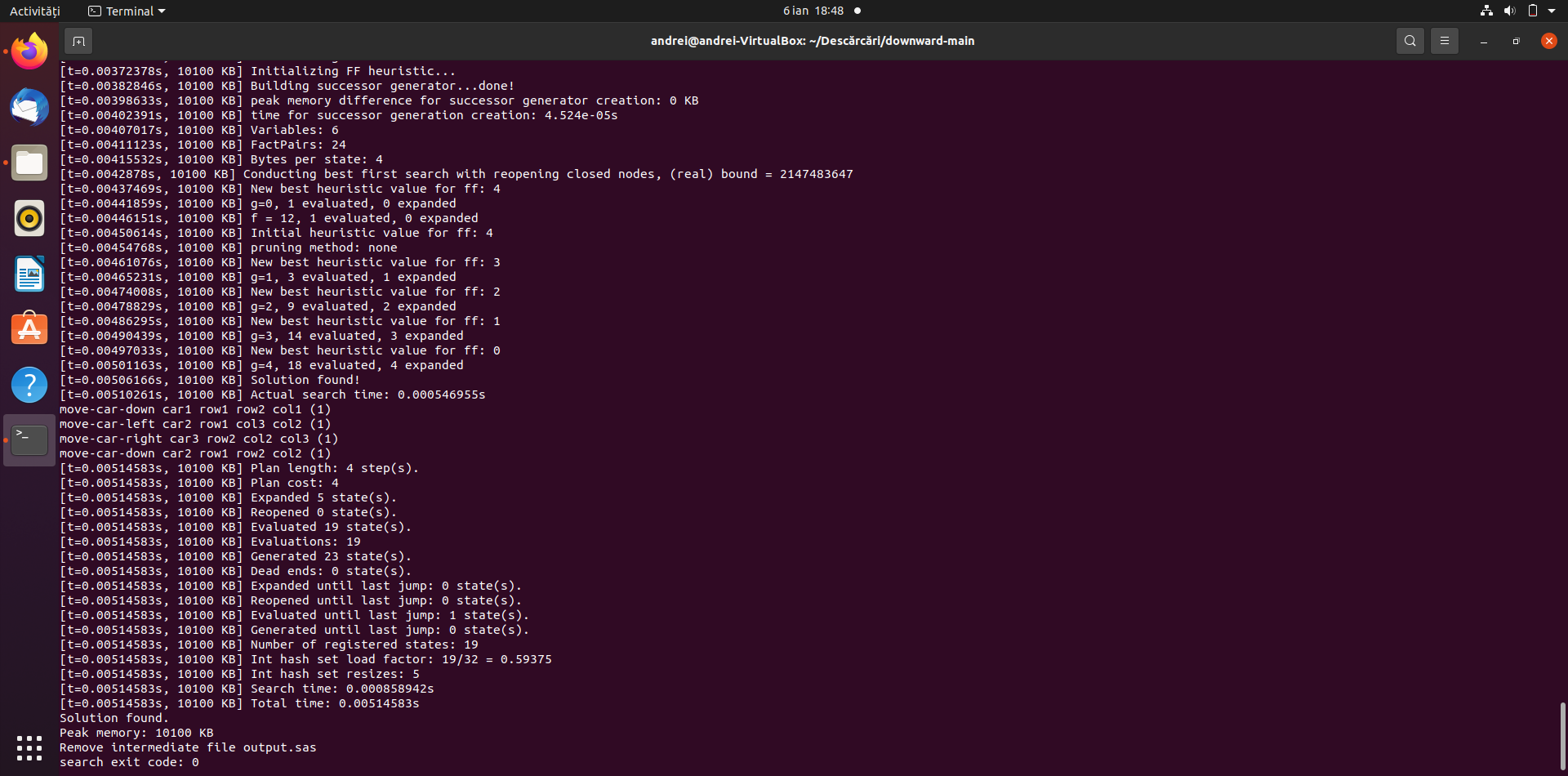


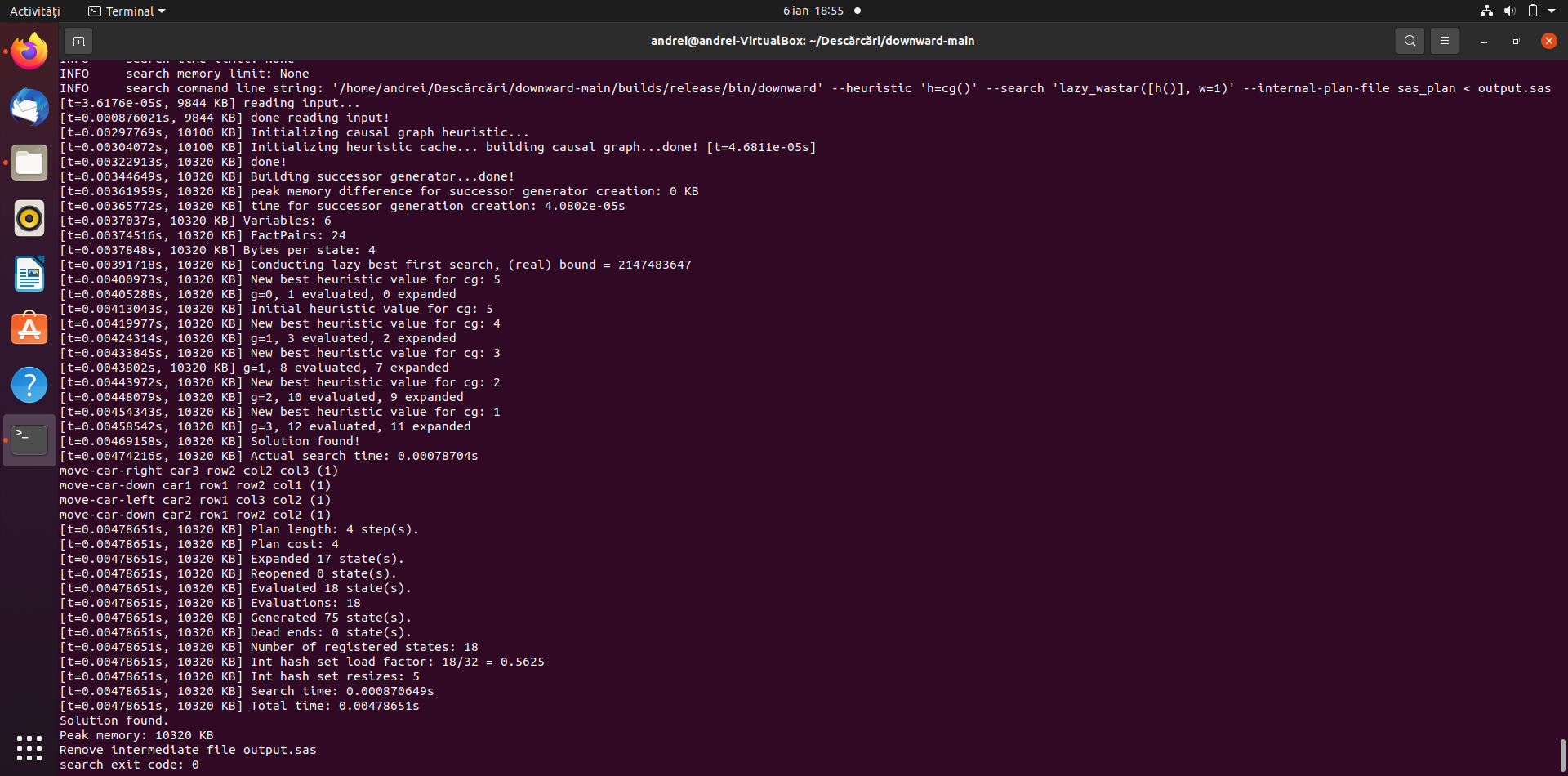
În poza de mai sus se poate observa codul de la primul caz descris, cel cu 2 linii, 3 coloane și 3 mașini. În cele ce urmează am rulat programul cu mai multe euristici pentru a observa diferența.

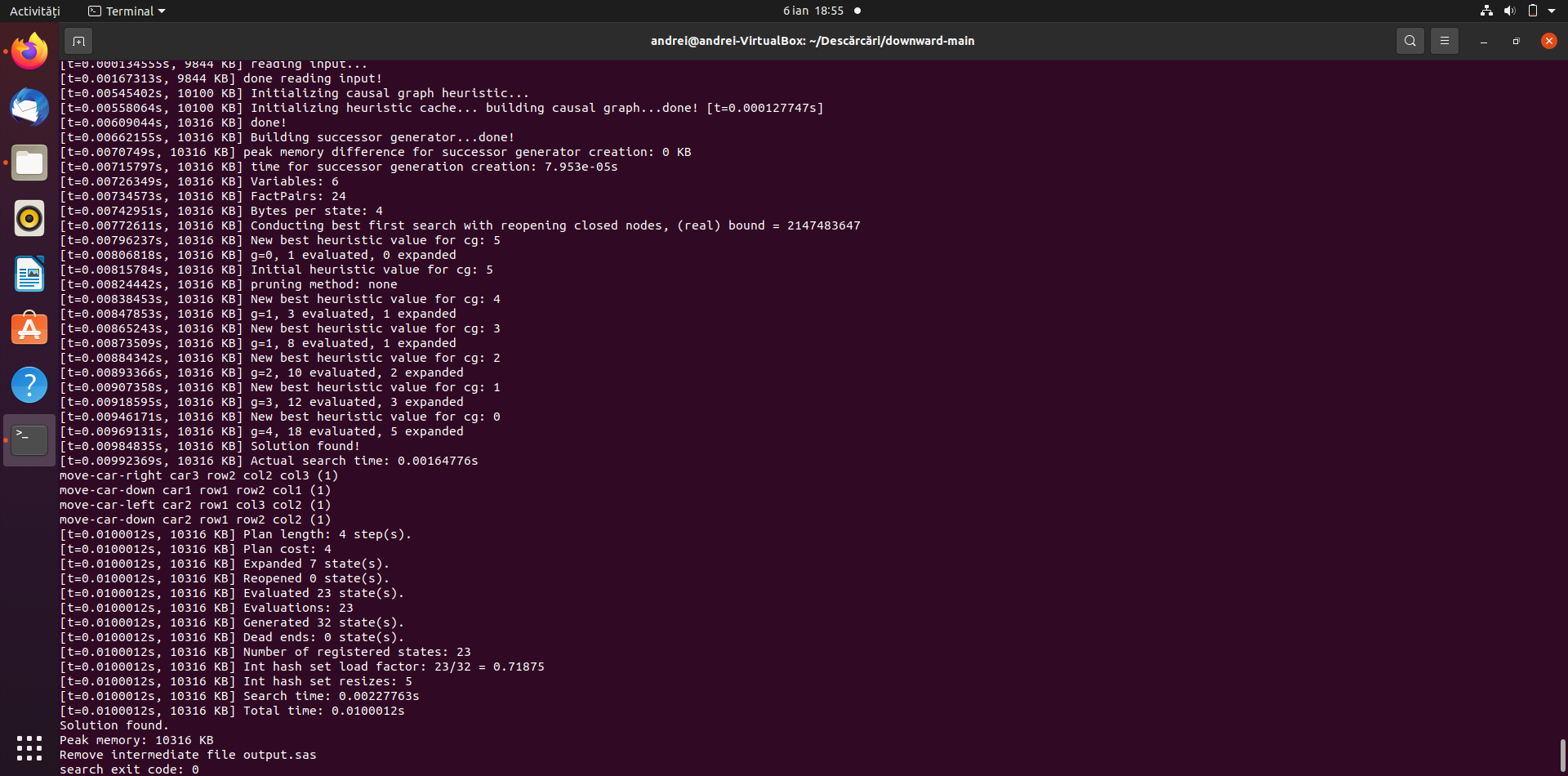
În toate cele 3 cazuri problema a fost rezolvată în 4 pași dar cu timpi diferiti. Totodata soluția din cazurile 1 și 2 (când a fost rulat programul cu euristica astar) difera de soluția obtinuta în cazurile 3 și 4.



./fast-downward.py parking.pddl parking\_p01.pddl --heuristic "h=ff()" --search "astar(h)"

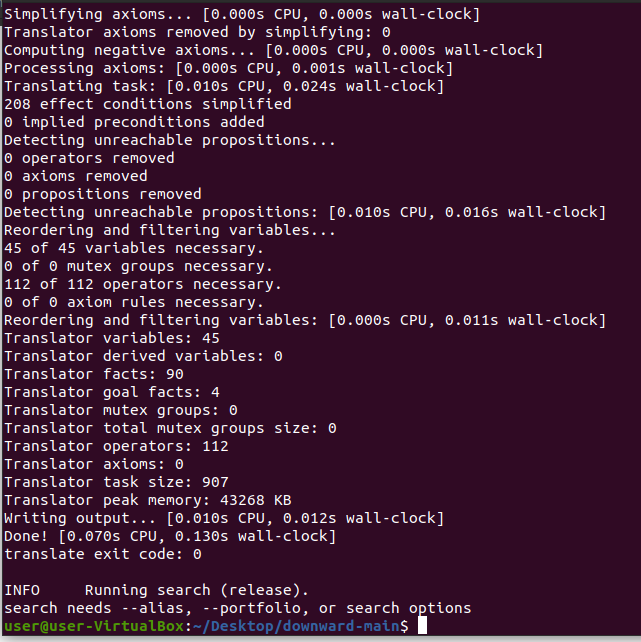
./fast-downward.py parking.pddl parking\_p01.pddl --heuristic "h=ff()" --search "astar(weight(h,3))"

./fast-downward.py parking.pddl parking\_p01.pddl --heuristic "h=cg()" --search "lazy\_wastar([h()], w=1)"

./fast-downward.py parking.pddl parking\_p01.pddl --heuristic "h=cg()" --search "eager\_wastar([h()], w=1)"

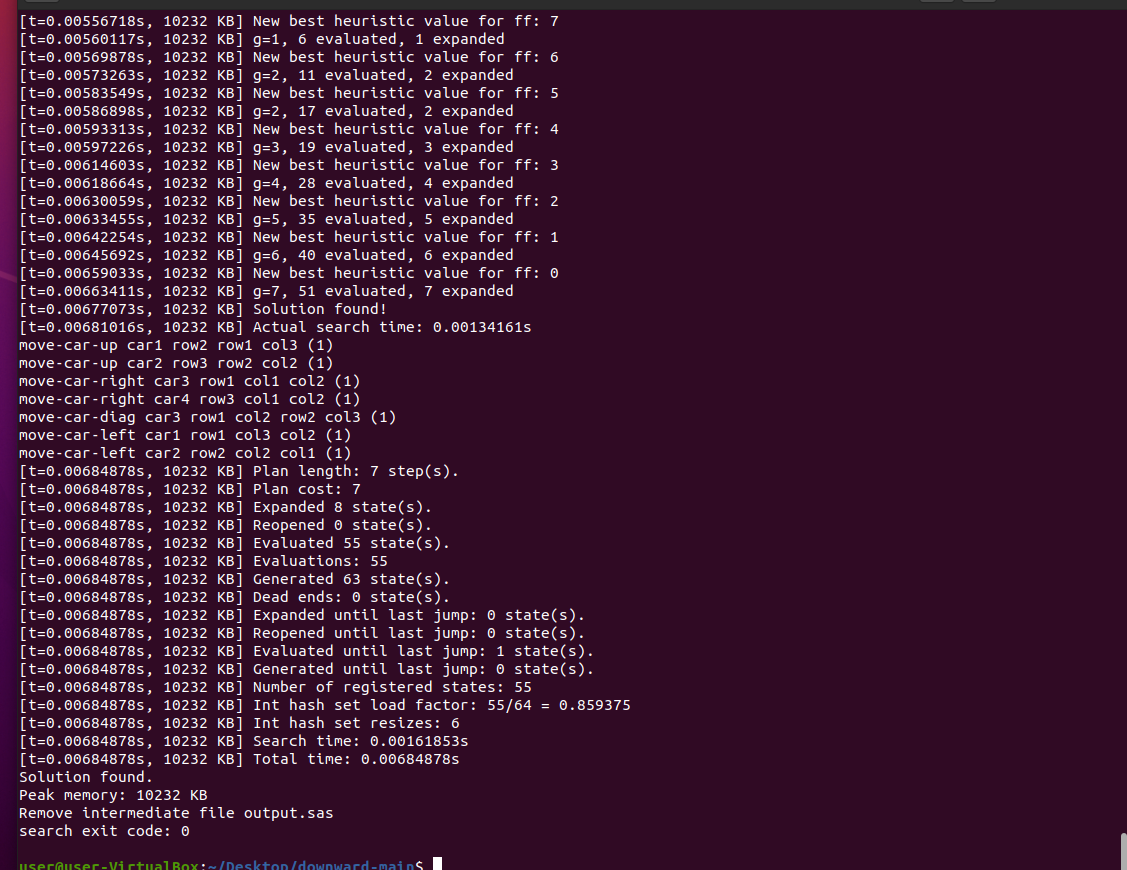
# 3.Etapa avansată a proiectului

Pentru problema cu parking de 3x3 în cazul în care rulez planner-ul fără nicio euristică, obțin următoarele rezultate:



Îmi va genera un fișier de output(nu a reușit să finalizeze task-ul, ca urmare am doar outpus.sas, nu și sas\_plan) care îl regăsiți în materialele anexate sub denumirea de “output.sas(no euristic)” .

În cazul în care rulăm programul având ca euristică h=ff și algoritm de căutare astar(h), atunci consola ne indică următoarele resurse consumate:



În acest caz s-a executat fiecare din cele cinci mișcări puse la dispoziție pentru a atinge goal-ul .

Se poate observa că în acest caz am regăsit o soluție într-un timp de 0.00684s, iar peak memory a fost de 10232 KB. Rezultatul din sas\_plan arată în felul următor:

(move-car-up car1 row2 row1 col3)

(move-car-up car2 row3 row2 col2)

(move-car-right car3 row1 col1 col2)

(move-car-right car4 row3 col1 col2)

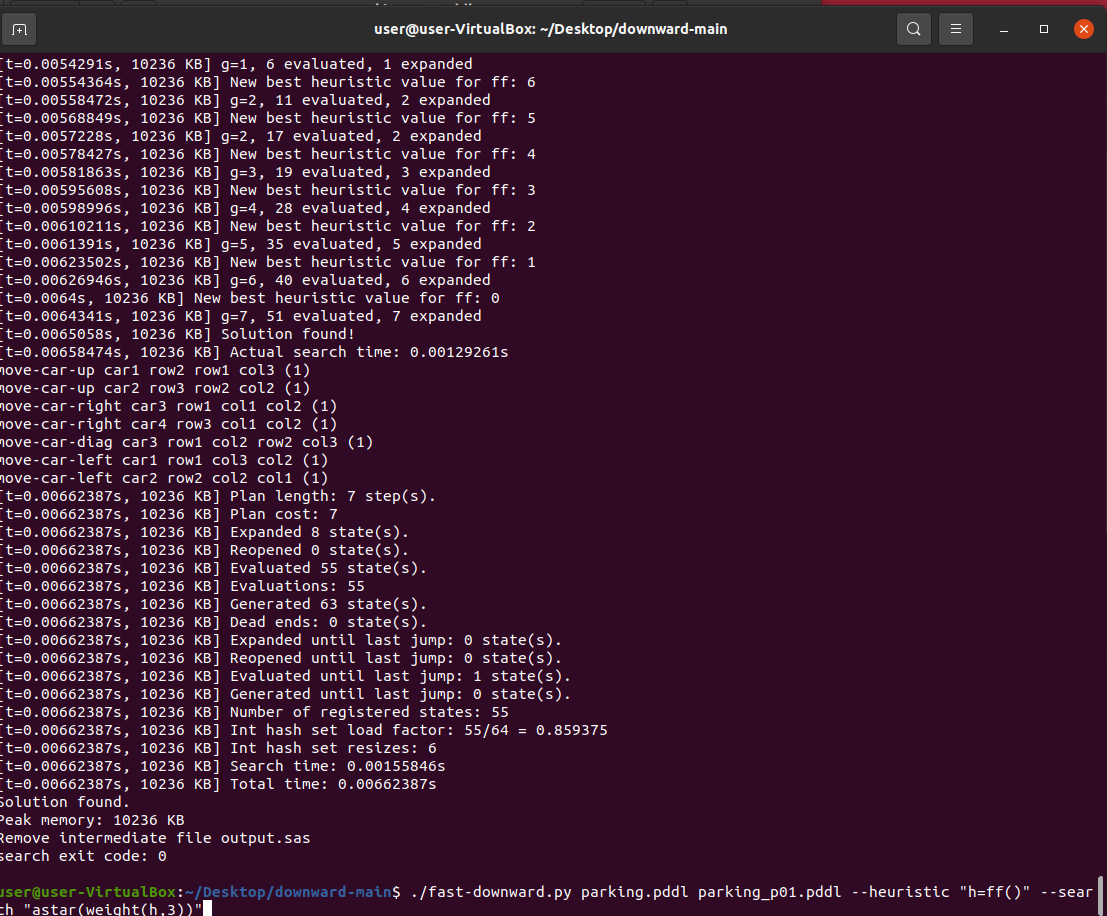
(move-car-diag car3 row1 col2 row2 col3)

(move-car-left car1 row1 col3 col2)

(move-car-left car2 row2 col2 col1)

; cost = 7 (unit cost)

Dacă schimbăm puțin modul de rulare al algoritmului și vom alege să rulăm tot cu astar, dar cu ponderi, utilizând comanda ./fast-downward.py parking.pddl parking\_p01.pddl --heuristic "h=ff()" --search "astar(weight(h,3))" , vom obține următoarele rezultate:



În ceea ce privește numărul de stări expandate, respectiv generate acesta se păstrează ca la variana fără ponderi, dar observăm o ușoară scădere a timpului de search, ca urmare și a timpului total. Timpul de search fiind în acest caz 0,001558 s . Fișierul de out arată în felul următor:

(move-car-up car1 row2 row1 col3)

(move-car-up car2 row3 row2 col2)

(move-car-right car3 row1 col1 col2)

(move-car-right car4 row3 col1 col2)

(move-car-diag car3 row1 col2 row2 col3)

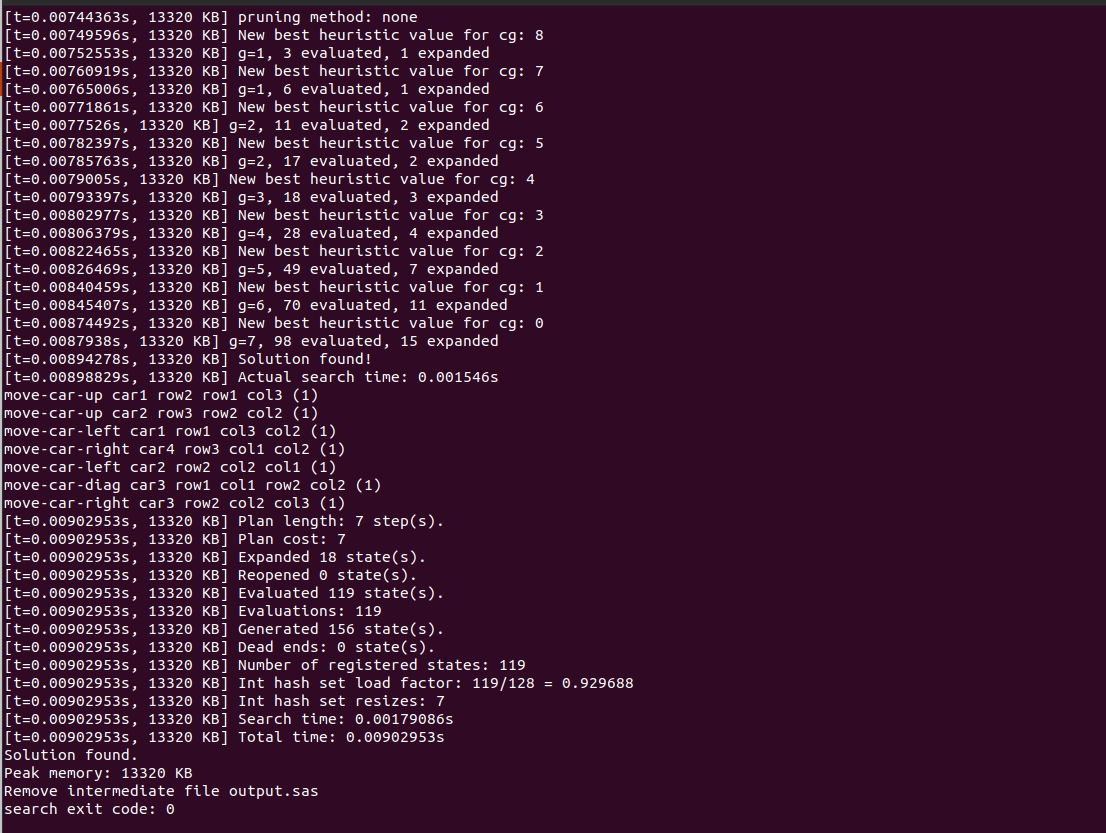
(move-car-left car1 row1 col3 col2)

(move-car-left car2 row2 col2 col1)

; cost = 7 (unit cost)

În ceea ce privește soluția din acest caz, observăm că este identică cu cea din cazul astar neponderat.

În cazul în care vom rula cu euristica h=cg(), iar algoritmul de search va fi eager\_wastar([h()],w=1), atunci rezultatul îmi va arăta mai rău decât cu primele două variante.



Numărul de noduri expandate este de 18, dar stările evaluate sunt în număr de 119, iar stările generate sunt în număr de 156 . Timpul de search este mai mare ajungând la 0,0179s, iar timpul total la 0,00902s. Fișierul de out arată în felul următor:

(move-car-up car1 row2 row1 col3)

(move-car-up car2 row3 row2 col2)

(move-car-left car1 row1 col3 col2)

(move-car-right car4 row3 col1 col2)

(move-car-left car2 row2 col2 col1)

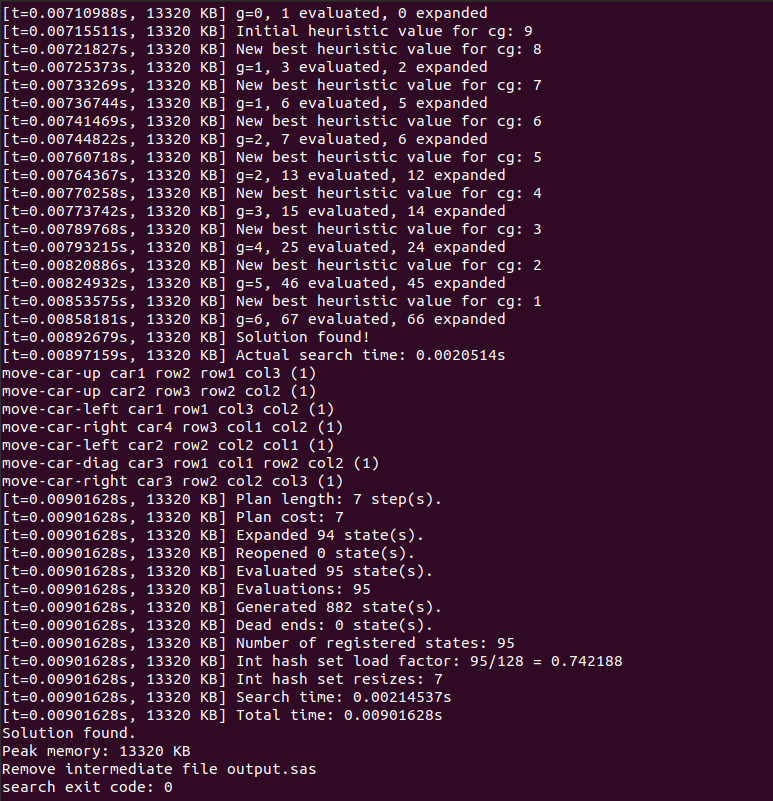
(move-car-diag car3 row1 col1 row2 col2)

(move-car-right car3 row2 col2 col3)

; cost = 7 (unit cost)

Se observă că această soluție este diferită față de celelalte, se realizează mutări diferite, spre exemplu cea de pe diagonală efectuandu-se aici la poziția șase, față de celelalte când se face la poziția cinci. Costul rămâne același, de 7.

În cazul în care voi rula cu ./fast-downward.py parking.pddl parking\_p01.pddl --heuristic "h=cg()" --search "lazy\_wastar([h()], w=1)" , adică cu astar weighted varianta lasy, obțin următoarele rezultate:



În acest caz se poate observa că numărul de stări generate este destul de mare, 882 stări, 94 de stări expandate, iar timpul de search este mai mare decât la variantele precedente, ajungând la 0.00214537s, iar timpul total fiind de 0.00901628s.

(move-car-up car1 row2 row1 col3)

(move-car-up car2 row3 row2 col2)

(move-car-left car1 row1 col3 col2)

(move-car-right car4 row3 col1 col2)

(move-car-left car2 row2 col2 col1)

(move-car-diag car3 row1 col1 row2 col2)

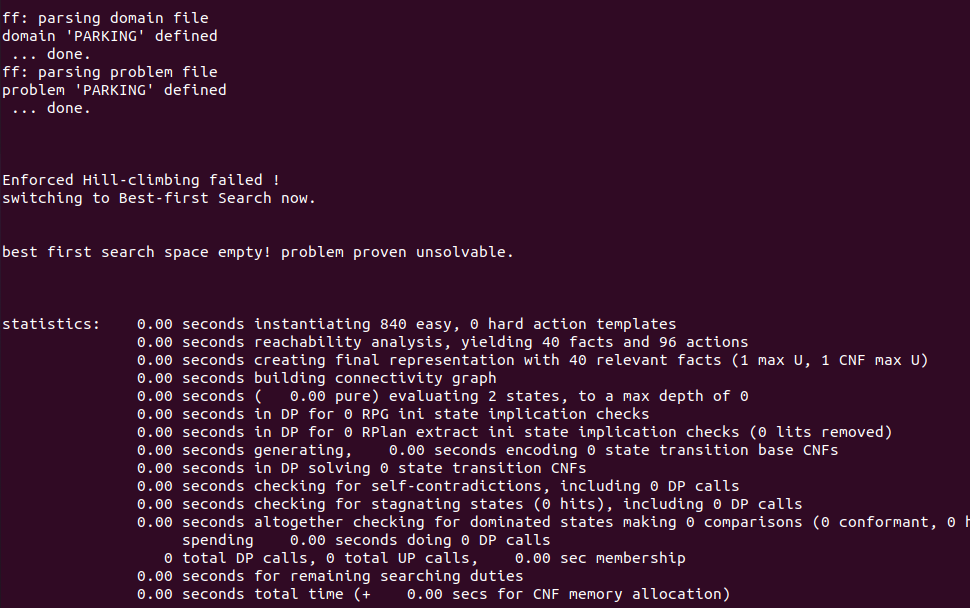
(move-car-right car3 row2 col2 col3)

; cost = 7 (unit cost)

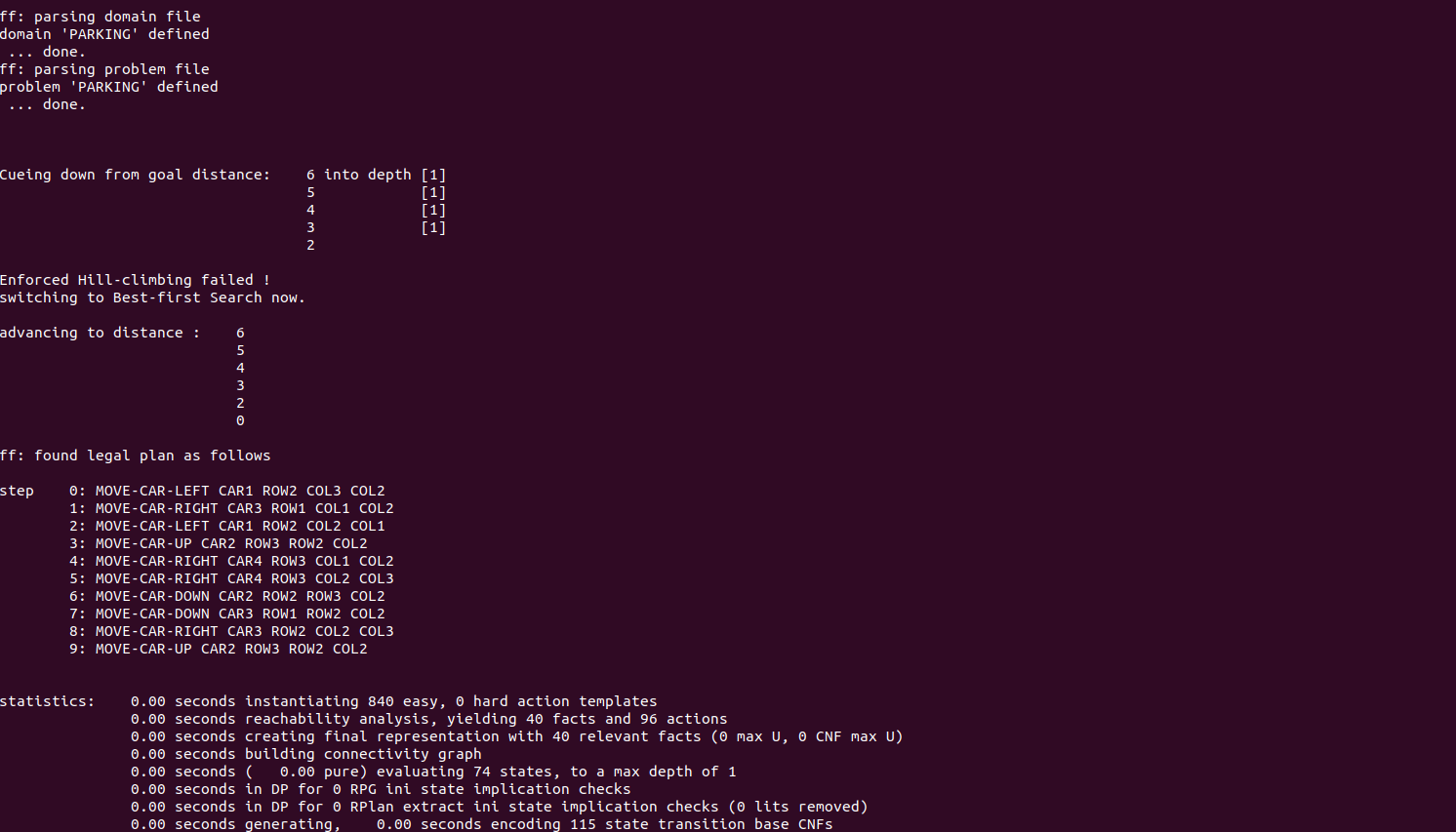
Soluția este identică cu cea anterior prezentată la eager\_wastar.

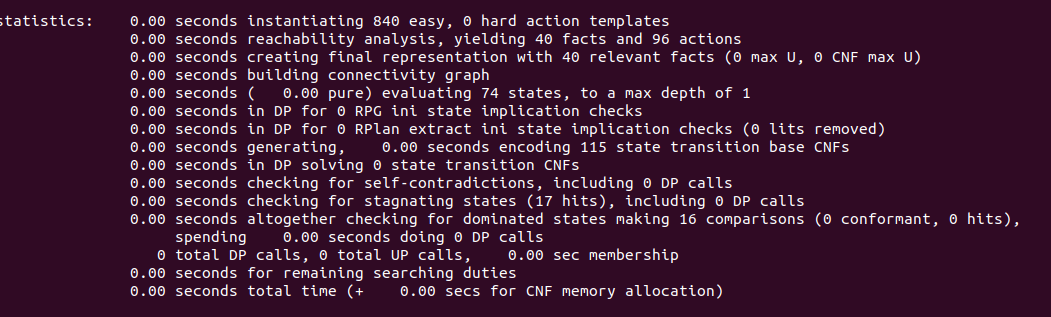
Contingent planning

În cazul în care dorim să rulăm problema cu stări inițiale necunoscute, am presupus următorul caz: nu știm dacă se află mașina3 la poziția (1,1), nici dacă avem liber la pozițiile (1,2) și (2,3). Astfel, după rulare soluția nu a fost găsită.cDe altfel, Enforced Hill-climbing a eșuat, planificatorul a recurs la rezolvarea prin Best-first Search și nu s-a putut, ca urmare a declarat demonstrația problemei ca fiind unsolvable.



Așadar, am presupus un caz mai simplu, în care nu știm doar dacă sunt libere pozițiile (1,2) , respectiv (2,3). În acest caz, planificatorul se descurcă mai bine astfel avem următorul rezultat:





Se observă că în acest caz ff a găsit un plan. În schimb, dacă ne dorim să facem mai mult de două poziții libere necunoscute, ff va avea probleme în găsirea planului.

Bibliography

Introduction to Artificial Intelligence - Adrian Groza, Radu Razvan Slavescu and Anca Marginean