

UNIVERSITÀ DELLA CALABRIA

DIPARTIMENTO DI INGEGNERIA INFORMATICA, MODELLISTICA,
ELETTRONICA E SISTEMISTICA



Corso di Laurea Magistrale in
**Ingegneria Informatica - Indirizzo Artificial Intelligence
and Machine Learning**

Relazione Progetto Intelligenza Artificiale

Candidati:

Matteo Capalbo - Davide Di Stefano - Nicola Gabriele

ANNO ACCADEMICO 2022/2023

Indice

Indice	1
1 Modellazione del problema	2
1.1 Definizione del dominio in PDDL1.2	2
1.1.1 Definizione dei tipi	2
1.1.2 Definizione dei predicati	3
1.1.3 Definizione delle azioni	5
2 Algoritmo di Planning e definizione di una Euristica	11
2.1 Introduzione	11
2.2 Modifica del metodo di ricerca A*	11
2.3 Euristica dipendente dal dominio	12
2.4 Plan generato e prestazioni	14
2.4.1 Istanza 1	14
2.4.2 Istanza 2	15
2.4.3 Istanza 3	17
3 Planning Temporale e implementazione del problema in ROS2	20
3.1 Planning Temporale	20
3.1.1 Trasformazione del Dominio	20
3.1.2 Definizione del file di Problema	26
3.2 Robotics Planning	27

1 Modellazione del problema

Viene richiesto di modellare, utilizzando il linguaggio PDDL, un problema logistico-emergenziale in cui un gruppo di robot deve consegnare presso alcune postazioni, in cui si trovano persone, delle casse contenenti diverse risorse (medicine, cibo, strumenti etc...). Vengono fornite alcune informazioni di base quali: le persone non si spostano dalla propria locazione, i robot trasportano le casse tramite cassette, le casse possono essere riempite con qualsiasi tipo di contenuto e che le persone possono necessitare di uno o più tipi di contenuti.

1.1 Definizione del dominio in PDDL1.2

Di seguito saranno illustrate, argomentate e motivate le scelte di progettazione in termini di: tipi, predicati e azioni utilizzate per la modellazione del dominio del problema. Inoltre, per la modellazione del dominio sono stati utilizzati i seguenti requirements PDDL:

- **strips**
- **typing**

1.1.1 Definizione dei tipi

Al fine di modellare le caratteristiche specifiche del dominio in esame, sono stati definiti diversi tipi ausiliari:

- **content**: modella il contenuto che è possibile inserire all'interno delle box; medicine, strumenti e cibo saranno istanze del tipo content.
- **location**: modella i luoghi fisici in cui persone, scatole, vettori e altri oggetti del dominio possono trovarsi
- **person**: modella le persone a cui devono essere consegnate le risorse

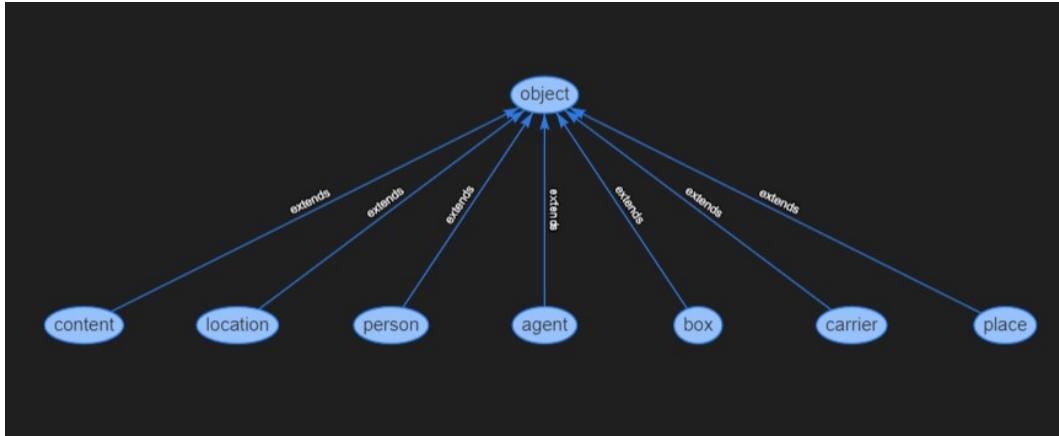


Figura 1: gerarchia dei tipi definiti

- **agent**: modella i robot che si occupano di eseguire le azioni per consegnare le risorse
- **box**: modella le cassette in cui vengono riposte le risorse per essere trasportate
- **carrier**: modella i vettori su cui i robot caricano le cassette contenenti le risorse per trasportarle tra le varie location
- **place**: modellano i posti disponibili su ciascun carrier, questo tipo non modella un oggetto fisico ma è utile al fine di rappresentare la limitatezza dei posti disponibili su ciascun vettore di trasporto

1.1.2 Definizione dei predicati

La modellazione dello stato del dominio viene definito da un insieme di predicati logici. Nel caso in esame si è reso necessario definire i seguenti:

- **(in ?o - object ?l - location)**: risulta soddisfatto quando l'oggetto **o** si trova nella location **l**, in questo caso il predicato deve poter essere applicato ad uno qualsiasi degli oggetti presenti nel dominio, pertanto il primo argomento è il tipo più in alto nella gerarchia.

```

(:predicates
  (in ?o - object ?l - location) 16 3 3
  (need ?p - person ?c - content) 2 2
  (has ?p - person ?c - content) 1
  (hasSomething ?p - person) 1 ;gestione or
  (inBox ?b - box ?c - content) 2 1 2
  (emptyBox ?b - box) 1 2 1
  (boxOnPlace ?b - box ?p - place) 1 1 1
  (emptyPlace ?p - place) 1 1 1 ;per gestire la load
  (fullPlace ?p - place) 1 1 1 ;per gestire l'unload
  (placeOnCarrier ?p - place ?c - carrier) 2
  (needAll ?p - person) 1 ;gestione or
  (needSomething ?p - person) 1 1
)

```

Figura 2: predicati

- **(need ?p - person ?c - content)**; risulta soddisfatto se la persona **p** necessita della risorsa **c** (che nel caso in esame può essere cibo, medicine o strumenti).
- **(has ?p - person ?c - content)**; risulta soddisfatto se la persona **p** è in possesso della risorsa **c**.
- **(hasSomething ?p - person)**; risulta soddisfatto se la persona **p** possiede almeno una delle risorse di cui necessita (questo predicato si rende necessario per la gestione di quelle persone che necessitano di un insieme di risorse ma risulteranno soddisfatte già nel momento in cui una qualsiasi di queste gli venga recapitata).
- **(inBox ?b - box ?c - content)**; risulta soddisfatto se la box **b** contiene la risorsa **c**.
- **(emptyBox ?b - box)**; risulta soddisfatto se la box **b** è vuota.

- (**boxOnPlace ?b - box ?p - place**); risulta soddisfatto se la box **b** sta occupando il posto **p** (quindi si trova carica sul vettore di appartenenza del posto **p**).
- (**emptyPlace ?p - place**); risulta soddisfatto se il posto **p** è vuoto.
- (**fullPlace ?p - place**); risulta soddisfatto se il posto **p** è occupato.
- (**placeOnCarrier ?p - place ?c - carrier**); risulta soddisfatto se il posto **p** appartiene al vettore **c**.
- (**needAll ?p - person**); risulta soddisfatto se la persona **p** sarà soddisfatta solo quando tutte le risorse di cui necessita le saranno recapitate.
- (**needSomething ?p - person**); risulta soddisfatto se la persona **p** sarà soddisfatta nel momento in cui almeno una delle risorse di cui necessita le sarà recapitata.

1.1.3 Definizione delle azioni

Abbiamo modellato la morfologia del dominio attraverso la definizione dei tipi, ne abbiamo modellato la rappresentazione dello stato attraverso la definizione di predicati. Rimangono da definire le azioni attraverso cui lo stato del dominio si trasforma. Di seguito vengono descritte brevemente le azioni presenti.

move

modella l'azione attraverso la quale, un robot (a bordo di un vettore), si sposta tra 2 location.

- **parametri:** l'azione coinvolge:
 - il robot che vuole spostarsi.
 - il carrier con cui il robot si sposta.
 - la location di partenza.
 - la location di destinazione.

- **precondizioni:** affinché la move possa avere luogo deve valere che:
 - il robot deve trovarsi nella location indicata nei parametri.
 - il carrier deve trovarsi nella location indicata nei parametri.
- **effetti:** l'azione move ha i seguenti effetti:
 - il robot non si trova più nella location di partenza e si trova nella location di destinazione.
 - il carrier non si trova più nella location di partenza e si trova nella location di destinazione.

load

Modella l'azione con cui una box viene carita sul vettore per essere trasportata verso un'altra location.

- **parametri:** l'azione coinvolge:
 - il robot che effettua il caricamento.
 - la box oggetto del caricamento.
 - il carrier su cui la box viene caricata.
 - il posto sul carrier che verrà occupato dalla box.
 - la location in cui avviene l'azione.
- **precondizioni:** affinché la load possa avvenire deve valere che:
 - il robot deve trovarsi nella location indicata nei parametri.
 - il carrier deve trovarsi nella location indicata nei parametri.
 - la box deve trovarsi nella location indicata nei parametri.
 - il posto su cui si sta caricando la box deve appartenere al carrier specificato.
 - il posto su cui si sta caricando la box deve essere vuoto.
- **effetti:** la load ha i seguenti effetti:

- la box occupa il posto sul carrier.
- lo stato del posto passa da "vuoto" a "pieno/occupato".
- la box non si trova più nella location indicata nei parametri.

unload

modella l'azione che permette al robot di scaricare una box dal carrier.

- **parametri:** la unload coinvolge:
 - il robot che effettua lo scarico.
 - la box oggetto dello scarico.
 - il carrier dal quale la box viene scaricata.
 - il posto sul carrier attualmente occupato dalla box che deve essere scaricata.
 - la location in cui avviene l'azione.
- **precondizioni:** affinché la unload possa avvenire deve valere che:
 - il robot deve trovarsi nella location indicata nei parametri.
 - il carrier deve trovarsi nella location indicata nei parametri.
 - la box deve occupare il posto specificato.
 - il posto dal quale si sta scaricando la box deve appartenere al carrier specificato.
 - il posto su cui si sta caricando la box deve essere occupato.
- **effetti:** la unload ha i seguenti effetti:
 - la box non occupa più il posto.
 - la box si trova nella location in cui lo scarico è avvenuto.
 - il posto non è più occupato/pieno.
 - il posto è vuoto.

vent

Modella l'azione con cui il robot svuota una box dalle risorse da consegnare (nel caso di persone che richiedono obbligatoriamente la consegna di tutte le risorse di cui necessitano).

- **parametri:** la vent coinvolge:
 - l'agente che svuota la box.
 - la box che viene svuotata.
 - il luogo in cui l'azione avviene.
 - il contenuto estratto dalla box.
 - la persona a cui il contenuto viene consegnato.
- **precondizioni:** affinché la vent possa avvenire deve valere che:
 - la risorsa specificata deve essere contenuto della box.
 - il robot deve trovarsi nella location specificata.
 - la box deve trovarsi nella location specificata.
 - la persona deve trovarsi nella location specificata.
 - la persona deve necessitare del tipo di contenuto. estratto dalla box.
 - la persona vuole la consegna di tutto ciò di cui necessita.
- **effetti:** gli effetti della vent sono i seguenti:
 - la risorsa non è più nella box.
 - la box è vuota.
 - la persona non necessita più della risorsa consegnata.
 - la persona possiede la risorsa consegnata.

ventOr

Modella l'azione con cui il robot svuota una box dalle risorse da consegnare (nel caso di persone che richiedono la consegna di almeno una delle risorse di cui necessitano).

- **parametri:** la ventOr coinvolge:
 - l'agente che svuota la box.
 - la box che viene svuotata.
 - il luogo in cui l'azione avviene.
 - il contenuto estratto dalla box.
 - la persona a cui il contenuto viene consegnato.
- **precondizioni:** affinché la ventOr possa avvenire deve valere che:
 - la risorsa specificata deve essere contenuto della box.
 - il robot deve trovarsi nella location specificata.
 - la box deve trovarsi nella location specificata.
 - la persona deve trovarsi nella location specificata.
 - la persona deve necessitare del tipo di contenuto estratto dalla box.
 - la persona vuole la consegna di almeno una delle risorse di cui necessita.
- **effetti:** gli effetti della ventOr sono i seguenti:
 - la risorsa non è più nella box.
 - la box è vuota.
 - la persona non necessita più di alcuna risorsa.
 - la persona possiede almeno una delle risorse di cui necessitava.

fill

Modella l'azione con cui una box viene riempita

- **parametri:** la fill coinvolge:
 - l'agente che riempie la box.
 - la box che viene riempita.

- il luogo in cui l'azione avviene.
 - il contenuto inserito nella box.
- **precondizioni:** affinché la fill possa avvenire deve valere che:
 - la box è vuota.
 - il robot deve trovarsi nella location specificata.
 - la box deve trovarsi nella location specificata.
 - il contenuto da inserire nella box deve trovarsi nella location specificata.
 - **effetti:** la fill ha i seguenti effetti:
 - la box non è più vuota.
 - la risorsa si trova all'interno della box.

```
(:action move
:parameters (?a - agent ?c - carrier ?from ?to - location)
:precondition (and (in ?a ?from) (in ?c ?from))
:effect (and (in ?a ?to) (in ?c ?to) (not (in ?a ?from)) (not (in ?c ?from)))
)

(:action load
:parameters (?a - agent ?c - carrier ?p - place ?b - box ?l - location)
:precondition (and (in ?a ?l) (in ?c ?l) (in ?b ?l) (placeOnCarrier ?p ?c) (emptyPlace ?p) )
:effect (and (boxOnPlace ?b ?p) (not (in ?b ?l)) (not (emptyPlace ?p)) (fullPlace ?p) )
)

(:action unload
:parameters (?a - agent ?c - carrier ?p - place ?b - box ?l - location)
:precondition (and (in ?a ?l) (in ?c ?l) (boxOnPlace ?b ?p) (placeOnCarrier ?p ?c) (fullPlace ?p) )
:effect (and (not (boxOnPlace ?b ?p)) (in ?b ?l) (not (fullPlace ?p)) (emptyPlace ?p) )
)

(:action vent
:parameters (?a - agent ?b - box ?c - content ?p - person ?l - location)
:precondition (and (inBox ?b ?c) (in ?a ?l) (in ?b ?l) (in ?p ?l) (need ?p ?c) (needAll ?p) )
:effect (and (not (inBox ?b ?c)) (emptyBox ?b) (not (need ?p ?c)) (has ?p ?c))
)

(:action ventOr ;gestione or
:parameters (?a - agent ?b - box ?c - content ?p - person ?l - location)
:precondition (and (inBox ?b ?c) (in ?a ?l) (in ?b ?l) (in ?p ?l) (need ?p ?c) (needSomething ?p))
:effect (and (not (inBox ?b ?c)) (emptyBox ?b) (not (need ?p ?c)) (not (needSomething ?p)) (hasSomething ?p) )
)

(:action fill
:parameters (?a - agent ?b - box ?c - content ?l - location)
:precondition (and (emptyBox ?b) (in ?a ?l) (in ?b ?l) (in ?c ?l))
:effect (and (not (emptyBox ?b)) (inBox ?b ?c))
)
```

Figura 3: azioni

2 Algoritmo di Planning e definizione di una Euristica

2.1 Introduzione

La realizzazione del planner per la generazione dei piani delle tre istanze da risolvere è stata effettuata grazie alla libreria *PPDL4J*. Sebbene *PDDL4J* metta a disposizione diverse euristiche con cui poter combinare gli algoritmi di ricerca, sfruttando le classi messe a disposizione da tale libreria, abbiamo realizzato un apposito algoritmo di ricerca e un'euristica adeguata e funzionale per il tipo di dominio a nostra disposizione. Sulla base di ciò che ci offre la libreria, abbiamo testato anche le istanze con le euristiche *Max* e *Sum*, attribuendo a queste un certo peso. Consapevolmente abbiamo deciso di rinunciare a ottenere dei piani ottimi, questo perché ci siamo accorti, specie con istanze più difficili da risolvere, che l'algoritmo di ricerca non terminasse entro i 10 minuti prestabiliti nella traccia. Per questa ragione, abbiamo deciso di sacrificare l'ammissibilità dell'euristica (come detto prima attribuendo anche un peso maggiore di 1 sul valore restituito dalle euristiche), per trovare però in un tempo accettabile i piani che soddisfacessero le istanze.

2.2 Modifica del metodo di ricerca A*

Il metodo di ricerca sviluppato, come introdotto precedentemente, prevede una modifica di A*. Studiando la struttura del problema e in particolare la composizione del *goal*, abbiamo intuito che, specialmente per le istanze con più *subgoals* da risolvere, questi ultimi potessero essere risolti in modo indipendente. L'assunzione nasce dal fatto che, poiché non abbiamo priorità di consegna e che la consegna di un oggetto a una persona non influisce sulle altre consegne da effettuare, i *subgoals* che compongono il *goal* sono indipendenti tra di loro. Questo ci ha permesso di trattare la risoluzione dei vari *subgoals* singolarmente e in modo sequenziale. Quindi il planner, in un primo momento

suddivide il goal principale in *subgoals* più piccoli, in seguito avvia ogni volta un'esecuzione di A^* per risolvere in modo sequenziale il gruppo di *subgoals*. Il risultato è che ad ogni iterazione di A^* , otteniamo dei *subplans* che risolvono i *subgoals* suddivisi dal goal originale. Il piano completo viene così ricavato effettuando un *merge* di tutti i *subplans* generati. Il planner, così costruito, può ricevere un parametro che è il numero di suddivisioni da effettuare sul *goal* originale. Ovviamente anche in questo caso è possibile stabilire per A^* le euristiche già definite dalla libreria di *PDDL4J* oltre all'euristica da noi definita, che descriviamo nella prossima sezione.

2.3 Euristica dipendente dal dominio

Oltre alla definizione di un planner specifico, ci siamo adoperati per creare un'euristica che potesse essere adatta per risolvere il particolare tipo di problema delle consegne di oggetti alle persone. Il funzionamento dell'euristica è molto semplice: il calcolo della distanza dal goal, somma diverse quantità qui descritte:

1. Conteggio del numero di subgoals che ancora devono essere soddisfatti.
2. Conteggio del numero di box vuote.
3. Conteggio del numero di box riempite con oggetti inutili (per oggetti "inutili" si intendono oggetti di cui le persone non hanno bisogno ma che si trovano all'interno delle box).

Di seguito, riportiamo il metodo *solve* del planner e il metodo *estimate* dell'euristica da noi definita. Nei prossimi paragrafi sono mostrati tutti i plan generati dalle diverse tecniche da noi testate, con annessi tempi di ricerca, numero di azioni che compongono i piani e occupazione della memoria.

```

@Override
public Plan solve(Problem problem) throws ProblemNotSupportedException {
    //array di utilità
    Problem[] subProblems = split(problem,split);
    Node[] subSolutions = new Node[subProblems.length];
    Plan[] subplans = new Plan[subProblems.length];

    //parametri algoritmi
    SearchStrategy.Name strategyName = SearchStrategy.Name.ASTAR;
    StateHeuristic.Name heuristic = StateHeuristic.Name.EMERGENCY_PROBLEM;
    int timeout = 1000000;

    //inizialmente usiamo A*
    StateSpaceSearch alg0 = StateSpaceSearch.getInstance(strategyName, heuristic, heuristic_weigth);
    subSolutions[0] = alg0.searchSolutionNode(subProblems[0]);
    subplans[0] = alg0.extractPlan(subSolutions[0],subProblems[0]);
    System.out.println("piano 0: ");
    if(subplans[0] != null)
        printPlan(subplans[0],problem);
    else System.out.println("piano nullo");
    System.out.println("fine piano");
    //risolviamo i sottoproblemi rimanenti usando ModifiedAStar
    for(int i = 1; i<subProblems.length; i++){
        subSolutions[i-1].setParent(null);
        StateSpaceSearch alg = new ModifiedAStar(timeout,heuristic,heuristic_weigth,subSolutions[i-1]);
        subSolutions[i] = alg.searchSolutionNode(subProblems[i]);
        subplans[i] = alg.extractPlan(subSolutions[i],subProblems[i]);
        System.out.printf("piano %d: \n",i);
        if(subplans[i] != null)
            printPlan(subplans[i],problem);
        else System.out.println("piano nullo");
        System.out.println("fine piano");
    }
    List<Action> sol = new LinkedList<>();
    for(Plan p: subplans)
        if(p != null)
            sol.addAll(p.actions());
    Plan solution = new SequentialPlan();
    Iterator<Action> it = sol.iterator();
    for(int i = 0; i<sol.size(); i++)
        solution.add(i,it.next());
    return solution;
}

```

Figura 4: Metodo *solve*

```

@Override
public int estimate(State state, Condition goal) {
    super.setGoal(goal);
    this.expandRelaxedPlanningGraph(state);
    boxConstraints(state,goal);
    BitVector positiveFluents = goal.getPositiveFluents();
    int count = 0;
    for(int i = positiveFluents.nextSetBit(0); i>=0; i = positiveFluents.nextSetBit(i+1)){
        BitVector tmp = new BitVector();
        tmp.set(i);
        if(state.satisfy(new Condition(tmp,new BitVector()))count++;
    }
    BitVector negativeFluents = goal.getNegativeFluents();
    for(int i = negativeFluents.nextSetBit(0); i>=0; i = negativeFluents.nextSetBit(i+1)){
        BitVector tmp = new BitVector();
        tmp.set(i);
        if(state.satisfy(new Condition(tmp,new BitVector()))count++;
    }
    int value = goal.cardinality() - count;

    return value+boxConstraints(state,goal)+countSatisfied(state, "emptybox");
}

```

Figura 5: Metodo *estimate*

2.4 Plan generato e prestazioni

2.4.1 Istanza 1

```
00: (      fill a1 b1 food depot)
01: (      fill a1 b2 food depot)
02: ( fill a1 b4 medicine depot)
03: ( fill a1 b5 medicine depot)
04: (load a1 c1 place1 b5 depot)
05: (load a1 c1 place2 b2 depot)
06: (load a1 c1 place3 b4 depot)
07: (load a1 c1 place4 b1 depot)
08: (      move a1 c1 depot l2)
09: ( unload a1 c1 place2 b2 l2)
10: (      vent a1 b2 food p3 l2)
11: (      move a1 c1 l2 l1)
12: ( unload a1 c1 place3 b4 l1)
13: ( vent a1 b4 medicine p1 l1)
14: ( unload a1 c1 place4 b1 l1)
15: (      vent a1 b1 food p1 l1)
16: ( unload a1 c1 place1 b5 l1)
17: ( vent a1 b5 medicine p2 l1)
```

Figura 6: Piano dell'istanza 1

INSTANCE 1					
HEURISTIC	WEIGHT	SPLIT	TIME (s)	N. ACTIONS	MEMORY (Mb)
MAX	1	NO	>10 min	---	---
MAX	5	NO	576	19	0,73
MAX	10	NO	186	21	0,73
SUM	1	NO	259	23	0,73
SUM	5	NO	13	23	0,73
SUM	10	NO	10	23	0,73
OUR_OWN	1	NO	OFM	---	---
OUR_OWN	5	NO	OFM	---	---
OUR_OWN	10	NO	598	17	0,73

Figura 7: Prestazioni sull'istanza 1

2.4.2 Istanza 2

```
00: ( fill a1 b1 medicine depot)
01: ( fill a1 b2 tools depot)
02: ( load a1 c2 place3 b2 depot)
03: ( move a1 c2 depot l1)
04: ( unload a1 c2 place3 b2 l1)
05: ( vent a1 b2 tools p1 l1)
06: ( load a2 c1 place1 b1 depot)
07: ( move a2 c1 depot l4)
08: ( unload a2 c1 place1 b1 l4)
09: ( vent a2 b1 medicine p5 l4)
10: ( load a1 c2 place3 b2 l1)
11: ( move a1 c2 l1 depot)
12: ( fill a1 b3 tools depot)
13: (unload a1 c2 place3 b2 depot)
14: ( fill a1 b2 medicine depot)
15: ( load a1 c2 place3 b2 depot)
16: ( load a1 c2 place4 b3 depot)
17: ( move a1 c2 depot l4)
18: ( unload a1 c2 place4 b3 l4)
19: ( move a2 c2 l4 l1)
20: ( unload a2 c2 place3 b2 l1)
21: ( vent a2 b2 medicine p2 l1)
22: ( vent a1 b3 tools p5 l4)
23: ( load a1 c1 place1 b1 l4)
24: ( load a1 c1 place2 b3 l4)
25: ( move a1 c1 l4 depot)
26: (unload a1 c1 place2 b3 depot)
27: ( fill a1 b3 medicine depot)
28: (unload a1 c1 place1 b1 depot)
29: ( fill a1 b1 food depot)
30: ( load a1 c1 place1 b1 depot)
31: ( load a1 c1 place2 b3 depot)
32: ( move a1 c1 depot l5)
33: ( unload a1 c1 place1 b1 l5)
34: ( vent a1 b1 food p6 l5)
35: ( move a1 c1 l5 l2)
36: ( unload a1 c1 place2 b3 l2)
37: ( vent a1 b3 medicine p3 l2)
38: ( load a1 c1 place1 b3 l2)
39: ( move a1 c1 l2 depot)
40: (unload a1 c1 place1 b3 depot)
41: ( fill a1 b3 medicine depot)
42: ( load a2 c2 place3 b2 l1)
43: ( move a2 c2 l1 depot)
44: (unload a1 c2 place3 b2 depot)
45: ( fill a1 b2 medicine depot)
46: ( load a1 c1 place2 b2 depot)
47: ( load a1 c2 place3 b3 depot)
48: ( move a2 c2 depot l3)
49: ( unload a2 c2 place3 b3 l3)
50: ( move a1 c1 depot l5)
51: ( vent a2 b3 medicine p4 l3)
52: ( unload a1 c1 place2 b2 l5)
53: ( vent a1 b2 medicine p6 l5)
54: ( load a1 c1 place1 b1 l5)
55: ( load a1 c1 place2 b2 l5)
56: ( move a1 c1 l5 depot)
57: (unload a1 c1 place1 b1 depot)
58: ( fill a1 b1 tools depot)
59: (unload a1 c1 place2 b2 depot)
60: ( fill a1 b2 food depot)
61: ( load a1 c1 place1 b2 depot)
62: ( load a1 c1 place2 b1 depot)
63: ( move a1 c1 depot l3)
64: ( unload a1 c1 place1 b2 l3)
65: ( move a1 c1 l3 l5)
66: ( unload a1 c1 place2 b1 l5)
67: ( vent a1 b1 tools p6 l5)
68: ( vent a2 b2 food p4 l3)
69: ( load a1 c1 place1 b1 l5)
70: ( move a1 c1 l5 depot)
71: (unload a1 c1 place1 b1 depot)
72: ( fill a1 b1 food depot)
73: ( load a1 c1 place1 b1 depot)
74: ( move a1 c1 depot l4)
75: ( unload a1 c1 place1 b1 l4)
76: ( vent a1 b1 food p5 l4)
```

Figura 8: Piano dell'istanza 2

INSTANCE 2					
HEURISTIC	WEIGHT	SPLIT	TIME (s)	N. ACTIONS	MEMORY (Mb)
MAX	1	NO	OFM	---	---
MAX	1	2	>10 min	---	---
MAX	1	6	252	78	1,98
MAX	5	2	>10 min	---	---
MAX	5	6	67	80	1,98
MAX	10	2	OFM	---	---
MAX	10	6	47	80	1,98
SUM	1	2	OFM	---	---
SUM	1	6	60	78	1,98
SUM	5	2	5	80	1,98
SUM	5	6	4	80	1,98
SUM	10	2	5	80	1,98
SUM	10	6	4	80	1,98
OUR_OWN	1	2	>10 min	---	---
OUR_OWN	1	6	>10 min - 960s	78	1,98
OUR_OWN	5	2	>10 min	---	---
OUR_OWN	5	6	149	77	1,98
OUR_OWN	10	2	>10 min	---	---
OUR_OWN	10	6	28	82	1,98

Figura 9: Prestazioni sull'istanza 2

2.4.3 Istanza 3

```
000: (    fill a1 b1 tools depot) 039: (    move a1 c2 l3 depot)
001: (  load a1 c1 place2 b1 depot) 040: (unload a1 c2 place3 b2 depot)
002: (    move a2 c1 depot l1) 041: (    fill a1 b2 food depot)
003: (  unload a2 c1 place2 b1 l1) 042: (  load a1 c2 place3 b2 depot)
004: (    fill a1 b4 tools depot) 043: (    move a1 c2 depot l4)
005: (  ventor a2 b1 tools p1 l1) 044: (  unload a1 c2 place3 b2 l4)
006: (  load a1 c2 place3 b4 depot) 045: (    vent a1 b2 food p5 l4)
007: (    move a1 c2 depot l7) 046: (  load a2 c1 place2 b3 l3)
008: (  unload a1 c2 place3 b4 l7) 047: (    move a2 c1 l3 depot)
009: (    vent a1 b4 tools p8 l7) 048: (unload a2 c1 place2 b3 depot)
010: (    move a1 c2 l7 depot) 049: (  fill a2 b3 medicine depot)
011: (  fill a1 b3 medicine depot) 050: (  load a2 c1 place2 b3 depot)
012: (  load a1 c2 place3 b3 depot) 051: (    move a2 c1 depot l4)
013: (    move a1 c2 depot l1) 052: (  unload a1 c1 place2 b3 l4)
014: (  unload a1 c2 place3 b3 l1) 053: (  vent a1 b3 medicine p5 l4)
015: (  vent a1 b3 medicine p2 l1) 054: (  load a1 c2 place4 b3 l4)
016: (    move a1 c1 l1 depot) 055: (    move a2 c2 l4 depot)
017: (  fill a1 b2 medicine depot) 056: (unload a2 c2 place4 b3 depot)
018: (  load a1 c1 place1 b2 depot) 057: (    fill a2 b3 tools depot)
019: (    move a1 c1 depot l2) 058: (  load a2 c2 place3 b3 depot)
020: (  unload a1 c1 place1 b2 l2) 059: (    move a2 c2 depot l4)
021: (  vent a1 b2 medicine p3 l2) 060: (  unload a1 c2 place3 b3 l4)
022: (  load a1 c1 place2 b2 l2) 061: (    vent a1 b3 tools p5 l4)
023: (    move a1 c1 l2 depot) 062: (  load a1 c2 place3 b2 l4)
024: (unload a1 c1 place2 b2 depot) 063: (    move a1 c2 l4 depot)
025: (  fill a1 b2 medicine depot) 064: (unload a1 c2 place3 b2 depot)
026: (  load a1 c1 place1 b2 depot) 065: (    fill a1 b2 food depot)
027: (    move a1 c1 depot l3) 066: (  load a1 c2 place3 b2 depot)
028: (  unload a1 c1 place1 b2 l3) 067: (    move a1 c2 depot l5)
029: (  vent a1 b2 medicine p4 l3) 068: (  unload a1 c2 place3 b2 l5)
030: (  load a2 c2 place4 b3 l1) 069: (    vent a1 b2 food p6 l5)
031: (    move a2 c2 l1 depot) 070: (  load a2 c1 place2 b3 l4)
032: (unload a2 c2 place4 b3 depot) 071: (    move a2 c1 l4 depot)
033: (    fill a2 b3 food depot) 072: (unload a2 c1 place2 b3 depot)
034: (  load a2 c2 place4 b3 depot) 073: (  fill a2 b3 medicine depot)
035: (    move a2 c2 depot l3) 074: (  load a2 c1 place1 b3 depot)
036: (  unload a1 c2 place4 b3 l3) 075: (    move a2 c1 depot l5)
037: (    vent a1 b3 food p4 l3) 076: (  unload a1 c1 place1 b3 l5)
038: (  load a1 c2 place3 b2 l3) 077: (  vent a1 b3 medicine p6 l5)
```

Figura 10: Piano dell'istanza 3, parte 1

```

078: (      load a1 c2 place4 b3 15) 113: (      fill a1 b2 food depot)
079: (      move a2 c2 15 depot) 114: (   load a1 c1 place1 b2 depot)
080: (unload a2 c2 place4 b3 depot) 115: (      move a1 c1 depot 17)
081: (      fill a2 b3 tools depot) 116: (   unload a1 c1 place1 b2 17)
082: (   load a2 c2 place3 b3 depot) 117: (      vent a1 b2 food p8 17)
083: (      move a2 c2 depot 15) 118: (   load a1 c1 place2 b2 17)
084: (   unload a1 c2 place3 b3 15) 119: (      move a1 c1 17 depot)
085: (      vent a1 b3 tools p6 15) 120: (unload a1 c1 place2 b2 depot)
086: (   load a1 c2 place4 b3 15) 121: (   fill a1 b2 medicine depot)
087: (      move a2 c2 15 depot) 122: (   load a1 c1 place1 b2 depot)
088: (unload a2 c2 place4 b3 depot) 123: (      move a1 c1 depot 17)
089: (      fill a2 b3 food depot) 124: (   unload a1 c1 place1 b2 17)
090: (   load a2 c2 place3 b3 depot) 125: (   vent a1 b2 medicine p8 17)
091: (      move a2 c2 depot 16)
092: (   unload a2 c2 place3 b3 16)
093: (      vent a2 b3 food p7 16)
094: (   load a2 c2 place4 b3 16)
095: (      move a2 c2 16 depot)
096: (unload a2 c2 place4 b3 depot)
097: (   fill a2 b3 medicine depot)
098: (   load a2 c2 place4 b3 depot)
099: (      move a2 c2 depot 16)
100: (   unload a2 c2 place4 b3 16)
101: (   vent a2 b3 medicine p7 16)
102: (   load a2 c2 place4 b3 16)
103: (      move a2 c2 16 depot)
104: (unload a2 c2 place4 b3 depot)
105: (      fill a2 b3 tools depot)
106: (   load a2 c2 place4 b3 depot)
107: (      move a2 c2 depot 16)
108: (   unload a2 c2 place4 b3 16)
109: (      vent a2 b3 tools p7 16)
110: (   load a1 c1 place1 b2 15)
111: (      move a1 c1 15 depot)
112: (unload a1 c1 place1 b2 depot)

```

Figura 11: Piano dell'istanza 3, parte 2

INSTANCE 3					
HEURISTIC	WEIGHT	SPLIT	TIME (s)	N. ACTIONS	MEMORY (Mb)
MAX	5	2	OFM	---	---
MAX	10	2	OFM	---	---
MAX	5	8	2285 (>10 min)	116	3,46
MAX	10	8	1758 (>10 min)	122	3,46
MAX	5	16	88	126	3,46
MAX	10	16	84	126	3,46
SUM	5	2	14	126	3,46
SUM	10	2	14	126	3,46
SUM	5	8	10	126	3,46
SUM	10	8	13	126	3,46
SUM	5	16	6	126	3,46
SUM	10	16	6	126	3,46
OUR_OWN	5	2	>10 min	---	---
OUR_OWN	10	2	>10 min	---	---
OUR_OWN	5	8	>10 min	---	---
OUR_OWN	10	8	449	127	3,46
OUR_OWN	5	16	78	126	3,46
OUR_OWN	10	16	51	126	3,46

Figura 12: Prestazioni sull'istanza 3

3 Planning Temporale e implementazione del problema in ROS2

3.1 Planning Temporale

Viene richiesto di convertire il file di dominio realizzato precedentemente, associando poi una durata temporale alle differenti azioni, in particolare sono state utilizzate le *durative actions*.

3.1.1 Trasformazione del Dominio

E' importante sottolineare che a differenza del primo punto si utilizzerà ora PDDL 2.1. Ai fini di una migliore comprensione, saranno commentate solo le parti che hanno subito delle variazioni rispetto alla versione originaria già commentata nel paragrafo 1.

Sono stati aggiunti i riferimenti alle *durative-actions* e ai *fluents*:

```
(:requirements :strips :typing :durative-actions :fluents)
```

Figura 13: Requirements Dominio Temporale

I tipi precedentemente definiti sono stati resi tutti sotto-tipi di localizable, questo perchè PDDL2.1 presenta problemi nel momento in cui viene usato il tipo object.

Per quanto riguarda i predicati, si segnala l'aggiunta del predicato *freeagent*, che permette di definire se l'agente in questione è libero o meno, questo per evitare che si verifichi l'esecuzione di due azioni contemporaneamente da parte di uno stesso agente.

```
(:predicates
  (in ?o - localizable ?l - location)
  (need ?p - person ?c - content)
  (has ?p - person ?c - content)
  (hassomething ?p - person) ;gestione or
  (inbox ?b - box ?c - content)
  (emptybox ?b - box)
  (boxonplace ?b - box ?p - place)
  (emptyplace ?p - place) ;per gestire la load
  (fullplace ?p - place) ;per gestire l'unload
  (placeoncarrier ?p - place ?c - carrier)
  (freeagent ?g - agent) ;ci dice se l'agent g è libero oppure no
  (needall ?p - person) ;gestione or
  (needsomething ?p - person)
)
```

Figura 14: Predicati Dominio Temporale

Abbiamo quindi introdotto 6 funzioni, che ci aiuteranno meglio a definire la durata da assegnare alle durative-actions che introdurremo a breve:

```
(:functions
  (weightbox ?b - box)
  (weightcarrier ?c - carrier)
  (weightcontent ?c - content)
  (moveduration)
  (fillduration)
  (loadduration)
)
```

Figura 15: Funzioni

- La funzione **weightbox** è stata definita per modellare il peso del box, che viene incrementato e decrementato in base ai content che verranno caricati in esso.
- La funzione **weightcarrier** è stata definita per modellare il peso totale del carrello, tenendo conto delle box caricate su di esso.
- La funzione **weightcontent** è stata definita per modellare il peso dei differenti content, come richiesto dalla traccia del progetto.
- La funzione **moveduration** è stata definita per modellare un lower bound per quanto riguarda la durata delle azioni di move.

- La funzione **fillduration** è stata definita per modellare un lower bound per quanto riguarda la durata delle azioni di fill.
- La funzione **ventduration** è stata definita per modellare un lower bound per quanto riguarda la durata delle azioni di vent.

Prima di introdurre le durative-action si possono applicare delle considerazioni valide per tutte le azioni, ovvero, la durata dell'azione stessa sarà sempre il risultato del prodotto tra la durata minima dell'azione presa in considerazione, e il peso dell'oggetto che si sta spostando. Ad esempio per la move la durata sarà pari a: *moveduration* * *weightcarrier*. Inoltre affinché l'azione possa iniziare, l'agente deve risultare libero.

Si riportano dunque le durative-actions introdotte:

- **move**

```
(:durative-action move
  :parameters (?a - agent ?c - carrier ?from - location ?to - location)
  :duration (= ?duration (* (weightcarrier ?c) (moveduration)))
  :condition (and
    (at start (in ?a ?from))
    (at start (in ?c ?from))
    (at start (freeagent ?a))
  )
  :effect (and
    (at start (not(freeagent ?a)))
    (at start (not (in ?a ?from)))
    (at start (not (in ?c ?from)))
    (at end (in ?a ?to))
    (at end (in ?c ?to))
    (at end (freeagent ?a))
  )
);end move
```

- load

```
(:durative-action load
  :parameters (?a - agent ?c - carrier ?p - place ?b - box ?l - location)
  :duration (= ?duration (* (weightbox ?b) (loadduration)))
  :condition (and
    (at start (freeagent ?a))
    (over all (in ?a ?l))
    (over all (in ?c ?l))
    (over all (in ?b ?l))
    (over all (placeoncarrier ?p ?c))
    (at start (emptyplace ?p))
  )
  :effect (and
    (at start (not (freeagent ?a)))
    (at end (not (in ?b ?l)))
    (at start (not (emptyplace ?p)))
    (at start (fullplace ?p))
    (at end (boxonplace ?b ?p))
    (at end (increase (weightcarrier ?c) (weightbox ?b)))
    (at end (freeagent ?a))
  )
);load
```

- unload

```
(:durative-action unload
  :parameters (?a - agent ?c - carrier ?p - place ?b - box ?l - location)
  :duration (= ?duration (* (weightbox ?b) (loadduration)))
  :condition (and
    (at start (freeagent ?a))
```



```

        (over all(in ?a ?l))
        (over all(in ?c ?l))
        (at start(boxonplace ?b ?p))
        (over all(placeoncarrier ?p ?c))
        (at start(fullplace ?p))
    )
:effect (and
    (at start (not(freeagent ?a)))
    (at start(in ?b ?l))
    (at start(not (fullplace ?p)))
    (at start(emptyplace ?p))
    (at end(not (boxonplace ?b ?p)))
    (at end (decrease (weightcarrier ?c) (weightbox ?b)))
    (at end (freeagent ?a))
)
);unload

```

- vent

```

(:durative-action vent
:parameters (?a - agent ?b - box ?c - content ?p - person ?l - location)
:duration (= ?duration (* (weightcontent ?c) (fillduration)))
:condition (and
    (at start (freeagent ?a))
    (at start (inbox ?b ?c))
    (over all (in ?a ?l))
    (over all (in ?b ?l))
    (over all (in ?p ?l))
    (at start (need ?p ?c))
    (over all (needall ?p))
)

```

```

:effect (and
  (at start (not(freeagent ?a)))
  (at start (emptybox ?b))
  (at start (not (need ?p ?c)))
  (at end (not (inbox ?b ?c)))
  (at end (has ?p ?c))
  (at end (decrease (weighbox ?b) (weightcontent ?c)))
  (at end (freeagent ?a))
)
);vent

```

- fill

```

(:durative-action fill
  :parameters (?a - agent ?b - box ?c - content ?l - location)
  :duration (= ?duration (* (weightcontent ?c) (fillduration)))
  :condition (and
    (at start (freeagent ?a))
    (at start(emptybox ?b))
    (over all(in ?a ?l))
    (over all(in ?b ?l))
    (over all(in ?c ?l))
  )
  :effect (and
    (at start (not(freeagent ?a)))
    (at start (not(emptybox ?b)))
    (at end (inbox ?b ?c))
    (at end (increase (weighbox ?b) (weightcontent ?c)))
    (at end (freeagent ?a))
  )
);fill

```

3.1.2 Definizione del file di Problema

A questo punto si è proceduto con la modifica del file di problema definito per la risoluzione dell'istanza 2 descritta precedentemente nel paragrafo 2. Le modifiche apportate al file corrispondono all'aggiunta delle funzioni definite precedentemente, ovvero: *weighbox*, *weighcarrier*, *weighcontent*, *moveduration*, *loadduration* e *fillduration*. Le modifiche apportate sono le seguenti:

```
...
(:init
  (= (weighbox b1) 1)
  (= (weighbox b2) 1)
  (= (weighbox b3) 1)
  (= (weighbox b4) 1)
  (= (weighbox b5) 1)
  (= (weighcarrier c1) 6)
  (= (weighcontent medicine) 1)
  (= (weighcontent food) 2)
  (= (weighcontent tools) 3)
  (= (moveduration) 5)
  (= (loadduration) 3)
  (= (fillduration) 2)
...

```

Come da richiesta sono stati assegnati i pesi 1-2-3 rispettivamente a medicine-food-tools. Abbiamo poi deciso di assegnare un peso pari a 1 alle box vuote e pari a 6 al carrello vuoto. Fissando inoltre la durata delle varie azioni assegnando alla move 5, 3 alla load e all'unload e 2 alla fill.

3.2 Robotics Planning

A questo punto, attraverso il planner LPG-TD, è stato generato il piano presente nella 16. Dopo di che attraverso uno script python è stata trasformata la sintassi del piano così che risultasse compatibile con POPF, che è il planner pre-implementato all'interno di plansys2.

```
0.0003:  (fill a1 b5 medicine depot) [2.0000]
2.0005:  (load a1 c1 place4 b5 depot) [6.0000]
8.0007:  (fill a1 b2 food depot) [4.0000]
12.0010: (fill a1 b4 food depot) [4.0000]
16.0012: (load a1 c1 place3 b4 depot) [9.0000]
25.0015: (fill a1 b1 food depot) [4.0000]
29.0018: (load a1 c1 place1 b2 depot) [9.0000]
38.0020: (fill a1 b3 medicine depot) [2.0000]
40.0023: (load a1 c1 place2 b3 depot) [6.0000]
46.0025: (move a1 c1 depot l1) [80.0000]
126.0027: (unload a1 c1 place1 b2 l1) [9.0000]
135.0030: (move a1 c1 l1 l2) [65.0000]
200.0033: (unload a1 c1 place3 b4 l2) [9.0000]
209.0035: (vent a1 b4 food p3 l2) [4.0000]
213.0038: (move a1 c1 l2 l1) [50.0000]
263.0040: (vent a1 b2 food p1 l1) [4.0000]
267.0042: (unload a1 c1 place4 b5 l1) [6.0000]
273.0045: (vent a1 b5 medicine p2 l1) [2.0000]
275.0048: (unload a1 c1 place2 b3 l1) [6.0000]
281.0050: (vent a1 b3 medicine p1 l1) [2.0000]
```

Figura 16: Piano generato con LPG-TD

A questo punto per eseguire il piano mediante l'utilizzo di plansys2, si è reso necessario creare un workspace (*progettoRobot*) all'interno del workspace di plansys2. *progettoRobot* contiene i diversi file che risultano necessari per poter utilizzare in modo corretto plansys2. In particolare:

- Sono state implementate per tutte le azioni, le *fakeaction* corrispondenti in linguaggio C++, un esempio di fakeaction si può vedere nella figura 17.
- E' stato definito il file di launch python, che è necessario per avviare i nodi ROS per l'esecuzione del nostro piano

```

using namespace std::chrono_literals;

class VentAction : public plansys2::ActionExecutorClient
{
public:
    VentAction()
    : plansys2::ActionExecutorClient("vent", 200ms)
    {
        progress_ = 0.0;
    }

private:
    void do_work()
    {
        if (progress_ < 1.0) {
            progress_ += 0.02;
            send_feedback(progress_, "Vent running");
        } else {
            finish(true, 1.0, "Vent completed");

            progress_ = 0.0;
            std::cout << std::endl;
        }

        std::cout << "\r\e[K" << std::flush;
        std::cout << "Venting ... [" << std::min(100.0, progress_ * 100.0) << "%] " <<
            std::flush;
    }

    float progress_;
};

int main(int argc, char ** argv)
{
    rclcpp::init(argc, argv);
    auto node = std::make_shared<VentAction>();

    node->set_parameter(rclcpp::Parameter("action_name", "vent"));
    node->trigger_transition(lifecycle_msgs::msg::Transition::TRANSITION_CONFIGURE);

    rclcpp::spin(node->get_node_base_interface());

    rclcpp::shutdown();

    return 0;
}

```

Figura 17: Vent FakeAction

- Sono stati inoltre inseriti il file di dominio e i commands da fornire al terminale plansys2 così da settare dominio e istanze del problema.
- Sono inoltre presenti come da consuetudine il file CMakeList.txt e package.xml, utili per effettuare la build della cartella e per settare le dipendenze necessarie.

Dopo aver definito tutti i file necessari all'esecuzione, è opportuno eseguire diversi comandi all'interno del terminale. All'interno del workspace *progettoRobot* è necessario eseguire i seguenti comandi:

```

colcon build --symlink-install
source install/setup.bash
ros2 launch crazy_robot crazy_robot_launch.py

```

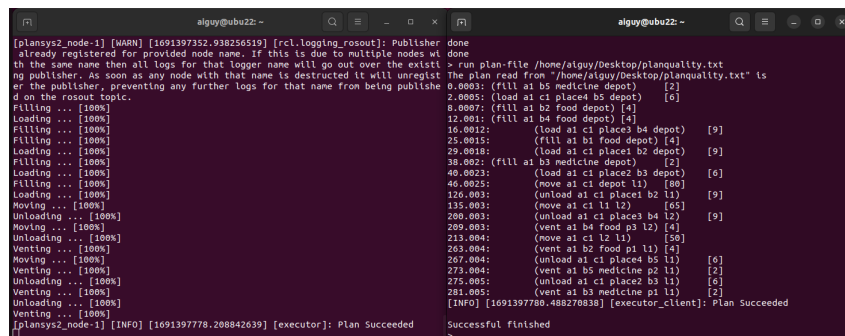
Dopo di che, in un altro terminale, bisogna lanciare il comando:

```
ros2 run plansys2_terminal plansys2_terminal
```

All'interno del terminale n°2, bisogna caricare il file dei commands per settare le istanze del problema, procedendo poi all'esecuzione del file di plan, generato precedentemente

```
source /plansys2_ws/src/ros2_planning_system_examples
      /progettoRobot/src/crazy_robot/launch/commands.txt
run plan-file /plansys2_ws/src/ros2_planning_system_examples
              /progettoRobot/src/crazy_robot/launch/planquality.txt
```

L'output ottenuto nei due terminali è presente nella Figura 18, dalla quale è possibile apprezzare come il piano viene correttamente eseguito da plansys2.



```
[plansys2_node-1] [WARN] [1691397352.938256519] [rcl_logging_rosout]: Publisher
already registered for provided node name. If this is due to multiple nodes wi
th the same name then all logs for that logger name will go out over the existi
ng publisher. As soon as any node with that name is destructed it will unregist
er the publisher, preventing any further logs for that name from being publishe
d on the rosout topic.
Filling ... [100%]
Loading ... [100%]
Filling ... [100%]
Loading ... [100%]
Filling ... [100%]
Loading ... [100%]
Filling ... [100%]
Loading ... [100%]
Filling ... [100%]
Loading ... [100%]
Moving ... [100%]
Unloading ... [100%]
Moving ... [100%]
Unloading ... [100%]
Venting ... [100%]
Moving ... [100%]
Unloading ... [100%]
Venting ... [100%]
Unloading ... [100%]
Venting ... [100%]
Unloading ... [100%]
Venting ... [100%]
[plansys2_node-1] [INFO] [1691397778.208842639] [executor]: Plan Succeeded
Successful finished
>
[plansys2_node-1] [WARN] [1691397352.938256519] [rcl_logging_rosout]: Publisher
already registered for provided node name. If this is due to multiple nodes wi
th the same name then all logs for that logger name will go out over the existi
ng publisher. As soon as any node with that name is destructed it will unregist
er the publisher, preventing any further logs for that name from being publishe
d on the rosout topic.
done
run plan-file /home/alguy/Desktop/planquality.txt
The plan read from "/home/alguy/Desktop/planquality.txt" is
2.0003: (fill a1 b5 medicine depot) [2]
2.0005: (load a1 c1 place4 b5 depot) [6]
8.0007: (fill a1 b2 food depot) [4]
12.0011: (fill a1 b4 food depot) [4]
16.0012: (load a1 c1 place3 b4 depot) [9]
25.0015: (fill a1 b1 food depot) [4]
29.0018: (load a1 c1 place1 b2 depot) [9]
38.0021: (fill a1 b3 medicine depot) [2]
40.0023: (load a1 c1 place2 b3 depot) [6]
46.0025: (move a1 c1 depot l1) [80]
126.003: (unload a1 c1 place1 b2 l1) [9]
135.003: (move a1 c1 l1 l2) [65]
209.003: (unload a1 c1 place3 b4 l2) [9]
209.003: (vent a1 b4 food p3 l2) [4]
213.004: (move a1 c1 l2 l1) [50]
263.004: (vent a1 b2 food p1 l1) [4]
267.004: (unload a1 c1 place4 b5 l1) [6]
273.004: (vent a1 b5 medicine p2 l1) [2]
275.005: (unload a1 c1 place2 b3 l1) [6]
281.005: (vent a1 b3 medicine p1 l1) [2]
[INFO] [1691397780.488270838] [executor_client]: Plan Succeeded
Successful finished
>
```

Figura 18: Output ottenuto dall'esecuzione del piano