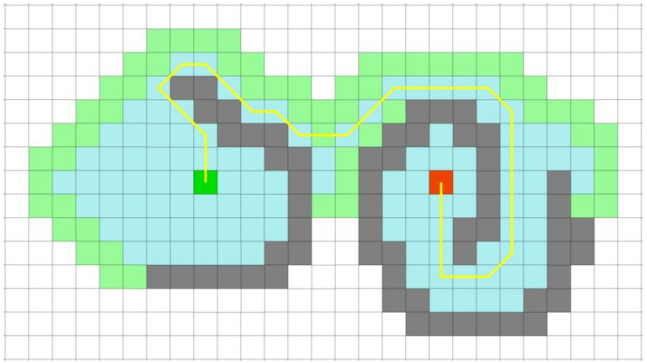
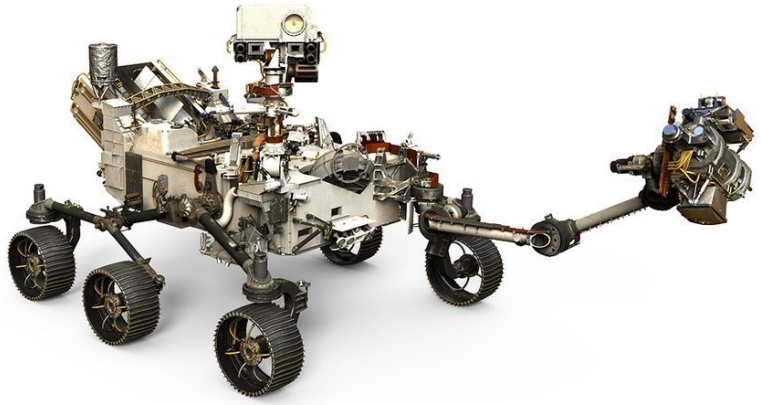
ROSPlan

Jean Massardi - 9 novembre 2017

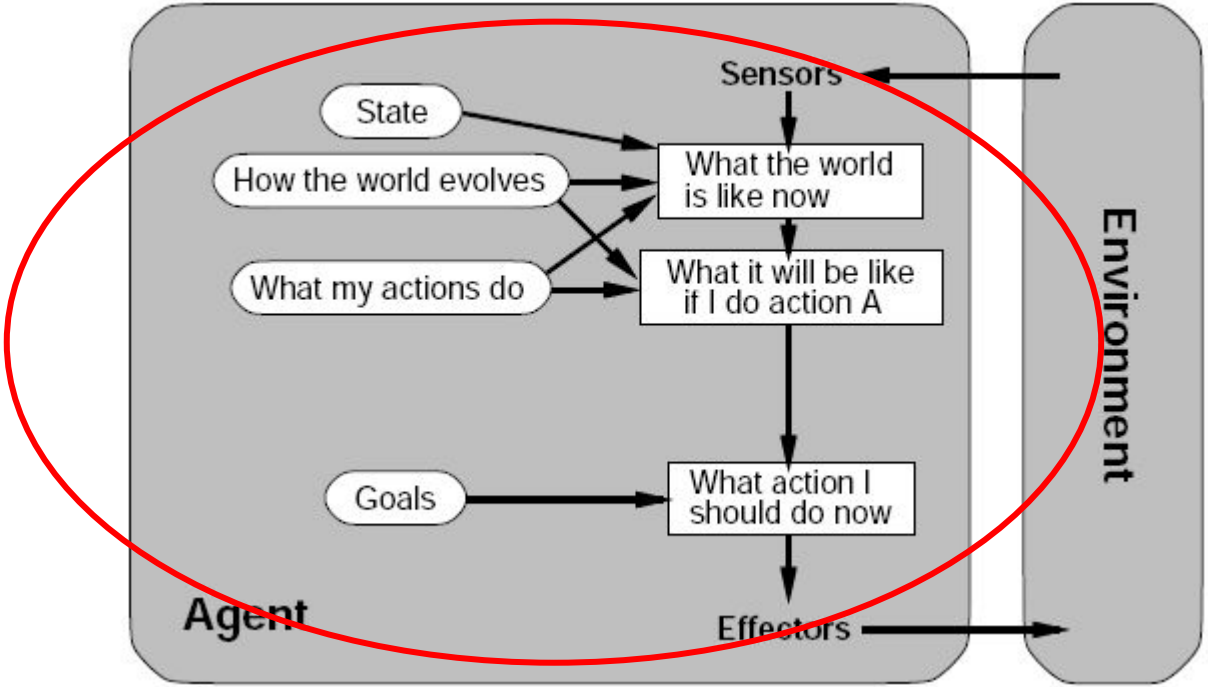
Introduction

Comment passer de l’algorithme à un agent intelligent ?





Introduction



Planification

''La planification en Intelligence artificielle consiste à sélectionner et à ordonnancer des actions permettant d'atteindre un but donné à partir d'une base de connaissance sur les actions possibles. Cette dernière contient des préconditions et des effets.''

Eric Beaudry

Variables :
Robots (r1)
Colis (c1,c2)
Emplacement(e1,e2,e3,)

État Initial :
loc(r1) = e1
loc(c1) = e2
loc(c2) = e3

Actions :
Deplacer(r,e)
-pre()
-post(loc(r) = e)

Attraper(r,c)
- pre(loc(r) = loc(c))
- post(loc(c) = r)

Relacher(r,c)
-pre()
-post(loc(c) = loc(r))

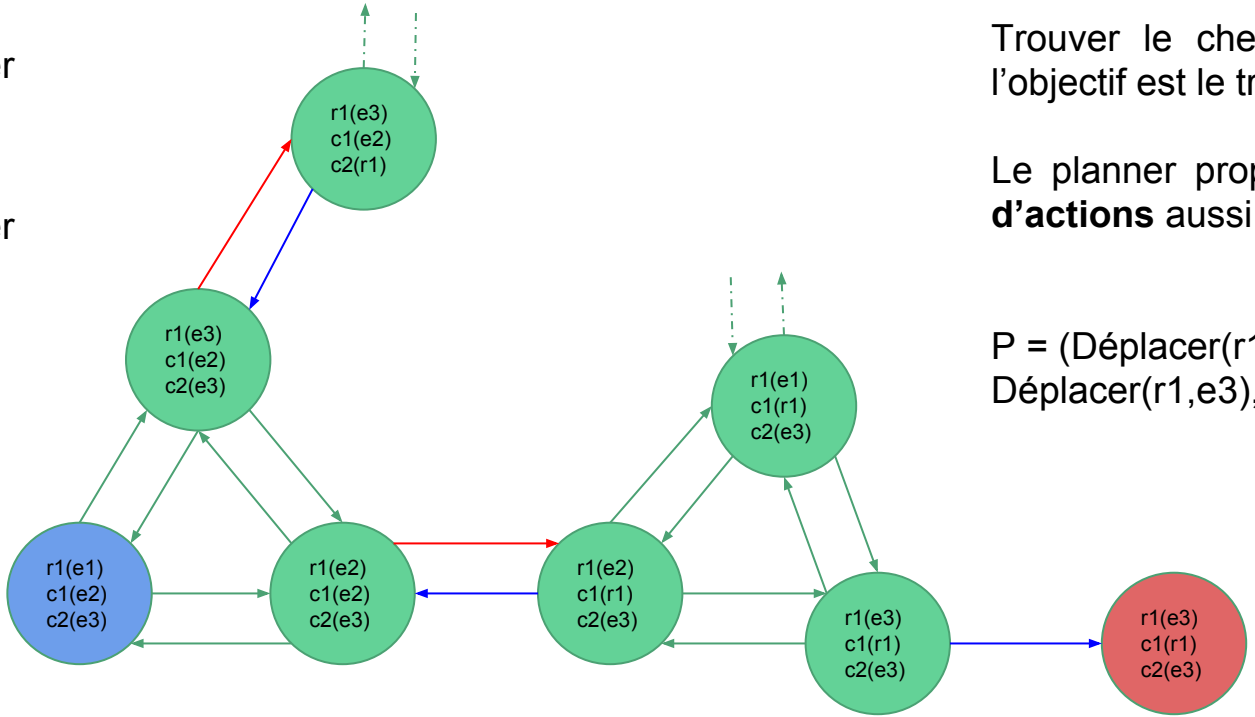
Objectif :
loc(c1) = e3

Planification

Déplacer

Attraper

Relacher

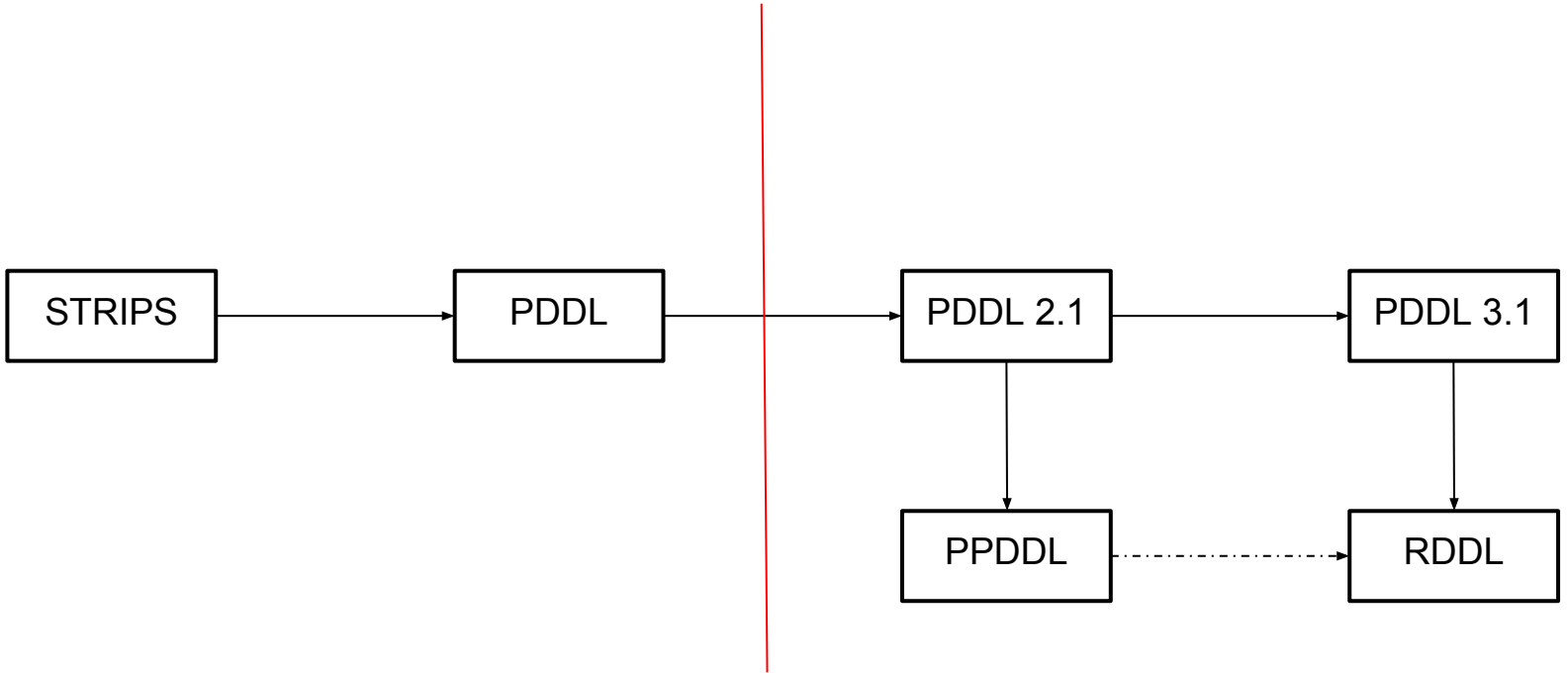


Trouver le chemin entre l'état initial et l'objectif est le travail du **planner**.

Le planner propose en sortie une **suite d'actions** aussi appelé un **plan**.

P = (Déplacer(r1,e2), Attraper(r1,c1)), Déplacer(r1,e3), Relâcher(r1,c1))

Langage de définition



PDDL

Domaine

- Les actions
- Les prédicats

Problème

- L’état initial
- Les objectifs
- Les objets

PDDL - exemple

```
(define (domain gripper-strips)
  (:predicates (room ?r)  (ball ?b) (gripper ?g) (at-robby ?r)
               (at ?b ?r) (free ?g) (carry ?o ?g))
  (:action move
   :parameters (?from ?to)
   :precondition (and  (room ?from) (room ?to) (at-robby ?from))
   :effect (and (at-robby ?to) (not (at-robby ?from))))
  (:action pick
   :parameters (?obj ?room ?gripper)
   :precondition (and (ball ?obj) (room ?room) (gripper ?gripper)
                      (at ?obj ?room) (at-robby ?room) (free ?gripper))
   :effect (and (carry ?obj ?gripper) (not (at ?obj ?room))
                (not (free ?gripper))))
  (:action drop
   :parameters (?obj ?room ?gripper)
   :precondition (and (ball ?obj) (room ?room) (gripper ?gripper)
                      (carry ?obj ?gripper) (at-robby ?room))
   :effect (and (at ?obj ?room) (free ?gripper)
                (not (carry ?obj ?gripper)))))
```

PDDL - exemple

```
(define (problem strips-gripper2)
  (:domain gripper-strips)
  (:objects rooma roomb ball1 ball2 left right)
  (:init (room rooma)
         (room roomb)
         (ball ball1)
         (ball ball2)
         (gripper left)
         (gripper right)
         (at-robby rooma)
         (free left)
         (free right)
         (at ball1 rooma)
         (at ball2 rooma))
  (:goal (at ball1 roomb)))
```

Exécution

L’exécution d’un plan peut entraîner un certain nombres de problèmes :

- Incertitude entre la description d’une action au niveau plan et la description d’une action au niveau contrôleurs (exemple : Déplacer(coor)).
- Que faire si le plan n’est plus valable (et comment le détecter)
- Que faire en cas d’incertitude de résultats sur certaines actions (i.e si un modèle n’est plus déterministe en application)

Problématique

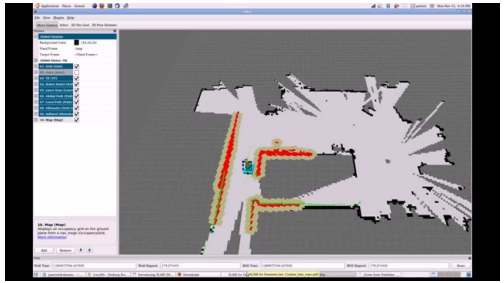
Il n’y a pas de framework standard pour l’intégration des planners sur des robots.

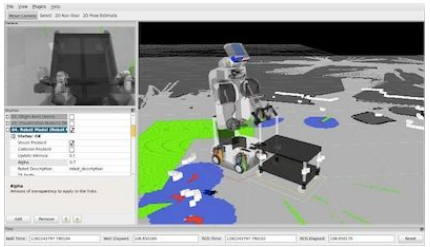
Chaque projet à sa propre intégration, ce qui rend les solutions développés peu réutilisable

ROS

ROS : **R**obot **O**perating **S**ystem

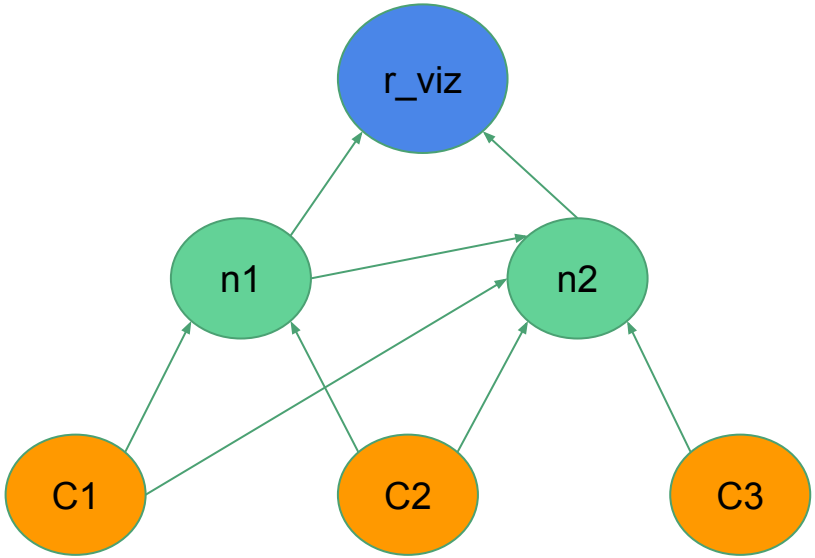
Collections d’outils, de frameworks et de drivers pour pouvoir développer et contrôler des robots.





ROS

Se base sur un système de **nodes**. Les nodes sont un **ensemble hiérarchiques** de **composants** communiquant les uns avec les autres en **temps réel**. On trouve à la **base** les **capteurs** et en **haut** les **interfaces homme-machine**.

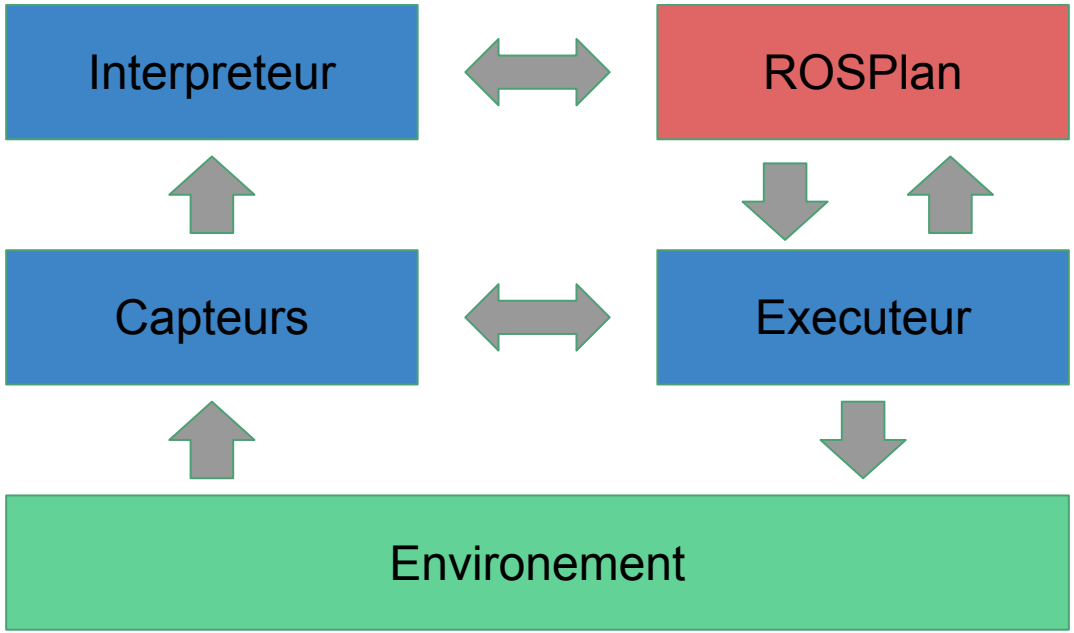


ROSPlan

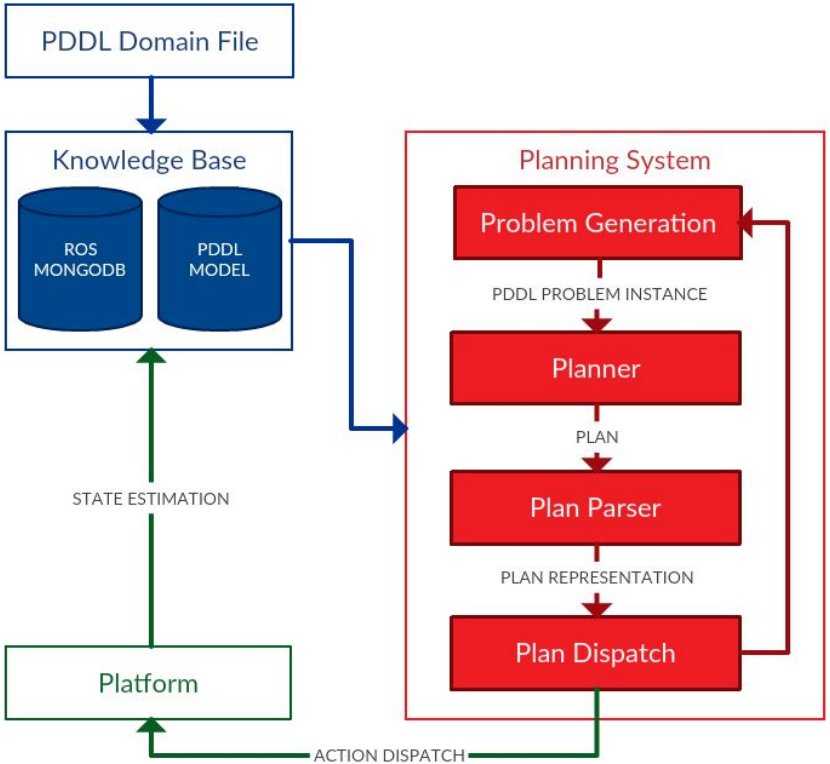
Répond à cette problématique en donnant un framework capable de :

1. Intégrer une base de connaissances
2. Intégrer un planner
3. Générer un état initial à partir d’un environnement
4. Transformer le plan proposé par le planner en actions de bas niveaux, interprétable par les controllers
5. Supporter les changements d'environnements et les échecs des actions

ROSPlan



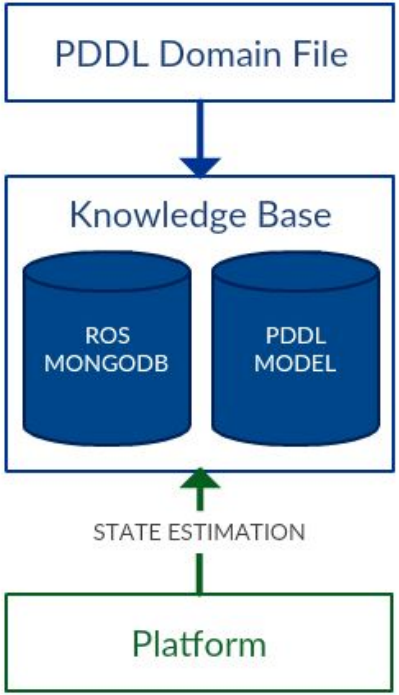
ROSPlan


PDDL Domain File
Knowledge Base
ROS MONGODB
PDDL MODEL
Planning System
Problem Generation
PDDL PROBLEM INSTANCE
Planner
PLAN
Plan Parser
PLAN REPRESENTATION
Plan Dispatch
STATE ESTIMATION
Platform
ACTION DISPATCH

Knowledge base

Le **Knowledge base node** a pour mission:

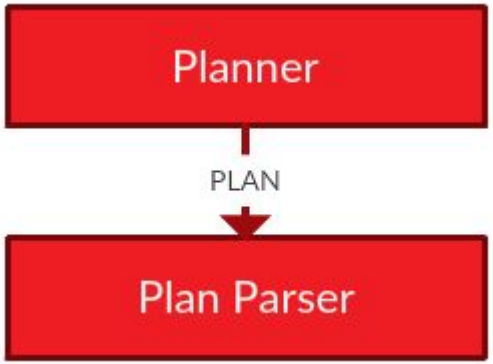
- De servir de **base de données**
- De servir **mémoire des états** en cours et passés
- Contenir les **domaines** sous forme PDDL
- Faire le lien entre les **informations** fournies par les capteurs/contrôleurs avec les **données** (**Ontologie**)
- Si il y’a un changement dans les variables, **peut ordonner une nouvelle planification** au planner



Planner

Dans la version proposée, il s’agit de **POPF**.

POPF est un planner utilisant des **plans partiels** et du **chaînage avant** (forward chaining) dans son raisonnement.



AVANTAGES : connu, déjà utilisé en robotique, au palmarès du concours de planners ICAPS 2011

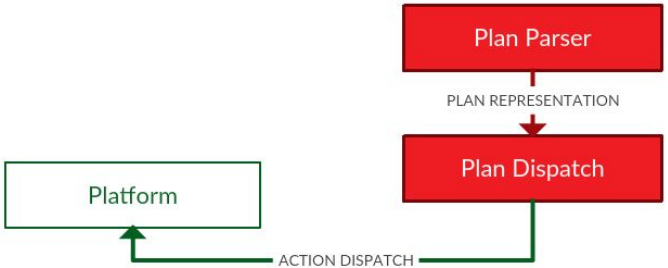
INCONVÉNIENTS : Supporte une version partielle de PDDL 2.1 (ne supporte pas les conditions négatives)

Plan dispatch

Transforme le plan en suite d'actions **interprétables** par les contrôleurs

Vérifie que les actions (niveau contrôleurs) n'ont **pas échouées**. Si elles ont échoué, il peut prendre **trois décisions**:

1. **continuer le plan**
2. **recommencer** l'action
3. **planifier à nouveau**



Conclusion

- Répond à la problématique posée
- Expérimentation peu intéressante
- La version de PDDL pour l’intégration est une version incomplète. Quid de l’utilisation de PPDDL ou RDDL ?
- Le framework est-il compatible avec un planner de type MDP ou PO-MDP ?