

Multi-agent systems Project 3

STRIPS planner

SUPERVISED BY:
DR. ZARGAYOUNA MAHDI

ELABORATED BY:
Arfaoui Fairouz
Marnissi Skander

M2 SIA
2020-2021

Table of content:

The AI's World

The domain

The problem

Implementation

Simulateur en ligne

Execution on Node.js

Solutions

Depth First Search

Breadth First Search

A* Search

References

Introduction

Finding actions that lead from an initial state to a goal state, can be considered an old challenge of artificial intelligence. STRIPS is a planning methodology for solving this challenge, where planning is performed under certain assumptions and where plan generation is separated from its execution. It can be used by intelligent agents and autonomous robots. It involves intelligently executing a sequence of actions in order to achieve a goal.

The Stanford Research Institute Problem Solver (STRIPS) is an automated planning technique that works by executing a domain and problem to find a goal. With STRIPS, we first describe the world by providing objects, actions, preconditions, and effects.

A common language for writing STRIPS domain and problem sets is the Planning Domain Definition Language (PDDL).

PDDL is intended to describe a domain, that is, what predicates there are, what actions are possible, what the structure of compound actions is, and what the effects of actions are. It's a relatively easy approach to writing simple AI planning problems.

The AI's World

The domain

We have been working on implementing a solution to alleviate the difficulties of planning a college day's schedule.

The student usually has to go through multiple steps in order to get to his university. Due to the current circumstances, additional mandatory rules and procedures have now been introduced which impedes the process.

We have then accordingly, predetermined all the possible actions that a student may perform to attend a course in a timely manner.

Action	Parameters	Precondition	Effect
walk	student location1 location2	student has a mask student at location1 location1~location2 < 3km	student at location1 student not at location2
open	student gadget location	student at location gadget at location gadget not open	gadget is open
collect-mask	student gadget1 gadget2 location	student at location gadget1 at location gadget2 in gadget1 gadget1 is open	student has mask
collect-navigo	student location gadget	student at location gadget at location	student has navigo
generate-authorization	student location	student at location	student has authorization
take-train	student location1 location2 train	student at location1 student has authorization location1~location2 > 20km train at location1	student not at location1 student at location2 train not at location1
take-course	student1 student2 location	student1 at location student1 has mask student2 at location	student took the course
drive-car*	student location1 location2 car	student at location1 student has authorization location1~location2 > 20km car at location1	student not at location1 student at location2 car not at location1

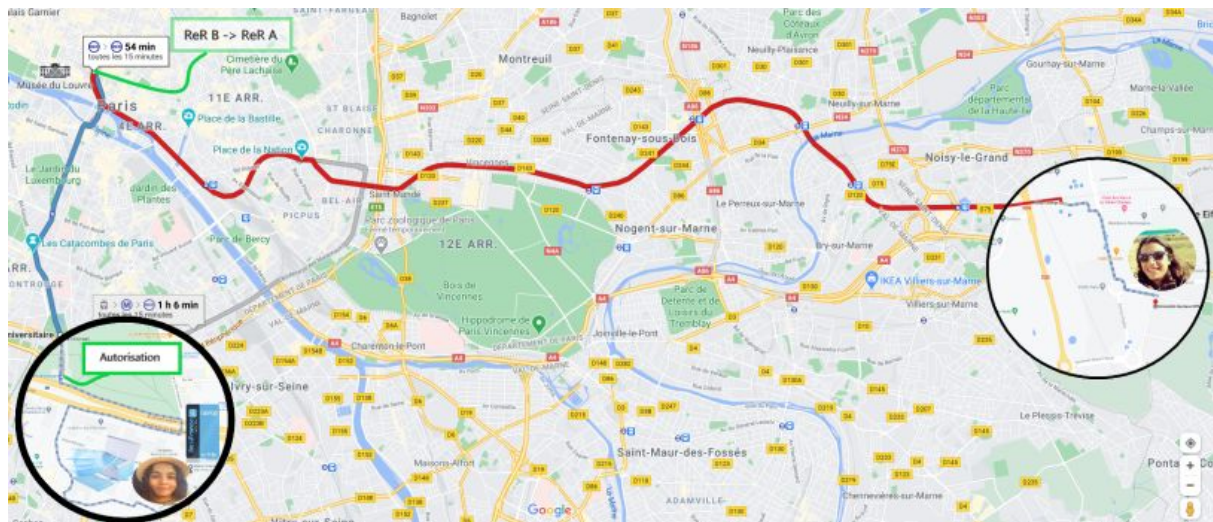
drive-car** action will only be utilised in the **A search section of the solutions.

All these actions are written in Strips PDDL format in a domain.txt file, where the types of the necessary objects are specified.

The types we are going to use in our project are as follows: Student - Location - Gadget - Train - Car.

The problem

Once the domain is described, we have to create a problem file which details an initial and final state. STRIPS will then search all the possible states, starting from the initial one while executing various actions, until it reaches a goal.



The problem.txt file defines the objects we are going to use:

- 6 locations : home citeU chateletB chateletA noisyChamps gustaveEiffel + (parkingCiteU parkingGustaveEiffel)**
- 2 students : Fairouz and Skander
- 3 gadgets : Mask, Box and Navigo
- 2 trains : RER A and RER B
- 1 car : Ford **

An initial state:

- The agent is at home
- All the gadgets are at home too (and a mask inside a box)
- Distances between locations (so that the agent will know if he can walk or take a train)
- The positions of the vehicles
- A nice friend is waiting for the “agent” in a certain position.

A goal:

- Attends a course.

****The two parkings and the car will only be used in the Solutions::A* Search section.**

Implementation

Simulateur en ligne

The implementation of a STRIPS project can be done online thanks to a simulator powered by the strips node.js library: STRIPS-Fiddle which is created by a software developer named Kory Becker.

With STRIPS-Fiddle, the program is executed locally in the web browser and the output is displayed on the page.

Here's an example of a program that aims to reach a certain location:

STRIPS-Fiddle

Home

About

Contact

Run

BFS

DFS

Save

Sign in / Join

Domain

Select an existing Domain

<Create your own>

Name

Exemple

Code

```
(:action take-train
:parameters (?sid - student ?l1 - location ?l2 - location ?v - vehicle)
:precondition (and (at ?sid ?l1) (at ?v ?l1) (288M ?l1 ?l2) (has-mask ?sid) (has-navigate ?sid) (has-authorization ?sid))
:effect (and (at ?sid ?l2) (not (at ?sid ?l1)) (not (at ?v ?l1))))

(:action take-course
:parameters (?sid1 - student ?sid2 - student ?l - location)
:precondition (and (at ?sid1 ?l) (at ?sid2 ?l) (has-mask ?sid1))
:effect (and (course-taken ?sid1) (course-taken ?sid2)))
```

Problem

Select an existing Problem

<Create your own>

Name

Exemple

Code

```
(define (problem siteU-gustaveEiffel)
(:domain trip)

(:objects
  Fairouz Skander - student
  home gustaveEiffel siteU chateletA chateletA noisyChamps - location
  mask box navigate authorization- gadget
  BEBA BEBA - vehicle
)

(:init
```

STRIPS-Fiddle

Home

About

Contact

Run

BFS

DFS

Save

Sign in / Join

Output

Initializing, this may take a couple of seconds or minutes, depending upon the domain. Please wait ..

Using breadth-first-search.

Depth: 0, 6 child states.

Depth: 1, 10 child states.

Depth: 10, 1966 child states.
Depth: 10, 1966 child states.
Depth: 10, 1966 child states.

Solution found in 10 steps!

1. open Fairouz box home
2. collect-mask Fairouz home box mask
3. collect-navigo Fairouz home navigo
4. walk Fairouz home citeU
5. generate-authorization Fairouz citeU
6. take-train Fairouz citeU chateletB RERB
7. walk Fairouz chateletB chateletA
8. take-train Fairouz chateletA noisyChamps RERA
9. walk Fairouz noisyChamps gustaveEiffel
10. take-course Fairouz Skander gustaveEiffel

[Back to Top](#)

STRIPS-Fiddle makes the planning easier since we don't need to use other tools.

Execution on Node.js

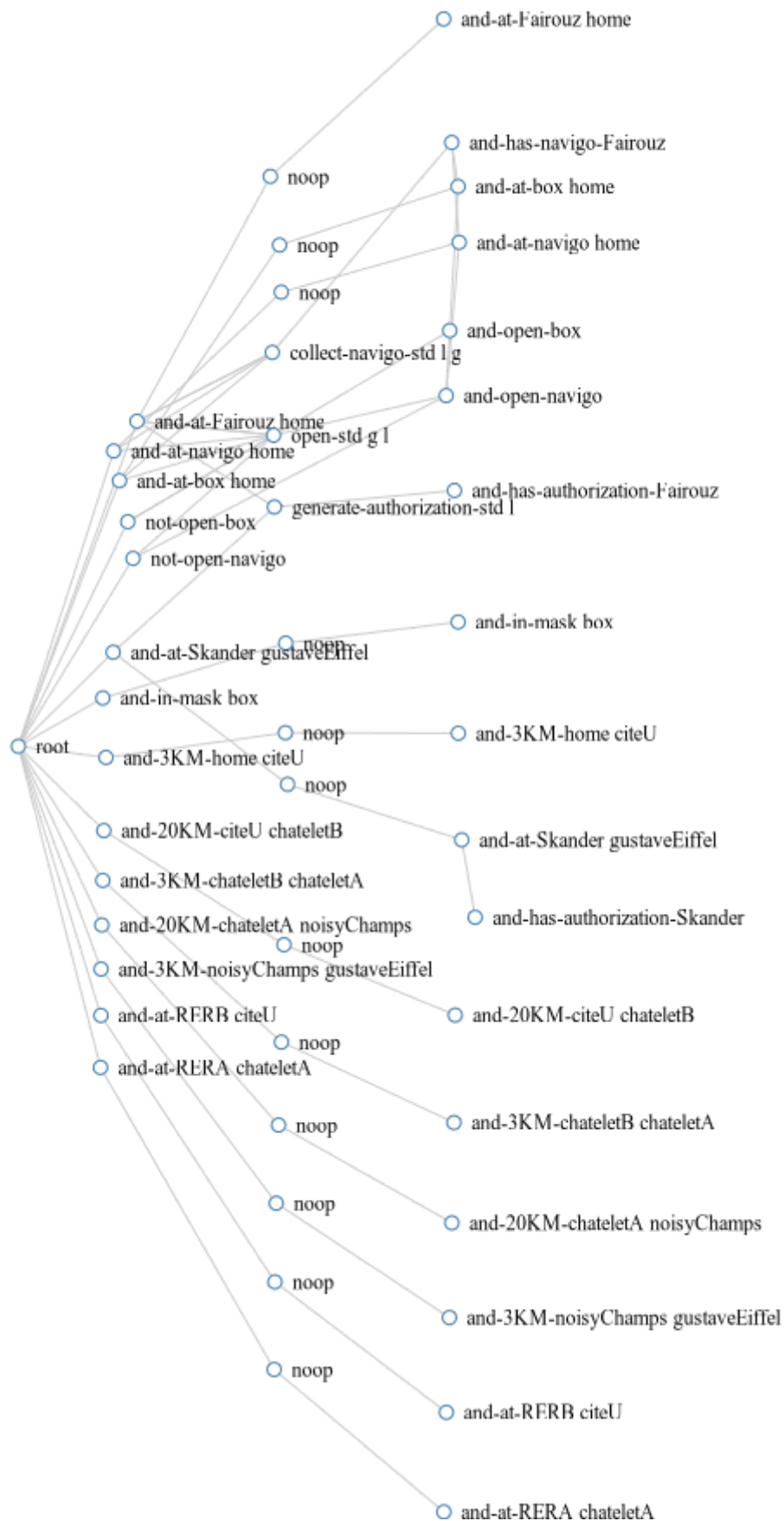
To locally run the program, we used the node.js strips library, which supports breadth-first, depth-first, and A* searching.

Domains and problems (in PDDL format) can be loaded from the plain text files into the node.js strips library.

The default settings use depth-first-search and return the first solution found. Although we can still change the used method and consequently the number of solutions.

Also with STRIPS AI planning, a graph can be generated, containing all the available states and actions that bring you to every state. In order to present a readable graph, we have decided to only display the first 3 layers.

For every action, 'precondition' represents parent literals. "effect" represents child literals. Any action unnamed "noop" (no-operation) represents an applicable action, based on the preceding literals. "noop" is simply a placeholder action which carries each literal forward, from one layer to the next.



Solutions

STRIPS provides us with a slick aspect which is the possible application of different calculations to reach different states by exploring several arrangements. Regular strategies incorporate 'Breadth First Search', Depth First Search', and A*.

Depth First Search

With 'Depth First Search', the initial state is evaluated and all the child states are pushed onto a stack or queue. The AI at that point drops down the tree to the following child until either an objective is found, or no more child states exist. At the point when it reaches a dead-end, it backtracks until another accessible child state is found.

Although 'Depth First Search' might not find the optimal solution to a STRIPS artificial intelligence planning problem, it can be faster than breadth-first-search in some cases.

```
fairouz@fairouz: ~/Bureau/MAS-DayPlanner
Fichier Édition Affichage Rechercher Terminal Aide
fairouz@fairouz:~/Bureau/MAS-DayPlanner$ time node depth-first-search.js
Using depth-first-search.

Depth: 0, 6 child states.
Depth: 1, 5 child states.
Depth: 2, 5 child states.
Depth: 3, 6 child states.
Depth: 4, 2 child states.
Depth: 5, 2 child states.
Depth: 6, 2 child states.
Depth: 5, 2 child states.
Depth: 6, 2 child states.
Depth: 4, 6 child states.
Depth: 5, 2 child states.
Depth: 6, 2 child states.
Depth: 7, 3 child states.
Depth: 8, 3 child states.
Depth: 9, 3 child states.
Depth: 10, 3 child states.
Depth: 11, 3 child states.
- Solution found in 12 steps!
1. open Fairouz navigo home
2. open Fairouz box home
3. collect-mask Fairouz home box mask
4. collect-navigo Fairouz home navigo
5. walk Fairouz home citeU
6. generate-authorization Skander gustaveEiffel
7. generate-authorization Fairouz citeU
8. take-train Fairouz citeU chateletB RERB
9. walk Fairouz chateletB chateletA
10. take-train Fairouz chateletA noisyChamps RERA
11. walk Fairouz noisyChamps gustaveEiffel
12. take-course Fairouz Skander gustaveEiffel

real    0m0.347s
user    0m0.389s
sys      0m0.032s
fairouz@fairouz:~/Bureau/MAS-DayPlanner$
```

Although the search for the solution is quite fast (0.347 seconds). We can see on the output of the terminal that Depth First Search does not give a coherent result.

We can already see that at the first step the agent opened "Home" to take the "Navigo", which is not acceptable for us who developed the domain in which we work.

On the other hand we can see at step 7 that an authorization has been generated for Skander. This may seem right at the beginning but it is not the case. Indeed, our agent here is Fairouz, she is the one who "plays" all the actions and therefore needs an authorization. On the other hand Skander can be another agent in another application and thus it is not on this algorithm that we will see its actions.

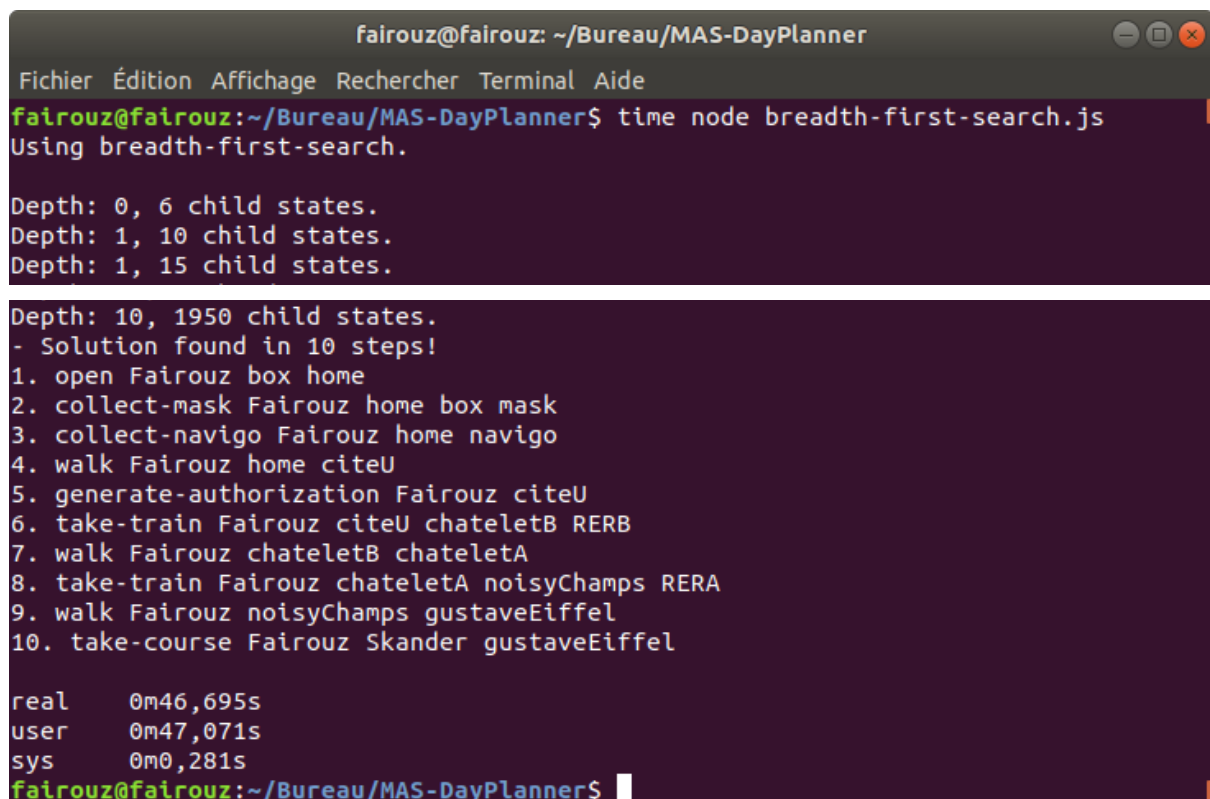
⇒ With Depth First Search, our problem was not solved properly.

Breadth First Search

'Breadth first search' finds the most optimal solution to a STRIPS problem. It searches from the initial state, and evaluates all child states that are available. It only evaluates at the current depth, completely checking all states before moving to the next depth level.

In this manner, if any of the child states results in the goal state, it can stop searching right there and return the solution. This solution will always be the shortest. However, because breadth first search scans every single child state at the current depth level, it could take a long time to search, especially if the tree is very wide.

In our case we have reached depth 10 with 1950 child states, and indeed it took much time than 'Depth First Search'



```
fairouz@fairouz: ~/Bureau/MAS-DayPlanner
Fichier Édition Affichage Rechercher Terminal Aide
fairouz@fairouz:~/Bureau/MAS-DayPlanner$ time node breadth-first-search.js
Using breadth-first-search.

Depth: 0, 6 child states.
Depth: 1, 10 child states.
Depth: 1, 15 child states.

Depth: 10, 1950 child states.
- Solution found in 10 steps!
1. open Fairouz box home
2. collect-mask Fairouz home box mask
3. collect-navigo Fairouz home navigo
4. walk Fairouz home citeU
5. generate-authorization Fairouz citeU
6. take-train Fairouz citeU chateletB RERB
7. walk Fairouz chateletB chateletA
8. take-train Fairouz chateletA noisyChamps RERA
9. walk Fairouz noisyChamps gustaveEiffel
10. take-course Fairouz Skander gustaveEiffel

real    0m46,695s
user    0m47,071s
sys     0m0,281s
fairouz@fairouz:~/Bureau/MAS-DayPlanner$
```

Let's set our attention towards step 5, indeed, authorization generation can be done at any step before "take-train", but Breadth First Search knew exactly where to put it, and that's exactly before the agent needs it.

⇒ Aside from the fact that it takes a lot of time to run (47 seconds), Breadth First Search answers perfectly to our problem.

A* Search

The most intelligent of the searching techniques for solving a STRIPS PDDL artificial intelligence AI planning problem is to use A search. A is a heuristic search. This means it uses a formula or calculation to determine a cost for a particular state. A state that has a cost of 0 is our goal state. A state that has a very large value for cost would be very far from our goal state.

Before selecting the next state, A assigns a cost to each one, based upon certain characteristics of the state. It then chooses the next lowest-cost state to move to, with the idea being that the lowest costing states are the ones most likely to result in the goal.

To see the utility of this method, we chose to create another road leading to the **Gustave-Eiffel** University by car, and we added a drive car action:

```
(:action drive-car
  :parameters (?std - student ?l1 - location ?l2 - location ?c - car)
  :precondition (and (at ?std ?l1) (at ?c ?l1) (20KM ?l1 ?l2) (has-authorization ?std))
  :effect (and (at ?std ?l2) (not (at ?std ?l1)) (not (at ?c ?l1))))
)
```

home ⇒ **parkingCiteU** ⇒ **parkingGustaveEiffel** ⇒ **gustaveEiffel**.

Obviously this way is shorter and less complicated, so if we try to solve our problem with Breadth First Search, we will have the solution in 7 steps, indicating that we have to take the car.

On the other hand, in real life, it's always better to take the train (pollution, traffic, **Fairouz** doesn't have a car in Paris yet...).

This is where the role of A* comes in.

For searching for the best road, we used landmark-based heuristics. We want to move **Fairouz** from **home** to **gustaveEiffel**. Therefore, we assign a cost of 10 to all states by default and every time Fairouz takes the train, we reduce the cost by 5.

Finally, we can see that our artificial intelligence has chosen the longest but most correct path according to our needs.

```
fairouz@fairouz: ~/Bureau/MAS-DayPlannerAStarExemple
Fichier Édition Affichage Rechercher Terminal Aide
fairouz@fairouz:~/Bureau/MAS-DayPlannerAStarExemple$ time node breadth-first-search.js
- Solution found in 7 steps!
1. open Fairouz box home
2. collect-mask Fairouz home box mask
3. walk Fairouz home parkingCiteU
4. generate-authorization Fairouz parkingCiteU
5. drive-car Fairouz parkingCiteU parkingGustaveEiffel Ford
6. walk Fairouz parkingGustaveEiffel gustaveEiffel
7. take-course Fairouz Skander gustaveEiffel

real    0m47,838s
user    0m48,268s
sys     0m0,188s
fairouz@fairouz:~/Bureau/MAS-DayPlannerAStarExemple$ time node A\*\ search.js
- Solution found in 10 steps!
1. open Fairouz box home
2. collect-mask Fairouz home box mask
3. collect-navigo Fairouz home navigo
4. walk Fairouz home citeU
5. generate-authorization Fairouz citeU
6. take-train Fairouz citeU chateletB RERB
7. walk Fairouz chateletB chateletA
8. take-train Fairouz chateletA noisyChamps RERA
9. walk Fairouz noisyChamps gustaveEiffel
10. take-course Fairouz Skander gustaveEiffel

real    0m17,802s
user    0m18,044s
sys     0m0,068s
fairouz@fairouz:~/Bureau/MAS-DayPlannerAStarExemple$
```

When it comes to runtime, A* takes more time than Depth First Search but still less than Breadth First Search.

⇒ A* is the most intelligent of the searching techniques for solving a STRIPS PDDL artificial intelligence planning problem.

References

Artificial Intelligence Planning with STRIPS, A Gentle Introduction:

<http://www.primaryobjects.com/2015/11/06/artificial-intelligence-planning-with-strips-a-gentle-introduction/>

AI Planning with STRIPS

<https://www.npmjs.com/package/strips>

Homepage for the strips library

<https://github.com/primaryobjects/strips>

STRIPS-Fiddle:: An online editor for creating STRIPS automated planning programs using PDDL

<https://stripsfiddle.herokuapp.com/>

Writing Planning Domains and Problems in PDDL

<http://users.cecs.anu.edu.au/~patrik/pddlman/writing.html>