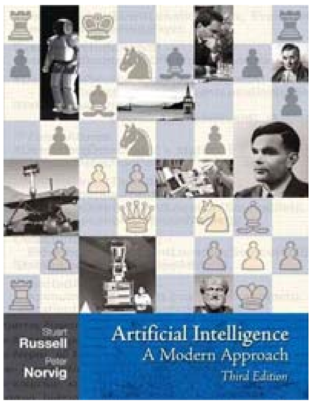
PROJECT PRESENTATION # 3 - Planning and Search

Notebook:	Artificial Intelligence NanoDegree		
Created:	10/16/2017 8:52 AM	Updated:	10/17/2017 9:06 AM
Author:	Matthew R. Versaggi		
URL:	https://stripsfiddle.herokuapp.com/		

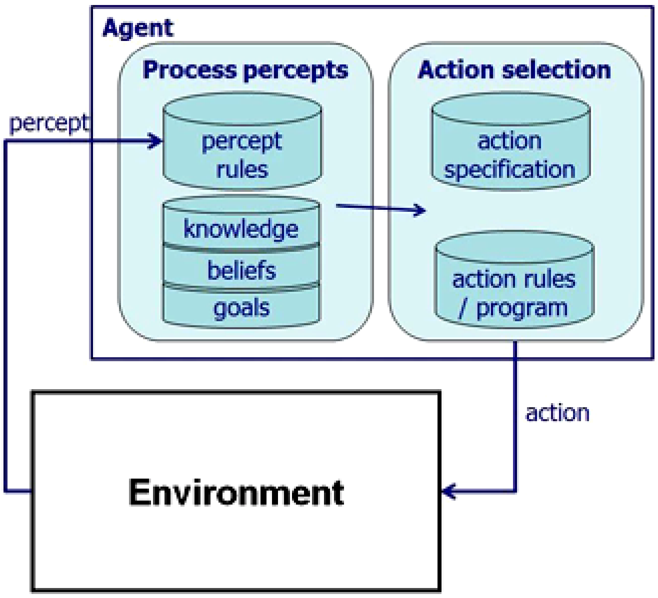
PROJECT PRESENTATION # 3 - Planning and Search

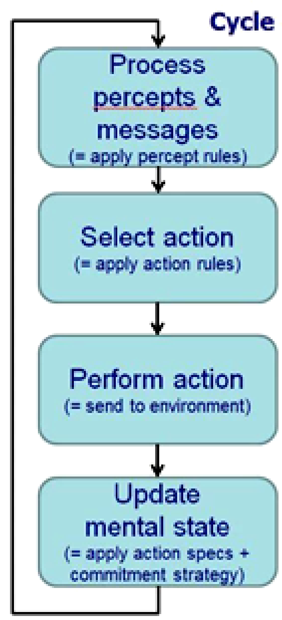


Overview of the book

The **main unifying theme** is the idea of an **intelligent agent**. We define AI as the study ofagents that receive percepts from the environment and perform actions.

GOAL: **Cognitive Intelligent Agents**



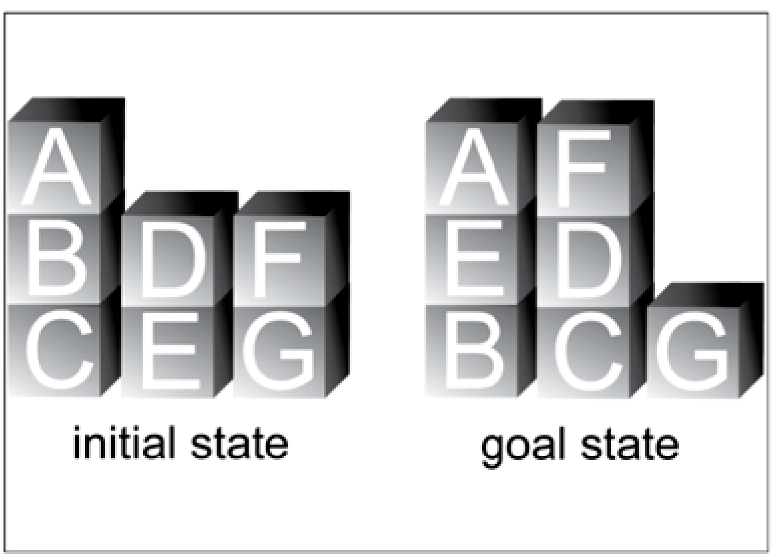


SITE:
https://goalapl.atlassian.net/wiki/spaces/GOAL/overview?mode=global

Wiki:
https://en.wikipedia.org/wiki/GOAL_agent_programming_language

- **Planning is central to Intelligent Agents**

GOAL agent program [edit]



An example blocks world problem

Part 1 - Planning problems

THEORY:
https://medium.com/towards-data-science/ai-planning-historical-developments-edcd9f24c991

Example of applying the **STRIPS** language to an **Air Cargo transport system** using a planning search agent. Suppose we have an initial state of Cargo 1 at SFO, Cargo 2 at JFK, Plane 1 at SFO and Plane 2 at JFK.

We want to formulate an optimal plan to transport Cargo 1 to JFK and Cargo 2 to SFO. Summarizing this problem description, we have:

```
Init(At(C1, SFO) ∧ At(C2, JFK)
 ∧ At(P1, SFO) ∧ At(P2, JFK)
 ∧ Cargo(C1) ∧ Cargo(C2)
 ∧ Plane(P1) ∧ Plane(P2)
 ∧ Airport(JFK) ∧ Airport(SFO))

Goal(At(C1, JFK) ∧ At(C2, SFO))
```

PYTHON:

```
def air_cargo_p1() -> AirCargoProblem:
     cargos = ['C1', 'C2']
     planes = ['P1', 'P2']
     airports = ['JFK', 'SFO']
     pos = [expr('At(C1, SFO)'),
          expr('At(C2, JFK)'),
          expr('At(P1, SFO)'),
          expr('At(P2, JFK)'),
          ]
     neg = [expr('At(C2, SFO)'),
          expr('In(C2, P1)'),
          expr('In(C2, P2)'),
          expr('At(C1, JFK)'),
          expr('In(C1, P1)'),
          expr('In(C1, P2)'),
          expr('At(P1, JFK)'),
          expr('At(P2, SFO)'),
          ]
     init = FluentState(pos, neg)
     goal = [expr('At(C1, JFK)'),
             expr('At(C2, SFO)'),
             ]
return AirCargoProblem(cargos, planes, airports, init, goal)
```

PDDL and STRIPS Fiddle URL:
https://stripsfiddle.herokuapp.com/

AIR CARGO PROBLEM: UDACITY Style

Inspiration:
https://medium.com/towards-data-science/ai-planning-historical-developments-edcd9f24c991

Udacity GitHub:
https://github.com/udacity/AIND-Planning

Project Submission Result: (P1, S1)

> python run_search.py -p 1 -s 1

Solving Air Cargo Problem 1 using **breadth_first_search**...

Expansions	Goal Tests	New Nodes
43	56	180

Plan length: 6 Time elapsed in seconds: 0.03145325632251838

1. Load(C1, P1, SFO)
2. Load(C2, P2, JFK)
3. Fly(P2, JFK, SFO)
4. Unload(C2, P2, SFO)
5. Fly(P1, SFO, JFK)
6. Unload(C1, P1, JFK)

FIDDLE Air Cargo Planner Result:

Solution found in 6 steps!

1. LOAD C1 P1 SFO

2. LOAD C2 P2 JFK

3. FLY P1 SFO JFK

4. UNLOAD C1 P1 JFK

5. FLY P2 JFK SFO

6. UNLOAD C2 P2 SFO

CODE REVIEW:

MY_AIR_CARGO_PROBLEMS.PY

TODO: Implement methods and functions in `my_air_cargo_problems.py`

- `AirCargoProblem.`**`get_actions`** method including **`load_actions`** and **`unload_actions`** sub-functions
- `AirCargoProblem.`**`actions`** method
- `AirCargoProblem.`**`result`** method
- `air_cargo_`**`p2`** function
- `air_cargo_`**`p3`** function

Part 2 - Domain-independent heuristics

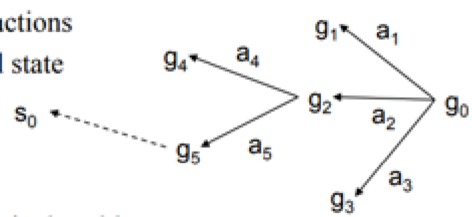
PLANNING GRAPHS and HEURISTICS

THEORY:

Motivation:

Motivation

- A big source of inefficiency in search algorithms is the *branching factor*
  - the number of children of each node
- e.g., a backward search may try lots of actions that can't be reached from the initial state



- One way to reduce branching factor:
- First create a *relaxed problem*
  - Remove some restrictions of the original problem
    - » Want the relaxed problem to be easy to solve (polynomial time)
  - The solutions to the relaxed problem will include all solutions to the original problem
- Then do a modified version of the original search
  - Restrict its search space to include only those actions that occur in solutions to the relaxed problem

Inspiration Links: - Graph Planning Techniques

- http://pages.mtu.edu/~nilufer/classes/cs5811/2009-fall/lecture-slides/cs5811-ch11b-graphplan.pdf
- https://www.cs.umd.edu/~nau/planning/slides/chapter06.pdf

CODE REVIEW:

MY_PLANNING_GRAPH.PY

TODO: Implement a Planning Graph with automatic heuristics in `my_planning_graph.py`

- `PlanningGraph.`**`add_action`**`_level` method
- `PlanningGraph.`**`add_literal`**`_level` method
- `PlanningGraph.`**`inconsistent_effects`**`_mutex` method
- `PlanningGraph.`**`interference`**`_mutex` method
- `PlanningGraph.`**`competing_needs`**`_mutex` method
- `PlanningGraph.`**`negation`**`_mutex` method
- `PlanningGraph.`**`inconsistent_support`**`_mutex` method
- `PlanningGraph.`**`h_levelsum`** method

Part 3 - Heuristic Analysis

- **See Heuristic Analysis PDF file.**
- **See Test Results Spreadsheet file.**

Part 4 - Research Review

[Tip: The book *Artificial Intelligence: A Modern Approach* by Norvig and Russell is chock full of references in the Bibliographical and Historical notes at the end of Chapter 10.]

AI Planning Historical Developments:
https://medium.com/towards-data-science/ai-planning-historical-developments-edcd9f24c991

Recent Advances in AI Planning:
https://homes.cs.washington.edu/~weld/papers/pi2.pdf

Progress in AI Planning Research and Applications:
https://www.researchgate.net/publication/242415929_Progress_in_AI_Planning_Research_and_Applications

An overview of recent algorithms for AI Planning:
http://www.cs.toronto.edu/~sheila/2542/w06/readings/RintanenHoffmann01.pdf