CS310 – AI Foundations

Andrew Abel

March 2024

Week 10: Introduction to Planning

Welcome!

- Welcome to CS310 week 10!
- This week's topics
  - Introduction to planning
  - Domain and Problem
  - PDDL
- Thinking Task
  - No tutorial sheet this week
- Lab:
  - Planning

Next week...Week 11

- No lecture or tutorial next week
  - I'm at a conference!
- Labs will run as normal
  - No week 11 lab sheet, just sign off for week 10
  - Week 10 (this week) is the final lab sheet
- Revision class will run before the exam
  - Date tbc, will likely be online
  - I'll open a revision forum for you to ask questions!

Previous topics

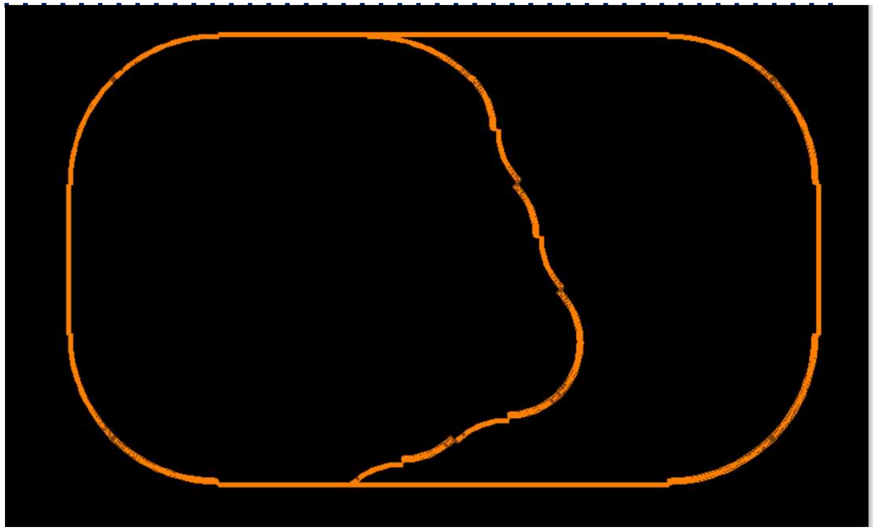
- Search to identify a goal state
- Game trees to identify optimal values
- Logical Agents to use facts about the world to decide actions
- Neural networks to classify and predict

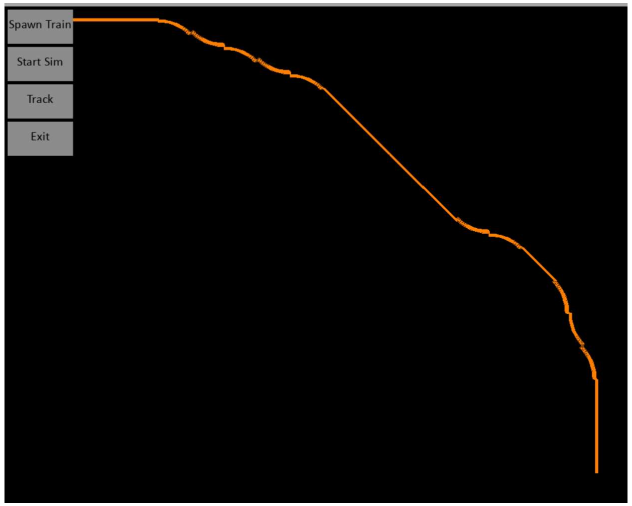
Student Projects – The Railways

- Overall aim is to create a full model railway world, including track design, management, and real world control
- Divided into several projects
  - 1. Physical representation
    - For CES students
  - 2. Simulation, visualisation and management
  - 3. Track layout and design
- 2023 – Three project students
- 2024 – Projects will be available

The Railways 1 – Track layout

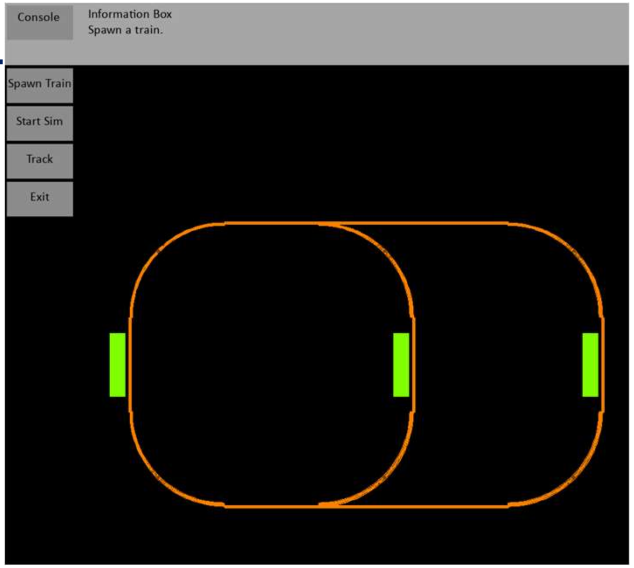
- Given a pile of track pieces, can we make a good track layout?
  - What IS a good track layout?
- We use A* search (initially) and Monte Carlo Tree Search, using evaluation functions
  - Multiple corners
  - More straights
  - Cover area
  - Join up
- Can generate tracks of different styles

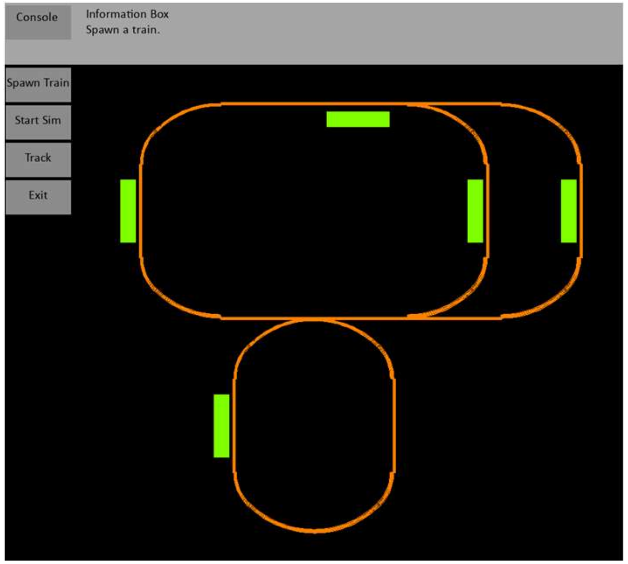




The Railways 2 – Visualisation and Simulation

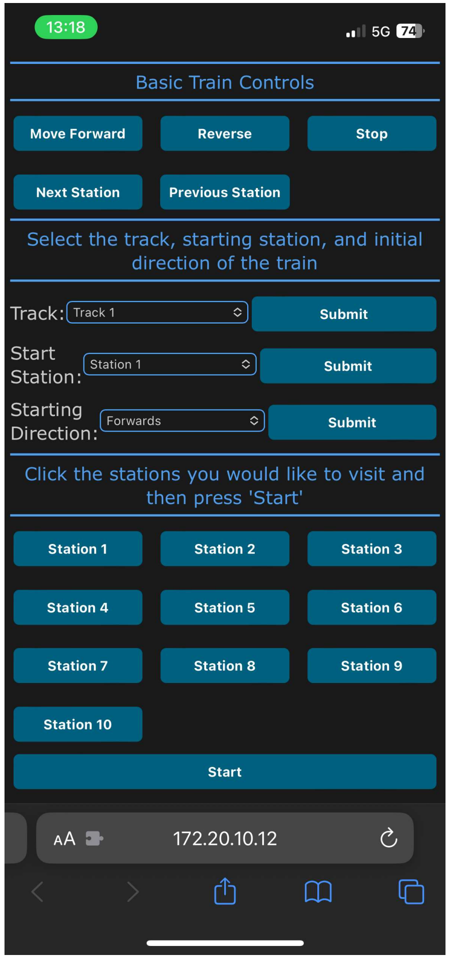
- Given a track design, can we visualise and simulate in Python
- Can choose number of trains, track design
- Uses planning (today's lecture!) to plan routes
    - Multiple trains can run around the same track
    - Routing handled by computer

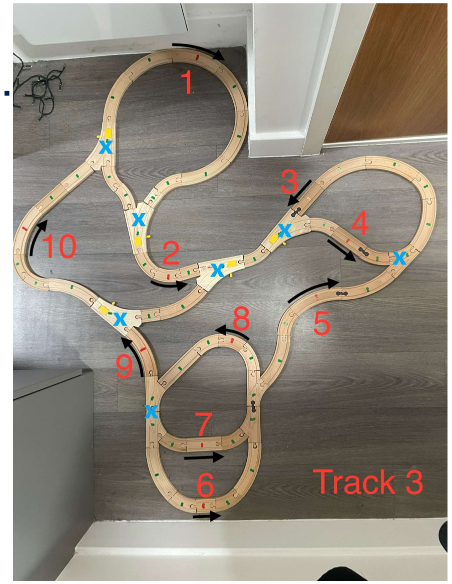


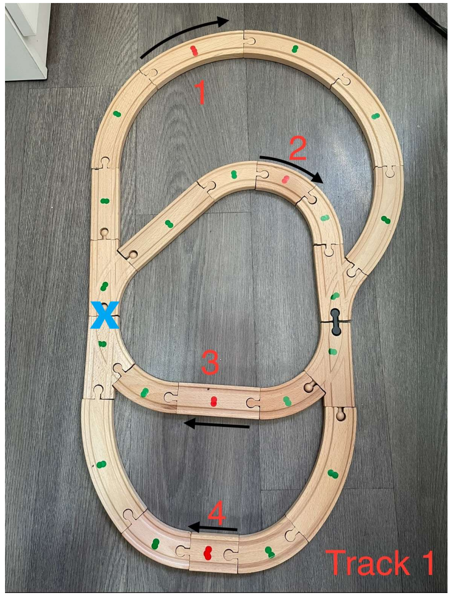


The Railways 3 - Modelling

- Automate train journeys on a physical track
- Use planning to plan complex routes
- Linked with mobile phone
- Uses RPi Pico



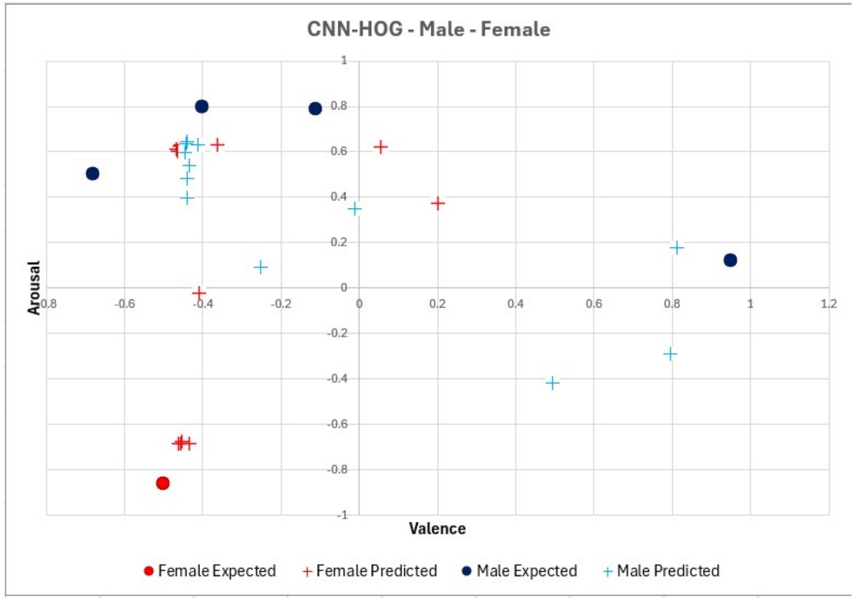




Emotion Recognition

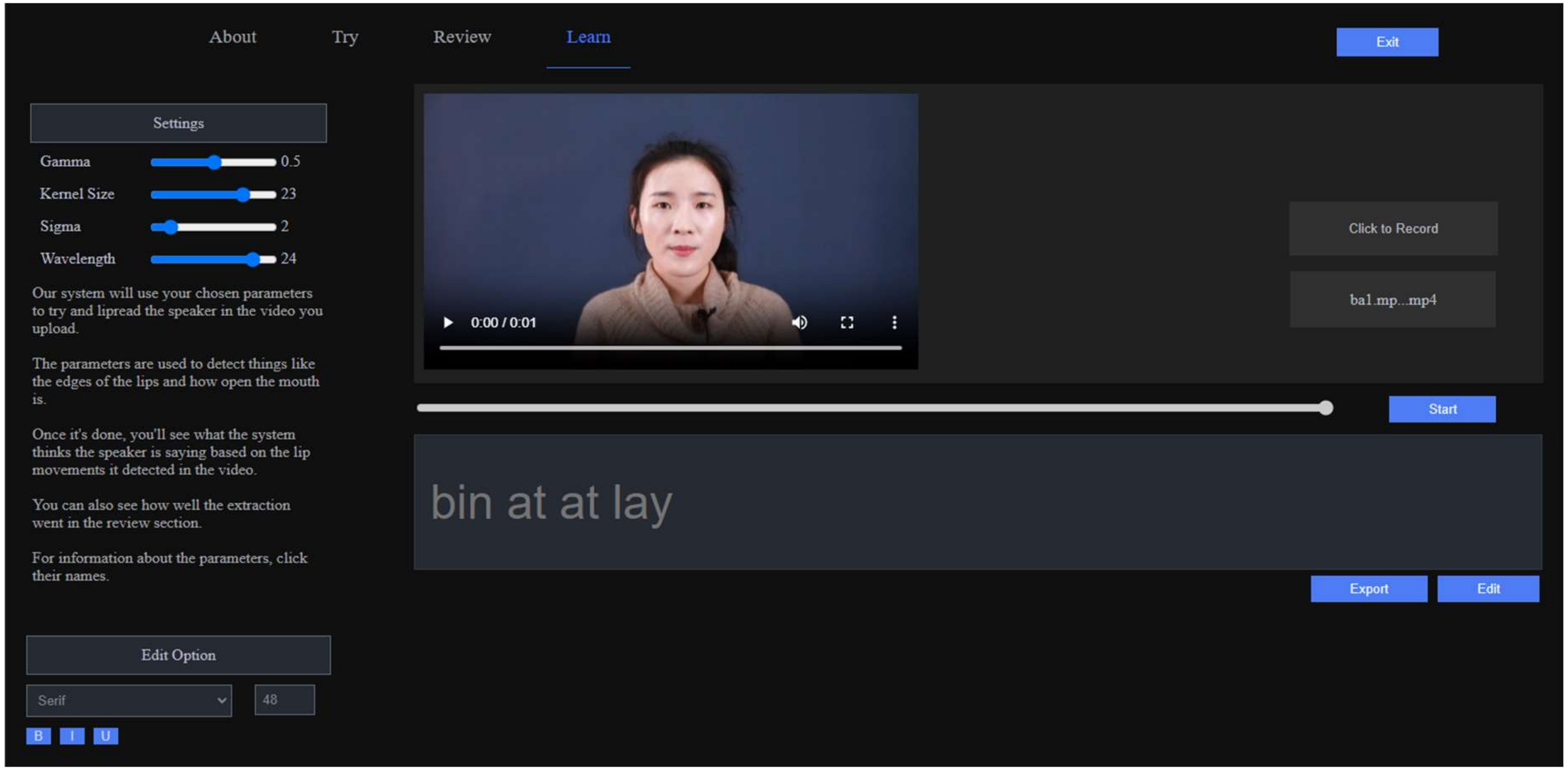
- We are trying to recognise emotions from datasets
- Using different features
  - Lip reading features, head movement etc.
- Self learned features
  - Convolutional Neural Networks
- Neural networks
  - Feedforward and Recurrent

```
model = Sequential()
#model.add(BatchNormalization(synchronized=True))
model.add(LSTM(120, input_shape=(input_shape[1], input_shape[2]), activation='tanh', return_sequences=True))
model.add(LSTM(100, activation='tanh', return_sequences=True))
model.add(Dropout(0.1))
model.add(LSTM(80, activation='tanh', return_sequences=True))
model.add(Dense(80))
model.add(LSTM(40, activation='tanh', return_sequences=True))
model.add(LSTM(20, activation='tanh', return_sequences=True))
model.add(LSTM(10, activation='tanh', return_sequences=True))
model.add(LSTM(5, activation='tanh'))
#model.add(Dense(52))  # Output layer with 2 units for valence and arousal
model.add(Dense(1))
return model
```



Speech Recognition

- Use Feature-based lipreading
- Play videos and predict speech
- Machine learning with software development



This week Planning

- Short Definition:
  - Planning is the act of thinking before acting.
- Longer Definition:
  - Planning is the process of choosing and organising actions that lead towards a goal, based on a high-level description of the world.
- Everyone plans
  - We have a goal state (survive lecture, drive home, go on holiday)
  - We have some knowledge about the world
    - Car pedal and switch operations
    - Route map
    - Other traffic on road
  - Some actions we can take
    - Accelerate, brake, turn etc.

Planning

- Domain specific planning uses representation or methods that are adapted to solving a specific problem.
  - many important domains: path and motion planning, manipulation planning, communication planning, etc.
- Domain-independent planning uses a general representation and technique that is applicable across different domains.
  - still many kinds of general planning: online and offline; discrete and continuous; deterministic and non-deterministic; fully- and partially observable; sequential and temporal.

Uses of Planning

Planning Uses

- Planning is rational behaviour
  - Uses facts about the world, makes logical decisions

- Very widely used
  - How to navigate a room
  - How to fix space station problems
  - How to balance electricity grid loads
  - Very important in robotics
- The output is not the solution, but the series of steps to reach the solution
  - What steps are needed to solve the problem

Planner

- A planner is a system that finds a sequence of actions to accomplish a specific task
- Given a set of facts and possible actions, will search for a state that matches goal start
  - i.e. Search trees!

Planning Problem

- The main components of a planning problem are:
    - a description of how the world behaves and the capabilities of the agent (e.g. the action library).
        - The domain
    - a description of the initial situation (the initial state).
    - a description of the desired situation (the goal)
- A basic planning formalism represents the state of the world and actions using propositional variables. Such a (classical) planning problem is a tuple: $< F, A, I, G >$, where:
    - F is a set of (Boolean) propositions.
    - A is a set of deterministic actions.
    - The set of states S is the power set of F, $S = 2^F$.
    - $s_o \in S$ is the initial state.
    - $G : S \rightarrow \{T, \perp\}$ is the goal function.

Planning Problem

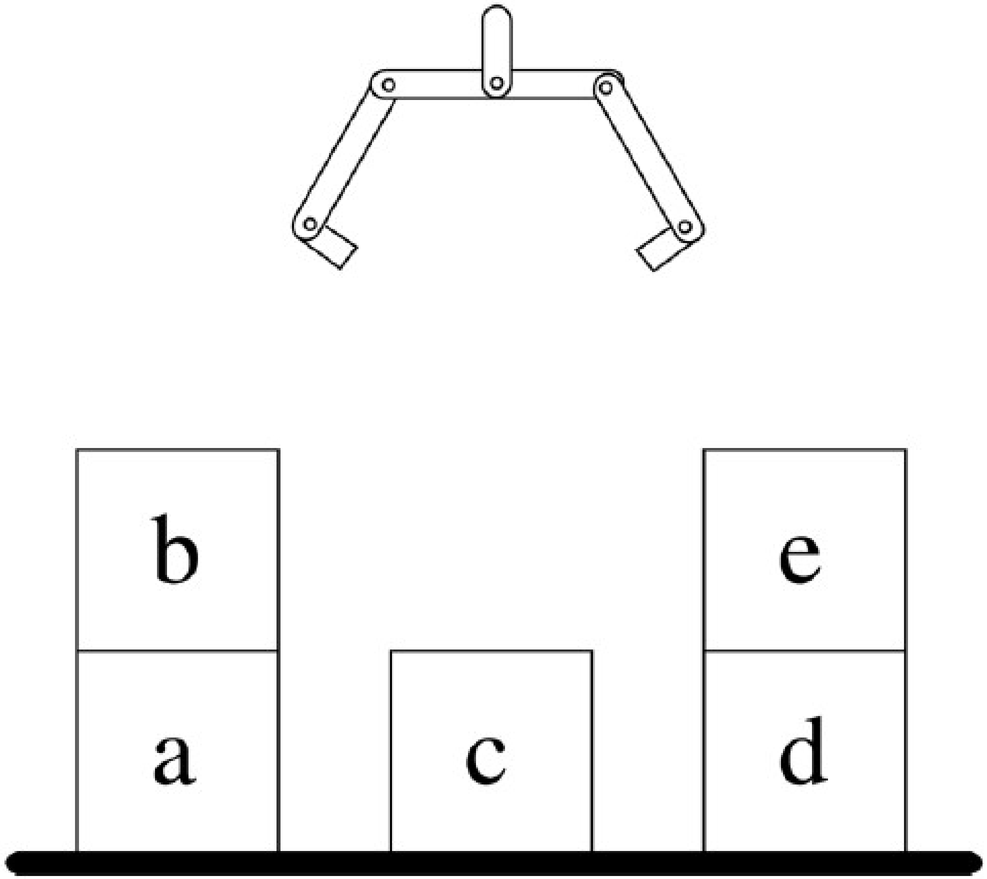
- A (classical) planning problem is a tuple: < F,A, I ,G >, where:
  - F is a set of (Boolean) propositions.
  - A is a set of deterministic actions.
  - The set of states S is the power set of F, $S = 2^F$.
  - $s_o \in S$ is the initial state.
  - $G : S \rightarrow \{T, \perp\}$ is the goal function
- Each action $a \in A$ consists of:
  - $pre(a) \subseteq F$ (simple preconditions)
  - $add(a) \subseteq F$ (add effects)
  - $del(a) \subseteq F$ (delete effects)

Classical Planner Assumptions

- They operate on basic STRIPS actions
  - Stanford Research Institute Problem Solver
  - A formal language for planning, first developed in 1971
- Important assumptions:
  - The agent is the only source of change in the world, otherwise the environment is static
  - All the actions are deterministic
  - The agent is omniscient: knows everything it needs to know about start state and effects of actions
  - The goals are categorical, i.e. unambiguous, and the plan is considered successful iff all the goals are achieved

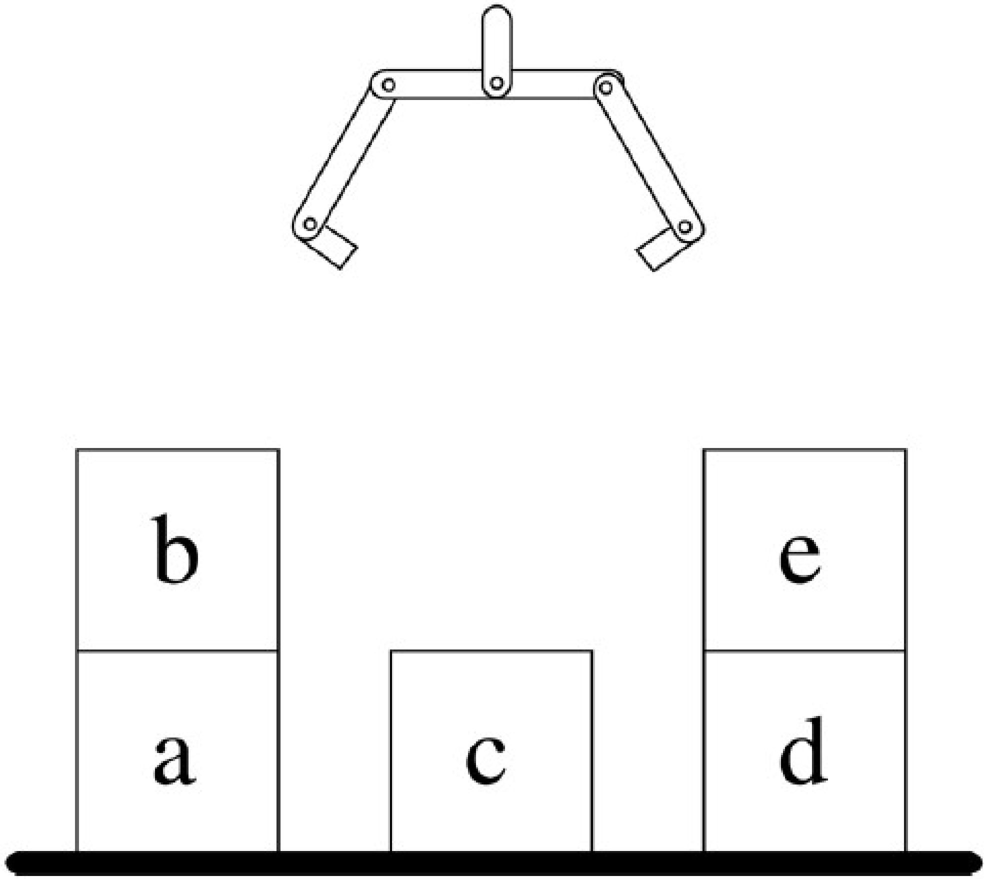
Blocks world

- Can be represented using predicates
  - i.e. statements of facts
- We want to model the world and all relevant facts
- Where are all the blocks? What is the gripper holding?
- What actions can we take to change the world?
- What do we want to produce?



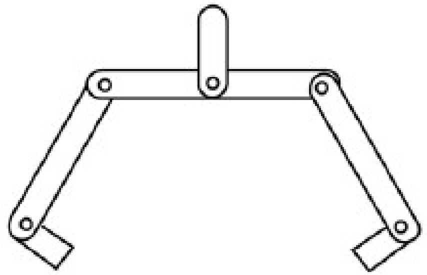
Blocks world - Predicates

- ontable(a)
- ontable(c)
- ontable(d)
- on(b,a)
- on(e,d)
- clear(b)
- clear(c)
- clear(e)
- gripping()
- The current state of the world

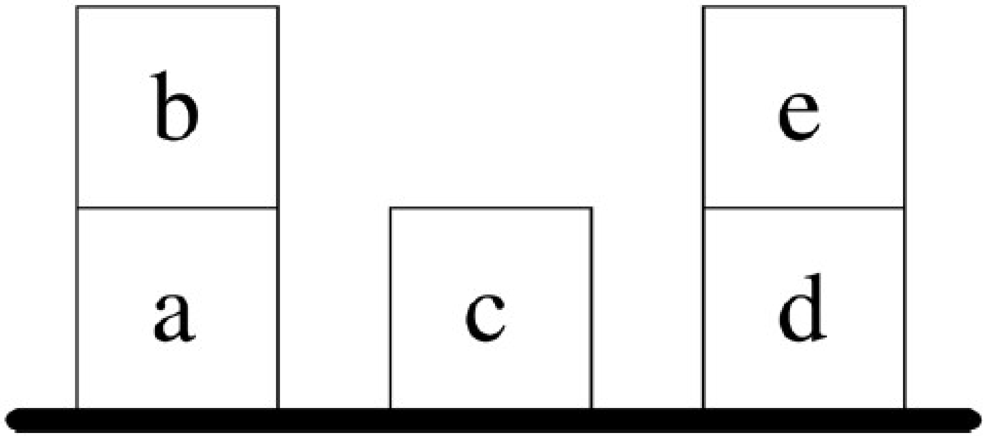


Blocks world - Actions

- We consider the problem abstractly
  - We assume movement will be handled, and that the robot arm will reach the block



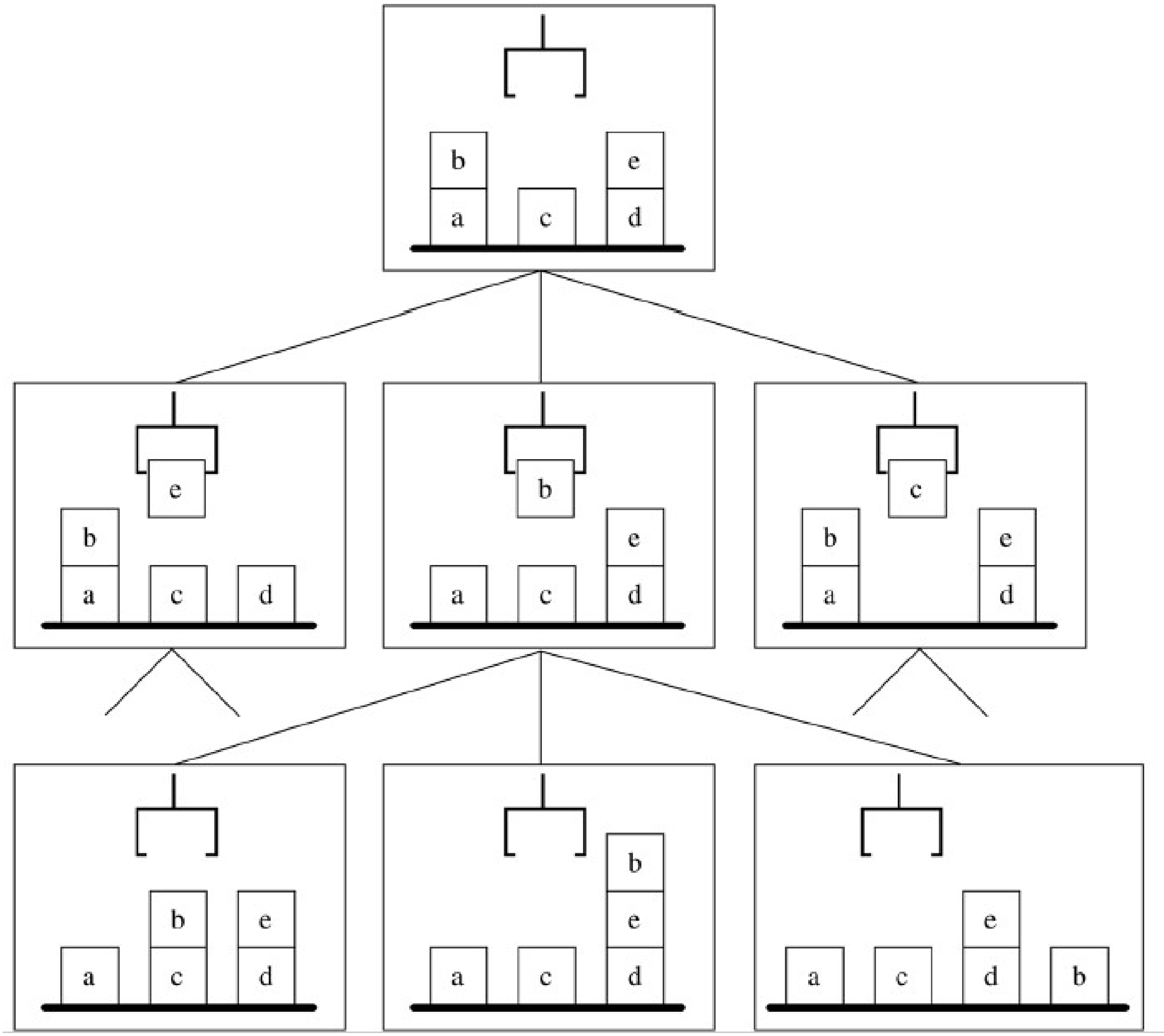
- **pickup (W):**
  - pick up block W from its current location on the table and hold it



- **putdown (W):**
  - place block W on the table
- **stack (U, V) V):**
  - place block U on top of block V
- **unstack (U, V) V):**
  - remove block U from the top of block V and hold it

Blocks World State Space

- Possible actions
- Resulting next states
- This should be familiar!



Blocks World Actions

- In STRIPS, we have operators (actions)
- Operators are defined with:
  - Name
  - Parameters
  - Preconditions
  - Results
    - Predicates added to domain
    - Predicates deleted from domain

Blocks World Operators

- Pickup(X)
  - Parameters
    - X – a block
  - Preconditions
    - gripping() ^ clear(X) ^ ontable(X)
  - Added to domain
    - gripping(X)
  - Deleted
    - ontable(X) ^ gripping() ^ clear(X)
- This action relies on certain facts being true in order to take place, and then changes the facts afterwards.

pickup(X)

P: gripping() $\wedge$ clear(X) $\wedge$ ontable(X)
A: gripping(X)
D: ontable(X) $\wedge$ gripping() $\wedge$ clear(X)

Blocks World Operators

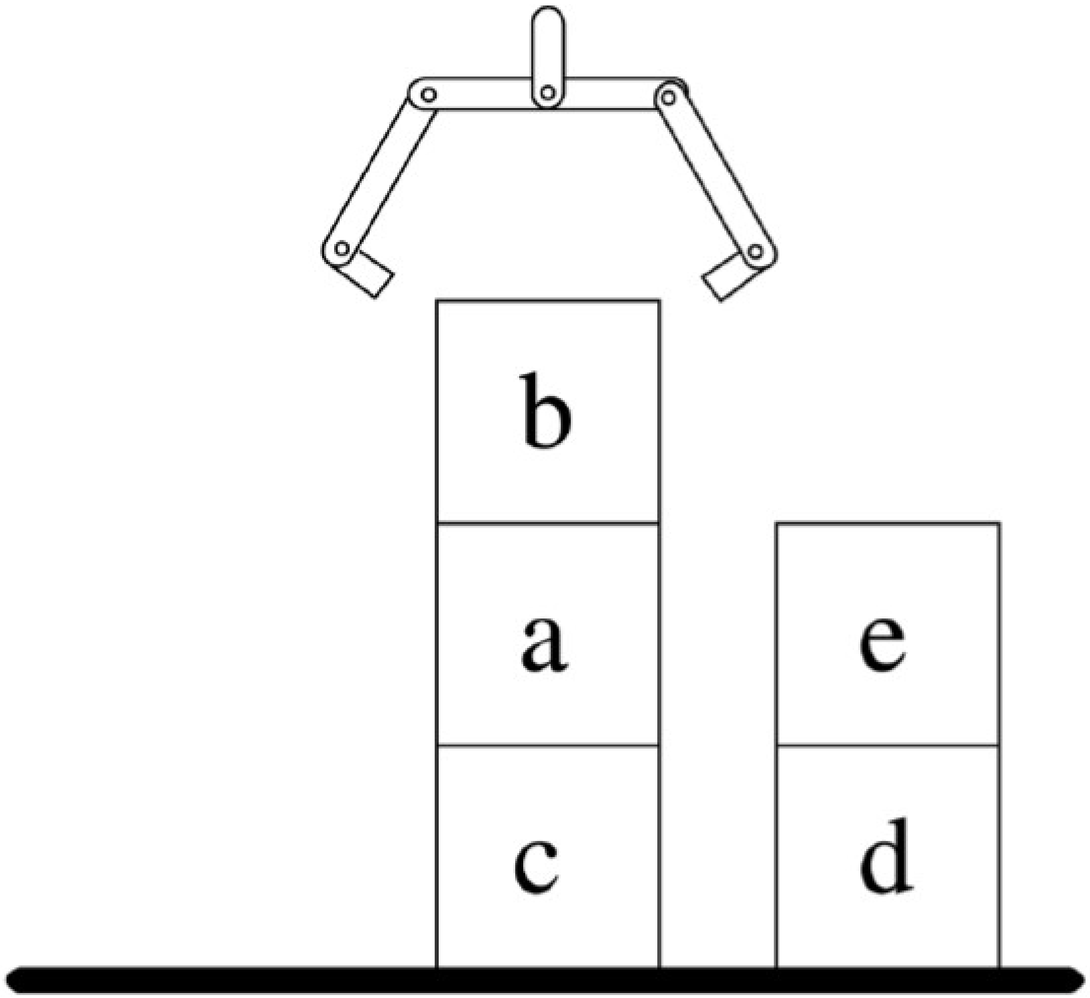
pickup(X)	P: gripping() $\wedge$ clear(X) $\wedge$ ontable(X) A: gripping(X) D: ontable(X) $\wedge$ gripping() $\wedge$ clear(X)
putdown(X)	P: gripping(X) A: ontable(X) $\wedge$ gripping() $\wedge$ clear(X) D: gripping(X)
stack(X,Y)	P: gripping(X) $\wedge$ clear(Y) A: on(X,Y) $\wedge$ gripping() $\wedge$ clear(X) D: gripping(X) $\wedge$ clear(Y)
unstack(X,Y)	P: gripping() $\wedge$ clear(X) $\wedge$ on(X,Y) A: gripping(X) $\wedge$ clear(Y) D: on(X,Y) $\wedge$ gripping() $\wedge$ clear(X)

Operators

- Pre and post conditions are conjunctions, i.e all must be true
- Note that we have 2 operators for pick up, why?
    - Different actions have different pre and post conditions
    - We have to either take from table, or take from stacks

Goal State

- Have to define our goal
- ontable(c)
- ontable(d)
- on(a,c)
- on(b,a)
- on(e,d)
- clear(b)
- clear(e)
- gripping()



Goal State

- Can be much more complex than simple structures
  - For example, task is to navigate train through a series of stations in a given order
  - Put all light switches on

```
(:goal (and
  (on switch_1)
  (on switch_2)
))
```

  - Navigate 2 people to a destination

```
(:goal
     (and
      (outsidetaxi scott)
      (outsidetaxi stuart)
      (plocation scott graham_hills)
      (plocation stuart livingstone_tower)
)
```

State Space

- Predicates can now represent a state
- Initial state:
    - ontable(a) ^ ontable(c) ^ontable(d) ^ on(b,a) ^ on(e,d) ^ clear(b) ^ clear(c) ^ clear(e) ^ gripping()
- Goal State:
    - ontable(c) ^ ontable(d) ^ on(a,c) ^ on(b,a) ^ on(e,d) ^ clear(b) ^ clear(e) ^ gripping()
- We can search this!

State Space

- Search the state space
    - Each node is a state
- Root of tree is initial state
- Nextstates are valid actions
- Goal test involves checking if goal set is a subset of a given state
- You know about this!

Planning - Hard Problem

- Large branching factor, overwhelming number of possibilities
- If we add extra blocks or extra grippers, suddenly the problem becomes much greater
  - Familiar from your search work
- There are many ways blocks can interact to potentially reach the goal
  - Sometimes making one predicate true may make another false
- When subgoals are compatible, i.e., they do not interact, they are said to be linear (or independent, or serializable).
- Life is easier for a planner when the subgoals are independent because then divide-and-conquer works.
  - Solve one subgoal at a time

Planners

- Given knowledge of the world, come up with a solution
  - We have our goal state
    - i.e. Facts that are true
  - We have our actions
    - How to get next states
  - We have initial state
- Planning is then performed by dedicated planners

Planning Summary

- Similar to what we have learned so far
  - Given an initial state, try to solve goal
- Representing problems using STRIPS

PDDL

Planning Domain Definition Language

- PDDL is a language for encoding classical planning tasks.
- Planning Domain Definition Language
  - https://planning.wiki/
- First defined in 1998
  - Ghallab, M., Knoblock, C., Wilkins, D., Barrett, A., Christianson, D., Friedman, M., Kwok, C., Golden, K., Penberthy, S., Smith, D., Sun, Y., & Weld, D. (1998). PDDL - The Planning Domain Definition Language.
- Designed to be a general purpose planning/scheduling specifier
  - Has been modified many times since then
- Several PhD students in this department focused on planning
- Andrew's note – arguably PDDL didn't take off like it might have been hoped, people generally prefer to make their own specialised planning software

Planning Domain Definition Language

- PDDL represents the properties of the world and the state of the world
  - Can be solved with PDDL planners
- Tasks are separated into two files:
  - 1. Domain File, which contains:
    - Predicates that describe the properties of the world.
    - Operators that describe the way in which the state can change.
  - 2. Problem File, which contains:
    - Objects: the things in the world.
    - The initial state of the world.
    - The goal specification.
- Propositions about the world and actions are found by applying the object terms to the predicates and operators.

Format - PDDL

- Heavy use of (brackets)
- A fairly formal language
- Variables defined with a ?
- For example, to create a predicate for a taxi being at a location:

```
(tlocation ?taxi1 - taxi ?location1 - location)
```

- Here we define a predicate tlocation
  - Has 2 variables ?taxi1 of type taxi and ?location1 of type location

- (:requirements :strips :equality :typing :conditional-effects)

Example: Domain File

```
(define (domain simple_switches)
    (:requirements :typing)
    (:types switch)
    (:predicates
        (off ?s - switch) (on ?s - switch))
    (:action switch_on
        :parameters (?s - switch)
        :precondition (off ?s)
        :effect (and (not (off ?s)) (on ?s))
    )
```

- Domain name
  - We call this one “simple switches”
- Requirements
  - What aspects of PDDL we are going to use
  - :typing –can have different types of variables
- https://planning.wiki/ref/pddl/requirements
- For lab this week, we can have:
  - : strips – basic add and delete
  - : typing – similar to classes
  - : equality – allows use of “=“ to compare objects
  - : conditional-effects – “when”, i.e. when x is true, do y

```
(:requirements :strips :equality :typing :conditional-effects)
```

Example: Domain File

- Types of variables we can have
  - Separated by a space
  - Just one here, switch
- Can have as many as we want
  - Cars, trains, stations, locations, people
- Can also have types as a subtype, but don't need to worry about this!

```
(define (domain simple_switches)
    (:requirements :typing)
    (:types switch)
    (:predicates
        (off ?s - switch) (on ?s - switch))
    (:action switch_on
        :parameters (?s - switch)
        :precondition (off ?s)
        :effect (and (not (off ?s)) (on ?s))
    )
```

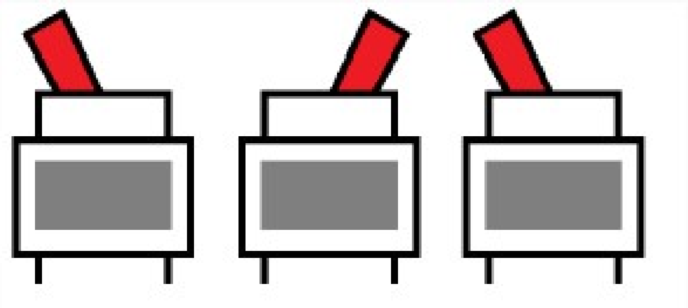
Example: Domain File

- Possible predicates
  - Simplest case
  - Switch can be on or off
- Note that we are not specifying a specific switch.
  - Any variable that is of type switch can have the predicate off = true, or off=false
  - We are dealing with Booleans here
- Variables defined with ?
  - (off ?s – switch)
  - Variable s is a switch and can be off

```
(define (domain simple_switches)
    (:requirements :typing)
    (:types switch)
    (:predicates
        (off ?s - switch) (on ?s - switch))
    (:action switch_on
        :parameters (?s - switch)
        :precondition (off ?s)
        :effect (and (not (off ?s)) (on ?s))
    )
```

Example: Domain File

- Switches
- 2 predicates, a switch can be on or off
- Currently just one action – switch on



```
(define (domain simple_switches)
    (:requirements :typing)
    (:types switch)
    (:predicates
        (off ?s - switch) (on ?s - switch))
    (:action switch_on
        :parameters (?s - switch)
        :precondition (off ?s)
        :effect (and (not (off ?s)) (on ?s))
    )
```

Example: Domain File

- Actions defined with :action
- Note brackets around action
- Logic can be represented
  - not(off ?s)
  - and((on ?s) (off ?s))
  - or((off ?s) (off ?s2))
- Only one action in this world, cannot switch switches off!

```
(define (domain simple_switches)
    (:requirements :typing)
    (:types switch)
    (:predicates
        (off ?s - switch) (on ?s - switch))
    (:action switch_on
        :parameters (?s - switch)
        :precondition (off ?s)
        :effect (and (not (off ?s)) (on ?s))
    )
```

Action

- One action here, named switch_on
- It takes only 1 parameter
  - A switch
- It has a precondition that the switch passed in must be off,
  - Off ?s = true
- Effect, use of “and” for multiple effects
  - Off ?s = false
  - On ?s = true

```
(define (domain simple_switches)
    (:requirements :typing)
    (:types switch)
    (:predicates
        (off ?s - switch) (on ?s - switch))
    (:action switch_on
        :parameters (?s - switch)
        :precondition (off ?s)
        :effect (and (not (off ?s)) (on ?s))
    )
```

Domain and Problem

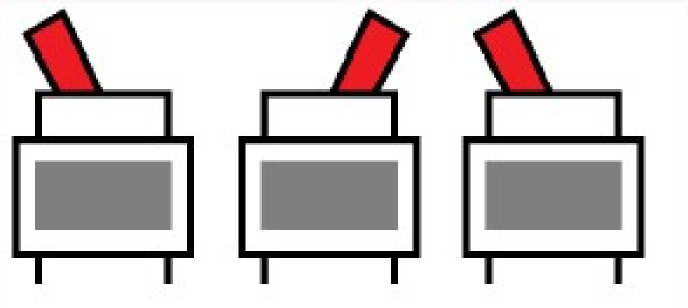
- Note again, we do not specify a specific state in the domain
  - We specify that we can have switches, and that the switches can be off and on.
  - We do not specify how many switches we have
- We use a problem file to provide the state of the world
  - Problem file has to be compatible with domain

Example: Problem File

- Link to correct domain
- Define 3 switches
  - In our world, we have 3 objects
- Set their initial values as all off
- We use the variable types in domain file
  - s1 – switch
  - (off s1)
- We set our goal
  - Can be any goal we want, so long as it relates to the world

```
(define (problem more_switches)
    (:domain simple_switches)
    (:objects s1 s2 s3 - switch)
    (:init (off s1) (off s2) (off s3))
    (:goal (and (on s1) (on s2) (on s3)))
)
```

Example: Problem File



- Link to correct domain
- Define 3 switches
- Set their initial values as all off
- Goal is to have all switches on
- Note, we do not specify actions to take, the planner determines route to goal

```
(define (problem more_switches)
    (:domain simple_switches)
    (:objects s1 s2 s3 - switch)
    (:init (off s1) (off s2) (off s3))
    (:goal (and (on s1) (on s2) (on s3)))
)
```

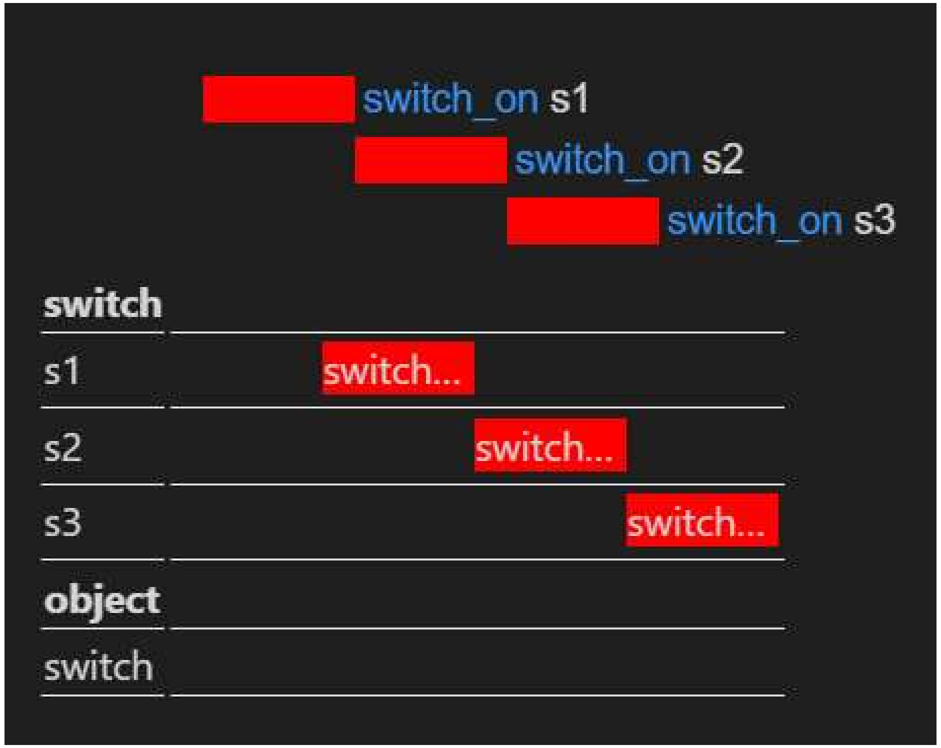
Example: Plan

A plan for a classical planning problem is a sequence of actions that are applicable from the initial state and lead to a state that satisfies the goal:

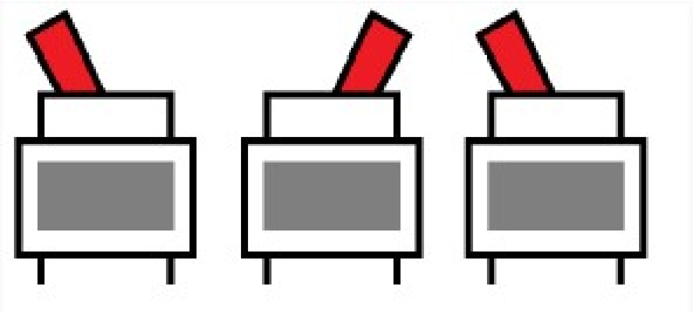
(switch_on s1)

(switch_on s2)

(switch_on s3)

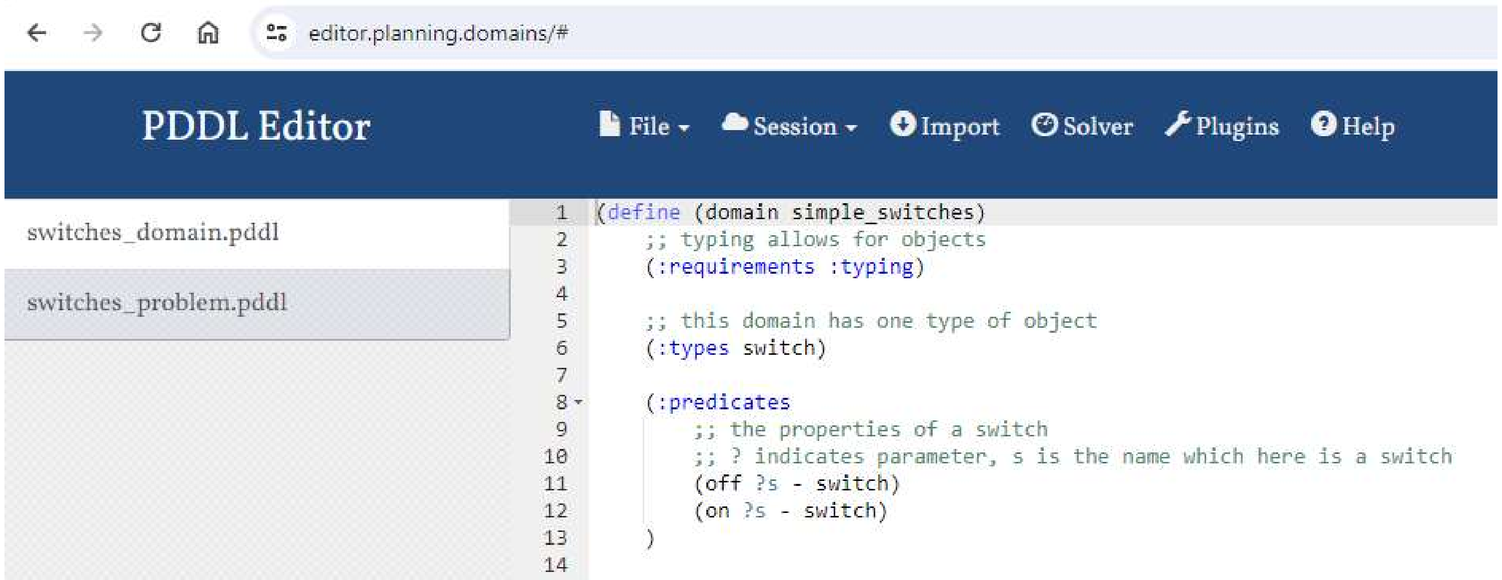


- This plan can then potentially be used as input for a robot/program



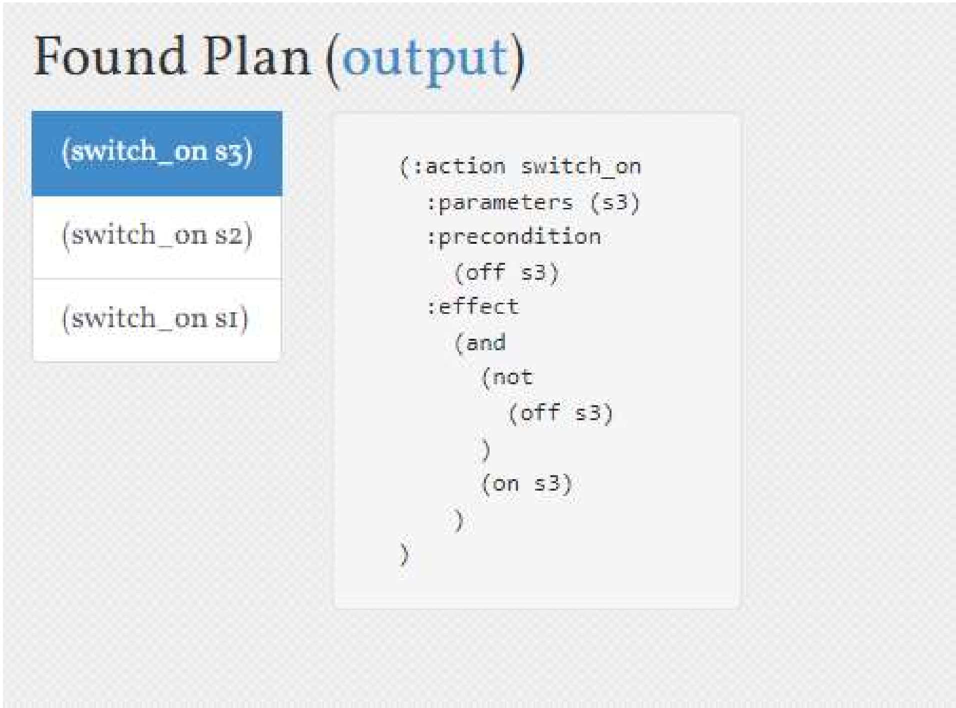
PDDL Editor

- Online editor to solve plans
  - http://editor.planning.domains/#
- Can upload problem and domain
- Can also install PDDL for VSCode



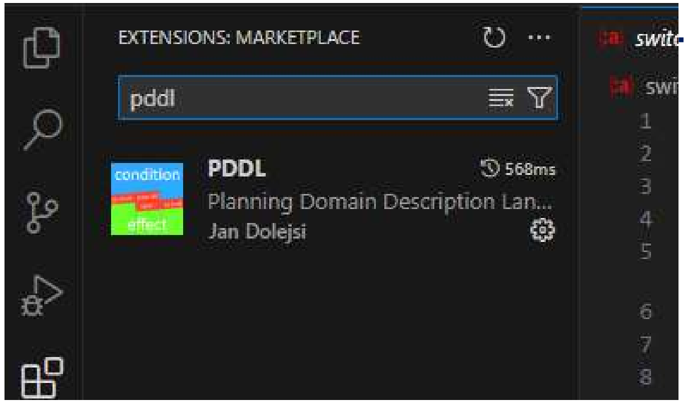
Plan Output

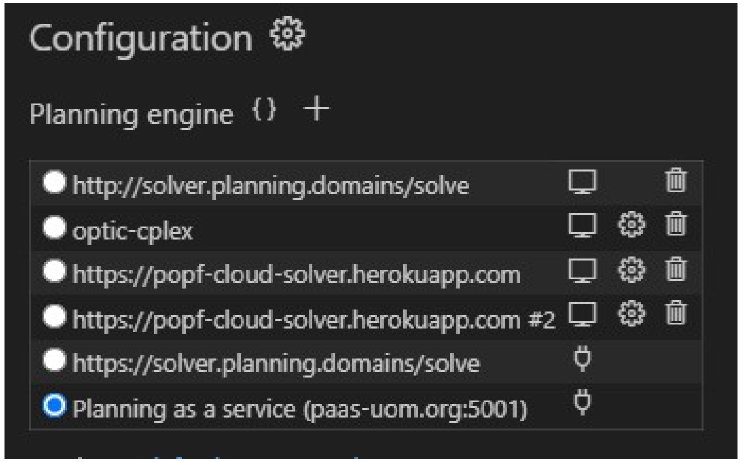
- Using Online planner
- Will give you plan



PDDL in VSCode

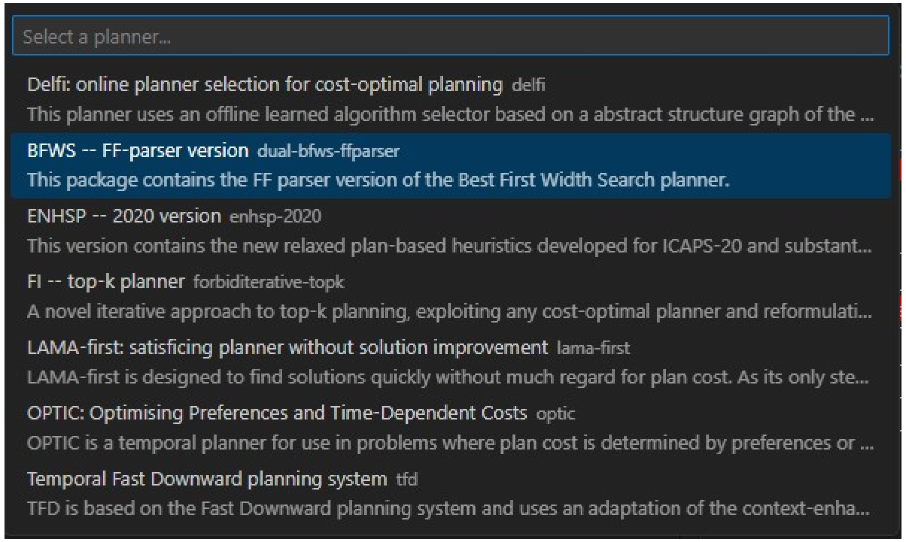
- Can install PDDL in VSCode
- Uses an online planner by default
  - This is much easier in Linux!
  - In Linux, can use whatever you want
- Can select planner in PDDL setup page
  - In Windows, choose “Planning as a service”





PDDL in VSCode

- Run planner by selecting problem or domain and then Alt-P
  - Choose a planner



Solving Puzzles

- How about we make it harder?
- Currently very easy, can switch on anything!
- Now we have 5 switches in a row:
  - A switch can only be switched on if it has a neighbour that is already on.
  - The five switches are in a row so that each switch has two neighbours, except the two at the ends of the row which only have one.
  - The five switches are in initial positions: {off, off, on, off, off}.
- What do we need to do?

Domain

- We know the switches need to be in a row
  - Add a neighbour predicate
- Make a tweak to the switch on action
- Now we need 2 parameters
  - Any 2 switches
- Switches must satisfy neigbour predicate

```
(:predicates
    ;; the properties of a switch
    (off ?s - switch)
    (on ?s - switch)
    (neighbours ?s1 ?s2 - switch)
)
```

```
(:action switch_on
    :parameters (?s ?s2 - switch)
    :precondition (and
        (off ?s)
        (on ?s2)
        (neighbours ?s ?s2)
    )
    :effect (and
        (not (off ?s))
        (on ?s)
    )
)
```

Problem File

- Here we need to make changes
  - Add some extra switches
- Initialise by defining initial positions (off or on)
- Also define relationships between switches
- Note, everything must be specified!
- If A and B are neighbours, it is not given that B and A are neighbours, need to specify!
- Set our goal, and off we go!

```
(:objects
        ;; switches
        switch_1 - switch
        switch_2 - switch
        switch_3 - switch
        switch_4 - switch
        switch_5 - switch
    )
```

```
    (:init
        (off switch_1)
        (on switch_2)
        (off switch_3)
        (on switch_4)
        (off switch_5)
        ;; neighbours
        (neighbours switch_1 switch_2)
        (neighbours switch_2 switch_1)
        (neighbours switch_2 switch_3)
        (neighbours switch_3 switch_2)
        (neighbours switch_3 switch_4)
        (neighbours switch_4 switch_3)
        (neighbours switch_4 switch_5)
        (neighbours switch_5 switch_4)
    )
```

Planning Resources

- A lot of information at the Planning Wiki
  - https://planning.wiki/
- Also a textbook
  - An Introduction to the Planning Domain Definition Language. Haslum, P., Lipovetzky, N., Magazzeni, D.,
  - Chapter 2 available on MyPlace

Summary

- Small Introduction to planning
  - Defining the world using logic
  - Using search to search through states to identify goal
- PDDL
  - Language to define planning problems
  - This week
- Next Topic
  - No more new topics!

Thanks for listening!