Principles of Computer Game Design and Implementation

Lecture 25

We already learned

- Decision trees
- Finite state machines
- Behaviour trees

Outline for today

- planning

Combining Actions

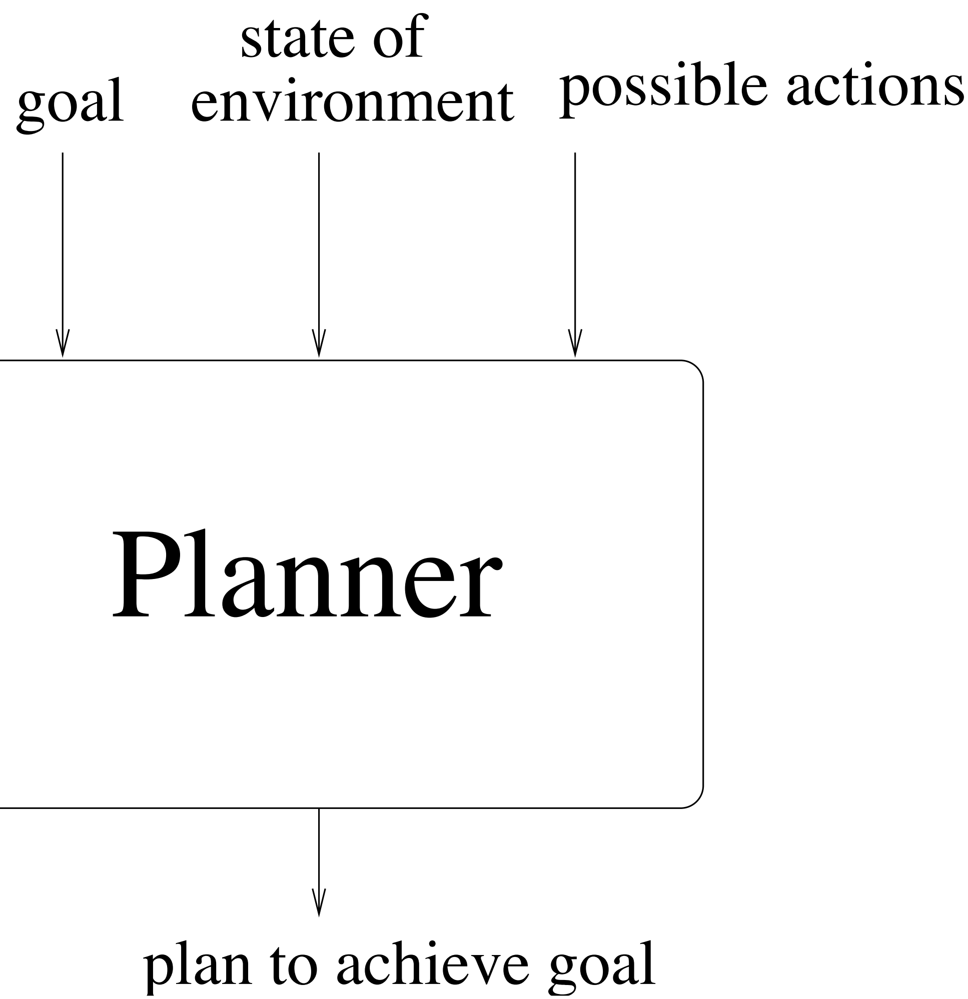
- In previous lectures, behaviour of game entities was ***defined*** by the AI developer
- Behaviour trees can be seen as reactive plans
  - React to changes in the environment
  - Options are prescribed

- In traditional AI, computer is asked to ***find*** sequences of actions

AI Planning

- Planning in AI is the problem of finding a sequence of primitive actions to achieve some goal.
- The sequence of actions is the system's plan which then can be **executed**.
- Planning requires the following:
  - representation of goal to achieve;
  - knowledge about what actions can be performed; and
  - knowledge about state of the world.

Architecture of a Planner



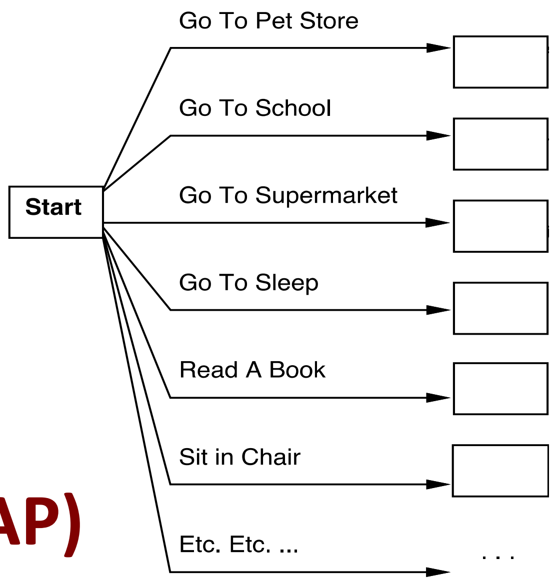
Planning in Games

- A character may have one or more goal (motives)
- Every goal has insistence – a number
- Actions fulfil goals (to some extent)

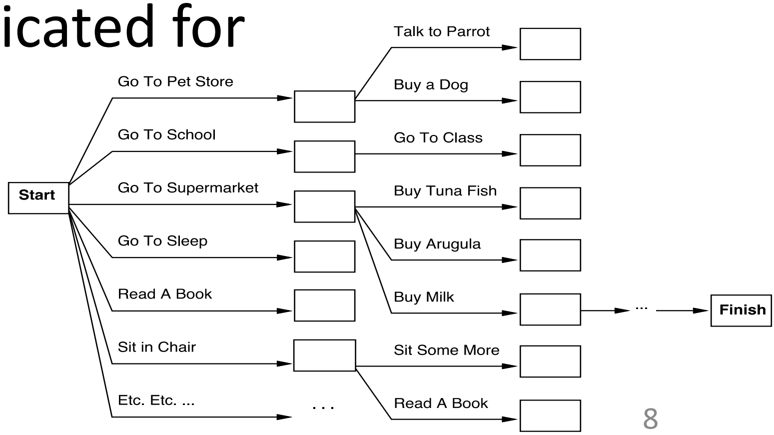
- *Actions can be combined into a PLAN*

GOB vs GOAP

- **Goal-oriented behaviour (GOB)**
- **Main problem: selecting an action**
  - Restricts design decisions



- **Goal-oriented action planning (GOAP)**
- **Main problem: finding a sequence of actions**
  - Often considered to be too complicated for games
    - But F.E.A.R. !



GOB: Simple Selection

- Goals:
  - Eat = 4;   Sleep = 3
- Actions:
  - Get-Raw-Food (Eat - 3)
  - Get-Snack (Eat – 2)
  - Sleep-in-Bed (Sleep – 4)
  - Sleep-on-Sofa (Sleep – 2)

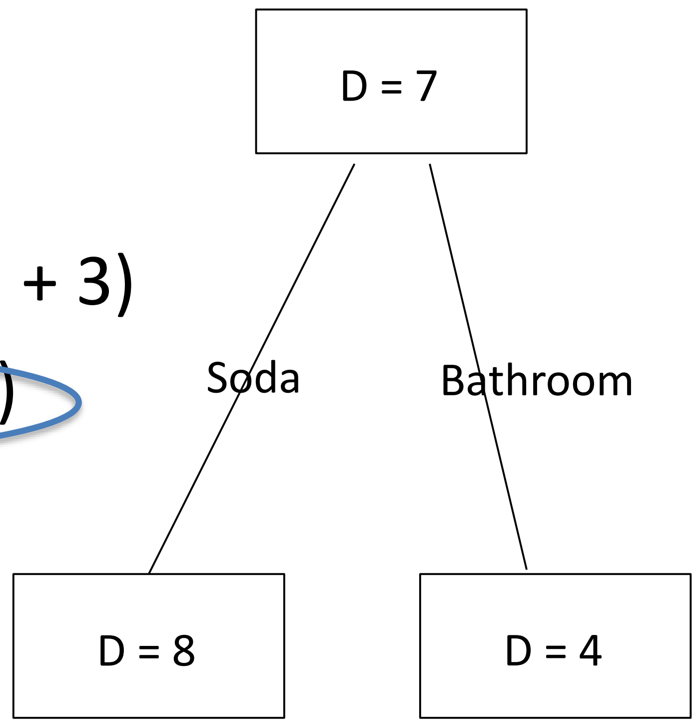
- Choose the most pressing goal;
- Find an action that most fulfils it

Works reasonably well when actions do not have side effects

GOB: Overall Utility

$$\textbf{Discontentment} = \sum_{\text{goals}} \text{insistence}$$

- Goals:
  - Eat = 4; Bathroom = 3
- Actions:
  - Drink-Soda (Eat – 2; Bathroom + 3)
  - Visit-Bathroom (Bathroom – 4)

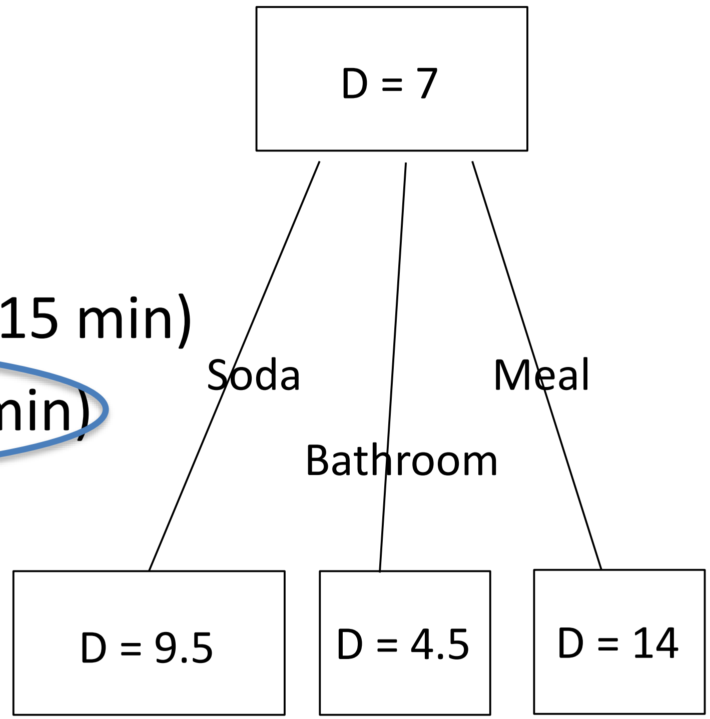


Works well when actions dependency is limited

Overall Utility: Discontentment + Timing

$$\text{Discontentment} = \sum_{\text{goals}} \text{insistence}$$

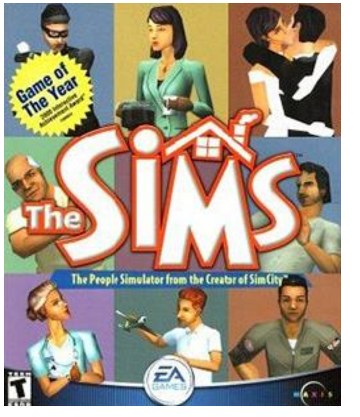
- Goals:
  - Eat = 4 + 4 per hour;
  - Bathroom = 3 + 2 per hour
- Actions:
  - Drink-Soda (Eat – 2; Bathroom + 3; 15 min)
  - Visit-Bathroom (Bathroom – 4; 15 min)
  - Cook-meal (Eat – 5; 2h)



Works well when actions dependency is limited

Actions Available

- Actions defined centrally are too inflexible
- **Smart object** insert actions into AI entities
  - Oven offers a `cook` action
  - Meat offers an `eat` action
    - But how to locate such objects?



- **"Smelly GOB"**
  - Actions smell with the goal it achieves
    - `cook` smells of Eat
  - Smells spread
  - Agents follow smell towards greatest concentration

Where GOB Fails

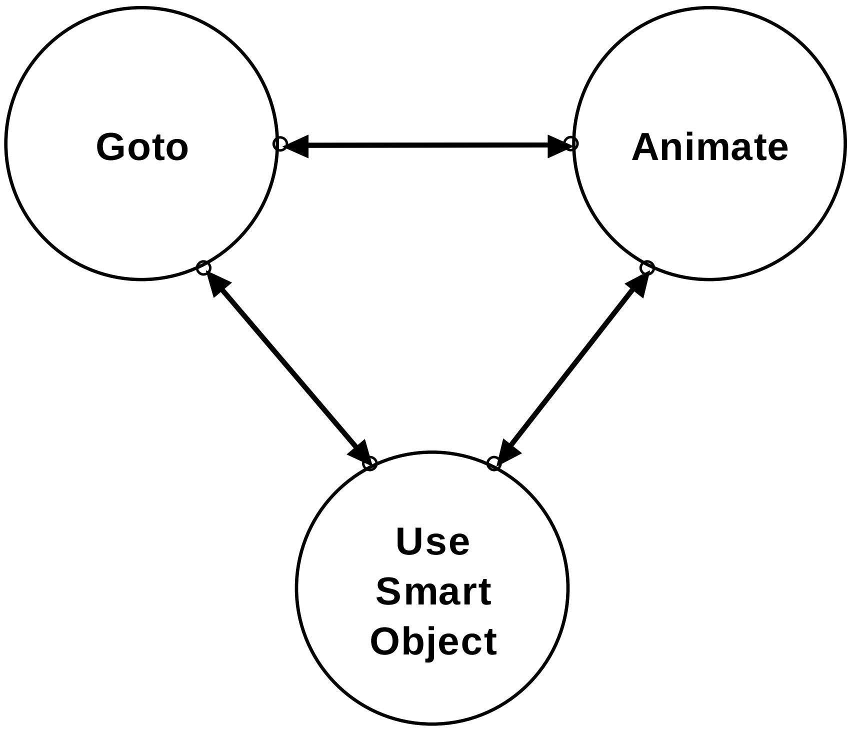
Energy level = 5

- Goals:
  - Heal = 4; Kill-Ogre = 3
- Actions:
  - Fireball (Kill-Ogre – 2); 3 Energy slots
  - Lesser-Healing (Heal – 2); 2 Energy slots
  - Greater-Healing (Heal – 4); 3 Energy slots

Does not work due to one action prohibiting another!

Planning in Games

- AI Behaviour
  - FSM used in F.E.A.R.



But goto where???

Use what???

F.E.A.R. uses planning to answer these questions

Jeff Orkin. GDC’06 “Three States and a Plan: The A.I. of F.E.A.R.”

Planning in F.E.A.R.

Design principle:

– Create interesting spaces for combat and let the AI act

AI Agents:

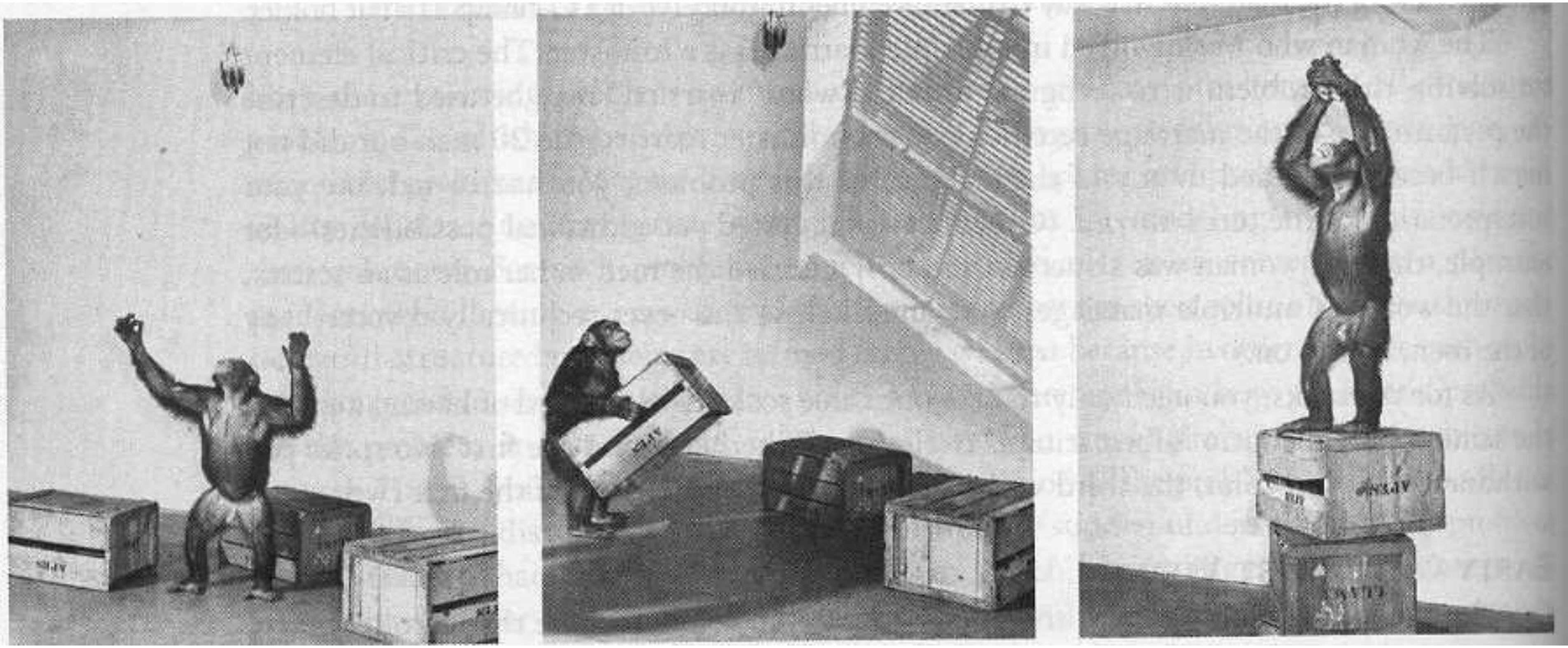
- Dodge
- Take cover
- …
- Dodge roll
- Ambush

Jeff Orkin. GDC’06 “Three States and a Plan: The A.I. of F.E.A.R.”

STRIPS Planning Language

- **ST**anford **R**esearch **I**nstitute **P**roblems **S**olver
- Uses predicate logic language to represent
  - **state** of environment;
  - **goal** to be achieved;
  - **actions** available to agents.

Example: Monkey, Box and Banana



A monkey is at the door into a room. A banana hangs from the ceiling in the middle of the room. The monkey wants the banana, but is not tall enough to get it. There is a box at the window which the monkey can climb on to get at the banana.

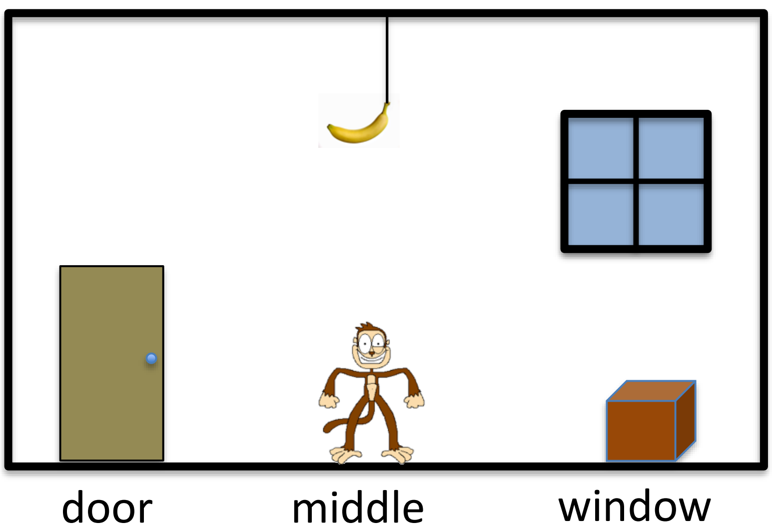
First-Order Predicates

- States can be described using:
  - MonkeyAt(x) Monkey is at location x
  - BoxAt(x) Box is at location x
  - BananaAt(x) Banana is at location x
  - StandsOn(x) Monkey stands on x
  - hasBanana True if Monkey has Banana

0-ary predicate
(proposition)

State Description

- State is a **conjunction** of **ground** and **function-free** atoms
- MonkeyAt(middle) $\wedge$ BoxAt(window) $\wedge$ BananaAt(middle) $\wedge$ StandsOn(floor)



Closed world assumption:

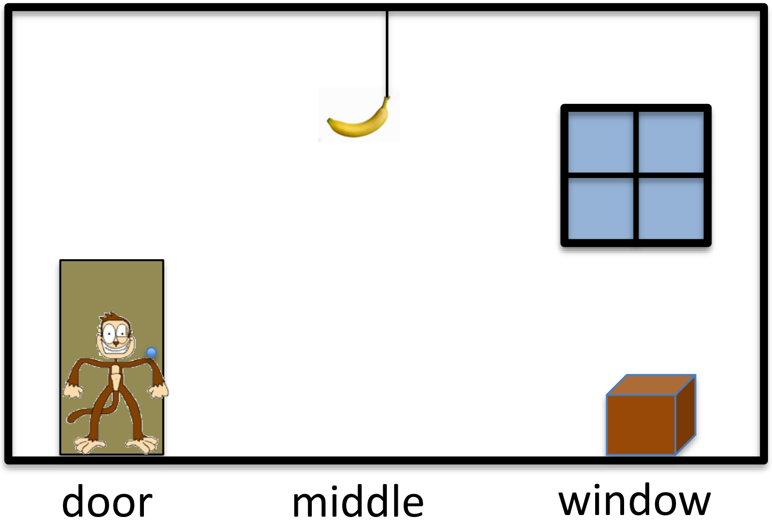
¬hasBanana

¬MonkeyAt(window), …

not stated – not true

Initial State

- State in which planning starts
- MonkeyAt(door) $\wedge$ BoxAt(window) $\wedge$ BananaAt(middle) $\wedge$ StandsOn(floor)



Goal State

- Goal is a particular state:

$$\text{hasBanana}$$

- A state S ***satisfies*** goal G if S contains all atoms from G (and possibly more)

hasBanana $\wedge$ MonkeyAt(door)

hasBanana $\wedge$ MonkeyAt(middle) $\wedge$ BoxAt(middle)

hasBanana $\wedge$ MonkeyAt(middle) $\wedge$ StandsOn(box)

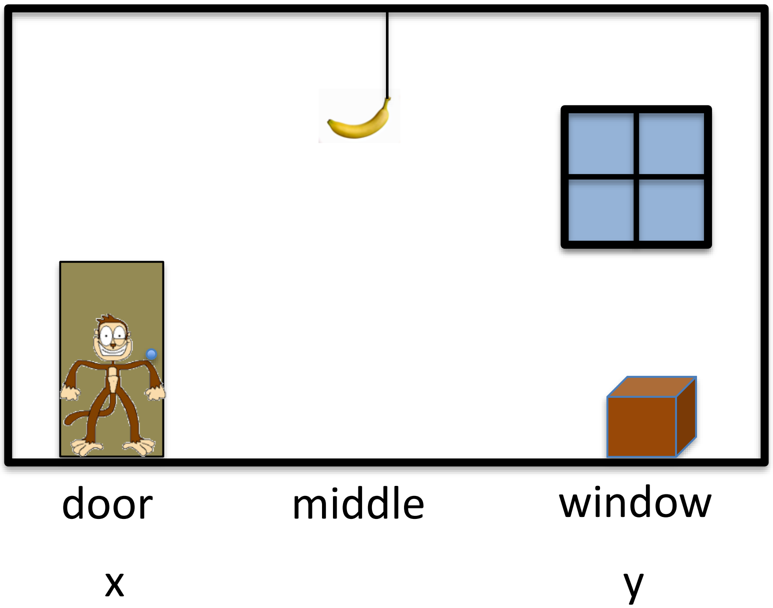
- All satisfy the goal

Actions

- Each action has
  - a *name*: which may have arguments;
  - a *pre-condition list*: list of facts which must be true for action to be executed;
  - a *delete list*: list of facts that are no longer true after action is performed;
  - an *add list*: list of facts made true by executing the action.
- Each of these may contain *variables*.

Example: Walk

- `Walk(x,y):`
  - pre: `MonkeyAt(x)`
  - del: `MonkeyAt(x)`
  - add: `MonkeyAt(y)`



- Action instantiation: `Walk(door,window)`
  - `x = door`
  - `y = window`

Example: Other Actions

- `ClimbUp(x)`
  - pre: `MonkeyAt(x), BoxAt(x), BananaAt(x), StandsOn(floor)`
  - del: `StandsOn(floor)`
  - add: `StandsOn(box)`
- `MoveBox(x, y)`
  - pre: `MonkeyAt(x), BoxAt(x)`
  - del: `MonkeyAt(x), BoxAt(x)`
  - add: `MonkeyAt(y), BoxAt(y)`
- `TakeBanana(x)`
  - pre: `MonkeyAt(x), BoxAt(x), BananaAt(x), StandsOn(box)`
  - del: -
  - add: `hasBanana`

Instead of $\wedge$

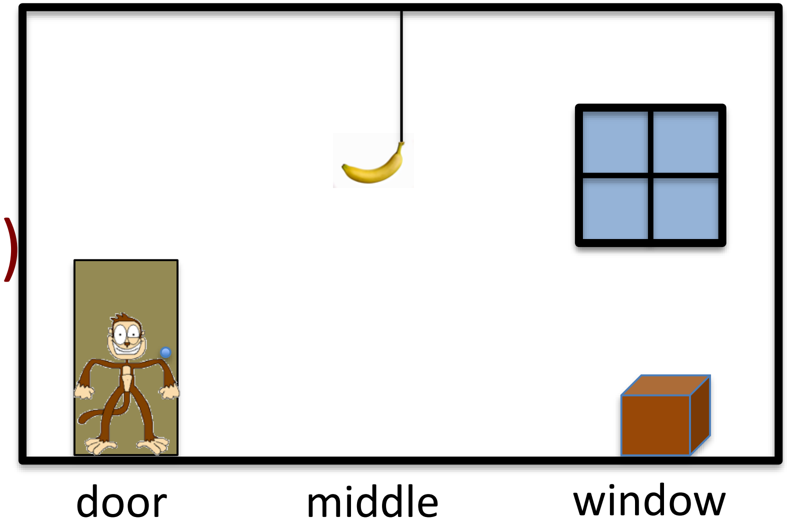
Action Effect

- The result of executing action A in state S is a state S’ such that
- S’ is identical to S except
  - Any atom from the *add list* of A is added to S’
  - Any atom from the *delete list* of A is deleted from S’
  - All other atoms do not change their value!

Frame condition

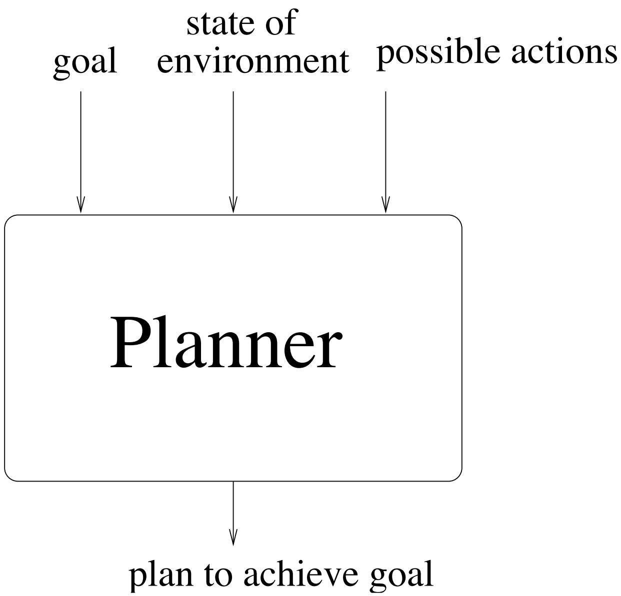
STRIPS Plan

- A sequence (list) of actions with variables replaced with values
  - Move(door,window)
  - MoveBox(window, middle)
  - ClimbUp(middle)
  - TakeBanana(middles)



Planning Algorithm

- There are numerous approaches to planning
  - Progressive/regressive planning
  - Partial planning
  - *Graphplan*
  - Reduction to sat
  - …
    - There is a planner competition

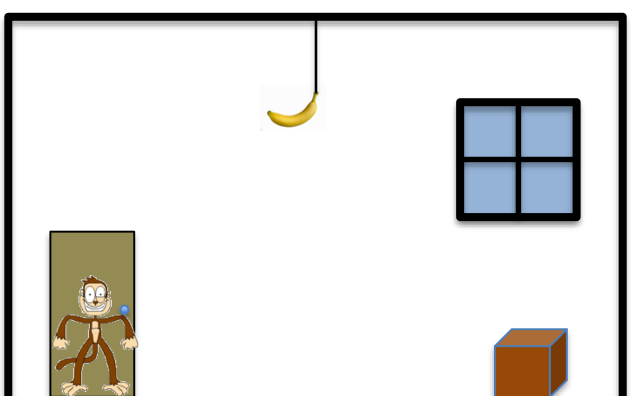


Planning in F.E.A.R. (1)

- States represented as arrays
  - One value per predicate

`[door, window, middle, floor, false]`

MonkeyAt	BoxAt	BananaAt	StandsOn	hasBanana
enum	enum	enum	enum	Boolean



Goal:

`[_, _, _, _, true]`

Planning in F.E.A.R. (2)

- Procedural `pre, add` and `del`

- E.g.
  - `Walk(x,y):`

[door, window, middle, floor, false]

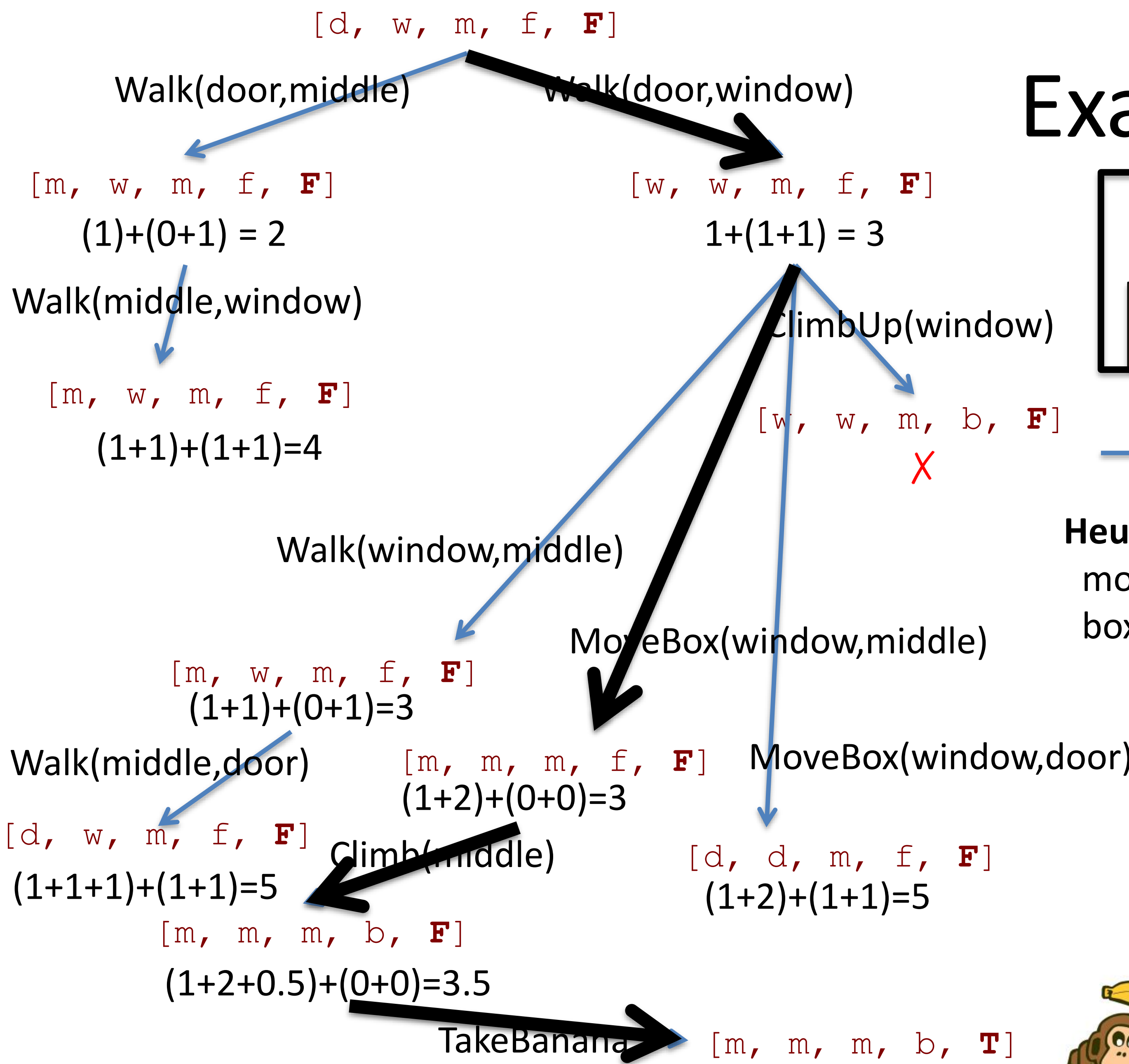
```
if (state[0]== x) {
        state[0] = y;
}
```

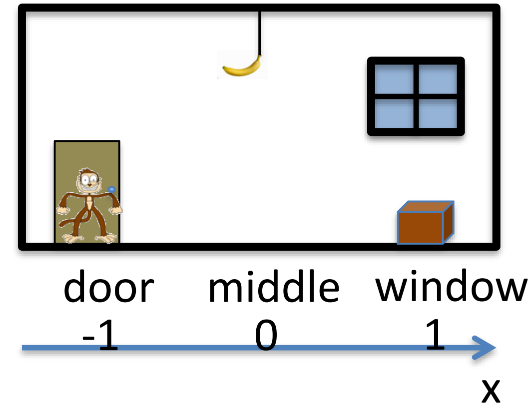
Planning in F.E.A.R. (3)

- Assign costs to actions
  - `Walk` costs 1
  - `MoveBox` costs 2
  - `ClimbUp` costs 0.5
  - `TakeBanana` costs 0.1

- Use A* search algorithm to find a plan
  - Heuristic needed

Example





Heuristic:
monkey-middle distance +
box-middle distance

Planning in Games

- Quite an effort even with A*
- Most time spent on *pathfinding*
  - *Where* to go rather than what goal to pursue
    - Will address the pathfinding problem next

- Hierarchical plans:
  - In order to carry out a higher-level plan, the planner must first refine the plan in order to produce a complete plan in terms of ground-level operations.

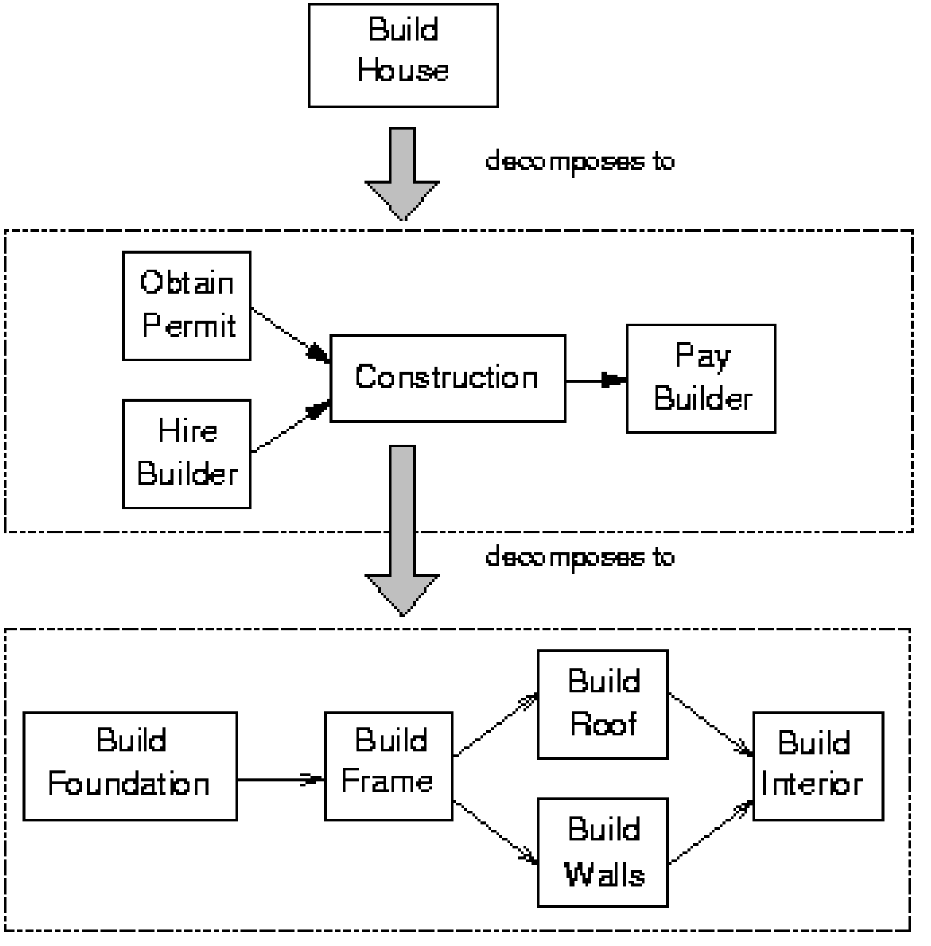
Hierarchical Task Network (HTN)

- use **abstract operators** to **incrementally** decompose a planning problem from a **high-level goal** statement to a **primitive plan network**
- **Primitive operators** represent actions that are **executable**, and can appear in the final plan
- **Non-primitive operators** represent **goals** (equivalently, **abstract actions**) that require further decomposition to be executed

HTN operator: Example

```
OPERATOR decompose
PURPOSE: Construction
CONSTRAINTS:
    Length (Frame) <= Length (Foundation),
    Strength (Foundation) > Wt(Frame) + Wt(Roof)
        + Wt(Walls) + Wt(Interior) + Wt(Contents)
PLOT: Build (Foundation)
   Build (Frame)
      PARALLEL
         Build (Roof)
            Build (Walls)
      END PARALLEL
      Build (Interior)
```

HTN planning: Example



Some Games Using GOAP Architectures

- F.E.A.R. 2005
- Condemned: Criminal Origins 2005
- S.T.A.L.K.E.R.: Shadow of Chernobyl 2007
- Ghostbusters 2008
- Silent Hill: Homecoming 2008
- Fallout 3 2008
- Empire: Total War 2009
- F.E.A.R. 2: Project Origin 2009
- Demigod, 2009
- Just Cause 2 2010
- Transformers: War for Cybertron 2010

http://web.media.mit.edu/~jorkin/goap.html