Human-like artificial creatures

2. Reactive planning

Cyril Brom

Faculty of Mathematics and Physics

Charles University in Prague

brom@ksvi.mff.cuni.cz

Outline

1. Recapitulation
   - action selection problem, artificial mind, architecture of a virtual being
2. Reactive planning
3. If-then rules
   - simple reactive planning
   - simple hierarchical reactive planning
   - PyPOSH example
4. Finite state machines
   - basic
   - hierarchical
   - Probabilistic
   - Quake3 hFSM example
5. Conclusion

If-then rules

if p then A

a precondition, an antecedent

an action, an effect, a consequent...

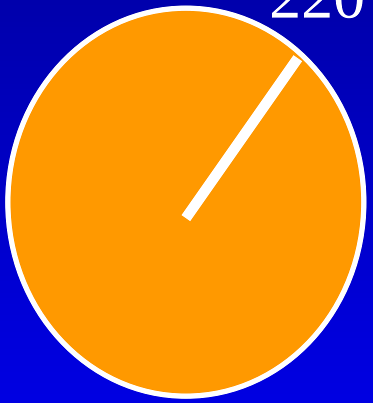
If-then rules

- A rule **fires** if its condition holds
- A reactive plan consist of tens of if-then rules
- All rules are "evaluated at once"
  - think in parallel!
- Technically, the parallelism must be "transformed" to a serial program.

A thermostat

The regulator is set on 220^{o}C:

1. IF temperature > 225^{o}C,
   THEN switch the heater off.
2. IF temperature < 215^{o}C,
   THEN switch the heater on.



Why is the temperature tested for 225 / 215 instead of 220?

What to do when more rules fires in the same instant?

Simple reactive planning

- Assign a priority to each rule:

A robot picking up mushrooms:

```
# When starts: not at home && be in picking state
1. if see_obstacle then change_direction
2. if basketful_of_m. and picking then stop_picking
3. if see_mush. and picking then pick_up_the_mush.
4. if midday and picking then stop_picking
5. if home then END
6. if picking then move_random
7. if not_picking then move_home
```

What does the robot do when it sees a mushroom, but it is returning home?

subsumption architecture:
[Brooks, 1986; Wooldridge, 2002]

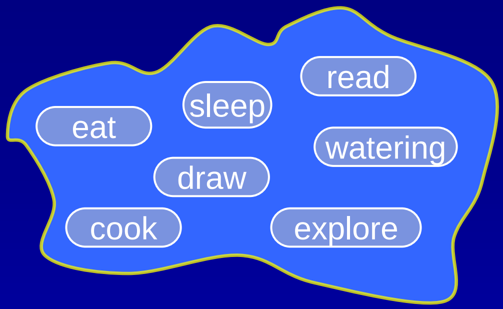
Simple hierarchical reactive planning

1. **if** bla1 **and** bla2 **then** SubGoal1
2. **if not** bla1 **and** bla3 **then** SubGoal2
3. **if** bla4 **then** SubGoal3
   - **3.1 if** A **then** Sub^{2}GoalA
   - **3.2 if** B **then** Sub^{2}GoalB
   - **3.3 if** C **then** Sub^{2}GoalC
     - .
     - .
     - .
   - **3.4 if** D **then** Sub^{2}GoalD
4. **if not** bla3 **and** bla2 **then** SubGoal4
5. **if** bla1 **and** bla3 and bla8 **then** SubGoal5
6. **if** blabla **then** SubGoal6
7. **if** bla2 **or (** bla3 **and not** bla7 **) then** SubGoal7

- Think hierarchically!

[Bryson, 2001; Nilsson, 1994; etc.]

Simple hierarchical reactive planning



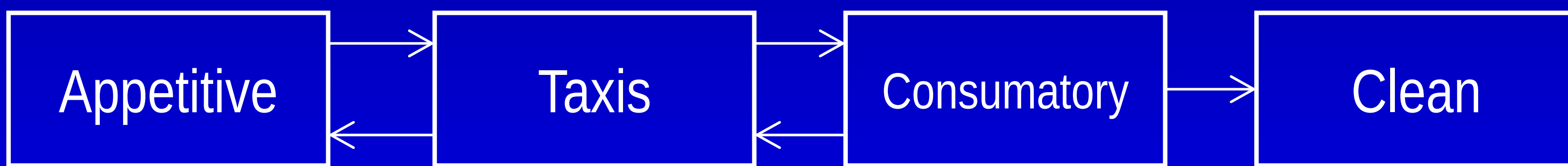
- Behaviour is decomposed hierarchically
  - top-level goals, sub-goals, tasks, atomic actions
- Every reactive plan is expressed by means of a set of trees
- Every root of a tree corresponds to a top-level goal
  - AND trees, AND-OR trees
- How to create a decomposition?

Simple hierarchical reactive planning

(a hierarchical top-down decomposition)

Watering:

goal: the garden is watered



- Find & take a can
- Fill the can

- Go next to a dry bed

- Water the bed

- Empty the can
- Put down the can

...cycles are possible!

...an ethology model

Simple hierarchical reactive planning

a decomposition example (watering)

- the highest priority has the goal condition, the second highest is the cleaning
- order the task in the normal/the reverse order [Bryson, 2001]

```
1. if garden_watered and cleaned then COMMIT
2. if garden_watered then subGoal_Clean
```

Clean

```
3. if not_hold_any_can then subGoal_FindTakeCan
4. if can_in_hands and empty then subGoal_FillUpTheCan
```

App.

```
5. if know_about_dry_bed & not_stand_nextTo_theBed
   then subGoal_GoThere
```

Taxi

```
6. if stand_nextTo_theBed and theBad_dry then
   atomicWatering
```

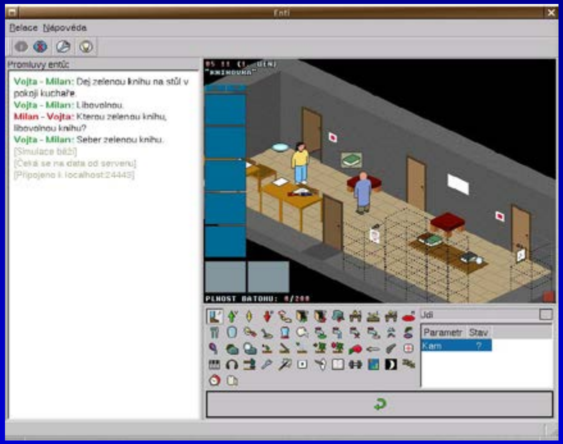
Cons.

Simple hierarchical reactive planning
top-level goals

- How to select a top-level goal to perform?
  - a schedule + interrupts
  - drives + interrupts
  - a drama manager (Façade)
  - planning and future-directed intentions (BDI)

ENTs

an example

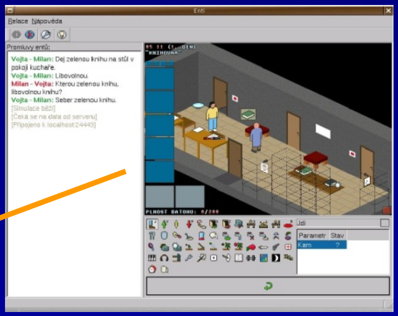


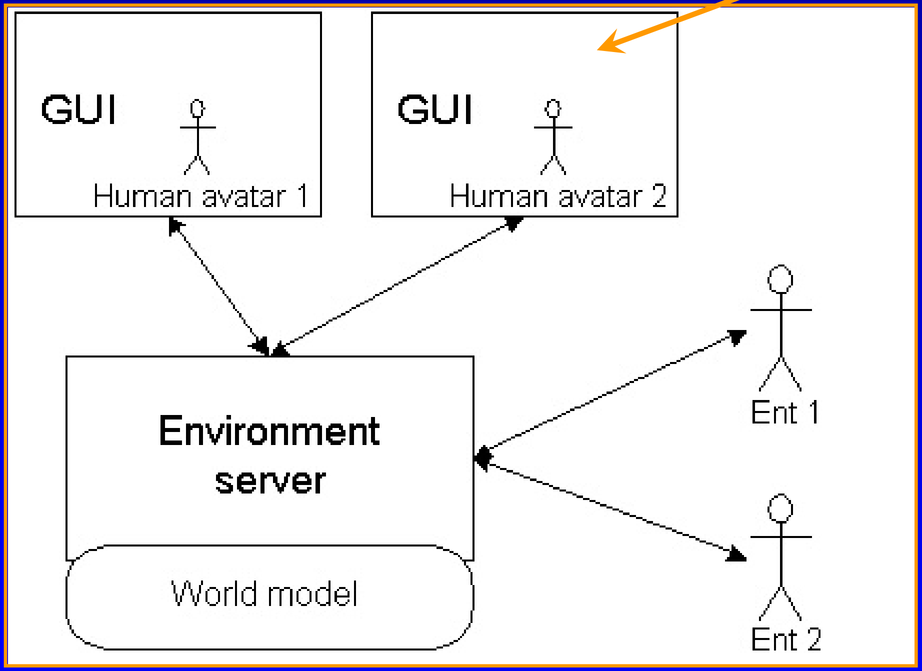
[Bojar et al., 2002; 2005]

- Chess-like topology, 2½ D world
- Discrete time (time-steps)
  - a step = 20 sec.
- Embodied
- 20 internal drives
  - hunger, thirst,...
- 60 atomic actions
  - aWalk, aPickUp, aWater, aEat,...
- Two hands + an inventory
- Face no particular direction in the world
  - an illusion of orientation is caused by the GUI only
- Understand a simplified version of Czech language
- Driven by scripts in E language

ENTs

system architecture





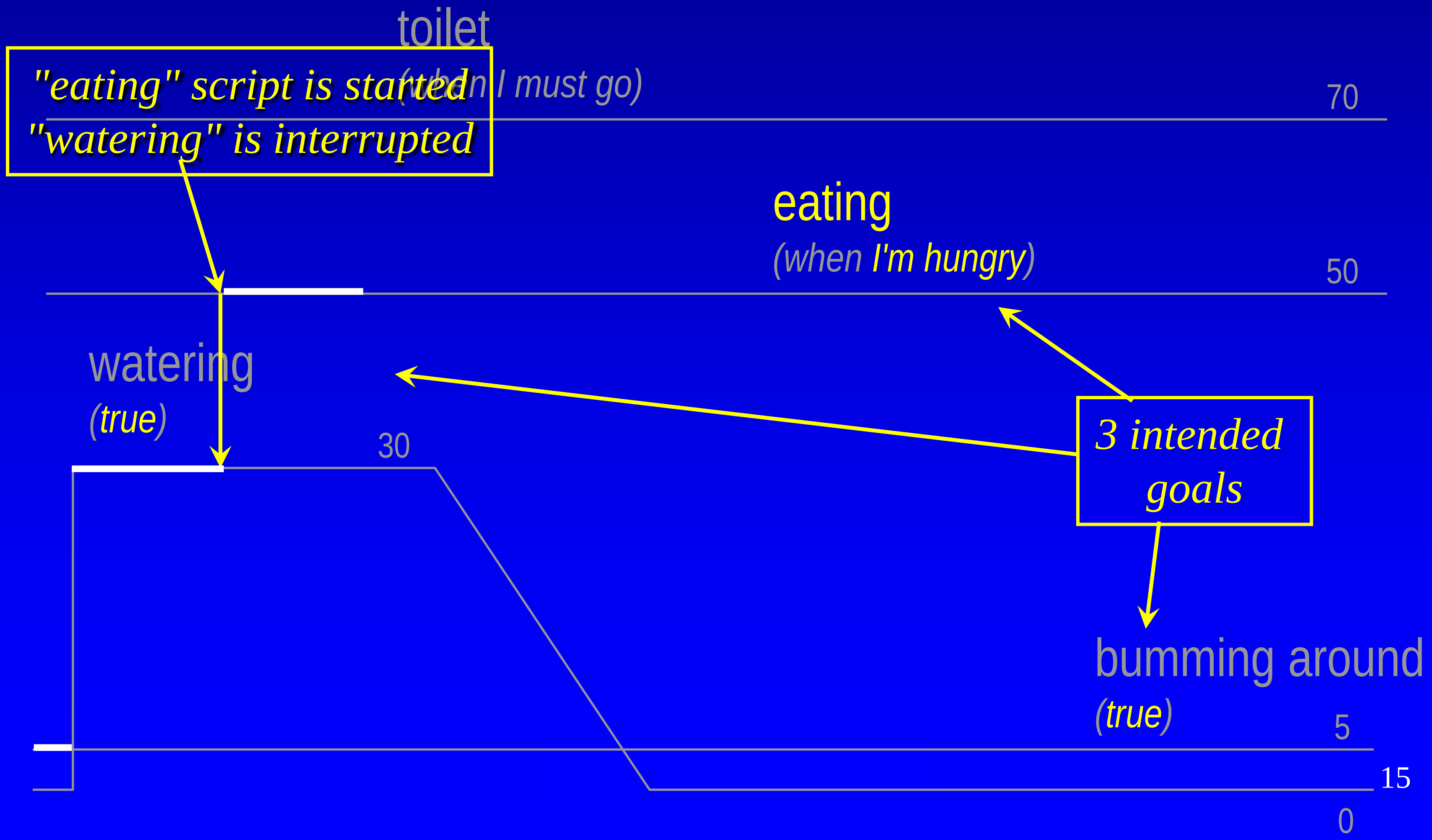
- 3 independent programs for Linux
  - entiserver (ES): the server of a virtual world
  - entiprohlizec: the graphical user interface
  - ent: the ent's control program (artificial mind)
- It is possible to instantiate different world models
  - we will use a model of a family house

Top-level goals

Four intended top-level goals of the gardener…

What is on the top?

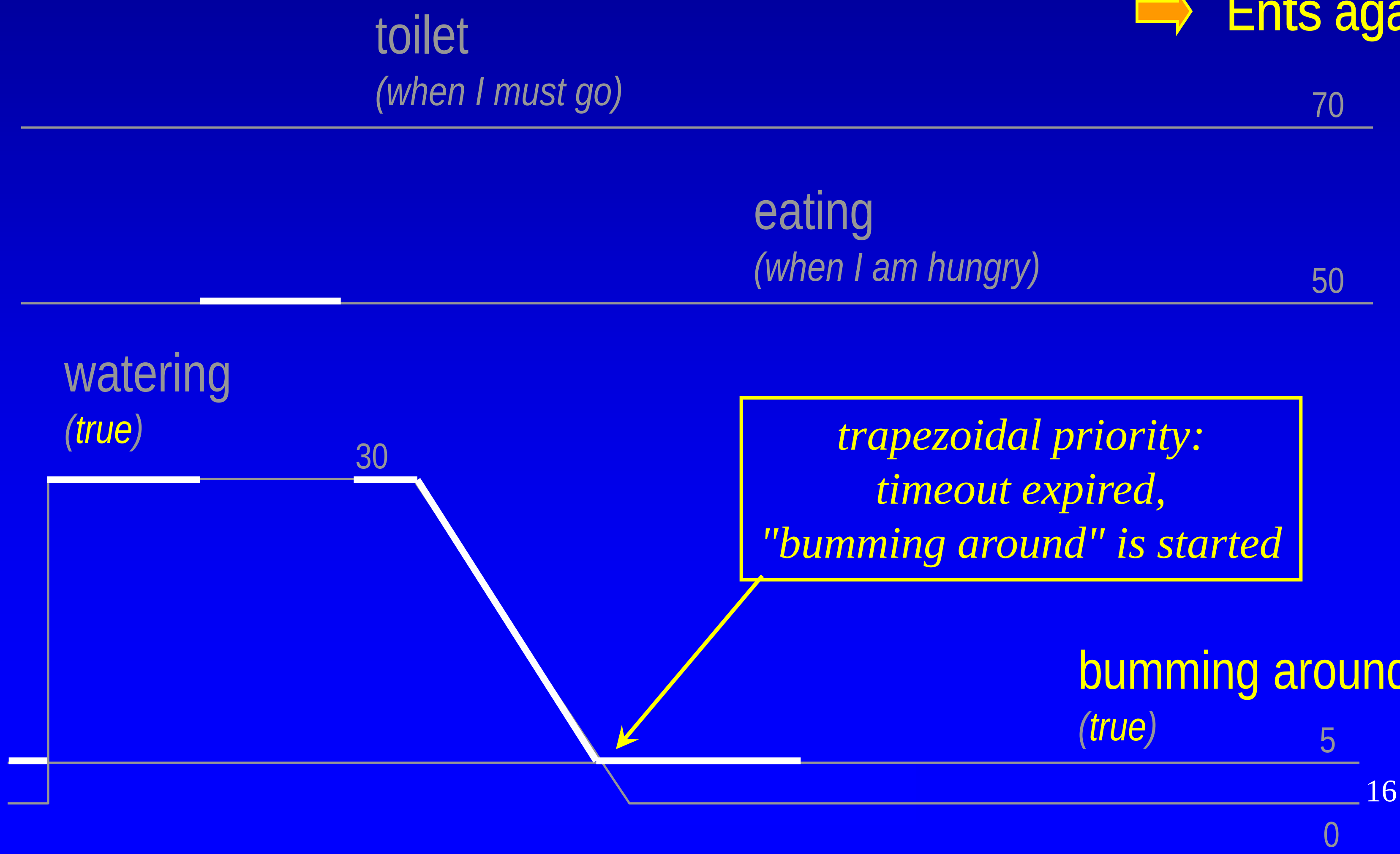
Three active goals



What is on the top?

Bumming around

Ents again

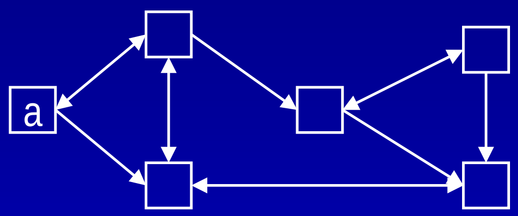


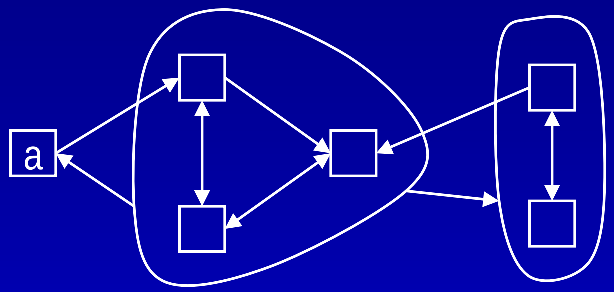
Outline

1. Recapitulation
   - action selection problem, artificial mind, architecture of a virtual being
2. Reactive planning
3. If-then rules
   - simple reactive planning
   - simple hierarchical reactive planning
   - PyPOSH example
4. Finite state machines
   - basic
   - hierarchical
   - Probabilistic
   - Quake3 hFSM example
5. Conclusion

Finite state machines

FSM & HFSM (1)





Standard "finite-state machine" (FSM) is a tuple:

< { <label, T, script> }, a >

- <label, T, script> is a *state*
  - a *label* is a name of the state
  - a *script* is a code associated with the state
  - *T* is a set of rules that trigger transition to another state (i.e. transition function)
- *a* is a currently active state

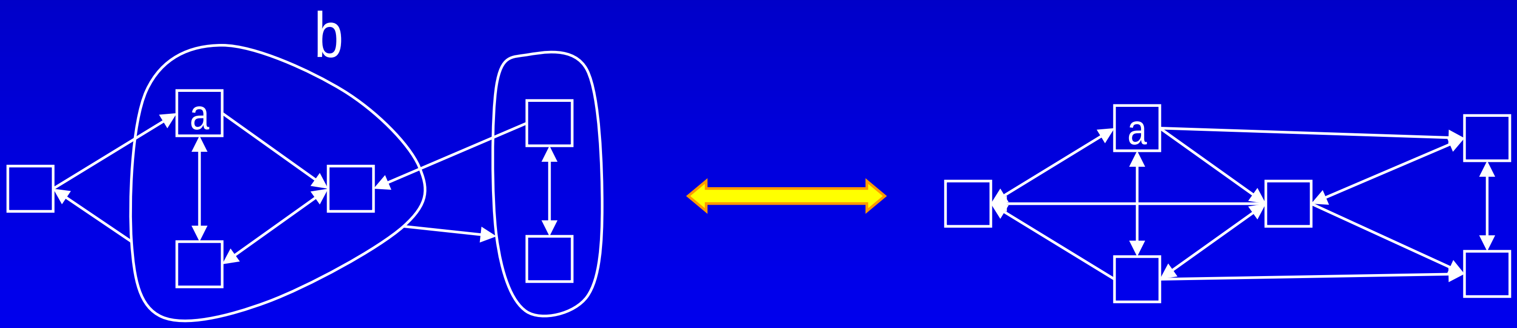
Hierarchical "finite-state machine" (HFSM) is a tuple:

< { <label, T, sc> }, A >

- <label, T, sc> is a *state*
  - a *label* is a name of the state
  - a *sc* is either a code associated with the state (i.e. a *script*), or a set of the names of the state's substates
  - *T* is a set of rules that trigger transition to another state (i.e. transition function)
- *A* is a set of currently active states
  - a path from a root-state to a leaf-state

FSM & HFSM (2)

- FSM and HFSM are computationally equivalent
  - HFSM avoids "spaghetti design"



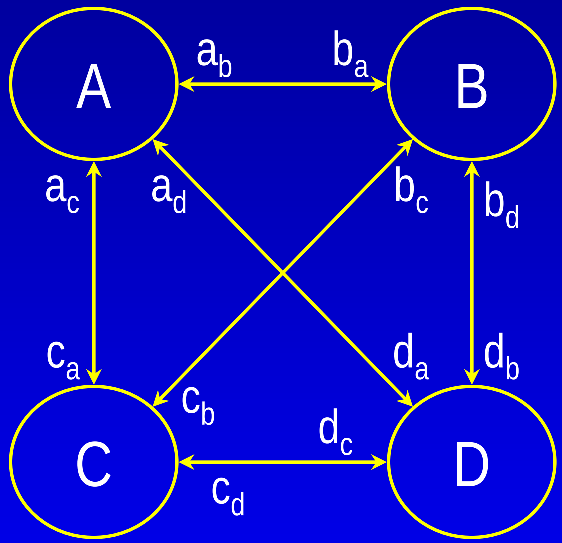
[Isla, 2005]
[Champandard, 2003]

Are finite state machines computationally equivalent to Touring machines?

SRP vs. FSM

1. if a_c or b_c or d_c then C
2. if a_b or c_b or d_b then B
3. if b_a or c_a or d_a then A
4. if a_d or b_d or c_d then D

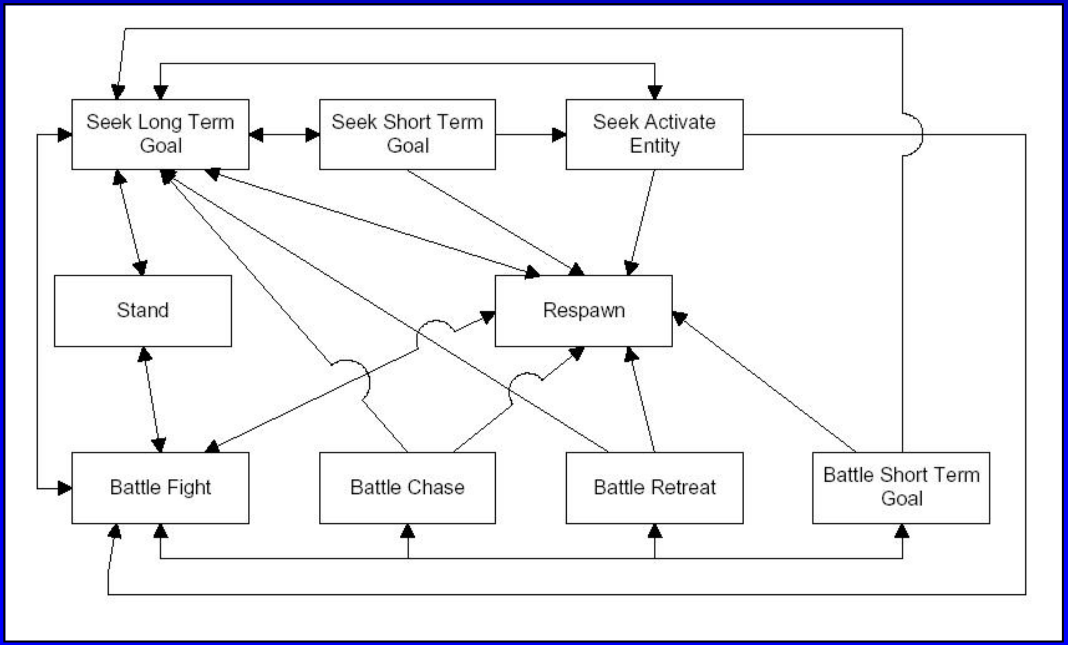
a note: z_x also tests whether the FSM is in state Z



- priorities
- "spaghetti design"

HFSM example
Quake bot

- High level decision control only
- In each FSM-node, a bot chooses among possible goals associated with the node
- Standard HFSM

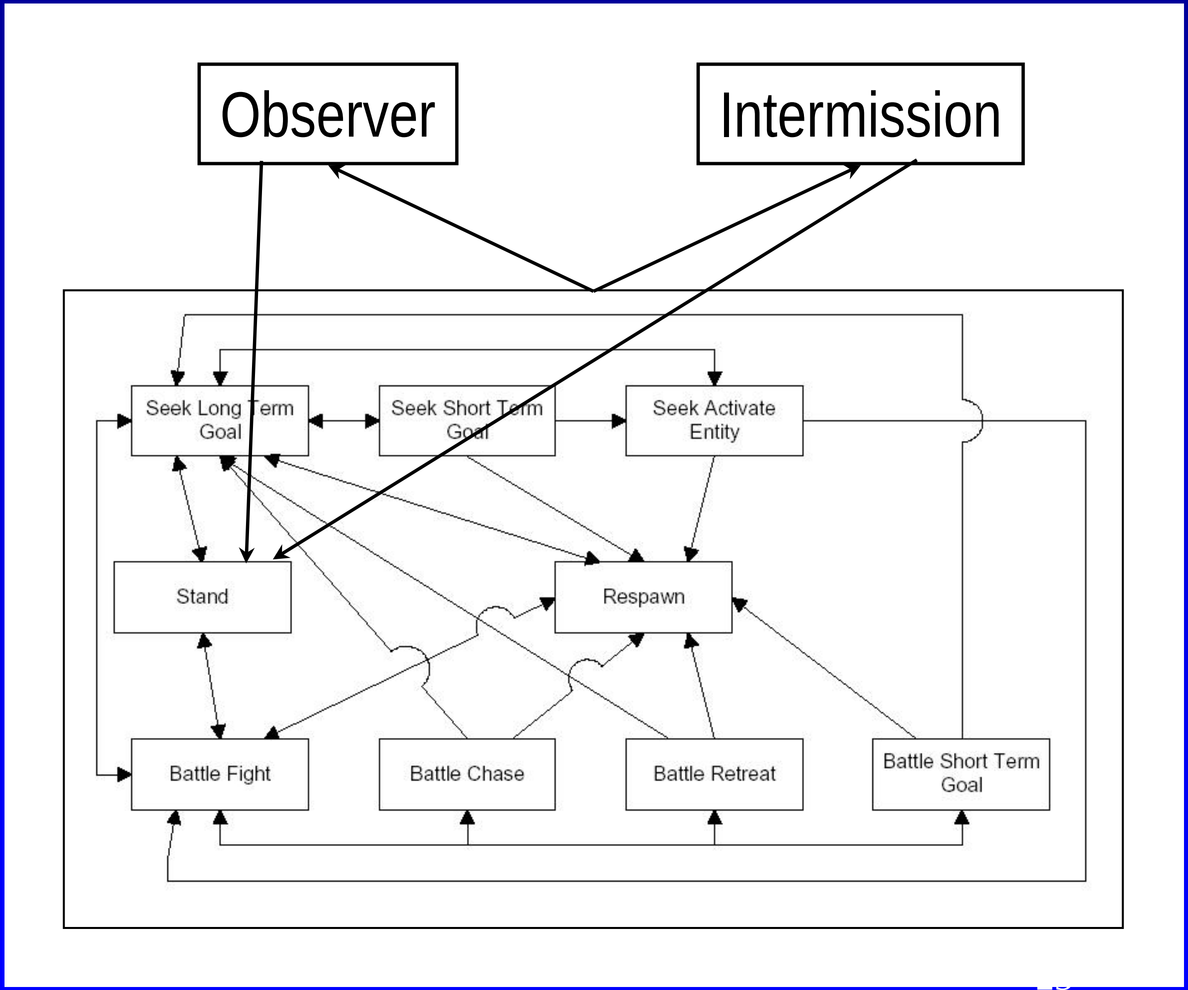


[van Waveren, 2001]

HFSM example

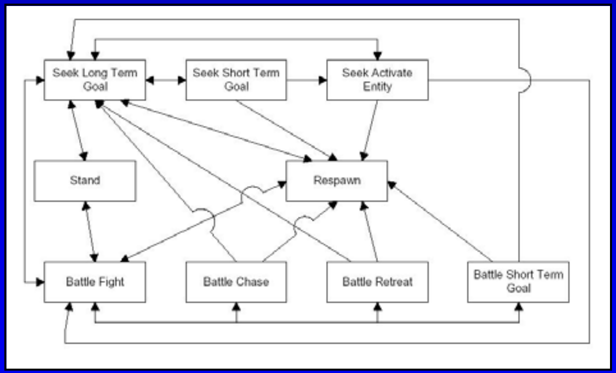
Quake bot

- High level decision control only
- In each FSM-node, a bot chooses among possible goals associated with the node
- Standard HFSM
- The if-then rules "in each node" are written in C



HFSM example
Quake bot

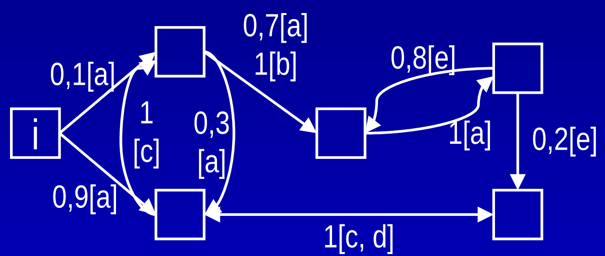
- In each FSM-node, a bot chooses among possible goals associated with the node
  - fuzzy decision (how much do I want to pick this weapon up?)
  - long term-goals vs. short term goals
- E.g. "battle fight":
  - acquiring enemy
  - selecting weapon
  - aiming and approaching
  - shooting
- Different techniques can be used in each node
  - low-level navigation
  - voting system
  - planning



van Waveren (c) 2001

Time (seconds)	Event or decision	Current AI node	Current goal
18.1	The bot named Grunt enters the game.	Stand	-
	Bot spawns.	Stand	-
		Seek LTG	
	Bot decides to retrieve item.	Seek LTG	Retrieve rocket launcher
	Bot decides to retrieve nearby item.	Seek LTG	Retrieve rocket launcher
		Seek NBG	Retrieve bullets
19.9	Picked up bullets.	Seek NBG	Retrieve bullets
		Seek LTG	Retrieve rocket launcher
20.6	Bot decides to retrieve nearby item.	Seek LTG	Retrieve rocket launcher
		Seek NBG	Retrieve shotgun
21.5	Enemy in sight.	Seek NBG	Retrieve shotgun
		Battle NBG	Kill the enemy & retrieve shotgun.
22.7	Picked up shotgun & bot wants to retreat.	Battle NBG	Kill the enemy & retrieve shotgun.
		Battle Retreat	Retreat & retrieve rocket launcher.
23.8	Bot decides to retrieve nearby item.	Battle Retreat	Retreat & retrieve rocket launcher.
		Battle NBG	Retrieve armor shard.
25.5	Picked up armor shard.	Battle NBG	Retrieve armor shard.
	Enemy out of sight & bot decides to chase.	Battle Retreat	Retreat & retrieve rocket launcher.
		Battle Chase	Chase enemy.
25.9	Bot decides to retrieve nearby item.	Battle Chase	Chase enemy.
		Battle NBG	Retrieve armor shard.
28.2	Picked up armor shard.	Battle NBG	Retrieve armor shard.
		Battle Chase	Chase enemy.
31.9	Enemy in sight.	Battle Chase	Chase enemy.
		Battle Fight	Kill the enemy.
32.3	Enemy out of sight.	Battle Fight	Kill the enemy.
		Battle Chase	Chase the enemy.
33.4	Enemy in sight.	Battle Chase	Chase the enemy.
		Battle Fight	Kill the enemy.
33.5	Enemy out of sight.	Battle Fight	Kill the enemy.
		Battle Chase	Chase the enemy.
35.4	Enemy in sight.	Battle Chase	Chase the enemy.

Probabilistic FSM models



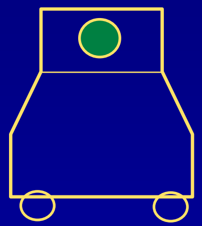
- Probabilistic "finite-state machine" (PFSM) is a tuple:
  < { <label, T^p, script> }, a >
- <label, T^p, script> is a *state*
  - a *label* is a name of the state
  - a *script* is a code associated with the state
  - T^p is a set of rules that trigger a transition to another state with a given probability
- *a* is the currently active state

Reactive planning - recapitulation

Recapitulation

- Reactive planning is a bunch of methods of driving behaviour of virtual beings
- Each method determines the next action in every instant in "a timely fashion"
- SHRP
  - if-then rules
  - priorities
  - AND-OR trees
- FSM
  - states
  - transitions

Implementation



rationale:

- Special-purpose languages:
  - rules
    - JAM [Hubber, 1999]
    - E [Bojar et al., 2002]
    - PyPOSH [Kwong, 2003]
    - ABL [Mateas, 2002]
    - (Soar)
  - FSM
    - AI. Implant...
    - Softimage

```
step
if someone-shoot-at-me do { .. }
if someone-asked-me do { .. }
if I-am-hungry do { .. }
if I-need-toilet do { .. }
if I-am-sleepy do { .. }
step
if someone-shoot-at-me do { .. }
if someone-asked-me do { .. }
if I-am-hungry do { .. }
if I-need-toilet do { .. }
if I-am-sleepy do { .. }
pick-up-mark
if someone-shoot-at-me do { .. }
if someone-asked-me do { .. }
if I-am-hungry do { .. }
if I-need-toilet do { .. }
if I-am-sleepy do { .. }
pick-up-mark
if someone-shoot-at-me do { .. }
...
```

Questions?

References

- BOD, POSH
  - Joanna Bryson. The Behavior-Oriented Design of Modular Agent Intelligence. In: *Proceedings of Agent Technologies, Infrastructures, Tools, and Applications for E-Services*, pages 61-79, Springer LNCS 2592, Berlin, Germany, 2003.
  - Kwong, A. *A Framework for Reactive Intelligence through Agile Component-Based Behaviours*. Master thesis, University of Bath (2003)
  - Joanna Bryson. *Intelligence by Design: Principles of Modularity and Coordination for Engineering Complex Adaptive Agents.* PhD thesis, Massachusetts Institute of Technology, 2001.
- Gamebots:
  - Adobbati, R., Marshall, A. N., Scholer, A., and Tejada, S.: Gamebots: A 3d virtual world test-bed for multi-agent research. In: Proceedings of the 2nd International Workshop on Infrastructure for Agents, MAS, and Scalable MAS, Canada (2001)
- ENTs
  - O. Bojar, C. Brom, M. Hladík, V. Toman: The Project ENTs: Towards Modeling Human-like Artificial Agents. In *SOFSEM 2005 Communications*, pages 111–122, Liptovský Ján, Slovak Republic, January 2005.
  - Project Ent homepage: `http://ckl.ms.mff.cuni.cz/~bojar/enti/`

References

- FSM
  - Waveren, J. M. P. van: The Quake III Arena Bot. Master thesis. Faculty ITS, University of Technology Delft (2001)
  - Champandard, A.J.: AI Game Development: Synthetic Creatures with learning and Reactive Behaviors. New Riders, USA (2003)
  - Softimage, Bahavior: http://www.softimage.com/products/behavior
- Façade, ABL
  - Mateas, M.: Interactive Drama, Art and Artificial Intelligence. Ph.D. Dissertation. Department of Computer Science, Carnegie Mellon University (2002)
- Other
  - Brooks, A. R.: Intelligence without reason. In: Proceedings of the 1991 International Joint Conference on Artificial Intelligence, Sydney (1991) 569-595
  - Huber, M. J.: JAM: A BDI-theoretic mobile agent architecture. In: Proceedings of the 3rd International Conference on Autonomous Agents (Agents'99). Seatle (1999) 236-243
  - Soar project: http://www.eecs.umich.edu/~soar/
  - Isla, D.: Handling Complexity in the Halo 2 AI. Game Developers Conference, GDC 2005, http://www.gamasutra.com/gdc2005/features/20050311/isla_01.shtml

References

- AI & agents
  - S. J. Russell and P. Norvig: *Artificial Intelligence: a Modern Approach*. Prentice-Hall, Englewood Cliffs, NJ.
  - M. Wooldridge: *An Introduction to MultiAgent Systems*. John Wiley & Sons, 1995
- Other
  - Brooks, A. R.: Intelligence without reason. In: *Proceedings of the 1991 International Joint Conference on Artificial Intelligence*, Sydney (1991) 569-595
  - Huber, M. J.: JAM: A BDI-theoretic mobile agent architecture. In: *Proceedings of the 3rd International Conference on Autonomous Agents* (Agents'99). Seatle (1999) 236-243

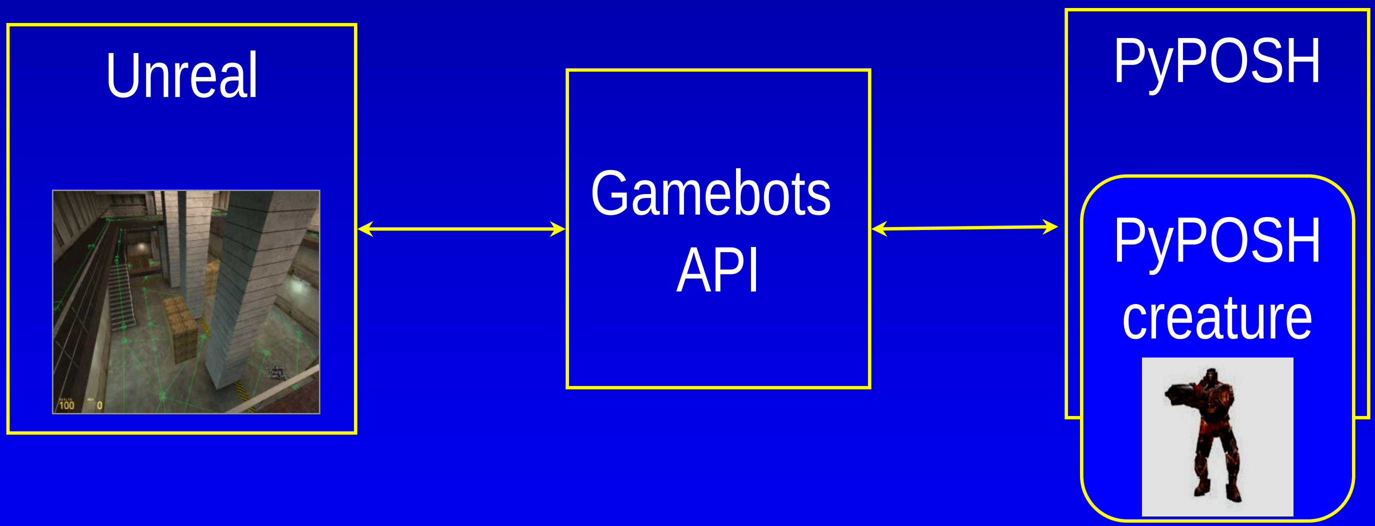
POSH

POSH & BOD

- Behavioural oriented design
  - behavioural decomposition
- POSH: Parallel-rooted, Ordered Slip-stack Hierarchical
  - a method that exploits hierarchical if-then rules
  - several languages
    - POSH: in lisp or C++
    - PyPOSH: Python implementation
    - jyPOSH: Jython implementation (interoperates with Java)

[Bryson et al., 2001 - 2006]

PyPOSH in Unreal - architecture

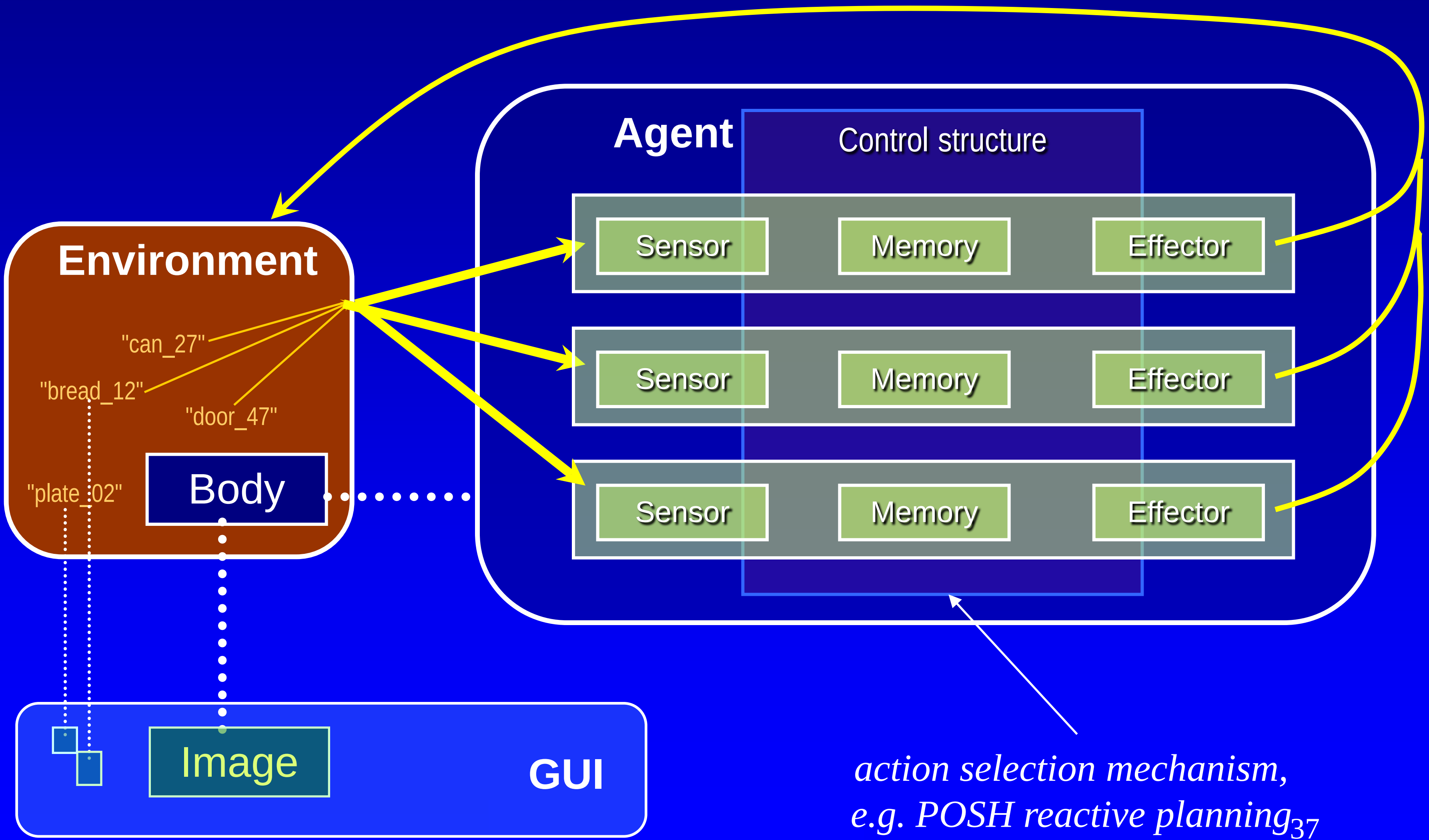


[IGN Entertainment, 1996-2006]

[Adobatti et al., 2000]

[Kwong, 2003]

Behavioural oriented creature



action selection mechanism, e.g. POSH reactive planning

Behaviours as objects

- Object
  - properties/variables
  - methods
- Behaviour
  - states/variables (memory)
  - primitive elements of the reactive plan which present the interface to the behaviour
    - senses
    - acts
  - learning

POSH - control structure I

- Action pattern
  - a sequence of actions
  - e.g., "baa" and look at it (sheep)
- A competence: { *s*; *s* is a competence step }
  - steps that can be performed in different orders (i.e., a set of sequences)
  - one of the steps can be a goal step
  - the competence returns a value: $\top$ if the goal is accomplished, $\bot$ if none of its steps fire
  - a competence step: <p, r, a, [n]>
    - a priority, a releaser, an action, a number of retries
    - the action can be another competence

[Bryson, 2001]

POSH - control structure II

- A drive collection: { d; d is a drive element }
  - the root of the hierarchy
  - a drive element: <p, r, a, A, [f]>
    - *p* – a priority
    - *r* – a releaser
    - *a* – a currently active element of the drive element (a sub-element)
    - *A* – the top element (i.e., a collection, action pattern, or an action) of the drive element → slip-stack
    - *f* – a maximum frequency at which this drive element is visited
      - e.g., jump every five seconds
  - for any cycle of the action selection, only the drive collection itself and at most one other POSH element will have their releasers examined
- One drive element can suspend temporarily another drive element
  - a competence step cannot interrupt another competence step
- When the suspending drive element terminates, the suspended drive element continues

PyPOSH

Python

```
def init_senses( self ):
    self.add_sense( "see-player", self.see_player )
    ...

def init_acts( self ):
    self.add_act( "move-player", self.move-player )
    ...

def see_player( self ):
    ...
```

"Lisp"

top-level → *checking period* → *prio: 1, 2, 3*

```
(RDC life (goal( (fail) ) )
    ( drives
        (( hit( trigger( * (hit-object)(is-rotating False) ) ) avoid ))
        (( follow( trigger( (see-player) ) ) follow-player ))
        (( wander( trigger( (succeed) ) ) wander-around ))
    ) )
```

timeout condition — *terminate condition*

```
(C wander-around (minutes 10) (goal( (see-player) ) )
    ( elements
        (( close-enough( trigger( (close-to-player) ) ) stop-bot ))
        (( move( trigger( (see-player) ) ) move-player ))
    ) )
```

if → *then*

PyPOSH

```
def init_senses( self ):
    self.add_sense( "see-player", self.see_player )
    ...

def init_acts( self ):
    self.add_act( "move-player", self.move-player )
    ...

def see_player( self ):
    ...


(RDC life (goal( (fail) ) )
    ( drives
        (( hit( trigger( (hit-object)(is-rotating False) ) ) avoid ))
        (( follow( trigger( (see-player) ) ) follow-player ))
        (( wander( trigger( (succeed) ) ) wander-around ))
    ) )

(C wander-around (minutes 10) (goal( (see-player) ) )
    ( elements
        (( close-enough( trigger( (close-to-player) ) ) stop-bot ))
        (( move( trigger( (see-player) ) ) move-player ))
    ) )
```

PyPOSH

```
def init_senses( self ):
    self.add_sense( "see-player", self.see_player )
    ...

def init_acts( self ):
    self.add_act( "move-player", self.move-player )
    ...

def see_player( self ):
    ...


(RDC life (goal( (fail) ) )
    ( drives
        (( hit( trigger( (hit-object)(is-rotating False) ) ) avoid ))
        (( follow( trigger( (see-player) ) ) follow-player ))
        (( wander( trigger( (succeed) ) ) wander-around ))
    ) )

(C wander-around (minutes 10) (goal( (see-player) ) )
    ( elements
        (( close-enough( trigger( (close-to-player) ) ) stop-bot ))
        (( move( trigger( (see-player) ) ) move-player ))
    ) )
```

PyPOSH

```
def init_senses( self ):
    self.add_sense( "see-player", self.see_player )
    ...

def init_acts( self ):
    self.add_act( "move-player", self.move-player )
    ...

def see_player( self ):
    ...


(RDC life (goal( (fail) ) )
    ( drives
        (( hit( trigger( (hit-object)(is-rotating False) ) ) avoid ))
        (( follow( trigger( (see-player) ) ) follow-player ))
        (( wander( trigger( (succeed) ) ) wander-around ))
    ) )

(C wander-around (minutes 10) (goal( (see-player) ) )
    ( elements
        (( close-enough( trigger( (close-to-player) ) ) stop-bot ))
        (( move( trigger( (see-player) ) ) move-player ))
    ) )
```