

**Class 2:  
Some Lisp; The Nature of  
Agents; Search & Game  
Playing**

Selmer Bringsjord

May 20, 1999

## Misc Stuff

- Reminder: Class pictorial roster, w/ email addresses & web sites...
- We are learning to assemble toolkit ... for **building** things
  - anchor learning with constant eye at (micro) construction
  - and games provide opportunities, e.g., GUESS WHO?
  - but also ponder your own domain, constantly

# Programming Paradigms

- Procedural
  - e.g., Turing machines, Register machines
  - and familiar languages (e.g., Pascal)
- Functional
  - “Pure” Lisp
- Declarative
  - only  $\approx$  Prolog
  - Theorem Provers (for us: OTTER)

## First Look @ Code

- NASA and immobots and Lisp<sub>j</sub>
- Response<sub>r</sub>
- Some Lisp
  - Ralph’s interrogation re. net worth
  - “virtual fences”
- On the declarative front: SECRETS  
in OTTER

## The Agent Approach

- What is agent, put simply?
  - Skinner would be proud
  - See the picture: 2.1
- Rationality vs. Omniscience: the case of the 747 door
- Definition of Rationality<sub>*h*</sub>

## **Rationality**

That which is rational at a given time depends on four things:

1. The performance measure that defines degrees of success.
2. The percept sequence.
3. What the agent knows about the environment
4. The actions the agent can perform.

## Definition of ideal rational agent:

**Ideal Rational Agent** For each possible percept sequence, such an agent does whatever action is expected to maximize its performance measure, on the basis of the evidenced provided by the percept sequence and whatever built-in knowledge the agent has.

- What do you think? Is this acceptable?

We're concentrating on the program:

- agent = architecture + **program**

# Intelligent Agents; Some Questions

- Why would an ideal rational agent need to have what R&N call ‘autonomy’?
- What would a PAGE description for
  - BRUTUS
  - SHERLOCK
    - \* SECRETS
    - \* GUESS WHO?
  - S<sup>3</sup>G BOT

look like?

- WILLARD: A simple reflex agent

# Environments ...

## Properties of Environments

**Accessible** The agent's sensory apparatus gives it access to the complete state of the environment.

**Deterministic** The next state of the environment is completely determined by the current state and the actions selected by the agent.

**Episodic** The agent's experience is divided into "episodes," each episode consisting of the agent perceiving and then acting.

**Static** If the environment can change while the agent is deliberating, then the environment is dynamic; otherwise it's static.

**Discrete** There are a limited number of distinct, clearly defined percepts and actions we say that the environment is discrete.

<b>Env.</b>	<b>Acc</b>	<b>Det</b>	<b>Ep</b>	<b>Static</b>	<b>Dis</b>
Chess	Yes	Yes	No	Semi	Yes
Hyperbot	Yes	Yes	Yes	Yes	Yes
BRUTUS	?	?	?	?	?
SHERLOCK	?	?	?	?	?
S <sup>3</sup> G	?	?	?	?	?

- GUESS WHO? and the environment simulator. . .
- GUESS WHO? and problem-solving agents (search: Figure 3.1). . .

## Search & Game Playing

- Problem-Solving Agent (pseudocode)
- Example: Problem in 8-queens problem
  - Goal Test, Path Cost, States, Operators
  - explore, build on your own here; we'll return to it

## Minimax, Chess, Go...

- The minimax algorithm
  - Tic-Tac-Toe (figure)
  - Two-Ply Example (figure)
- Is there a perfect winning strategy for Chess? Yes!
  - but too computationally demanding, so evaluation function
  - and  $\alpha\beta$  pruning (figures)
- “Chess is Too Easy”
- What about Go and Poker, though?

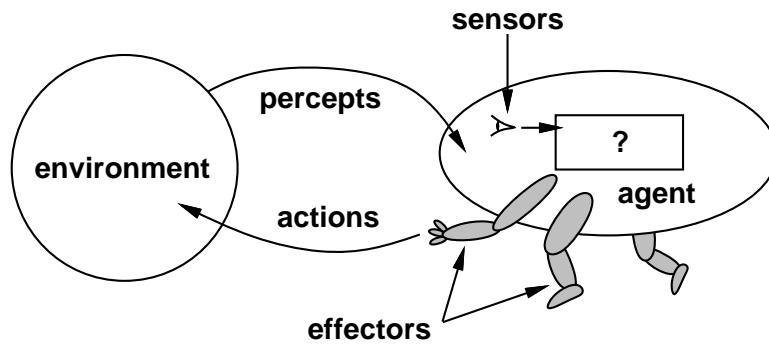


Figure 1: Agent Overview.

Agent Type	Percepts	Actions	Goals	Environment
Medical diagnosis system	Symptoms, findings, patient's answers	Questions, tests, treatments	Healthy patient, minimize costs	Patient, hospital
Satellite image analysis system	Pixels of varying intensity, color	Print a categorization of scene	Correct categorization	Images from orbiting satellite
Part-picking robot	Pixels of varying intensity	Pick up parts and sort into bins	Place parts in correct bins	Conveyor belt with parts
Refinery controller	Temperature, pressure readings	Open, close valves; adjust temperature	Maximize purity, yield, safety	Refinery
Interactive English tutor	Typed words	Print exercises, suggestions, corrections	Maximize student's score on test	Set of students

Figure 2: Sample PAGE Descriptions.

```

function SIMPLE-REFLEX-AGENT(percept) returns action
  static: rules, a set of condition-action rules

  state ← INTERPRET-INPUT(percept)
  rule ← RULE-MATCH(state, rules)
  action ← RULE-ACTION[rule]
  return action

```

Figure 3: Simple Reflex Agent.

```

function REFLEX-AGENT-WITH-STATE(percept) returns action
  static: state, a description of the current world state
           rules, a set of condition-action rules

  state ← UPDATE-STATE(state, percept)
  rule ← RULE-MATCH(state, rules)
  action ← RULE-ACTION[rule]
  state ← UPDATE-STATE(state, action)
  return action

```

Figure 4: Simple Reflex Agent With Internal State.

Environment	Accessible	Deterministic	Episodic	Static	Discrete
Chess with a clock	Yes	Yes	No	Semi	Yes
Chess without a clock	Yes	Yes	No	Yes	Yes
Poker	No	No	No	Yes	Yes
Backgammon	Yes	No	No	Yes	Yes
Taxi driving	No	No	No	No	No
Medical diagnosis system	No	No	No	No	No
Image-analysis system	Yes	Yes	Yes	Semi	No
Part-picking robot	No	No	Yes	No	No
Refinery controller	No	No	No	No	No
Interactive English tutor	No	No	No	No	Yes

Figure 5: Examples of Environments and Their Clocks.

```

procedure RUN-ENVIRONMENT(state, UPDATE-FN, agents, termination)
  inputs: state, the initial state of the environment
           UPDATE-FN, function to modify the environment
           agents, a set of agents
           termination, a predicate to test when we are done

  repeat
    for each agent in agents do
      PERCEPT[agent] ← GET-PERCEPT(agent, state)
    end
    for each agent in agents do
      ACTION[agent] ← PROGRAM[agent](PERCEPT[agent])
    end
    state ← UPDATE-FN(actions, agents, state)
  until termination(state)

```

Figure 6: Basic Environment Simulator.

```

function SIMPLE-PROBLEM-SOLVING-AGENT(p) returns an action
  inputs: p, a percept
  static: s, an action sequence, initially empty
           state, some description of the current world state
           g, a goal, initially null
           problem, a problem formulation

  state ← UPDATE-STATE(state, p)
  if s is empty then
    g ← FORMULATE-GOAL(state)
    problem ← FORMULATE-PROBLEM(state, g)
    s ← SEARCH(problem)
  action ← RECOMMENDATION(s, state)
  s ← REMAINDER(s, state)
  return action

```

Figure 7: Simple Problem-Solving Agent

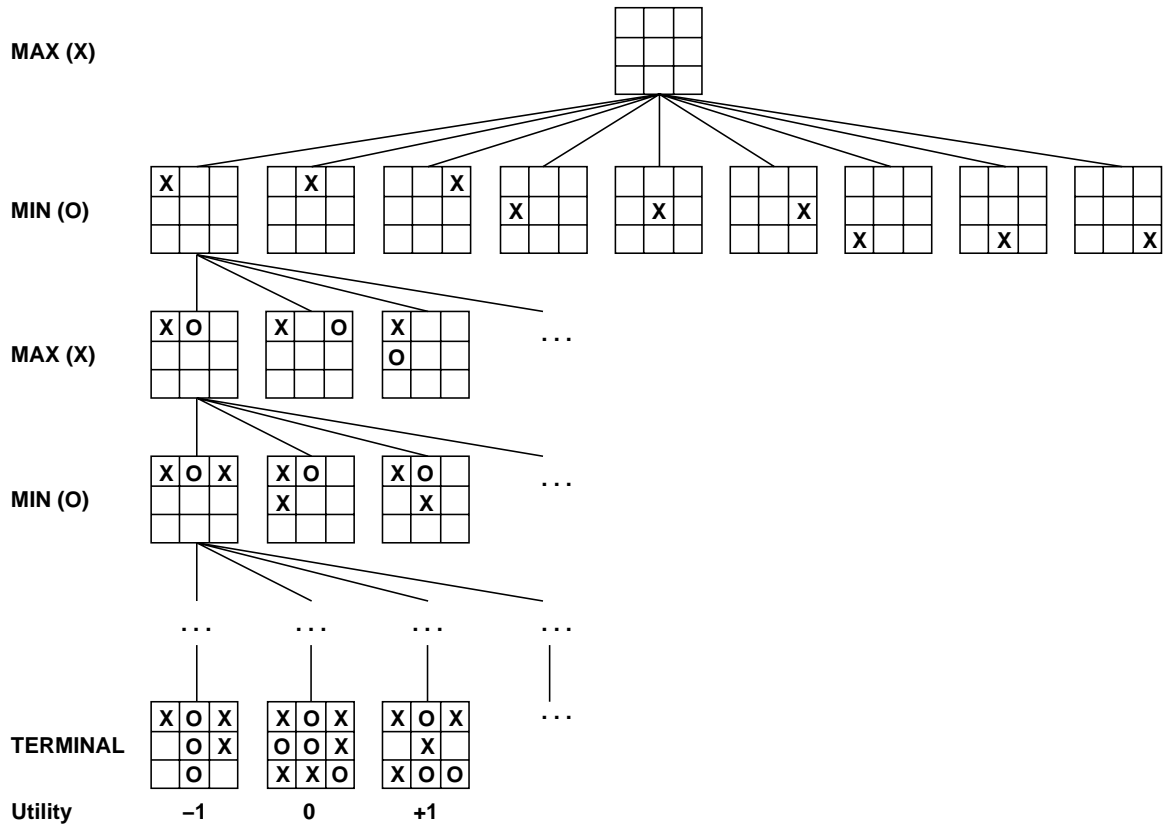


Figure 8: Search Tree for T-T-T.

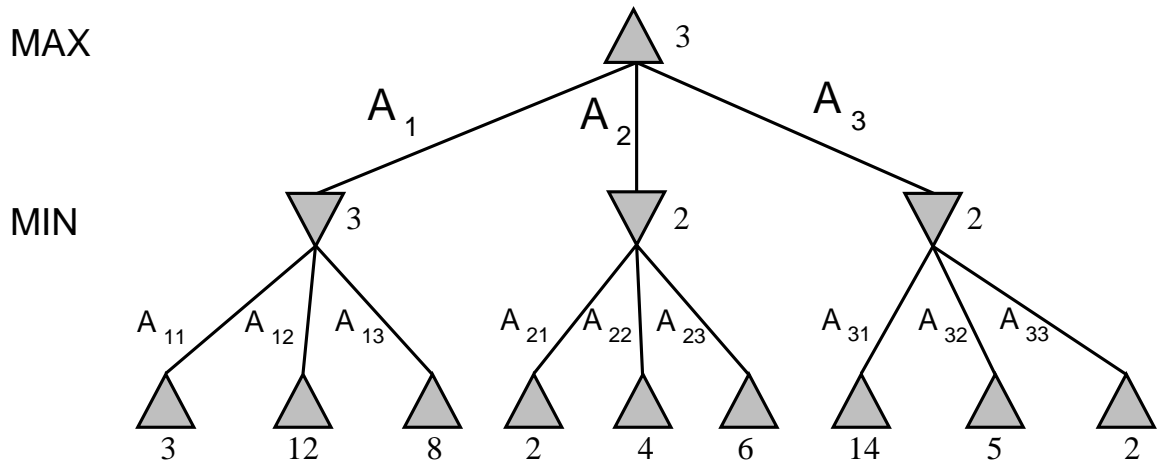


Figure 9: Two-Ply Game.

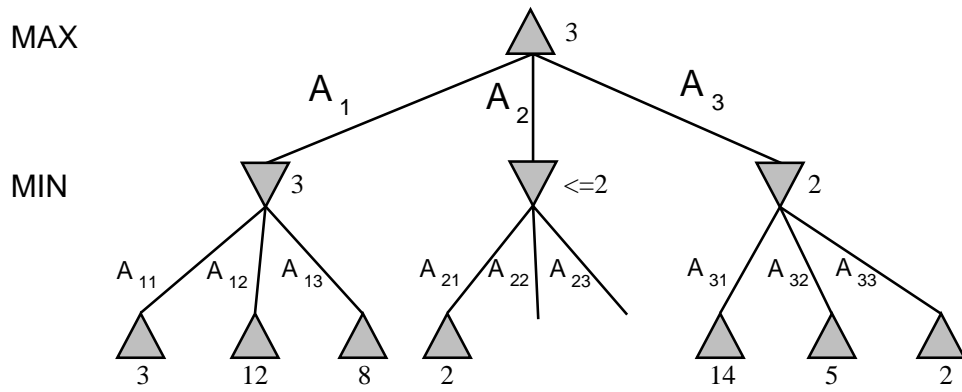


Figure 10: Two-Ply Game Generated by Alpha-Beta.

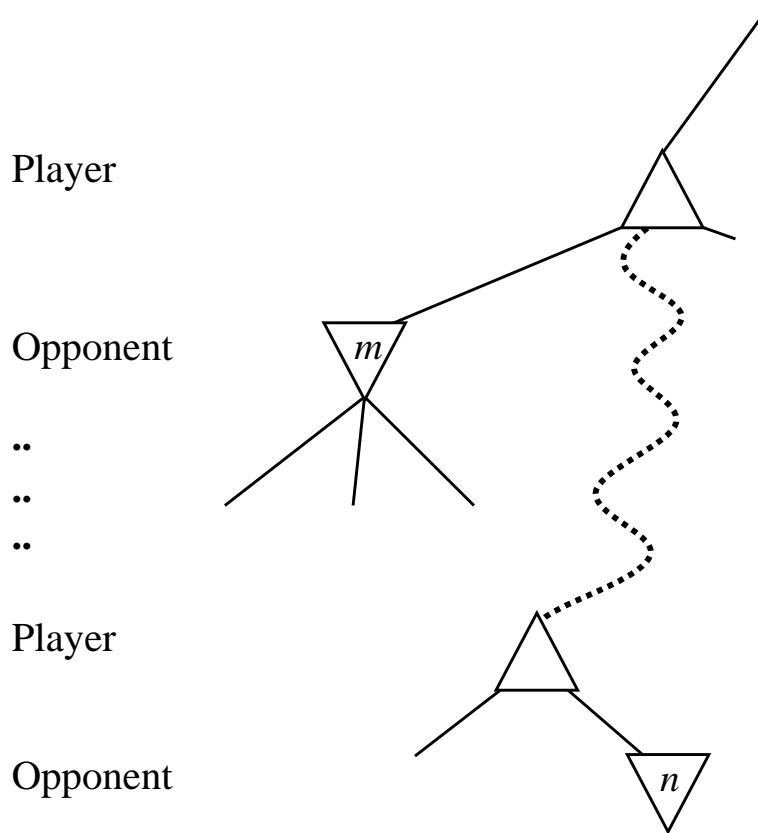


Figure 11: Generalized Alpha-Beta.