Home

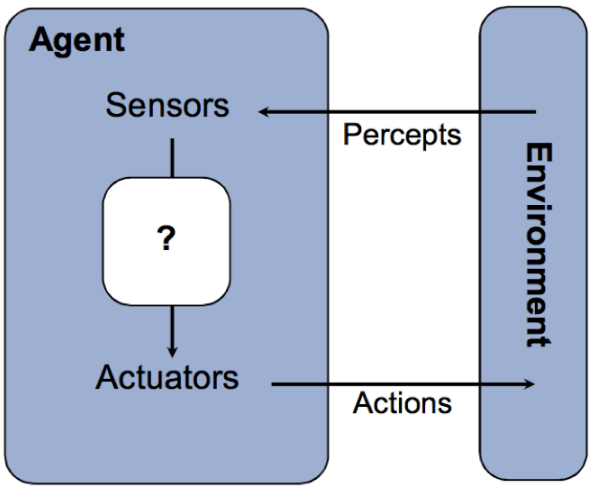
Lecture 2 - Agents and the Environment (/csci/index.php/9-csci-3202-lecture-notes/24-lecture-2-agents-and-the-environment)

Details

Written by Rhonda Hoenigman
Published: 11 August 2015
Hits: 594

What is an agent?

An **agent** is any entity that can perceive its environment through sensors and acts on its environment through effectors.



If you think of humans as agent, we have eyes, ears, and other organs for sensing our world, and then hands, legs, etc that are effectors. What we are most interested in AI is what decisions are made, and how, given the data that comes in through the sensors. In the Agent image shown above, it's the box with the ? that we're going to be discussing all semester.

A rational agent uses percepts of its environment to select rational actions, which is an action that maximizes its utility function. Other considerations for the agent include:

- Is the environment partially or fully observable?
- Is the agent in a single- or multi-agent environment?
- Is the environment deterministic or stochastic? Are the agent's actions deterministic or stochastic?
- Is the environment episodic or sequential? Does the agent perform an action and then get a response before deciding on the next action?
- Is the environment static or dynamic? Does it change while the environment is making a decision?
- Are time and space in the environment discrete or continuous?
- Is the environment known or unknown? Will the agent know the effects of its actions or does it need to learn them?

In the video game Pacman (https://www.youtube.com/watch?v=uswzriFIf_k), the pacman is the agent with a few objectives, depending on who's playing the game.

Pacman's objectives

1. Eat all of the dots quickly.
2. Get as many points as possible.
3. Don't die

Rational actions for Pacman

1. Move toward dots.
2. Move avoid ghosts
3. Move toward fruit.

Considerations for Pacman as an agent

1. What is pacman's environment? *(dots, ghosts, fruit, walls)*
2. What does pacman perceive? (The pacman perceives what the person playing pacman perceives.) *(everything)*
3. What are pacman's actions? *(move in a direction, NSEW)*
4. How do pacman's actions update the environment? *(dots disappear, fruit disappears, ghosts change direction)*

Reflex vs. goal-based agents

Imagine you were writing a program to play pacman. As a pacman player, you understand the rules of the game and when playing, you observe the entire game board to see where the ghosts are and you take the shortest path to uneaten dots. You also go after fruit. You want to incorporate all of this knowledge into a computer program.

Building a Pacman game, you need

- A representation of the pacman world
- Defined goals for pacman
- The ability to maintain the game state
- Search algorithms to find the rational course of action for pacman.

Reflex agents

Reflex agents are the simplest agents and don't have all of the functionality just described. **They choose an action based on the current perception of the world rather than planning or considering future consequences**. For example, in pacman, there could be a rule such as "move away from ghosts", or "If ghost within two moves, move opposite direction". This is a good rule in the short term for staying alive, but could lead pacman to be trapped in a corner. For human agents, a reflex action is one that we take without thinking. If you touch a hot stove, you immediately move your hand away without considering any other dangers in doing so.

A reflex pacman agent will be far easier to implement than one that plans its actions using goals it has set. However, it's also limited in its capabilities and really not that interesting.

Goal-based agents

Goal-based agents plan ahead by considering "what if" a certain action is taken. What will the state of the environment be after the action and does it move the agent closer to its goal? To support goal-based agents, a model of the environment is needed that shows how the environment evolves in response to those actions. The model likely will not include the entire environment, but rather, just the elements of the environment that are needed to measure success. For example, if the goal in pacman is to eat dots, then there needs to be a measure of how many dots are in the environment. If the goal is to score points, then points needs to be measured. Knowing how many points are scored for a given action, as compared to other actions, provides pacman with information for choosing an action.

Searching for a plan to achieve goals

Whether we're talking about pacman, or something else, achieving goals requires having a plan.

A plan is the set of steps you're going to take to achieve that goal you've set.

For pacman, the goal could be to get the dots quickly or to get the most points, and each of those goals requires a plan for the actions that pacman is going to take. Different goal = different plan.

Search algorithms are used to identify the best plan from the set of possible plans. You've all heard of algorithms such as breadth-first and depth-first search. In these algorithms, there is a starting state and a set of possible steps that can be taken from that starting state. The search algorithms evaluate the next states until the goal state is found.

To set up a problem as a search problem, it needs to be formulated in a search framework.

Search framework

- State space: details of the environment that matter for achieving the goal
- Successor function: determines new state from current state given costs and actions
- Start state: current state of the environment
- Goal state: state of the environment we want to achieve
- Goal test: are we there yet? e.g. are all dots gone?

The search algorithm uses the formulation to find a solution to the problem. The solution is the sequence of actions, aka a plan, that transforms the start state into the goal state. Clearly, some plans are better than others. For example, pacman could have a goal to eat all dots by moving randomly about the game board. Or, pacman could have a plan to identify the remaining dots and move along the shortest path to eat the dots. A search algorithm would evaluate both plans and show that the targeted approach was better than the random approach using a utility function and a definition for "better". A utility function is how performance is measured.

State space

The state space includes all of the variables in the environment where the agent lives, and the combinations of values that those variables can have. A well-defined state space is required for creating goal-based agents to know whether a goal has been achieved. In any environment, there are typically elements in a state space that are not needed for measuring a goal and including them would generate way too many variable combinations to evaluate in a timely fashion.

World State

Includes all variables in the environment even if they're not needed in the search problem.

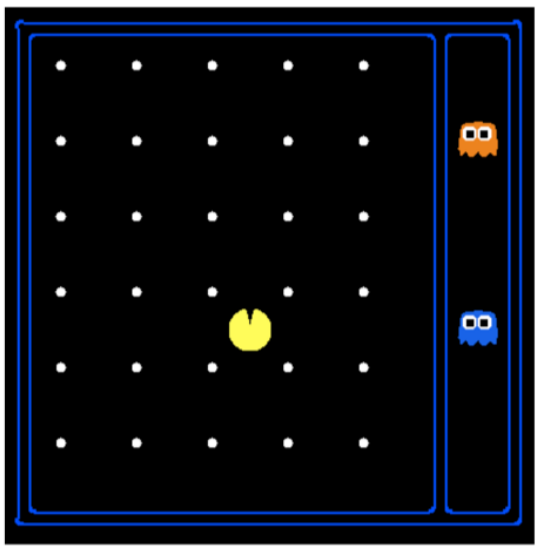
Search State

Includes only the variables needed to solve the problem.

In the pacman example shown here, there are four variables in the world state space and $120*(2^{30})*(12^2)*4$ combinations of those variables. The size of the state space is $120*(2^{30})*(12^2)*4$.

World State Example

- World state:
  - Agent positions: 120
  - Food count: 30
  - Ghost positions: 12
  - Agent facing: NSEW
- How many
  - World states?
    $120x(2^{30})x(12^2)x4$
  - States for pathing?
    120
  - States for eat-all-dots?
    $120x(2^{30})$



In simple pacman example shown in image, the world state includes:

- Pacman's position
- Ghosts position
- Pellets remaining
- Pacman's direction (NSEW)

Search State Examples

The search state uses only the relevant information from the world state needed to solve the problem. No other information about the environment is included, which reduces the search space.

Example: Find a path for pacman from current location to a destination location

State: (x,y) location

Next state: current location + step in NSEW direction, set in a successor function

Goal test: (x,y) location = END

In this example, only the location is needed. The next state is determined in a successor function that uses the current location and the direction that pacman is facing and taking one step in that direction.

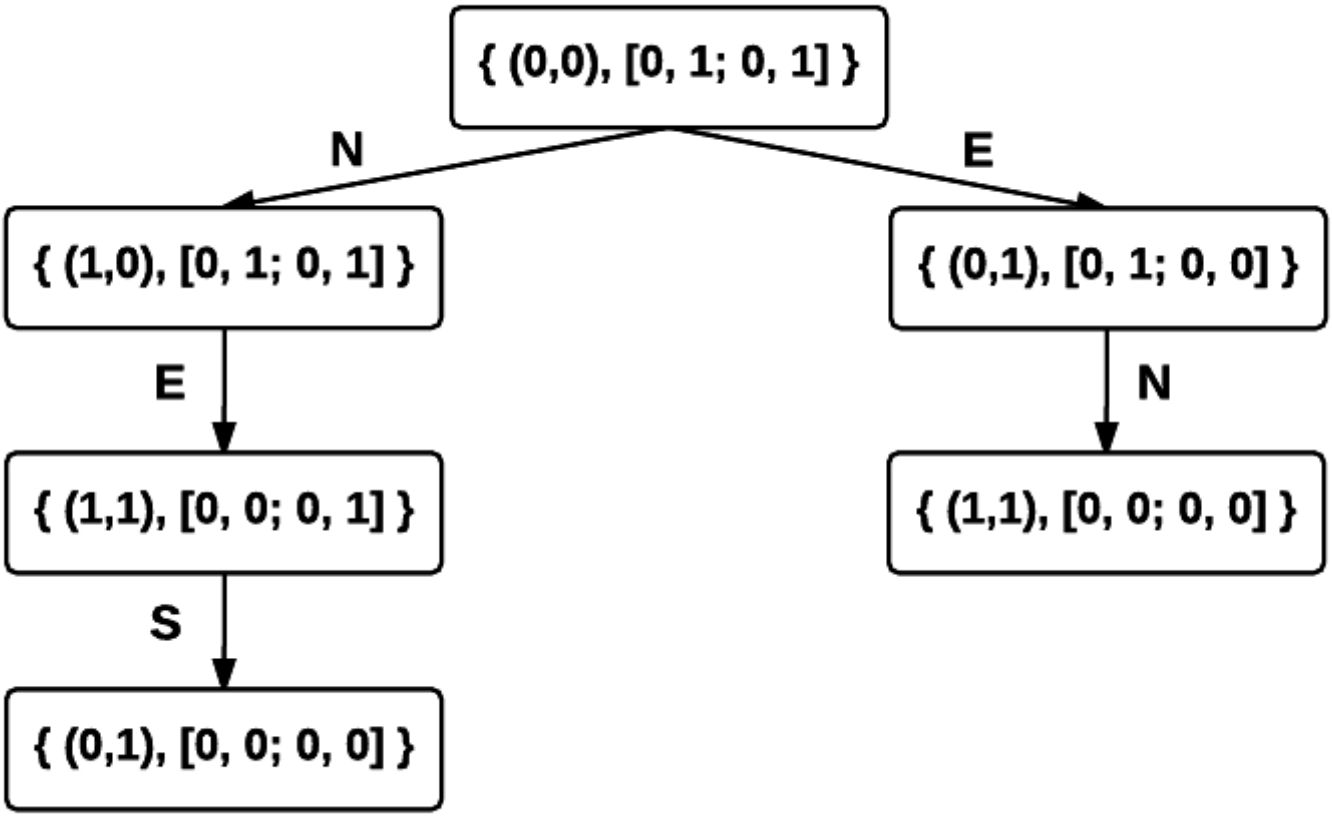
Example: Eat all of the dots

State: (x,y) location and matrix of dots remaining

Next state: current location + step in NSEW direction and matrix of dots remaining after movement

Goal test: All dots gone

This example also uses the location, but also keeps track of how many dots are remaining. There is still nothing intelligent about how dots are consumed. The successor function moves pacman one step from its current location in a specified direction, and also updates the dot map. Starting from (0,0) in the simple example shown above, the state sequences generated by the successor function would look like:



Neither of these problems include information about the positions of the ghosts. However, if the goal also included staying alive, and the ghosts could attack, then ghost position would also need to be included and the size of the state space would increase.