EXERCISES 2

INTELLIGENT AGENTS

2.1 Agents and Environments

Exercise 2.DFAG

Define in your own words the following terms: agent, environment, sensor, actuator, percept, agent function, agent program.

The following are just some of the many possible definitions that can be written:

- *Agent*: an entity that perceives and acts; or, one that *can be viewed* as perceiving and acting. Essentially any object qualifies; the key point is the way the object implements an agent function. (Note: some authors restrict the term to *programs* that operate *on behalf of* a human, or to programs that can cause some or all of their code to run on other machines on a network, as in **mobile agents**.)

Mobile agent

- *Environment*: the world in which the agent is situated, on which the agent's actuators produce effects and from which the agent's sensors receive percepts.
- *Sensor*: part of the agent whose state is affected by the environment and whose state can enter into the agent's computations directly as a value.
- *Actuator*: part of the agent whose state can be set by the agent's computations and affects the environment.
- *Percept*: the observed value of the sensor state (usually, all the sensors taken together) at a given time.
- *Agent function*: given an agent viewed as a fixed physical object, the agent function that specifies the agent's action in response to every possible percept sequence.
- *Agent program*: that program which, combined with a machine architecture, produces an agent function. In our simple designs, the program takes a new percept on each invocation and returns an action. Note that the agent program is generally not simply "implementing" the agent function exactly, because the program may take more than one time step to return an action.

Exercise 2.AGEX

For each of the following agents, specify the sensors, actuators, and environment: microwave oven, chess program, autonomous supply delivery plane.

The following are just some of the many possible definitions that can be written:

- *Microwave oven*: Here it is important to view the oven as the agent, not the human who is using it; this means the buttons are sensors, not actuators! It might seem that there are no decisions to make but the internal controller logic can be quite complex, particularly if there are additional sensors. For safety, no power should be supplied if the door is open!
  - *Sensors*: Button state, door-open sensor. Some microwaves have humidity sensors, temperature sensors, or acoustic sensors (for popcorn).
  - *Actuators*: Turn on/turn off/adjust microwave power; sound completion alarm; indicate illegal button state.
  - *Environment*: The contents of the oven and the state of the door (open/closed).
- *Chess program*:
  - *Sensors*: A typical program accepts keyboard, mouse, or touchscreen input indicating the opponent's intended move, resignation, hint request, etc.
  - *Actuators*: Display a legal move, or the board resulting from the move.
  - *Environment*: At a minimum, the environment includes a representation of the board state. There are some subtle issues related to whether this is identical to the program's own internal representation or the displayed board that is accessible to the opponent. A more sophisticated program might include the human opponent as part of the environment, and try to build a model of that human's playing style and ability in order to improve its chances of winning or make the game more entertaining/instructive.
- *Autonomous supply delivery plane*: The answers here are quite similar to those for the autonomous taxi in Section 2.3. Details can be found in numerous online sources describing current autonomous delivery systems.
  - *Sensors*: GPS, air speed (e.g., Pitot tube), altimeter (actually an air pressure sensor, so reported altitude depends on meteorological conditions), sensors reporting state of each actuator, 3-axis accelerometer, possibly a camera.
  - *Actuators*: power to propeller, ailerons, elevators, rudder; drop payload.
  - *Environment*: geographical area of flight, atmospheric conditions.

Exercise 2.AGFN

This exercise explores the differences between agent functions and agent programs.

a. Can there be more than one agent program that implements a given agent function? Give an example, or show why one is not possible.

b. Are there agent functions that cannot be implemented by any agent program?

c. Given a fixed machine architecture, does each agent program implement exactly one agent function?

d. Given an architecture with n bits of storage, how many different possible agent programs are there?

e. Suppose we keep the agent program fixed but speed up the machine by a factor of two. Does that change the agent function?

Although these questions are very simple, they hint at some very fundamental issues. Our answers are for the simple agent designs for *static* environments where nothing happens while the agent is deliberating; the issues get even more interesting for dynamic environments.

a. Yes; take any agent program and insert null statements that do not affect the output.

b. Yes; the agent function might specify that the agent print $true$ when the percept is a Turing machine program that halts, and $false$ otherwise. (Note: in dynamic environments, for machines of less than infinite speed, the rational agent function may not be implementable; e.g., the agent function that always plays a winning move, if any, in a game of chess.)

c. Yes; the agent's behavior is fixed by the architecture and program.

d. There are 2^n agent programs, although many of these will not run at all. (Note: Any given program can devote at most n bits to storage, so its internal state can distinguish among only 2^n past histories. Because the agent function specifies actions based on percept histories, there will be many agent functions that cannot be implemented because of lack of memory in the machine.)

e. It depends on the program and the environment. If the environment is dynamic, speeding up the machine may mean choosing different (perhaps better) actions and/or acting sooner. If the environment is static and the program pays no attention to the passage of elapsed time, the agent function is unchanged.

2.2 Good Behavior: The Concept of Rationality

Exercise 2.PRMT

Suppose that the performance measure is concerned with just the first T time steps of the environment and ignores everything thereafter. Show that a rational agent's action may depend not just on the state of the environment but also on the time step it has reached.

This question tests the student's understanding of environments, rational actions, and performance measures. Any sequential environment in which rewards may take time to arrive will work, because then we can arrange for the reward to be "over the horizon." Suppose that in any state there are two action choices, a and b, and consider two cases: the agent is in state s at time T or at time $T-1$. In state s, action a reaches state s' with reward 0, while action b reaches state s again with reward 1; in s' either action gains reward 10. At time $T-1$, it's rational to do a in s, with expected total reward 10 before time is up; but at time T, it's rational to do b with total expected reward 1 because the reward of 10 cannot be obtained before time is up.

Students may also provide common-sense examples from real life: investments whose payoff occurs after the end of life, exams where it doesn't make sense to start the high-value question with too little time left to get the answer, and so on.

The environment state can include a clock, of course; this doesn't change the gist of the answer—now the action will depend on the clock as well as on the non-clock part of the state—but it does mean that the agent can never be in the same state twice.

Exercise 2.VACR

Let us examine the rationality of various vacuum-cleaner agent functions.

a. Show that the simple vacuum-cleaner agent function described in Figure 2.3 is indeed rational under the assumptions listed on page 58.
b. Describe a rational agent function for the case in which each movement costs one point. Does the corresponding agent program require internal state?
c. Discuss possible agent designs for the cases in which clean squares can become dirty and the geography of the environment is unknown. Does it make sense for the agent to learn from its experience in these cases? If so, what should it learn? If not, why not?

Notice that for our simple environmental assumptions we need not worry about quantitative uncertainty.

a. It suffices to show that for all possible actual environments (i.e., all dirt distributions and initial locations), this agent cleans the squares at least as fast as any other agent. This is trivially true when there is no dirt. When there is dirt in the initial location and none in the other location, the world is clean after one step; no agent can do better. When there is no dirt in the initial location but dirt in the other, the world is clean after two steps; no agent can do better. When there is dirt in both locations, the world is clean after three steps; no agent can do better. (Note: in general, the condition stated in the first sentence of this answer is much stricter than necessary for an agent to be rational.)

b. The agent in (a) keeps moving backwards and forwards even after the world is clean. It is better to do $NoOp$ once the world is clean (the chapter says this). Now, since the agent's percept doesn't say whether the other square is clean, it would seem that the agent must have some memory to say whether the other square has already been cleaned. To make this argument rigorous is more difficult—for example, could the agent arrange things so that it would only be in a clean left square when the right square was already clean? As a general strategy, an agent *can* use the environment itself as a form of **external memory**—a common technique for humans who use things like appointment calendars and knots in handkerchiefs. In this particular case, however, that is not possible. Consider the reflex actions for $[A, \mathit{Clean}]$ and $[B, \mathit{Clean}]$. If either of these is $NoOp$, then the agent will fail in the case where that is the initial percept but the other square is dirty; hence, neither can be $NoOp$ and therefore the simple reflex agent is doomed to keep moving. In general, the problem with reflex agents is that they have to do the same thing in situations that look the same, even when the situations are actually quite different. In the vacuum world this is a big liability, because every interior square (except home) looks either like a square with dirt or a square without dirt.

External memory

c. If we consider asymptotically long lifetimes, then it is clear that learning a map (in some form) confers an advantage because it means that the agent can avoid bumping into walls. It can also learn where dirt is most likely to accumulate and can devise an optimal inspection strategy. The precise details of the exploration method needed to construct a complete map appear in Chapter 4; methods for deriving an optimal inspection/cleanup strategy are in Chapter 23.

Exercise 2.KPER

Describe three different task environments in which the performance measure is easy to specify completely and correctly, and three in which it is not.

Obviously there are many possible answers here. For the "easy" case, it makes sense to consider artificially defined task environments: electronic calculators (answers correct to the required number of significant digits, elapsed time), chess programs (win/draw/loss subject to time constraints), spider solitaire (number of suits completed, number of moves, elapsed time). But even in these environments, one must be careful, because such performance measures ignore *all other considerations*. For example, a general-purpose intelligent agent that has the ability to display messages or access the internet might improve its chess performance by acquiring additional hardware. Marvin Minsky pointed out that a machine designed to calculate as many digits of π as possible could take over the world to do so.

For the "hard" case, almost any task involving humans will do: tutor, automated taxi, digital personal assistant, military strategist, and so on. Although Section 2.3 lists the elements of the taxi's performance measure (reaching destination, fuel consumption, wear and tear, trip time, traffic laws, politness to other drivers, safety, passenger comfort, profit), the appropriate tradeoffs among these are far from clear. For example, if the taxi is 200 yards from the destination but stuck in a traffic jam that will take 20 minutes to clear, should it suggest to the passengers that they walk the remaining 200 yards? And how does this decision depend on the weather and general safety and legality of this premature dropoff location?

In addition to difficult tradeoffs, there will typically be attributes that are omitted from the performance measure that do in fact matter and can be affected by the agent. Obviously it doesn't make sense to ask students to list the attributes that are omitted, because they could simply include them! Instead, one might ask which attributes are "obvious" and which are "non-obvious," i.e., likely to be omitted on a first or second pass.

Exercise 2.EVSA

Write an essay on the relationship between evolution and one or more of autonomy, intelligence, rationality, and learning.

Exercise 2.AGTF

For each of the following assertions, say whether it is true or false and support your answer with examples or counterexamples where appropriate.

a. An agent that senses only partial information about the state cannot be perfectly rational.

b. There exist task environments in which no pure reflex agent can behave rationally.

c. There exists a task environment in which every agent is rational.

d. The input to an agent program is the same as the input to the agent function.

e. Every agent function is implementable by some program/machine combination.

f. Suppose an agent selects its action uniformly at random from the set of possible actions. There exists a deterministic task environment in which this agent is rational.

g. It is possible for a given agent to be perfectly rational in two distinct task environments.

h. Every agent is rational in an unobservable environment.

i. A perfectly rational poker-playing agent never loses.

j. For a simple reflex agent in a partially observable environment, a randomized policy can outperform any deterministic policy.

k. There is a model-based reflex agent that can remember all of its percepts.

l. Suppose agent A1 is rational and agent A2 is irrational. There exists a task environment where A2's actual score will be greater than A1's actual score.

a. *An agent that senses only partial information about the state cannot be perfectly rational.*
False. Perfect rationality refers to the ability to make good decisions given the sensor information received. Moreover, in *some* environments, an agent can deliberately ignore part of its sensor information and still be perfectly rational. For example, a physical chess agent can ignore flecks of dust on the chessboard and pretty much anything that isn't on the chessboard or the clock.

b. *There exist task environments in which no pure reflex agent can behave rationally.*
True. A pure reflex agent ignores previous percepts, so cannot obtain an optimal state estimate in a partially observable environment. For example, correspondence chess is played by sending moves; if the other player's move is the current percept, a reflex agent could not keep track of the board state and would have to respond to, say, "a4" in the same way regardless of the position in which it was played.

c. *There exists a task environment in which every agent is rational.*
True. For example, in an environment with a single state, such that all actions have the same reward, it doesn't matter which action is taken. More generally, any environment that is reward-invariant under permutation of the actions will satisfy this property.

d. *The input to an agent program is the same as the input to the agent function.*
False. The agent function, notionally speaking, takes as input the entire percept sequence up to that point, whereas the agent program takes the current percept only.

e. *Every agent function is implementable by some program/machine combination.*
False. For example, the environment may contain Turing machines and input tapes and the agent's job is to solve the halting problem; there is an agent *function* that specifies the right answers, but no agent program can implement it. Another example would be an agent function that requires solving intractable problem instances of arbitrary size in constant time.

f. *Suppose an agent selects its action uniformly at random from the set of possible actions. There exists a deterministic task environment in which this agent is rational.*
True. This is a special case of (c); if it doesn't matter which action you take, selecting

randomly is rational.

g. *It is possible for a given agent to be perfectly rational in two distinct task environments.*
True. For example, we can arbitrarily modify the parts of the environment that are unreachable by any optimal policy as long as they stay unreachable.

h. *Every agent is rational in an unobservable environment.*
False. Some actions are stupid—and the agent may know this if it has a model of the environment—even if one cannot perceive the environment state.

i. *A perfectly rational poker-playing agent never loses.*
False. Unless it draws the perfect hand, the agent can always lose if an opponent has better cards. This can happen for game after game. The correct statement is that the agent's expected winnings are nonnegative.

j. *For a simple reflex agent in a partially observable environment, a randomized policy can outperform any deterministic policy.*
True. Some vacuum robots use this idea to get themselves "unstuck" from corners. A partially observable environment can have hidden state that makes any particular deterministically chosen action fail when executed in response to a given percept, whereas a randomized policy may eventually chose the right action (if there is one). As an example, consider a reflex agent that needs the PIN to ulock a phone. A randomized agent will eventually emit the correct sequence. whereas a deterministic agent can only emit the same sequence over and over.

k. *There is a model-based reflex agent that can remember all of its percepts.*
True. A model-based agent updates an internal state representation—its "memory"—for each new percept. Memorizing the percepts is possible (assuming unbounded memory, which is a reasonable caveat). This may not be the most perspicuous representation of the current state of the world, but any other such representation of the current state could be computed from it on demand, for example by running any other model-based reflex agent's state estimation algorithm on the memorized percept sequence.

l. *Suppose agent A1 is rational and agent A2 is irrational. There exists a task environment where A2's actual score will be greater than A1's actual score.*
True. Rational decisions are defined by expected outcomes, not actual outcomes. A1 might just be unlucky.

2.3 The Nature of Environments

Exercise 2.PEAS

For each of the following activities, give a PEAS description of the task environment and characterize it in terms of the properties listed in Section 2.3.2.

- Playing soccer.
- Exploring the subsurface oceans of Titan.
- Shopping for used AI books on the Internet.
- Playing a tennis match.

- Practicing tennis against a wall.
- Performing a high jump.
- Knitting a sweater.
- Bidding on an item at an auction.

Many of these can actually be argued either way, depending on the level of detail and abstraction.

A. Partially observable, stochastic, sequential, dynamic, continuous, multi-agent.
B. Partially observable, stochastic, sequential, dynamic, continuous, single agent (unless there are alien life forms that are usefully modeled as agents).
C. Partially observable, deterministic, sequential, static, discrete, single agent. This can be multi-agent and dynamic if we buy books via auction, or dynamic if we purchase on a long enough scale that book offers change.
D. Fully observable, stochastic, episodic (every point is separate), dynamic, continuous, multi-agent.
E. Fully observable, stochastic, episodic, dynamic, continuous, single agent.
F. Fully observable, stochastic, sequential, static, continuous, single agent.
G. Fully observable, deterministic, sequential, static, continuous, single agent.
H. Fully observable, strategic, sequential, static, discrete, multi-agent.

Exercise 2.VCES

Implement a performance-measuring environment simulator for the vacuum-cleaner world depicted in Figure 2.2 and specified on page 58. Your implementation should be modular so that the sensors, actuators, and environment characteristics (size, shape, dirt placement, etc.) can be changed easily. (*Note:* for some choices of programming language and operating system there are already implementations in the online code repository.)

The code repository implements a vacuum-cleaner environment. Students can easily extend it to generate different shaped rooms, obstacles, dirt generation processes, sensor suites, and so on.

Exercise 2.ENVP

For each of the following task environment properties, rank the example task environments from most to least according to how well the environment satisfies the property. Lay out any assumptions you make to reach your conclusions.

a. *Fully Observable*: driving; document classification; tutoring a high-school student in calculus; skin cancer diagnosis from images.

b. *Continuous*: driving; spoken conversation; written conversation; climate engineering by stratospheric aerosol injection.
c. *Stochastic*: driving; sudoku; poker; soccer.
d. *Static*: chat room; checkers; tax planning; tennis.

a. *Fully Observable*: document classification > skin cancer diagnosis from images > driving > tutoring a high-school student in calculus.
Document classification is a fairly canonical example of a (non-sequential) observable problem, because the correct classification depends almost entirely on the visible text of the document itself. There might be a slight influence from "provenance" information (date, authorship, etc.) that may not be directly observable. Skin cancer diagnosis can sometimes be done well from an image of the lesion, bot other factors such as patient age, changes in the lesion over time, medical history, and family history can be important. Driving is often considered to be observable because we imagine that we are making decisions based on what we see, but (1) velocity and turn signal status of other vehicles can be judged only from multiple image frames, and (2) assessing the intended actions of other vehicles may require accumulating information over an extended period—e.g., to determine if a vehicle is stopped or broken down, driving slowly or looking for an address or a parking spot, turning left or has forgotten to turn off the turn signal. Other vehicles, hedges, fog, and so on can obscure visual access to important aspects of the driving environment. Tutoring is almost completely unobservable: what matters is the student's level of understanding, learning style, basic math skills, etc. Clues must be gathered over days, weeks, and months.

b. *Continuous*: climate engineering > driving > spoken conversation > written conversation.
Climate engineering by aerosol injection is quintessentially continuous: the engineer must decide how much to inject, where, and when, and all of these are continuous quantities. The control actions of driving are mostly continuous (steering, acceleration/braking) but there are discrete elements (turn signal, headlights). More importantly, the problem is usually handled using discrete high-level actions (change lanes left, take exit, etc.) that have implementations as continuous control problems. This kind of discrete/continuous hierarchy is very common; playing chess in the physical world is a perfect example. Spoken conversation is closer to chess than driving: roughly speaking, we choose the discrete words to say and delegate the saying to continuous motor control routines. Prosody (volume, pitch, and speed variation) is, however, an important continuous element in how we speak that is largely absent from written communication.

c. *Stochastic*: poker > soccer > driving > sudoku.
In poker, nearly everything is determined by the fall of the cards, which is entirely stochastic from the viewpoint of the players. Both soccer and driving contain elements that are fairly deterministic, such as the flight of the ball and the response of the engine, and elements that are stochastic, such as tire punctures and the outcomes of tackles. Yet typically one can make reasonably reliable driving plans over many minutes, whereas it

is essentially impossible to predict the state of a soccer game one minute into the future. Sudoku, of course, is entirely deterministic.

d. *Static*: tax planning > checkers > chat room > tennis.
While no human activity is completely static, given the finite length of our lifetimes, tax planning comes close—the typical "deadline" to get it done is often weeks or months, and the relevant aspects of the environment (life/death, number of offspring, tax law) may change even more slowly. In checkers, the world state doesn't change until someone moves, but the clock ticks so the problem is semi-dynamic. In the chat room, long delays in replying are unacceptable, so it is a fairly real-time environment, but not nearly as real-time as tennis, where a delay of a split second often makes the difference between winning and losing a point.

2.4 The Structure of Agents

Exercise 2.SIRA

Implement a simple reflex agent for the vacuum environment in Exercise VACUUM-START-EXERCISE. Run the environment with this agent for all possible initial dirt configurations and agent locations. Record the performance score for each configuration and the overall average score.

Pseudocode for a reflex agent is given in Figure 2.8. For states 1, 3, 5, 7 in Figure 4.9, the performance measures are 1996, 1999, 1998, 2000 respectively. The average is 1998.25.

Exercise 2.VCPE

Consider a modified version of the vacuum environment in Exercise VACUUM-START-EXERCISE, in which the agent is penalized one point for each movement.

a. Can a simple reflex agent be perfectly rational for this environment? Explain.

b. What about a reflex agent with state? Design such an agent.

c. How do your answers to **a** and **b** change if the agent's percepts give it the clean/dirty status of every square in the environment?

a. No. Consider the reflex actions for $[A, \mathit{Clean}]$ and $[B, \mathit{Clean}]$. If either of these is NoOp, then the agent will fail in the case where that is the initial percept but the other square is dirty; hence, neither can be NoOp and therefore the simple reflex agent is doomed to keep moving, which loses points unnecessarily. In general, the problem with reflex agents is that they have to do the same thing in situations that *look the same*, even when the situations are *actually quite different*. In the vacuum world this is a big liability, because every interior square (except home) looks either like a square with dirt or a square without dirt.

b. Yes, a reflex agent with state can be perfectly rational. It simply needs to remember whether it has visited the other square. If so, it can do nothing after cleaning the current square (if needed). If not, it then goes to the other square and cleans it if needed.

c. In this case, a simple reflex agent can be perfectly rational. The agent can consist of a table with eight entries, indexed by percept, that specifies an action to take for each possible state. After the agent acts, the world is updated and the next percept will tell the agent what to do next. For larger environments, constructing a table is infeasible. Instead, the agent could run one of the optimal search algorithms in Chapter 3 and execute the first step of the solution sequence. Again, no internal state is *required*, but it would help to be able to store the solution sequence instead of recomputing it for each new percept.

Exercise 2.VACU

Consider a modified version of the vacuum environment in Exercise VACUUM-START-EXERCISE, in which the geography of the environment—its extent, boundaries, and obstacles—is unknown, as is the initial dirt configuration. (The agent can go Up and $Down$ as well as $Left$ and $Right$.)

a. Can a simple reflex agent be perfectly rational for this environment? Explain.

b. Can a simple reflex agent with a *randomized* agent function outperform a simple reflex agent? Design such an agent and measure its performance on several environments.

c. Can you design an environment in which your randomized agent will perform poorly? Show your results.

d. Can a reflex agent with state outperform a simple reflex agent? Design such an agent and measure its performance on several environments. Can you design a rational agent of this type?

a. Because the agent does not know the geography and perceives only location and local dirt, and cannot remember what just happened, it will get stuck forever against a wall when it tries to move in a direction that is blocked—that is, unless it randomizes.

b. One possible design cleans up dirt and otherwise moves in a randomly chosen direction. This is fairly close to what the early-model Roomba$^{\mathrm{TM}}$ vacuum cleaner does, although the Roomba has a bump sensor and randomizes only when it hits an obstacle. It works reasonably well but it can take a long time to cover all the squares at least once. Many modern robot cleaners build a map and derive an efficient cleaning pattern.

c. An example is shown in Figure S2.1. Students may also wish to measure clean-up time for linear or square environments of different sizes, and compare those to the efficient online search algorithms described in Chapter 4. It's worth noting that in the infinite-time limit on a bidirectional graph, an agent that moves randomly will spend time in each square proportional to its degree in the graph (the number of neighbors). Still, it can take a long time to get from one part of the graph to another. If the agent is in a long corridor, its expected travel after n time steps is $O(\sqrt{n})$ squares.

d. A reflex agent with state can build a map (see Chapter 4 for details). An online depth-first exploration will reach every state in time linear in the size of the environment; therefore, the agent can do much better than the simple reflex agent.

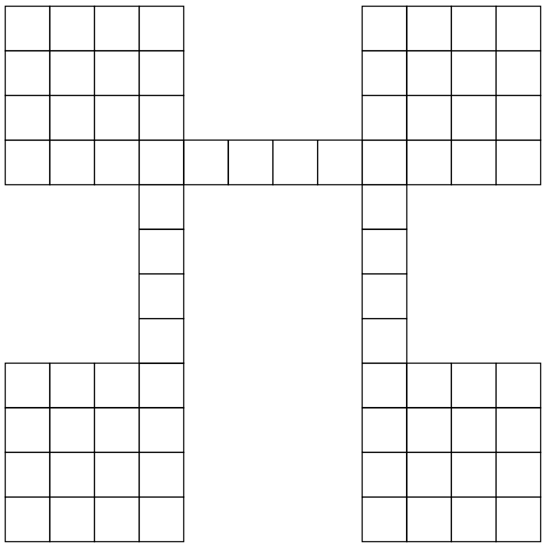

Figure S2.1 An environment in which random motion will take a long time to cover all the squares.

The question of rational behavior in unknown environments is a complex one but it is worth encouraging students to think about it. We need to have some notion of the prior probability distribution over the class of environments; call this the initial **belief state**. Any action yields a new percept that can be used to update this distribution, moving the agent to a new belief state. Once the environment is completely explored, the belief state collapses to a single possible environment. Therefore, the problem of optimal exploration can be viewed as a search for an optimal strategy in the space of possible belief states. This is a well-defined, if horrendously intractable, problem. Chapter 16 discusses some cases (called **bandit problems**) where optimal exploration is possible. Another concrete example of exploration is the Minesweeper computer game (see Exercise 7.22). For very small Minesweeper environments, optimal exploration is feasible although the belief state update is nontrivial to explain.

Exercise 2.VACB

Repeat Exercise VACUUM-UNKNOWN-GEOG-EXERCISE for the case in which the location sensor is replaced with a "bump" sensor that detects the agent's attempts to move into an obstacle or to cross the boundaries of the environment. Suppose the bump sensor stops working; how should the agent behave?

The problem appears at first to be very similar; the main difference is that instead of using the location percept to build the map, the agent has to "invent" its own locations (which, after all, are just nodes in a data structure representing the state space graph). When a bump is detected, the agent assumes it remains in the same location and can add a wall to its map. For grid environments, the agent can keep track of its (x, y) location and so can tell when it has returned to an old state. In the general case, however, there is no simple way to tell if a state is new or old.

Exercise 2.VFIN

The vacuum environments in the preceding exercises have all been deterministic. Discuss possible agent programs for each of the following stochastic versions:

a. Murphy's law: twenty-five percent of the time, the $Suck$ action fails to clean the floor if it is dirty and deposits dirt onto the floor if the floor is clean. How is your agent program affected if the dirt sensor gives the wrong answer 10% of the time?

b. Small children: At each time step, each clean square has a 10% chance of becoming dirty. Can you come up with a rational agent design for this case?

a. For a reflex agent, this presents no *additional* challenge, because the agent will continue to $Suck$ as long as the current location remains dirty. For an agent that constructs a sequential plan, every $Suck$ action would need to be replaced by "$Suck$ until clean." If the dirt sensor can be wrong on each step, then the agent might want to wait for a few steps to get a more reliable measurement before deciding whether to $Suck$ or move on to a new square. Obviously, there is a trade-off because waiting too long means that dirt remains on the floor (incurring a penalty), but acting immediately risks either dirtying a clean square or ignoring a dirty square (if the sensor is wrong). A rational agent must also continue touring and checking the squares in case it missed one on a previous tour (because of bad sensor readings). It is not immediately obvious how the waiting time at each square should change with each new tour. These issues can be clarified by experimentation, which may suggest a general trend that can be verified mathematically. This problem is a partially observable Markov decision process—see Chapter 16. Such problems are hard in general, but some special cases may yield to careful analysis.

b. In this case, the agent must keep touring the squares indefinitely. The probability that a square is dirty increases monotonically with the time since it was last cleaned, so the rational strategy is, roughly speaking, to repeatedly execute the shortest possible tour of all squares. (We say "roughly speaking" because there are complications caused by the fact that the shortest tour may visit some squares twice, depending on the geography.) This problem is also a partially observable Markov decision process.

Exercise 2.DFRA

Define in your own words the following terms: rationality, autonomy, reflex agent, model-based agent, goal-based agent, utility-based agent, learning agent.

The following are just some of the many possible definitions that can be written:

- *Rationality*: a property of agents that choose actions that maximize their expected utility, given the percepts to date.

- *Autonomy*: a property of agents whose behavior is determined by their own experience rather than solely by their initial programming.
- *Reflex agent*: an agent whose action depends only on the current percept.
- *Model-based agent*: an agent whose action is derived directly from an internal model of the current world state that is updated over time.
- *Goal-based agent*: an agent that selects actions that it believes will achieve explicitly represented goals.
- *Utility-based agent*: an agent that selects actions that it believes will maximize the expected utility of the outcome state.
- *Learning agent*: an agent whose behavior improves over time based on its experience.

Exercise 2.GBUT

Write pseudocode agent programs for the goal-based and utility-based agents.

The design of goal- and utility-based agents depends on the structure of the task environment. The simplest such agents, for example those in chapters 3 and 10, compute the agent's entire future sequence of actions in advance before acting at all. This strategy works for static and deterministic environments which are either fully-known or unobservable

For fully-observable and fully-known static environments a policy can be computed in advance which gives the action to by taken in any given state.

For partially-observable environments the agent can compute a conditional plan, which specifies the sequence of actions to take as a function of the agent's perception. In the extreme, a conditional plan gives the agent's response to every contingency, and so it is a representation of the entire agent function.

In all cases it may be either intractable or too expensive to compute everything out in advance. Instead of a conditional plan, it may be better to compute a single sequence of actions which is likely to reach the goal, then monitor the environment to check whether the plan is succeeding, repairing or replanning if it is not. It may be even better to compute only the start of this plan before taking the first action, continuing to plan at later time steps.

Pseudocode for simple goal-based agent is given in Figure S2.2. GOAL-ACHIEVED tests to see whether the current state satisfies the goal or not, doing nothing if it does. PLAN computes a sequence of actions to take to achieve the goal. This might return only a prefix of the full plan, the rest will be computed after the prefix is executed. This agent will act to maintain the goal: if at any point the goal is not satisfied it will (eventually) replan to achieve the goal again.

At this level of abstraction the utility-based agent is not much different than the goal-based agent, except that action may be continuously required (there is not necessarily a point where the utility function is "satisfied"). Pseudocode is given in Figure S2.3.

Exercise 2.FURN

Consider a simple thermostat that turns on a furnace when the temperature is at least 3 degrees below the setting, and turns off a furnace when the temperature is at least 3 degrees

```
function GOAL-BASED-AGENT(p̌ercept) returns an action
  persistent: štate, the agent's current conception of the world state
              m̌odel, a description of how the next state depends on current state and action
              ǧoal, a description of the desired goal state
              p̌lan, a sequence of actions to take, initially empty
              ǎction, the most recent action, initially none

  štate ← UPDATE-STATE(štate, ǎction, p̌ercept, m̌odel)
  if GOAL-ACHIEVED(štate,ǧoal) then return a null action
  if p̌lan is empty then
p̌lan ← PLAN(štate,ǧoal,m̌odel)
  ǎction ← FIRST(p̌lan)
  p̌lan ← REST(p̌lan)
  return ǎction
```

Figure S2.2 A goal-based agent.

```
function UTILITY-BASED-AGENT(p̌ercept) returns an action
  persistent: štate, the agent's current conception of the world state
              m̌odel, a description of how the next state depends on current state and action
              ǔtility-function, a description of the agent's utility function
              p̌lan, a sequence of actions to take, initially empty
              ǎction, the most recent action, initially none

štate ← UPDATE-STATE(štate, ǎction, p̌ercept, m̌odel)
if p̌lan is empty then
p̌lan ← PLAN(štate,ǔtility-function,m̌odel)
ǎction ← FIRST(p̌lan)
p̌lan ← REST(p̌lan)
return ǎction
```

Figure S2.3 A utility-based agent.

above the setting. Is a thermostat an instance of a simple reflex agent, a model-based reflex agent, or a goal-based agent?

The thermostat is best understood as a simple reflex agent. Although the temperature setting might be viewed as a goal, the agent has no notion of how its actions will lead to the goal; so it's better to view the temperature setting as part of the agent's percepts. A more complex control system might well have an internal model of the house, the behavior of its occupants, the capabilities of the heating system, and so on, and would be viewed as a goal-based system in much the same way as a self-driving car that tries to reach a specified destination quickly and cheaply.

Exercise 2.AGPR

Here is pseudocode for three agent programs A, B, C:

```
function A(percept)
return $f_A()$
```

```
function B(percept)
return $f_B$(percept)
```

```
function C(percept)
persistent: percepts, initially []
percepts ← push(percept,percepts)
return $f_C$(percepts)
```

In each of these agents, the function f is some arbitrary, possibly randomized, function of its inputs with no internal state of its own; the agent program runs on a computer with unbounded memory but finite clock speed. We'll assume also that the environment and its performance measure are computable.

a. Suppose the environment is fully observable, deterministic, discrete, single-agent, and static. For which agents, if any, is it the case that, for *every* such environment, there is *some* way to choose f such that the agent is perfectly rational?

b. Suppose the environment is *partially* observable, deterministic, discrete, single-agent, and static. For which agents, if any, is it the case that, for *every* such environment, there is *some* way to choose f such that the agent is perfectly rational?

c. Suppose the environment is partially observable, *stochastic*, discrete, single-agent, and *dynamic*. For which agents, if any, is it the case that, for *every* such environment, there is *some* way to choose f such that the agent is perfectly rational?

a. B, C. For a fully observable environment, only the current percept is required for an optimal decision. Because the environment is static, computation is not an issue. Note that Agent A cannot make optimal decisions because it always makes the *same* decision (or samples a decision from the same probability distribution), having no internal state.

b. C. Agent B, the reflex agent, cannot always function optimally in a partially observable environment because it ignores previous percepts and therefore fails to take into account relevant information for the current decision.

c. None of the agents can be optimal for an arbitrary dynamic environment, because we can make the environment complex enough to render optimal decisions infeasible for any finite-speed machine.