Agents

Finite-State Automata, Behaviors, Swarms, and such

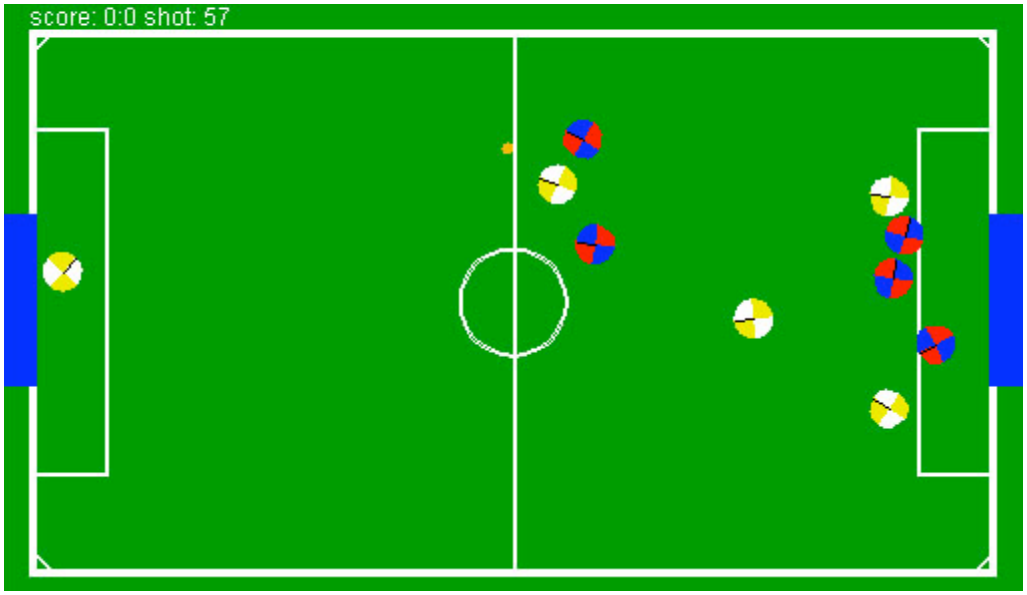
What is an Agent?

- Acting in its World
- Self-contained
- Autonomous
  - Acts in the world in response to information it is receiving from the world
  - No joysticks
- “Agent” is a heavily abused term

...our database agent...

A Soccer Game-playing Agent

- Things move fast. You don't have time to Think Deeply
- There's noise and randomness
- There's an opposing team and you don't know how they play
- There are other players on your team that you need to work with



- How would you program a soccer game-playing program?

Programming an Agent to Play Soccer

- How would you program this? Some possible paradigms:

Event-driven
Your program lies quiet until something “interesting” happens. This triggers a function you have registered just for that particular interesting thing. The function makes your robot do something.

Top-level Loop
Your program loops. Each loop it gathers the current sensor information and runs a function which decides what to do. *Is this different than event-driven?*

Set Play
Your program has one function which performs many actions in sequence.

Multiple Threads
Several simultaneous specialized programs fight over control of your agent.

State-Action Rules

- More or less event-driven. You have functions registered for certain *events* that may occur:

Event	**Function**
If I cannot see the ball	Rotate to the left 10°
If the ball is not directly in front of me	Rotate so that it is
If I am too far from the ball to kick it	Move forward
If I am close enough to the ball	Kick it towards the goal

State-Action Rules

In Python.

```
while(True):
    sensors = getSensors( )
    if (cannotSeeBall( sensors )):
        rotateToLeft(10)
    elif (ballNotDirectlyInFrontOfMe( sensors )):
        centerOnBall( sensors )
    elif (tooFarAway( sensors )):
        moveForward( )
    else:
        kickTowardsGoal( )
```

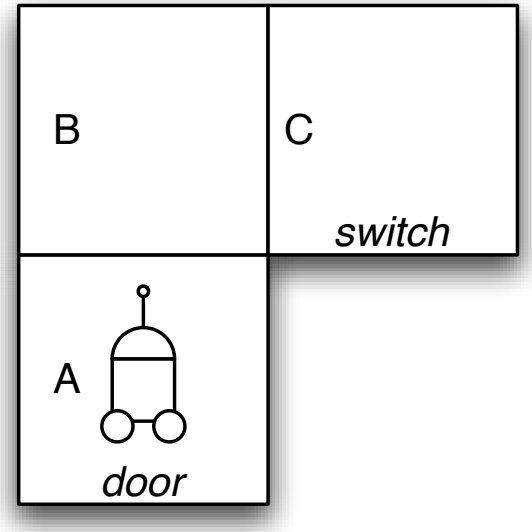
Over- and Under-specification

- What if you have more than one function registered for an event?

- What if an event occurs for which you have no function registered?

A New Scenario

- You have been tasked to program an *evil bloodthirsty patrolling robot game enemy agent* for the latest EA game, “Dancing with the Stars: The Game”



- The bad guy patrols three rooms, A, B, and C. You can’t go from A to C without going through B. He now wants to go out the door.

- In room A there is a door, presently closed. He’s in room A right now. In room C there is a switch. The switch opens the door.

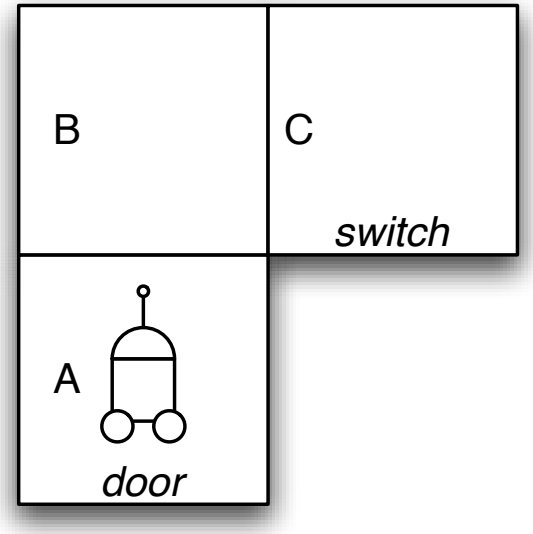
- Your bad guy can only tell if the switch is turned on when he’s in room C. Your bad guy can only tell if the door is open when he’s in room A. *Maybe “The Muppet Show: The Game” would have been better.*

Provide a Set of Rules to Solve This

- I'm waiting.

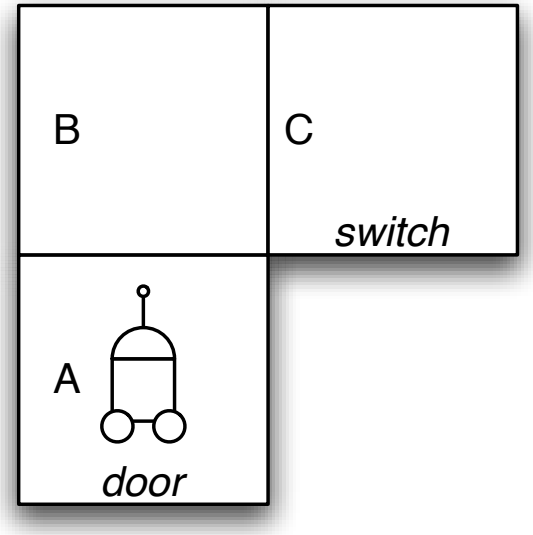
- **Possible events**
  In Room A and Door is Closed
  In Room A and Door is Open
  In Room B
  In Room C and Switch is "Closed"
  In Room C and Switch is "Open"

- **Possible actions**
  Go to Room B
  Go to Room A
  Go to Room C
  Open the Switch
  Go out the Door



Why This Couldn’t Be Done

- What should you do in room B?
- Go to room A?
- Or go to Room C?



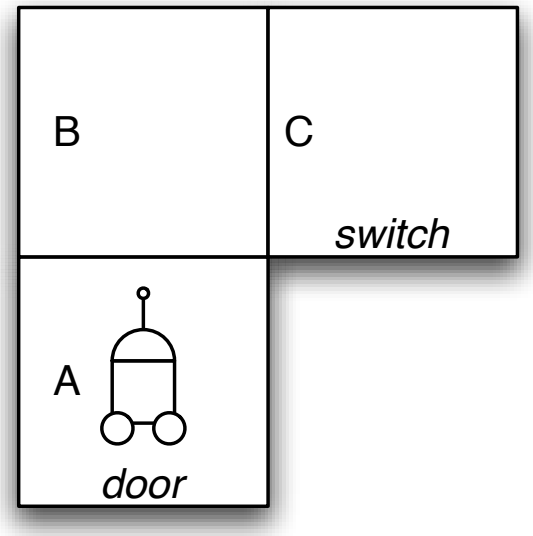
- What ability would have helped you make this decision?

Memory! We Need Memory!

- Let's change the program to a *four*-column table:

  Event Stored Action Change To

- "If **Event** has occurred, and my memory has the following thing **Stored**, then perform the following **Action** and possibly **Change** my memory **To** something else."

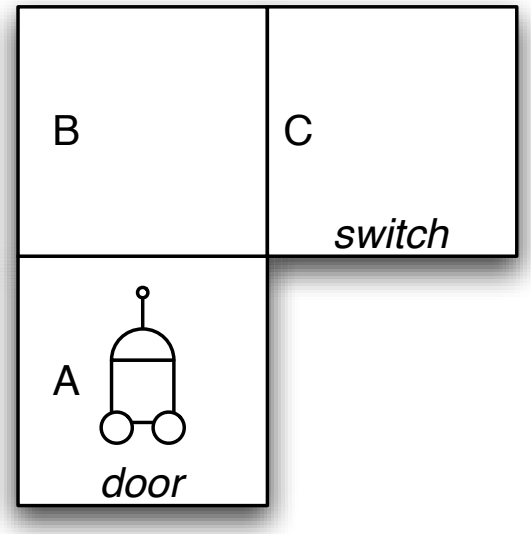


- The item being stored in memory is called the ***(Internal)* State of the Agent**

- Can you write the program now?

Memory! We Need Memory!

Event	Stored	Action	Change To
In A and Door Open	*who cares*	Go out Door	“door open”
In A and Door Closed	*who cares*	Go to B	“door closed”
In B	“door closed”	Go to C	“door closed”
In B	“door open”	Go to A	“door open”
In C and Switch Closed	*who cares*	Open Switch	“door open”
In C and Switch Open	*who cares*	Go to B	“door open”

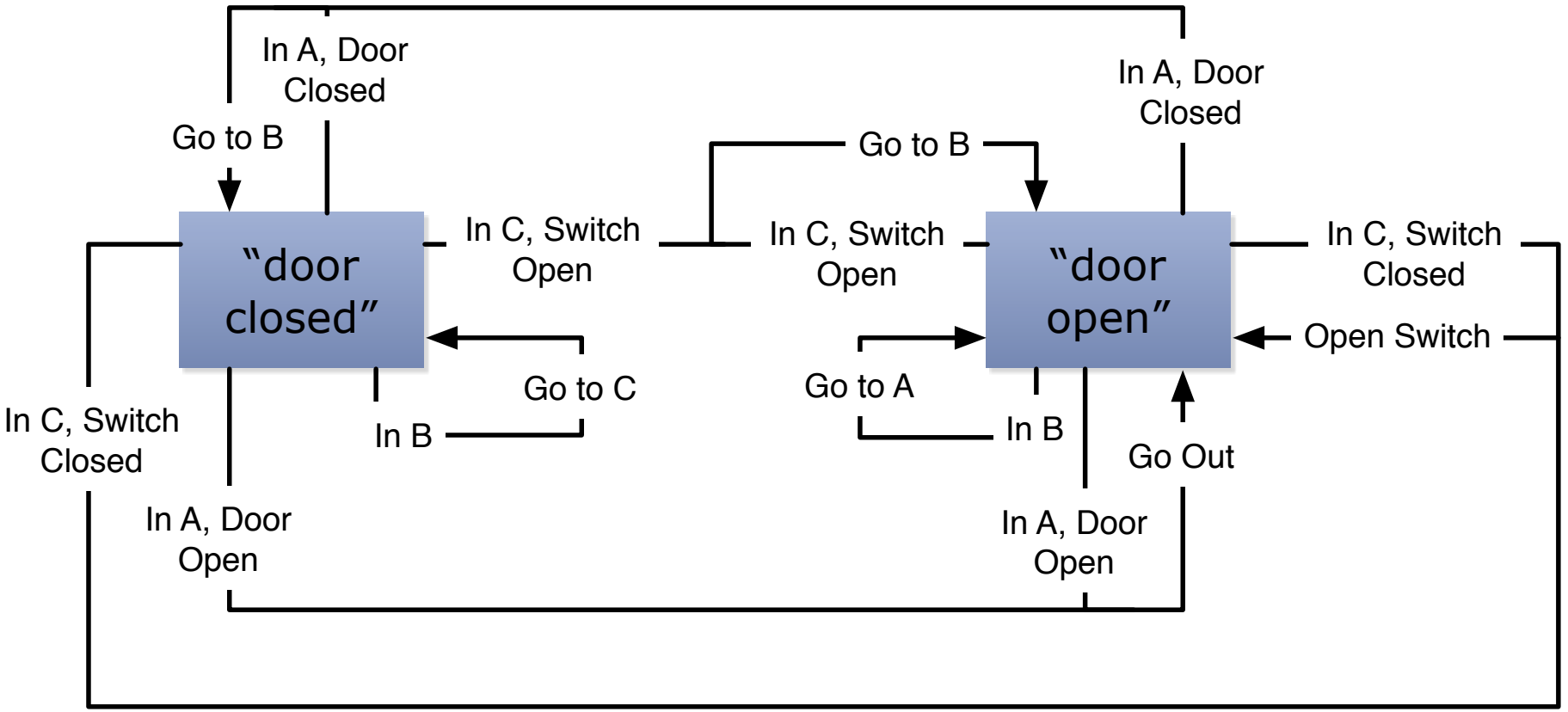


Expanded to all Event / Stored Possibilities

- I replaced each *don't care* with both a "door closed" *and* a "door opened" to have a line for each possibility

Event	**Stored**	**Action**	**Change To**
In A and Door Open	"door closed"	Go out Door	"door open"
In A and Door Open	"door open"	Go out Door	"door open"
In A and Door Closed	"door closed"	Go to B	"door closed"
In A and Door Closed	"door open"	Go to B	"door closed"
In B	"door closed"	Go to C	"door closed"
In B	"door open"	Go to A	"door open"
In C and Switch Open	"door closed"	Go to B	"door open"
In C and Switch Open	"door open"	Go to B	"door open"
In C and Switch Closed	"door closed"	Open Switch	"door open"
In C and Switch Closed	"door open"	Open Switch	"door open"

A Two-State Finite-State Automata



In Python (simplified)

```
state = “door open”
while(True):
    sensors = getSensors( )
    if room(sensors) == “A” and door(sensors) == “open”:
        goOut( )
        state = “door open”
    elif room(sensors) == “A” and door(sensors) == “closed”:
        goTo(“B”)
        state = “door closed”
    elif room(sensors) == “B” and state == “door open”
        goTo(“A”)
    elif room(sensors) == “B” and state == “door closed”
        goTo(“C”)
    elif room(sensors) == “C” and switch(sensors) == “closed”:
        openSwitch( )
        state = “door open”
    else:  // “c” and “open”
        goTo(“B”)
```

Another Example: Collecting Cans

- **Your robot can sense:**
  A can is in front of him
  That that a can is within gripping distance
  That he has a can in his gripper

- **Your robot can:**
  Move forward
  Turn randomly
  Rotate to the left a little bit
  Grip
  Release

- **Your robot wishes to:**
  Collect all the cans in the room into one big pile

What's the program?

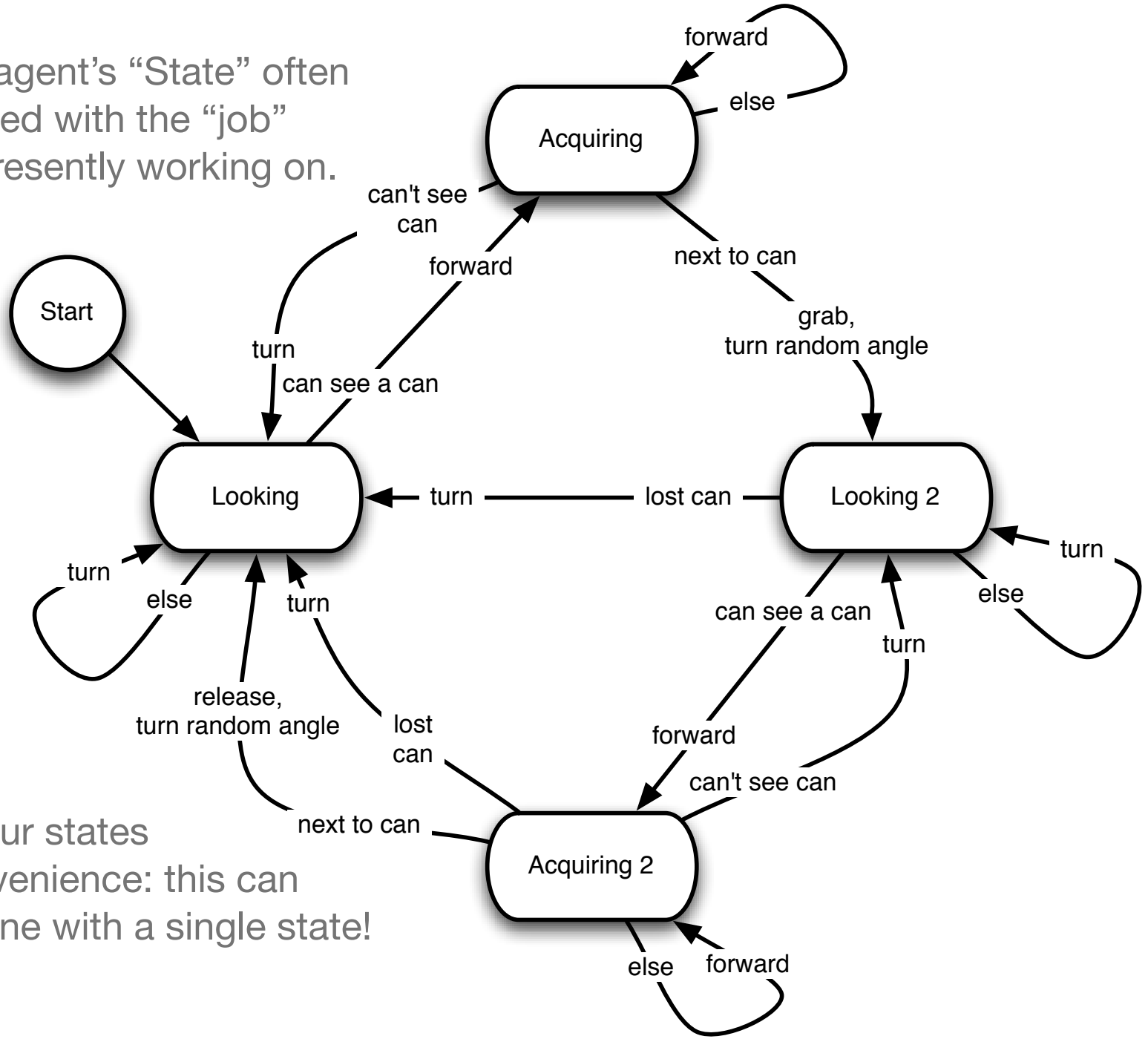
Another Example: Collecting Cans

- **The general idea:**

- Turn until I see a can.
  Home in on the can
  Grab the can
  Look for another can
  Home in on that can
  Release the first can next to it
  Rotate a bit randomly

- How would you write this as a set of simple rules?

- Note that the agent's "State" often is closely related with the "job" the agent is presently working on.



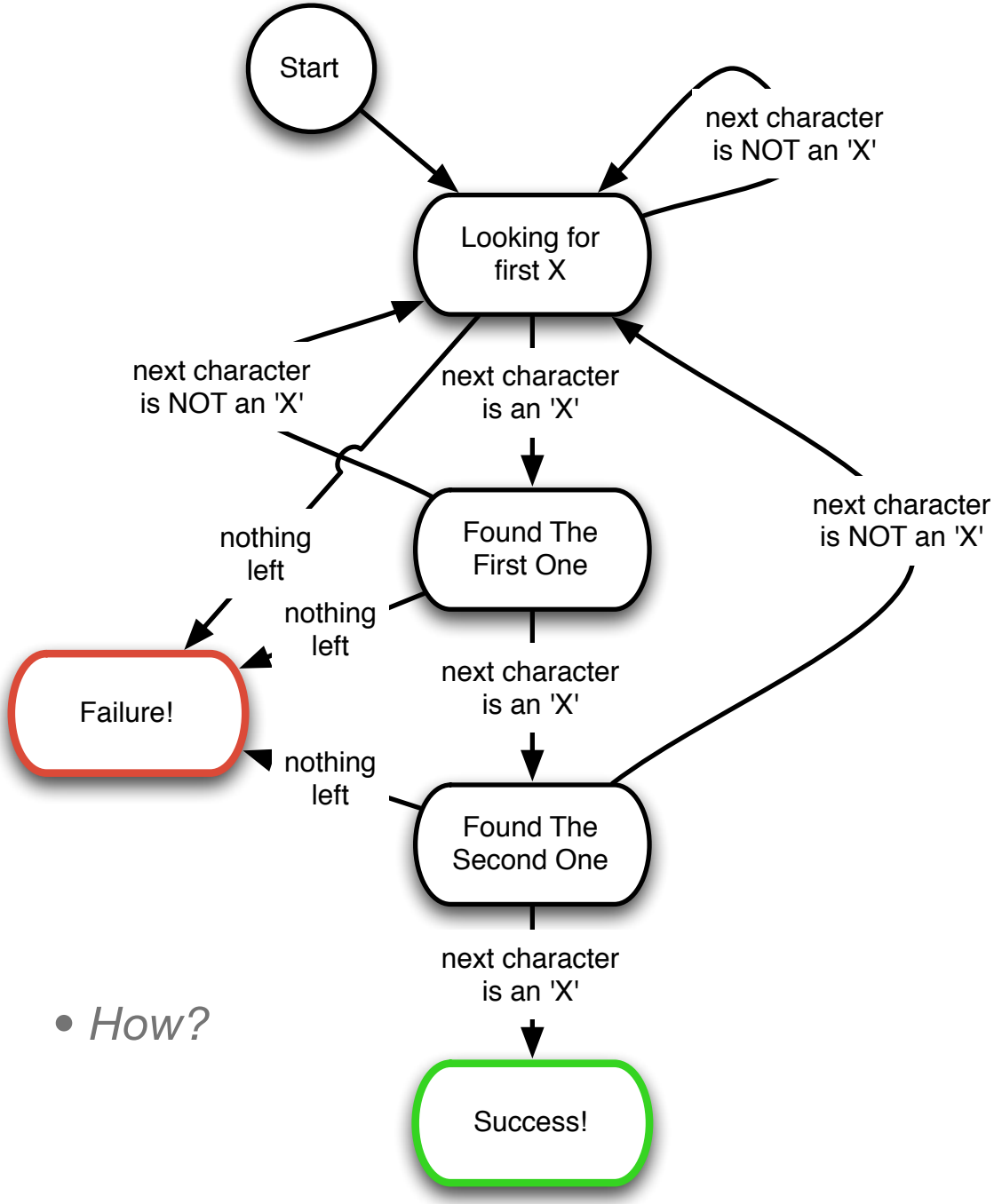
- By the way, four states are just a convenience: this can actually be done with a single state!

Other Uses

- Finite-State Automata aren't just used to move simple robot or game agents around. They're often used to construct machines which detect a sequence:

- ***Example:*** Given a sequence of letters, does the sequence contain three X's in a row?

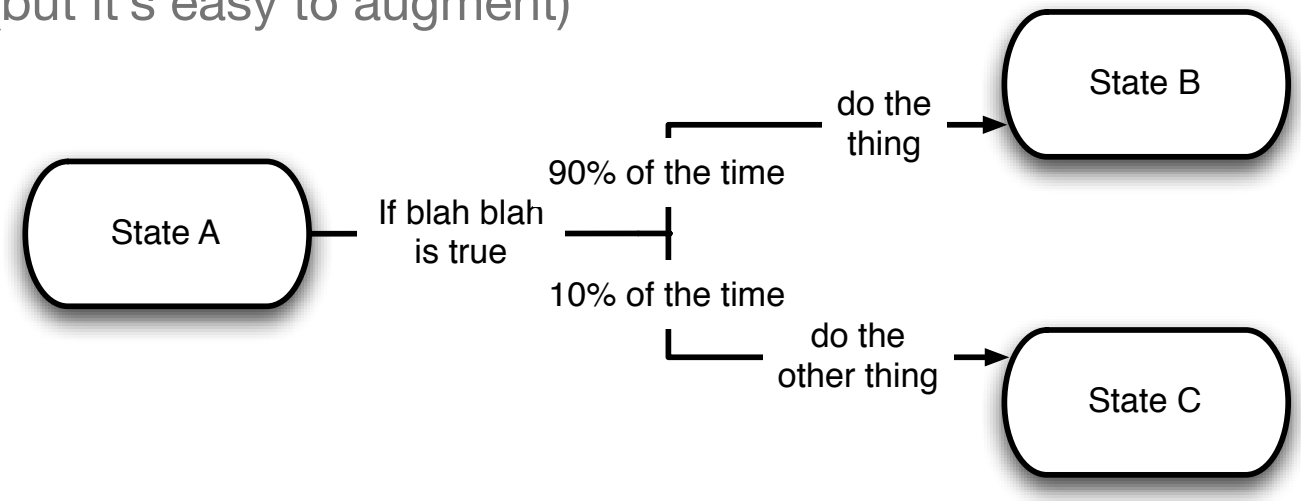
- ***Example:*** a coin-operated vending machine uses a Finite-state Automaton to determine how much money you've put in.



- *How?*

What Can’t Finite-State Automata Do?

- Stuff that requires arbitrarily large amount of memory
- Stuff that requires recursion
- Simultaneous Actions (but it’s easy to augment)
- Probabilistic Actions (but it’s easy to augment)



Distributed and Multiagent Systems

- A **distributed system** is commonly one where you have available to you multiple computational resources (computers say) and you're trying to figure out how to get them to perform some task together.

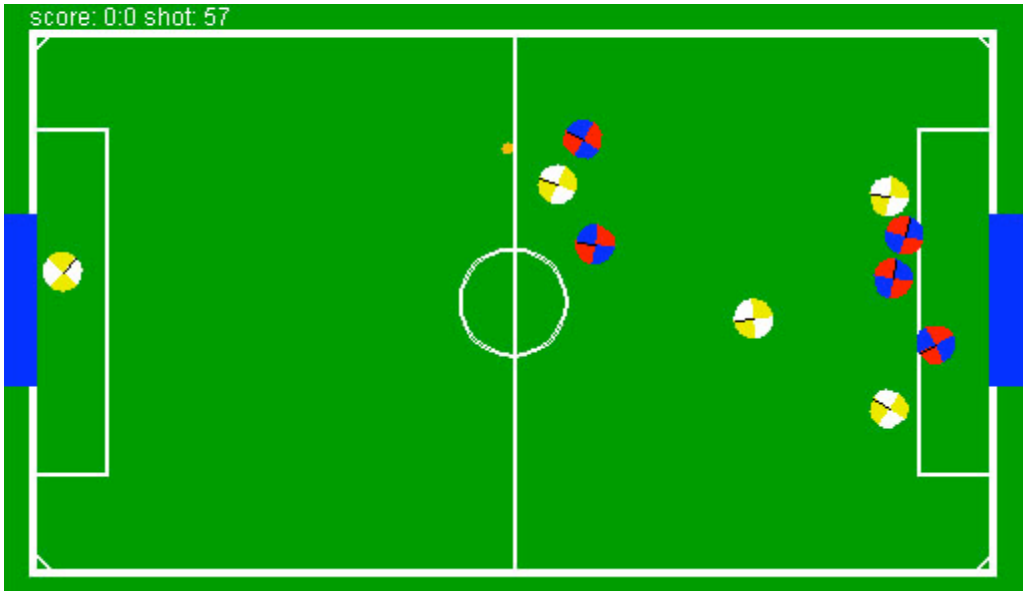
    - Parallel or cluster computing
      Distributed ad-hoc wireless networks
      Distributed sensor networks

- A **multiagent system** is a distributed system where the resources are **agents** who don't know enough about what the other agents are doing to work in lock-step. How can they avoid stepping on each others' toes?

    - Multirobotics, game-playing agents
      Web agents

Multiple Soccer Players

- Each of your soccer players is an agent.
- Your team of agents opposes *another* team of agents.
- Each agent must determine what he has to do to best help his team at any given time.



- **Example:** my teammate has the ball. Should I get open so he can kick to me? Should I get to a pre-agreed location? Should I stay back and defend my goal?

The Prisoner's Dilemma

- A two-player game. You are both in jail. Each of you can *rat out the other person* **(Defect)** or to *refuse to rat him out* **(Cooperate).**
- You don't know what the other person will do, and you can't talk to him.
- If you both **Cooperate**, you are both released after 1 year (score -1 each).
- If either of you **Cooperates** and the other **Defects**, the defector is released *now* (score 0) and the cooperator goes to prison for 3 year (score -3).
- If both of you **Defect** (rat on each other!) you both go to prison for 10 years (score -10).
- **Now: imagine you keep landing in jail! What if you repeatedly play this game?** Is there a strategy you should follow to minimize your years in jail?

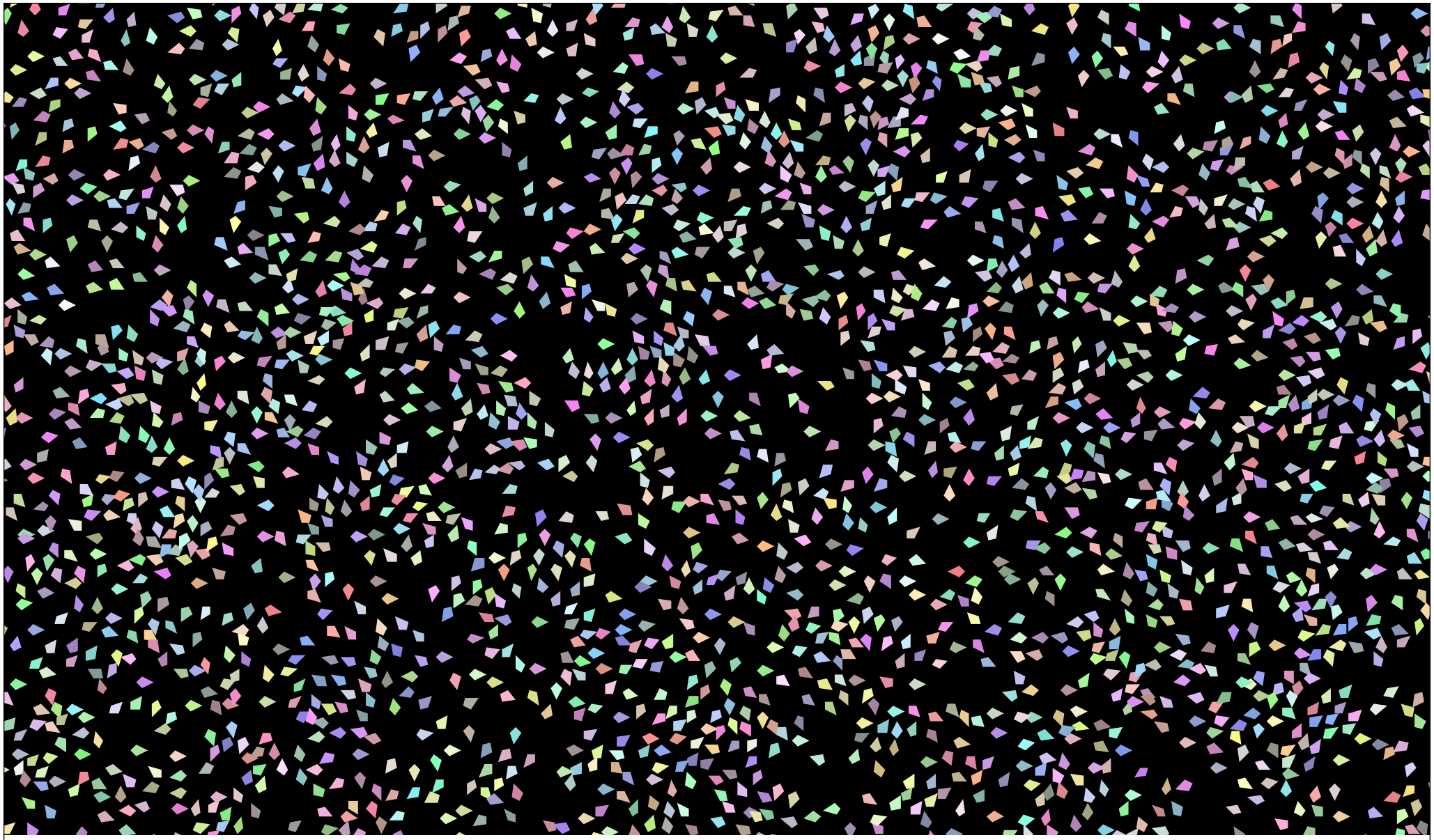
In Game Theory Form

		Agent 2: Cooperate	Agent 2: Defect
Agent 1	Cooperate	**1, 1**	**0, -3**
	Defect	**-3, 0**	**-10, -10**

yes,
it's a matrix

Tit-for-Tat

- The first time, cooperate.
- Then do whatever my partner did last time.
- What if there is noise in the system? That is, sometimes, rarely, you accidentally do the opposite of what you had intended?



Swarms

What if there are **lots of agents?**

Flocking

- You have a **swarm of thousands of simulated agents** and you'd like them to produce a **realistic flocking behavior.**

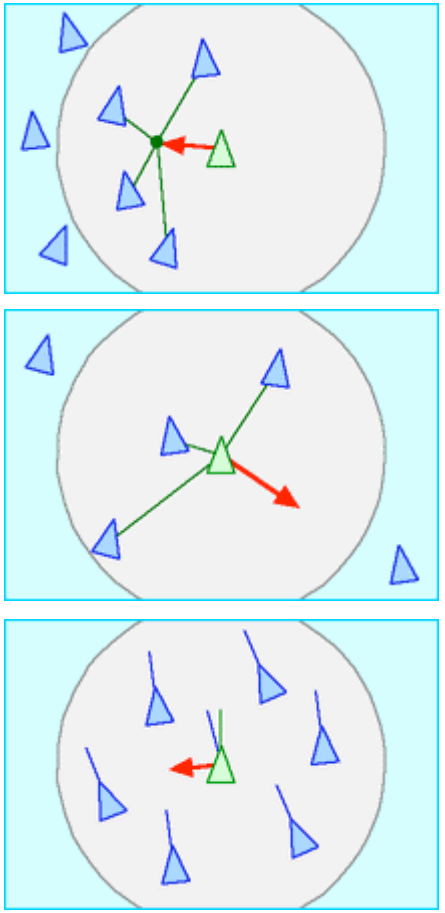
  Who would ever want that?

- How might we do this?

- How might we do this where each agent determines what to do on his own?

The “Boids” Flocking Algorithm: by Craig Reynolds

- Each agent gathers the locations of agents in his neighborhood, then uses these locations to compute some vectors:
  - **Cohesion Vector**: a vector towards the middle of those agents (“I want to go *with them!”)*
  - **Avoidance Vector:** a vector away from the agents: closer agents have a stronger effect (“I need space!”)
  - **Consistency Vector:** a vector in the direction everyone else is going.
  - ***Also...***
    Momentum Vector: the direction I went last time.
    Random Vector: a random vector
- Add up these five vectors, and that’s the direction I’ll go.



Ants and Pheromones

- Ands build up pheromone deposits along trails to work collectively as agents.

- We can simulate **foraging behaviors** in simulation, using:

  - **Two pheromones**

  - Each ant follows the same **two-state finite state automaton** and a **simple equation** for updating pheromone deposits.