Nature Inspired Computing

K.Dimopoulos

Angular motion

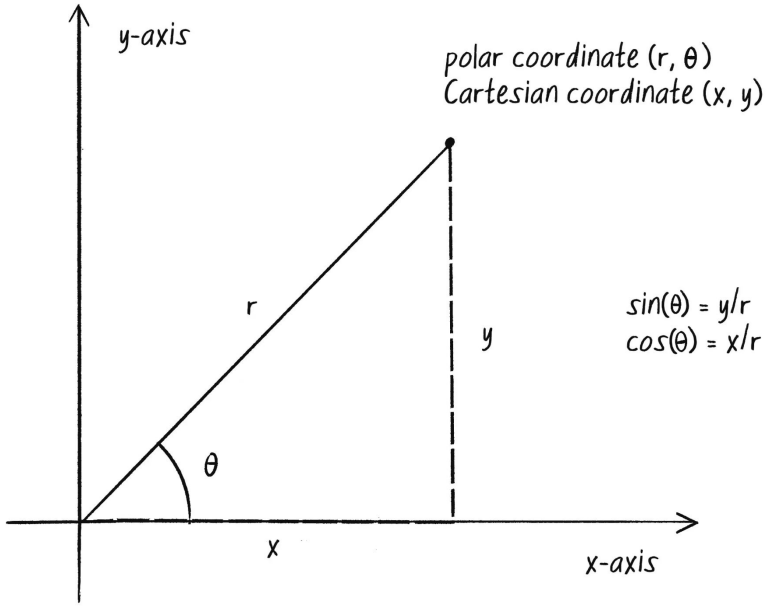
The duality of a vector

A vector can be seen as an arrow starting from the base of the coordinate system, with its point at some x,y coordinates. This coordinate system is known as the **Cartesian coordinate system**.

However it is possible to describe a vector as an angle (θ) and a magnitude (r) instead. This is known as the **polar coordinate system**.

There is a relationship between the polar (r, θ) and Cartesian (x,y) coordinates:

$x = r * \cos(\theta)$, $y = r * \sin(\theta)$



Polar to Cartesian transformation in P5.JS and vice versa

Given the polar coordinates (r, theta) it is easy to calculate the Cartesian coordinates (x, y):

```
let x = r * cos(theta);

let y = r * sin(theta);
```

To calculate it as a vector directly use the vector static function `fromAngle()`.

```
let position = p5.Vector.fromAngle(theta); //this is a unit vector

position.mult(r);
```

To get the polar coordinates from a vector:

```
let theta = this.velocity.heading();

let r = position.Mag();
```

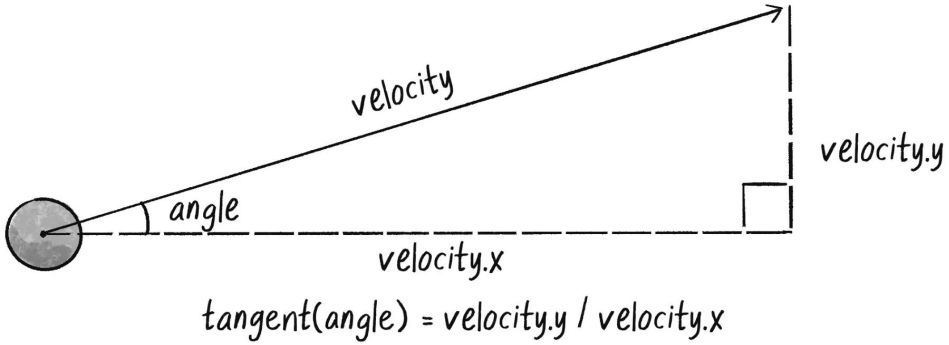
Heading of a vector

The direction that an object moves is dictated by the object's velocity vector.

The heading the object is the clockwise angle of the vector starting from the x axis. This can be calculated using the `atan2()` function (ark-of-tangent) , which is the reverse of `tan()`.

However P5.JS has a better way of calculating the angle, using the `heading()` function of a vector:

```
let angle = this.velocity.heading();
```



Angular motion

Another term for rotation is **angular motion**—that is, motion about an angle.

- Just as linear motion can be described in terms of velocity—the rate at which an object's position changes over time—angular motion can be described in terms of angular velocity—the rate at which an object's angle changes over time. By extension, angular acceleration describes changes in an object's angular velocity.

Like in linear motion velocity is updated by adding acceleration and position is updated by adding velocity, in angular motion angular velocity is updated by adding angular acceleration and angular position is updated by adding angular velocity.

- In fact, these angular motion formulas are simpler than their linear motion equivalents since the angle here is a scalar quantity (a single number), not a vector! This is because in 2D space, there's one axis of rotation; in 3D space, the angle would become a vector.

Angle between two vectors

The heading() static function of Vector calculates the angle between a Vector and the horizontal axis x.
If we want to calculate the angle between two vectors (v0 and v1) we can use the `angleBetween()` function of a vector.

This is the clockwise angle in radians
Remember:

- `angleMode(DEGREES)` use degrees instead of rads
- `angleMode(RADIANS)` use rads instead of degrees
- `radians(theta)` returns the angle theta expressed in degrees to radians
- `degrees(theta)` returns the angle theta expressed in radians to degrees.

angle between vectors

```
// Create p5.Vector objects.

  let v0 = createVector(1, 0);

  let v1 = createVector(0, 1);



  // Prints "1.570..." to the console.

  print(v0.angleBetween(v1));



  // Prints "-1.570..." to the console.

  print(v1.angleBetween(v0));
```

Primordial Particle Systems

A new system

In 2016 an article appear in *Scientific Reports*, titled "How a life-like system emerges from a simplistic particle motion law".

The authors demonstrated that from a simple particle motion law, structures of particles could emerge, that grew by adding to their structure other particles, multiplied by division of the structure, and generally exhibited a life cycle.

The particles are very simple. Every particle has a position and a constant velocity. However at each step, the velocity heading is updated based on counting how many particles are to the left and to the right. The particle has an innate inclination to turn toward the side that has the biggest number of particles within a radius.

The law of motion

At each step the particle wants to turn to the right by 180 degrees. In addition to that it wants to turn to the side with the most particles with a radius by an additional 17 degrees for every particle in the radius:

$$\Delta\varphi = \alpha - \beta * sign(R - L)*N$$

$\Delta\varphi$ = the angle to rotate at this t (a positive angle turns the particle clockwise)

α = 180 degrees

β = 17 degrees

R = total number of particles on the right

L = total number of particles on the left

$sign(R - L)$ = +1 if R > L, -1 otherwise

N = the total number of particles within a radius (R+L)

Color coding

In addition to the particles properties at each time the particle is colored according to the number of particles within its radius (r=5 or r=1 to 3) according to the rule:

if $15 < N_{r=5} \leq 35$ blue

else if $N_{r=5} > 35$ yellow

else if $13 \leq N_{r=5} \leq 15$ brown

else if $N_{r=1.3} > 15$ magenta

else green

Other considerations

In order for this to work we need to have a high concentration of particles:

density of 0.08 per space unit.

- That means that in a 500 by 500 canvas we need
- 500 x 500 x 0.8 = 20,000 particles!
- This number will grint our simulation to a holt as at each step we need to perform 19,999 operations/comparisons to calculate the $\Delta\varphi$.
- So the canvas needs to be small.

A toroidal type of borders is required.

The life cycle

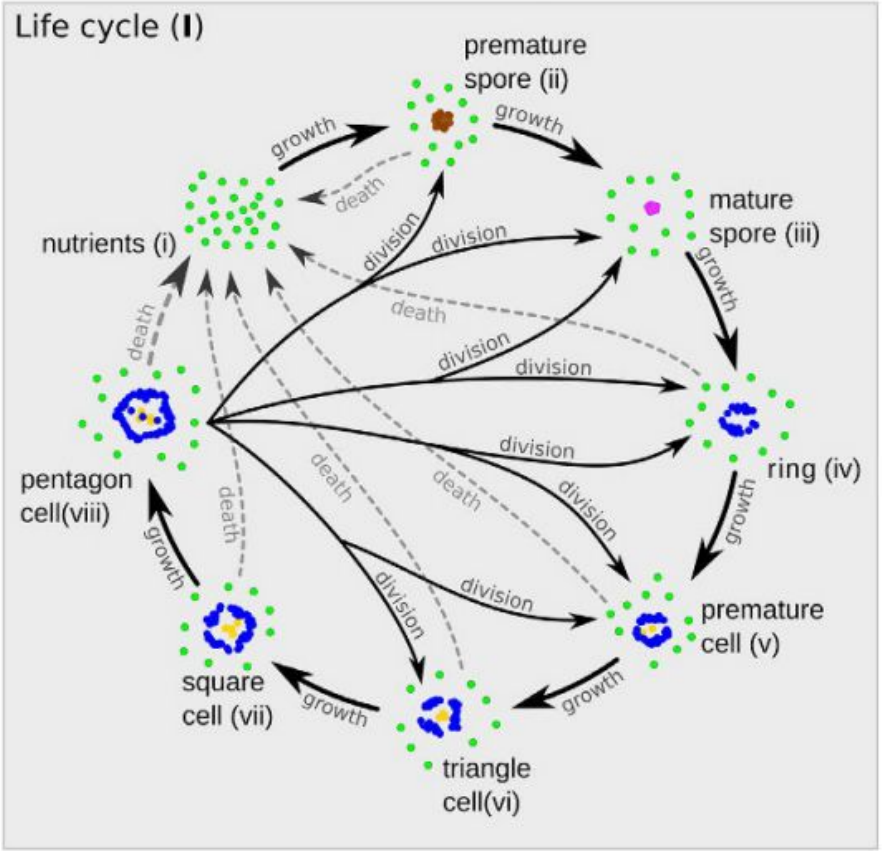
Initially spores are created. Then spores mature and become a ring structure that resembles a cell membrane.

the cell grows on several stages first by obtaining a nucleus, which keeps on growing

At the end the cell divides to either smaller cells, or creating new spores

Cells can also dissolve, thus dying.

Different combinations of α and β values yield different worlds



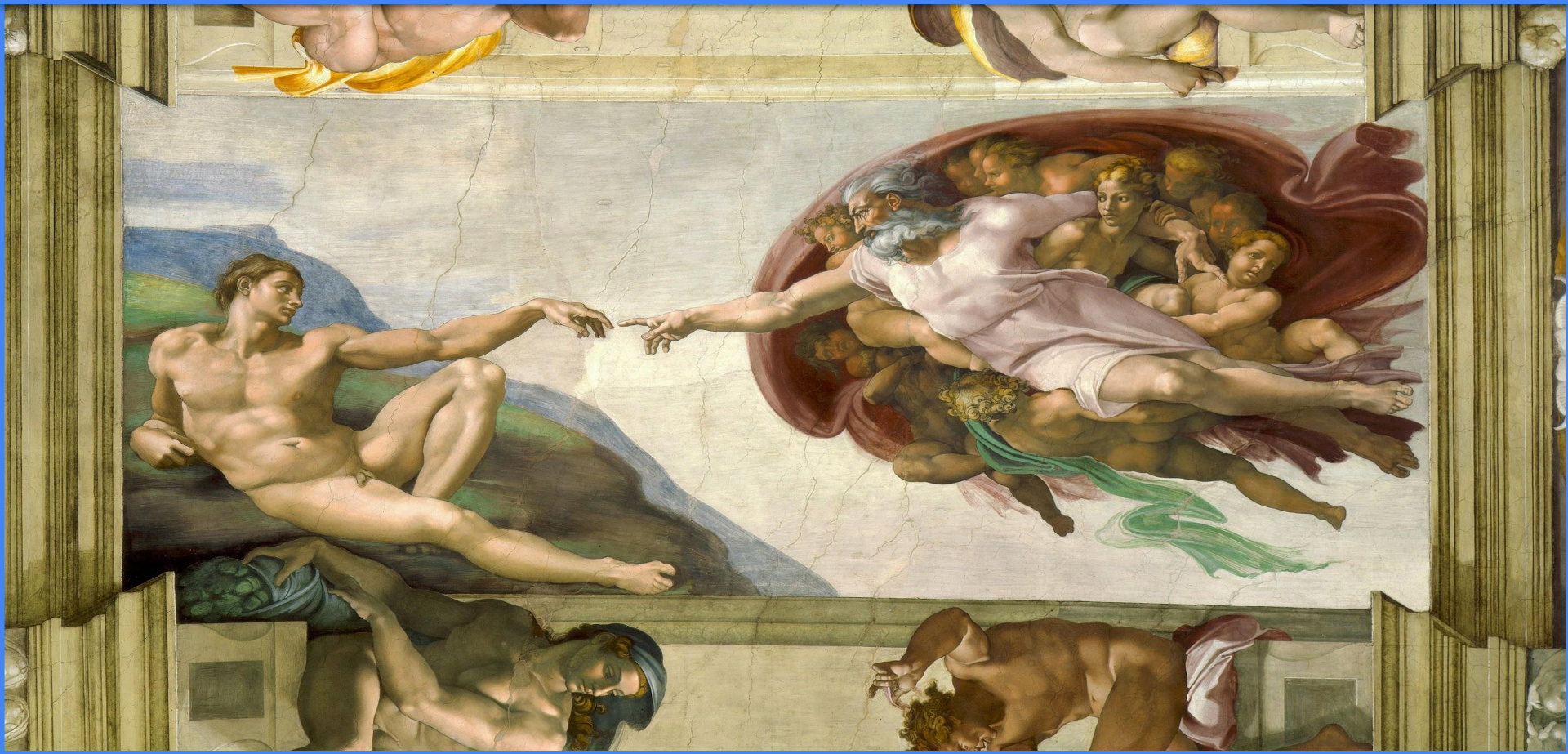
Learn more about PPS

Schmickl, T., Stefanec, M. and Crailsheim, K. (2016). How a life-like system emerges from a simplistic particle motion law. Scientific Reports, 6, 37969. DOI: 10.1038/srep37969

How life emerges from a simple particle motion law: Introducing the Primordial Particle System

Autonomous Agents

Breathing life to our simulation



The idea of agency

What does a travel agent and James Bond have in common?

- They are entities that make their own choices about how to act without any external influence or direct guidance, in order to achieve a goal.

This the the definition of an **autonomous agent**.

We will be using this in our programs to create autonomous programs that solve a bigger problem, but cooperating or competing.

There are many applications of autonomous agents today in all aspects of software development.

Components of an autonomous agent:

1. **An autonomous agent has a limited ability to perceive environment.**
    - It's also crucial that we consider the word limited here. An insect, for example, may only be aware of the sights and smells that immediately surround it.
2. **An autonomous agent processes the information from its environment and calculates an action.**
    - This will be the easy part for us, as the action is a force. The environment might tell the agent that there's a big scary-looking shark swimming right at it, and the action will be a powerful force in the opposite direction.
3. **An autonomous agent has no leader.**
    - This third principle is something we care a little less about. We will look at designing collections of autonomous agents that exhibit the properties of complex systems— intelligent and structured group dynamics that emerge not from a leader, but from the local interactions of the elements themselves.

Agent Properties

Properties of agents: Social Ability

Agents interact with each other and humans.

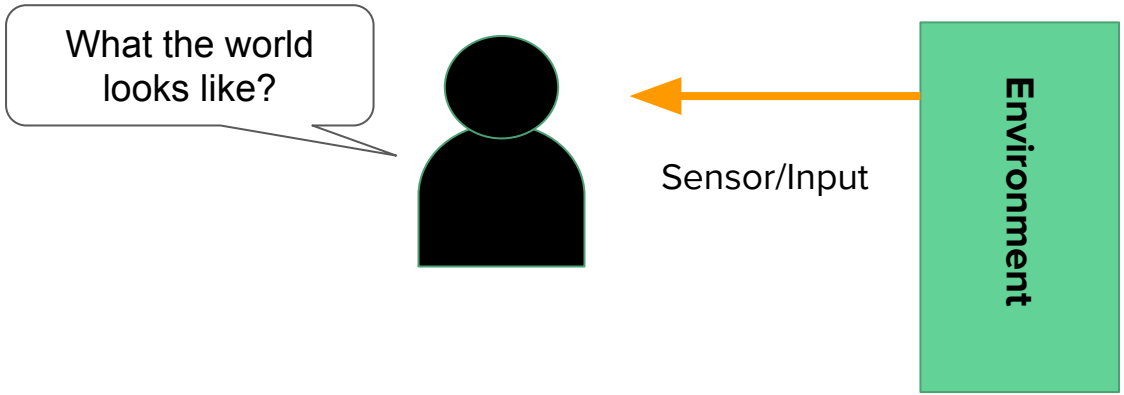
- *Interaction is not simply communication.*
- *Social communication is not trivial exchange of messages.*
- *It is about understanding each other in order to collaborate or negotiate or compete with others.*
- *Interaction is required to resolve conflicts that appear when agents trying to achieve their individual goals.*

More properties

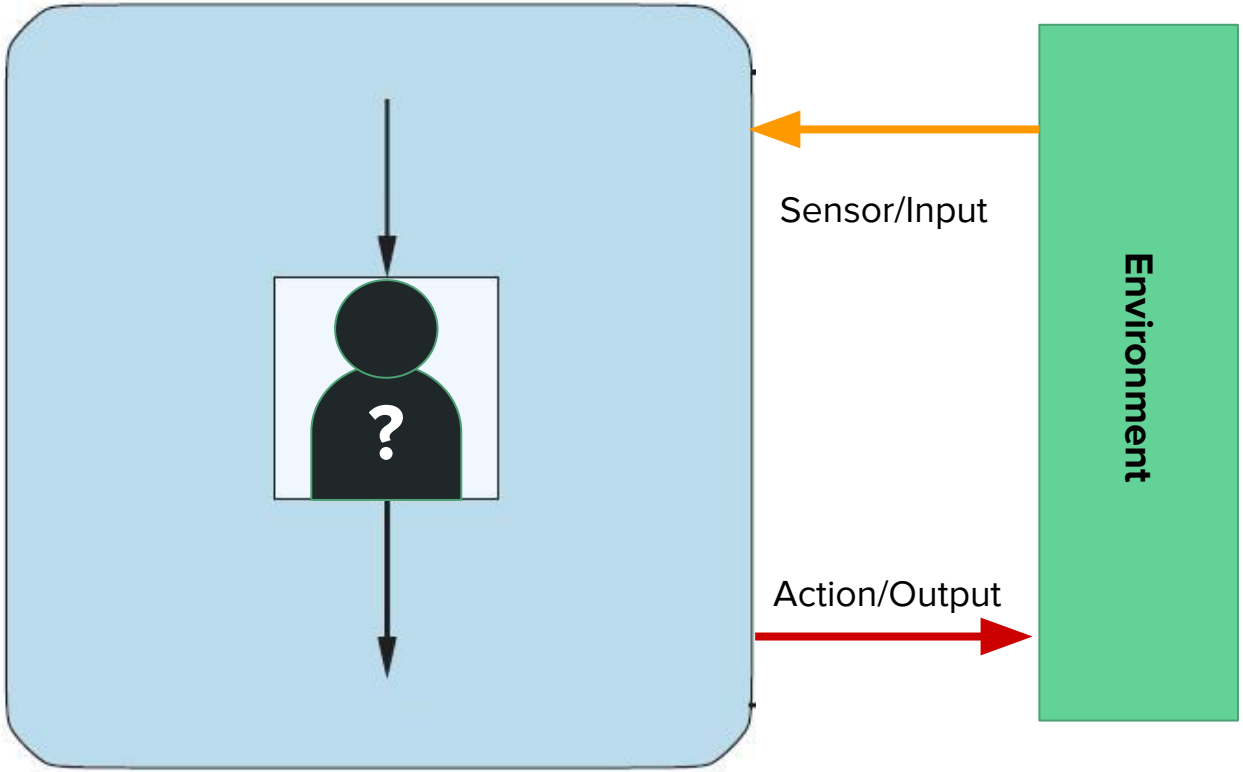
- **Mobility**: an agent is not static
- **Adaptivity**: an agent can be continuously adapted to its environment
- **Veracity**: an agent does not knowingly communication false information
- **Benevolence**: an agent does not have conflicting goals
- **Rationality**: an agent acts in order to achieve its goals

Agent Sensoring: The Transduction Problem

The **transduction problem** is the how to translate the real world sensory input to an accurate and adequate symbolic representation of the world, in time for that description to be useful. We will not discuss this issue during the module and we will make the assumption that this representation is already accurate.

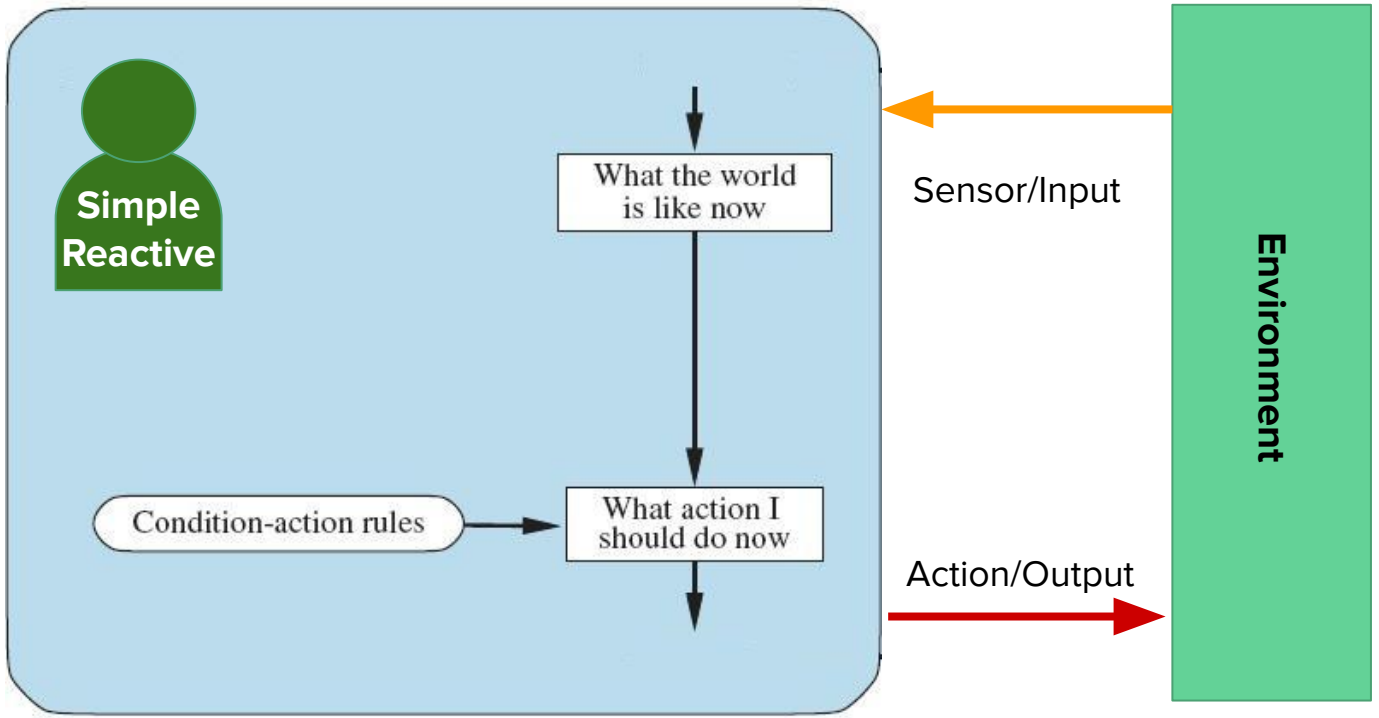


Agent Function: mapping percepts to actions



Types of Agents

Simple Reactive Agent

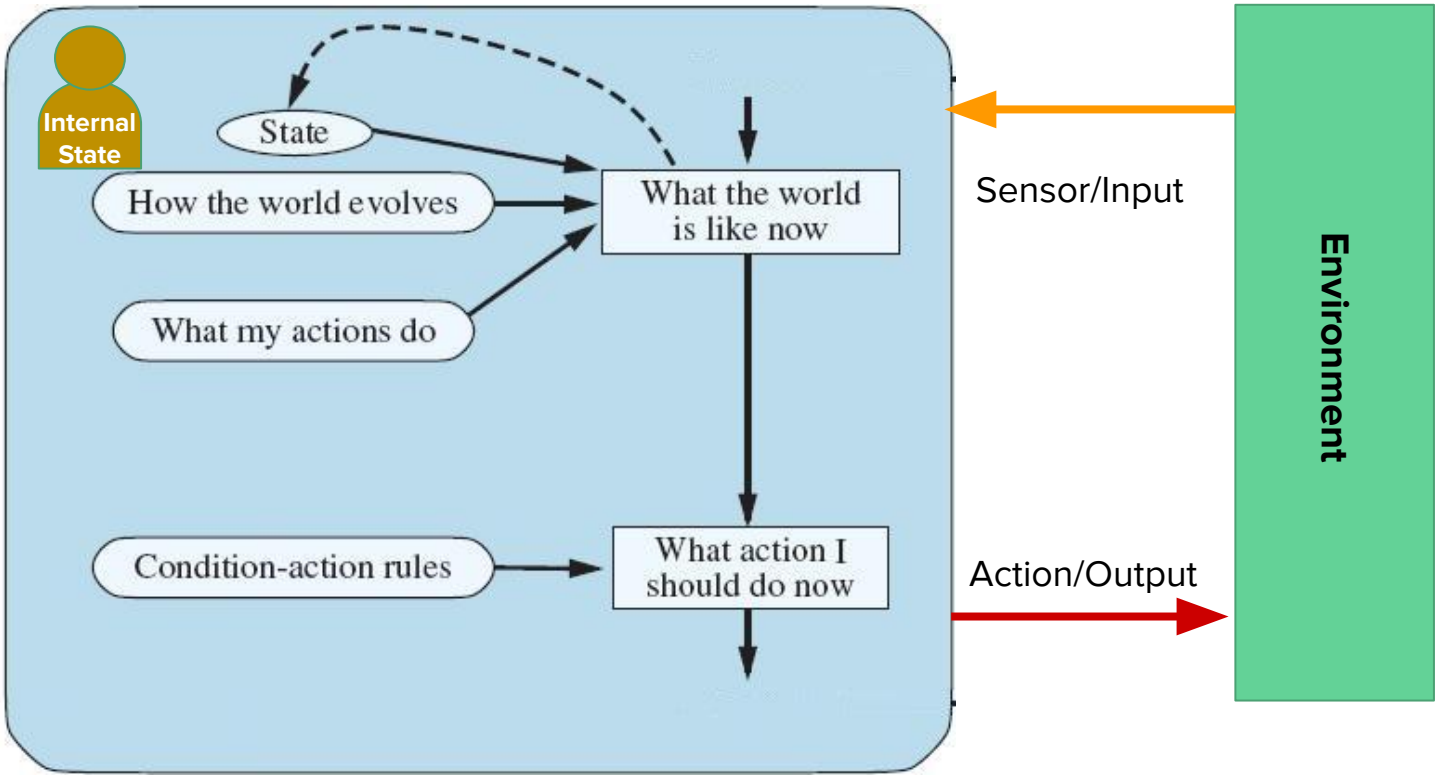


Reactive Agent Algorithm

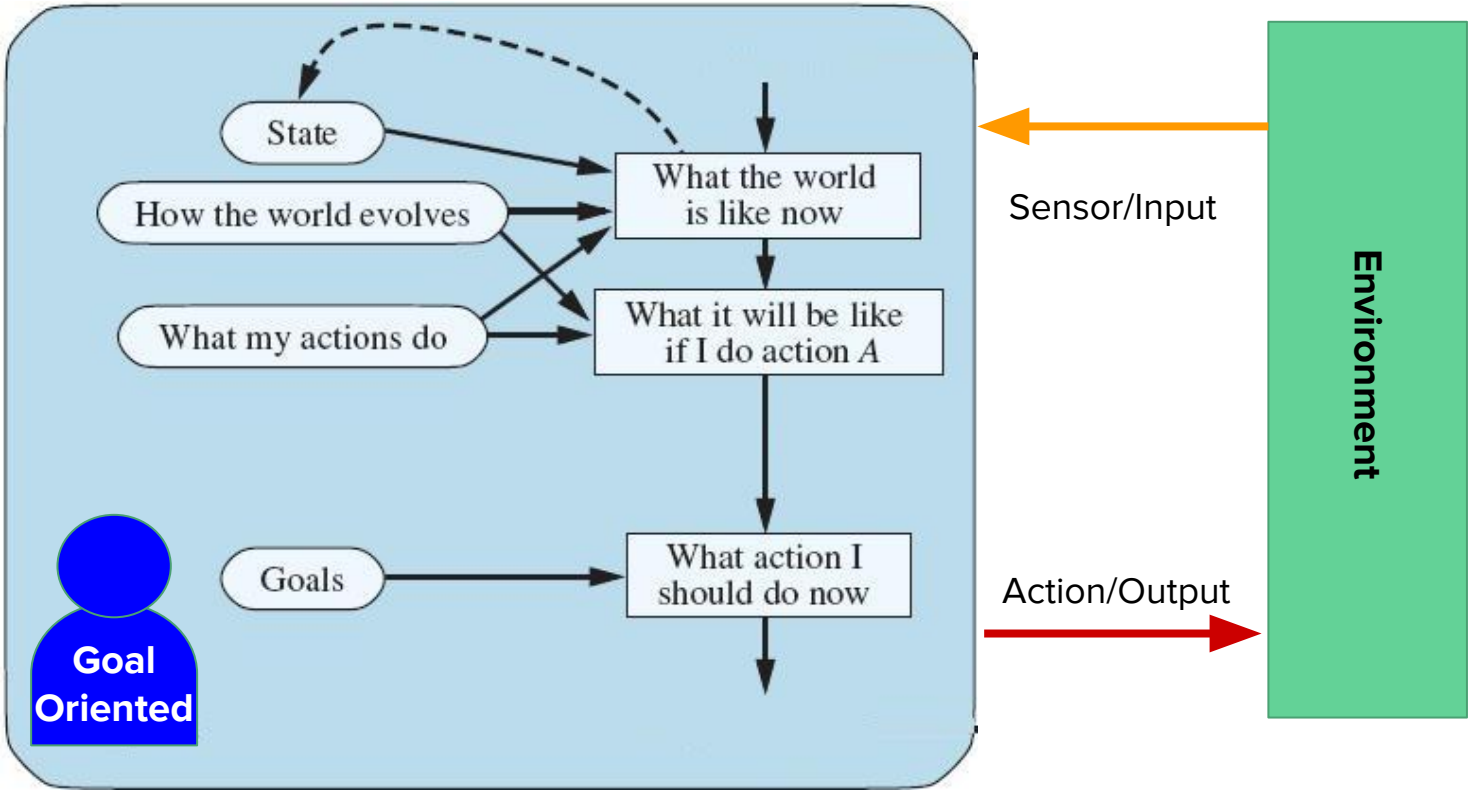
```
Function SimpleReactiveAgent(environmentstate)
    returns action
Static: set of condition-action rules

percept ← see(environmentstate)
rule ← match(percept, rules)
action ← apply(rule)
return action
```

Agent with internal state



Goal-oriented Agent

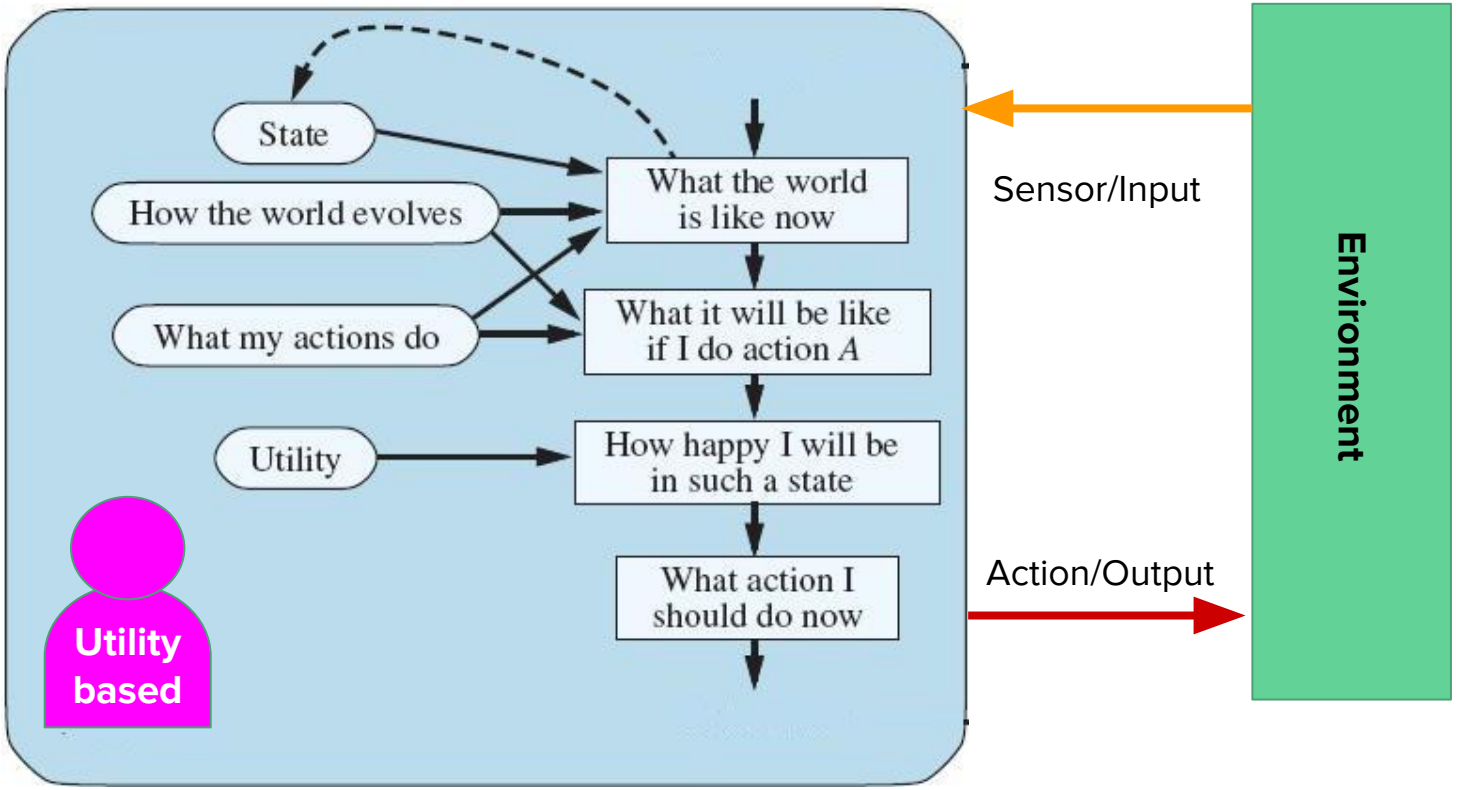


Goal-oriented Agent Algorithm

```
Function GoalBasedAgents(environmentstate)
    returns action
Static:   goal,  internalstate
    set of condition-action rules

percept ← see(environmentstate)
internalstate ← update(internalstate, percept)
rule ← match(goal, internalstate, rules)
action ← apply(rule)
return action
```

Utility-based Agent



Learning Agent