An introduction to **NetLogo**

University of Bologna

nicolas.lazzari2@studio.unibo.it

References

- https://ccl.northwestern.edu/netlogo/docs/
- https://ccl.northwestern.edu/netlogo/docs/dictionary.html

The slides are based on the slides from previous tutors of the course.

NetLogo?

NetLogo is a **programmable modeling environment** for simulating natural and social phenomena, based on Logo by Seymour Papert.

It is designed to model **complex system** development over time.

Why NetLogo?

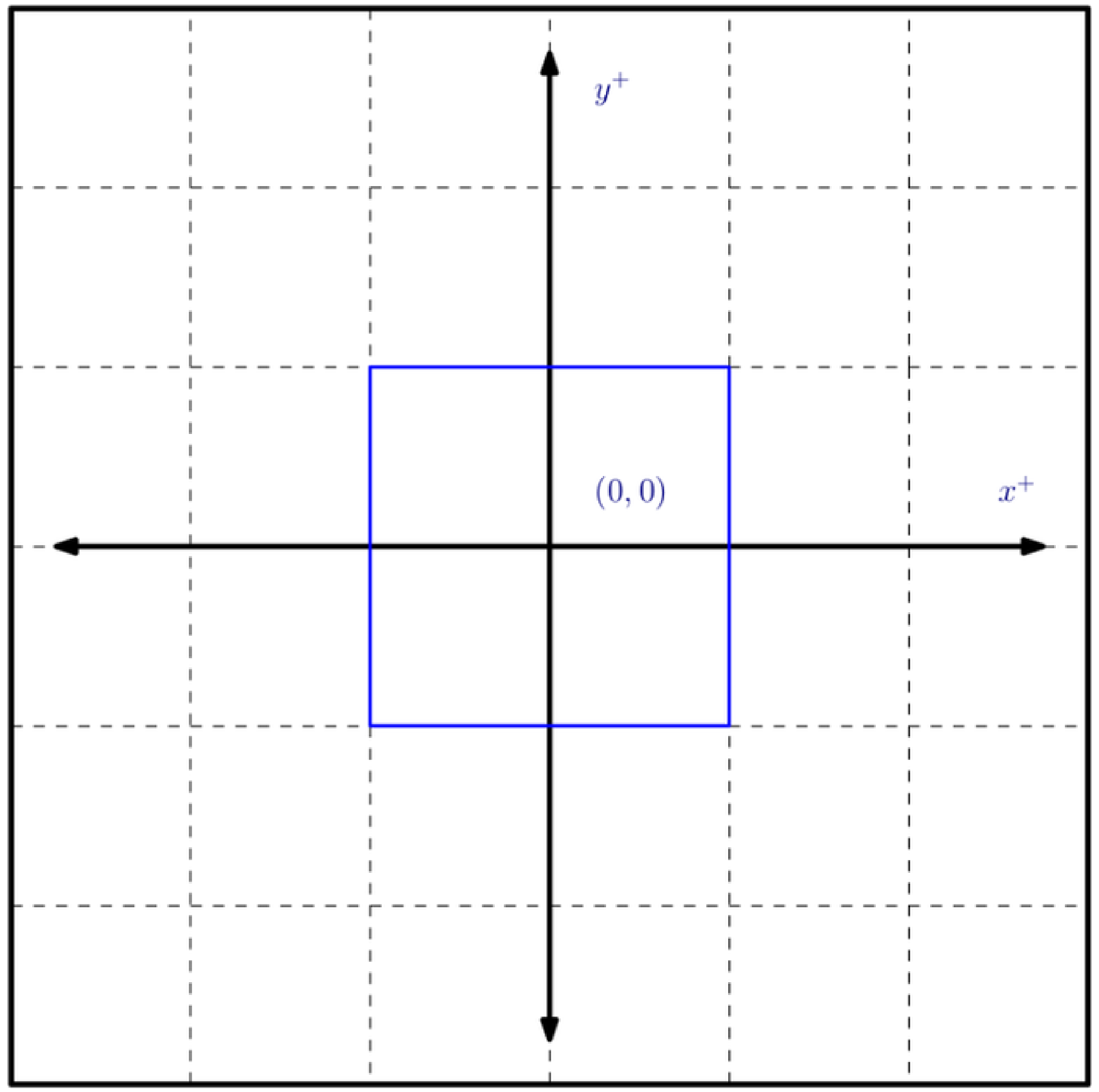
Extensive documentation and tutorials.

Large collection of pre-written simulations on Biology, medicine, physics, chemistry and more.

Free and open source.

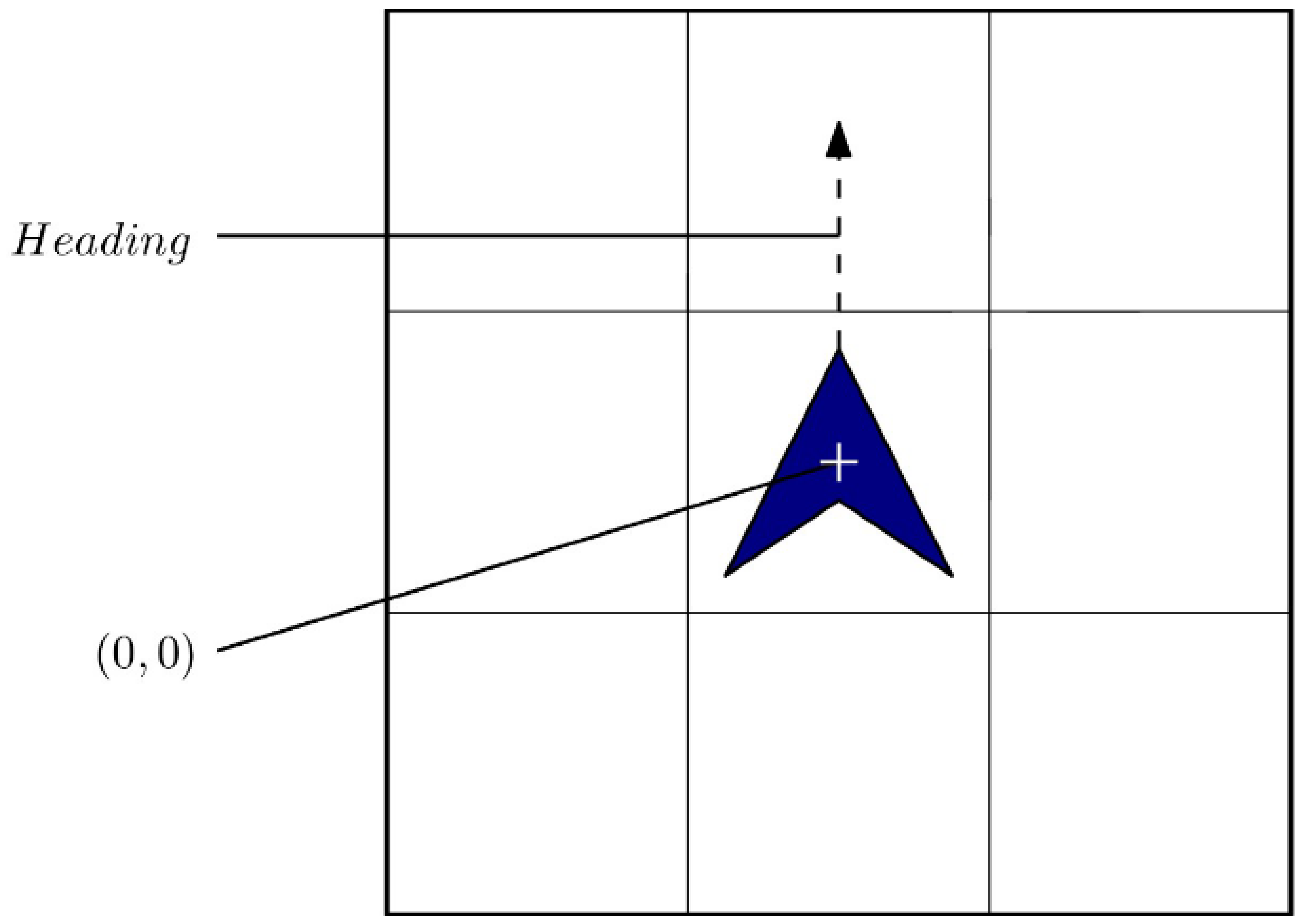
Runs on JVM.

The environment



The whole world is a **discrete grid.**
Each basic region is called a **patch.**

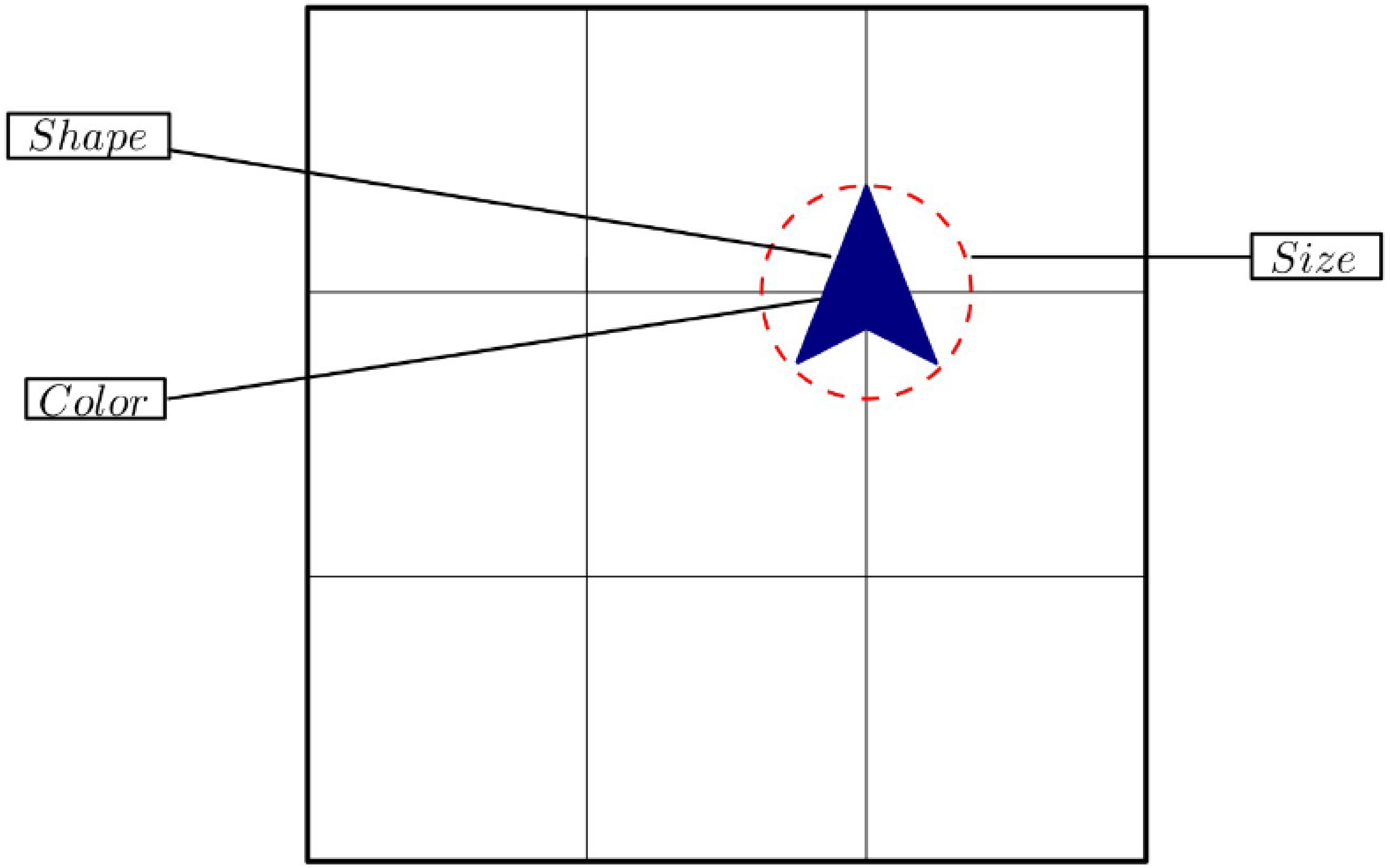
The agents



The environment is composed of agents called **turtles** that can independently move.

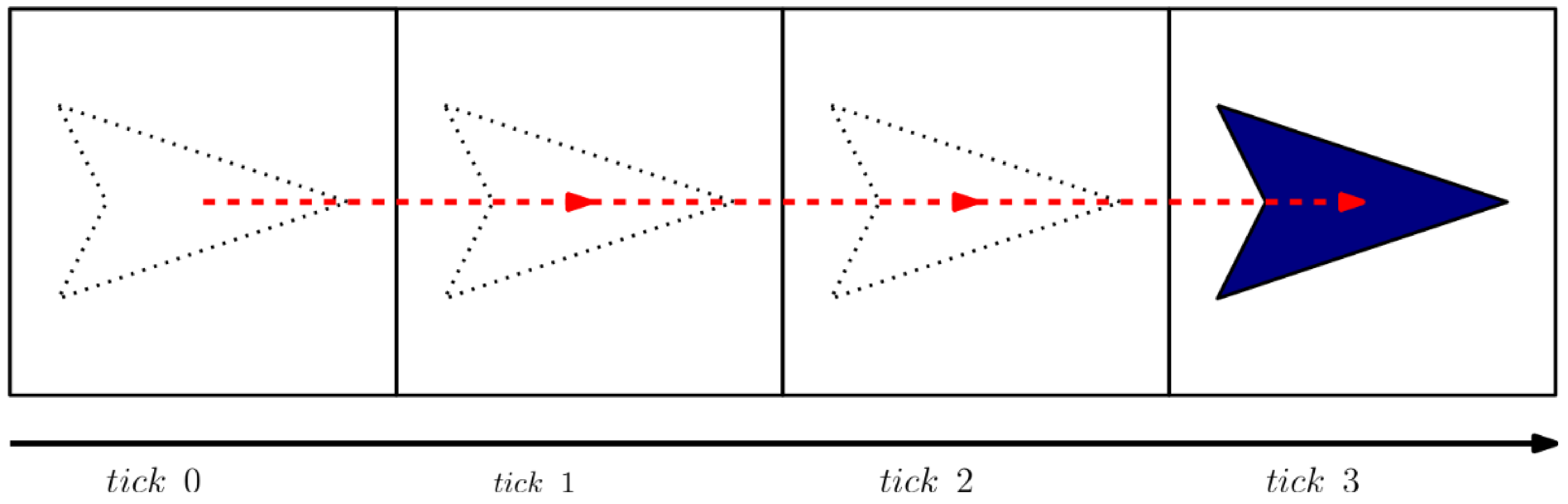
Each turtle has a **position**, coordinates, and a **heading**, expressed in degrees. *0º* is north.

The agents



It is characterized by **size**, **color** and **shape**.

The agents



Similar to space, time is discrete too. Agent actions are performed every **tick**.

The agents

Each agent is equipped with a set of **properties:**

- who
- heading
- xcor and ycor
- shape, size, color
- hidden?

The observer

The **observer** modifies the environment and the agents.

Through the use of **commands** turtles can be created, moved, modified and so on.

Some commands

To create a turtle the command create-turtles is used.

```
create-turtles <num>

create-turtles 1
```

Some commands

The command inspect turtle is used to inspect the properties of a turtle.

```
inspect turtle <whoID>

inspect turtle 0
```

Some commands

The observer *asks* to the environment (or to the turtles) to change their properties.

The instructions are either sent to a specific turtle (or patch) or to the entire set of turtles (or patches).

Some commands

	Turtles	Patches

One	`ask turtle <whoID> [ ... ]`
	`ask turtle 1 [ set color red ]`
		`ask patch <x> <y> [ ... ]`
		`ask patch 2 3 [ set pcolor red ]`

All	`ask turtles [ ... ]`
	`ask turtles [ set color red ]`
		`ask patches`
		`ask patches [ set pcolor red ]`

Some commands

see https://ccl.northwestern.edu/netlogo/docs/dictionary.html
for more commands.

Programming in NetLogo

Instructions tell agents what to do.

Instructions to agents can be classified according to three criteria:

- whether they are built into NetLogo (**primitive**) or user-implemented (**procedure**)
- whether the instruction produces an output (**report**) or not (**command**)
- whether an instruction takes inputs or not

Commands vs reporters

Commands are procedures that don't have any output, but only side effects on the environment.

```
to <command name>
   [...]
end
```

```
to go
   clear-all
   create-turtles 10
   ask turtles [ forward 1 ]
end
```

Commands vs reporters

Reporters are procedures that compute a value and report it.

```
to-report <reporter name>
  [...]
end
```

```
to-report double [ num ]
    report 2 * num
end
```

Input parameters

```
to <command name> [ parameters ]
   [...]
end
```

```
to createNTurtles [ num ]
    create-turtles num
end
```

On styling

There isn't an official NetLogo style guide.
Nonetheless the official documentation is fairly consistent and follows some good habits:

- use camel case beginning with a lower-case letter for procedure (e.g. myProcedure);
- do not use underscores in names;
- name command procedure with nouns and reporters with verbs.

Variables

Variables in NetLogo can be divided into three main groups:

Local variables, defined as part of a procedure

```
let <name> <value>
```

Agent variables, defined as part of each agent

```
<agent*>-own [ <name(s)> ]

ask <agent*> <id> [
    set <name> <value>
]
```

Global variables, accessible by every agent and procedure

```
globals [ <name(s)> ]

set <name> <value>
```

*Agents can be **turtles**, **patches**, **links**

Agentsets

When *asking* to update an agent variables a subset of all the agents, called **agentset**, can be used.
An agentset contains one or more agents, all of the same type, and it's always randomly ordered.

```
ask one-of turtles [ <command> ] ; randomly choose among the whole set

let some-patches patches with [ pxcor < 3 ] ; take patches with X < 3
ask some-patches [ set pcolor red ] ; change the color of the subset
```

Variable types

NetLogo variables are dynamically typed.

Primitive types are **numbers**, **booleans**, **lists**, **strings**, along with the usual operations (+, -, *, /, ^, >, >=, =, !=, <, <=, and, or, not, xor).

All numbers are floating points, be aware of approximations. When performing arithmetic operations be aware of spaces: the lack of parenthesis might bring ambiguity in parsing the operation and result in something different.

Conditionals

```
if (<condition>) [ <command(s)> ]
```

```
ifelse (<condition>)
      [ <command(s) if true ]
      [ <command(s) if false ]
```

```
ifelse-value (<condition>)
     [ <reporter(s) if true ]
     [ <reporter(s) if false ]
```

```
if (random-float 1 < 0.5)
      [ show "heads" ]
```

```
ifelse (random-float 1 < 0.5)
      [ show "head" ]
      [ show "tails" ]
```

```
ask turtles [
      set color ifelse-value (energy < 0)
      [ red ]
      [ green ]
]
```

Loops

```
loop [ <command(s)> ]
```

```
loop [ ifelse (counter > 100 )
    [ stop ]
    [ set counter counter + 1 ]
]
```

```
repeat <num> [ <command(s)> ]
```

```
repeat 5 [
   ask one-of turtles [ set color red ]
]
```

Lists

In NetLogo lists are **immutable**, **ordered** and potentially **heterogeneous**.

```
( list <element(s)> )        ( list 1 "two" true )

    [ <element(s)> ]         [ 1 "two" true ]
```

Program structure

The flexibility of NetLogo and its agent-centered way of building models quickly escalates to complex models that are difficult to work with.

Try to keep your structure as close as possible to:

1. **global variable** declaration;
2. **agent variable** declaration;
3. **setup** procedure, in which global variables are initialized, agents are created and the environment is initialized;
4. **go** procedure, which implements one cycle of the simulation.

Some useful features

Higher-order procedure

Even though NetLogo is not a higher-order language we can simulate this behaviour using **anonymous procedures/reporters**.

```
[ [ <var(s)> ] -> <body> ]          [ [ x y ] -> setxy y x ]
```

Higher-order procedure

When an anonymous procedure is assigned to a variable it is called a **task**. A task can be run using the primitive **run**.

```
globals [ stack push ]
to setup
    set stack []
    set push [                                  (run push 1)
        el -> set stack lput el stack
    ]
end
```

Higher-order reporter

Similarly to procedures, a task can be created from an anonymous reporter. It can then be evaluated using **runresult**.

```
let square [ a -> a * a ]
```

```
(runresult square 5)
```

Higher-order example

We want to define an update function that takes as input a payoff p and returns a new payoff p' s.t. p'(n) = p(n) + 1.

```
to-report update [ payoff ]
    report [ a -> ( runresult payoff a ) + 1 ]
end
                                                     apply (update p) n = (apply p n) + 1
to-report apply [ payoff argument ]
    report ( runresult payoff argument )
end
```

Map, filter and reduce

Map, **filter** and **reduce** are basic constructors that allows efficient and elegant operations on lists.

Map applies an anonymous-reporter to every element in a list.

```
map [ a -> a * a ] [1 2 3] ; results in [1 4 9]
```

Filter applied a predicate (in the form of anonymous-reporter) to a list and returns only those items that entails the predicate.

```
filter [ a -> a < 5 ] [1 9 2] ; results in [1 2]
```

Map, filter and reduce

Reduce applies an anonymous-reporter from left to right, resulting in a single value.

```
reduce [ [ a b ] ->  a + b ] [1 9 2] ; results in 12
```

Breeds

In NetLogo **breeds** are a way to *"subclass"* the turtle type.

```
breed [ <single name> <agentset name> ]
```

Graphing

Graphing can be confusing at times in NetLogo. Always think of it as a *special agent* that moves through the graph.

It is composed of two phases:

- **setup**, where you need to set the whole graph range (x-axis) along with each *special agent* pen properties, such as color and pen mode
- **update**, where you need to draw data. Update is called at every tick.