Computer Programming in the Physics Classroom

Lilian Clairmont, Ph.D.

Appomattox Regional Governor's School

lclairmont@args.us

Introduction

My intention today is to share with you my experience with two programming platforms that I believe are very useful in facilitating visualization in physics classrooms.

These platforms are:

I. **NetLogo:** a programming language and integrated development environment for agent-based modeling.

[**NetLogo**]

I. **Glowscript:** a programming environment that allows us to make web-based 3D models.

[**Glowscript**]

Physics demographics @ ARGS

AP students

- Started offering AP for the 2020-2021 year
- (20-21) Enrolment was 40% physics-60% non-physics students
- (21-22 and subsequent years) Enrolment 100% non-physics students

Non-AP students

- 2019-2020: strong class of students;
- 20-21: strong class of students
- 21-22 and subsequent years: less academic oriented students

Obstacles in Physics Visualization

- Difficult for students

  Obvious reason: other than position and changes in position, all other variables are invisible

- Solution: <u>Explore motion representations first</u>
  - – Recognizing the various types of motion by observing and cataloging position changes (motion maps, verbal descriptions, graphical, etc.)
  - – Classifying the various types of motion (@ rest, UM, UAM, UCM, etc.) based on patterns and concepts (inertia and equilibrium.)

Challenges with Kinematics Equations

AP students

- Have *some* difficulty with connecting equations to other representations;
- Familiarity improves after some "brute force" experiences (for example, where students realize that equations make it easier to complete certain tasks – PhET's "The Walking Man" lab)

Non-AP students

- Struggle relating algebraic and graphical representations of motion;
- Need more "hands-on" approaches.

→ storytelling: from 1D to graphs to (simple) constraint problems.

→ stop action movie

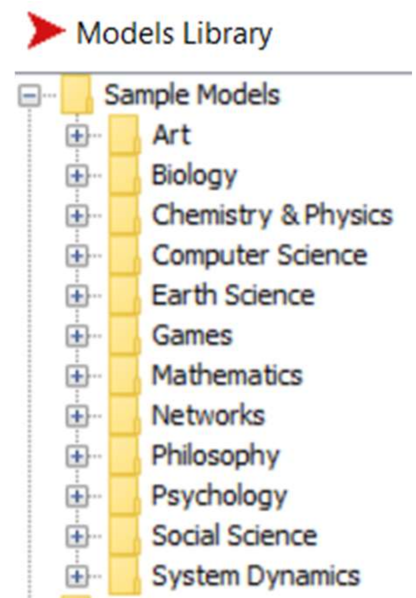
Both groups also struggle with constraint problems.

Visualization Tool

(I) Agent-Based Modeling with NetLogo

- An ***agent-based model (ABM)*** is a ***computational model*** for ***simulating*** the actions and interactions of ***autonomous agents (*)*** in order to ***understand a system's behavior*** and ***what governs its outcomes***.

NetLogo Models Library Categories



- **ABM** has applications in art, biology, chemistry, computer science, earth science, ecology, economics, games, mathematics, networks, philosophy, psychology, social science, system dynamics and more.

(*) Autonomous agents are individual or collective entities such as organizations or groups.

NetLogo: Programming Features

“Agents”

- Turtles (movable)
- Patches (stationary)
- Links (connectors between turtles)

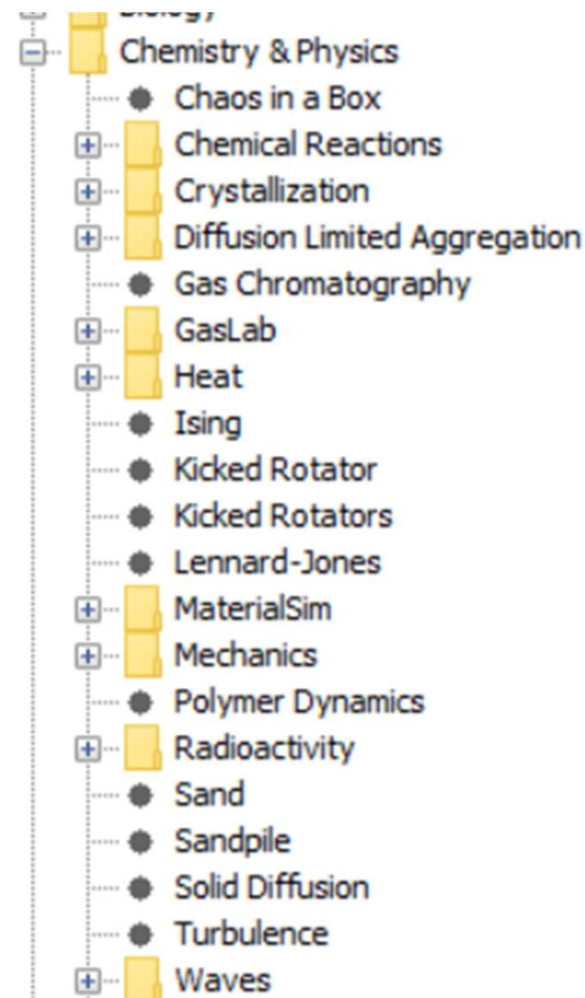
Interface

- Buttons, sliders, monitors, switches

Agents are…

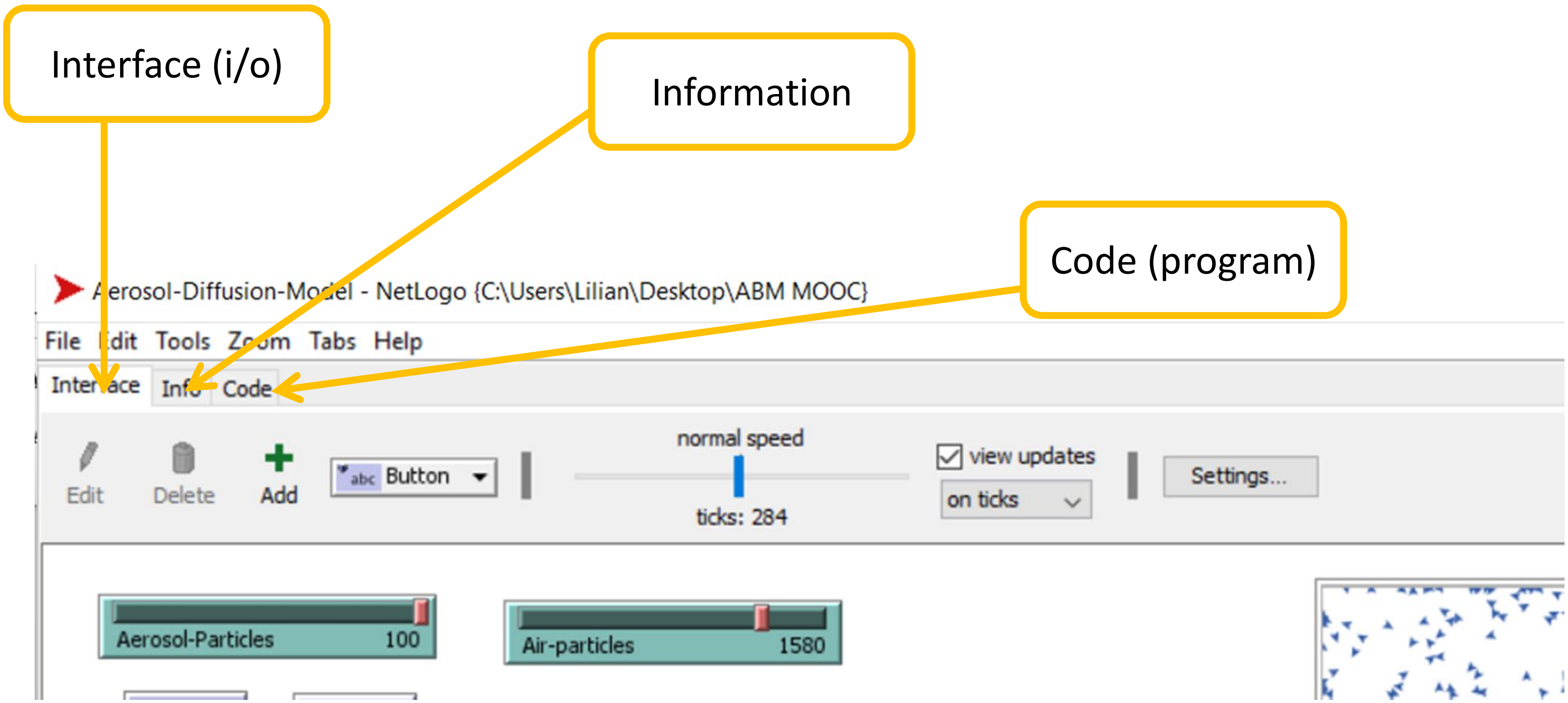
- Named
- Monitored (for programming purposes)

*** NetLogo: open source software ***

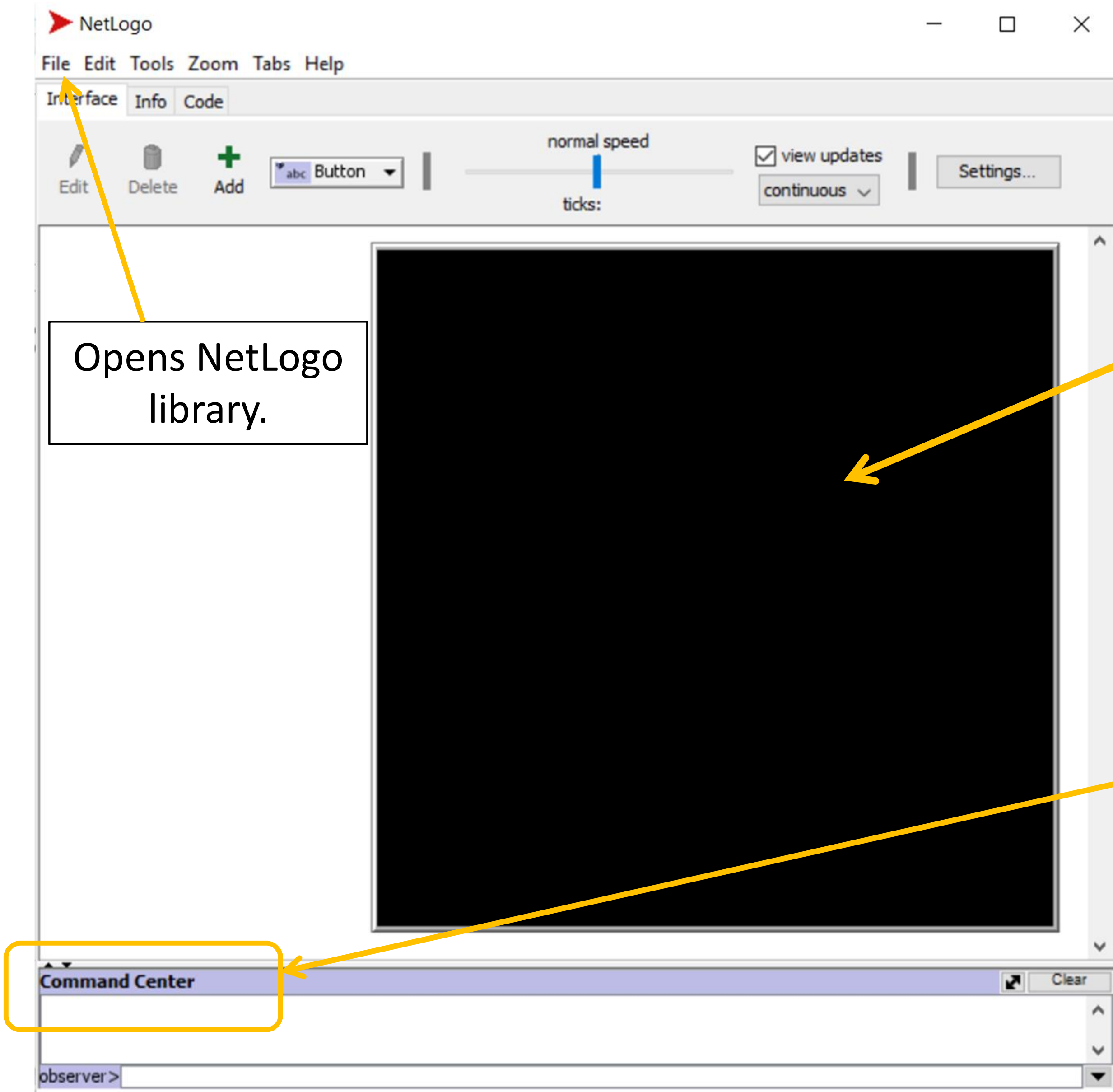


NetLogo: Platform Features

When you start NetLogo, you will see three tabs:



Interface Tab



Opens NetLogo library.

"World"

To interact directly with the agents

Info Tab

describes the program and how to run it

WHAT IS IT?

This code models the diffusion of aerosol particles in a room. I have calculated the dispersion time a long time ago, and found it to be 40 seconds for an average room, between opposite corners. This program is my simulation of the process. AEROSOL-PARTICLES are picked up by air particles (turtles) and carried away from the place where the aerosol was first sprayed (bottom left corner of world.)

The program is based on the “Follower” ABM. It is an oversimplification of the actual diffusion of aerosols, which is: (aerosol particles) collide with air particles, thus changing direction and eventually, spreading throughout a room.

HOW IT WORKS

explains what the agents do and/or how they act

The model’s environment is an average, typical square room, colored white. Most interactions are between turtles, except for the fact that turtles “know” where the walls are (wrapping around is disabled), and air particles check patches within FAR-RADIUS for aerosol ones.

Observer interactions with the world are through SETUP and GO buttons; interactions with turtles are through WAVER, AEROSOL-PARTICLES and AIR-PARTICLES sliders.

There are two agents: aerosol and air particles, represented by turtles of different colors. AEROSOL-PARTICLES are picked up by air particles if they are within a certain range from it (input = radius of the range for the air particle to look for an aerosol one.)

Air: blue, randomly placed (initially), size 0.5; can attach to aerosol turtles (and disperse them by carrying them along)

AEROSOL-PARTICLES: green, initially located at the bottom-left corner of world, size 0.8; can be picked up by air turtles, and will follow them (if picked up)

Code Tab

Find... Check | Procedures ▾ | ☑ Indent automatically

```
;; Beginning declarations

turtles-own [
  leader    ;; the turtle this turtle is following,
            ;; or nobody if not following
  follower  ;; the turtle that is following this turtle,
            ;; or nobody if not being followed
]

;; Set up and initialization

to setup

  clear-all

  ask patches [ set pcolor white ] ;; white backgrounc

  ;; creates air particles based on slider and random]

  create-turtles Air-Particles [
    setxy random-xcor random-ycor
    set color blue
    set size 0.5
    set leader nobody
    set follower nobody
  ]

  ;; creates aerosol particles based on slider at the

  crt Aerosol-Particles [
    setxy -16 -16
    set color green
    set size 0.8
    set leader nobody
    set follower nobody
  ]
```

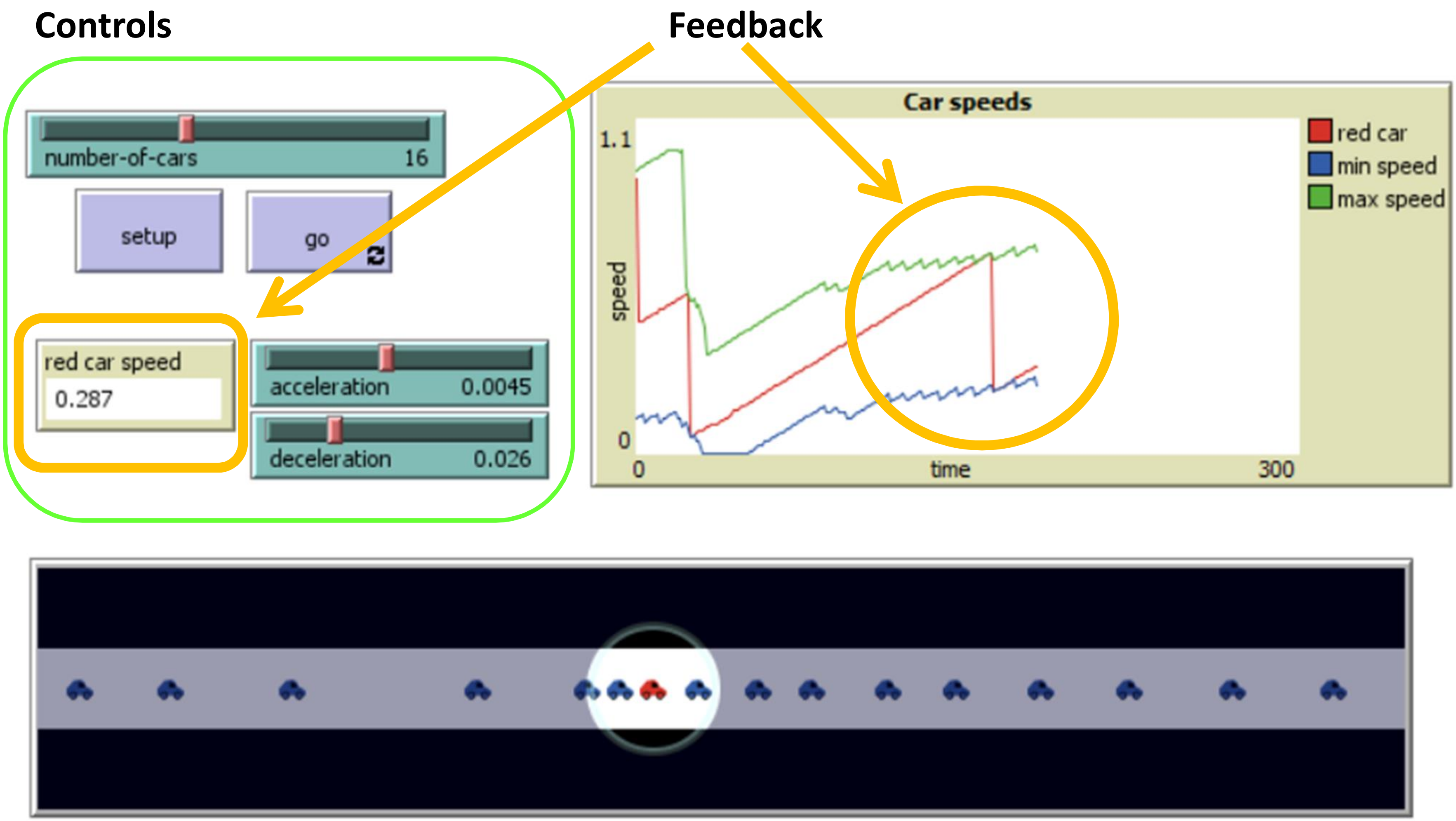
```
;; procedure to determine whether or not to attach to the aerosol particle

to attach  ;; air turtle procedure
  ;; find a random patch to test for aerosol around the air turtles
  let xd random (far-radius)
  let yd random (far-radius)
  if random 2 = 0 [ set xd (- xd) ]
  if random 2 = 0 [ set yd (- yd) ]
  ;; check for aerosol turtles on that patch
  let candidate one-of (turtles-at xd yd) with [color = green]
  ;; if we didn't find a suitable turtle, stop
  if candidate = nobody [ stop ]
  ;; we're all set, so latch on!
  ask candidate [ set follower myself ]
  set leader candidate
  ;; change our color
  ifelse follower = nobody
  [ set color orange ]
  [ set color blue ]
  ;; change our leader's color
  ask candidate
  [ ifelse leader = nobody
    [ set color orange ]
    [ set color green ] ]

end
```

NetLogo: Traffic Jam

- Mimics the movement of cars in a highway
- Agents follow two rules:

1. **Deceleration:** slows down if it sees a car close ahead;
2. **Acceleration:** speeds up if it doesn’t see a car ahead

NetLogo: Traffic Jam



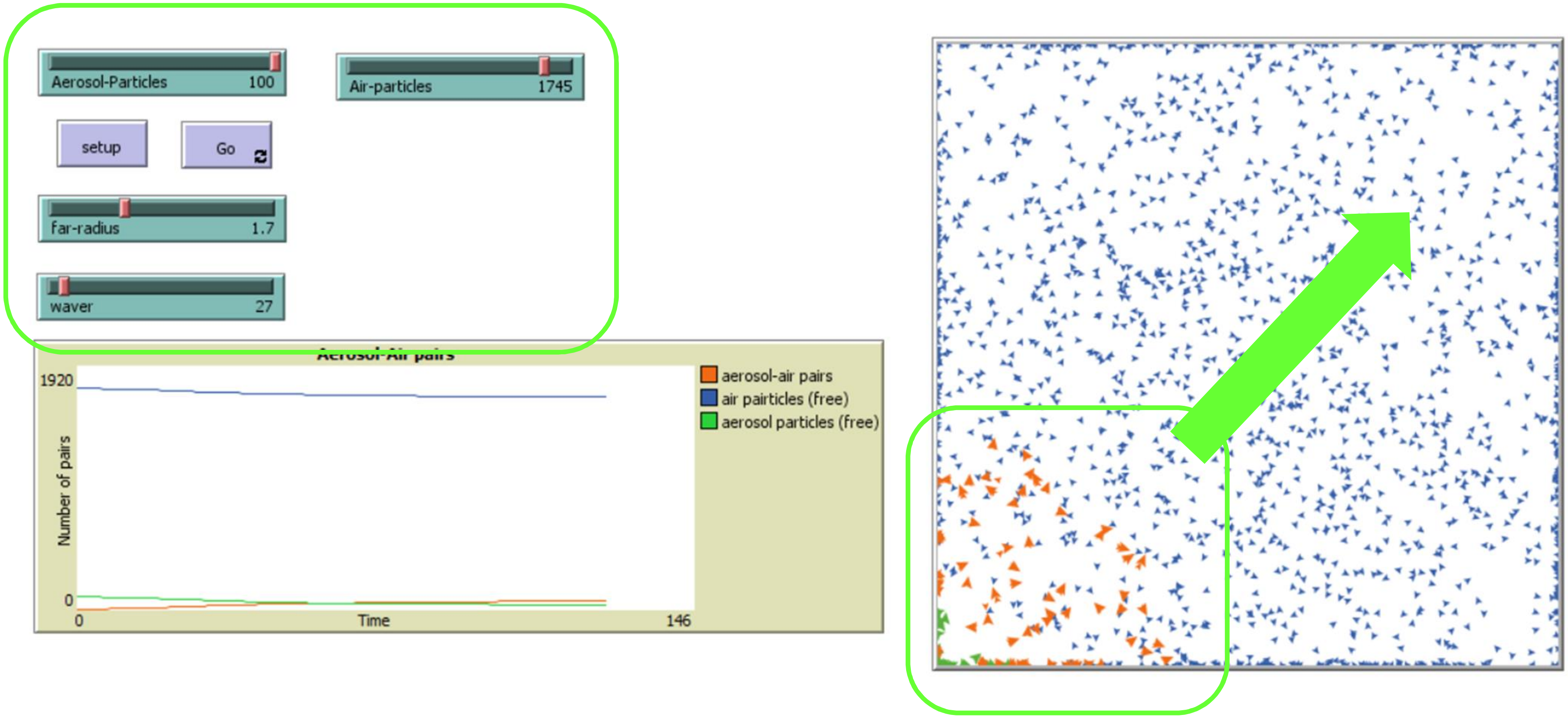
NetLogo: Aerosol Dispersion

- Simulates the spread of aerosol particles in a closed, average sized room
- Agents follow one rule:

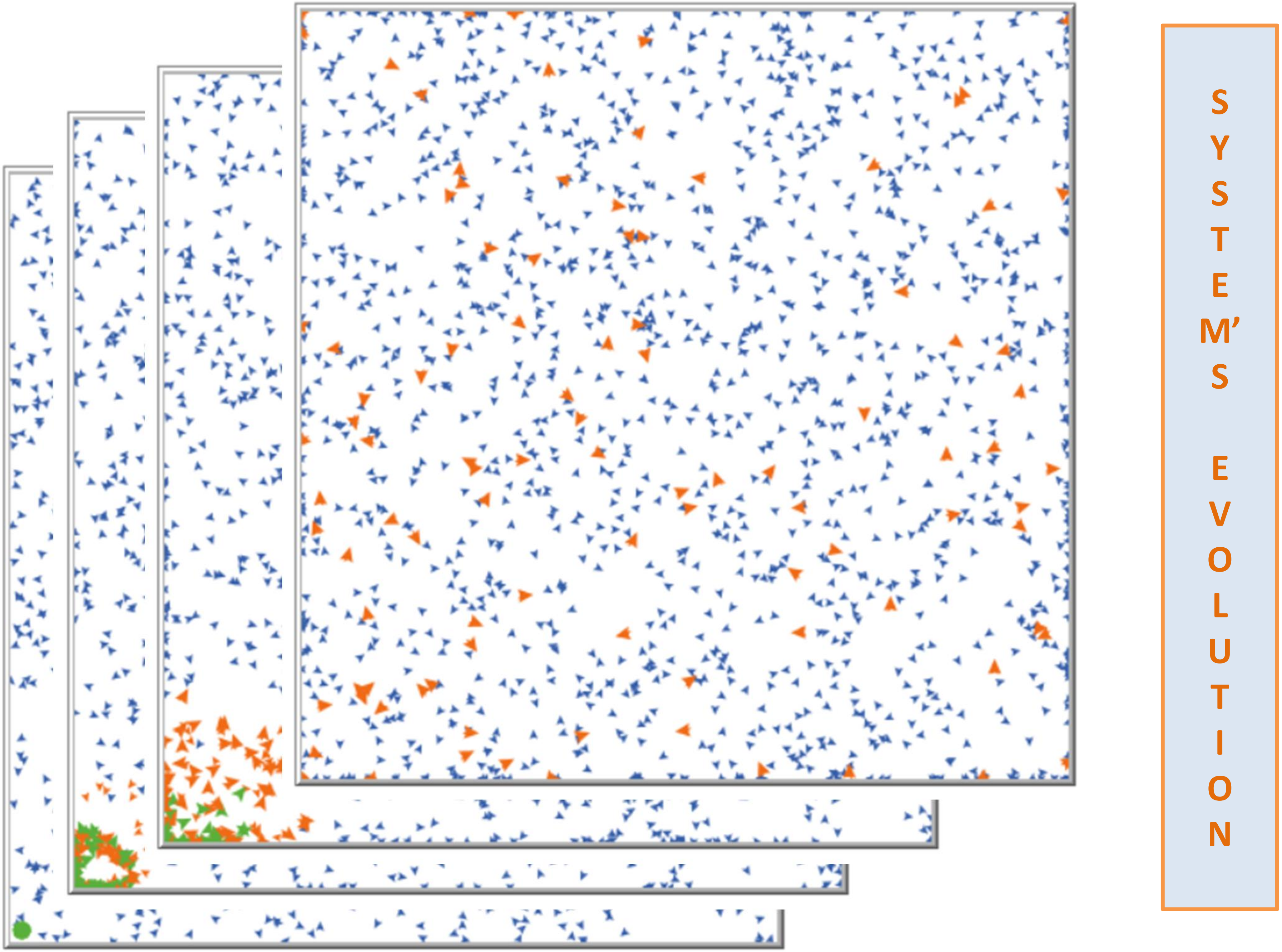
Neighboring

air particles pick up aerosol particles if they are in a pre-determined neighboring area.

NetLogo: Aerosol Dispersion



Aerosol particles change color as they get picked up by air particles.

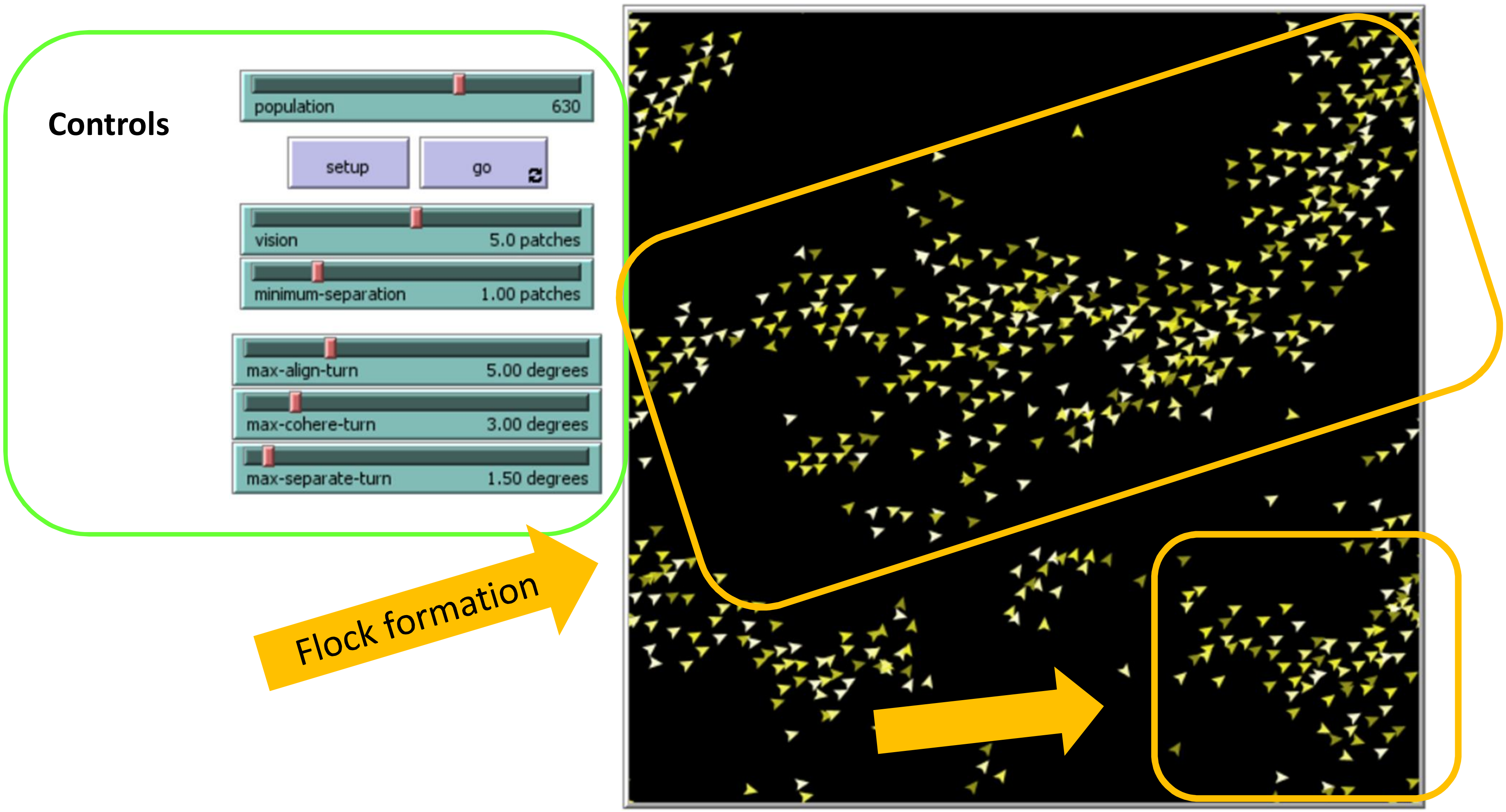

SYSTEMS' EVOLUTION

NetLogo: Flocking

- Attempts to mimic the flocking of birds
- Agents follow three rules:

1. **Separation:** Avoid birds that are too close;
2. **Cohesion:** Move towards nearby birds (rule #1 overrules #2 if they are too close);
3. **Alignment:** turns so that it is moving in the same direction as nearby ones are (moving.)

NetLogo: Flocking



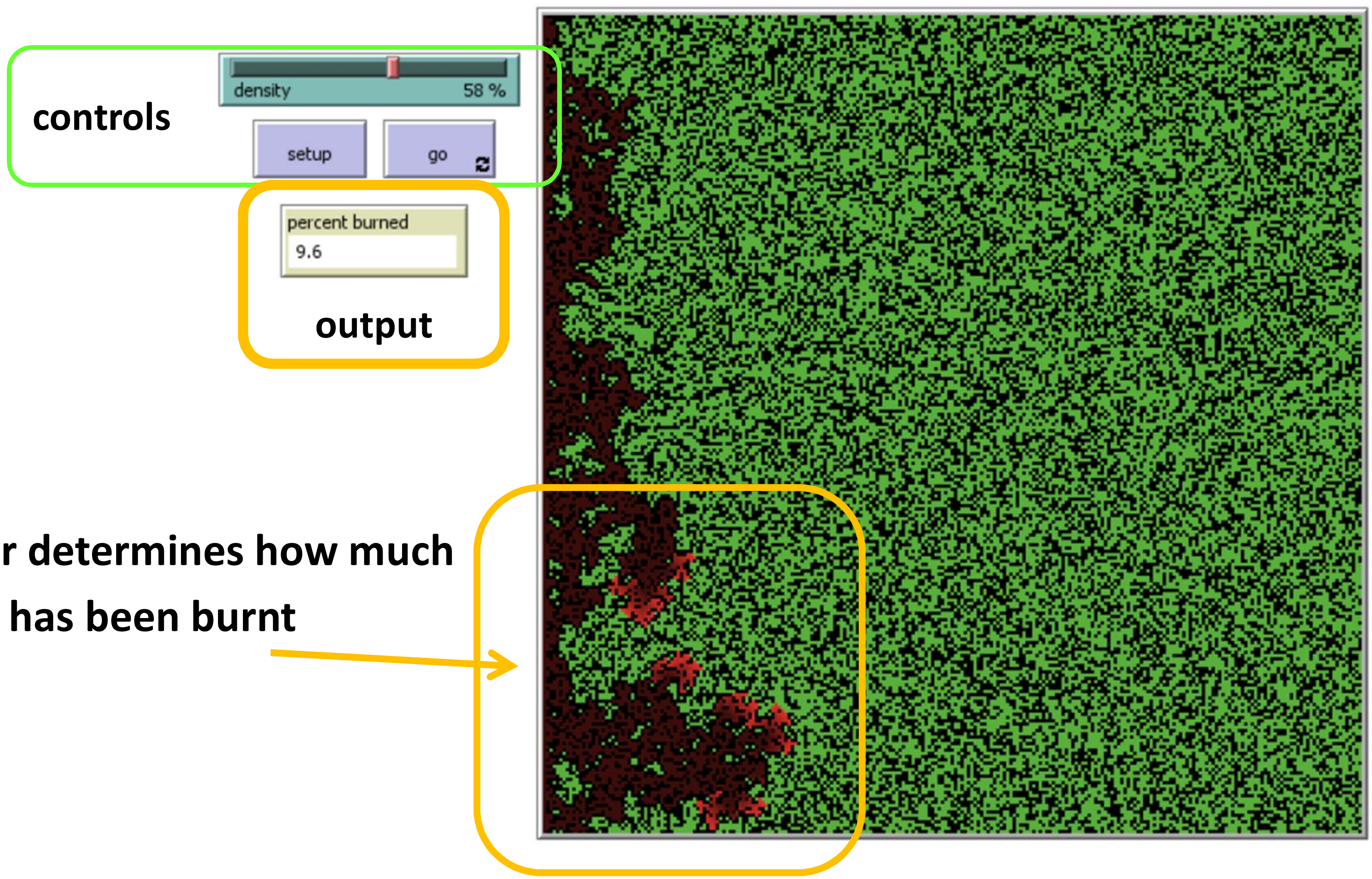
NetLogo: Fire

- Simulates the spread of a fire through a forest
- No wind
- Agents follow one rule:

Neighboring

Must have a tree in the neighboring patch for the fire to burn

NetLogo: Fire

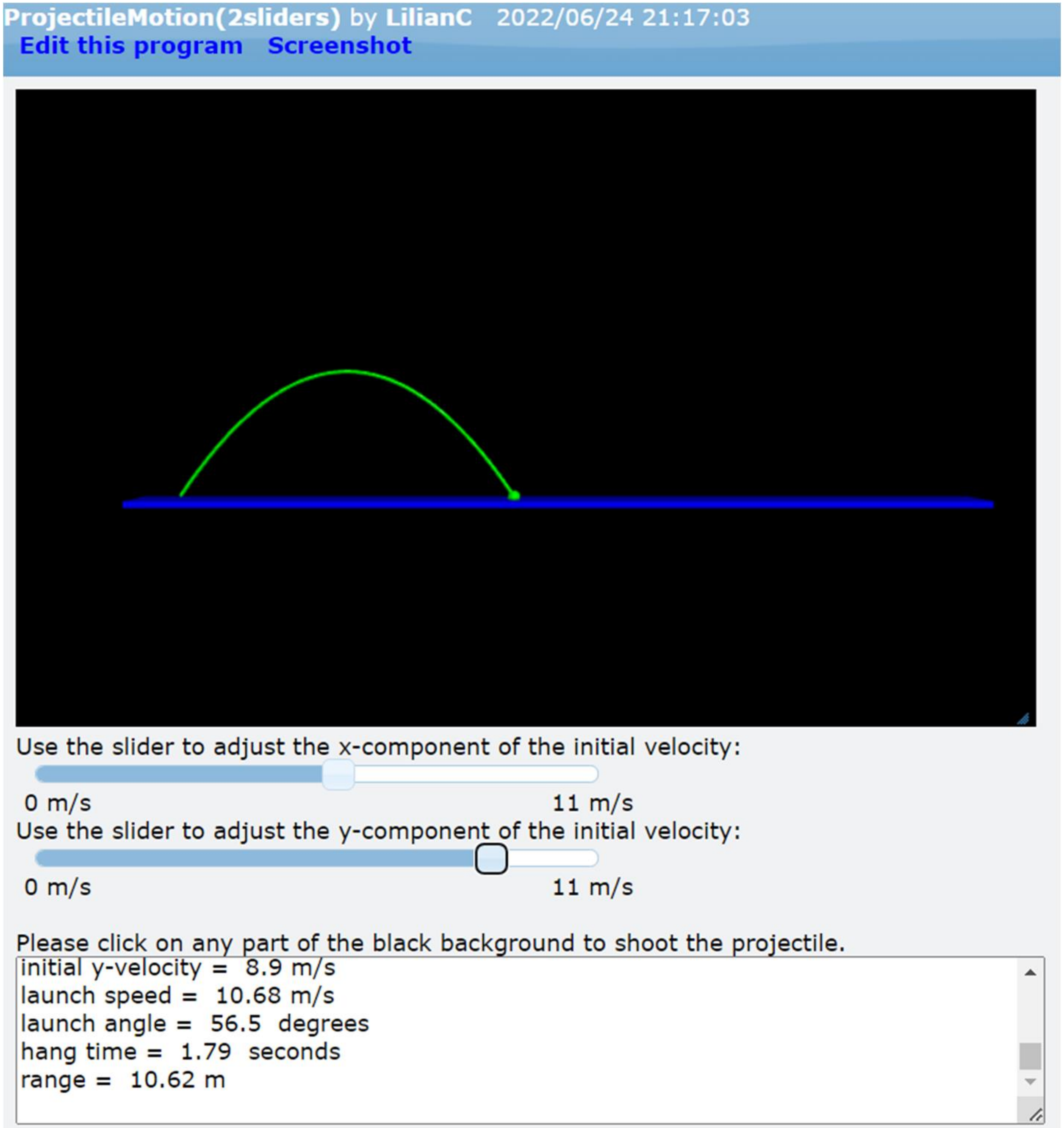


Color determines how much has been burnt

More Visualization Tools

(II) Computer programming with **Glow Script**

- Computer Animation of motion problems helps students understand motion and correlate what they see to the algebra equations representing them;
- **Language: Web VPython**
- Easy to run; easy to teach; easy to program
- Taps on students' creativity (always a ***bonus***)



Projectile Motion

* added sliders to enable choices for v1x and v1y

Code for Projectile Motion

creates the landscape

Must create gravity!

interactive

Calculates muzzle speed based on inputs

```
Web VPython 3.2
#Projectile Motion with x- and y- sliders
#programmer: Lilian Clairmont

#PM - Object definitions
PhysicsBlob=sphere(pos=vec(-11,-2.8,0), radius=0.2, vel=vec(0,0,0),color=color.green, make_trail=True)
ground=box(pos=vec(1,-3,0), length = 27, height = .2, width = 1, color=color.blue)

#x slider
scene.append_to_caption("Use the slider to adjust the x-component of the initial velocity:\n")  #  text above the slider bar
def ProjMotx(sx):
    print(sx.value)
    PhysicsBlob.vel.x = sx.value
slider(bind=ProjMotx, min = 0, max = 11, step = 0.1, value = 0)
scene.append_to_caption("\n 0 m/s                                                  11 m/s\n")   # define range of slider bar

#y slider
scene.append_to_caption("Use the slider to adjust the y-component                  text above the slider bar
def ProjMoty(sy):
    print(sy.value)
    PhysicsBlob.vel.y = sy.value
slider(bind=ProjMoty, min = 0, max = 11, step = 0.1, value = 0)
scene.append_to_caption("\n 0 m/s                                                  11 m/s\n")   # define range of slider bar

scene.append_to_caption("\n")   # blank line
scene.append_to_caption("Please click on any part of the black background to shoot the projectile.\n")  #  onscreen instructions to run the program
scene.waitfor("click")   #Click on the scene withthe projectile

print('initial x-velocity = ',PhysicsBlob.vel.x,'m/s')
print('initial y-velocity = ',PhysicsBlob.vel.y,'m/s')

#Calculating launch speed and angle
launchspeed=sqrt(PhysicsBlob.vel.x*PhysicsBlob.vel.x + PhysicsBlob.vel.y*PhysicsBlob.vel.y)

launchangle=atan(PhysicsBlob.vel.y/PhysicsBlob.vel.x)

#conversion from radians to degrees
launchangle=launchangle*180/pi
```

Code for Projectile Motion (cont'd)

```
#Rounding to one decimal place
launchspeed = launchspeed*100
launchspeed = round(launchspeed)
launchspeed = launchspeed/100
launchangle = launchangle*10
launchangle = round(launchangle)
launchangle = launchangle/10

print('launch speed = ',launchspeed,'m/s')
print('launch angle = ', launchangle,' degrees')

#Calculations
PhysicsBlob.acc=vec(0,-9.8,0)
t=-0.01
dt=0.01
while PhysicsBlob.pos.y>=2.8:
    t = t + dt
    rate(100)
    PhysicsBlob.vel = PhysicsBlob.vel + PhysicsBlob.acc * dt
    PhysicsBlob.pos = PhysicsBlob.pos + PhysicsBlob.vel * dt + .5*PhysicsBlob.acc*dt*dt

print('hang time = ',t,' seconds')
PhysicsBlobrange = PhysicsBlob.pos.x + 11
print('range = ',PhysicsBlobrange, 'm')
```

$$(blob)_{vel} = (blob)_{vel} + (blob)_{acc} \cdot dt$$

$$(blob)_{pos} = (blob)_{pos} + (blob)_{vel} \cdot dt + (blob)_{acc} \cdot dt \cdot dt$$

- Code equations are written as they would be in Physics;
- Easier for students to connect the motion to the algebra behind it.

Thank you

Lilian Clairmont, Ph.D.

www.linkedin.com/in/lilianfclairmont

lclairmont@args.us