SINDH MADRESSATUL ISLAM UNIVERSITY KARACHI
Chartered by Government of Sindh, Recognized by HEC

LAB MANUAL OF

AGENTBASE MODELING

Name: Muhammad Yahya
Father Name: Muhammad Hussain
ID: (BAI-23F-030)
SECTION: 5B

DEPARTMENT: BS ARTIFICIAL INTELLIGENCE
LAB INSTRUCTOR: FURQAN AHMED

FALL-2025

CERTIFICATE

DEPARTMENT OF ARTIFICIAL INTELLIGENCE

AGENT BASE MODELING

This is certified that Mr/Ms.**Muhammad Yahya** having Roll No**.BAI-23F-030** has successfully completed Laboratory work during FALL Semester 2025.

LAB INSTRUCTOR:
FURQAN AHMED

SIGNATURE

TABLE OF CONTENTS

LAB 1: INTRODUCTION TO NETLOGO AND INSTALLATION

OBJECTIVE

- Understand whatNetLogois.
- Learn how to install NetLogo on Windows.
- Explore the NetLogo interface and run a built-in simulation.

LANGUAGE/TOOL

- **Tool:** NetLogo 6.4.0 (or latest version)
- **Platform:** Windows (64-bit recommended)

OFFICIAL SITE: https://ccl.northwestern.edu/netlogo/

THEORY

NetLogo is a multi-agent modeling environment used to simulate how simple rules at the individual level can lead to complex patterns at the system level.

Agents in NetLogo

1. Turtles – moving agents (cars, animals, people).
2. Patches – the background grid (environment).
3. Links – connections between turtles (roads, social ties).
4. Observer – the global controller.

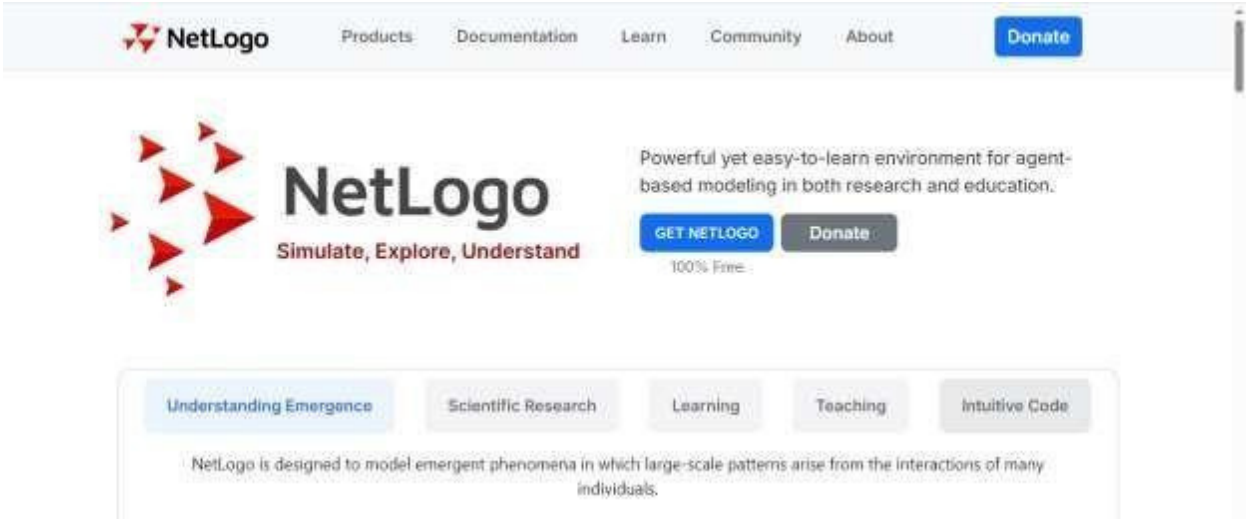
WHY USE NETLOGO?

- Beginner-friendly, yet powerful.
- Large Models Library (ready-to-use simulations).
- Widely used in education, AI, biology, and social sciences.

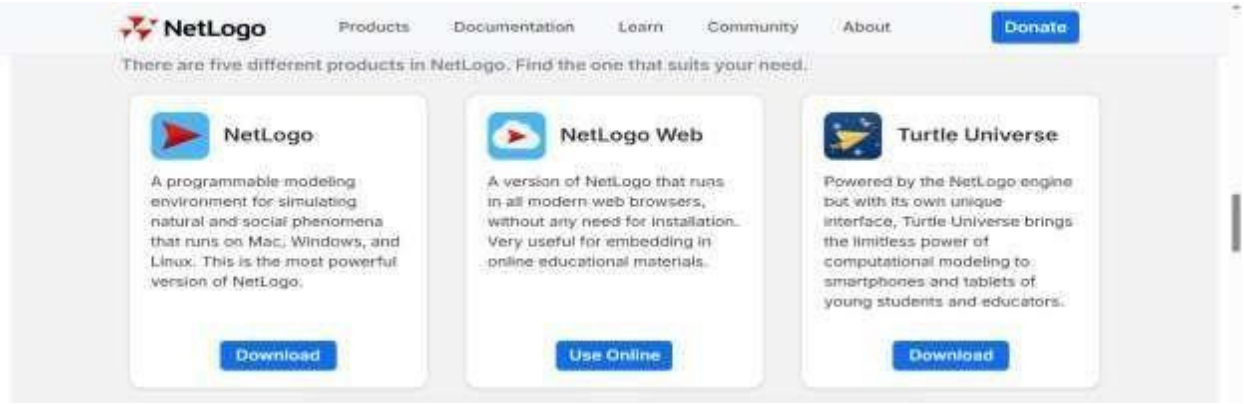
INSTALLATION ON WINDOWS

Step 1: Download

1. Open a browser and go to: https://ccl.northwestern.edu/netlogo/.


NetLogo
Products
Documentation
Learn
Community
About
Donate
NetLogo
Simulate, Explore, Understand
Powerful yet easy-to-learn environment for agent-based modeling in both research and education.
GET NETLOGO
Donate
100% Free
Understanding Emergence
Scientific Research
Learning
Teaching
Intuitive Code
NetLogo is designed to model emergent phenomena in which large-scale patterns arise from the interactions of many individuals.

2. Click Download NetLogo.



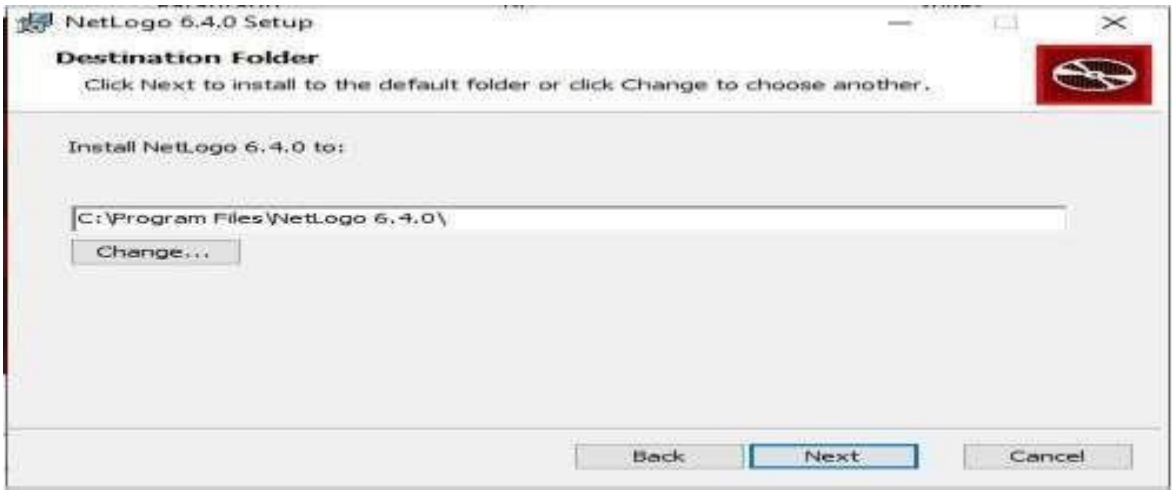
3. Select **Windows Installer (.exe)**.

Step 2: Install

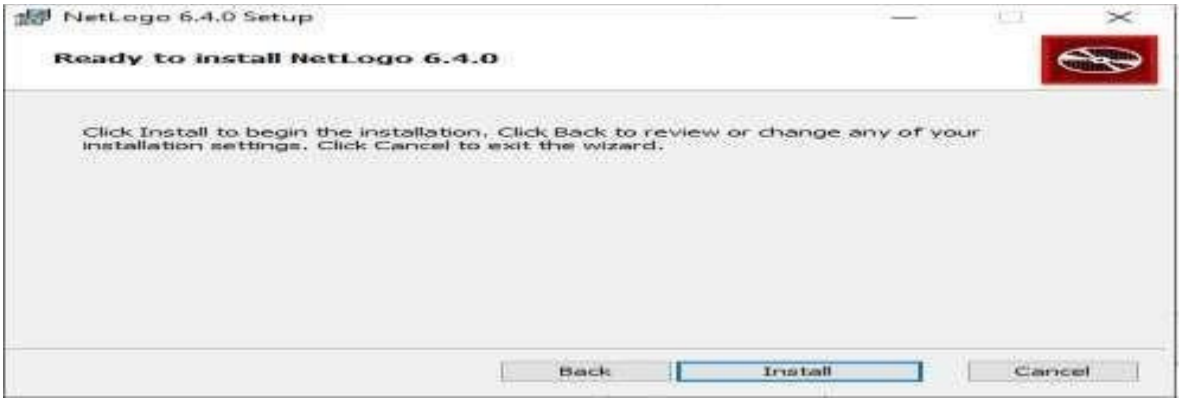
1. Locate the downloaded .exe file (in *Downloads*).
2. Double-click the file → installation wizard opens.

Follow the steps:

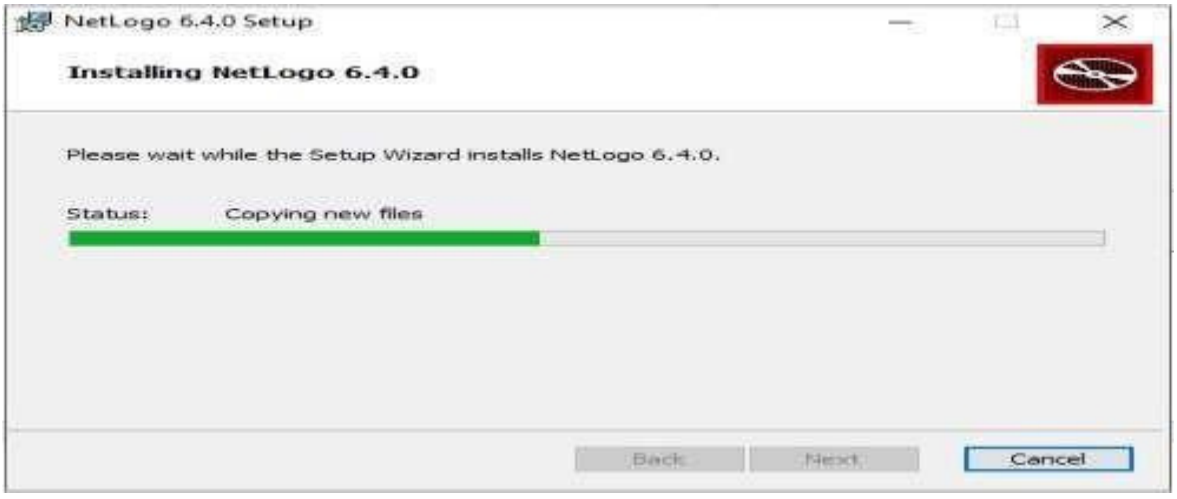
Click Next.



- Choose install location (default is fine).



- Click Install.



- Wait for installation to complete.

-

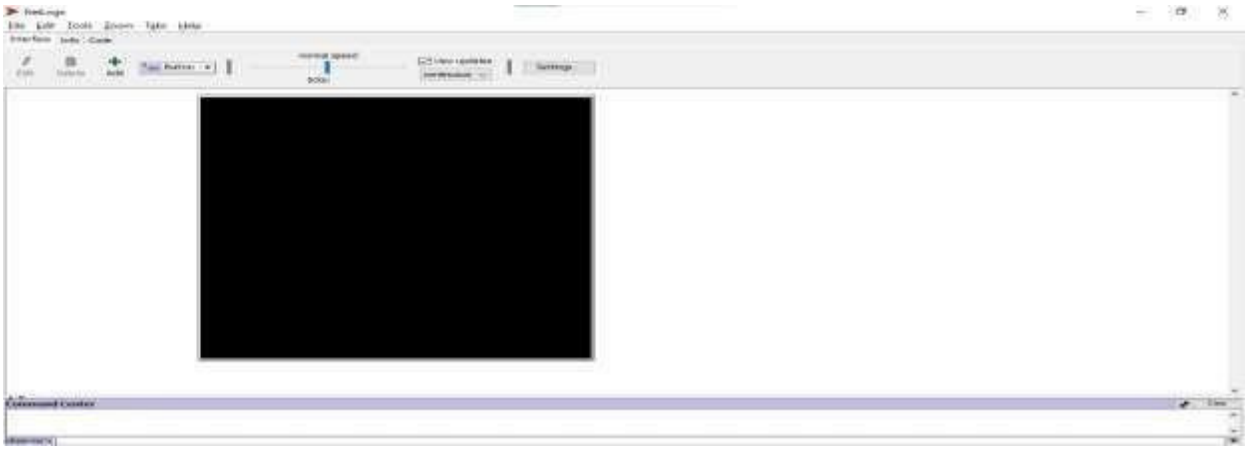
- Click Finish.

Step 3: Verify Installation

1. Go to Start Menu and search NetLogo 6.4.0.
2. Open it.
3. You should see the NetLogo main window with:

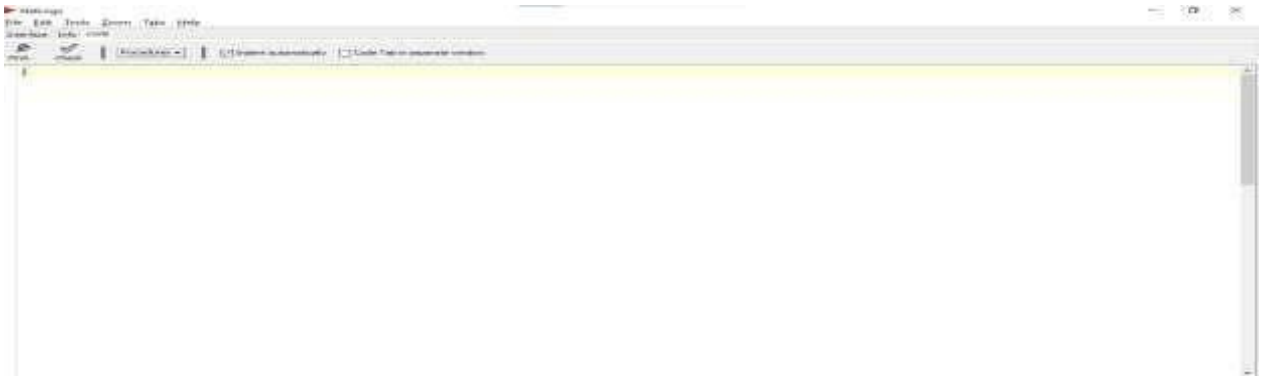
• Menu bar (File, Tools, etc.)

- Tabs: *Interface*, *Info*, *Code*
- Command Center at the bottom



- **Interface Tab:** Run models, adjust sliders, press buttons, see plots.

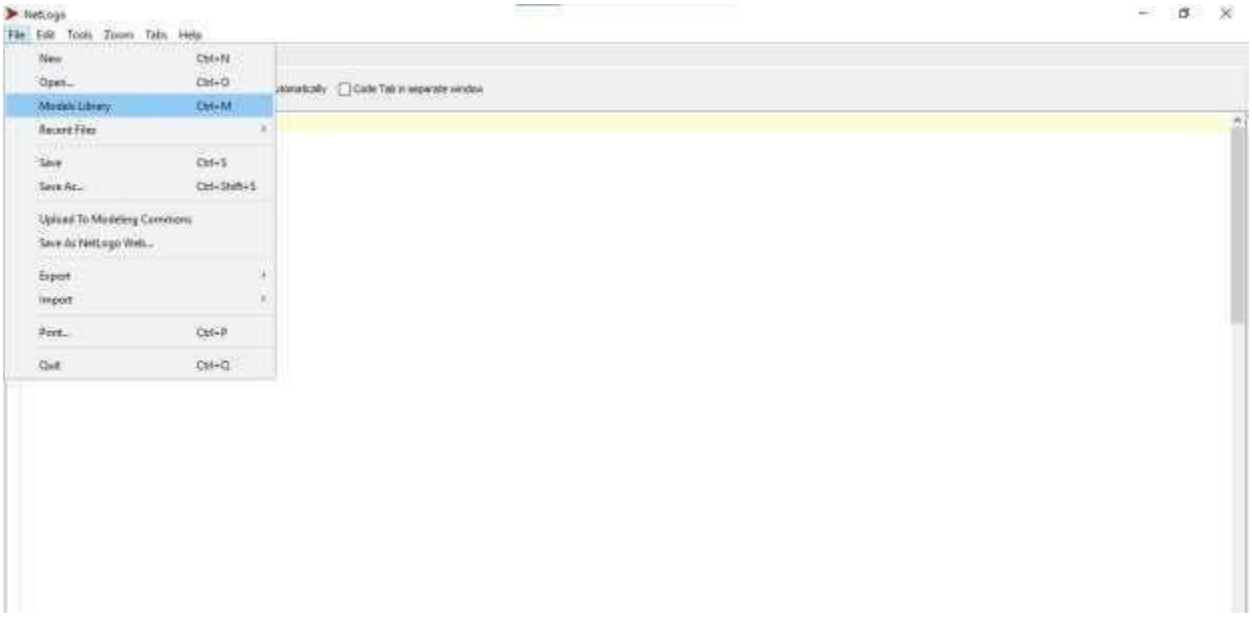
Info Tab: Description of the model and instructions.



- **Code Tab:** Whereyouwriteyourown NetLogo programs.
- **Command Center:** Forquickcommands.

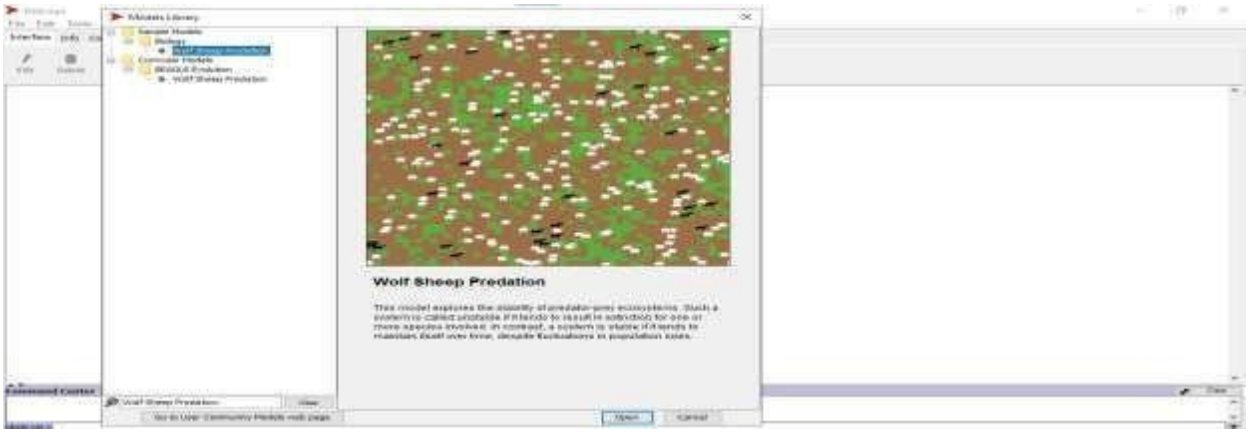
PRACTICAL EXERCISE (FIRST RUN)

1. OpenNetLogo.



-

2. Go to File Models Library.



3. Select Biology Wolf Sheep Predation.

4. Click Setup, then go.

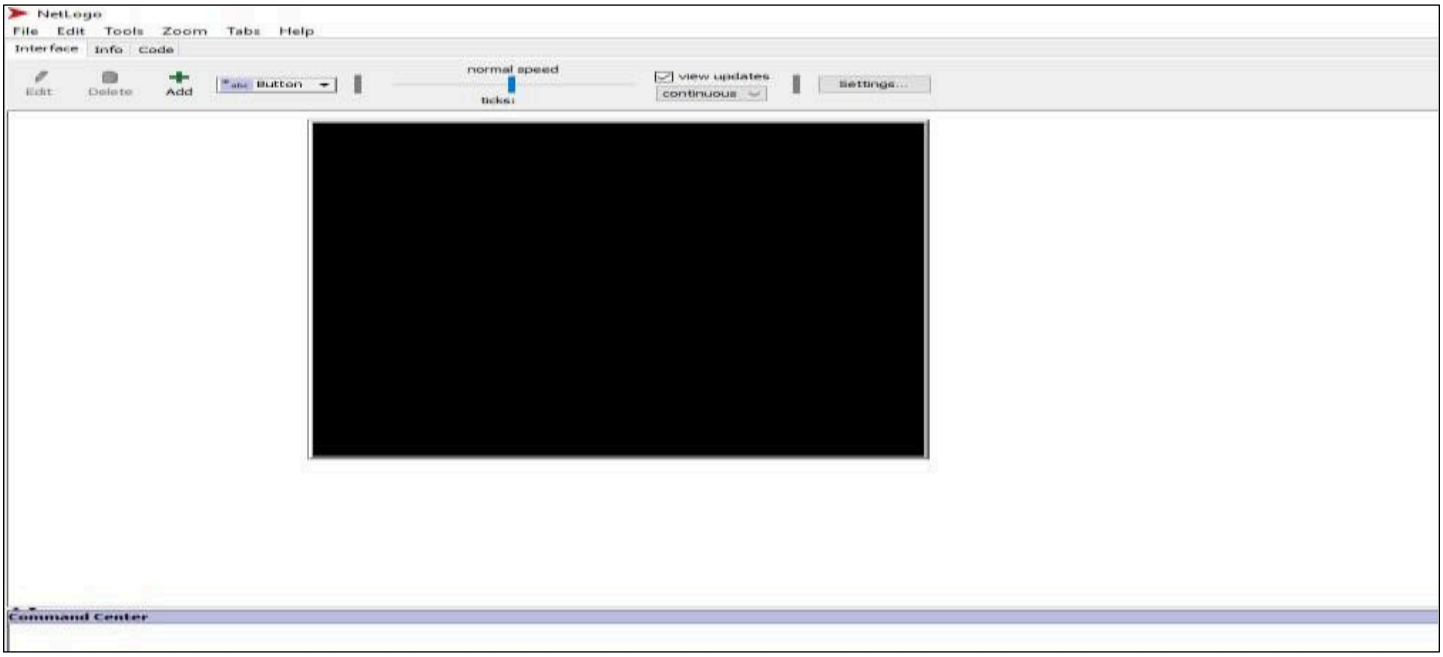
Observe wolves chasing sheep while sheep eat grass.

- Use the sliders to change parameters like "sheep-reproduce" or "wolf-gain-from-food" and see what happens.

STUDENT TASK

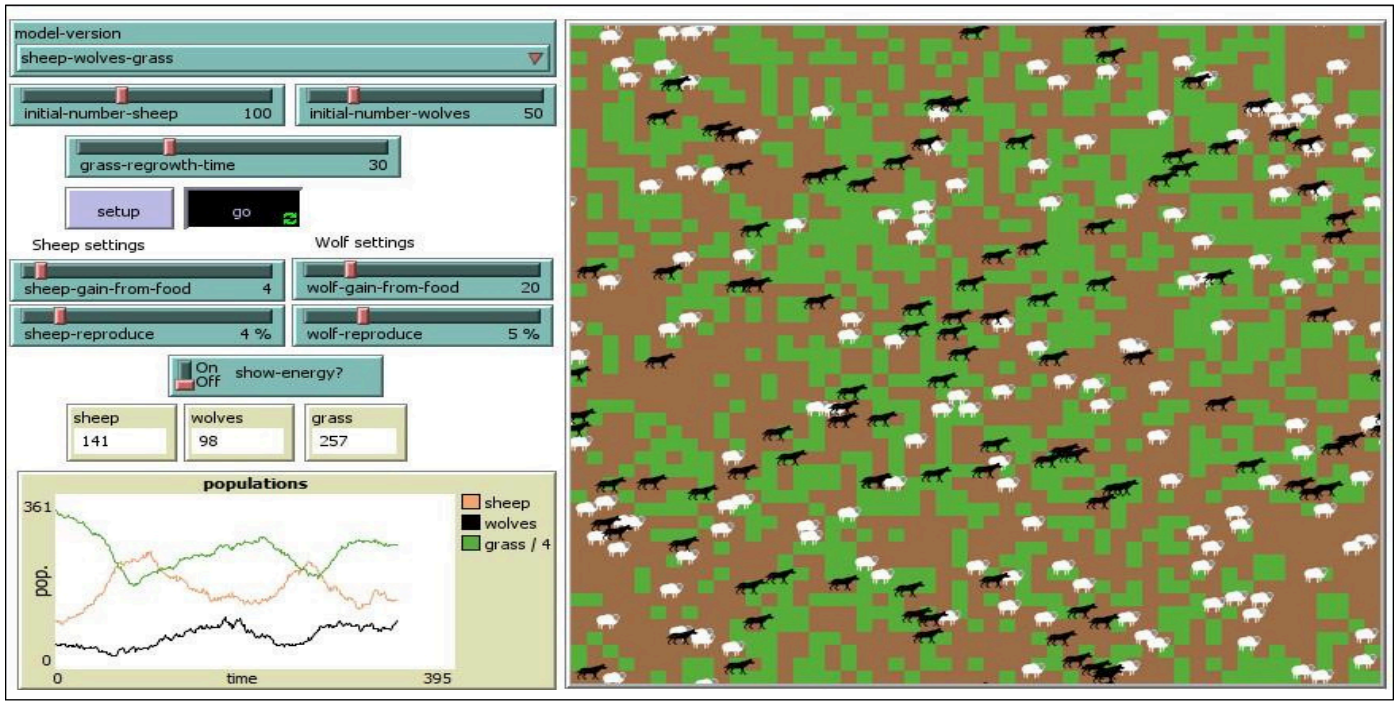
1. InstallNetLogo on Windows and take a screenshot of the main interface.

-

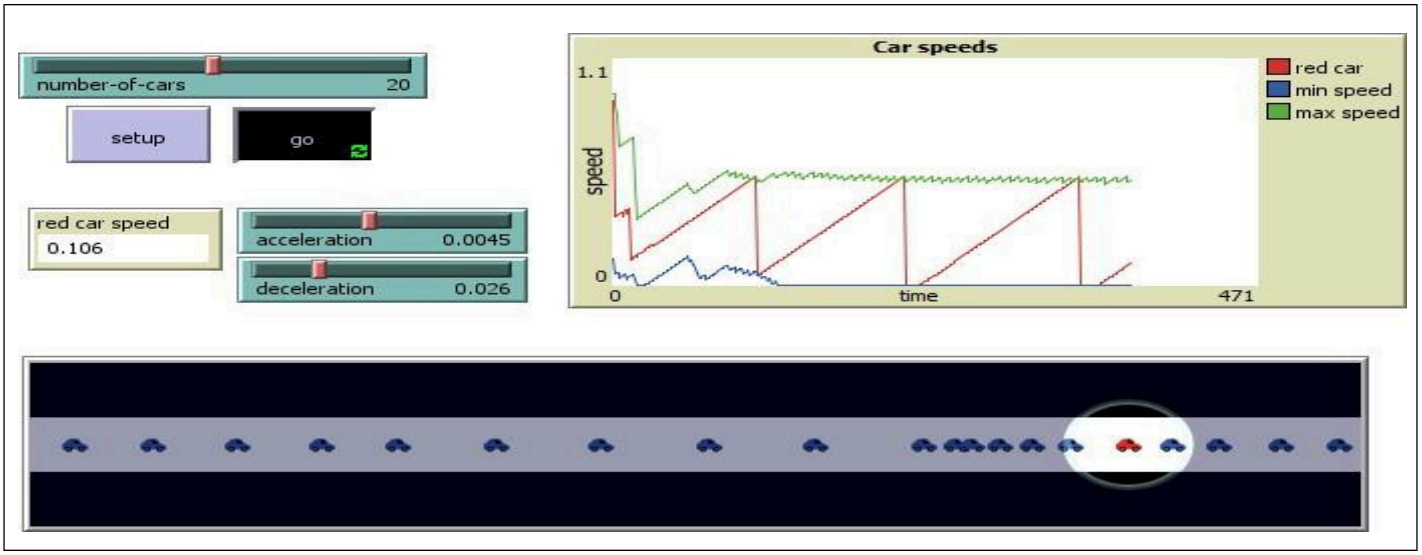

NetLogo
File Edit Tools Zoom Tabs Help
Interface Info Code
Edit
Delete
Add
Button
normal speed
ticks:
view updates
continuous
Settings...
Command Center

2. Openandruntwodifferentmodels from the library (e.g., Wolf Sheep Predation and Traffic Basic).

WOLF S HEEP P REDATION MODEL



TRAFIC B ASIC M ODEL



3. Identify which agents are turtles, patches, links, and observer in those models.

• In the **Wolf-Sheep Predation model**, the turtles are the *wolves* and *sheep*, the patches represent the *grass* on which they move and feed, there are *no links* in this model, and the *observer* controls the simulation and observes the population changes.

• In the **Traffic Basic model**, the turtles are the *cars*, the patches form the *road* on which the cars move, there are *no links* used, and the *observer* starts and monitors the movement of the cars.

4. Note down what happens when you change parameters (using sliders).

- In the **Wolf-Sheep Predation model**, changing sliders like the number of sheep, wolves, or grass regrowth rate changes how fast populations grow or decline.
- In the **Traffic Basic model**, changing sliders like car number, speed, or slowdown probability affects the traffic flow and congestion.

POST-LAB QUESTIONS & ANSWERS

1. What is NetLogo used for?

NetLogois used forbuilding andsimulating **agent-based models (ABM)**. It helps study how individual agents (like animals, cars,or people) interact with each other and their environment to create overall system behavior.

2. Explain the four types of agents in NetLogo.

- **Turtles:** Moving agents thatrepresentindividuals such as animals, cars,orpeople.
- **Patches:** Thebackground gridcellsonwhich turtles move; they can holdvalueslike color or grasslevel.
- **Links:** Connections betweenturtles,used to show relationships like networksorroads.
- **Observer:** The overall controllerthatruns commands, starts or stops thesimulation, and observesresults.

3. Write the correct steps to install NetLogo on Windows.

1. Goto theofficialNetLogowebsite(https://ccl.northwestern.edu/netlogo/).
2. Click **Download NetLogo** and choose theWindows version.
3. After downloading, openthe installer file(.exe).
4. Follow thesetupwizardand click **Next** tocomplete theinstallation.
5. Once installed,open NetLogo fromthe StartMenu or Desktopshortcut.

4. What are the three main tabs in the interface, and what is each used for?

- **Interface Tab:** Used to run the model, view graphics, sliders, and buttons.
- **Info Tab:** Shows a description of the model, its purpose, and instructions.
- **Code Tab:** Contains the NetLogo programming code that defines the model'sbehavior.

5. Which models did you try, and what patterns did you observe?

I tried the **Wolf-Sheep Predation** and **Traffic Basic** models.

In the Wolf-Sheep model, I observed cycles in population when sheep increase, wolves also increase, then both decrease alternately.

In the Traffic Basic model, I observed how traffic jams form when toomany cars are added or when slowdown probability increases.

LAB 2: WORKING WITH TURTLES IN NETLOGO

OBJECTIVE

- To understand how to create and control turtles in NetLogo.
- To execute turtle commands using the Command Center.
- To visualize how simple movement rules lead to complex behaviors.

LANGUAGE/TOOL

- **Tool:** NetLogo 6.4.0
- **Platform:** Windows
- **Location:** Command Center in the NetLogo interface

THEORY

In NetLogo, turtles are moving agents that can perform actions such as moving, turning, changing color, or interacting with the environment.

Each turtle acts independently, but together they can form patterns or simulate real-world systems such as traffic, animal movement, or people in a crowd.

KEY COMPONENTS

- **Observer:** The global controller that gives commands to all turtles or patches.
- **Turtles:** The moving agents created by the observer.
- **CommandCenter:** Thearea at the bottom where you type commands directly.

COMMON TURTLECOMMANDS

Command	Description
create-turtlesn	Creates *n* turtles at random locations
forward n	Moves the turtle forward *n* steps
back n	Moves the turtle backward *n* steps
left n	Turns the turtle left by *n* degrees
right n	Turns the turtle right by *n* degrees

set color colorname	Changes turtle color
set size n	Changes turtle size
set xcor value / set ycor value	Changes turtle position
random-xcor, random-ycor	Generates random coordinates
repeat [commands]	Repeats a set of commands multiple times
ask turtles [...]	Tells all turtles to do the commands inside brackets

LAB PROCEDURE

Step 1: OpenNetLogo

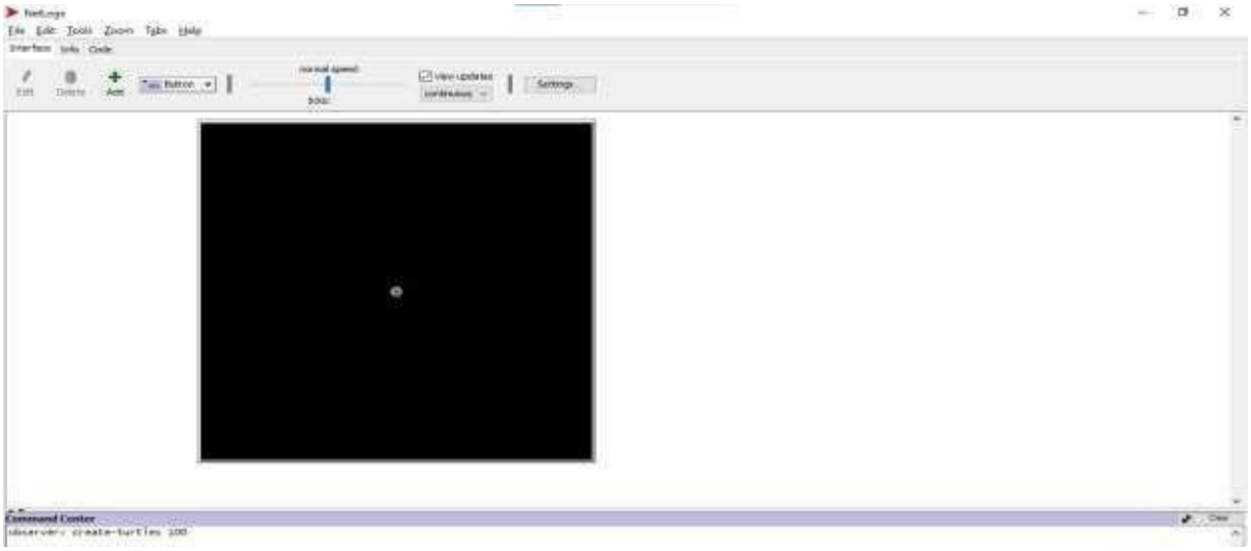
- LaunchNetLogoon your Windows computer.
- You willsee theInterface, Info, and Code tabs.
- At thebottom,locate the Command Center — this is where you will type commands.

Step 2: Run EachCommand

OBSERVER COMMANDS

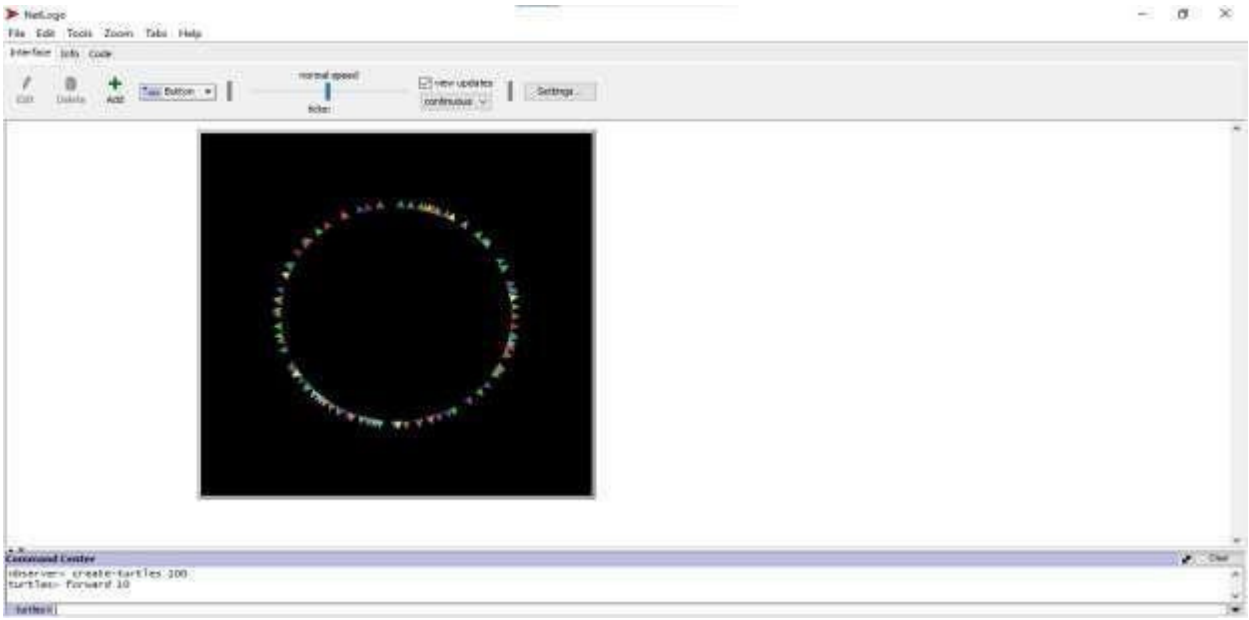
These are typed directly in theCommand Center

1. observer> create-turtles 100



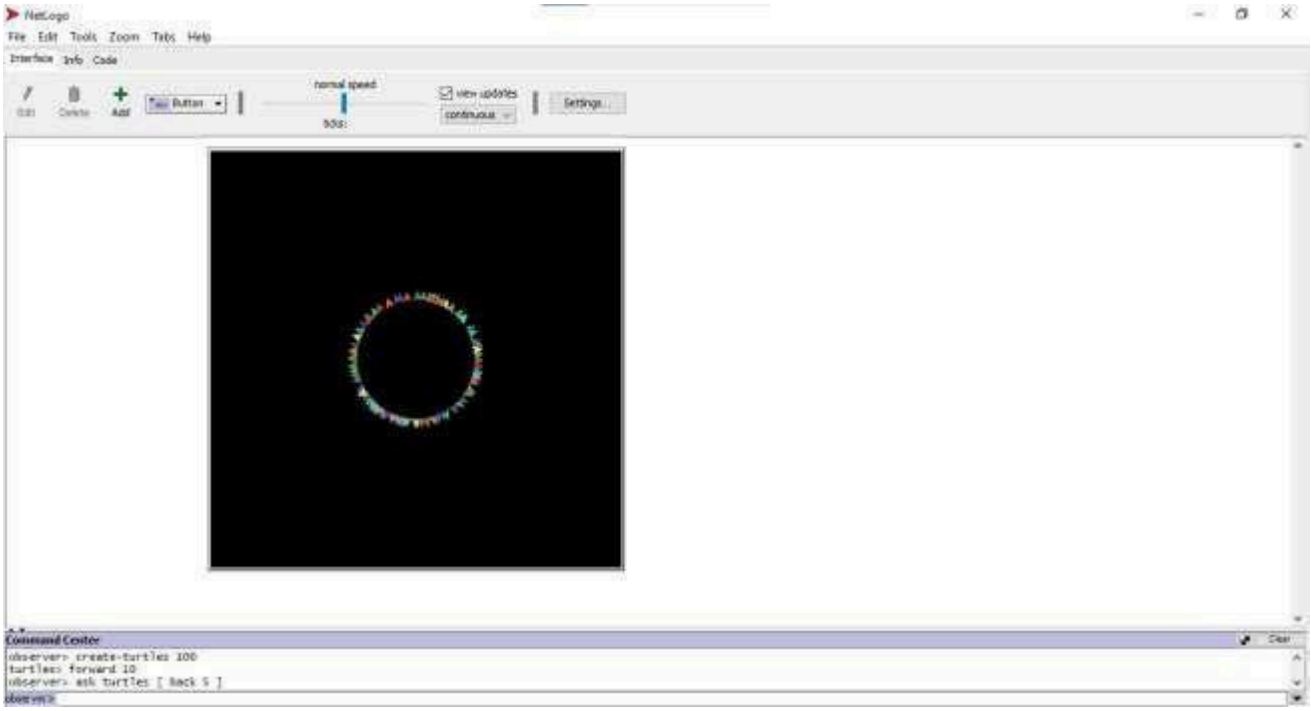
➤ Creates 100 turtles at the center of the world.

2. turtles> forward 10



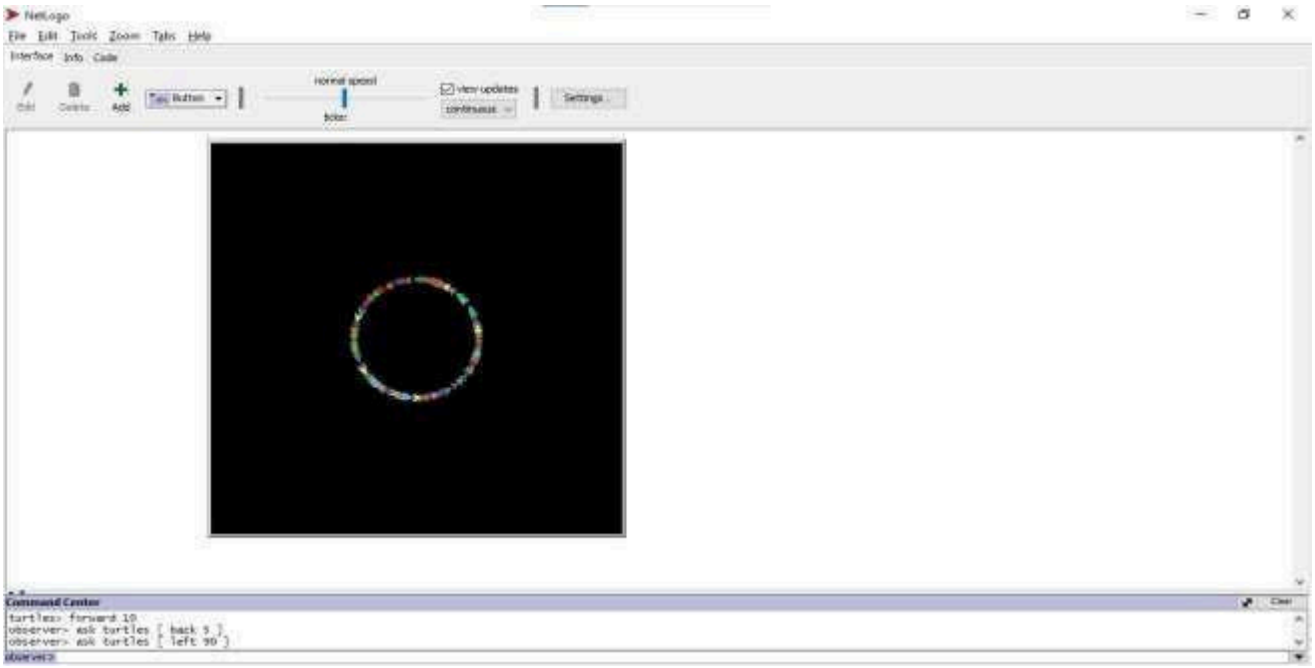
➤ Moves all turtles 10 steps forward from their starting point.

3. observer> ask turtles [back 5]



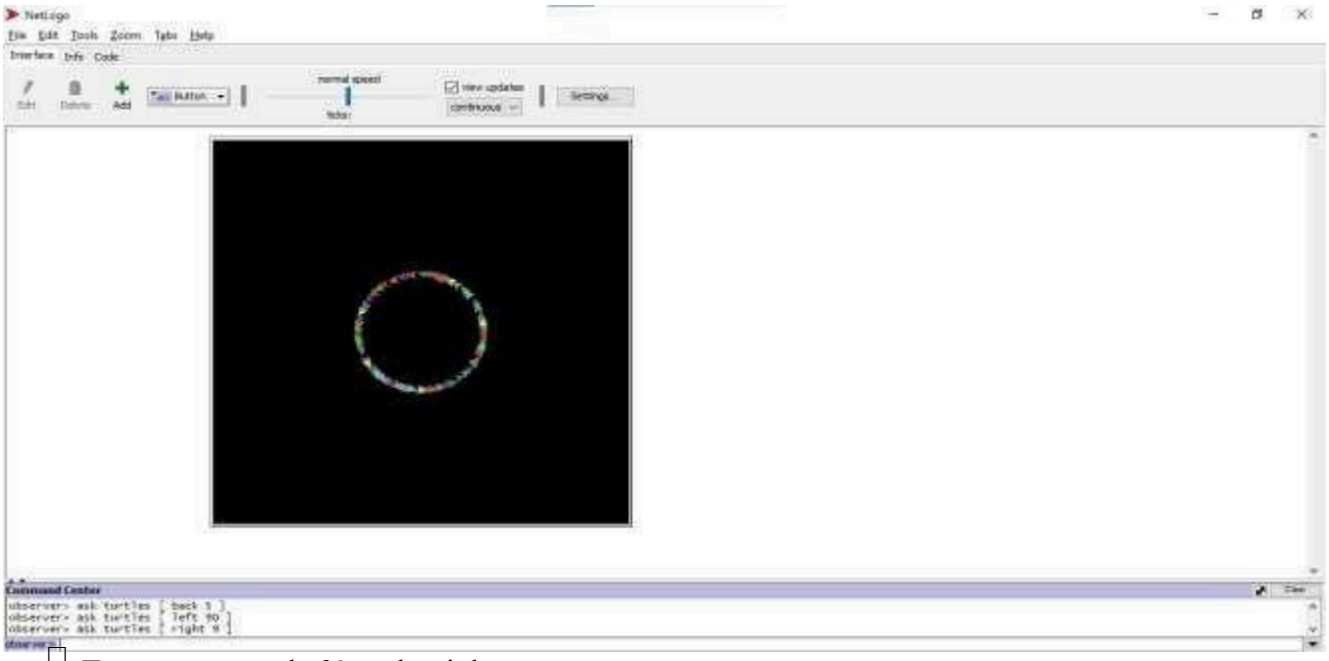
➤ Tells every turtle to move 5 steps backward.

4. observer> ask turtles [left 90]



➤ Turns every turtle 90° to the left.

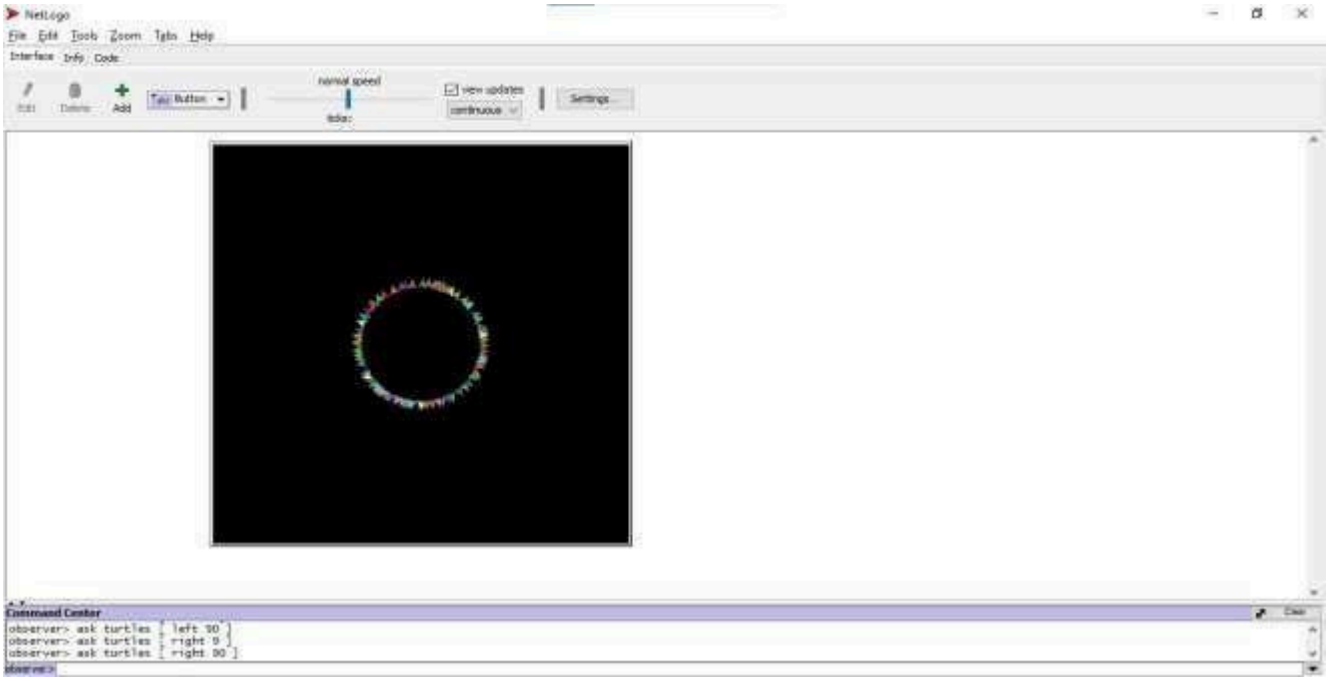
5. observer> ask turtles [right 9]



➤ Turns every turtle 9° to the right.

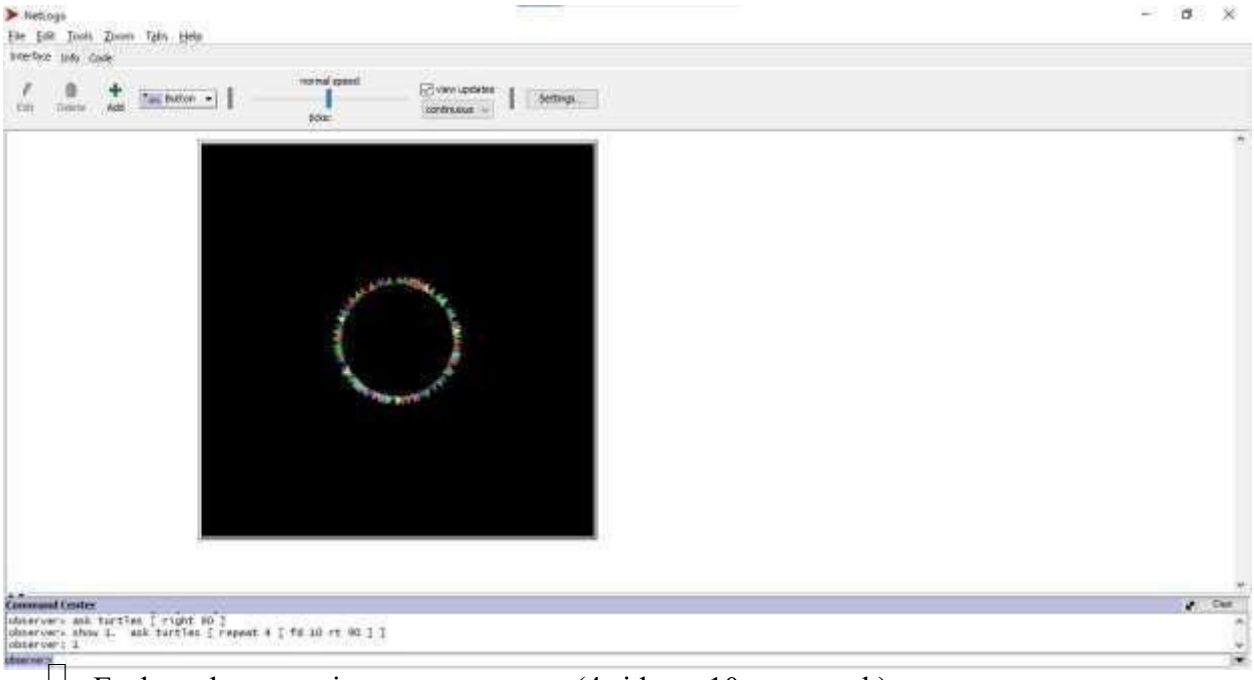
observer> ask turtles

6. [right 90]



➤ Turns every turtle 90° to the right again (back to original direction).

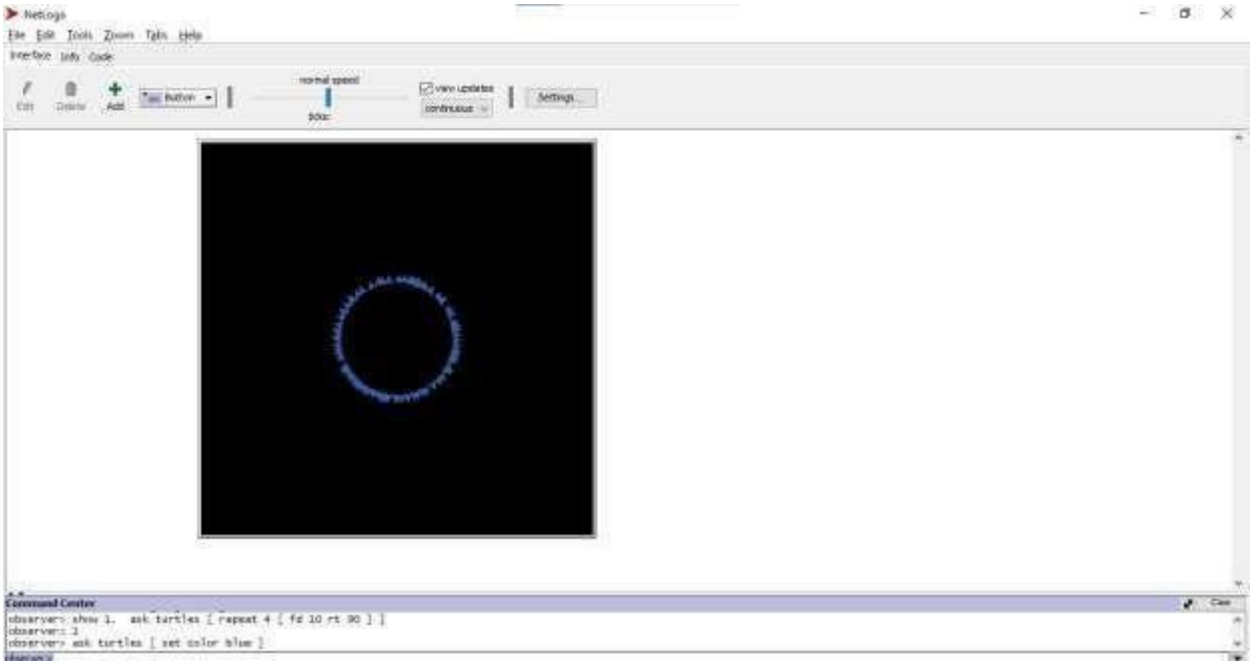
7. observer> ask turtles [repeat 4 [fd 10 rt 90]]



- Each turtle moves in a square pattern (4 sides × 10 steps each).

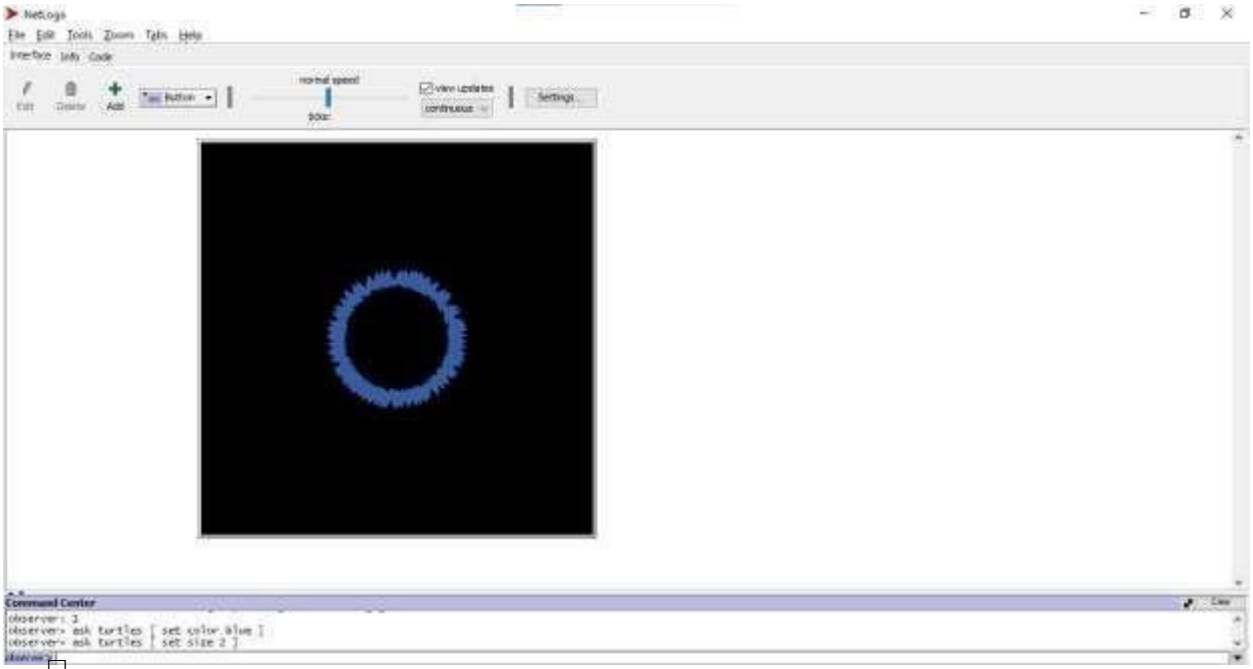
8. [set color blue]

observer> ask turtles



➤ Changes the color of all turtles to blue.

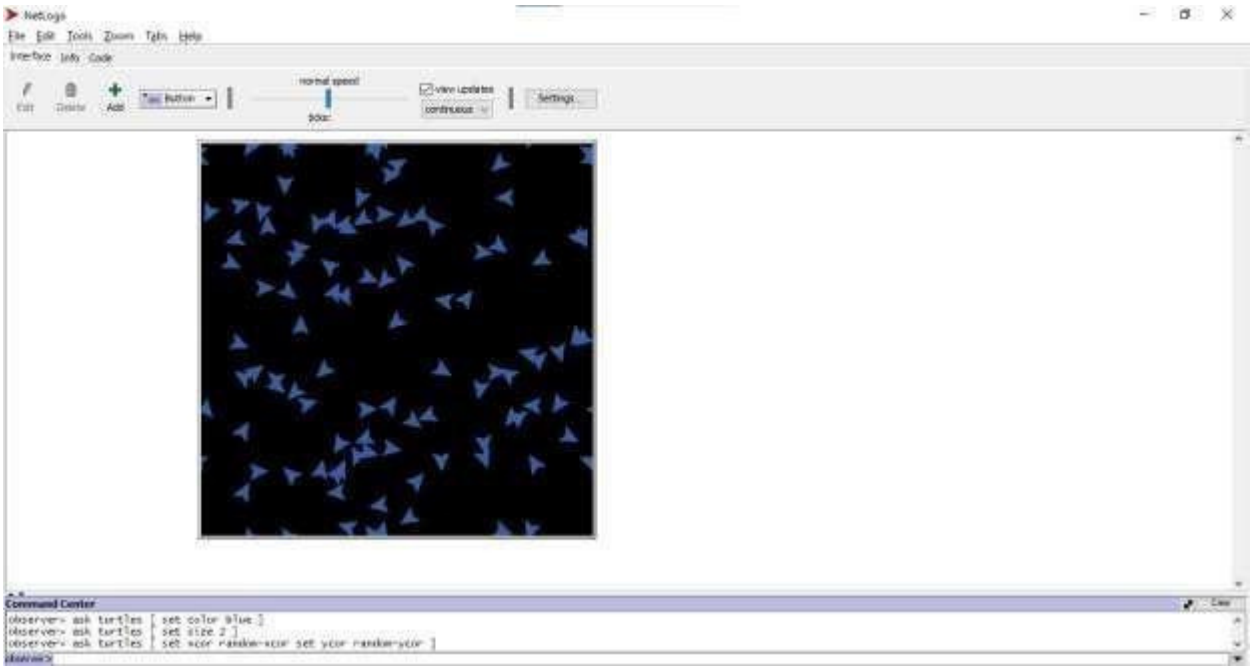
9. observer> ask turtles [set size 2]



- Makes each turtle twice its normal size.

10. [set xcor random-xcor set ycor random-ycor]

observer> ask turtles



➤ Moves each turtle to a random location on the grid.

Step 3:Observethe Results

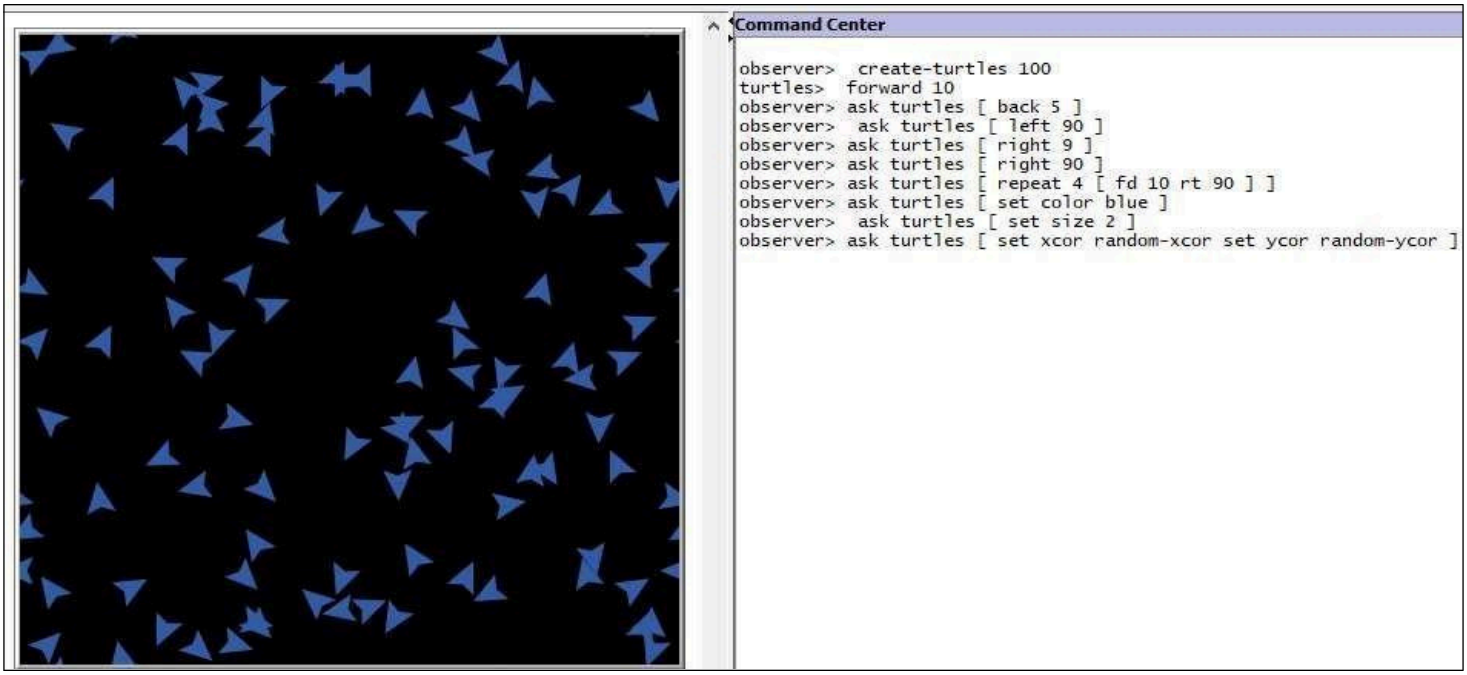
After running each command:

- Notice how the world changes visually.
- Observe movement, color, and size changes.
- Try running commands again with different numbers (e.g., create-turtles 200, fd 20).

STUDENT TASK

observer> ask turtles

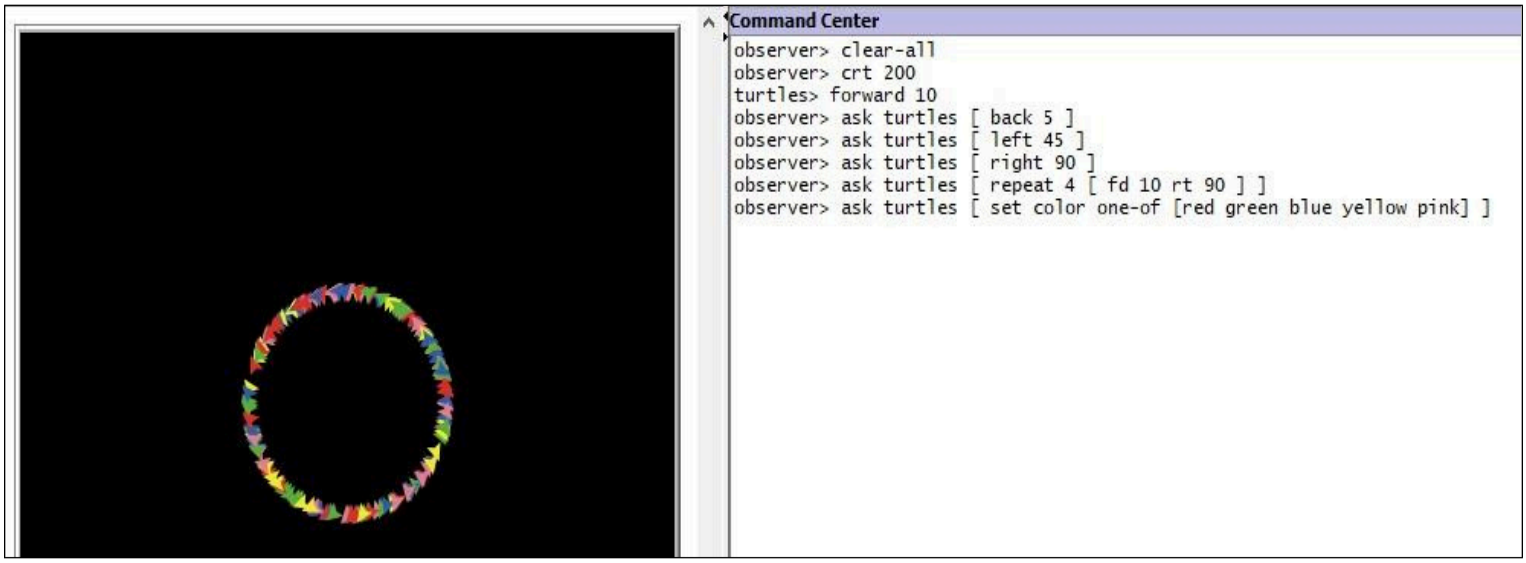
1. E xecute each command listed in the procedure.



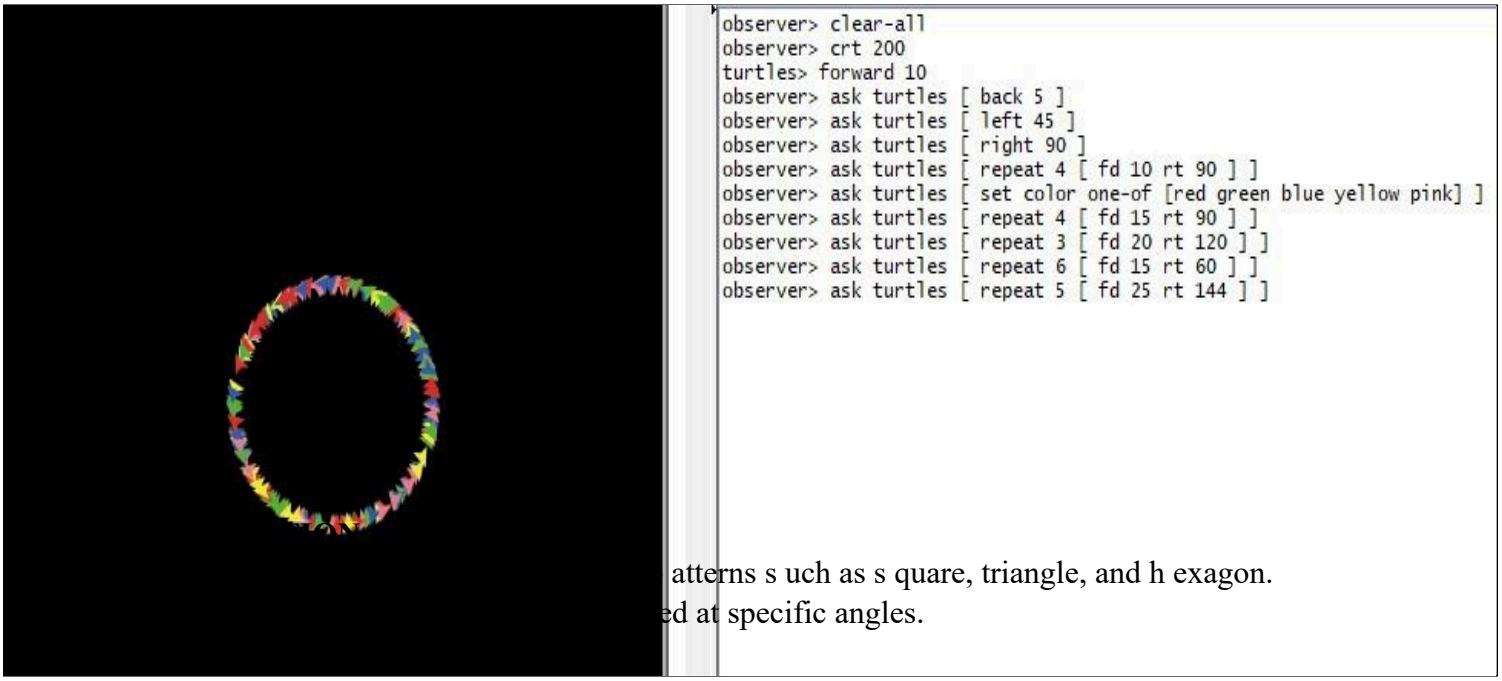
2. Change the number of turtles (e.g., 50, 200) and observe behavior differences.

- 200 turtles created screen looks crowded with many overlapping dots.

3. Change colors using set color red, green, yellow, etc

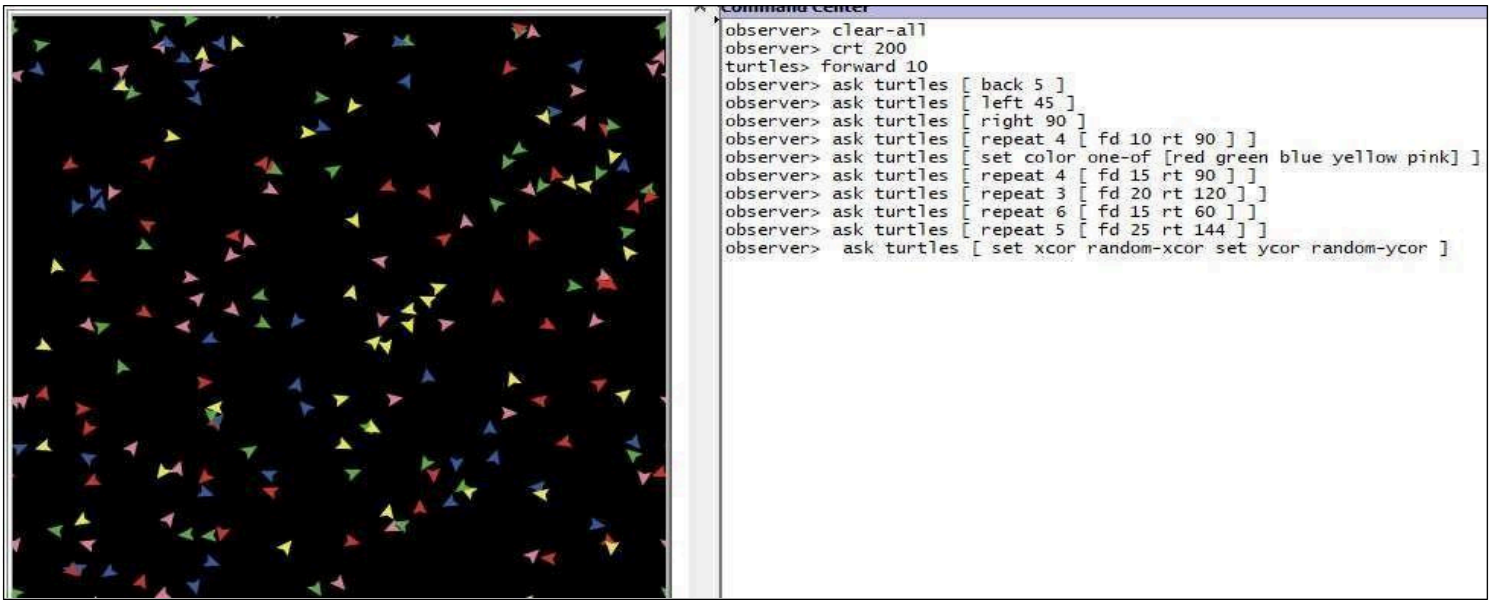


4. Try different movement patterns (e.g., repeat 6 [fd 15 rt 60] for a hexagon).



atterns s uch as s quare, triangle, and h exagon.
ed at specific angles.

5. W rite o bservations i n y our l ab notebook.



OBSERVATION:

- 200colorful turtles created successfully.
- Turtles moved forward, backward, and turned at different angles.
- Formed various geometric patterns like square, triangle, hexagon, and star.
- The world displayed a circular colorful pattern as turtles overlapped.

POST-LAB QUESTIONS

1. What is the purpose of the Command Center?

The Command Center is used to type and run commands to control agents and the model.

2. What does the command ask turtles [...] do?

The command ask turtles [...] tells all turtles to perform the actions inside the brackets.

3. How can you change a turtle's color and size?

You can change a turtle's color with set color red andsize with set size 2.

4. What happens when you use random-xcor and random-ycor?

Using random-xcor and random-ycor moves turtles to randompositions on the world.

5. What shape pattern is made by repeat 4 [fd 10 rt 90]?

The command repeat 4 [fd 10 rt 90] makes a **square** pattern.

LAB 3: DRAWING AND CONTROLLING TURTLES IN NETLOGO

OBJECTIVE

- To learnhow turtlescan draw shapes using movement commands.
- To understand the use of pen, loops, and removing turtles (die).
- To practice giving multiple commands in a single line.

LANGUAGE/TOOL

- **Tool:** NetLogo6.4.0
- **Platform:** Windows
- **Environment:** Command Center

THEORY

InNetLogo,turtles canmove, draw, and erase on the world grid. When the pen is down, each turtle leavesa visibletrail behind it, similar to drawing with a pen on paper. You can use loops such as repeat to createshapes and patterns. Finally, you can use the die command to remove turtles fromthe world.

Important Concepts

Concept	Description
Pen Control	pen-downstarts drawing; pen-up stops drawing.
Repeat Loop	Repeatsasetof commands multiple times.
clear-drawing	Erasesonlythe drawn lines (not the turtles).
die	Deletesaturtle from the simulation.
Multiple Commands	Can bewritten inside brackets [ ] separated by spaces.

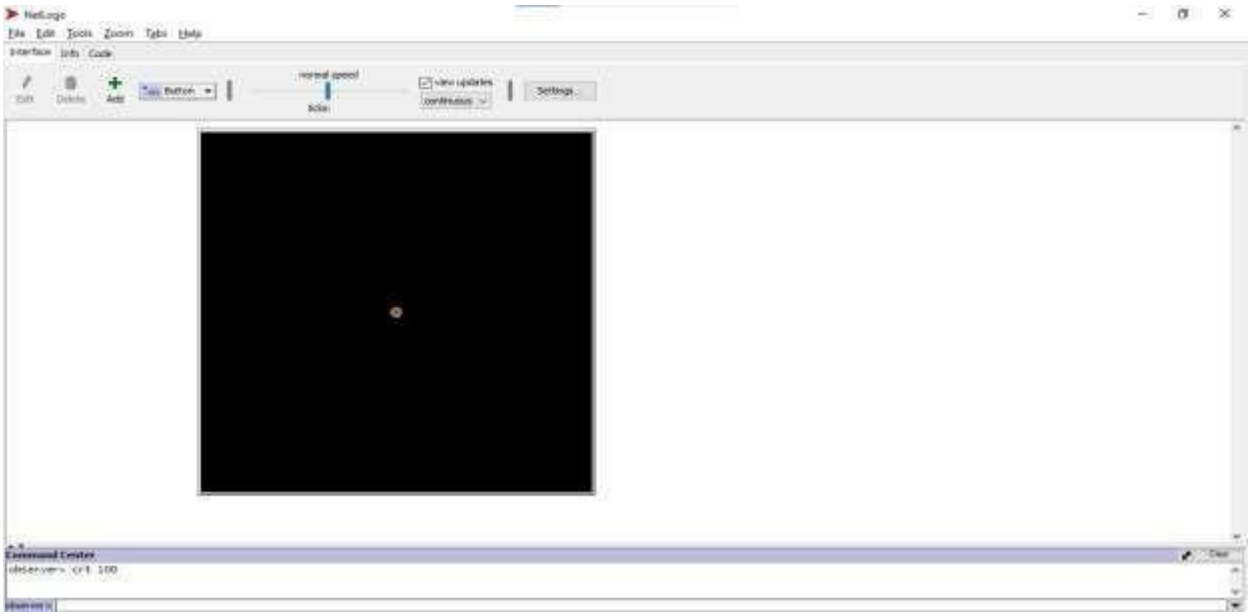
LAB PROCEDURE

Step 1: Open NetLogo

1. Start NetLogo.
2. Make sureyou are in theCommand Center (bottom window).
3. Keep thedefault mode asobserver.
4.

Step 2: Run Commands Step by Step

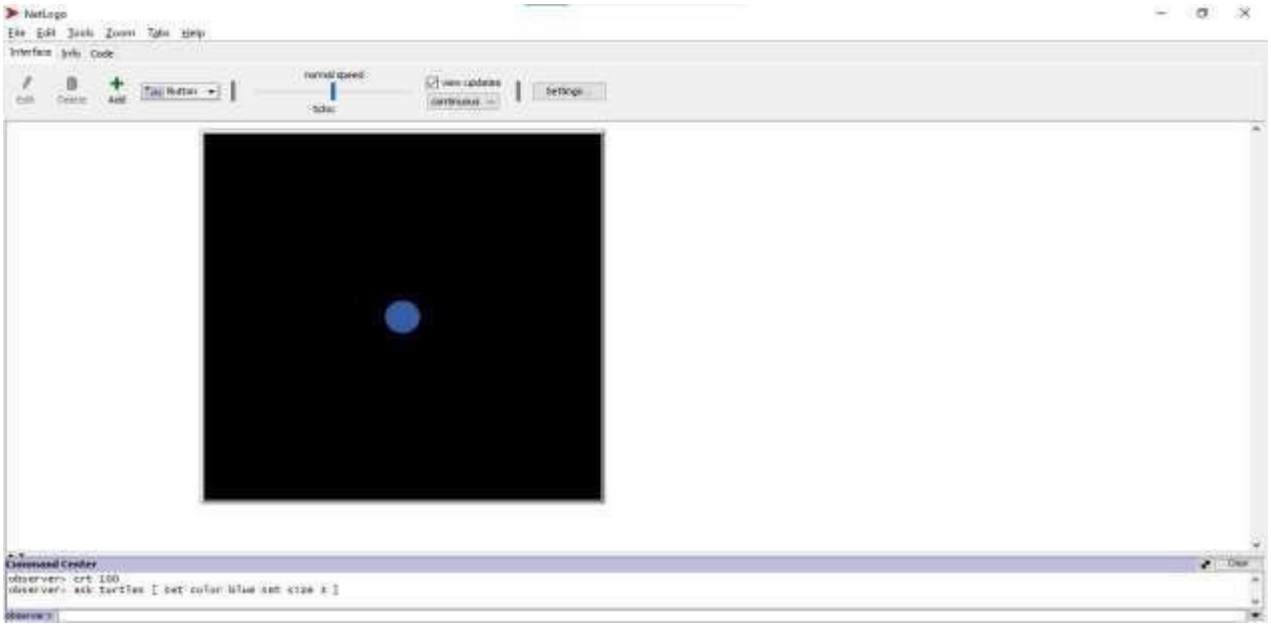
1. **Create Turtles** observer>crt100



➤ Creates 100 turtles at the center of the world.

2. Set Turtle Color and Size

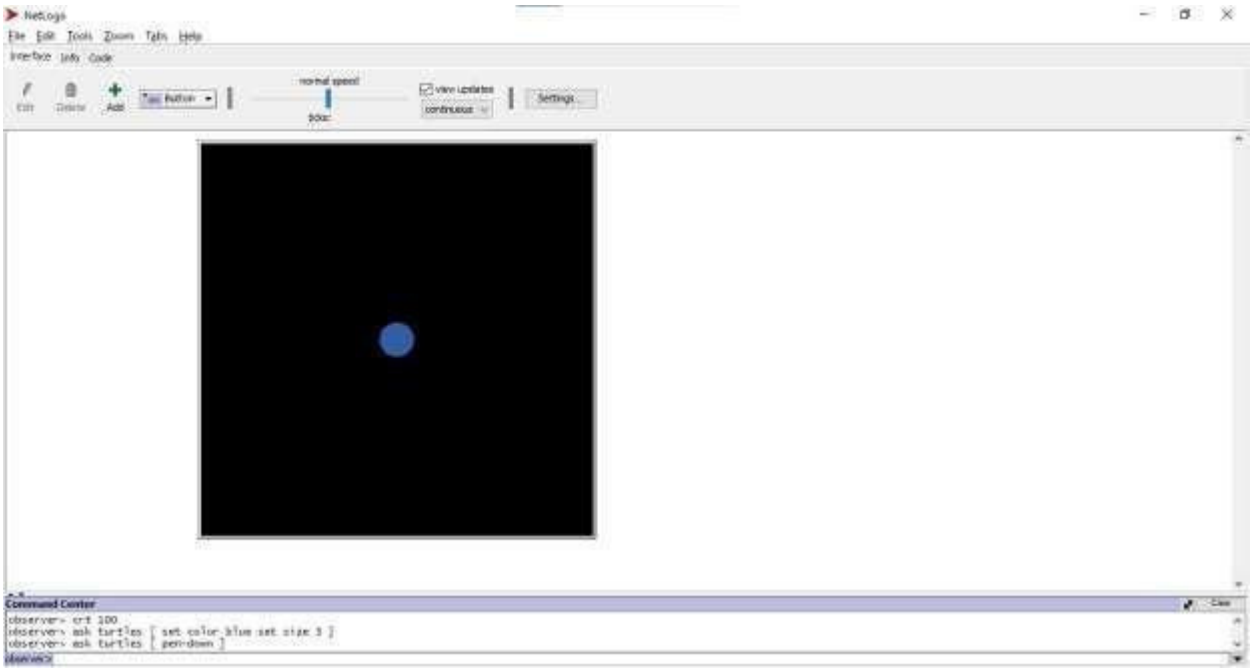
observer> ask turtles [set color blue set size 3]



➤ Changes all turtles to blueand makes them larger (size 3).

3. Enable Pen Drawing

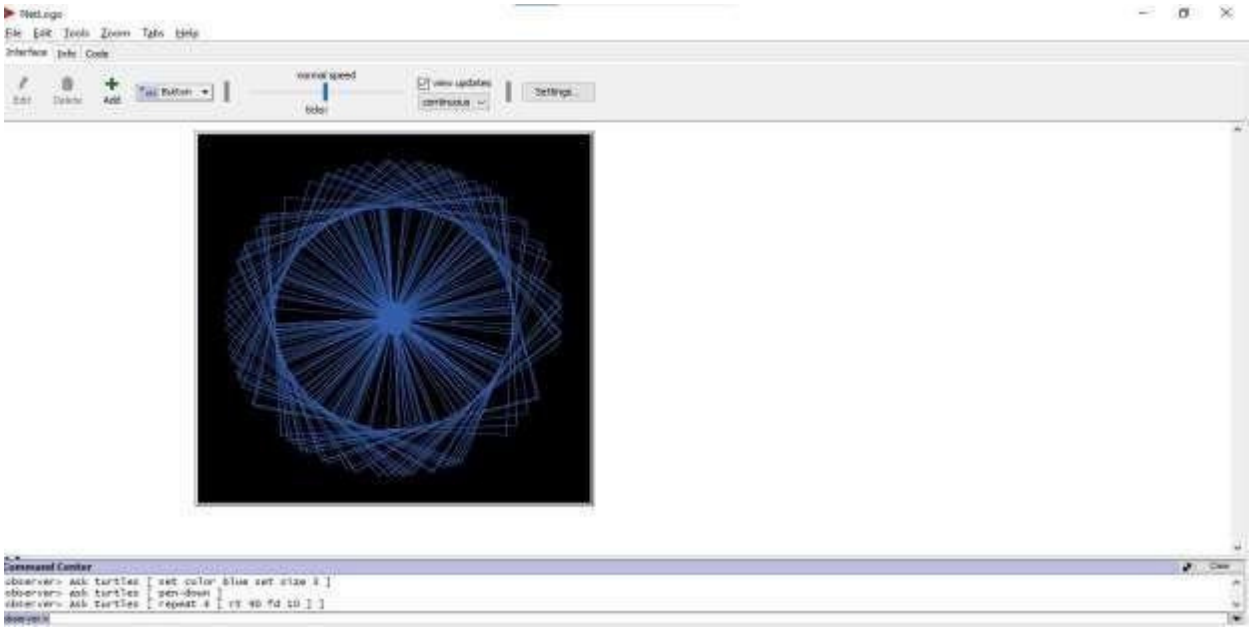
observer> ask turtles [pen-down]



➤ Lowers the pen for all turtles — now their movement will leave trails.

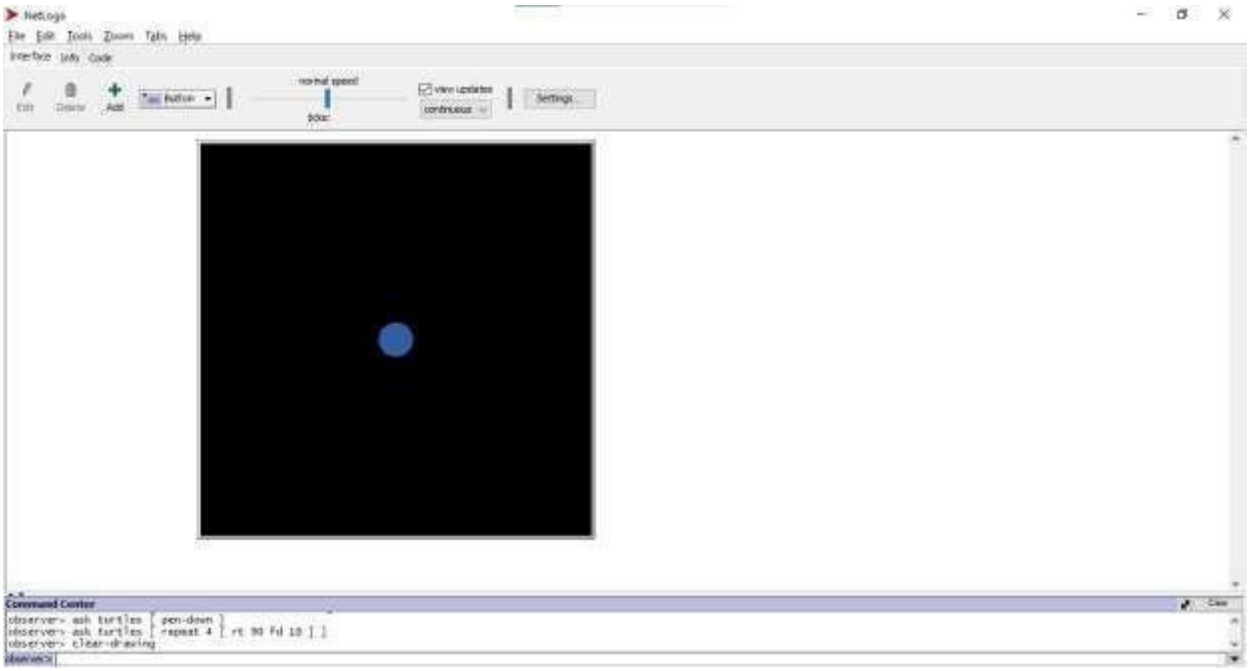
4. Draw a Square Shape

observer> ask turtles [repeat 4 [rt 90 fd 10]]



➤ Each turtle draws a small square by turning right 90° four times and moving forward 10 units each time.

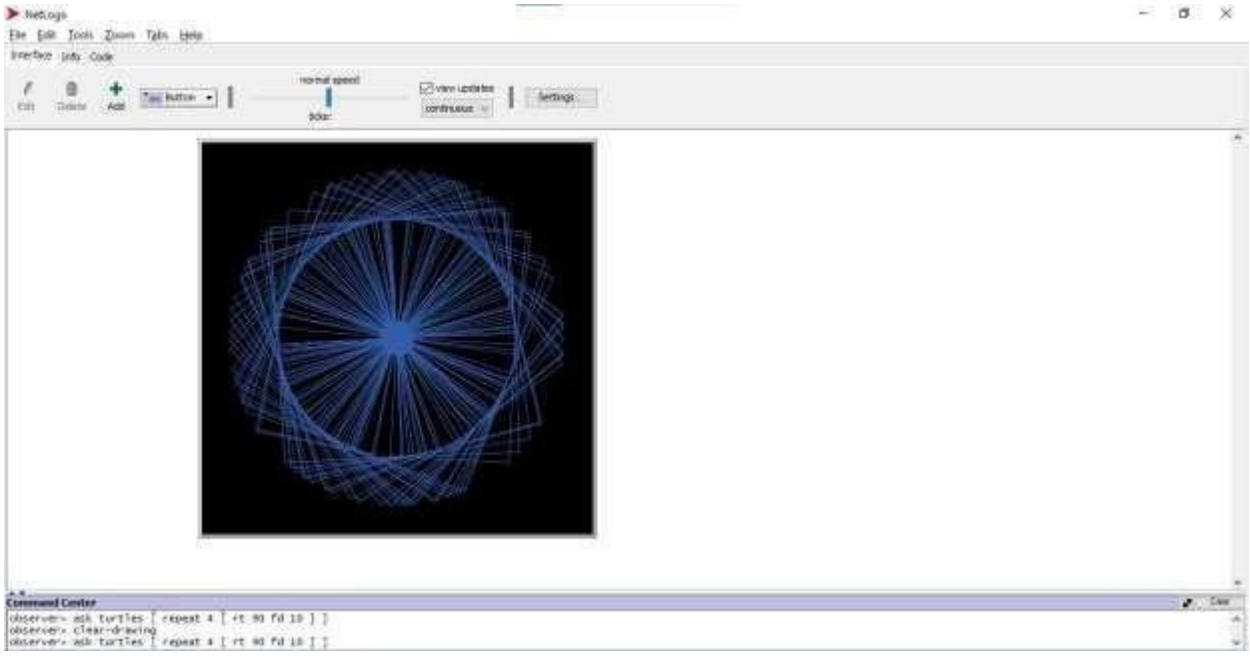
5. Clear Only the Drawing observer> clear-drawing



➤ Erases all trails drawn byturtles but keeps the turtles themselves.

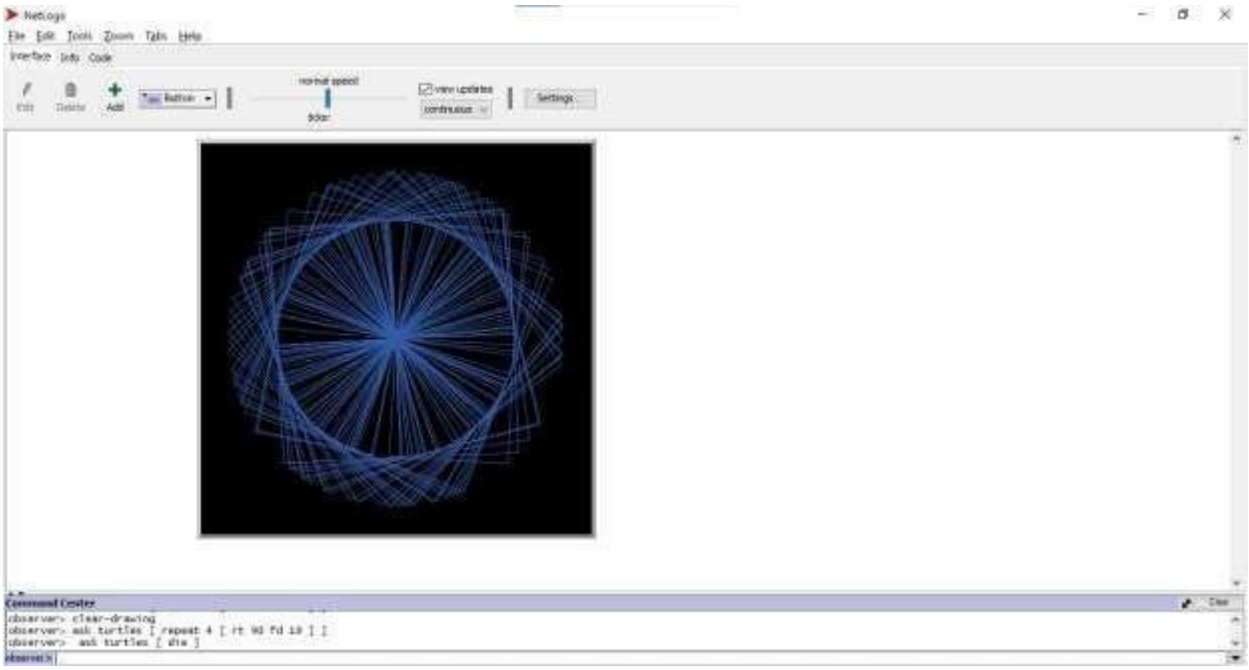
6. Redraw the Shape

observer> ask turtles [repeat4 [rt 90 fd 10]]



➤ Draws new squares again in the same way.

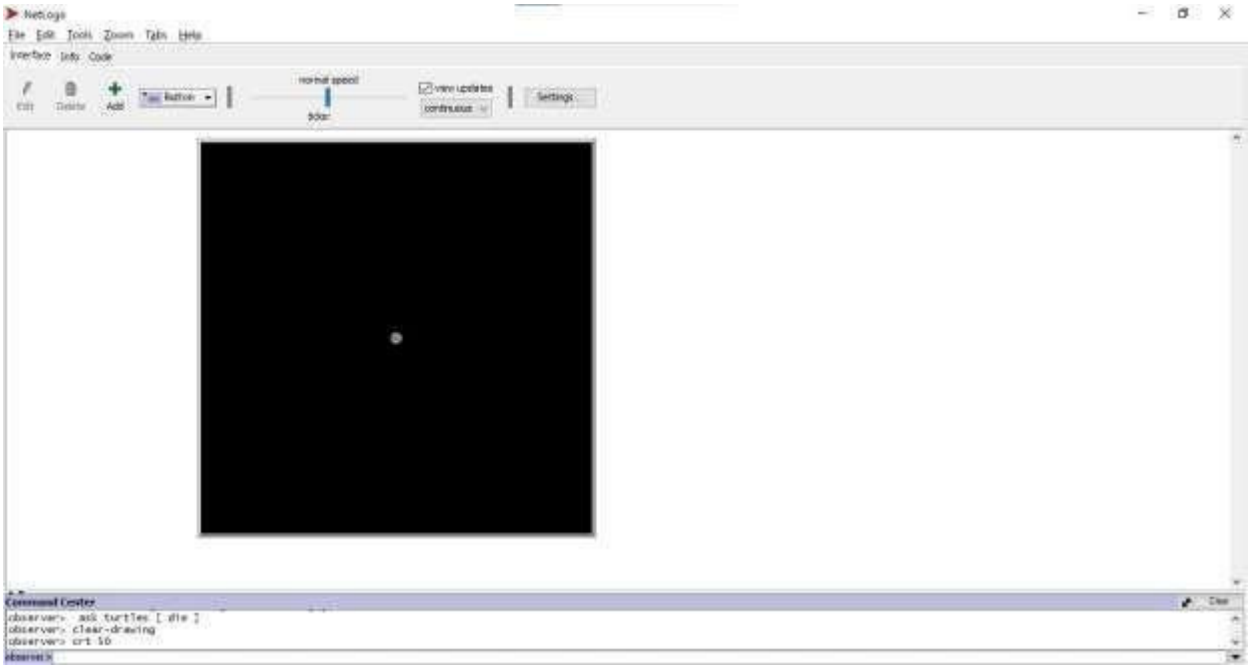
7. Delete All Turtles observer> ask turtles [die]



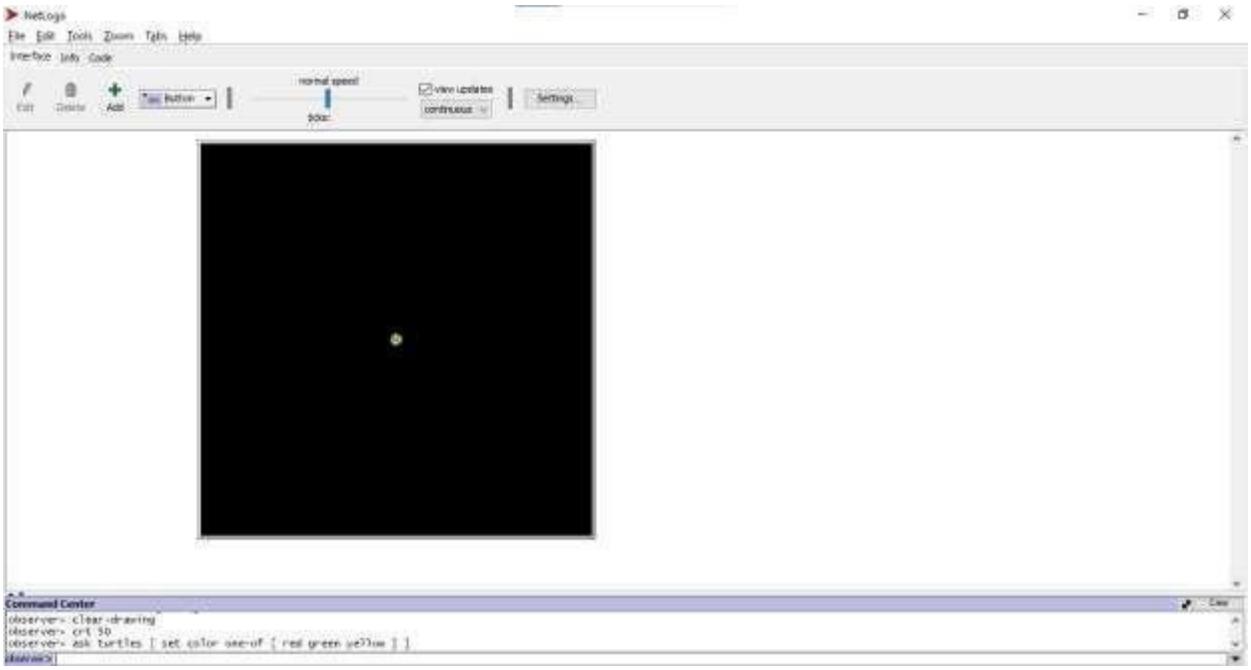
➤ Removes all turtles from the world.

ADDITIONAL PRACTICE COMMANDS

8. Create New Colored Turtlesobserver> crt 50

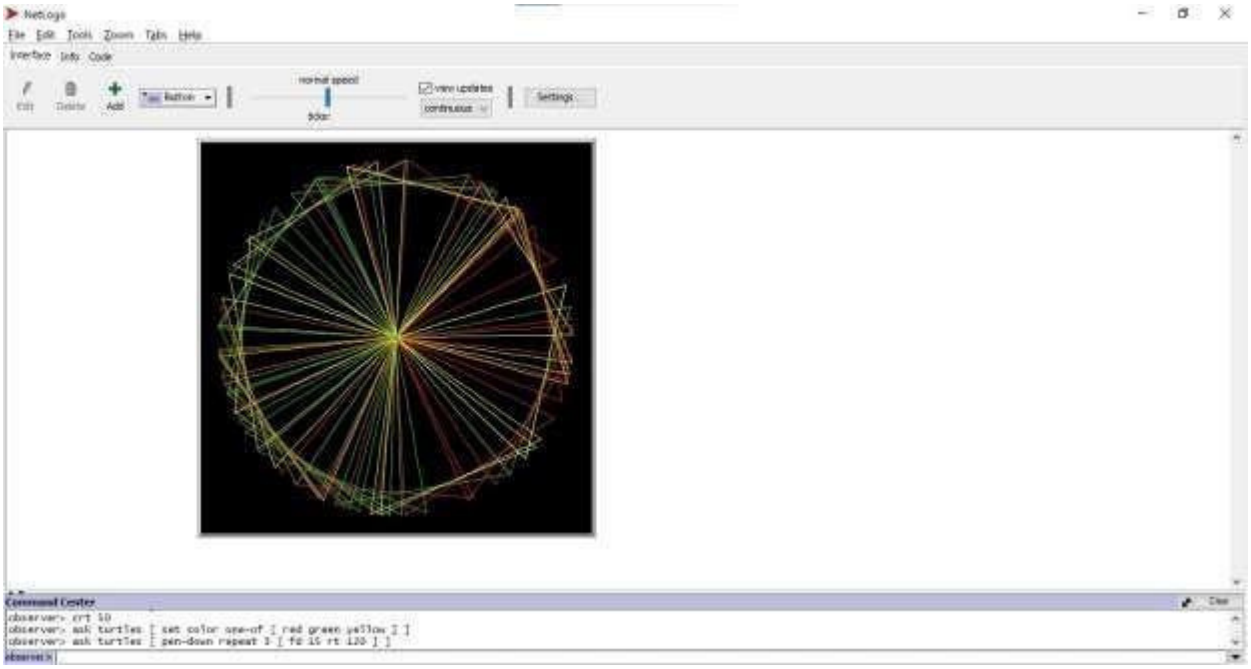


observer> ask turtles [set color one-of [red green yellow]]



➤ Creates 50 turtles with randomly selected colors.

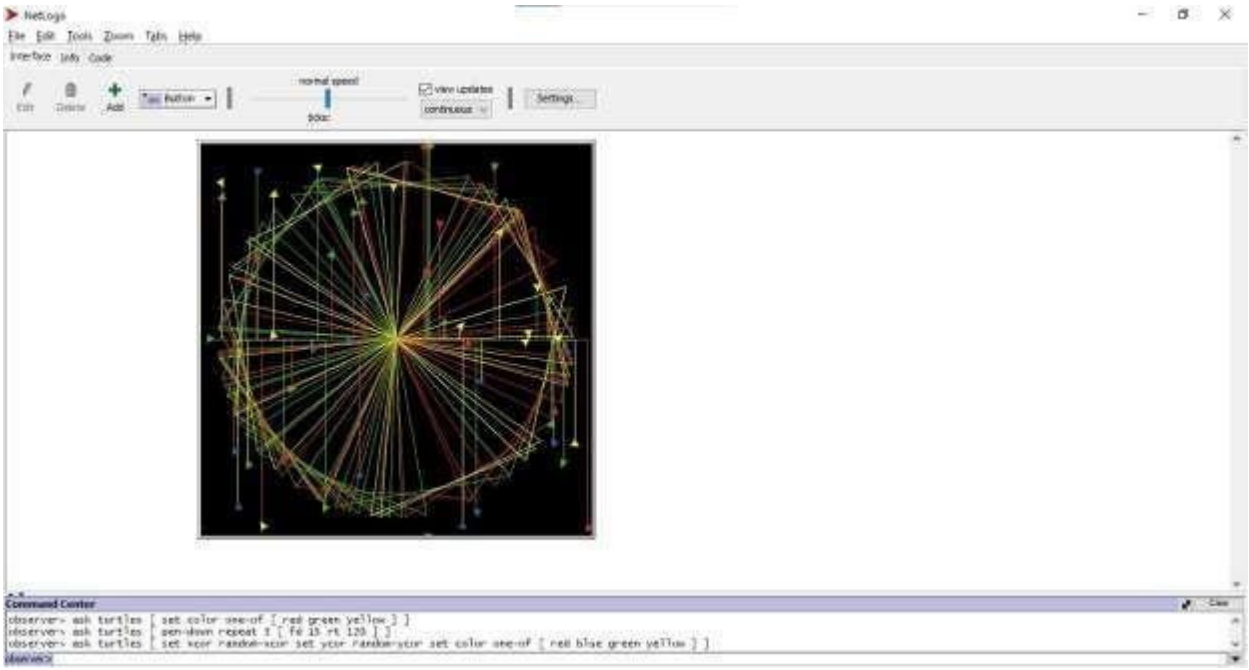
9. Draw Triangles observer> ask turtles [pen-down repeat 3 [fd 15 rt 120]]



➤ Each turtle draws a triangle.

10. Random Positions and Random Colors

observer> ask turtles [set xcor random-xcor set ycor random-ycor set color one-of [red blue green yellow]]



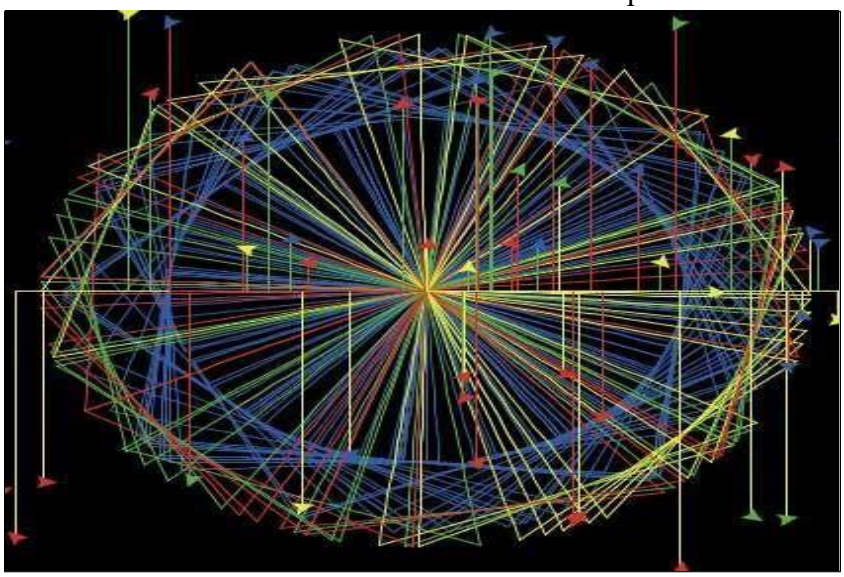
➤ Moves turtles torandom locations and assigns them random colors.

OBSERVATION

- Noticehow each turtle draws its own figure.
- Observedifferences when you clear the drawing vs. deleting turtles.
- Try running commands repeatedly and see how patterns form.

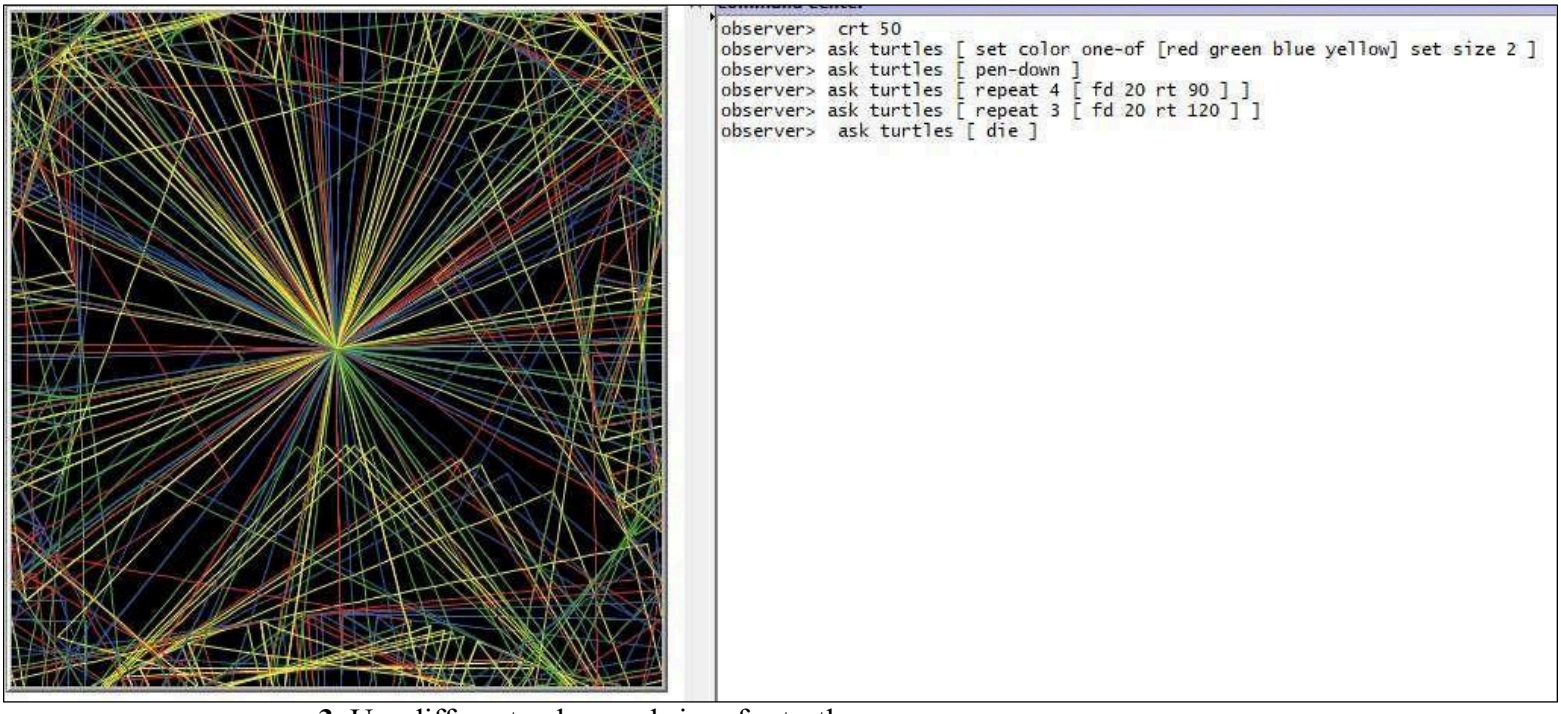
STUDENTTASK

1. E xecute all the commands l isted above in sequence.



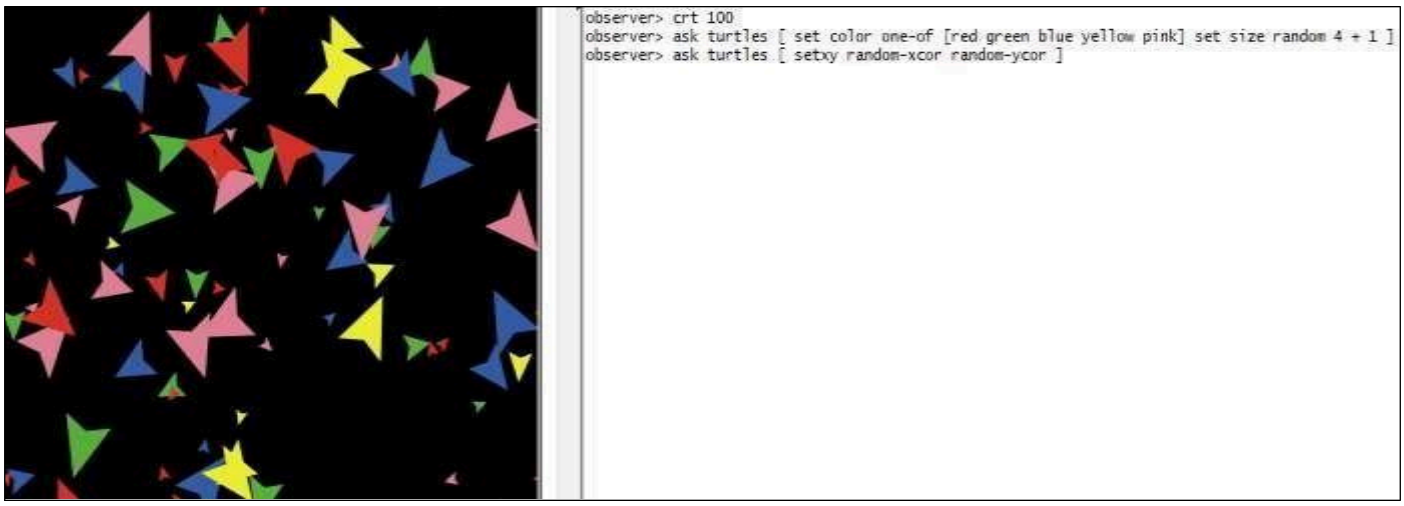
```
Command Center
observer> crt 100
observer>  ask turtles [ set color blue set size 3 ]
observer> ask turtles [ pen-down ]
observer> ask turtles [ repeat 4 [ rt 90 fd 10 ] ]
observer> clear-drawing
observer> ask turtles [ repeat 4 [ rt 90 fd 10 ] ]
observer>  ask turtles [ die ]
observer> crt 50
observer> ask turtles [ set color one-of [ red green yellow ] ]
observer> ask turtles [ pen-down repeat 3 [ fd 15 rt 120 ] ]
observer> ask turtles [ set xcor random-xcor set ycor random-ycor set color one-of [ red blue green yellow ]]
```

2. Create your own pattern by combining commands (e.g., square + triangle).



```
observer>  crt 50
observer> ask turtles [ set color one-of [red green blue yellow] set size 2 ]
observer> ask turtles [ pen-down ]
observer> ask turtles [ repeat 4 [ fd 20 rt 90 ] ]
observer> ask turtles [ repeat 3 [ fd 20 rt 120 ] ]
observer>  ask turtles [ die ]
```

3. Use different colors and sizes for turtles.



4. Compare the effects of clear-all, clear-drawing, and ask turtles [die].

Command	**Deletes Turtles?**	**Deletes Drawings?**	**Deletes Patches/World?**
clear-all	Yes	Yes Yes No	Yes No No
clear-drawing	No		
ask turtles [die]	Yes		

5. Record your observations in your notebook.

Turtle Pattern: Custom pattern (square + triangle) created successfully.
Colors & Sizes: Different colors and sizes made turtles easily distinguishable.
Spreading Turtles: Turtles spread apart clearly, pattern visible.
Clear Commands:

- clear-all → Everything cleared.
- clear-drawing → Drawingserased, turtles remain.
- askturtles [die] →Turtlesgone, drawings remain.

POST-LAB QUESTIONS ANSWERS

1. What is the purpose of the pen-down command?

It allows turtles to draw lines as they move. Without it, turtlesmove without leaving a trail.

2. What is the difference between clear-drawing and clear-all?

• clear-drawing → **erases only the drawings** ,turtlesand patches remain.

• clear-all → **resets everything** , including turtles, drawings, and patches.

3. How many times does the loop run in repeat 4 [rt 90 fd 10]?

The loop runs **4 times**, executing rt 90 fd 10 each time.

4. What happens when ask turtles [die] is executed?

All turtles are removed from the world, but drawings and patches remain.

5. How can you make turtles draw a triangle or hexagon pattern?

Use **loops** with fd (forward) and rt/lt (turn) commands:

- Triangle → repeat 3 [fd length rt 120]
- Hexagon → repeat 6 [fd length rt 60]

LAB4:WORKINGWITH PATCHES, TURTLES, AND LINKS IN NETLOGO

OBJECTIVE

- Tolearn how to manipulate patches and their colors.
- Toexplore turtle positioning using random coordinates.
- Tounderstand how turtles can interact and connect using links.
- Topractice combining multiple commands for complex visuals.

LANGUAGE/TOOL

- **Tool:** NetLogo 6.4.0
- **Platform:** Windows
- **Environment:** Command Center

THEORY

In NetLogo,the world is made up of patches (background cells) and turtles (moving agents). Patches canchange color or hold values, while turtles can move, draw, and form links to represent relationshipsbetween them.

- **Patches:** Stationary cells identified by coordinates (pxcor, pycor).
- **Turtles:** Mobile agents that can act independently or in groups.
- **Links:** Connections between turtles, useful for visualizing networks.

By using commands likeset pcolor, setxy, and create-link-with, we can simulate real-world systems suchas communication networks or spatial environments.

IMPORTANT CONCEPTS

Concept	Description
Patch Coloring	Patches canbe colored based on mathematical formulas or turtle interaction.
Random Positioning	setxy random-xcor random-ycor places turtles at random world coordinates.
Links	Connect turtles to form network structures.
facexy	Rotates turtles to face a specific coordinate point.
MultipleCommands	Combined inside [ ] for sequential execution.

LAB PROCEDURE

Step 1: Open NetLogo

1. Start NetLogo.
2. Ensure you are in the Command Center (bottom window).
3. Keep the mode as observer.

Step 2: Run Commands Step by Step

1. **Color Patches by Coordinates**

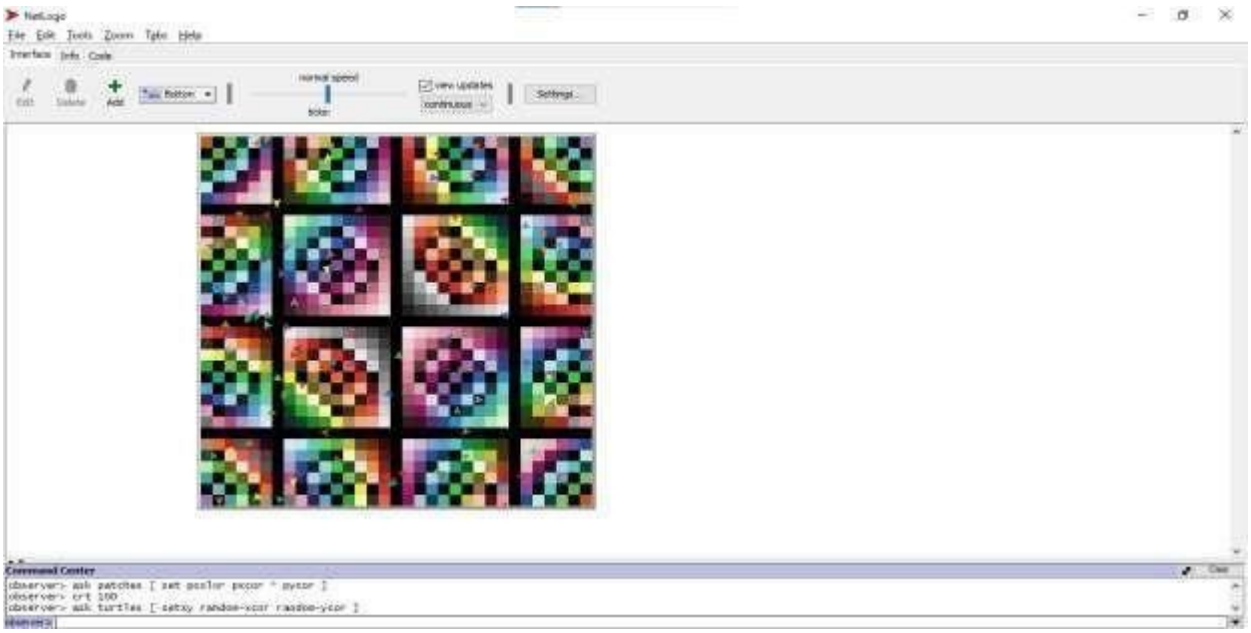
observer> ask patches [set pcolor pxcor * pycor]

➤ Colors each patch using the product of its X and Y coordinates, creating a gradient effect.

2. **Create 100 Turtle** observer> crt 100

➤ Generates 100 turtles at the center of the world.

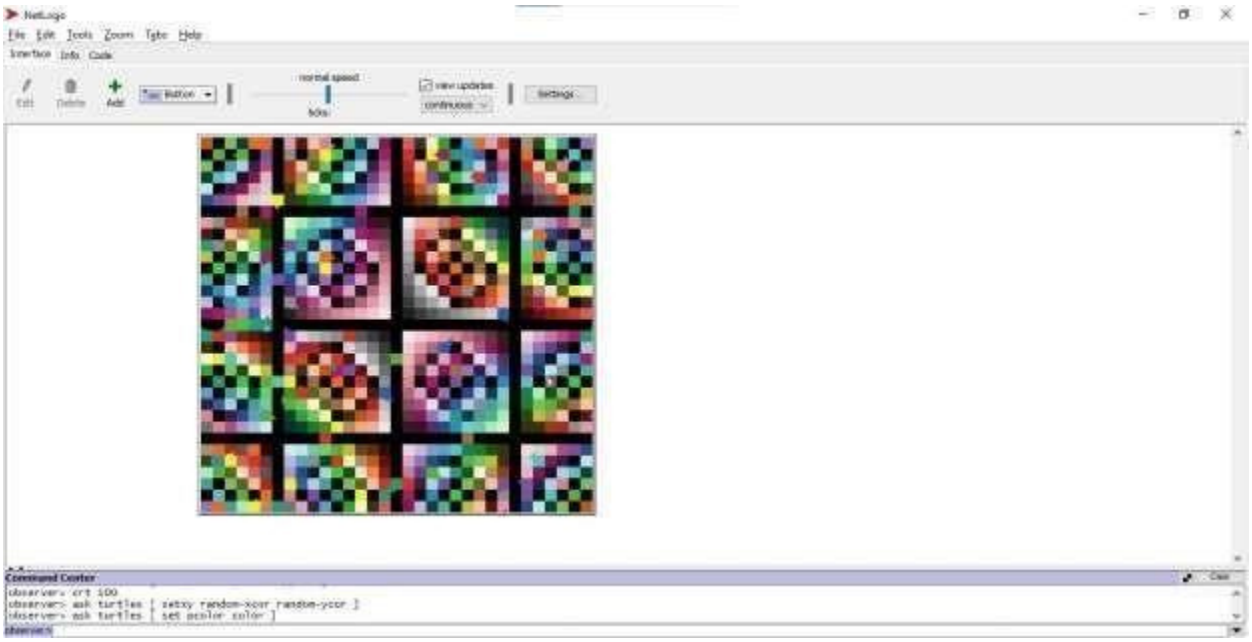
3. **Move Turtles to Random Locations** observer> ask turtles [setxy random-xcor random-ycor]



➤ Randomly scatters turtles across the world.

4. **Change the Patch Color Under Each Turtle**

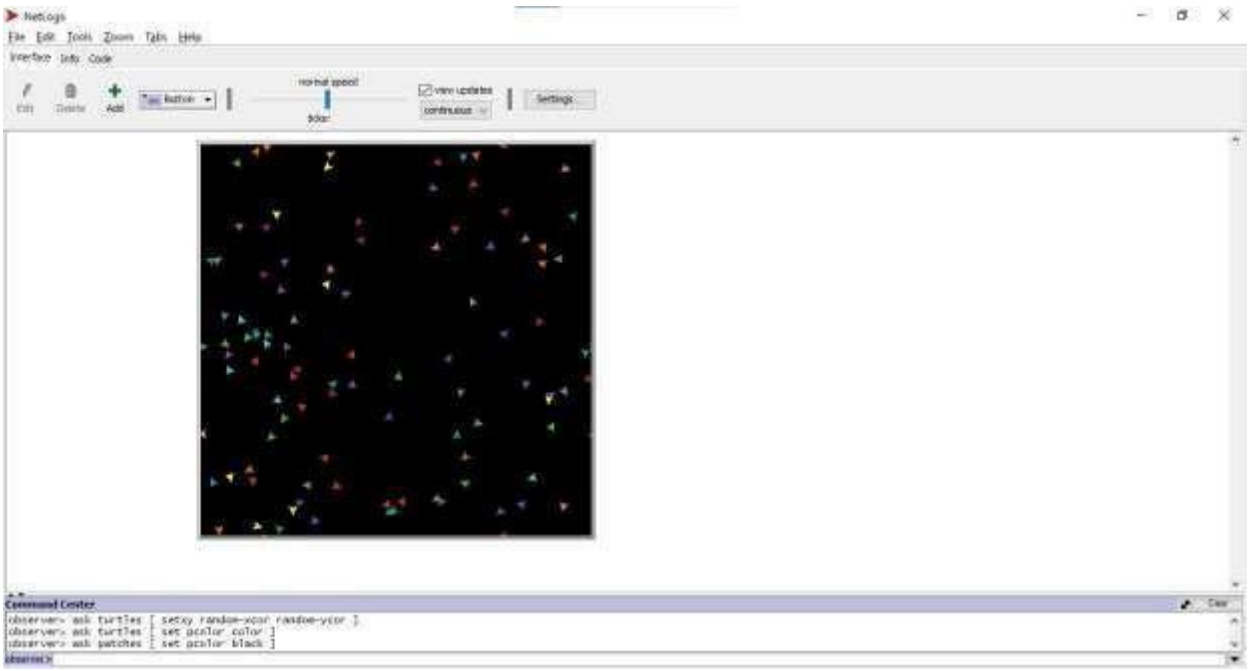
observer> ask turtles [set pcolor color]



➤ Each turtle colors the patch it stands on using its own color.

5. Reset All Patches to Black

observer> ask patches [set pcolorblack]



➤ Turns all patches black while turtles remain visible.

6. MakeTurtlesFaceTheirOwnCoordinates

observer> ask turtles [facexy pxcor pycor]

➤ Orients each turtle toward its own (x, y) location.

observer>

5. **Create Links Between Turtles**

ask turtles [create-link-with one-of other turtles

]

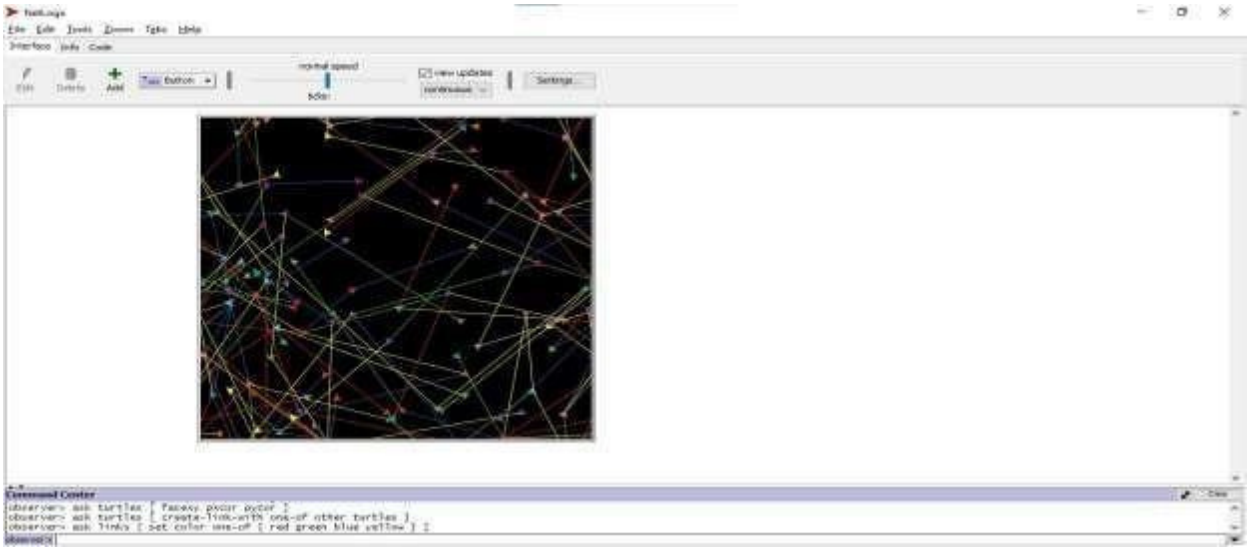
➤ Each turtle randomly connects to another turtle, forming a network pattern.

Step 3: Additional Practice Commands 8. **Color the Links**

Randomly observer> ask links [set color one-of [red green

blue yellow]]

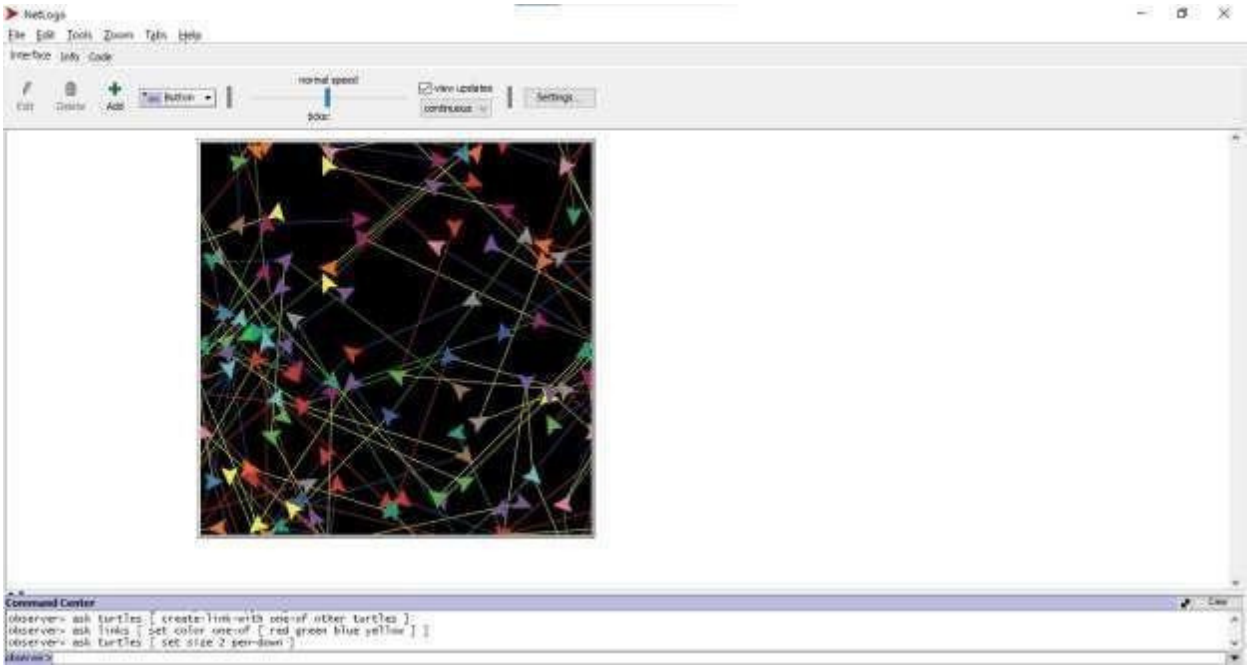
observer>



➤ Assigns random colors to all links.

9. **Resize and Enable Drawing**

ask turtles [set size 2 pen-down]

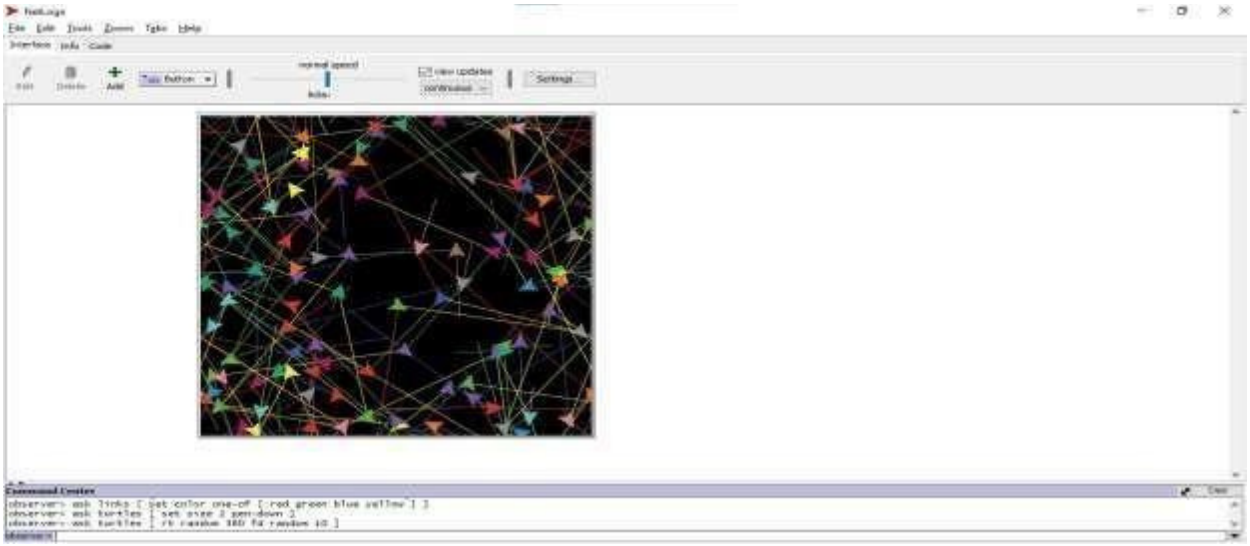


➤ Increases turtle size and allows drawing as they move.

10. **Move Turtles Randomly**

observer> ask turtles [rt random 360 fd random 10]

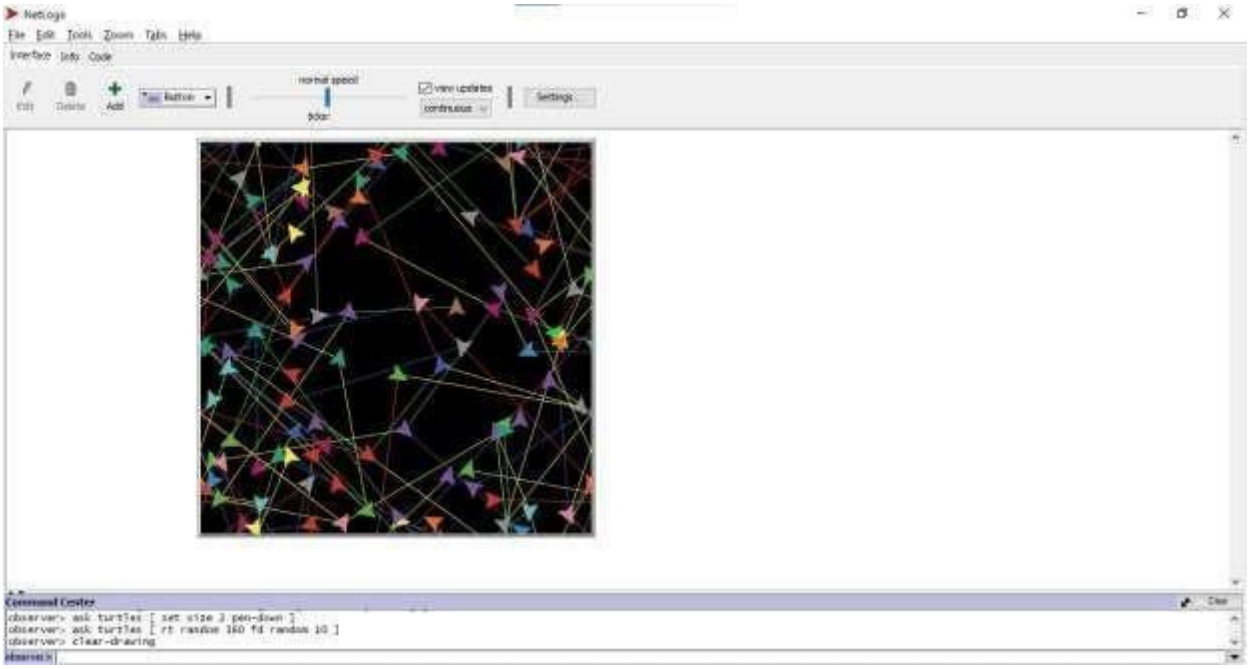
observer>



➤ Moves each turtle forward randomly to create abstract drawings.

11. **Clear Only the Drawing**

clear-drawing



➤ Erases trails but keeps turtles and links intact.

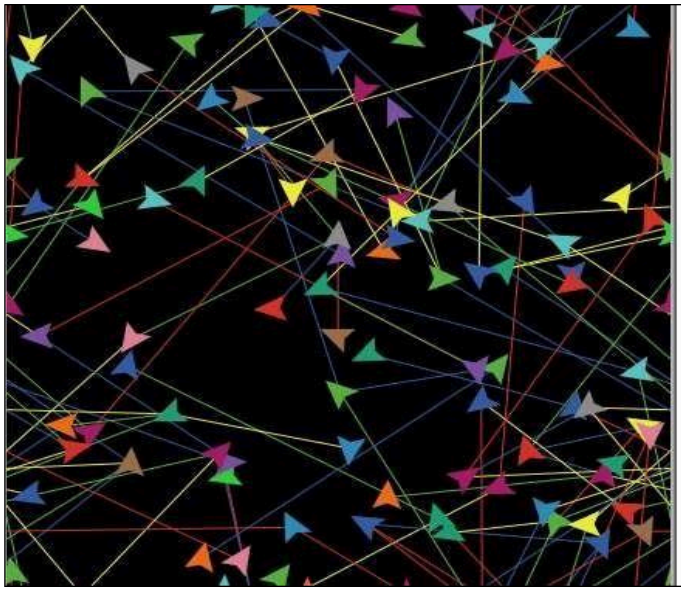
observer>

OBSERVATION

- Observe how patch color patterns change.
- Notice the random distribution of turtles and link formations.
- Experiment with the number of turtles (crt 50, crt 200) and observe the differences.

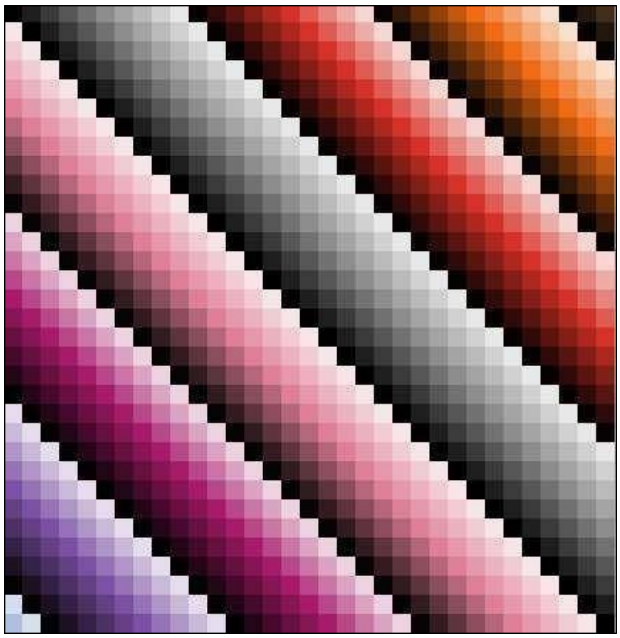
STUDENT TASK

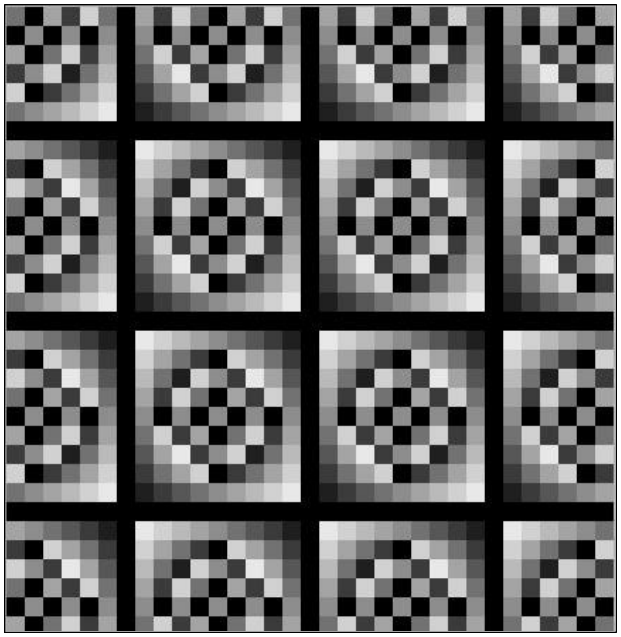
1. Runallthecommands i n s equence.



```
observer>  ask patches [ set pcolor pxcor * pycor ]
observer> crt 100
observer> ask turtles [ setxy random-xcor random-ycor ]
observer>  ask turtles [ set pcolor color ]
observer> ask patches [ set pcolor black ]
observer>  ask turtles [ facexy pxcor pycor ]
observer>  ask turtles [ create-link-with one-of other turtles ]
observer> ask links [ set color one-of [ red green blue yellow ] ]
observer>  ask turtles [ set size 2 pen-down ]
observer>  ask turtles [ rt random 360 fd random 10 ]
observer> clear-drawing
```

2. M odify p atch coloring formulas (e.g., pxcor + pycor).



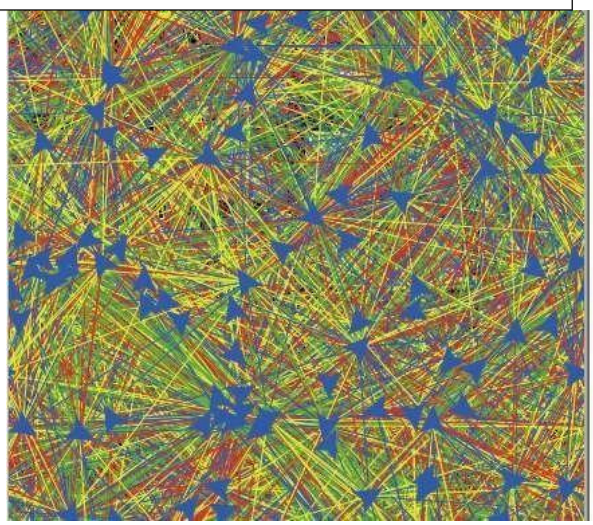


observer> ask patches [set pcolor pxcor + **3**p.yTcroyr c]onnecting turtles using create-links-with

instead

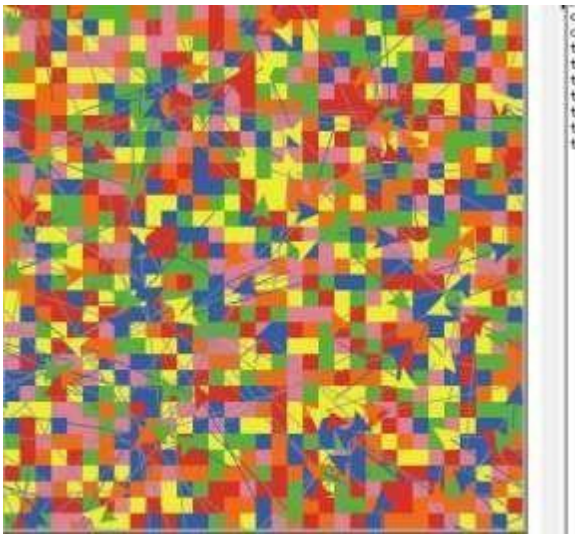
observer> ask patches [set pcolor (pxcor *pyacteo-rl)inmko-wd i1th0.]

ofcre



```
observer> crt 100
observer> ask turtles [ setxy random-xcor random-ycor ]
observer> ask turtles [set color blue]
observer> ask turtles [set size 2]
observer> ask turtles [ pen-down ]
observer> ask turtles [rt random 360 fd random 10]
observer> ask turtles [ create-links-with other turtles ]
observer> ask links [ set color one-of [ red green blue yellow ] ]
observer> clear-drawing
```

4. Apply different colors to patches and links.

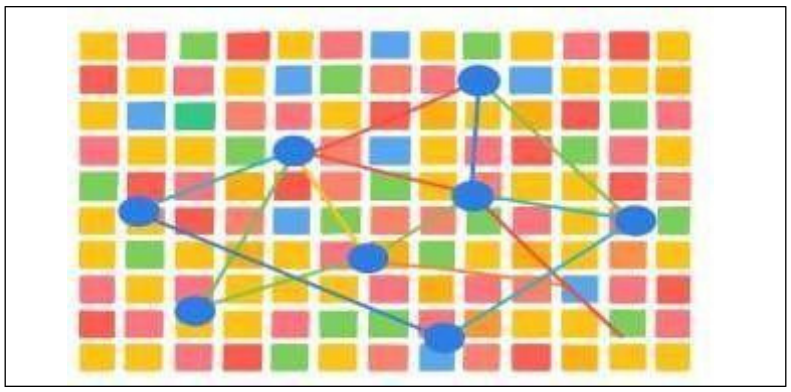


```
observer> ask patches [set pcolor one-of [ red green blue yellow orange pink ]]
observer> crt 100
turtles> setxy random-xcor random-ycor
turtles> set color blue
turtles> set size 2
turtles> pen-down
turtles> rt random 360 fd random 10
turtles> create-link-with one-of other turtles
turtles> set color one-of [ red green blue yellow orange ]
```

5. Sketch and describe the final visual pattern.

Final Visual Pattern:

The patches form a diagonal color gradient based on their coordinates. Turtles are randomly scattered, varying in size and color, appearing as bright dots on the background. Any links between turtles create simple networks, resulting in a colorful, semi-random pattern across the NetLogo world.



POST-LAB QUESTIONS ANSWER

1. What is the difference between a patch and a turtle?

Patch: Astationarysquarein theNetLogoworld;itforms the background grid and can have its own color and properties.

Turtle: A mobile agentthat can move around the world, change color, size, and interact with other turtles or patches.

2. What does facexy do?

It makes a turtle **face a specific coordinate** (x, y) in the world, so its heading points toward that location.

3. How are random-xcor and random-ycor useful?

They give **random coordinates** withintheworld boundaries, which is useful for **placing turtles randomly** on the patches.

4. What is the effect of create-link-with?

Itcreates a **link (line) between two turtles**,showing a connection or relationship.

This can be used to form networks or patterns.

5. How does clear-drawing differ from clear-all?

clear-drawing: Removes only the **lines and shapes drawn** by turtles; turtles and patches remain.

clear-all: Resets the **entire world**,removing turtles, links, drawings, and resetting patches to default.

LAB 5: CREATIVE PATTERN GENERATION USING TURTLES AND PATCHES

OBJECTIVE

- To explore how turtles and patches can work together to create colorful and dynamic patterns.
- To understand advanced turtle commands including linking, color gradients, and movement control.
- To observe how randomization can produce visually unique simulations.

LANGUAGE/TOOL

- **Tool:** NetLogo6.4.0
- **Platform:** Windows
- **Environment:** Command Center

THEORY

In NetLogo,the world is made up of two main types of agents:

- **Turtles:** movable agents that can draw, move, and interact with each other.
- **Patches:** fixed squares that make up the background and can change color or displaygradients.

By combining turtle movements,patch coloring, and linking commands, we can generate complex and attractive graphical patternsthat help us understand agent-based interactions.

IMPORTANT CONCEPTS

Concept	Description
Randomization	Addsvarietytopositions, colors, or directions.
Links	Connectturtlestoshow relationships or networks.
scale-color	Createscolorgradients on patches or agents.
Loops	Repeatactionsmultiple times to form shapes or movements.
Pen Control	Whendown,turtlesdraw lines as they move.

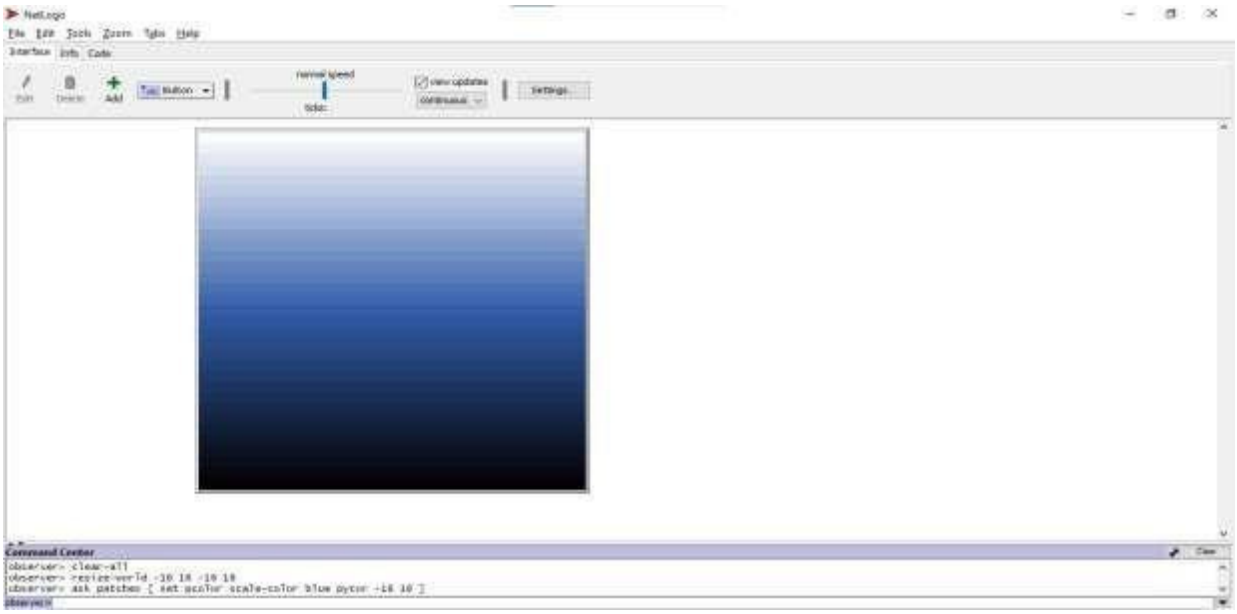
LAB PROCEDURE

Step 1: Open NetLogo

1. Launch **NetLogo**.
2. Gotothe **Command Center** atthebottom of the screen.
3. Makesureyouareinthe **observer** view. **Step 2: Prepare the World**

observer> clear-all observer> resize-world -16 16 -16 16

observer> ask patches [set pcolor scale-color blue pycor -1616]

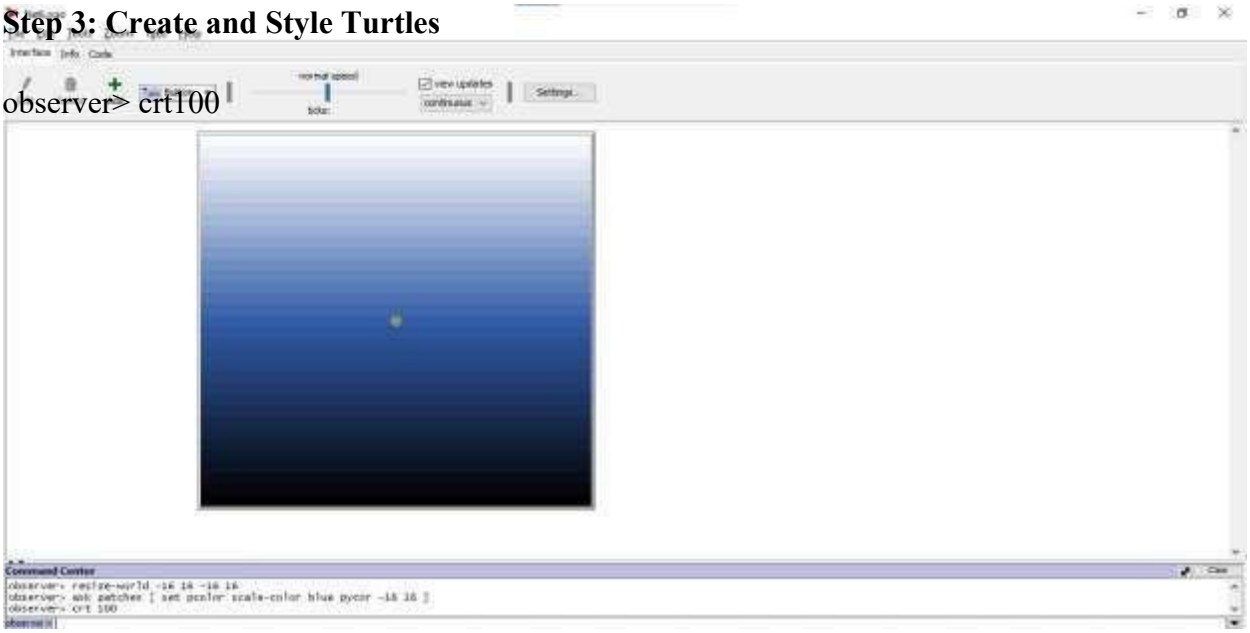


Explanation:

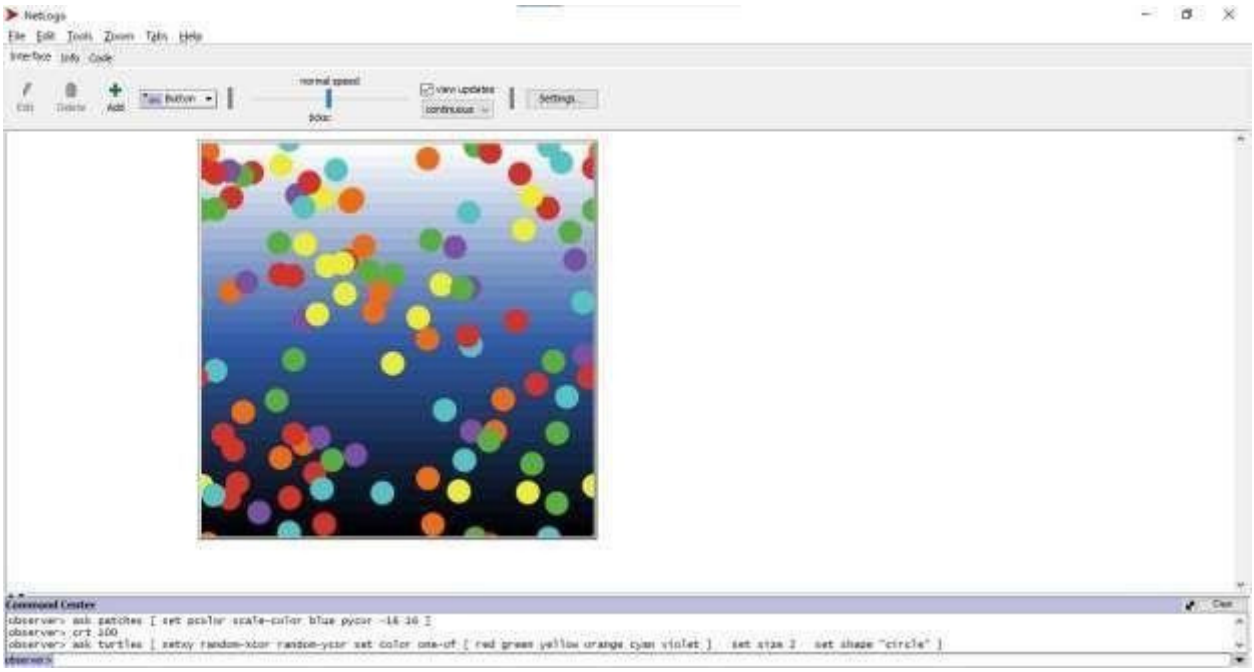
Clearsthe world, setsthevisiblegrid size, and creates a vertical blue gradient background.

Step 3: Create and Style Turtles

observer> crt100



observer> ask turtles [setxy random-xcor random-ycor set color one-of [red green yellow orange cyan violet] set size 2 set shape "circle"]



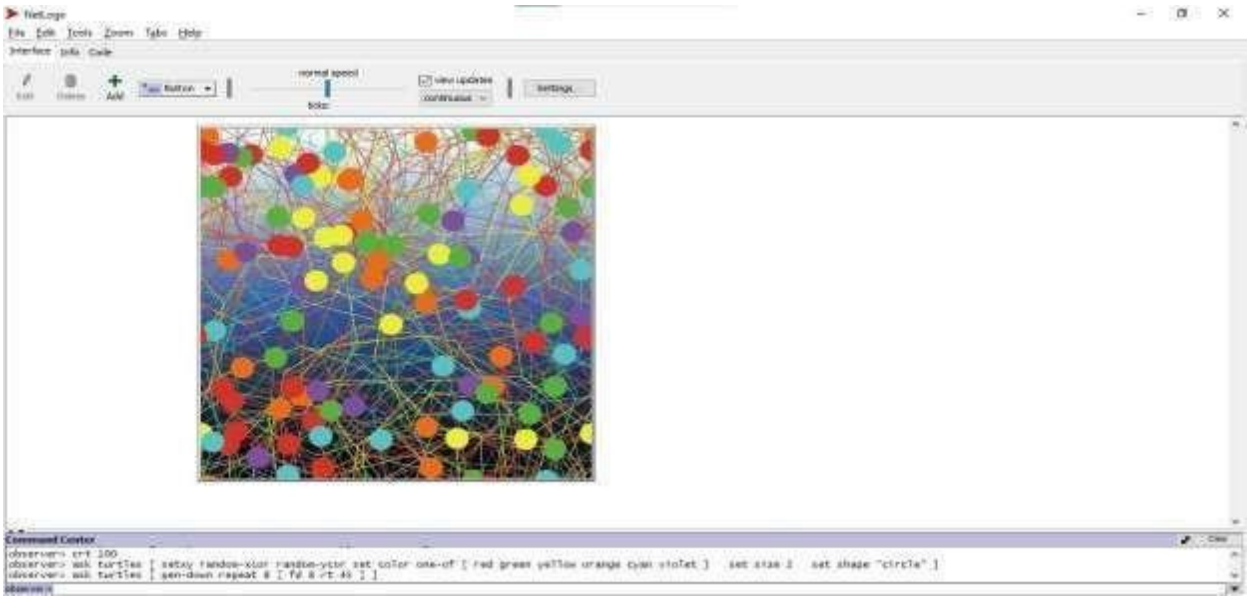
Explanation:

Creates100turtles at random positions with varied colors and circular shapes.

Step

4: Draw Star-Like Patternsobserver> ask turtles

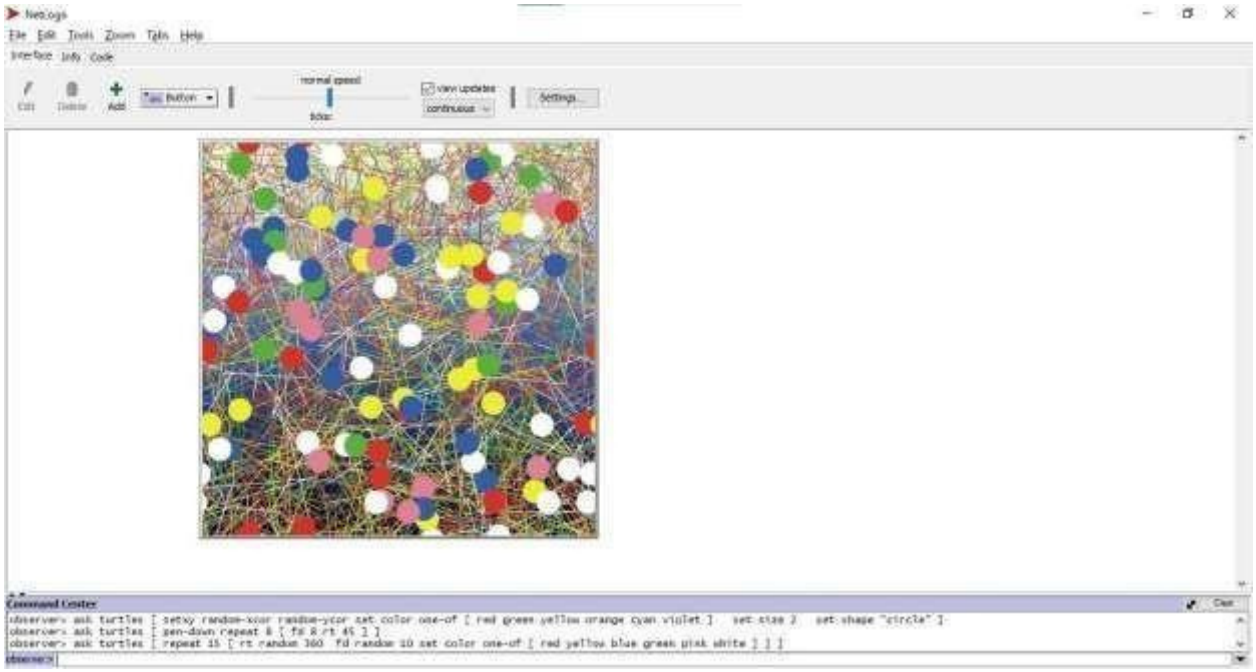
[pen-down repeat 8 [fd 8 rt 45]]



Each turtle draws an 8-pointed star by repeatedlymoving forward and turning right.

Step 5: Random Movement and Color Change

observer> askturtles [repeat15 [rtrandom360fdrandom 10 set color one-of [red yellow blue green pink white]]]

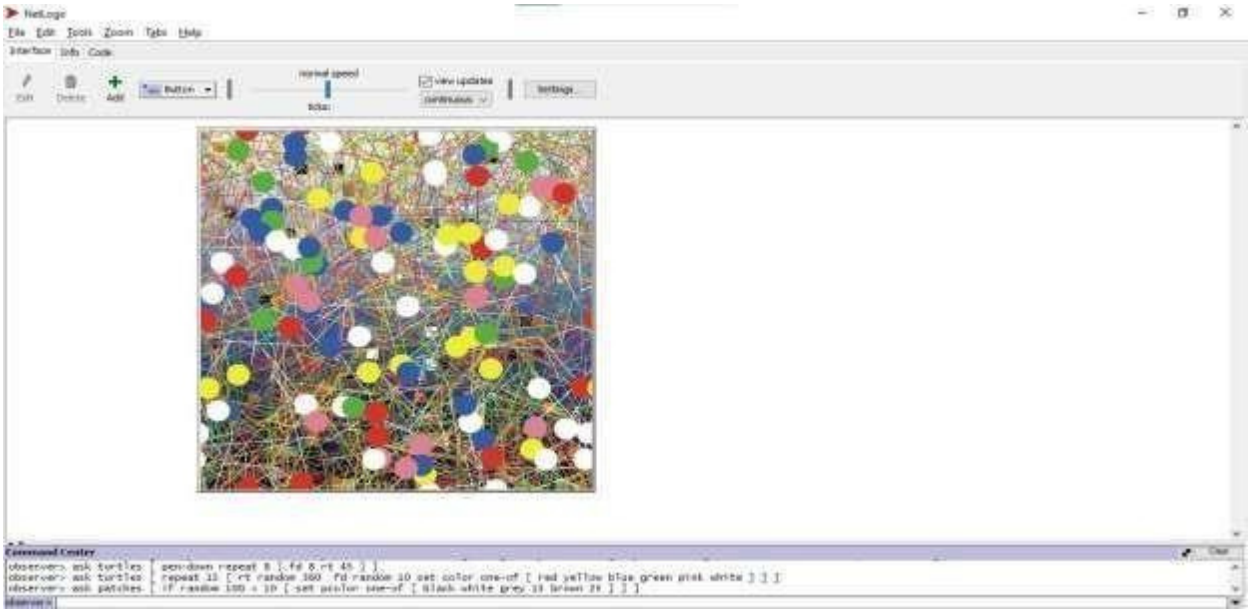


Step
Explanation:

Turtlesmoverandomly andchange colors, producing an animated colorful design.

6: Recolor Background Patches

observer>askpatches [ifrandom 100 < 10 [set pcolor one-of [black white grey 15 brown 25]]

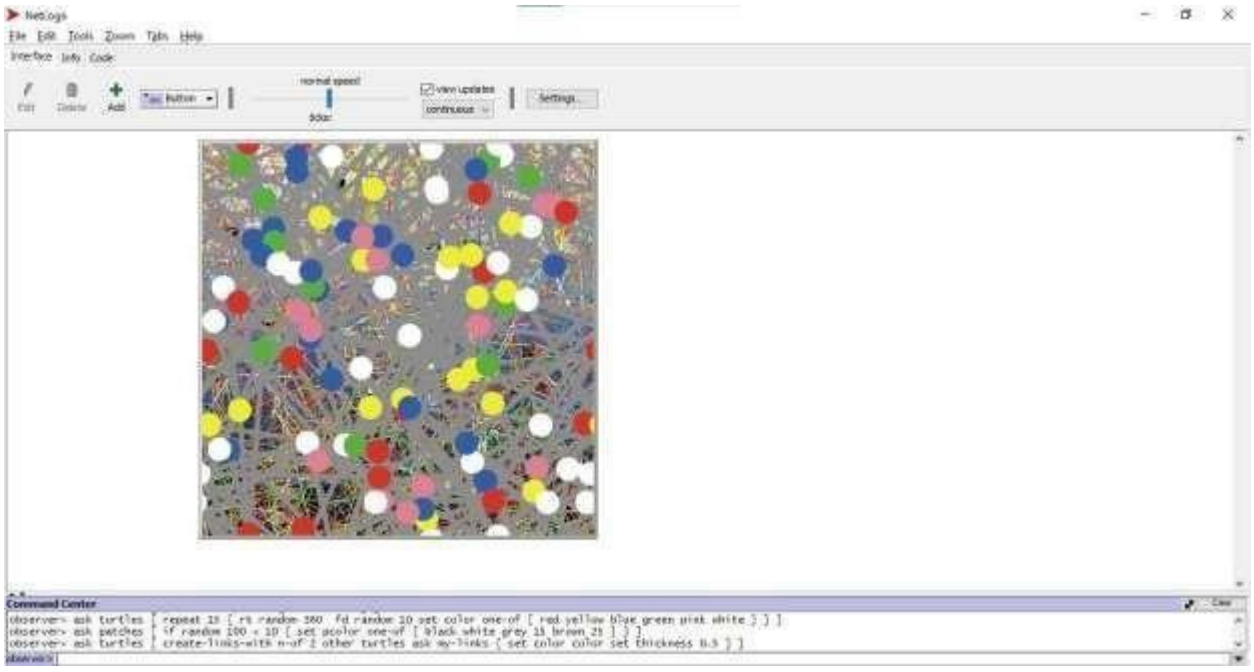


Explanation:

Some patcheschange color randomly to make the background more artistic.

Step 7: Connect Turtles with Linksobserver> ask turtles [create-links-with n-of 2 other turtles

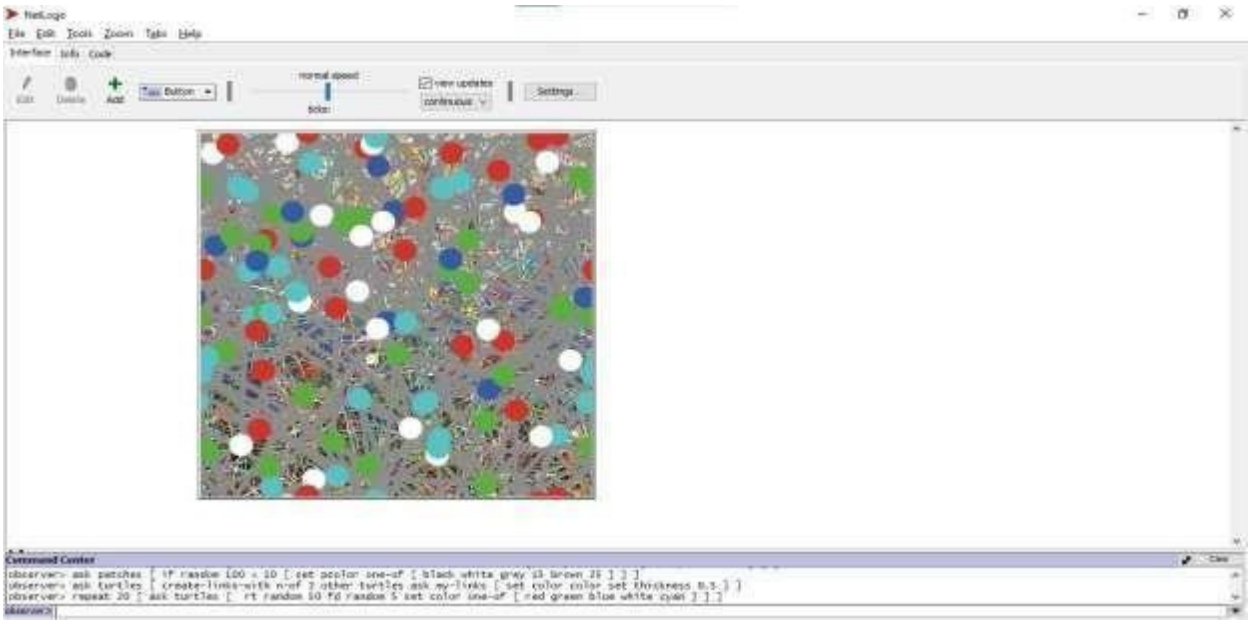
ask my-links [set color color set thickness 0.5]]



Each turtle connects with two others using colored lines, forming a network-like structure.

Step

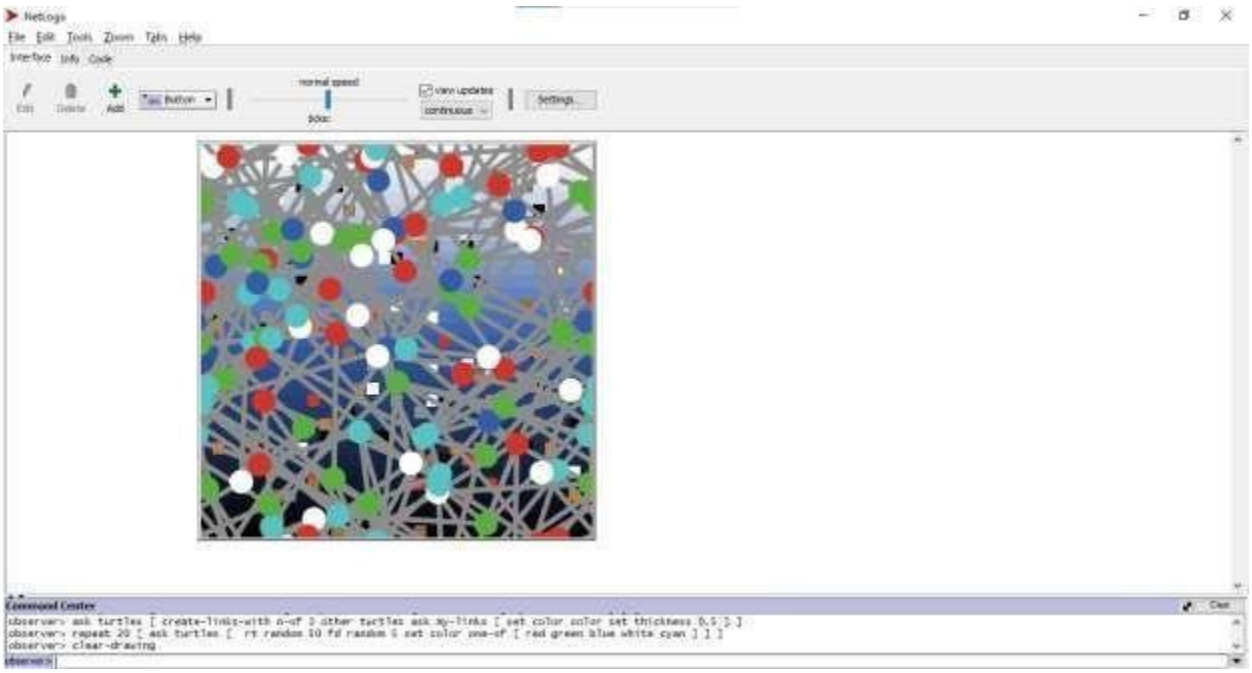
8: Animate the Turtles observer> repeat 20 [ask turtles [rt random 50 fd random 5 set color one-of[red green blue white cyan]]]



Explanation:

Turtlesmoveandchangecolorsdynamicallyfor20iterations, creating continuous motion.

Step 9: Clear Drawing Without Deleting Turtles observer> clear-drawing



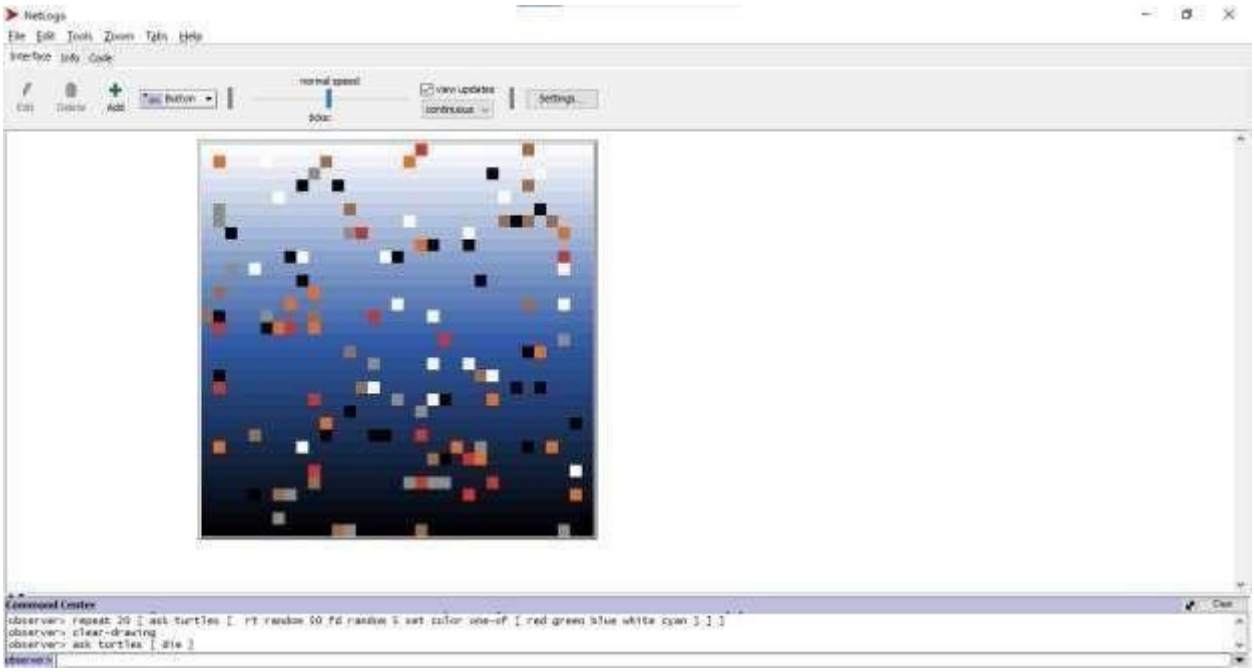
Explanation:

Erasesthetrailsdrawn by turtles but keeps them visible on the screen.

Step

10: Delete All Turtlesobserver> ask

turtles [die]



Removes all turtles from the simulation, leaving only the background pattern.

ADDITIONAL PRACTICE COMMANDS

Trythese extracommandsto explore morecreativity: observer> ask patches

[set pcolor scale-color red pxcor -1616] observer> ask turtles [set shape

"star" pen-down repeat 5 [fd 12 rt144]] observer> ask turtles [hatch 1 [

rt 180 fd4set color cyan]] observer> clear-drawing

EXPLANATION:

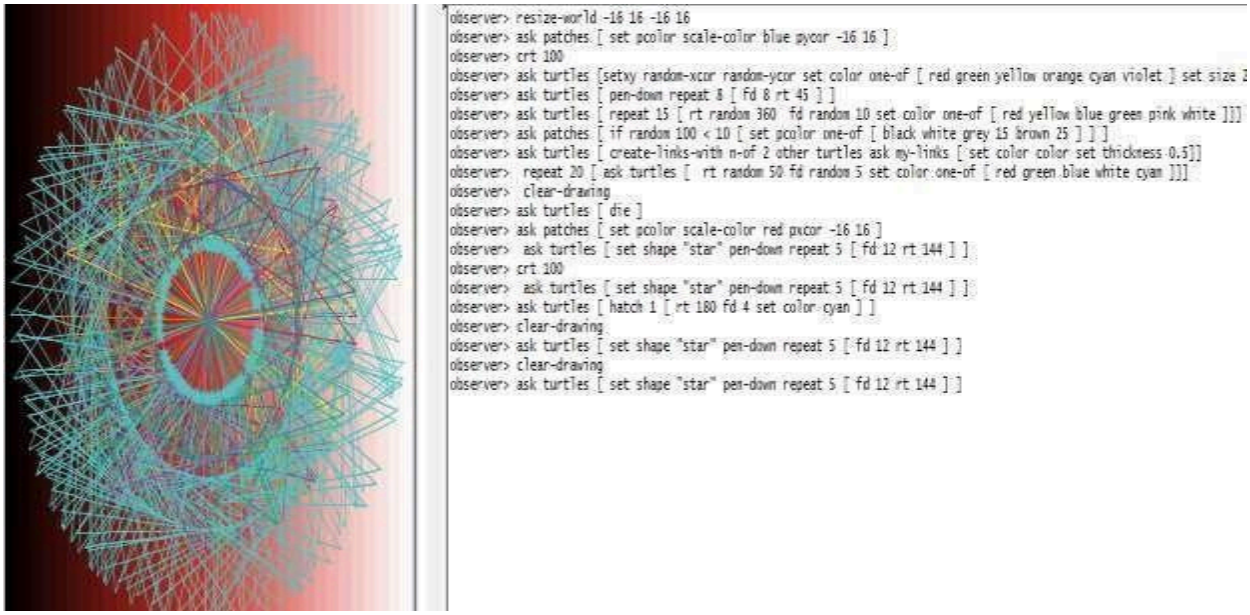
Thesecommandsintroduce new gradients, star shapes, cloned turtles, and clear drawings for further experimentation.

OBSERVATION

- Turtles canmove and draw simultaneously to form complex designs.
- The combination of colors and random movement produces unique results every run.
- The use of links adds structure and connection between agents.

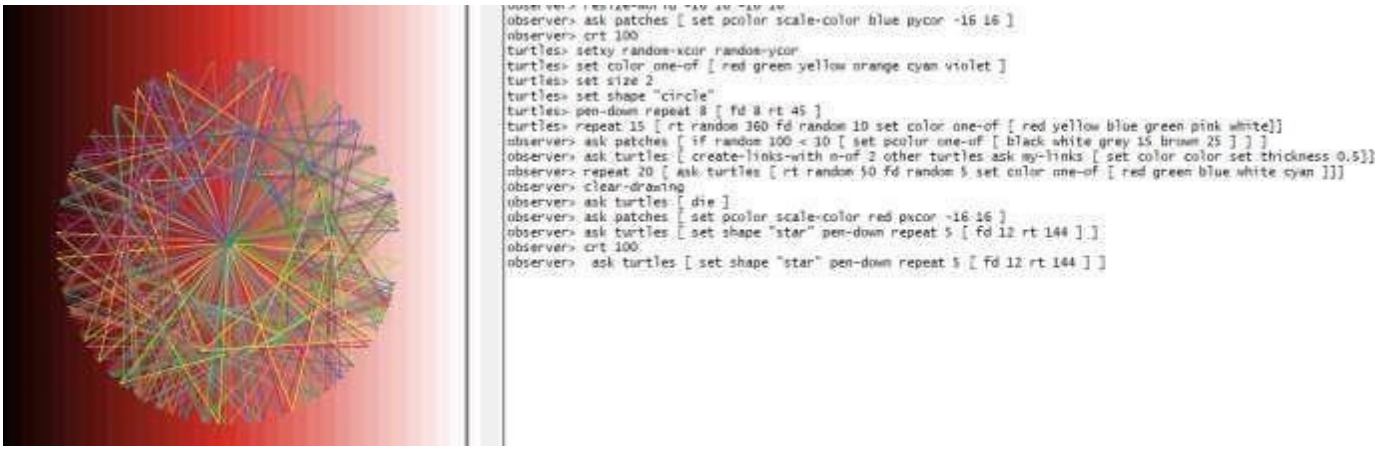
STUDENT TASK

1. Executeallcommands step by step in NetLogo.



```
observer> resize-world -16 16 -16 16
observer> ask patches [ set pcolor scale-color blue pycor -16 16 ]
observer> crt 100
observer> ask turtles [setxy random-xcor random-ycor set color one-of [ red green yellow orange cyan violet ] set size 2
observer> ask turtles [ pen-down repeat 8 [ fd 8 rt 45 ] ]
observer> ask turtles [ repeat 15 [ rt random 360  fd random 10 set color one-of [ red yellow blue green pink white ]]]
observer> ask patches [ if random 100 < 10 [ set pcolor one-of [ black white grey 15 brown 25 ] ] ]
observer> ask turtles [ create-links-with n-of 2 other turtles ask my-links [ set color color set thickness 0.5]]
observer>  repeat 20 [ ask turtles [  rt random 50 fd random 5 set color one-of [ red green blue white cyan ]]]
observer>  clear-drawing
observer> ask turtles [ die ]
observer> ask patches [ set pcolor scale-color red pxcor -16 16 ]
observer>  ask turtles [ set shape "star" pen-down repeat 5 [ fd 12 rt 144 ] ]
observer> crt 100
observer>  ask turtles [ set shape "star" pen-down repeat 5 [ fd 12 rt 144 ] ]
observer> ask turtles [ hatch 1 [ rt 180 fd 4 set color cyan ] ]
observer> clear-drawing
observer> ask turtles [ set shape "star" pen-down repeat 5 [ fd 12 rt 144 ] ]
observer> clear-drawing
observer> ask turtles [ set shape "star" pen-down repeat 5 [ fd 12 rt 144 ] ]
```

2. Modify the number of turtles and observe how the design changes.

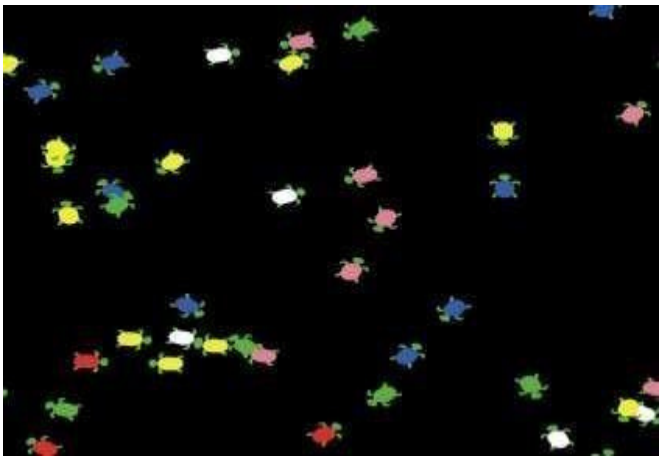


```
observer> ask patches [ set pcolor scale-color blue pycor -16 16 ]
observer> crt 100
turtles> setxy random-xcor random-ycor
turtles> set color one-of [ red green yellow orange cyan violet ]
turtles> set size 2
turtles> set shape "circle"
turtles> pen-down repeat 8 [ fd 8 rt 45 ]
turtles> repeat 15 [ rt random 360 fd random 10 set color one-of [ red yellow blue green pink white]]
observer> ask patches [ if random 100 < 10 [ set pcolor one-of [ black white grey 15 brown 25 ] ] ]
observer> ask turtles [ create-links-with n-of 2 other turtles ask my-links [ set color color set thickness 0.5]]
observer> repeat 20 [ ask turtles [ rt random 50 fd random 5 set color one-of [ red green blue white cyan ]]]
observer> clear-drawing
observer> ask turtles [ die ]
observer> ask patches [ set pcolor scale-color red pxcor -16 16 ]
observer> ask turtles [ set shape "star" pen-down repeat 5 [ fd 12 rt 144 ] ]
observer> crt 100
observer>  ask turtles [ set shape "star" pen-down repeat 5 [ fd 12 rt 144 ] ]
```

More turtles → denser, complex, overlapping design
Fewer turtles → sparse, simple, clear design

3. Createyourownpattern using loops and random colors.

4. Test the difference between clear-all and clear-drawing.**clear-drawing:**
clear all lines (links)



Clear-all: delete everything

5. Recordyourobservations and screenshots of yourpatterns.

OBSERVATION

In this lab, I learned how turtles and patches create different patterns using loops, colors, and movement. Changing the number of turtles affected the design's density, and I observed how clear-drawing and clear-all reset the world differently. Overall, it showed how simple commands can form creative patterns in NetLogo.

POST-LAB QUESTIONS ANSWERS

1. What is the function of the scale-color command?

It sets a color based on a numeric value, creating a gradient effect between two extremes (e.g., using coordinates to make smooth color transitions on patches).

2. How does randomization affect the pattern's appearance?

Randomization makes each pattern unique and unpredictableby changing turtle positions, directions, and colors every time the code runs.

3. What happens if you change the value of n-of in the linking command?

Changing the value of n-of increases or decreases how many other turtles eachturtle links with, affecting how dense or connected the network appears.

4. Which command removes turtles completely from the world?

The command ask turtles [die] removes all turtles from the world.

5. How can you make the turtles draw different shapes?

By using loops (repeat) with movement and turning commands (like fd, rt, lt) and adjusting angles or steps, turtles can draw shapes such as stars, squares, or circles.

LAB 6: TURTLE MAZE AND OBSTACLE NAVIGATION IN NETLOGO

OBJECTIVE

- Tolearn howturtles can navigate around obstacles using simple conditionallogic.
- Tounderstandpen control, loops, and patch sensing for creating interactivepaths.
- Toexploredynamic turtle responses to patch colors or boundaries.

LANGUAGE/TOOL

- **Tool:** NetLogo6.4.0
- **Platform:** Windows

- **Environment:** CommandCenter

THEORY

Turtles cansense the color ofthe patchthey are on or nearby patches. By using if statements, turtles canavoid obstacles orchangedirection. This lab demonstrates how simple rules and conditionsallow turtlesto navigateenvironments and interact with patch-based obstacles.

IMPORTANT CONCEPTS

Concept	Description
Pen Control	pen-downorpen-up determines whether movement leaves trails.
Patch Awareness	Turtlescanread the color of the patch they are on.
Conditional Logic	ifstatementshelp turtles respond to obstacles.
Loops	repeatallowsturtles to perform actions multiple times.
Navigation	Combiningmotion, sensing, and conditional decisions.

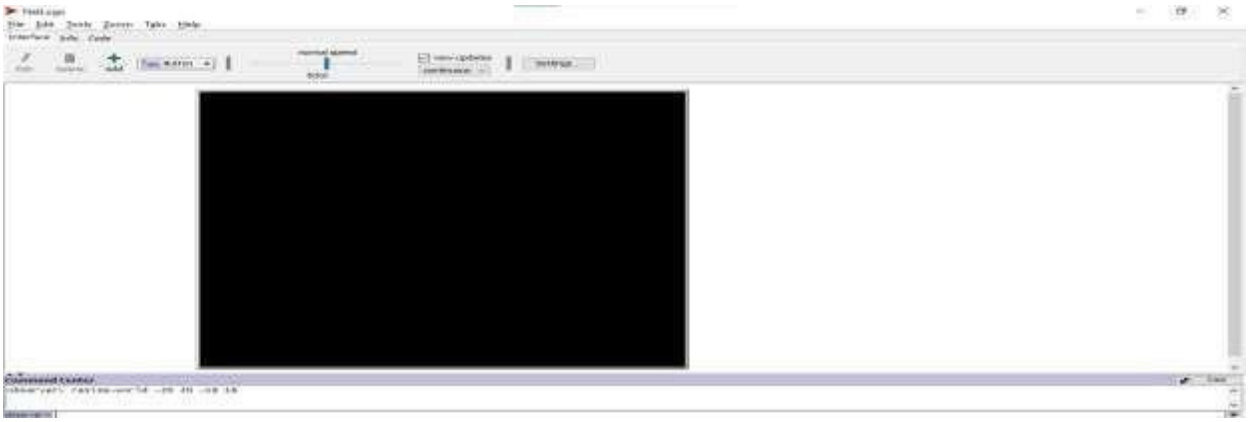
LAB PROCEDURE

Step 1: Open NetLogo

1. StartNetLogo.
2. Openthe Command Center window.
3. Makesure you are in observer mode.

Step 2: Prepare the World observer>

resize-world -2020 -20 20



observer> ask patches [set pcolor one-of [green black]]

Explanation:

Clearsthe world, defines boundaries, and randomly colors patches green (open space) or black (obstacles).

Step 3: Create Turtles

observer>crt20 [setcolor blue set size 2 setxy random-xcor random-ycor set heading random 360]

Explanation:

Creates20turtlesatrandom positions with random headings.

Step 4: Enable Drawing observer> ask turtles [pen-down]

Explanation:

Turtlesleavetrails when moving, showing their path through the maze.

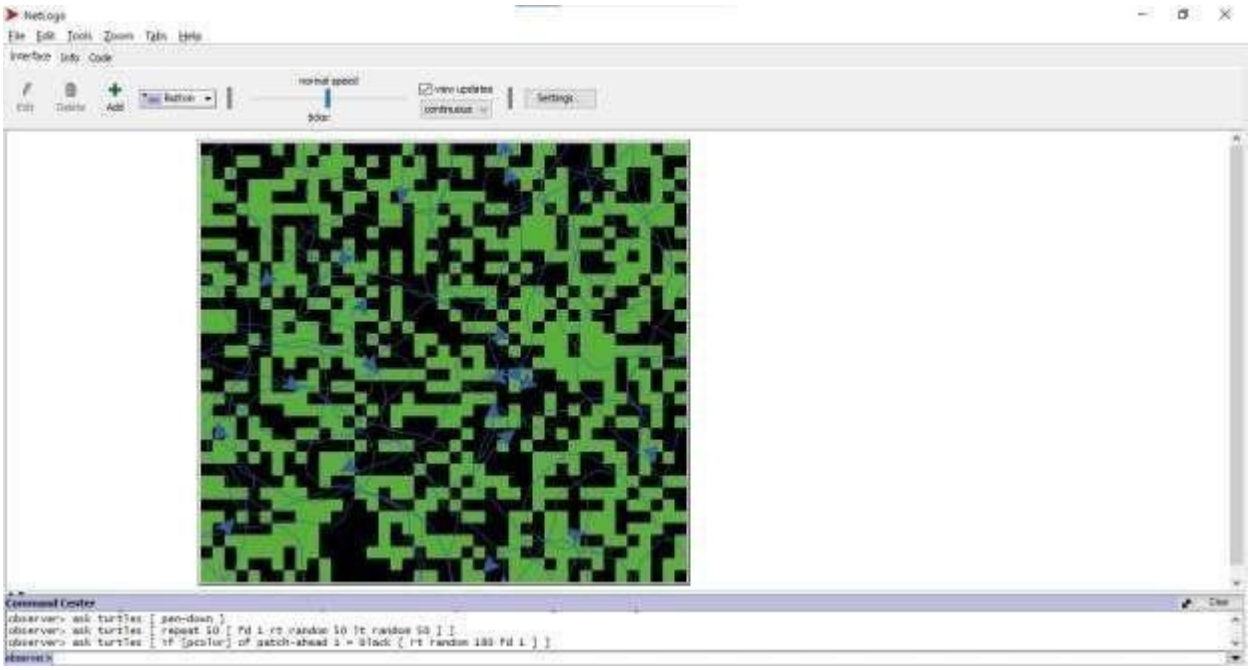
Step5: Basic Random Motion observer> ask turtles [repeat 50 [fd 1 rt random 50 lt random 50]]

Explanation:

Turtlesmoveforward with slight random turns to explore the environment.

Step 6: Obstacle Avoidance

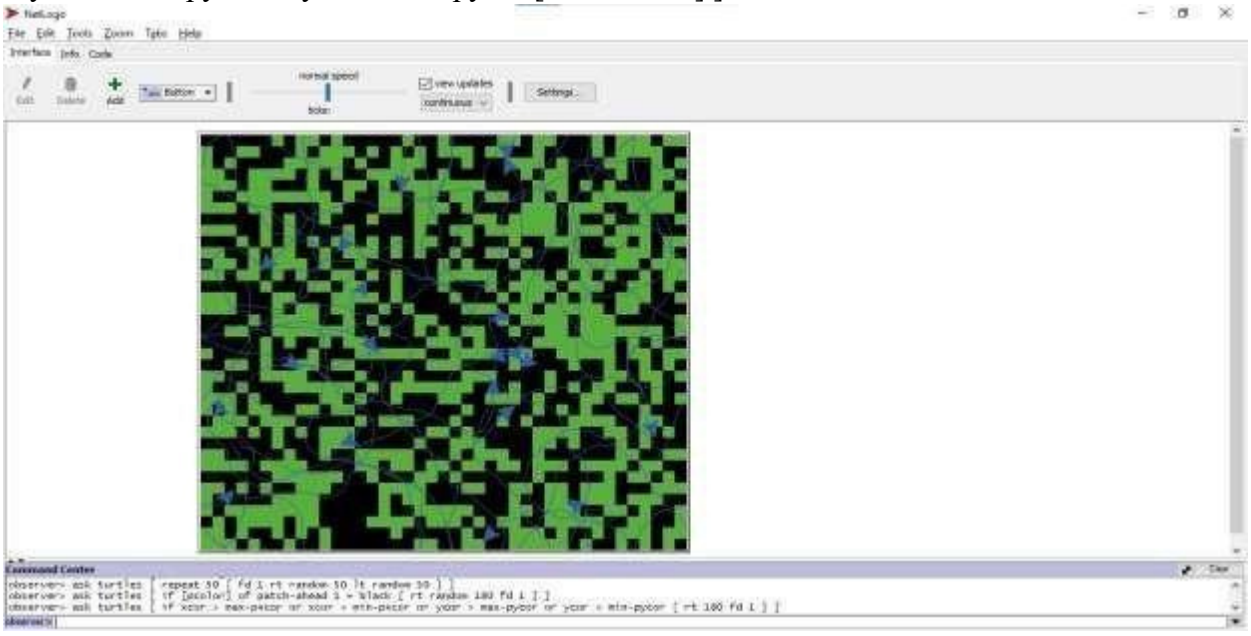
observer>askturtles [if[pcolor] of patch-ahead 1 = black [rt random 180 fd 1]]



Explanation:

Ifaturtleseesablack patch ahead (obstacle), it turns randomly to avoid it.

Step 7: Boundary Control observer> ask turtles [if xcor > max-pxcor or xcor < min-pxcor or ycor > max-pycor or ycor < min-pycor [rt 180 fd 1]]



Explanation:

Turtlesdetectworld boundariesand turn back to stay inside the environment. **Step 8: Color Change on Encounter**

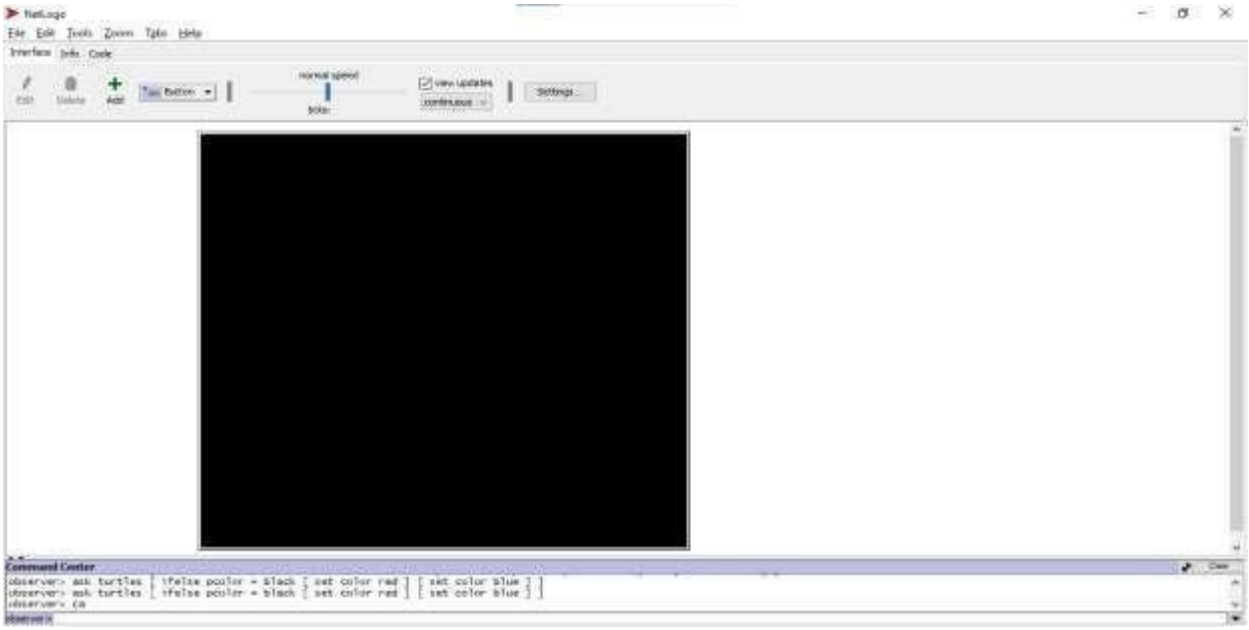
observer>askturtles [ifelsepcolor = black [set color red] [set color blue]]

Explanation:

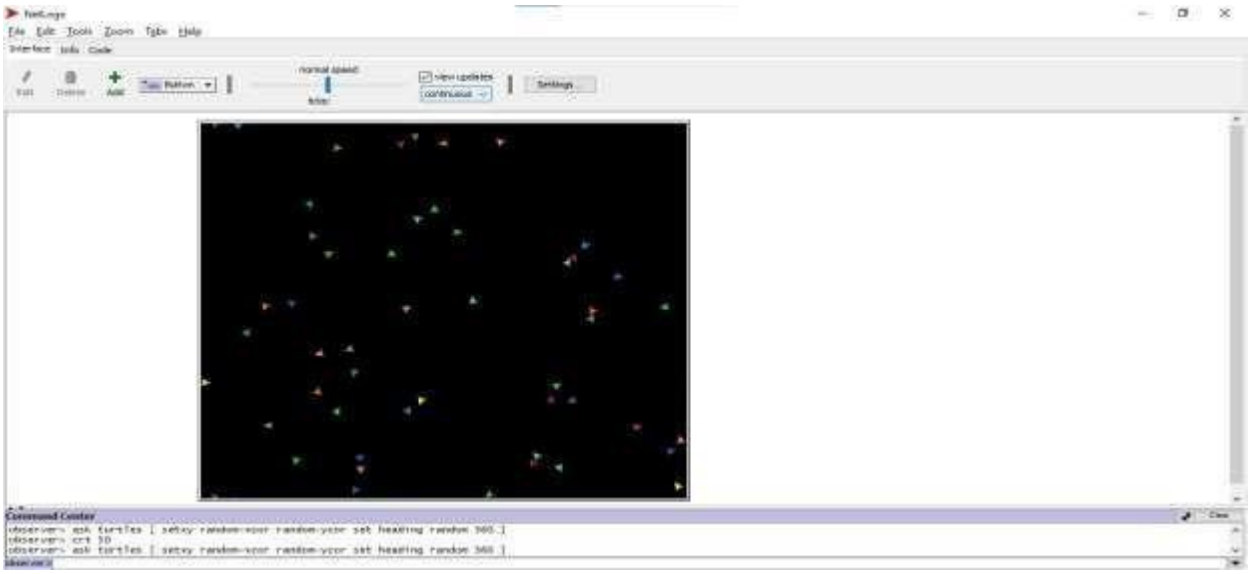
Turtlesturningred when on obstacles and blue on open paths provides visual feedback.

Step 9: ClearTrails and Reset

observer> ca

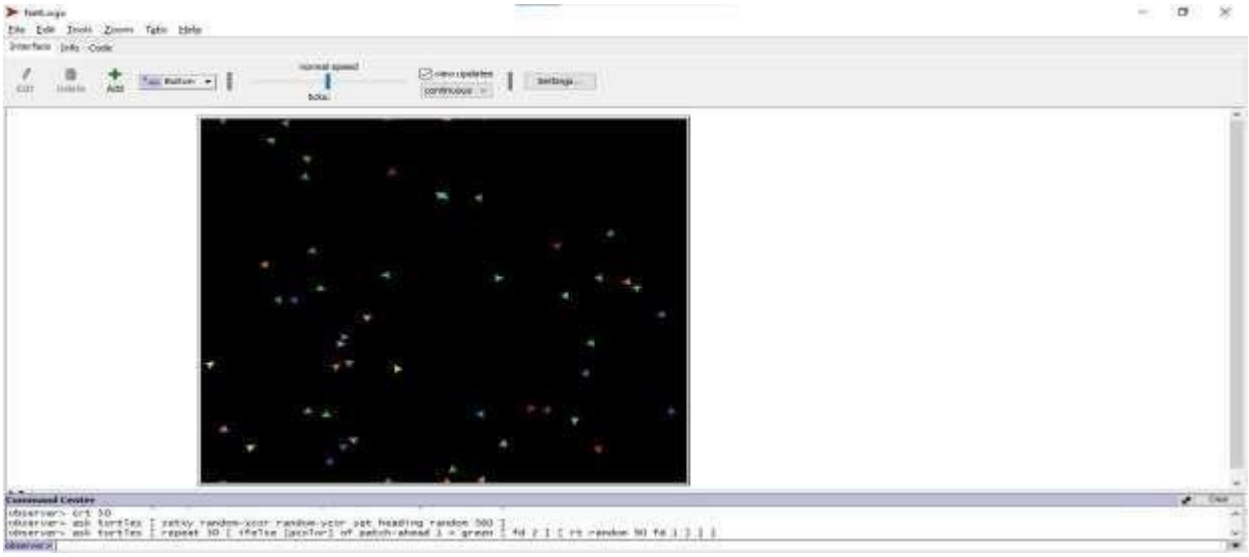


observer> ask turtles [setxy random-xcor random-ycor set heading random 360]



Explanation:
Clearstrailsandredistributes turtles for a new simulation run.

Step 10: Advanced Movement
observer> askturtles [repeat30[if [pcolor] of patch-ahead 1 = green [fd 2] else [rt random 90 fd 1]]]



Explanation:
Turtlesmovefaster through open space andcarefullynavigate around obstacles.

ADDITIONAL PRACTICE COMMANDS
observer> ask turtles [if xcor mod 2 = 0 [setsize3]else [set size 1.5]] observer> ask turtles [if color = red [rt 90fd1]]observer> ask patches [if pycor mod 3 = 0 [set pcolor grey]]observer> clear-drawing
EXPLANATION:

Students can experimentwith turtle size, turning rules, and patterned obstacles.

OBSERVATION

- Turtlesavoid black patches and navigate green areas.
- Trailsshow how turtles move around obstacles.
- Randomturns make navigation more natural.
- Adjustingturtle count or obstacle density changes navigation complexity.

STUDENT TASK

1. Execute all commands sequentially.

```
observer> resize-world -20 20 -20 20
observer> ask patches [ set pcolor one-of [ green black ] ]
observer>  crt 20 [ set color blue set size 2 setxy random-xcor random-ycor set heading random 360]
observer>  ask turtles [ pen-down ]
observer>  ask turtles [ repeat 50 [ fd 1 rt random 50 lt random 50 ] ]
observer> ask turtles [ if [pcolor] of patch-ahead 1 = black [ rt random 180 fd 1 ] ]
observer> ask turtles [ if xcor > max-pxcor or xcor < min-pxcor or ycor > max-pycor or ycor < min-pycor [ rt 180 fd 1 ] ]
observer> ask turtles [ ifelse pcolor = black [ set color red ] [ set color blue ] ]
observer> ca
observer> crt 50
observer>  ask turtles [ setxy random-xcor random-ycor set heading random 360 ]
observer> ask turtles [ if xcor mod 2 = 0 [ set size 3 ] ]
observer>  ask turtles [ if color = red [ rt 90 fd 1 ] ]
observer> ask patches [ if pycor mod 3 = 0 [ set pcolor grey ] ]
observer>  clear-drawing
```

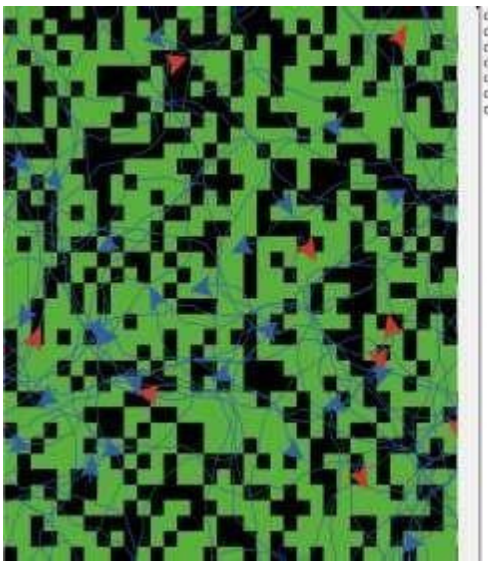
2. Increase turtle count to 30 or 40 and observe changes.

```
observer> resize-world -20 20 -20 20
observer> ask patches [ set pcolor one-of [ green black ] ]
observer>  crt 20 [ set color blue set size 2 setxy random-xcor random-ycor set heading random 360]
observer>  ask turtles [ pen-down ]
observer>  ask turtles [ repeat 50 [ fd 1 rt random 50 lt random 50 ] ]
observer> ask turtles [ if [pcolor] of patch-ahead 1 = black [ rt random 180 fd 1 ] ]
observer> ask turtles [ if xcor > max-pxcor or xcor < min-pxcor or ycor > max-pycor or ycor < min-p
observer> ask turtles [ ifelse pcolor = black [ set color red ] [ set color blue ] ]
observer> ca
observer> crt 50
observer>  ask turtles [ setxy random-xcor random-ycor set heading random 360 ]
observer> ask turtles [ if xcor mod 2 = 0 [ set size 3 ] ]
observer>  ask turtles [ if color = red [ rt 90 fd 1 ] ]
observer> ask patches [ if pycor mod 3 = 0 [ set pcolor grey ] ]
observer>  clear-drawing
observer> crt 50 [ set color blue set size 2 setxy random-xcor random-ycor set heading random 360]
```

OBSERVATION

Increasing the turtlecount made the pattern denser, with more overlaps and movements. More turtles changed colordue to black patches, and the world appeared more crowded and colorful.

3. Modify obstacle density (pcolor black frequency) and see how turtles respond.

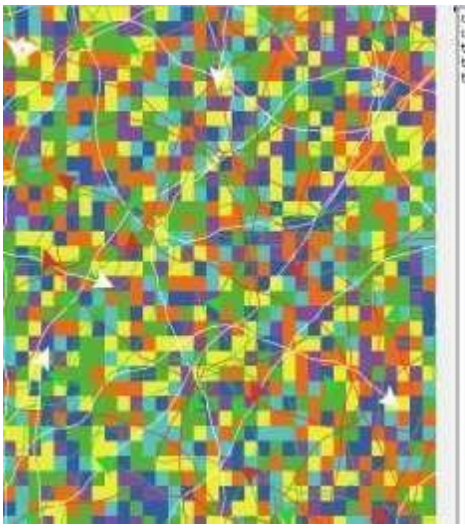


```
observer> ask patches [ifelse random 100 < 40 [ set pcolor black ] [ set pcolor green ]]
observer> crt 40 [ set color blue set size 2 setxy random-xcor random-ycor set heading random 360 ]
observer> ask turtles [ pen-down ]
observer> ask turtles [ repeat 50 [ fd 1 rt random 50 lt random 50 ] ]
observer> ask turtles [ if [pcolor] of patch-ahead 1 = black [ rt random 180 fd 1 ] ]
observer> ask turtles [ if xcor > max-pxcor or xcor < min-pxcor or ycor > max-pycor or ycor < min-pycor [ rt 180 fd 1 ] ]
observer> ask turtles [ ifelse pcolor = black [ set color red ] [ set color blue ] ]
```

OBSERVATION

When black patches(obstacles) increased, turtles had less space to move, turned more often, and their paths became irregular. With fewer black patches, their motion was smoother and more spread out.

4. Test different random turn ranges for smoother navigation.

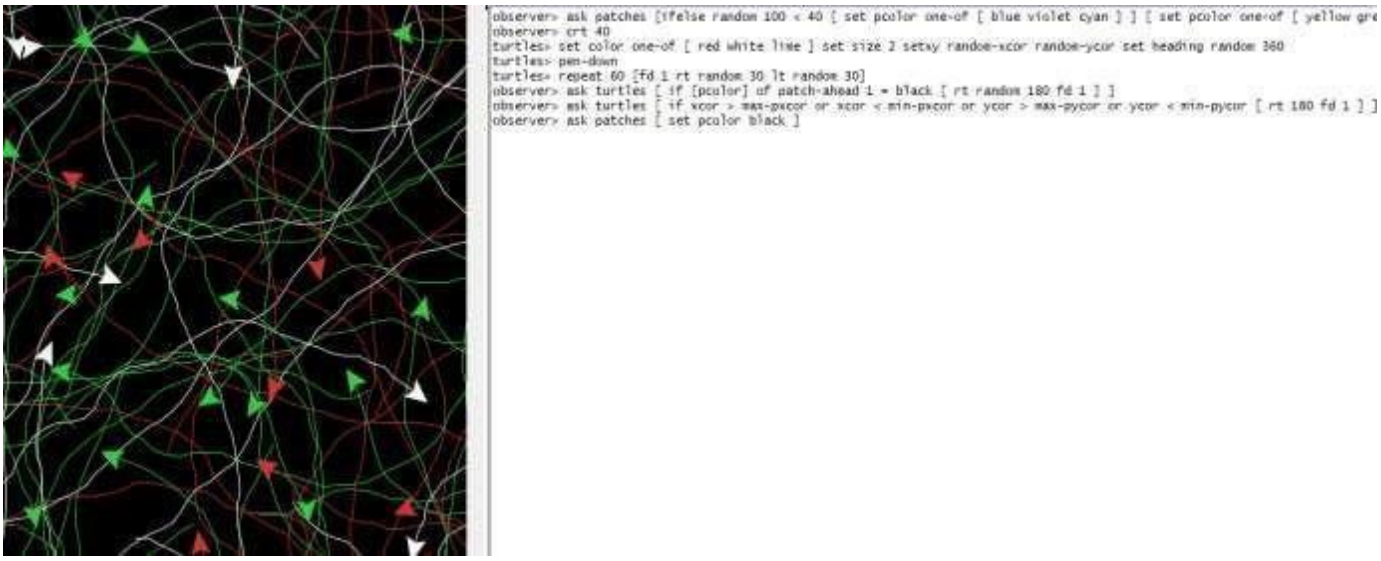


```
observer> ask patches [ifelse random 100 < 40 [ set pcolor one-of [ blue violet cyan ] ] [ set pcolor one-of [ yellow green ora
observer> crt 40
turtles> set color one-of [ red white lime ] set size 2 setxy random-xcor random-ycor set heading random 360
turtles> pen-down
turtles> repeat 60 [fd 1 rt random 30 lt random 30]
```

OBSERVATION

Smaller random turnvalues made turtles move smoothly, while larger ones caused quick, random turns and irregularpaths.

5. Record how turtlesadapt to obstacles and boundaries.



OBSERVATION

Turtlesavoidedblackpatches by turning away and bounced back when they reached the world boundaries, showingthey can adapt their movement to stay within the area.

POST-LAB QUESTIONS ANSWERS

1. How do turtles detect obstacles?

Turtles detect obstacles using thecommand **[pcolor] of patch-ahead 1**,which checks the color of the patch directly in frontof them.

2. Why do we use patch-ahead instead of patch-here?

Because **patch-ahead** letsturtles sensewhat's *in front* of them, while **patch-here** only checks the patch they are *currentlyon* .

3. How does changing random turn angles affect navigation?

Smaller angles make movement smooth and curved, while larger angles cause sharp, random, or chaotic paths.

4. What happens when obstacle density increases?

Turtles face more collisions, turn more often, andtheir paths become tighter and less smooth.

5. How can turtle trails help understand pathfinding behavior?

Turtle trails show how turtles move around obstacles, revealinghow effectively they navigate or adapt within the environment.

Lab 7: Heroes and Cowards Model Setup and Agent Properties

Objective

- To create a population of agents (people) with unique characteristics.
- To introduce personality differences using a chooser (Hero vs Coward).
- To assign friends, enemies, and personal attributes to each agent.
- To prepare the environment that will be used for behavioral simulation in Lab 9.

Language/Tool

- **Tool:** NetLogo6.4.0
- **Platform:** Windows
- **Environment:** Interface + Command Center

Theory

TheHeroes & Cowards model is an *agent-based social simulation* where people behave differently based on personality:

- **Heroes** move toward conflict.
- **Cowards** avoid danger.

Before coding behaviors, we must prepare:

1. A population of agents.
2. Attributes such as age, personality, friends, and enemies.
3. A chooser that lets the user decide the percentage of heroes vs cowards.
4. Spatial setup for agents to appear in random positions.

Thislabfocuseson setting the foundation of the model. **Important Concepts**

Concept	Description
Breeds	Differentcategories of agents.
Choosers	Interfacewidget to select personality group.
Agent Variables	Propertieslike age, type, friend-list, enemy-list.
Setup Procedure	Runsonceto initialize the world.

Lab Procedure

Step 1: Define Agent Variables

In Code tab:

```
breed [ persons person ]
persons-own [ age
personality friends
enemies
]
```

Explanation:

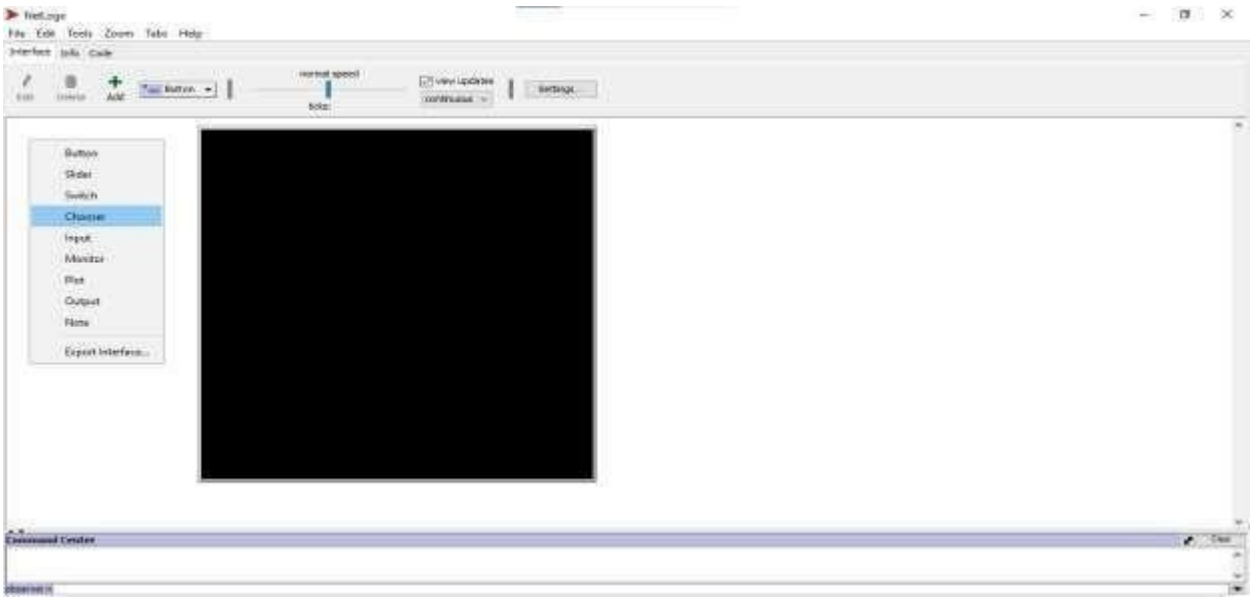
Eachpersonwill store an age, a personality label (hero or coward), and two lists of connections.

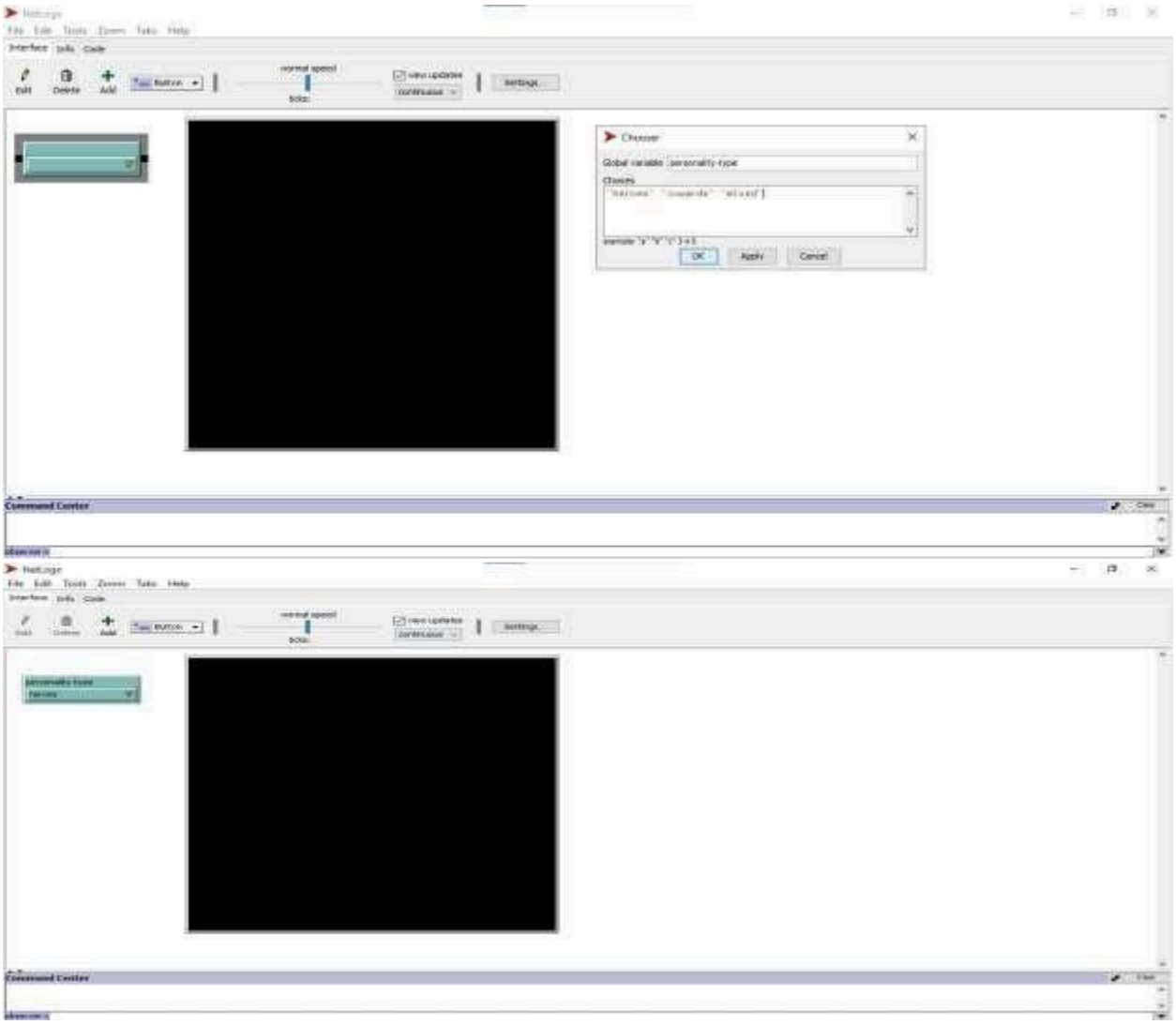
Step2:AddaChooser and Button in Interface

In the Interface tab:

Add chooser

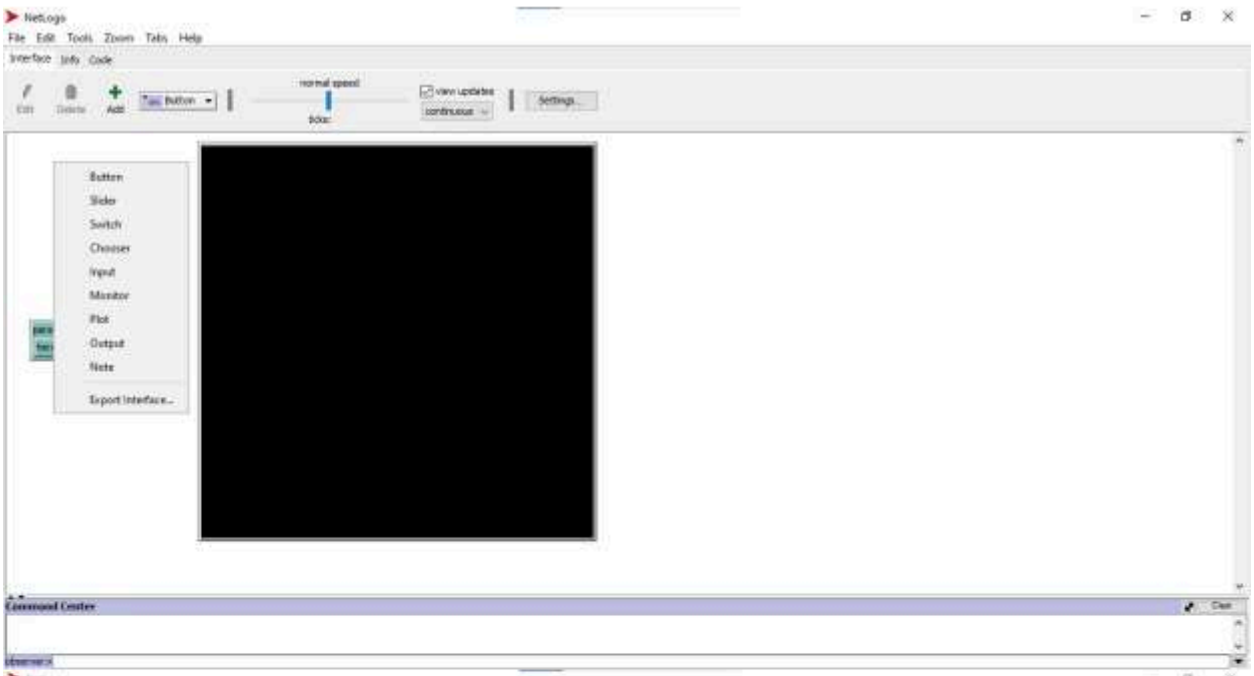
- Name: personality-type
- Choices: "heroes" "cowards"
  "mixed"

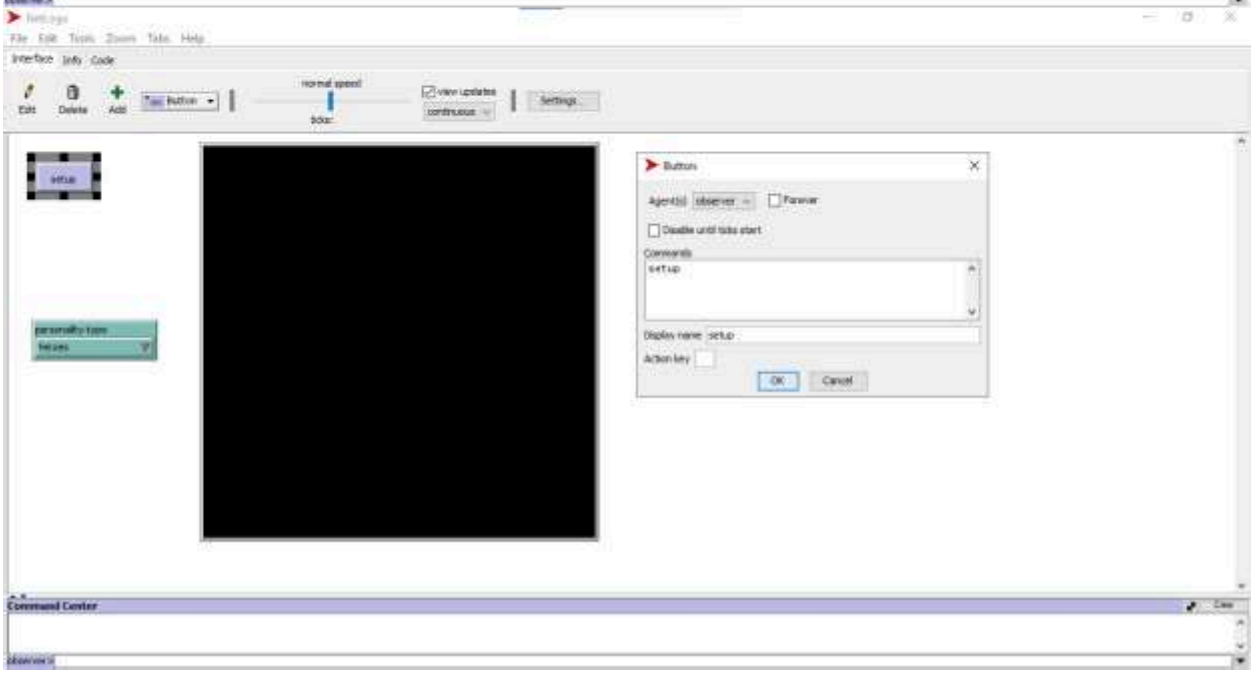




Add Button

- **Button Name:** setup
  - **Command:** `setup`
- **Forever:** unchecked





Step 3: Write the Setup Procedure

```
setup clear-all create-persons 100 [
setxy random-xcor random-ycor set
size 2 set age random 50 assign-
personality
]
```

assign-friends-and-enemies reset-

ticks end

Explanation:

Creates 100 people with random age and position, then assigns personality and social relationships.

Step 4: Assign Personality Based on Chooser to

assign-personality if personality-type = "heroes"

[set personality "hero" set color blue

]

if personality-type = "cowards" [set

personality "coward" set

color green

]

if personality-type = "mixed" [ifelse

random 2 = 0

[set personality "hero" set color blue]

[set personality "coward" set color green]

]

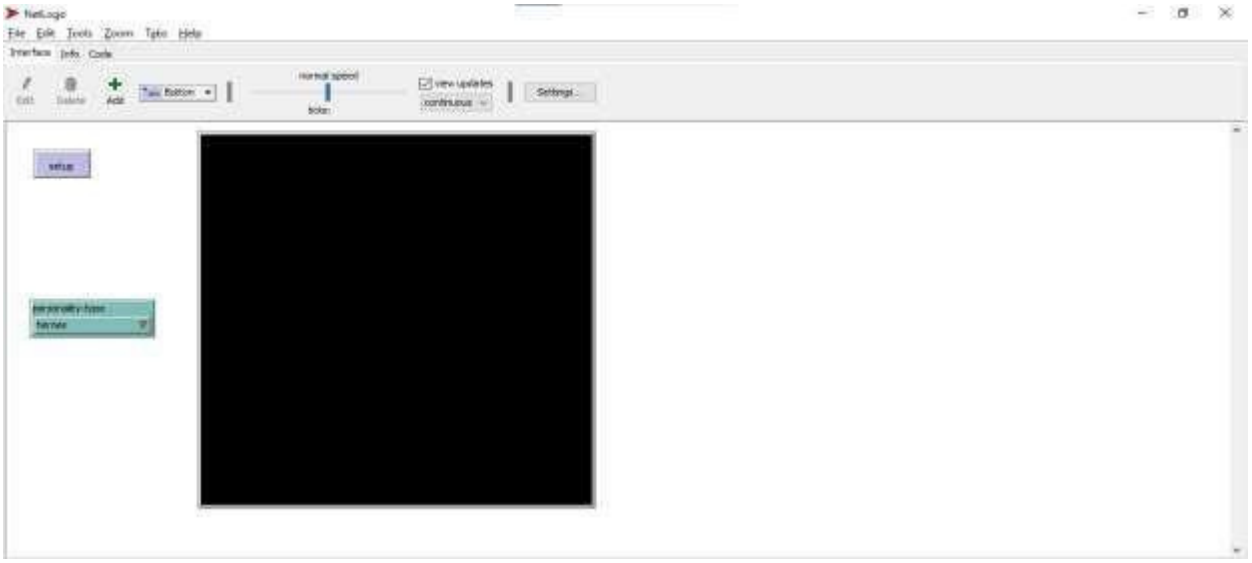
End

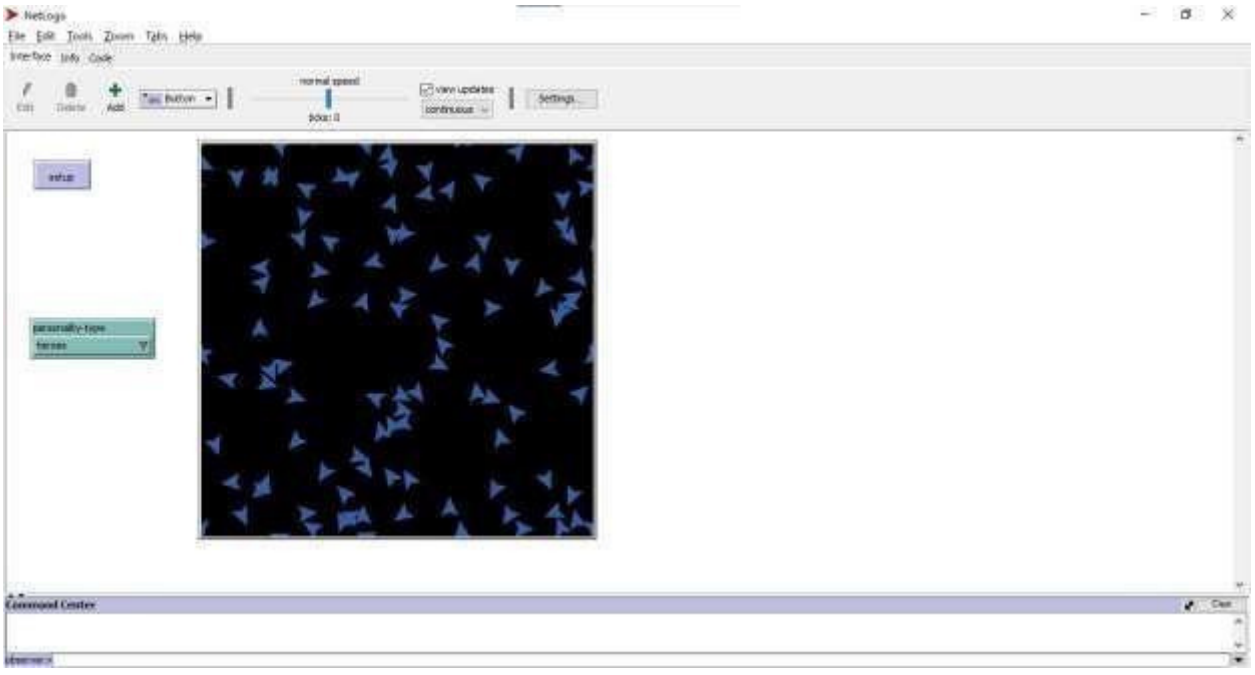
Explanation:

- Heroes turn blue
- Cowards turn green
- Mixed society randomly assigns roles **Step 5: Assign Friends and Enemies** to assign-friends-and-enemies ask persons [let others other persons set friends n-of 3 others set enemies n-of 3 others

]
end


NetLogo
File Edit Tools Zoom Tabs Help
Interface Info Code
Find... Check Procedures Indent automatically Code Tab in separate window
if personality-type = "mixed" [
ifelse random 2 = 0
[set personality "hero" set color blue]
[set personality "coward" set color green]
]
end
to assign-friends-and-enemies
ask persons [
let others other persons
set friends n-of 3 others
set enemies n-of 3 others
]
end


NetLogo
File Edit Tools Zoom Tabs Help
Interface Info Code
Edit Delete Add Button
normal speed
ticks: 0
view updates
continuous
Settings...
setup
personality-type
heroes


NetLogo
File Edit Tools Zoom Tabs Help
Interface Info Code
Edit Delete Add Button
normal speed
ticks: 0
view updates
continuous
Settings...
setup
personality-type
heroes
Command Center
Clear
observer>

Explanation:

Eachpersongets 3 friends and 3 enemies randomly selected from the population.

Additional Practice Commands

1. Changethenumber of persons:

```
set-default-shape persons "person"create-persons 200
```

2. Assign age categories: askpersons [if age<20 [set shape "circle"] if age>40 [set shape "square"]

```
]
```

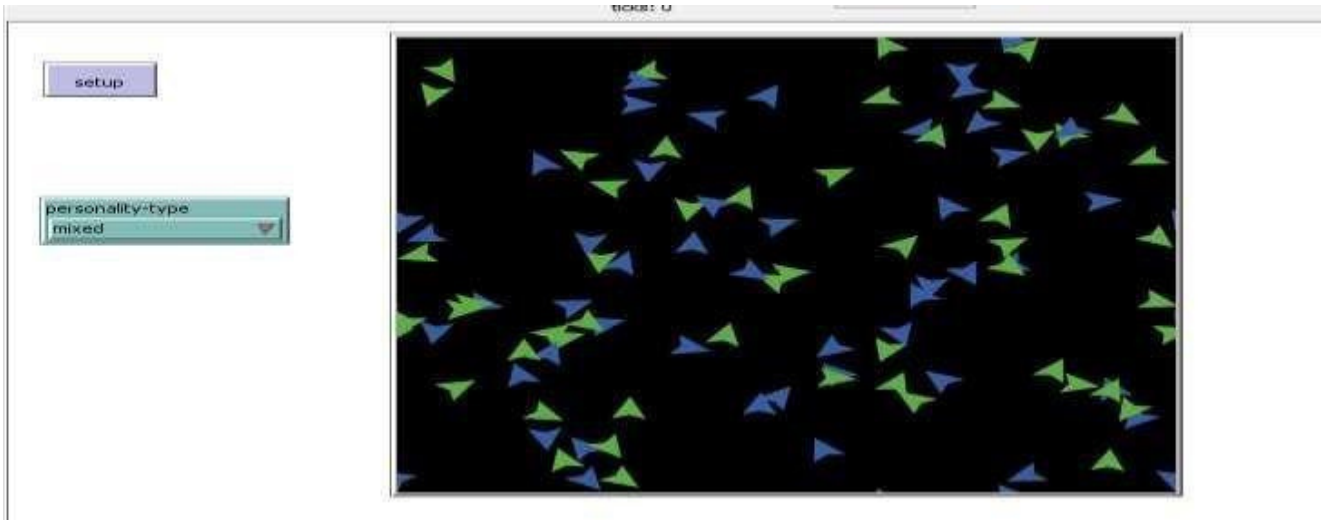
3. Highlightenemies: ask persons[askenemies [set color red]]

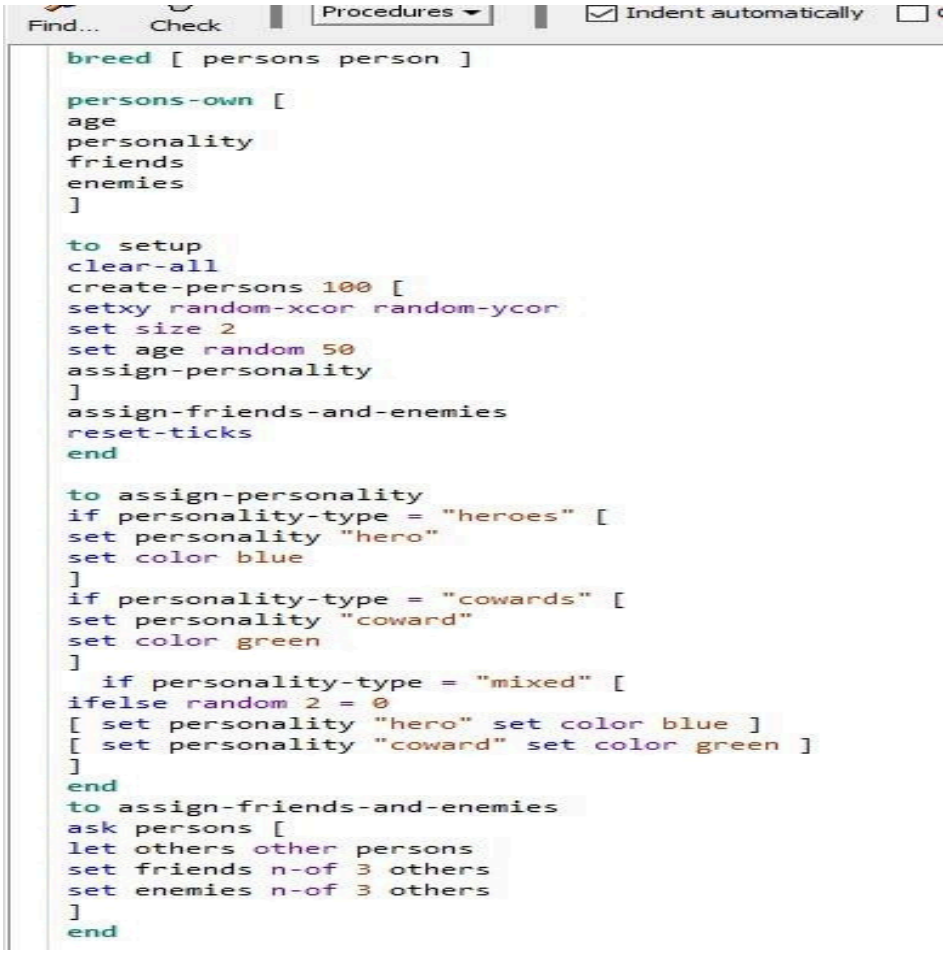
Observation

- Personalities appear as different colors.
- All people initialize with distinct ages and positions.
- Social structure (friends and enemies) is created even before movement begins.
- Thesystemisnow ready for advanced rules in Lab 9.

STUDENT TASK

1. Run the complete setup procedure and observe the world.

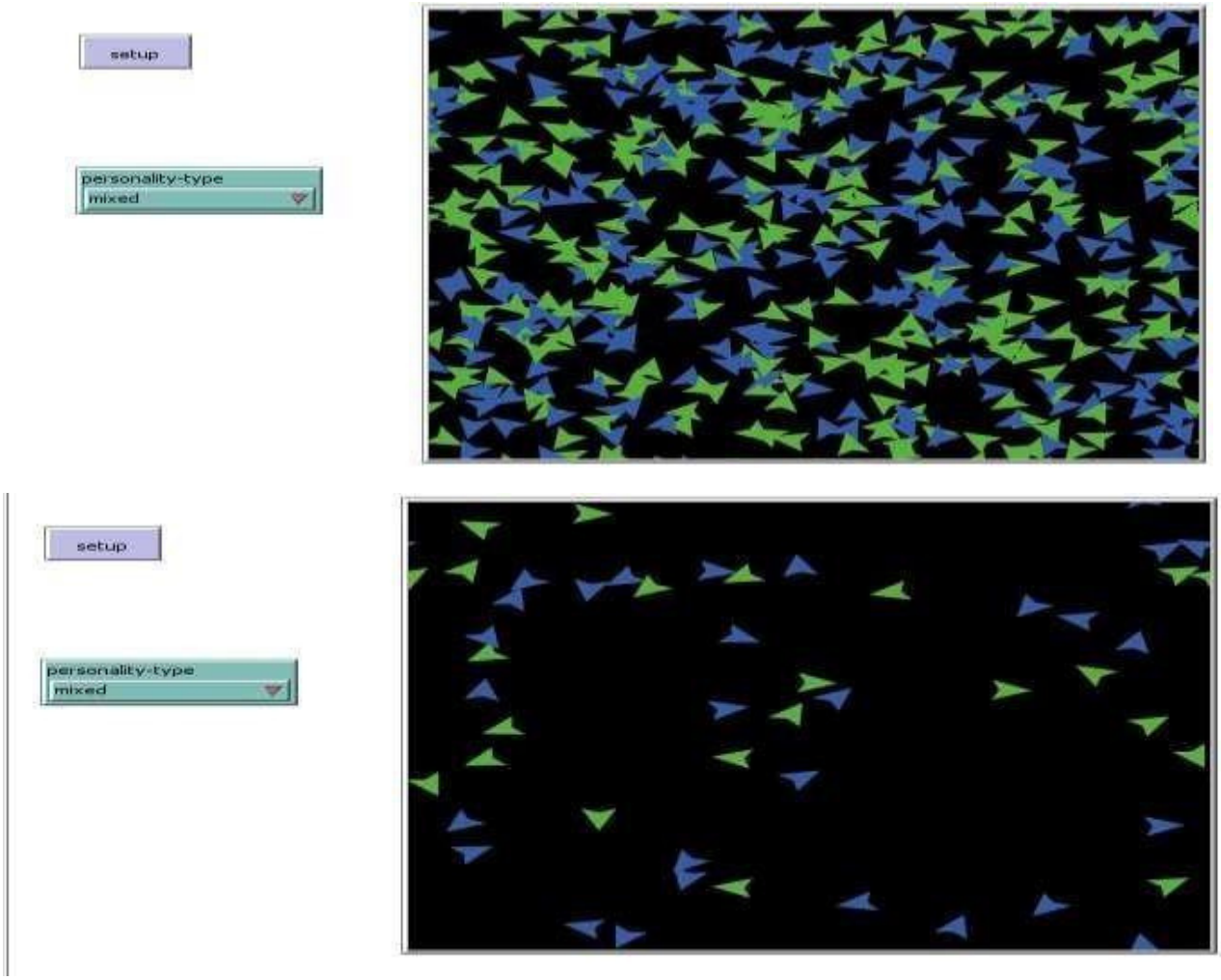




2. Change personality-type and compare the distribution of heroes and cowards.

- When the personality-type was set to **hero**, most agents behaved bravely and moved toward threats, resulting in a population dominated by heroes.
- When the personality-type was set to **coward**, agents avoided danger and moved away, causing cowards to dominate the population.
- In the **mixed** setting, both heroes and cowards were present, showing a balanced distribution with different movement and interaction behaviors.

3. Increase or decrease the number of persons.



Observation

When the number of persons was **increased**, the environment became more crowded, leading to more interactions between heroes and cowards. This increased the visibility of different behaviors.

When the number of persons was **decreased**, interactions were fewer, and agents moved more freely with less influence from others.

4. Modify the number of friends/enemies.

```
to assign-friends-and-enemies
ask persons [
let others other persons
set friends n-of 5 others
set enemies n-of 2 others
]
end
```

Observation

When the number of **friends** was increased, agents showed more cooperative behavior and tended to stay closer to familiar agents.
When the number of **enemies** was increased, agents showed more avoidance and aggressive reactions, resulting in frequent movement changes.
Balanced values of friends and enemies produced mixed and dynamic interactions among agents.

5. Record visible changes in colors, structure, and initialization.

During initialization, agents were randomly distributed across the environment. Heroes appeared in **blue** color, while cowards appeared in **green**, making personality types visually distinguishable.
Changes in model parameters affected the overall structure of the simulation, where crowding increased clustering and mixed personalities created visible variation in agent distribution.

Post-Lab Questions

1.Whydowedefinepersons-own variablesbeforeusingthem?
Defining persons-own variables allows each agent to store itsown properties, such as age, personality, friends, and enemies, which are used during the simulation to determine behavior.

2. What is the role of the chooser in this model?
The chooser lets the user select the personality type(hero, coward, or mixed)without changing the code, allowing easy experimentation and observation of different behaviors.

3. How does assigning friends and enemies affect later simulation behavior?
Friends and enemies influence interactions between agents, affecting movement, clustering, avoidance, and overall dynamics during the simulation.

4.Whyisrandomplacement usefulin agent-based modeling?
Random placement ensures a non-biased initial distribution of agents, allowing the simulation to reflect natural variability and explore different interaction patterns.

5.What isthepurposeofusingseparateprocedureslikeassign-personality?
Separate procedures improve code organization, readability, and reusability, making it easier to manage complex behaviors and assign properties systematically.

Lab 8: Strategy-Based Decision Making (El Farol Bar Model)

Objective

- To understand how agents make decisions using prediction-based strategies.
- To model attendance behavior in a crowded bar using limited memory.
- To assign multiple strategies to each agent and evaluate which strategy performs best.
- To simulatedecision-making under uncertainty using past attendance history.
- To exploreemergent behavior when many individuals predict the same event.

Tool / Environment

- **Software:** NetLogo 6.4.0
- **Platform:** Windows
- **Environment:** Interface + Code Tab

Theory

The *ElFarol Bar* problem is a classic agent-based economics simulation. The situation is:

- If too many people visit the bar, it becomes overcrowded.
- People want to go only if fewer than a threshold attend.
- No agent knows what others will do.
- Each agent must predict attendance based on past data.

Key Ideas

1. **Memory**

Eachagentstores a limited history of past attendance.

2. **Strategies**

Each agent has several prediction strategies.
A strategy is a list of weights used to calculate a predicted attendance.

3. **Best Strategy Selection**

After each tick, the agent checks which strategy would have given the most accurate prediction and adopts it for the next round.

4. **Decision Rule**

An agent goes to the bar if predicted attendance ≤ overcrowding threshold.

5. **Movement**

o Agents go to either home (green) or bar (blue) patches.

o Only one turtle can occupy a patch at a time.

This leads to complexsocial behavior even though each agent follows simple rules.

Important Concepts

Concept	Description
Globals	Shared information (attendance, history, bar/home areas).
Strategies	Weight-based prediction formulas.
Memory-size	How many past weeks an agent remembers.
Prediction	Estimated attendance for the next round.
Overcrowding-threshold	Maximum comfortable bar attendance.
move-to-empty-one-of	Ensures one turtle per patch (clean visualization).

LAB PROCEDURE

Step 1: Add Required Interface Elements

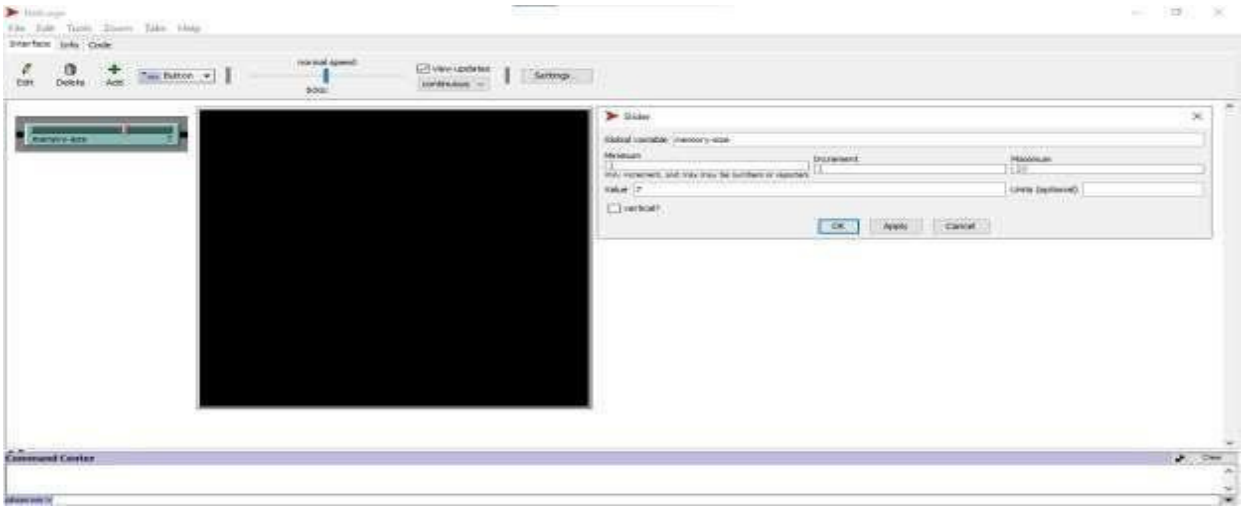
Sliders

Add these sliders in Interface:

1. **memory-size** o

Min: 1 o Max: 10

- Initial: 7

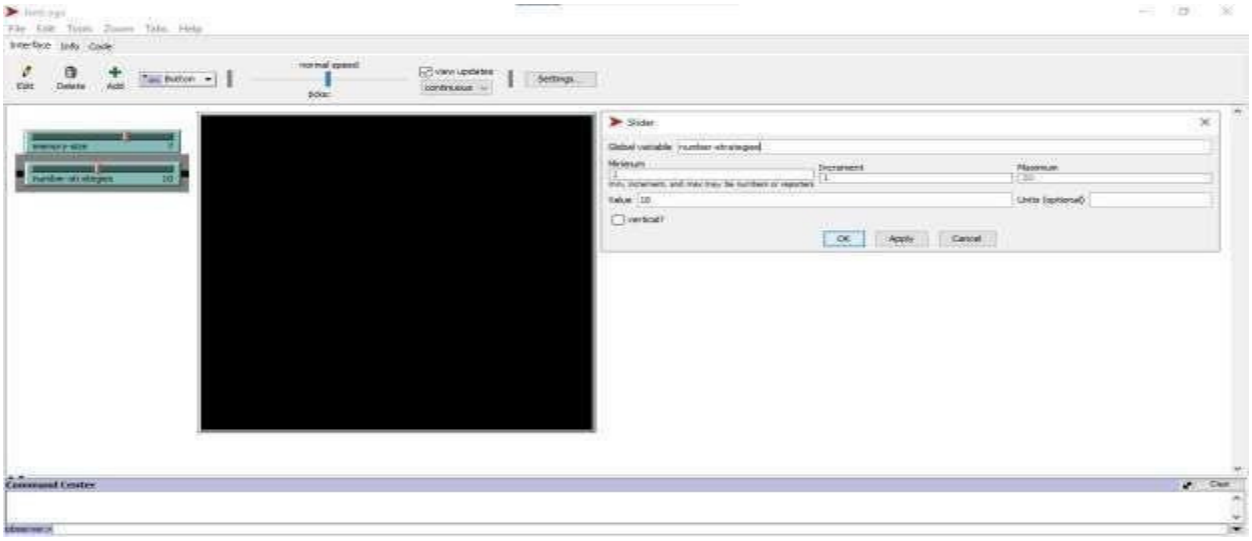


2. **number-strategies** ○

   Min:

   1 o Max: 20

   o Initial: 10

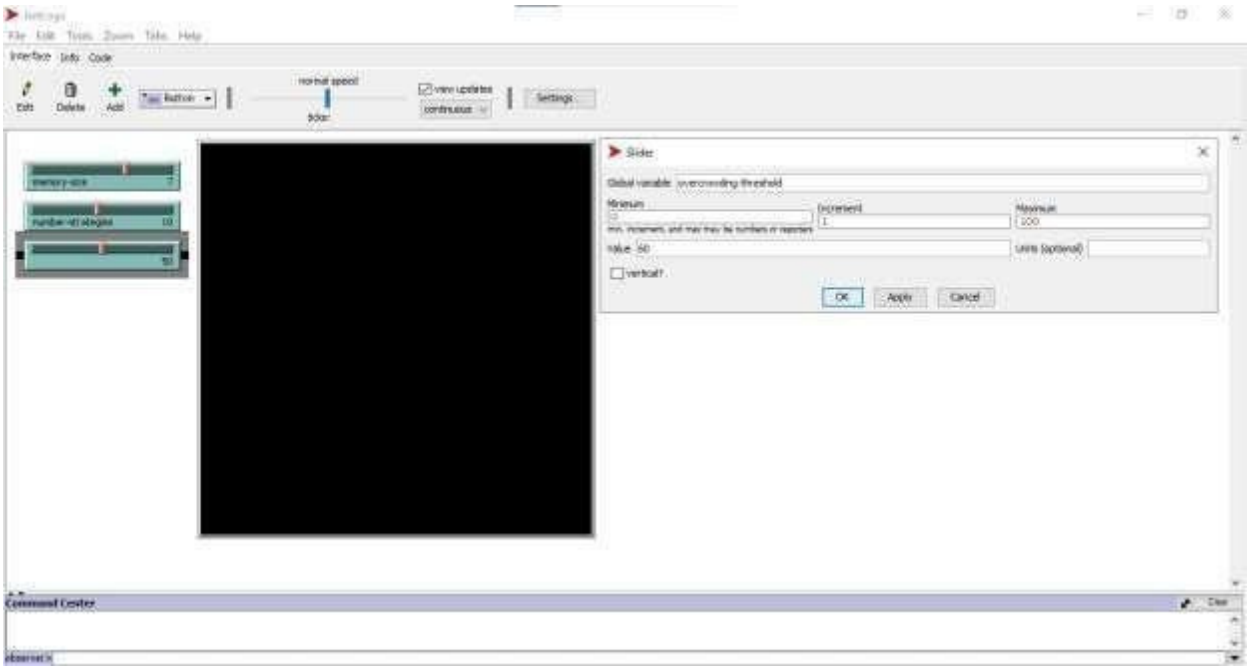


3. **overcrowding-**

   threshold o Min:

   0 o Max: 100 o

   Initial: 60

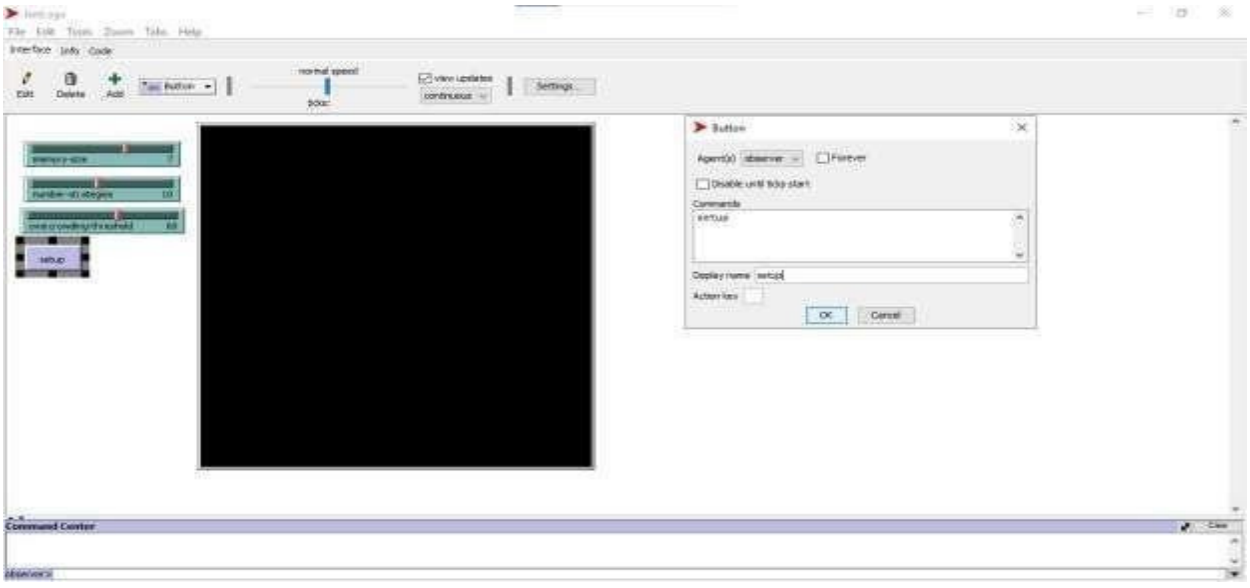


Buttons

1. **setup** ○

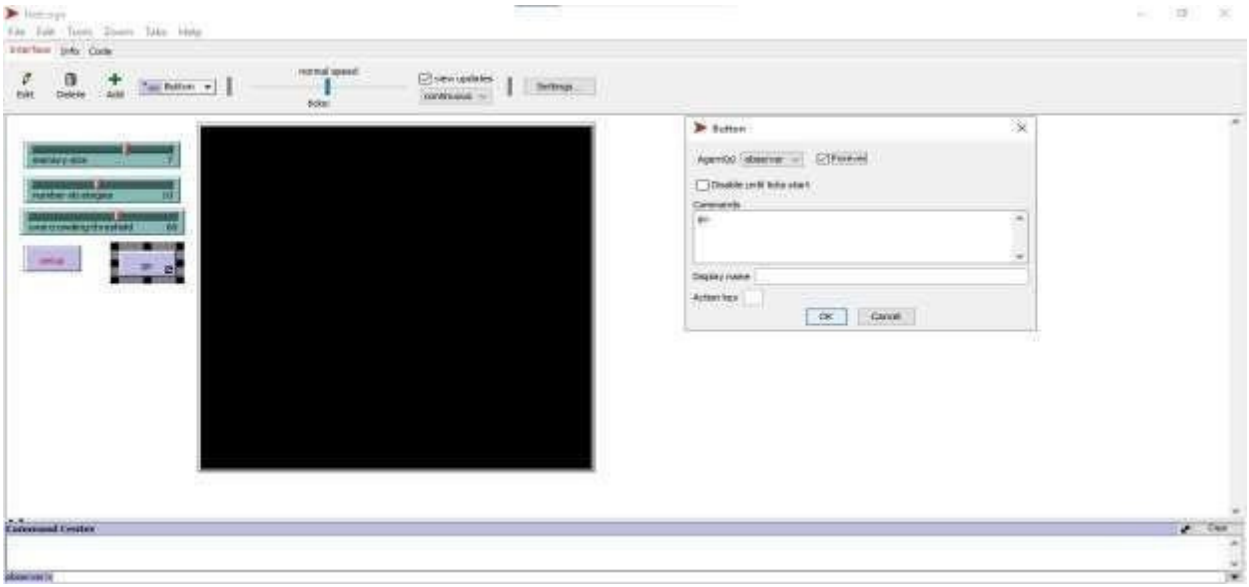
   Command: setup o

   Display name: setup



2. **go** ○ Command: go o

   Check: **Forever**



Add a Plot

- Right-click → **Plot**
- **Name:** Bar Attendance
- **X axis label:** Time
- **Y axis label:** Attendance
- **X min:** 0
- **X max:** 10
- **Y min:** 0
- **Y max:** 100

1. **Pen 1**
   - **Pen name:** attendance
   - **Pen update commands:** plot attendance

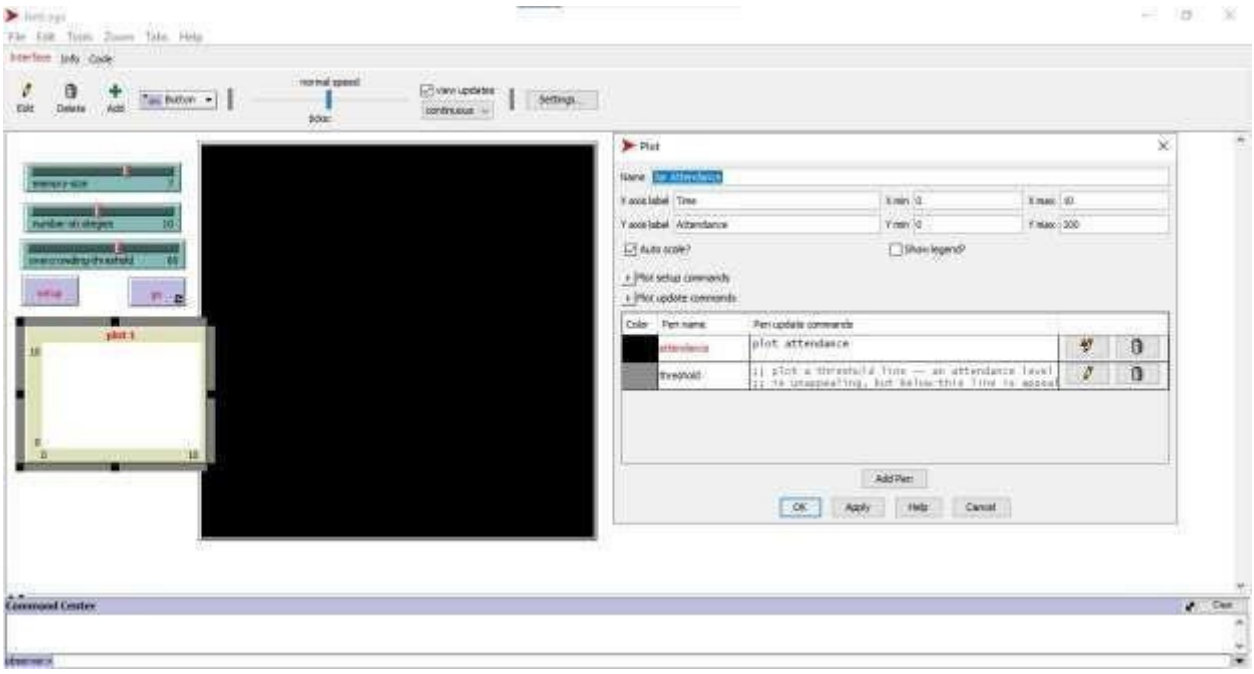
2. **Pen 2**
   - **Pen name:** threshold
   - **Pen update commands:**

;; plot a threshold line -- an attendance level above this line makes the bar

;; is unappealing, but below this line is appealing

plot-pen-reset plotxy 0 overcrowding- threshold

plotxy plot-x-max overcrowding- threshold



Step 2: Add the Full Code in the Code Tab

globals [attendance ;; the current attendanceatthe bar history ;; list of past values of attendance home-patches ;;agentset of green patches representing the residential area bar-patches;;agentset of blue patches representing the bar area crowded-patch ;;patch where we show the "CROWDED" label

]

turtles-own [

strategies ;; list of strategies best-strategy ;; index of the current best strategy attend? ;; true if the agent currently plans to attend the bar prediction ;; current prediction of the bar attendance

]

to setup clear-all set-default-shape turtles "person"

```
  ;; create the 'homes' set home-patches patches with [pycor < 0 or (pxcor <
     0 and pycor >= 0)] ask home-patches [ set pcolor green ]

  ;; create the 'bar' set bar-patches patches with [pxcor > 0 and
     pycor > 0] ask bar-patches [ set pcolor blue ]
  ;; initialize the previous attendance randomly so the agents have a history
  ;; to work with from the start set history n-values
     (memory-size * 2) [random 100]
  ;; the history is twice the memory, because we need at least a memory worth of
  history ;; for each point in memory to test how well the strategies would have worked

  set attendance first history


  ;; use one of the patch labels to visually indicate whether or not the
  ;; bar is "crowded" ask patch (0.75 * max-pxcor)
  (0.5 * max-pycor) [ set crowded-patch self set
  plabel-color red
  ]

  ;; create the agents and give them random strategies
  ;; these are the only strategies these agents will ever have though they ;;
     can change which of this "bag of strategies" they use every tick create-
     turtles 100 [ set color white move-to-empty-one-of home- patches set
     strategies n-values number-strategies [random-strategy] set best-
     strategy first strategies
    update-strategies
  ]

  ;; startthe clock reset-ticks
     end


to go
  ;; update the global variables ask crowded-patch
     [ set plabel "" ]
  ;; each agent predicts attendance at the bar and decides whether or not to go ask
  turtles [ set prediction predict-attendance best-strategy sublist history 0 memory-
  size set attend? (prediction <= overcrowding-threshold) ;; true or false


  ]
```

```
  ;; depending on their decision, the agents go to the bar or stay at home
  ask turtles [ ifelse attend? [ move-to-empty-one-of bar-patches set
  attendance attendance + 1 ]
  [ move-to-empty-one-of home-patches ]
  ]
  ;; if the bar is crowded indicate that in the view
  set attendance count turtles-on bar-patches if
  attendance > overcrowding-threshold [ ask
  crowded-patch [ set plabel "CROWDED" ]
  ]
  ;; update the attendance history
  ;; remove oldest attendance and prepend latest attendance
     set history fput attendance but-last history ;; the agents
     decide what the new best strategy is ask turtles [ update-
     strategies ]
  ;; advance the clock tick end

;; determines which strategy would have predicted the best results had it been used this round.
;; the best strategy is the one that has the sum of smallest differences between the
;; current attendance and the predicted attendance for each of the preceding
;; weeks (going back MEMORY-SIZE weeks)
;; this does not change the strategies at all, but it does (potentially) change the one
;; currently being used and updates the performance of all strategies to update-strategies
  ;; initialize best-score to a maximum, which is the lowest possible score let best-score
     memory-size * 100 + 1 foreach strategies [ the-strategy ->         let score 0        let week 1
     repeat memory-size [        set prediction predict-attendance the-strategy sublist history
     week (week + memory-size)         set score score + abs (item (week - 1) history -
     prediction) set week week + 1
    ]
    if (score <= best-score) [       set
     best-score score     set best- strategy
     the-strategy
    ]
  ]
end

;; thisreports a random strategy. a strategy is just a set of weights from -1.0 to 1.0 which
;; determines how much emphasis is put on each previous time period when making
;; anattendance prediction for the next time period to-report
random-strategy report n-values (memory-size + 1) [1.0 -
random-float 2.0] end
```

;; This reports an agent's prediction of the current attendance
;; using a particular strategy and portion of the attendance history. ;;
More specifically, the strategy is then described by the formula ;;
p(t) = x(t - 1) * a(t - 1) + x(t - 2) * a(t -2) +..
;; ... + x(t - MEMORY-SIZE) * a(t - MEMORY-SIZE) + c * 100,
;; where p(t) is the prediction at time t, x(t) is the attendance of the bar at time t,
;; a(t) is the weight for time t, c is a constant, and MEMORY-SIZE is an external paramete**to-report**
predict-attendance [strategy subhistory]
;; the first element of the strategy is the constant, c, in the prediction formula.
;; one can think of it as the the agent's prediction of the bar's attendance ;;
in the absence of any other data
;; then we multiply each week in the history by its respective weight report 100 * first strategy
+ sum (map [[weight week] -> weight * week] butfirst strategy subhistory) **end**

;; In this model it doesn't really matter exactly which patch
;; a turtle is on, only whether the turtle is in the home area
;; or the bar area. Nonetheless, to make a nice visualization ;;
this procedure is used to ensure that we only have one ;;
turtle per patch.
to move-to-empty-one-of [locations] ;; turtle procedure
move-to one-of locations while [any? other turtles-
here] [move-to one-of locations
]
end

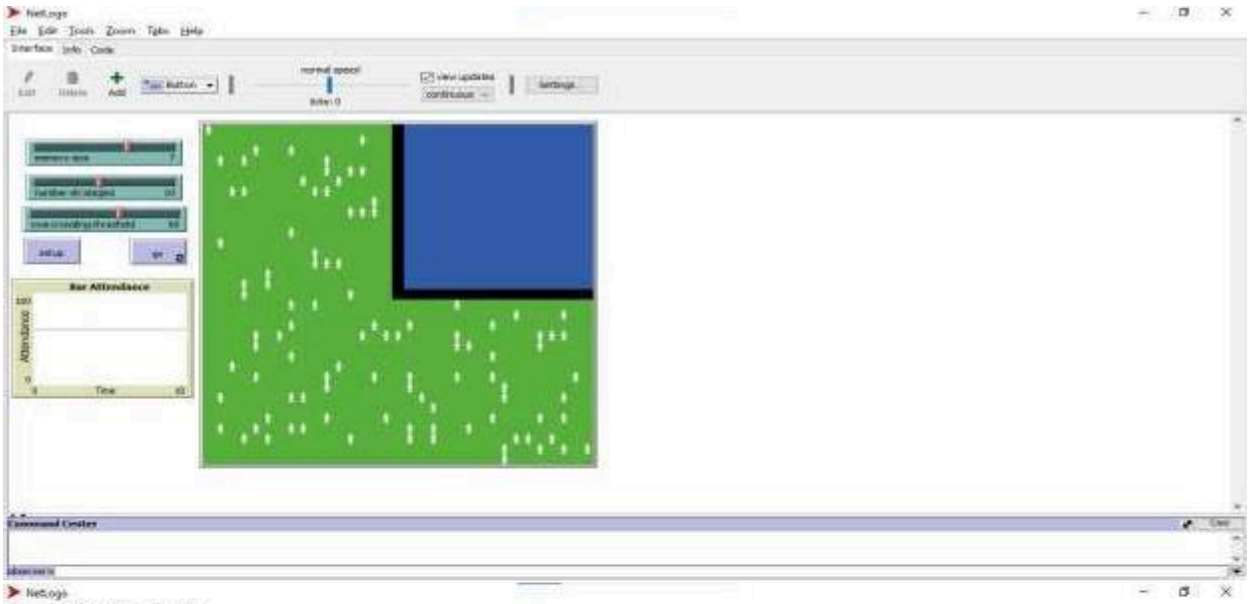

El Farol - NetLogo
File Edit Tools Zoom Tabs Help
Interface Info Code
Find... Check
Procedures
Indent automatically
Code Tab in separate window
set attendance first history
;; use one of the patch labels to visually indicate whether or not the
;; bar is "crowded"
ask patch (0.75 * max-pxcor) (0.5 * max-pycor) [
set crowded-patch self
set plabel-color red
]
;; create the agents and give them random strategies
;; these are the only strategies these agents will ever have though they
;; can change which of this "bag of strategies" they use every tick
create-turtles 100 [
set color white
move-to-empty-one-of home-patches
set strategies n-values number-strategies [random-strategy]
set best-strategy first strategies
update-strategies
]
;; start the clock
reset-ticks
end

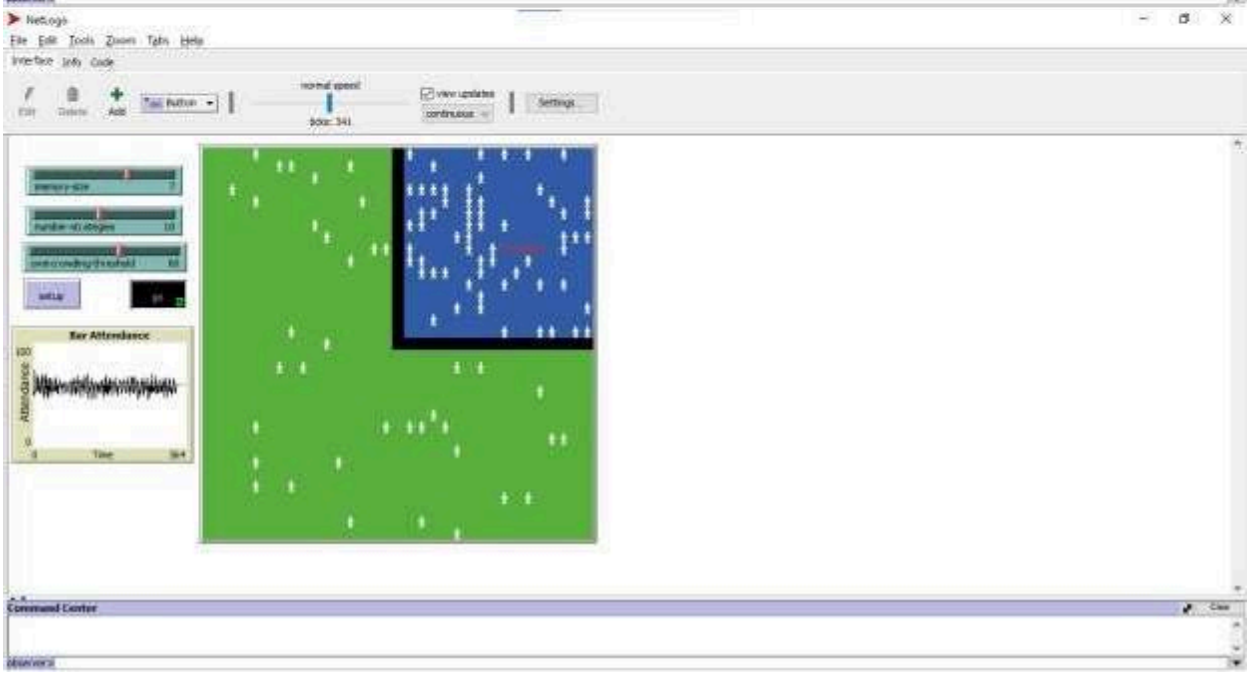

El Farol - NetLogo
File Edit Tools Zoom Tabs Help
Interface Info Code
Find... Check
Procedures
Indent automatically
Code Tab in separate window
to go
;; update the global variables
ask crowded-patch [set plabel ""]
;; each agent predicts attendance at the bar and decides whether or not to go
ask turtles [
set prediction predict-attendance best-strategy sublist history 0 memory-size
set attend? (prediction <= overcrowding-threshold) ;; true or false
]
;; depending on their decision, the agents go to the bar or stay at home
ask turtles [
ifelse attend?
[move-to-empty-one-of bar-patches
set attendance attendance + 1]
[move-to-empty-one-of home-patches]
]
;; if the bar is crowded indicate that in the view
set attendance count turtles-on bar-patches
if attendance > overcrowding-threshold [
ask crowded-patch [set plabel "CROWDED"]
]
;; update the attendance history
;; remove oldest attendance and append latest attendance
set history fput attendance but-last history
;; the agents decide what the new best strategy is
ask turtles [update-strategies]
;; advance the clock
tick
end


El Farol - NetLogo
File Edit Tools Zoom Tabs Help
Interface Info Code
Find... Check
Procedures
Indent automatically
Code Tab in separate window
;; determines which strategy would have predicted the best results had it been used this round.
;; the best strategy is the one that has the sum of smallest differences between the
;; current attendance and the predicted attendance for each of the preceding
;; weeks (going back MEMORY-SIZE weeks)
;; this does not change the strategies at all, but it does (potentially) change the one
;; currently being used and updates the performance of all strategies
to update-strategies
;; initialize best-score to a maximum, which is the lowest possible score
let best-score memory-size * 100 + 1
foreach strategies [the-strategy ->
let score 0
let week 1
repeat memory-size [
set prediction predict-attendance the-strategy sublist history week (week + memory-size)
set score score + abs (item (week - 1) history - prediction)
set week week + 1
]
if (score <= best-score) [
set best-score score
set best-strategy the-strategy
]
]
end
;; this reports a random strategy. a strategy is just a set of weights from -1.0 to 1.0 which
;; determines how much emphasis is put on each previous time period when making
;; an attendance prediction for the next time period
to-report random-strategy
report n-values (memory-size + 1) [1.0 - random-float 2.0]
end

```
;; This reports an agent's prediction of the current attendance
;; using a particular strategy and portion of the attendance history.
;; More specifically, the strategy is then described by the formula
;; p(t) = s(t - 1) * a(t - 1) + s(t - 2) * a(t -2) +...
;;      ... + s(t - MEMORY-SIZE) * a(t - MEMORY-SIZE) + c * 100,
;; where p(t) is the prediction at time t, s(t) is the attendance of the bar at time t,
;; a(t) is the weight for time t, c is a constant, and MEMORY-SIZE is an external parameter.
to-report predict-attendance [strategy subhistory]
  ;; the first element of the strategy is the constant, c, in the prediction formula
  ;; one can think of it as the the agent's prediction of the bar's attendance
  ;; in the absence of any other data
  ;; then we multiply each week in the history by its respective weight
  report 100 * first strategy + sum (map [ [weight week] -> weight * week ] butfirst strategy subhistory)
end

;; In this model it doesn't really matter exactly which patch
;; a turtle is on, only whether the turtle is in the home area
;; or the bar area.  Nevertheless, to make a nice visualization
;; this procedure is used to ensure that we only have one
;; turtle per patch.
to move-to-empty-one-of [locations]  ;; turtle procedure
  move-to one-of locations
  while [any? other turtles-here] [
    move-to one-of locations
  ]
end
```

Results:



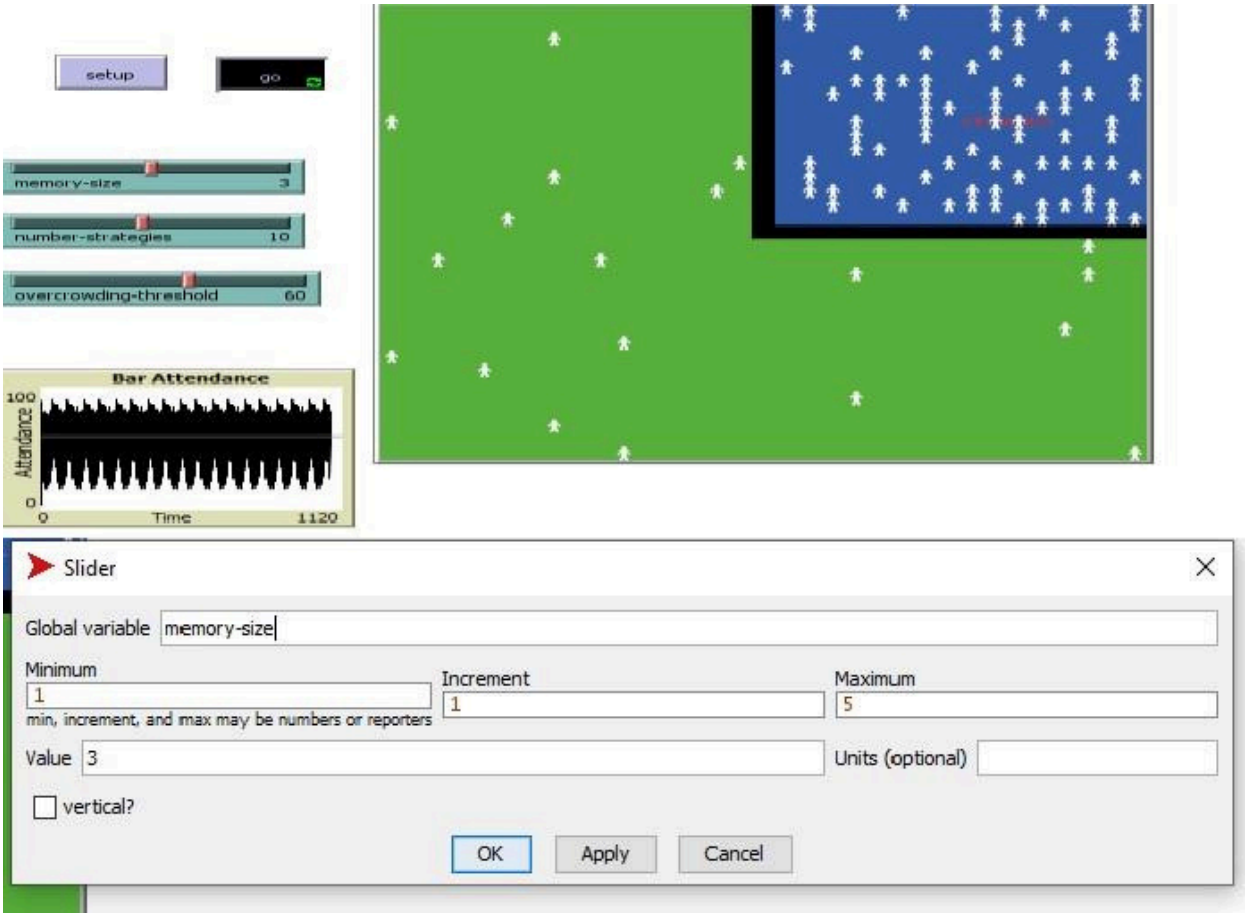


Observation

- Attendance fluctuates each tick based on predictions.
- When too many agents guess low attendance, the bar becomes crowded.
- The model displays CROWDED when attendance exceeds the threshold.
- Agents adapt their strategies over time, improving predictions.
- Complex group behavior emerges from simple rules.

STUDENT TASK

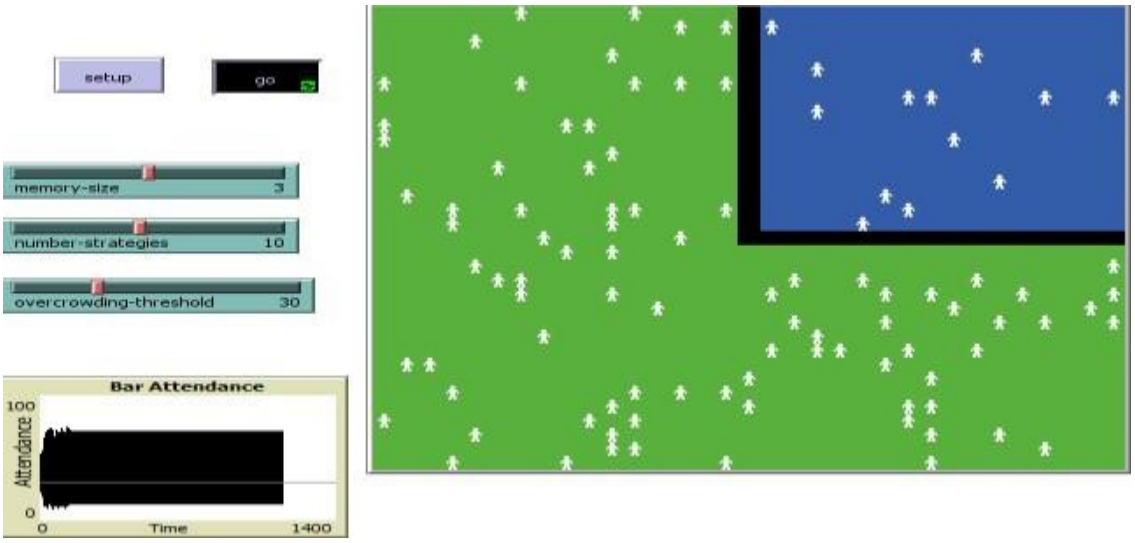
1. Run the modelwith different memory-size values and observe how prediction accuracy changes.



Observation

With a **memory-size of 3**, agents used the past three rounds of attendance to predict the next. Predictions were moderately accurate, leading to some fluctuations in bar attendance, but overall the agents were able to avoid extreme overcrowding most of the time.

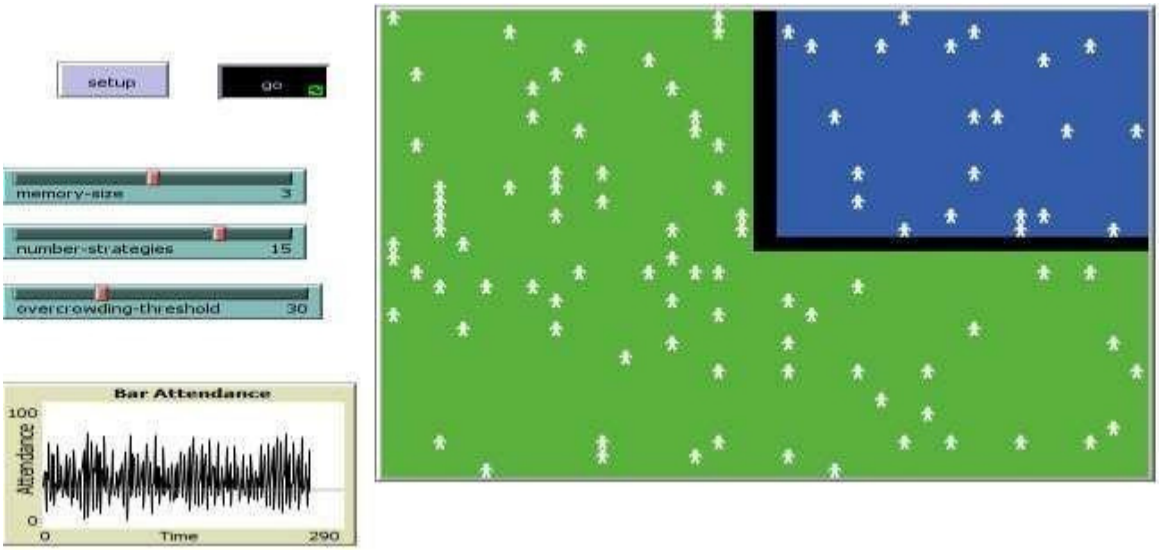
2. Change the overcrowding-threshold and analyze its effect on agent attendance.



OBSERVATION

Low threshold → 30: Bar reaches "crowded" status quickly → fewer agents decide to attend → lower attendance.

3. Increase number-strategies and check if behavior becomes more stable.

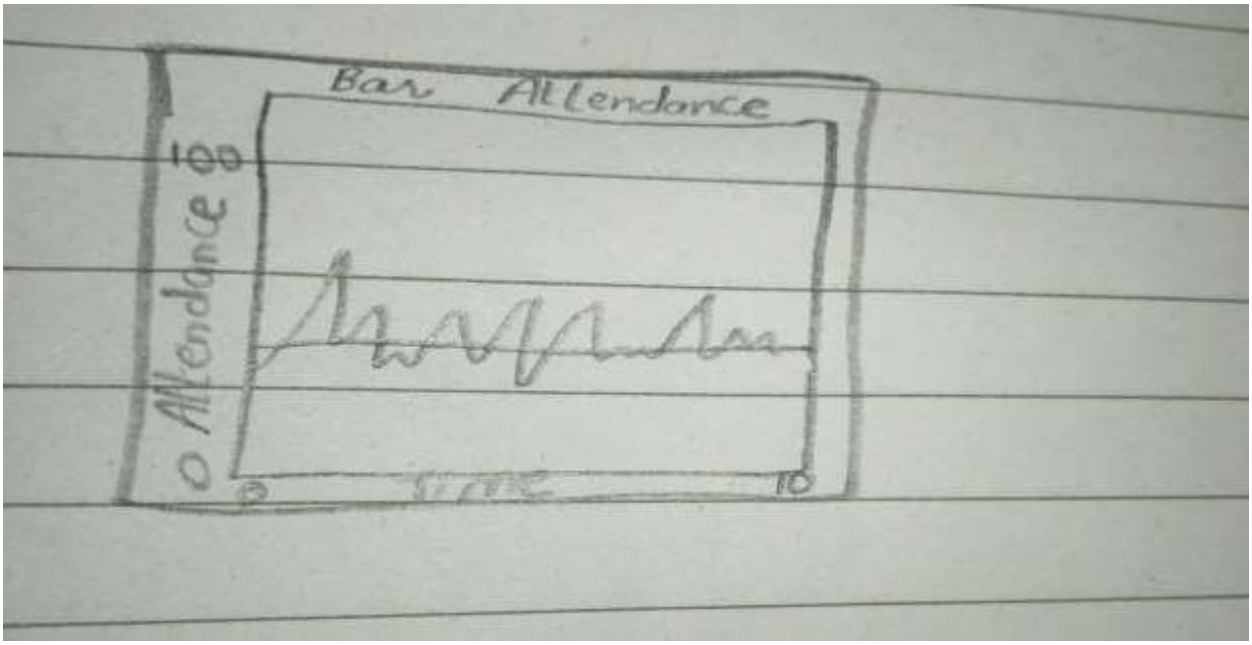


OBSERVATION

With **15 strategies per agent**, each agent had more options to choose from when predicting bar attendance. This allowed agents to adapt better to past attendance patterns, resulting in more stable behavior and smoother attendance fluctuations compared to models with fewer strategies. **4.** Compare early ticks vs late ticks — do predictions improve?

- In the **early ticks**, agents had limited history to base their predictions on, so attendance predictions were less accurate and fluctuated more.
- In the **late ticks**, as more attendance history accumulated, agents chose better-performing strategies, leading to more accurate predictions and smoother attendance patterns.

5. Draw a graph of attendance over time in your notebook.



Post-Lab Questions

1. Why does each agent need multiple prediction strategies?
Multiple prediction strategies allow agents to adapt to differentsituations and choose the most successful approach based on past outcomes.

2. How does memory-size influence the agent's decision-making?
Memory-size determines how much past information an agent can use,affecting its ability to make accurate predictions and improve decisions over time.

3. What is the importance of the overcrowding threshold?
The overcrowding threshold prevents too many agents from choosing the same option, promoting diversity in decisions and avoiding resource depletion.

4. Why is it necessary to update the best strategy every tick?
Updating the best strategy each tick ensures agents adapt to changing conditions and continue using the most successful strategy.

5. How does the model demonstrate emergent behavior?

The model shows emergentbehavioras complexpatterns arise from simple agent interactions, such as the overall distribution of heroes and cowards over time.

Lab 9: Brave and Cowardly Behavior Simulation in NetLogo

Objective

- To simulate two types of turtles: **brave** and **cowardly**.
- To assign each turtle a friend and an enemy.
- To program two different behaviors:

  o brave turtles move toward danger o

  cowardly turtles move away from

  danger
- To run the model using a continuous go loop.
- To introduce reproducible experiments using the preset procedure.

Language/Tool

- **Tool:** NetLogo6.4.0
- **Platform:** Windows
- **Environment:** Interface + Command Center + Code Tab

Theory

This model uses two simple ideas:

1. **Brave Behavior**
   - Brave turtles move towards the midpoint between their friend and enemy.
   - This behavior simulates risk-taking or approaching danger.
2. **Cowardly Behavior**
   - Coward turtles use their friend as a shield.
   - They move to a point behindtheir friend, away from the enemy.
3. **Friend and Enemy Selection**
   - Each turtle chooses: o **1 friend**

     o **1 enemy**
   - Theserelationshipsguidemovement.
4. **Chooser-Based Personalities** • Theuser selects from: brave, cowardly, or mixed
   - Determineshowturtlesarecolored and how they behave.

Lab Procedure

Step 1: Add Interface Items

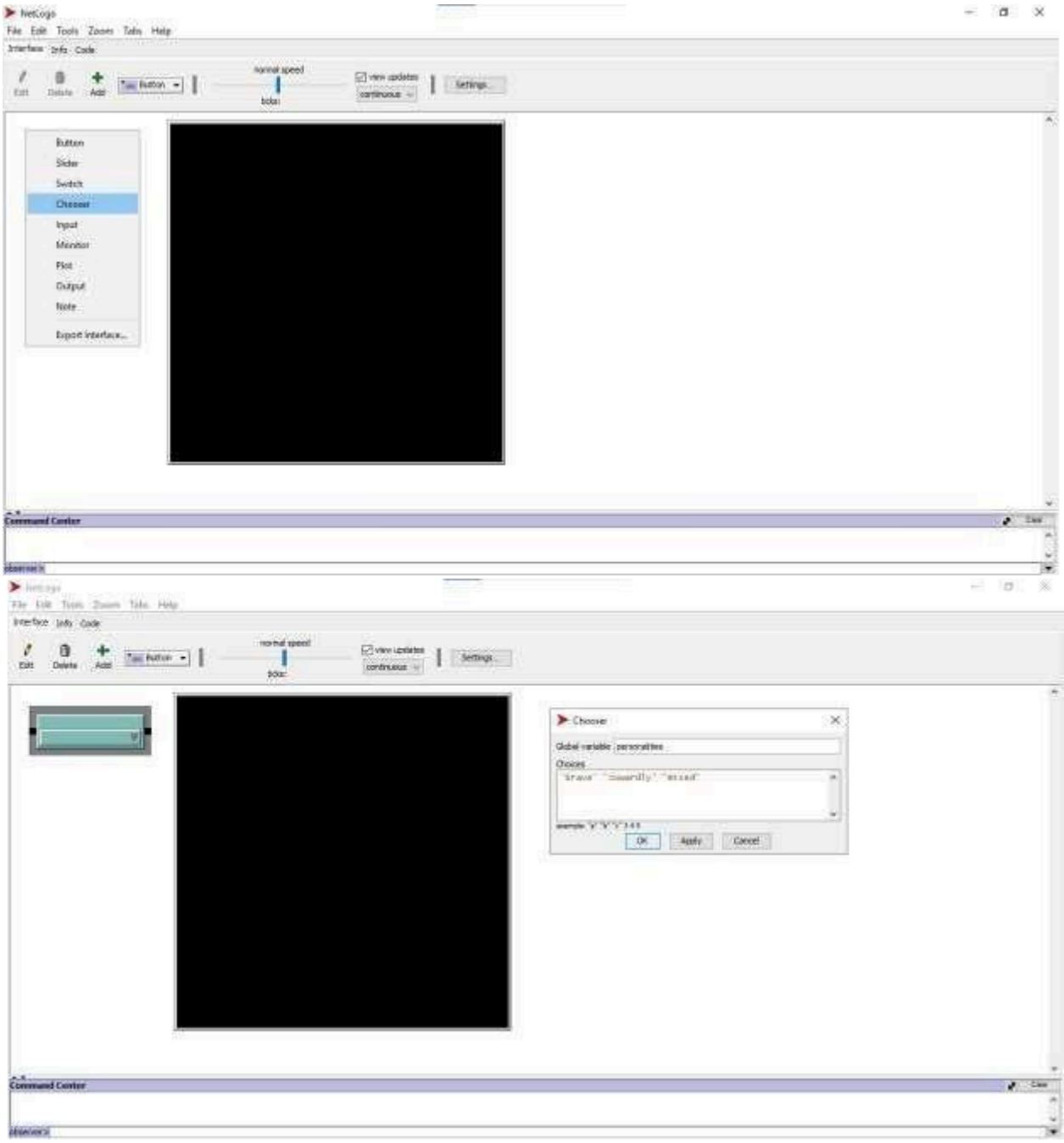
Add a Chooser:

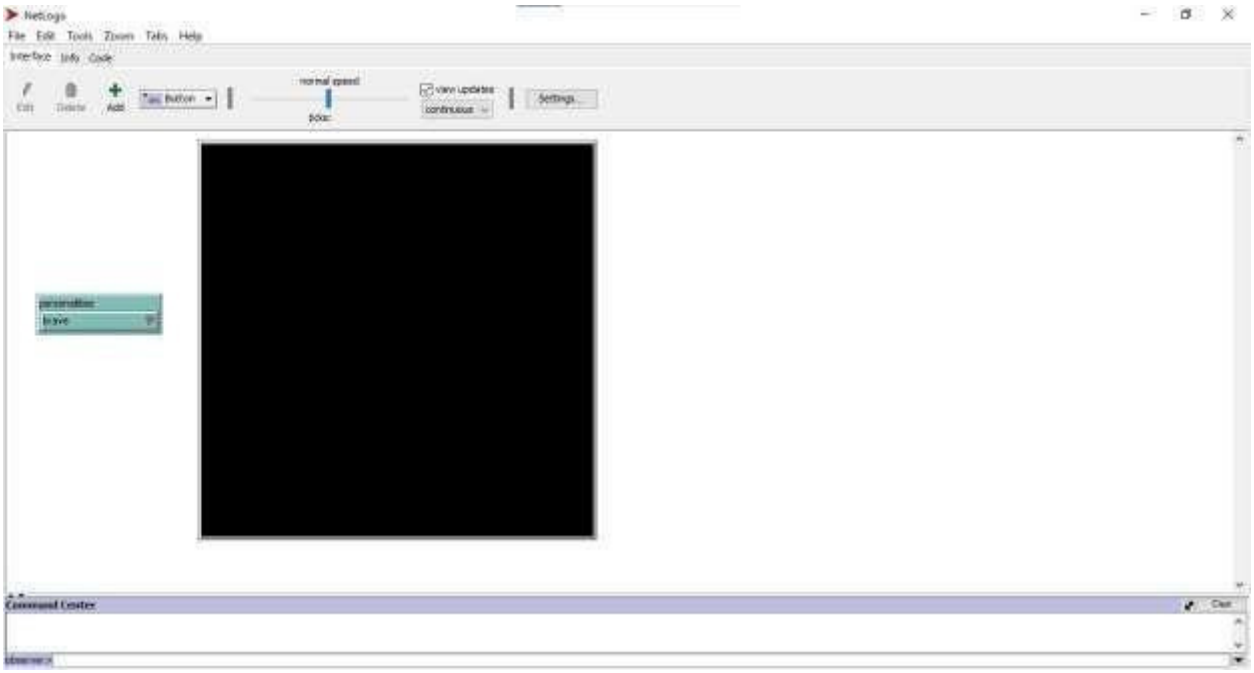
1. Go to the **Interface** tab.

2. Right-click → select **Chooser**.

3. Set:

   - **Name:** personalities
   - **Choices:** "brave" "cowardly" "mixed"





Add a Slider: 1. Right-click →

Slider

2. Set:

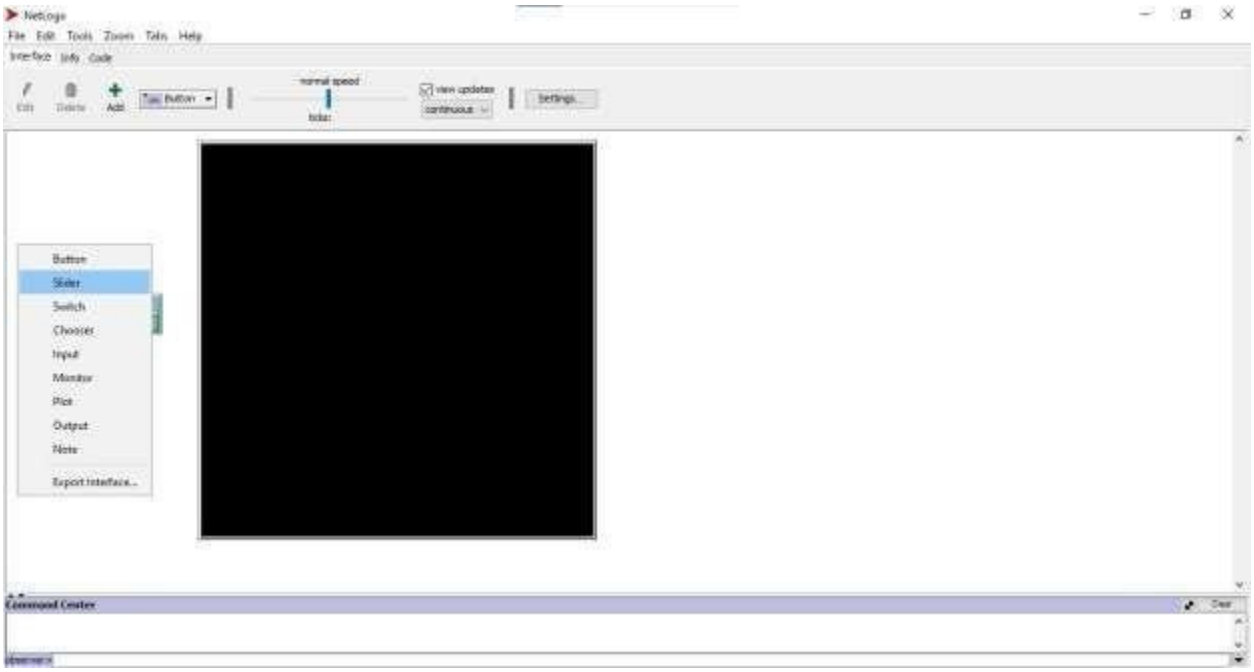
- **Name:**

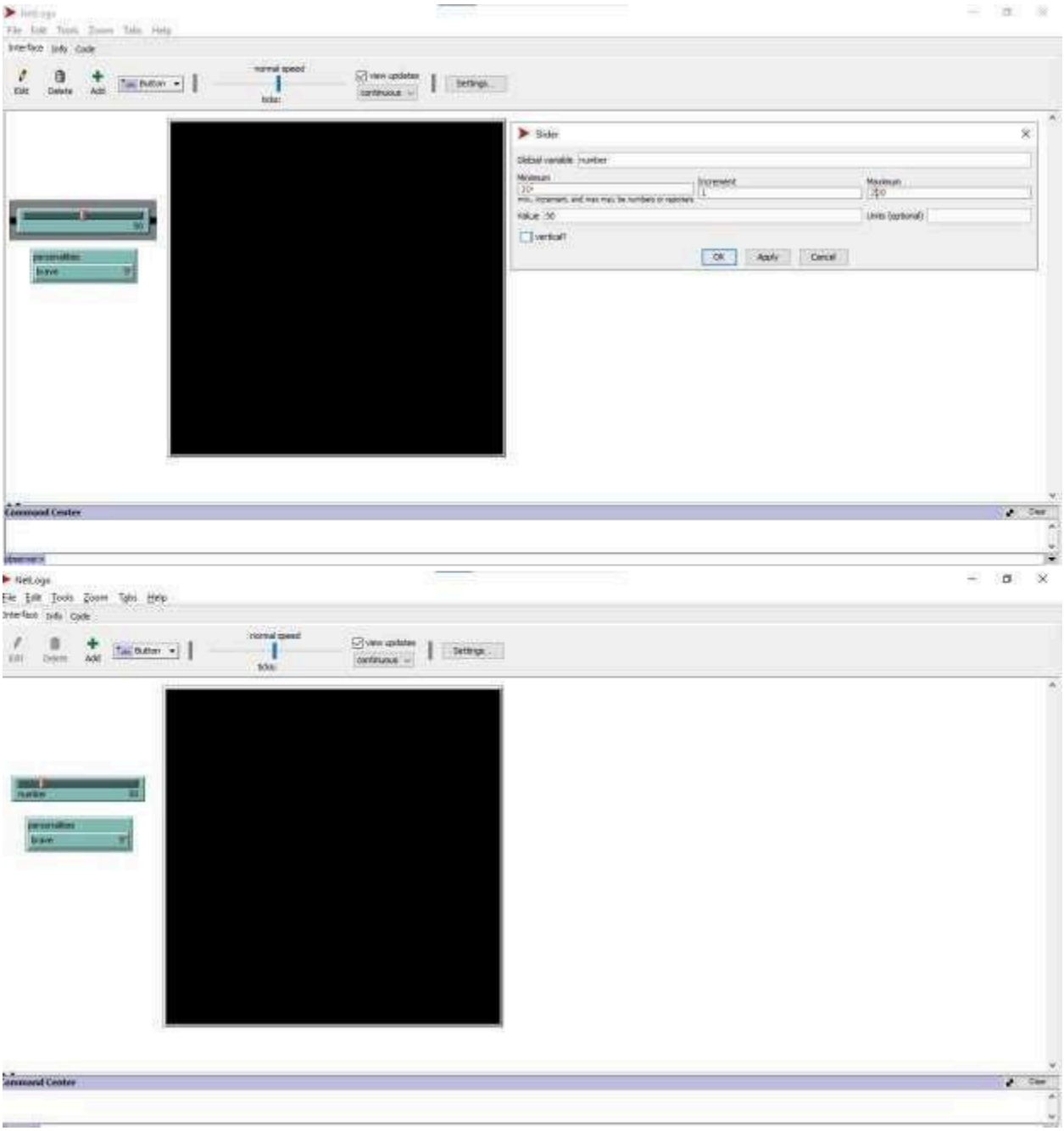
  number o

  Min: 10 o

  Max: 200

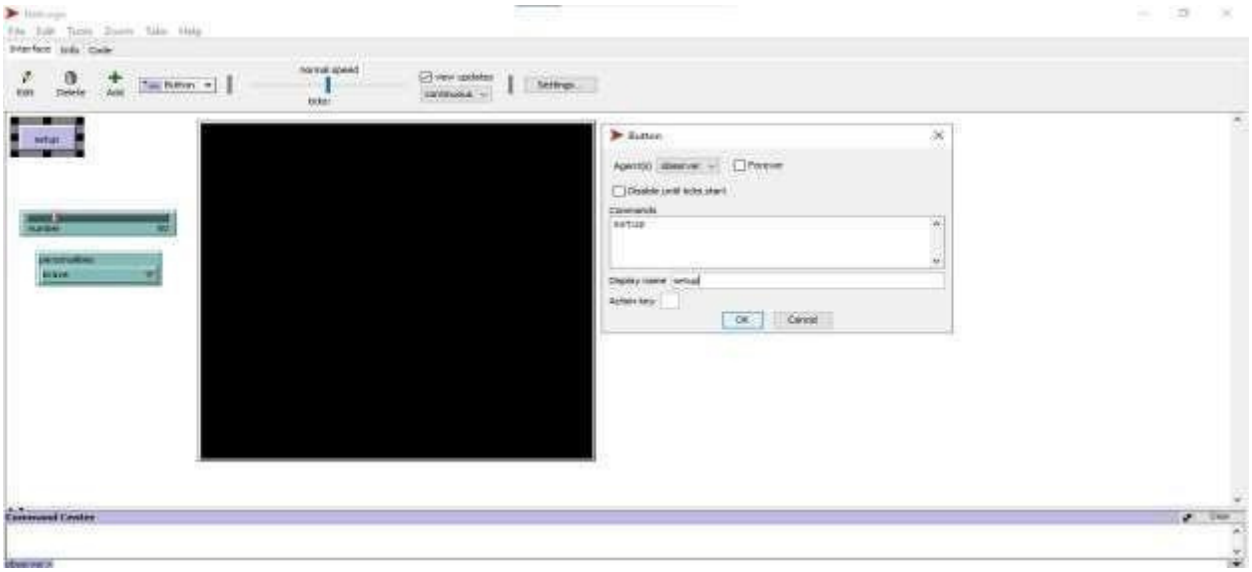
- **Initial value:** 50


NetLogo
File Edit Tools Zoom Tabs Help
Interface Info Code
Edit
Delete
Add
Button
normal speed
ticks
view updates
continuous
Settings...
Button
Slider
Switch
Chooser
Input
Monitor
Plot
Output
Note
Export interface...
Command Center
Clear



Add Buttons:

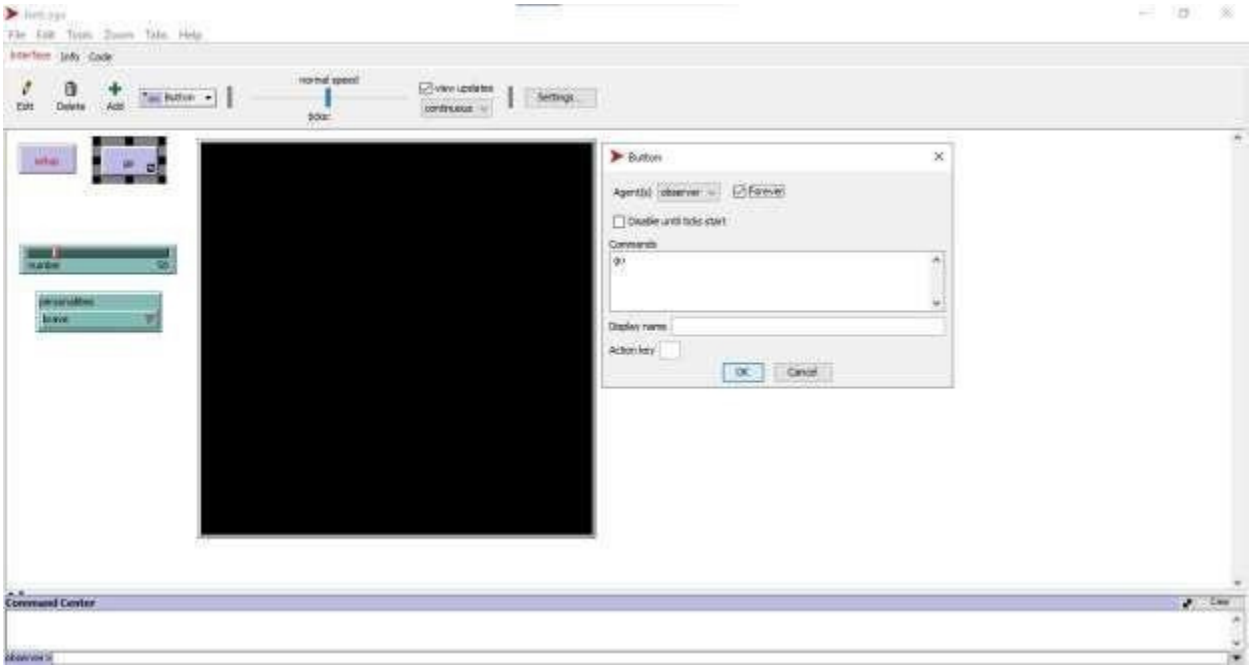
1. **Button 1:**
   - Command: setup
   - Name: setup

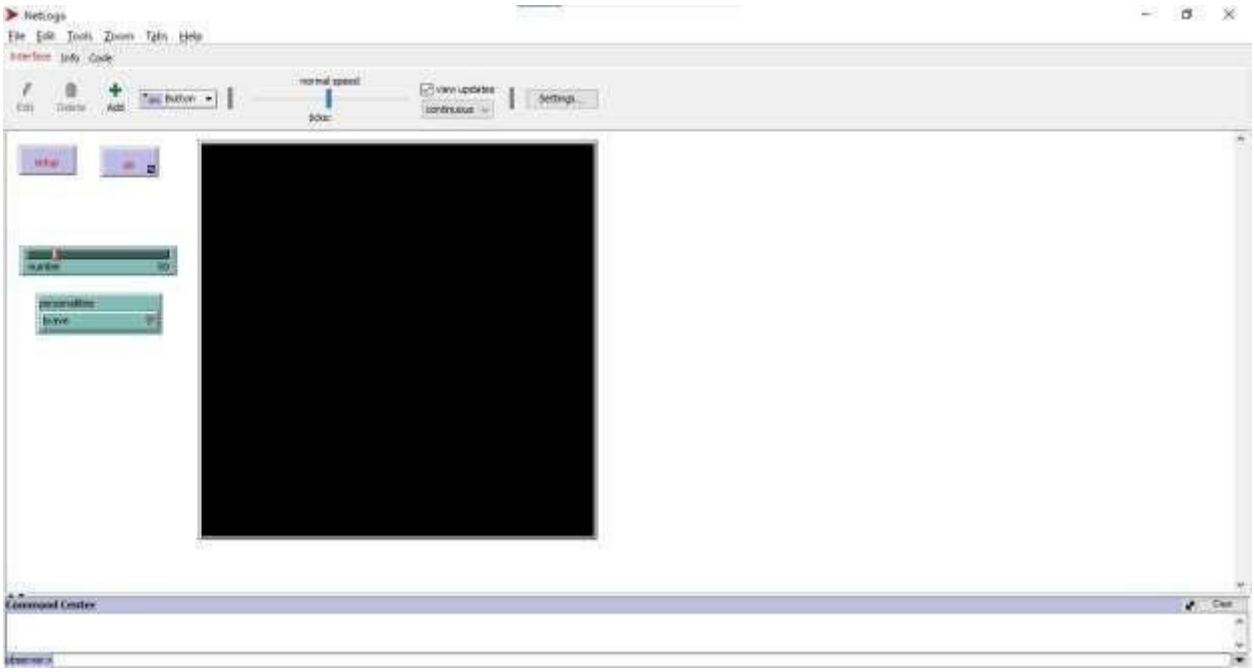


2. **Button 2:**

- Command: go
- Check:

Forever





Your interface is now ready.

Step 2: Add the Complete Code to the Code Tab

Now write below code the **Code Tab**:

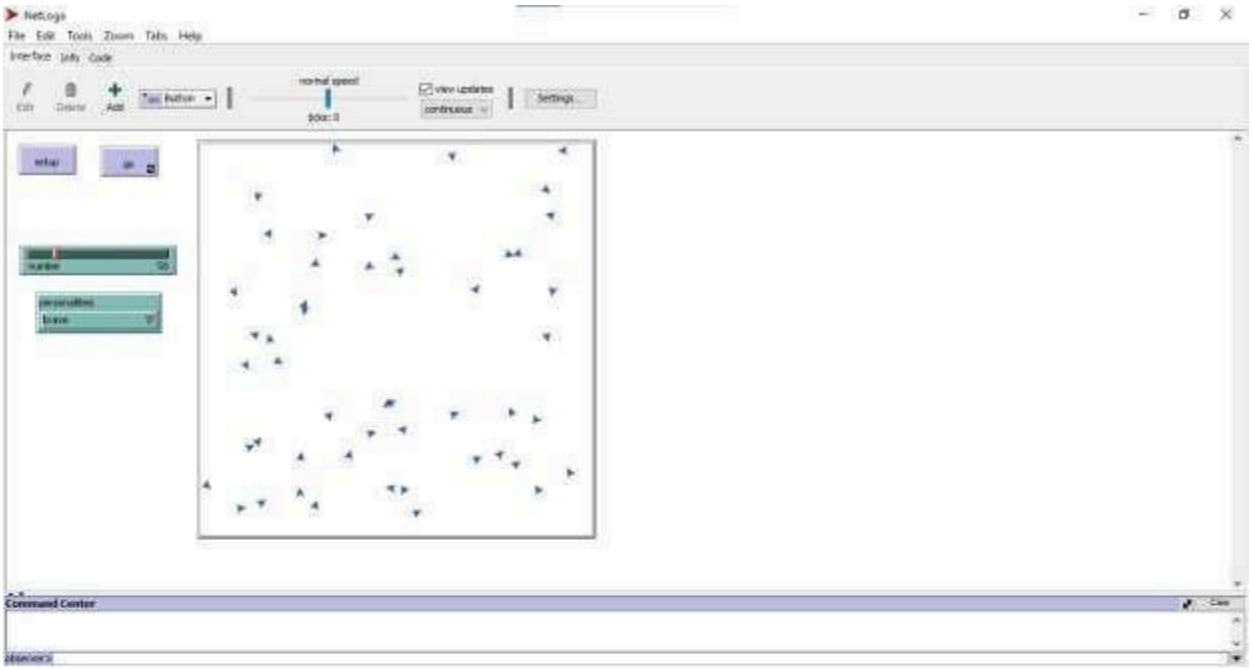
```
turtles-own [ friend enemy]

to setup clear-all ask patches [ set pcolor white ] ;; create a blank
    background create-turtles number [        setxy random- xcor
    random-ycor

;; set the turtle personalities based on chooser if
(personalities = "brave") [ set color blue ] if
(personalities = "cowardly") [ set color red ]
if (personalities = "mixed")     [ set color one-of [ red blue]]

  ;; choose friend and enemy targets set
    friend one-of other turtles
```

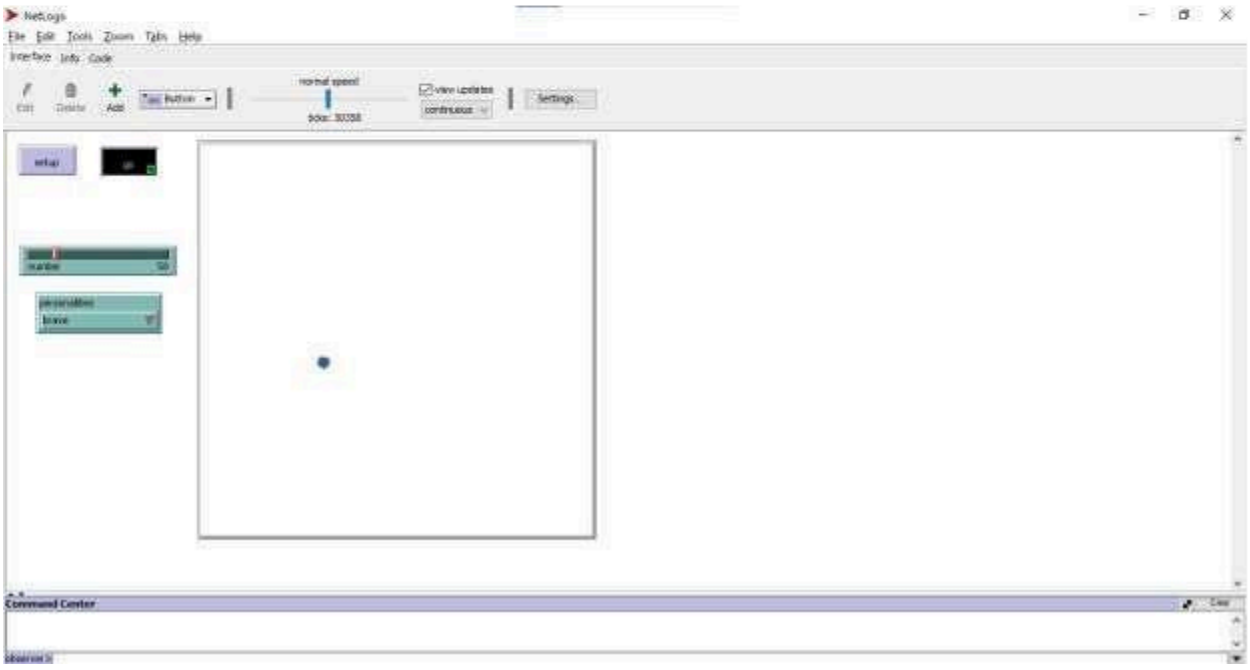
```
set enemy one-of other turtles

]
reset-ticks end to go ask turtles [ if
(color = blue) [ act-bravely ] if

(color

    = red) [ act-cowardly ]

  ]

tick

end   to   act-
bravely
  ;; move toward the midpoint of your friend and enemy facexy

    ([xcor] of friend + [xcor] of enemy) / 2

    ([ycor] of friend + [ycor] of enemy) / 2 fd 0.1 end

to act-cowardly
    ;; put your friend between you and your enemy facexy [xcor]
    of friend + ([xcor] of friend - [xcor] of enemy) / 2 [ycor] of friend

    + ([ycor] of friend - [ycor] of enemy) / 2 fd 0.1 end


to preset [ seed ]
;; sets up the model for use with a particular random seed and constant
;; model parameters, so that a particular pattern can be re-created. set

personalities "mixed" set number 68 random-seed seed setup end
```

Step 3: Run the Model

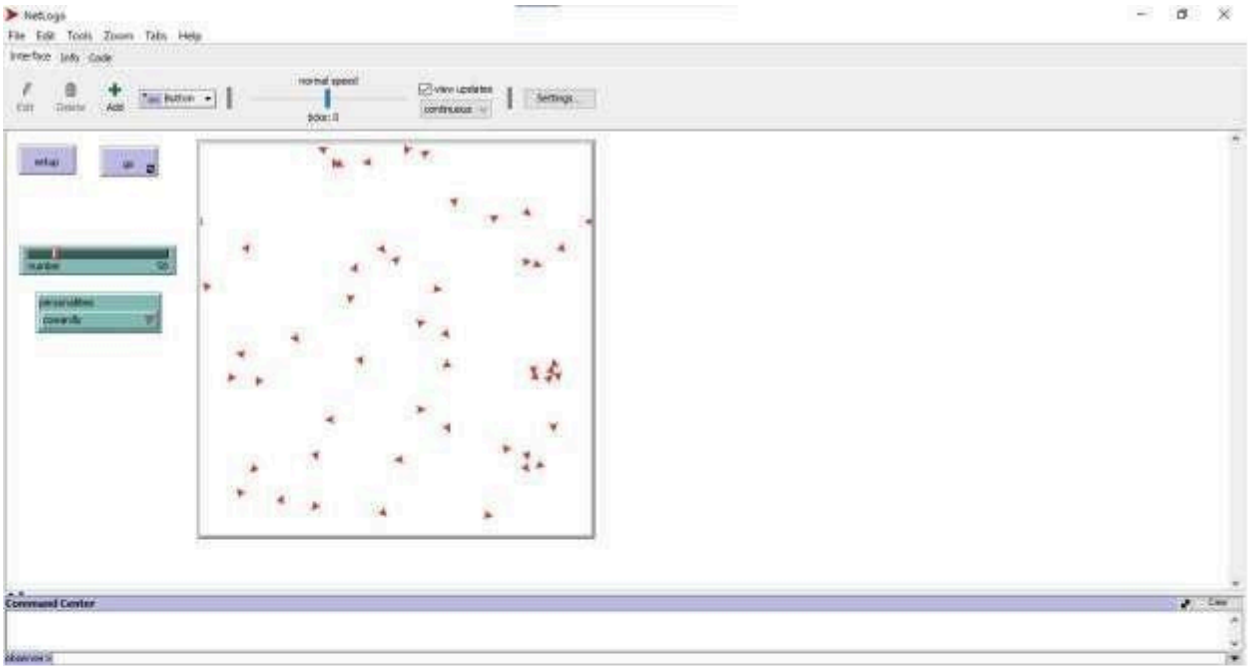
1. Select **braveçowardly** ,or **mixed** from chooser.
2. Set numberofturtles using the slider.
3. Click **setup**.
4. Click **go** (which runs forever).

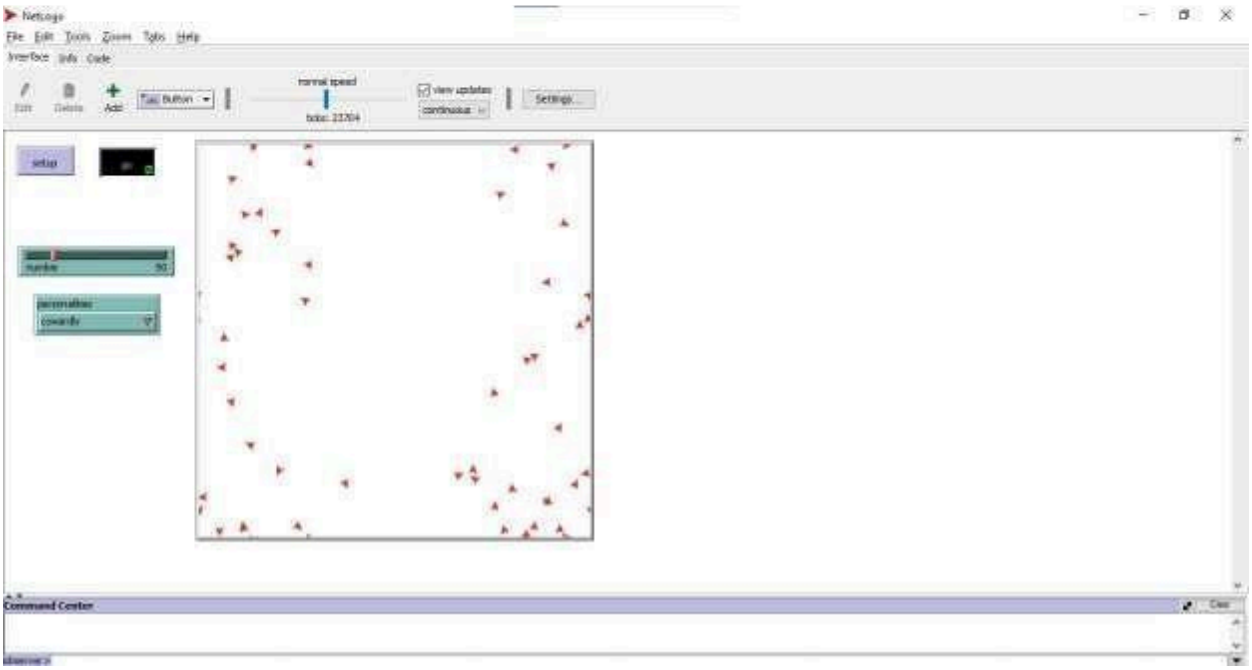
For **brave**



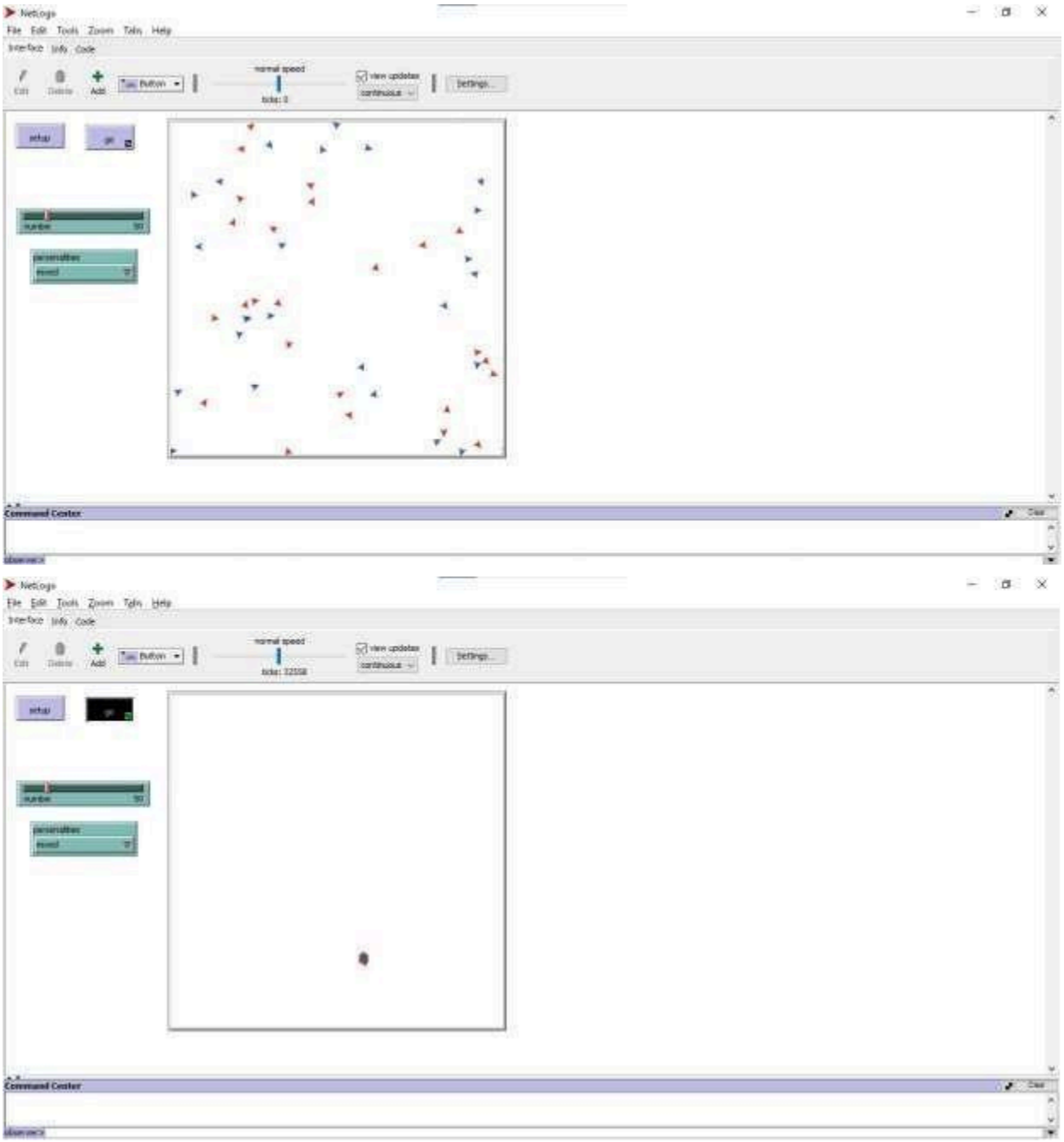


For Cowardly


NetLogo
File Edit Tools Zoom Tabs Help
Interface Info Code
Edit
Delete
Add
Button
normal speed
ticks: 0
view updates
continuous
Settings...
setup
go
number
50
personalities
cowards
Command Center
Clear
observer


NetLogo
File Edit Tools Zoom Tabs Help
Interface Info Code
Edit
Delete
Add
Button
normal speed
ticks: 23704
view updates
continuous
Settings...
setup
go
number
50
personalities
cowards
Command Center
Clear
observer

For Mixed



You will see:

- Blue turtles (brave) move toward conflict.
- Red turtles (cowardly) hide behind their friends.
- Mixed mode creates a dynamic blendof behaviors.

Additional Practice Commands

Try these after setup:

Highlight enemies: ask turtles [ask enemy [set pcolor yellow]] **Show friends in green:**

ask turtles[ask friend [set pcolor green]]

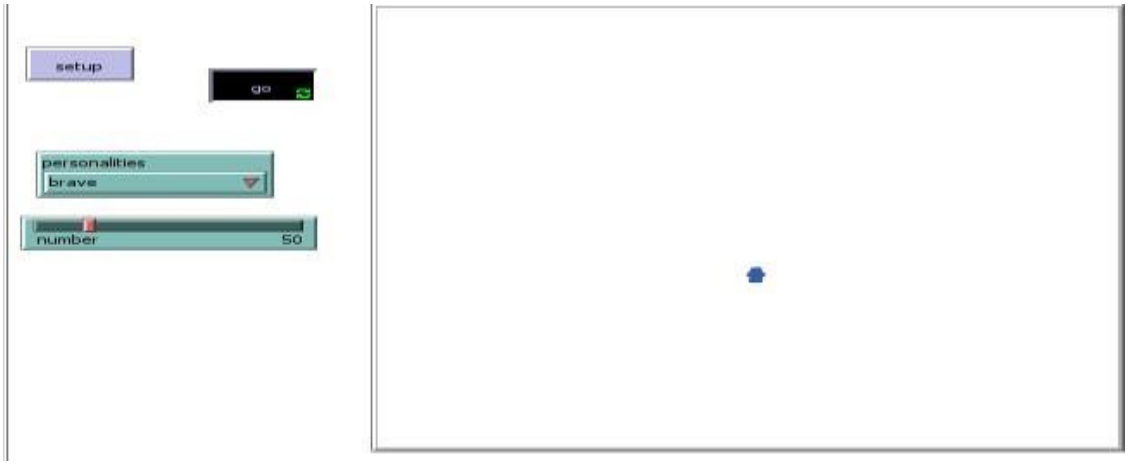
Slow motion: set speed 0.3

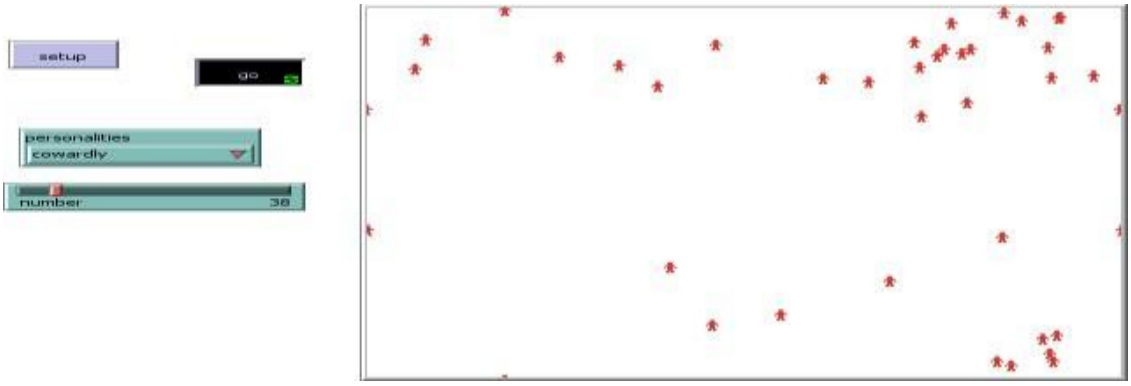
Observations

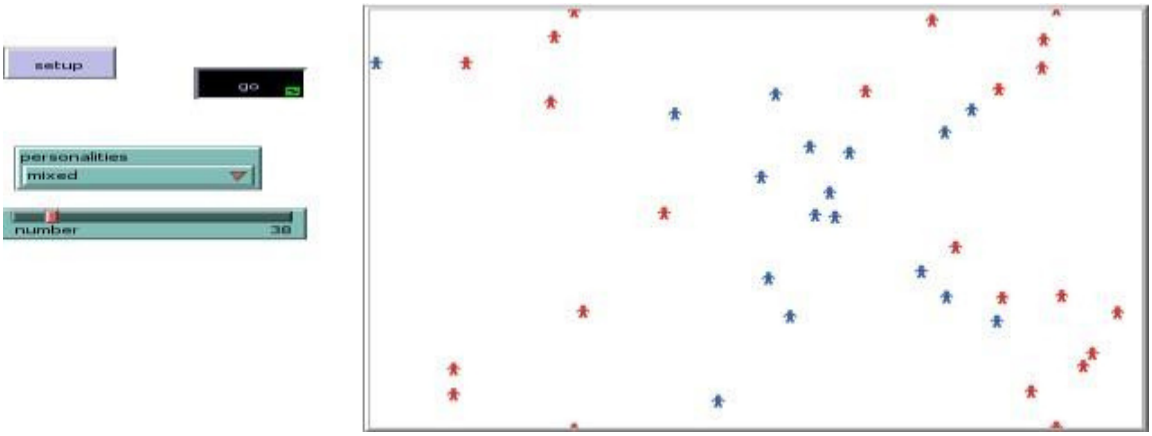
- Blue turtles tend tocluster in zones of conflict.
- Red turtles form small groups behind stronger turtles.
- Movement is smooth because turtles use continuous turning (facexy).
- Friend–enemy relationships create interesting social dynamics.

STUDENT TASK

1. Testbrave-only,cowardly-only, and mixed models.





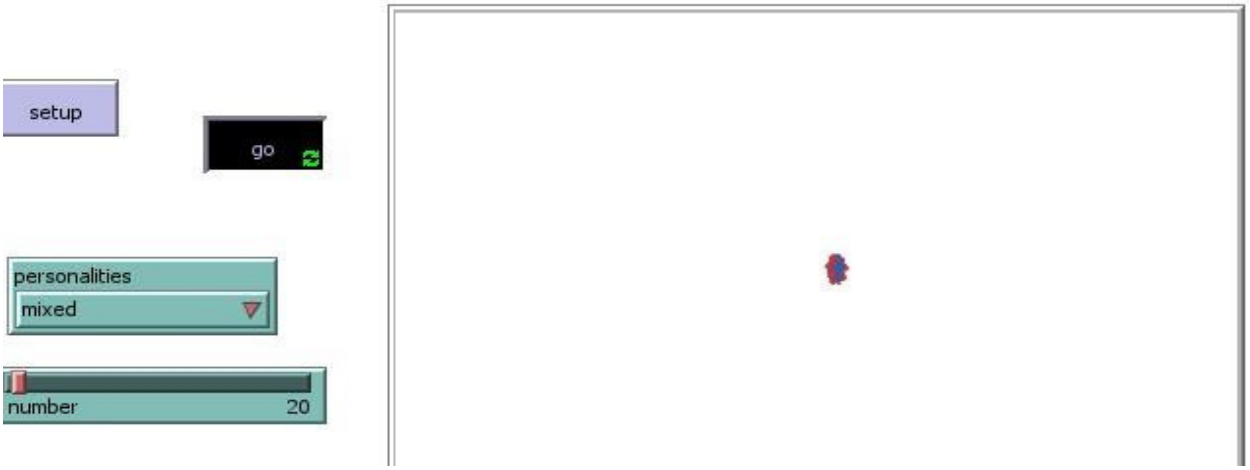


Observation

Brave-only: Allturtles move toward midpoints, leading to frequent clustering and interaction.

Cowardly-only: Turtles avoid direct encounters, spreading out more evenly.

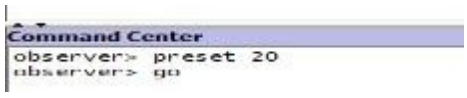
Mixed: A combination of clustering (brave) and avoidance (cowardly) creates dynamic patterns thataremore complex than single-type models.
2. Change turtle count and observe movement patterns.

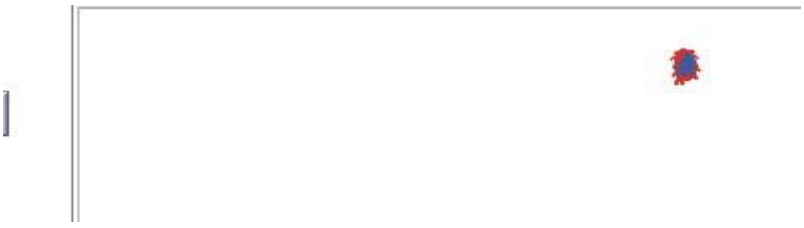


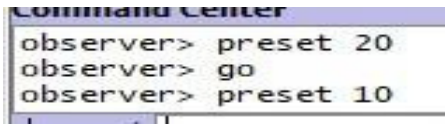
OBSERVATION

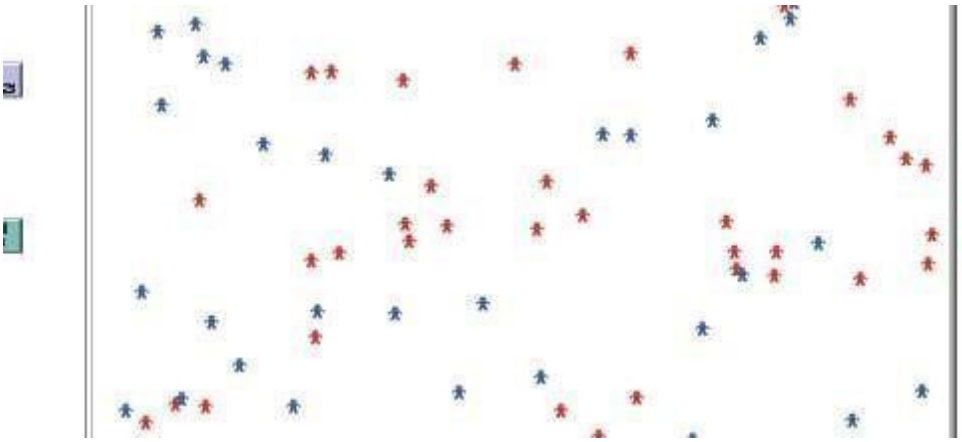
- **Low count (e.g., 20):** Turtles are spread out; interactions are fewer and clustering is minimal.
- **Medium count (e.g., 50):** More interactions occur; small clusters may form among brave turtles.
- **High count (e.g., 100):** Turtles are crowded; brave turtles form larger clusters, and cowardly turtles havemoreavoidance behavior.

3. Change random seeds using: preset 10, preset 20









Observation:

Changing therandom seed alters the initial positions of turtles and their friend/enemy assignments.With preset 10, turtles form certain clusters and movement patterns, while with preset 20, thedistribution and interactions differ. This shows that randomness affects the simulation outcomes, even though the overall behavior rules remain the same.

4. Note differences in clustering, spreading, and fleeing behavior.

The model shows clear differences in behavior based on personality and settings:

- **Clustering:** Brave turtles (blue) tend to move toward midpoints with friends and enemies, forming visible clusters.
- **Spreading:** Cowardly turtles (red) avoid enemies by positioning themselves between friends and enemies, resulting in more even spreading.
- **Fleeing:** Cowardly turtles actively move away from potential threats, while brave turtles approach interactions, showing contrasting movement strategies.
  These differences demonstrate how personality types influence spatial patterns and interactionsinthe model.

Post-Lab Questions

1. How do brave and cowardly behaviors differ in logic?
Braveturtlesmovetowardthemidpointof theirfriendandenemy, seeking interaction, while cowardly turtles position themselves between friend andenemy to avoid conflict.

2. Why does each turtle choose a friend and enemy?
Assigning a friendandenemy giveseachturtle referencepoints to determine movement and interaction strategies, creating dynamic behavior patterns.

3. How does facexy help turtles navigate?

The facexy command directs a turtle to face a specific coordinate, allowing precise movement toward or away from targets.

4. What role does the chooser play in this model?
The chooser lets the user select turtle personalities(brave, cowardly, mixed), determining the overall behavior dynamics of the simulation.

5. Why does random-seed create reproducible patterns?
Setting a random-seed ensures that all random choices (positions, friend/enemy selection) are the same each time, allowing experiments to be repeated withidentical outcomes.

Lab 10: Reward-Based Strategy Decision Model (El Farol Variant)

Objective

- To simulate how agents use prediction strategies to decide whether to attend a bar.
- To understand how rewards influence learning and color visualization.
- To record past attendance and use it for future decision-making.
- To implement strategy evaluation and selection based on performance.
- To modeladaptive behavior in a competitive environment.

Tool / Environment

- **Software:** NetLogo 6.4.0
- **Platform:** Windows
- **Environment:** Interface + Code Tab

Theory

This model is an extension of the **El Farol Bar Problem.**
Agents predict attendance using a "bag" of strategies.
Each strategy provides an attendance estimate based on a weighted memory of past attendance data.

Agents use this logic:

1. Predict attendance based on past history 2. Compare prediction

   with overcrowding threshold

   3. Decide:

      - Go to the bar ○

        Stay home

   4. Receive reward iftheyattend on a non-crowded day

   5. Learn to choose thestrategy that predicts best

   6. Turtle colorbecomesdarker red as reward increases

Key Components

Component Description

attendance numberofturtles currently in the bar
history last several weeksof attendance
strategies prediction formulas (weights)

reward increaseswhenagentattendsanon-crowded bar
best-strategy current most accurate strategy
crowded-patch patchdisplaying“CROWDED” message

Lab Procedure

Step 1: Add Required Interface Elements

Sliders

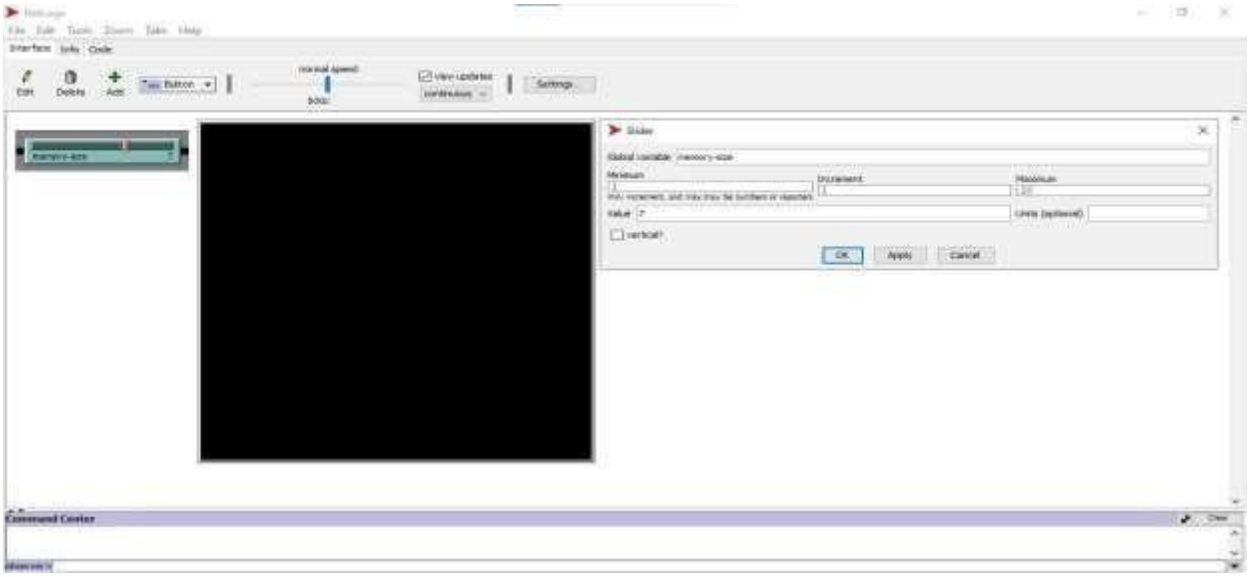
Add these sliders in Interface:

1. **memory-size** o

   Min: 1 o Max:

   10 o

   Initial: 7



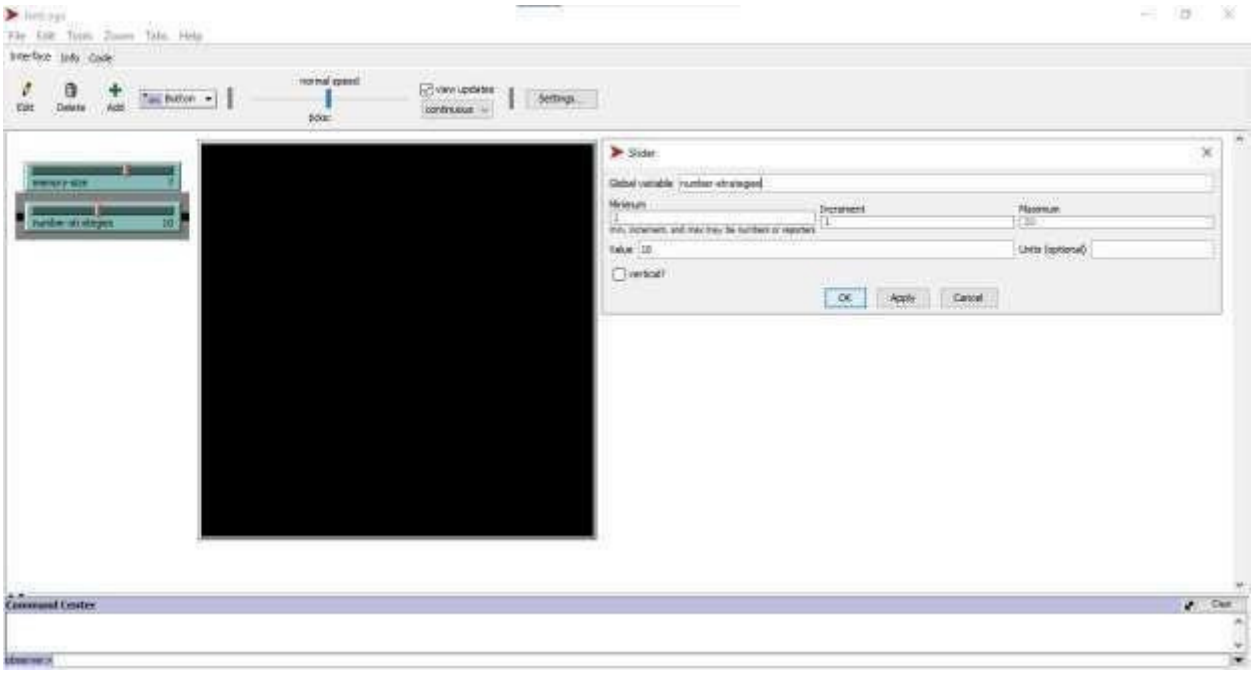
2. **number-**

   strategies o

   Min:

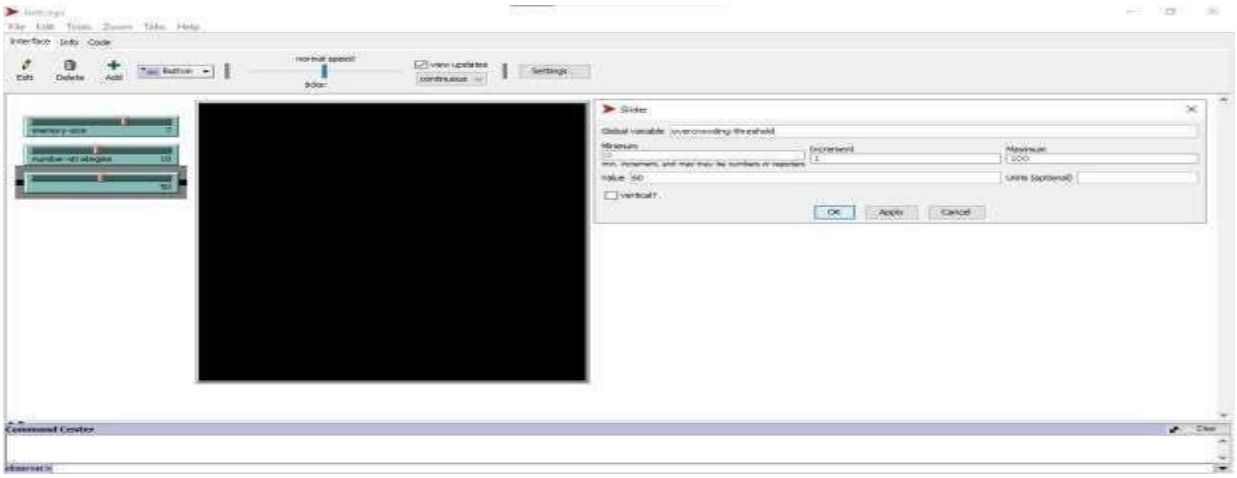
   1 o Max: 20

   o Initial: 10



3. **overcrowding-threshold**

   o Min: 0 o Max: 100 o Initial: 60

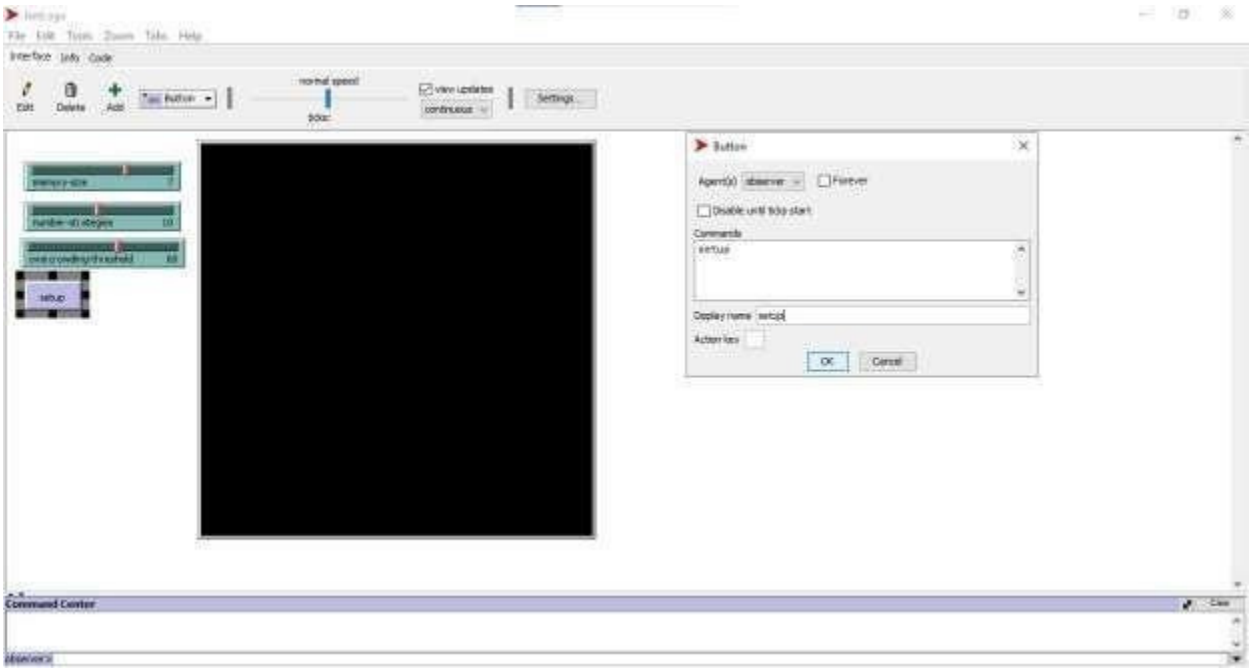


Buttons

1. **setup** ○

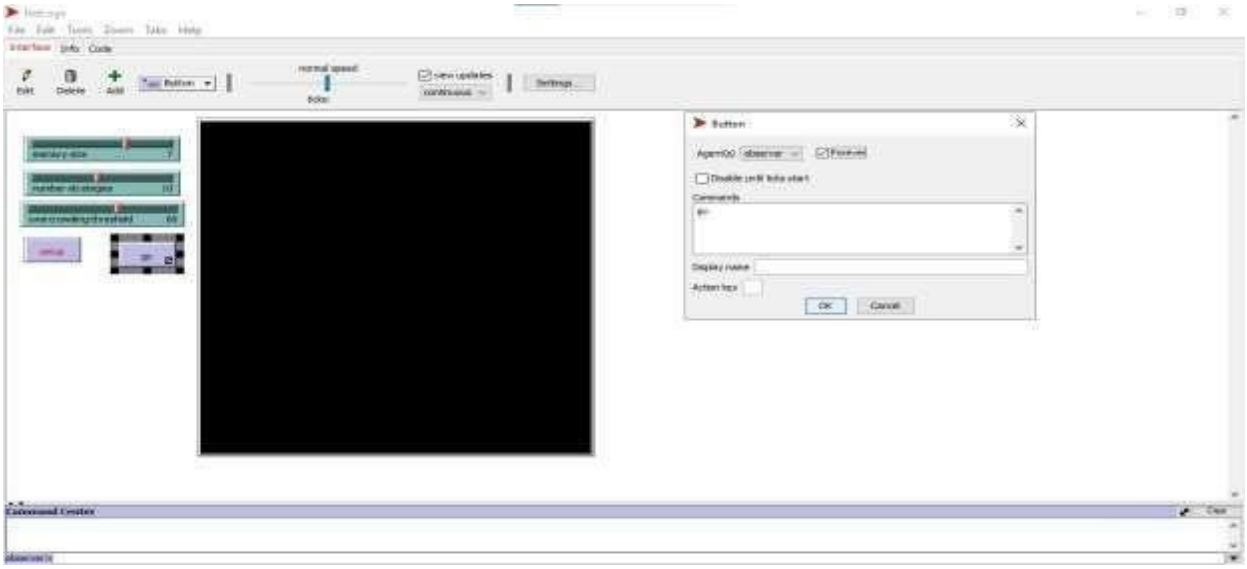
   Command: setup o

   Display name: setup



2. **go** ○ Command: go o

   Check: **Forever**



Add a Plot

- Right-click → **Plot**
- **Name:** Bar Attendance
- **X axis label:** Time
- **Y axis label:** Attendance

- **X min:** 0
- **X max:** 10
- **Y min:** 0
- **Y max:** 100

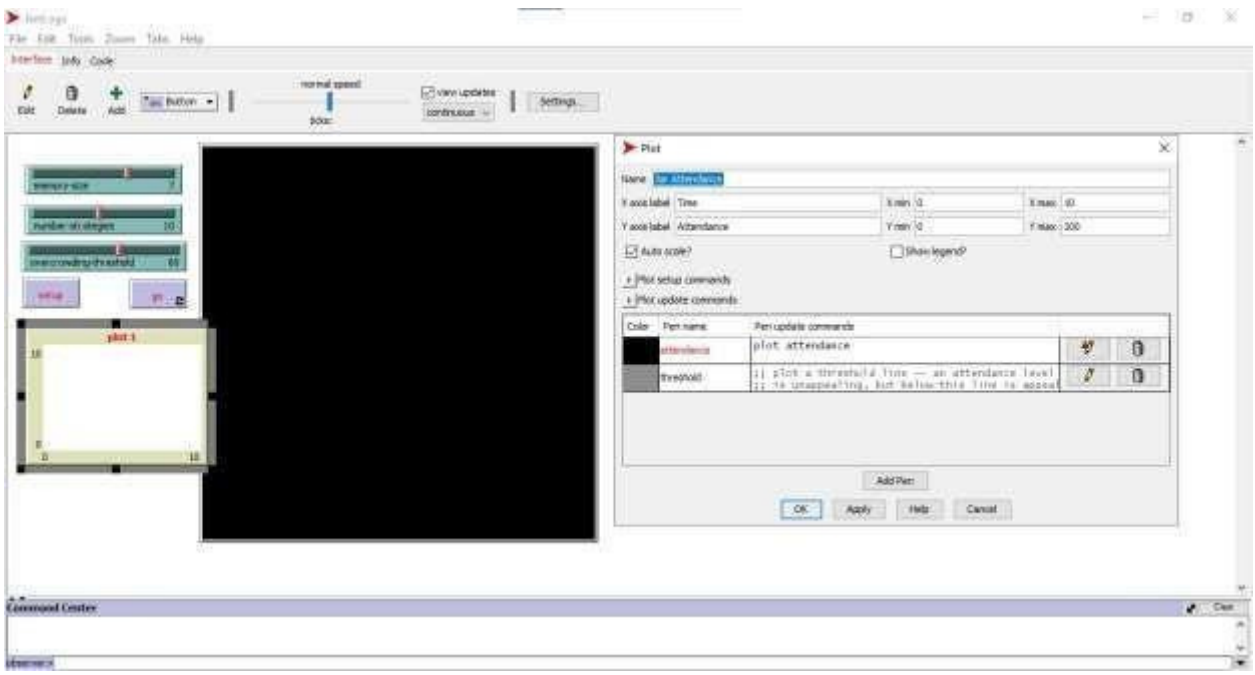
1. **Pen 1**
   - **Pen name:** attendance
   - **Pen update commands:** plot attendance

2. **Pen 2**
   - **Pen name:** threshold
   - **Pen update commands:**

;; plot a threshold line -- an attendance level above this line makes the bar

;; is unappealing, but below this line is appealing plot-pen-reset plotxy 0

overcrowding-threshold plotxy plot-x-max overcrowding-threshold



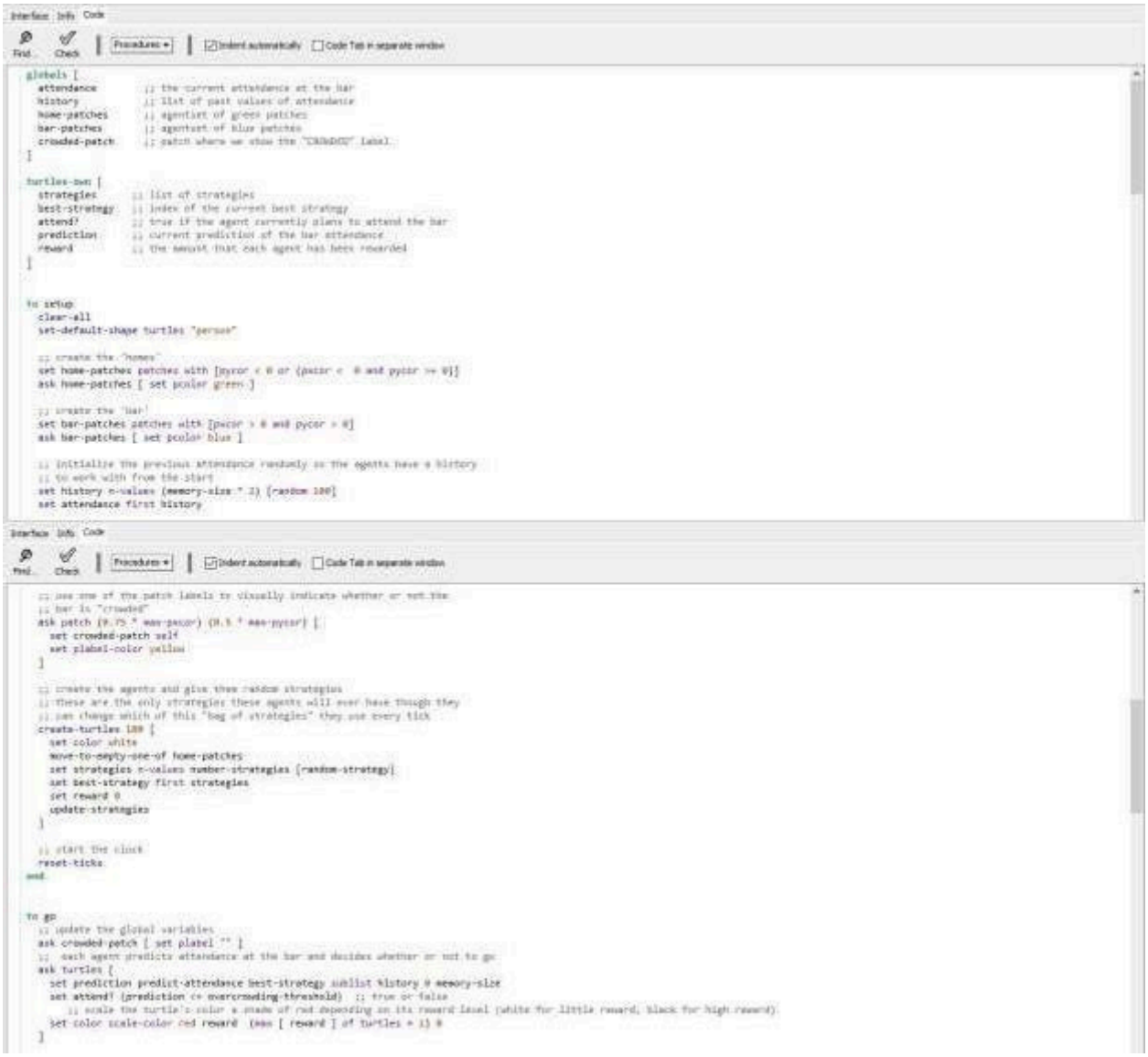
Step 2: Add the Full Code in the Code Tab

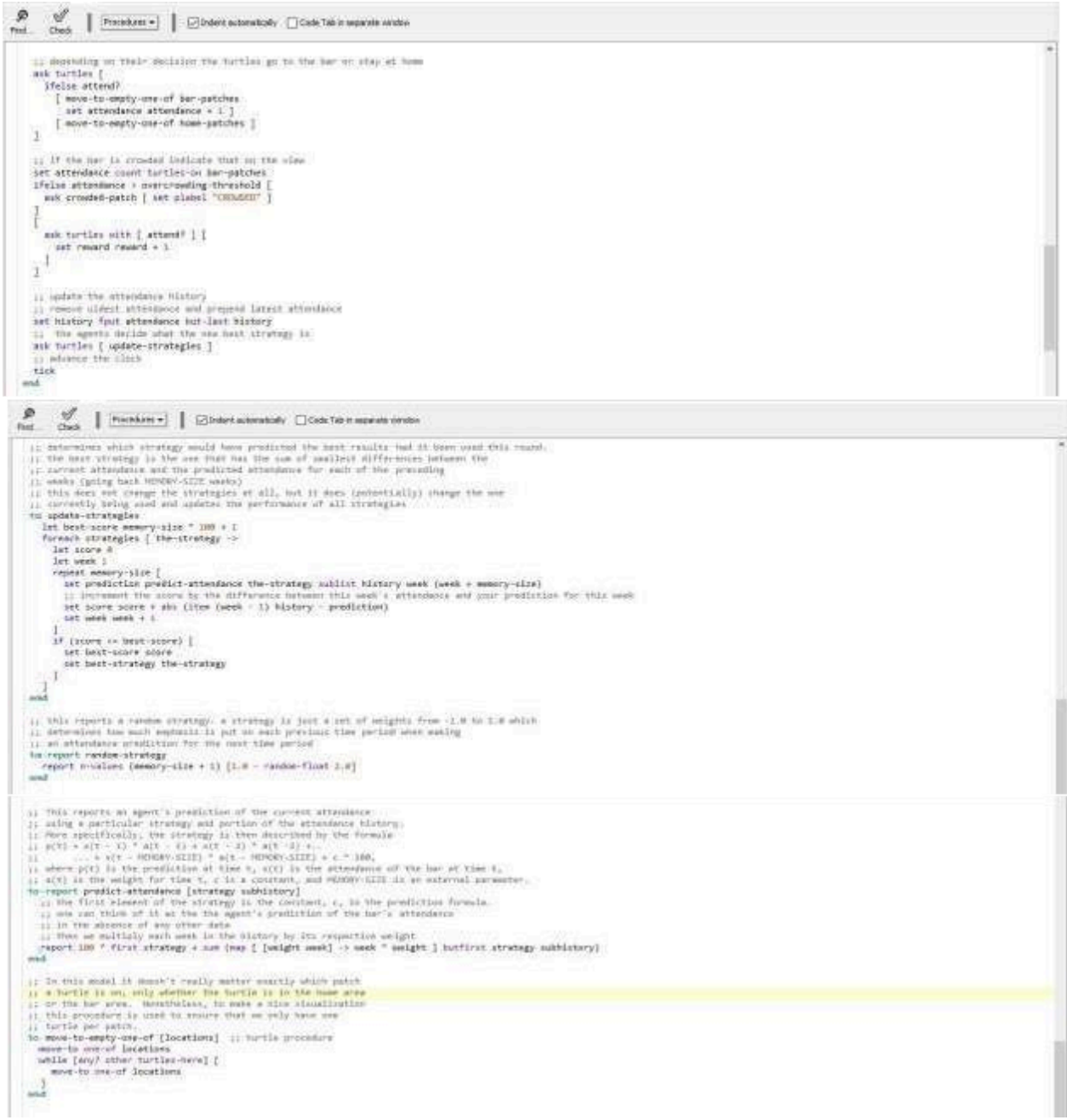
```
globals [ attendance
 ;; the current attendance at the bar history
;; list of past values of attendance home-patches
 ;; agentset of green patches bar-patches
 ;; agentset of blue patches crowded-patch
 ;; patch where we show the "CROWDED" label
]
turtles-own [
   strategies ;; list of strategies best-strategy ;; index of the current
     best strategy attend?       ;; true if the agent currently plans to
     attend the bar prediction              ;; current prediction of the bar
              attendance reward ;; the amount that each agent has been
                                                                  rewarded
]
to setup clear-all
     set-default-
     shape turtles
     "person"

  ;; create the 'homes' set home-patches patches with [pycor < 0 or (pxcor <
     0 and pycor >= 0)] ask home-patches [ set pcolor green ]
  ;; create the 'bar' set bar-patches patches with [pxcor > 0 and
     pycor > 0] ask bar-patches [ set pcolor blue ]

  ;; initialize the previous attendance randomly so the agents have a
  history ;; to work with from the start set history n-values (memory-size *
  2) [random 100] set attendance first history

  ;; use one of the patch labels to visually indicate whether or not
  the ;; bar is "crowded" ask patch (0.75 * max-pxcor) (0.5 * max-
  pycor) [         set crowded-patch self set plabel-color yellow
  ]

  ;; create the agents and give them random strategies
  ;; these are the only strategies these agents will ever have though they ;;
     can change which of this "bag of strategies" they use every tick create-
     turtles 100 [ set color white move-to-empty-one-of home- patches set
     strategies n-values number-strategies [random-strategy] set best-
     strategy first strategies set reward 0 update-strategies
  ]
```

```
  ;; startthe clock reset-ticks
     end


to go
  ;; update the global variables ask crowded-patch
     [ set plabel "" ]
  ;; each agent predicts attendance at the bar and decides whether or not to go ask
     turtles [      set prediction predict-attendance best-strategy sublist history 0
     memory-size set attend? (prediction <= overcrowding-threshold) ;; true or
     false
      ;; scale the turtle's color a shade of red depending on its reward level (white for little reward, black
     for high reward) set color scale-color red reward (max [ reward ] of turtles + 1) 0
  ]

  ;; depending on their decision the turtles go to the bar or stay at home
  ask turtles [ ifelse attend? [ move-to-empty-one-of bar-patches set
  attendance attendance + 1 ]
  [ move-to-empty-one-of home-patches ]
  ]

    ;; if the bar is crowded indicate that on the view
            set attendance count turtles-on bar-patches
       ifelse attendance > overcrowding-threshold [
       ask crowded-patch [ set plabel "CROWDED" ]
  ]
  [
    ask turtles with [ attend? ] [
      set reward reward + 1
    ]
  ]

  ;; update the attendance history
  ;; remove oldest attendance and prepend latest attendance
     set history fput attendance but-last history ;; the agents
     decide what the new best strategy is ask turtles [ update-
     strategies ]
  ;; advance the clock tick end

;; determines which strategy would have predicted the best results had it been used this round.
;; the best strategy is the one that has the sum of smallest differences between the
;; current attendance and the predicted attendance for each of the preceding
```

```
;; weeks (going back MEMORY-SIZE weeks)
;; this does not change the strategies at all, but it does (potentially) change the one
;; currently being used and updates the performance of all strategies to update-strategies let best-
    score memory-size * 100 + 1 foreach strategies [ the-strategy ->          let score 0       let week
    1 repeat memory-size [     set prediction predict-attendance the-strategy sublist history week
    (week + memory-size)      ;; increment the score by the difference between this week's
    attendance and your prediction for this week      set score score + abs (item (week - 1) history
    - prediction)         set week week + 1
  ]
  if (score <= best-score) [       set
    best-score score    set best- strategy
    the-strategy
  ]
 ]
end
;; this reports a random strategy. a strategy is just a set of weights from -1.0 to 1.0 which
;; determines how much emphasis is put on each previous time period when making
;; an attendance prediction for the next time period to-report
random-strategy report n-values (memory-size + 1) [1.0 -
random-float 2.0] end

;; This reports an agent's prediction of the current attendance
;; using a particular strategy and portion of the attendance history. ;;
More specifically, the strategy is then described by the formula ;;
p(t) = x(t - 1) * a(t - 1) + x(t - 2) * a(t -2) +..
;;     ... + x(t - MEMORY-SIZE) * a(t - MEMORY-SIZE) + c * 100,
;; where p(t) is the prediction at time t, x(t) is the attendance of the bar at time t,
;; a(t) is the weight for time t, c is a constant, and MEMORY-SIZE is an external paramete to-report
    predict-attendance [strategy subhistory]
 ;; the first element of the strategy is the constant, c, in the prediction formula.
 ;; one can think of it as the the agent's prediction of the bar's attendance ;;
 in the absence of any other data
 ;; then we multiply each week in the history by its respective weight report 100 * first strategy
    + sum (map [ [weight week] -> week * weight ] butfirst strategy subhistory) end

;; In this model it doesn't really matter exactly which patch
;; a turtle is on, only whether the turtle is in the home area
;; or the bar area. Nonetheless, to make a nice visualization ;;
this procedure is used to ensure that we only have one ;;
turtle per patch.
```

to move-to-empty-one-of [locations] ;; turtle procedure
move-to one-of locations while [any? other turtles-here] [move-to one-of locations
]
end





```
;; depending on their decision the turtles go to the bar or stay at home
ask turtles [
  ifelse attend?
    [ move-to-empty-one-of bar-patches
      set attendance attendance + 1 ]
    [ move-to-empty-one-of home-patches ]
]

;; if the bar is crowded indicate that on the view
set attendance count turtles-on bar-patches
ifelse attendance > overcrowding-threshold [
  ask crowded-patch [ set plabel "CROWDED" ]
]
[
  ask turtles with [ attend? ] [
    set reward reward + 1
  ]
]

;; update the attendance history
;; remove oldest attendance and prepend latest attendance
set history fput attendance but-last history
;;  the agents decide what the new best strategy is
ask turtles [ update-strategies ]
;; advance the clock
tick
end
```

```
;; determines which strategy would have predicted the best results had it been used this round.
;; the best strategy is the one that has the sum of smallest differences between the
;; current attendance and the predicted attendance for each of the preceding
;; weeks (going back MEMORY-SIZE weeks)
;; this does not change the strategies at all, but it does (potentially) change the one
;; currently being used and updates the performance of all strategies
to update-strategies
  let best-score memory-size * 100 + 1
  foreach strategies [ the-strategy ->
    let score 0
    let week 1
    repeat memory-size [
      set prediction predict-attendance the-strategy sublist history week (week + memory-size)
      ;; increment the score by the difference between this week's attendance and your prediction for this week
      set score score + abs (item (week - 1) history - prediction)
      set week week + 1
    ]
    if (score <= best-score) [
      set best-score score
      set best-strategy the-strategy
    ]
  ]
end

;; this reports a random strategy. a strategy is just a set of weights from -1.0 to 1.0 which
;; determines how much emphasis is put on each previous time period when making
;; an attendance prediction for the next time period
to-report random-strategy
  report n-values (memory-size + 1) [1.0 - random-float 2.0]
end
```

```
;; this reports an agent's prediction of the current attendance
;; using a particular strategy and portion of the attendance history.
;; More specifically, the strategy is then described by the formula
;; p(T) = a(T - 1) * a(t - 1) + a(T - 2) * a(t -2) + ...
;;        ... + a(T - MEMORY-SIZE) * a(t - MEMORY-SIZE) + c * 100,
;; where p(t) is the prediction at time t, a(t) is the attendance of the bar at time t,
;; a(t) is the weight for time t, c is a constant, and MEMORY-SIZE is an external parameter.
to-report predict-attendance [strategy subhistory]
  ;; the first element of the strategy is the constant, c, in the prediction formula.
  ;; one can think of it as the the agent's prediction of the bar's attendance
  ;; in the absence of any other data
  ;; then we multiply each week in the history by its respective weight
  report 100 * first strategy + sum (map [ [weight week] -> week * weight ] butfirst strategy subhistory)
end

;; In this model it doesn't really matter exactly which patch
;; a turtle is on, only whether the turtle is in the home area
;; or the bar area.  Nonetheless, to make a nice visualization
;; this procedure is used to ensure that we only have one
;; turtle per patch.
to move-to-empty-one-of [locations]  ;; turtle procedure
  move-to one-of locations
  while [any? other turtles-here] [
    move-to one-of locations
  ]
end
```

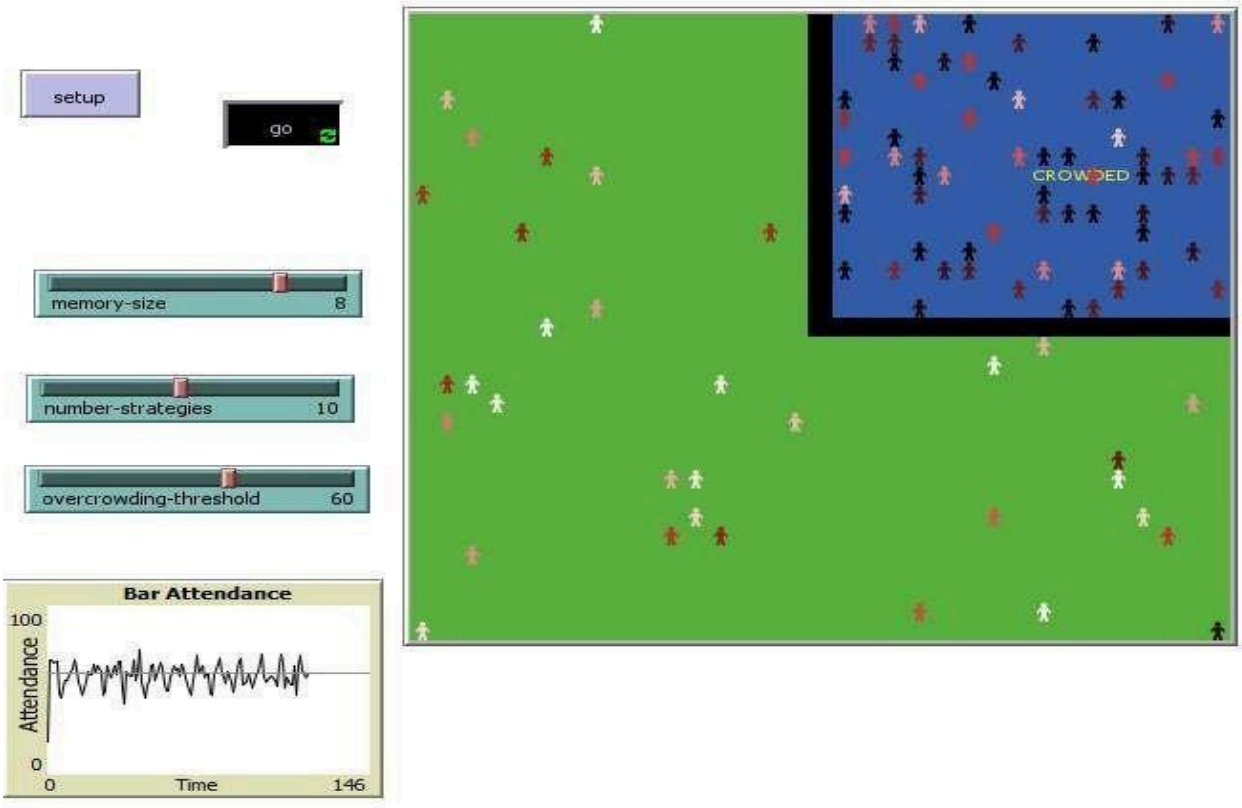
Output:

Observations

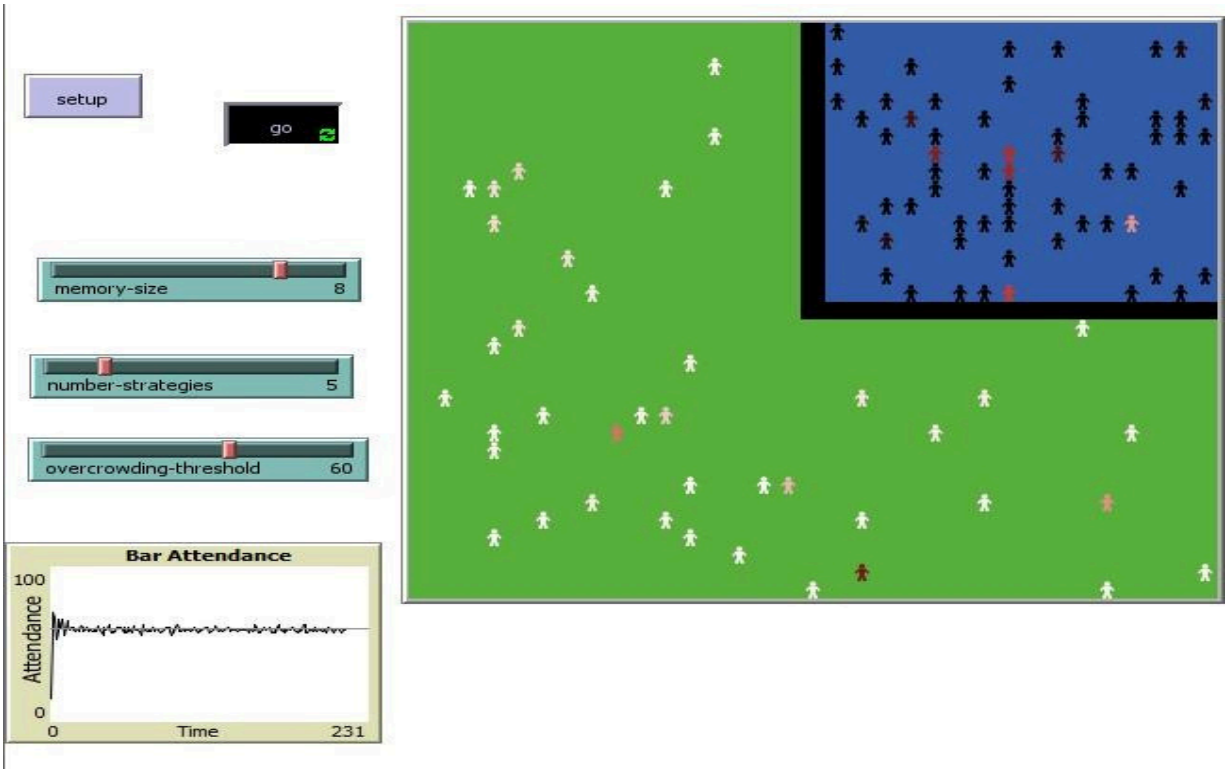
- Turtles turn darker red as their reward increases.
- Attendance oscillates based on predictions.
- “CROWDED” appears when attendance > threshold.
- Strategies improve over time as predictions become more accurate.
- Thebarrarely stabilizes because agents compete to outguess each other.

STUDENT TASK

1. Increasememory-size and note changes in prediction stability.



OBSERVATION With memory-sizeset to 8, agents use the past 8 weeks of attendance data to make predictions. This results in relatively stable and consistent predictions of bar attendance compared to smaller memory sizes. Fluctuations from tick to tick are reduced, and agents make more informed decisions about whether to attend the bar, leading to smoother attendance patterns over time. **2.** Decrease number-strategies and observe if agents perform worse.



Observation:

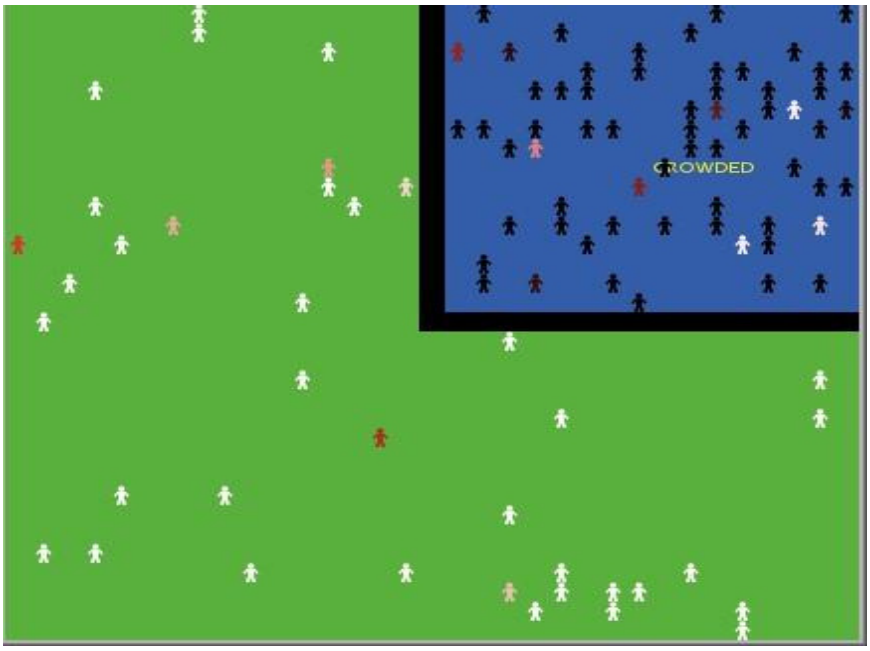
With the number of strategies reduced to 5, each agent has fewer options to choose from when predicting attendance. As a result, agents perform slightly worse compared to having more strategies, because they have a smaller pool of strategies to find the most accurate prediction. This leads to less optimal decisions and more fluctuations in attendance at the bar.

3. Run the model multiple times and note fluctuations in crowded days.

Running the model multiple times shows that the number of crowded days fluctuates across simulations. Even with the same parameters, random variations in agents' initial positions, strategies, and predictions lead to differences in which days the bar becomes crowded. This

demonstrates the role of randomness in the model and how small differences in initial conditions can affect outcomes.

4. Compare reward levels between regularly attending vs avoiding agents.



Observation:

Agents who regularly attend the bar when it is not crowded tend to accumulate higher rewards, as their predictions allow them to avoid overcrowding and gain points consistently. In contrast, agents who often avoid the bar or attend when it is crowded earn lower rewards. This demonstrates that successful prediction and decision-making directly influence an agent's reward level.

5. Modify overcrowding-threshold and study behavior shifts.

Observation:

Changingtheovercrowding-thresholdalters agents' behavior. Increasing the threshold allows moreagents to attend the barbefore it is considered crowded, resulting in higher attendanceandfewer agents avoidingthe bar. Decreasing the threshold makes the bar

crowded more quickly, causing more agents to stay home. This demonstrates how the overcrowding-threshold directly influences decision-making and overall attendance patterns.

Post-Lab Questions

1. Why do agents need multiple strategies instead of just one?

Multiple strategies give agents options to predict attendance moreaccurately. With only one strategy, agents may consistently make poor predictions if that strategy is not suitable for the current conditions.

2. How does reward influence future predictions?

Higher rewards indicate successful predictions. Agents tend to favor strategies that have performed well in the past, improving the accuracy offuture predictions.

3. What happens when memory-size is too small or too large?

A small memory-size leads to unstable predictions because agentsrely on limited historical data. A very large memory-size can slow adaptation, as agents weigh olddata too heavily, reducing responsiveness to recent changes.

4.Whydoesthe modeldisplay"CROWDED"?

The "CROWDED" label appears when the bar'sattendance exceeds the overcrowding-threshold, signaling that too many agents are attending andsome may need to avoid the bar to maximize reward.

5. How does the visualization help understand reward accumulation?

Color changes of turtles (white to black) represent reward levels, allowingquick visual comparison of agents who attend successfully versus those who avoid or fail. This helpsunderstand patterns of reward accumulation over time.

Lab 11: Strategy-Based Decision Making (El Farol Bar Model – Extension)

Objective

- To study decision-making under limited information
- To understand how agents, use prediction strategies
- To analyzeovercrowding behavior in shared resources
- To observereward-based learning using historical data

Tool / Environment

- **Software:** NetLogo 6.4.0

- **Platform:** Windows
- **Environment:** Interface + Code Tab

Theory

- ElFarol is a popularbarin Santa Fe that becomes unpleasant when it is too crowded. Each person must decidewhether to go or stay home without knowing how many others will attend. In this model, agentsdonot have perfect information. They use past attendance data
- and simple predictionstrategies to guess how crowded the bar will be. This is called bounded rationality becausedecisions are based on limited experience, not perfect calculation. Each agent has severalprediction strategies and checks which one worked best
- in the past. The most accuratestrategy is used to make the current decision. An agent goes to the bar only if the predictedattendance is below a fixed overcrowding limit. Over time, agentslearnand adapt their behavior. As a result, attendance keeps changing instead of
- settlingatonefixed value. This model shows how complex group behavior can emerge from simpleindividual decision rules, as explained by Brian Arthur (1994).

Key Components

Component Description attendance

numberofturtlescurrently in the bar **history** last

severalweeksofattendance **strategies**

prediction formulas (weights)

reward increases when agent attends a non-crowded bar
best-strategy currentmostaccurate strategy

crowded-patch patch displaying “CROWDED” message

Lab Procedure

Step 1: Add Required Interface Elements

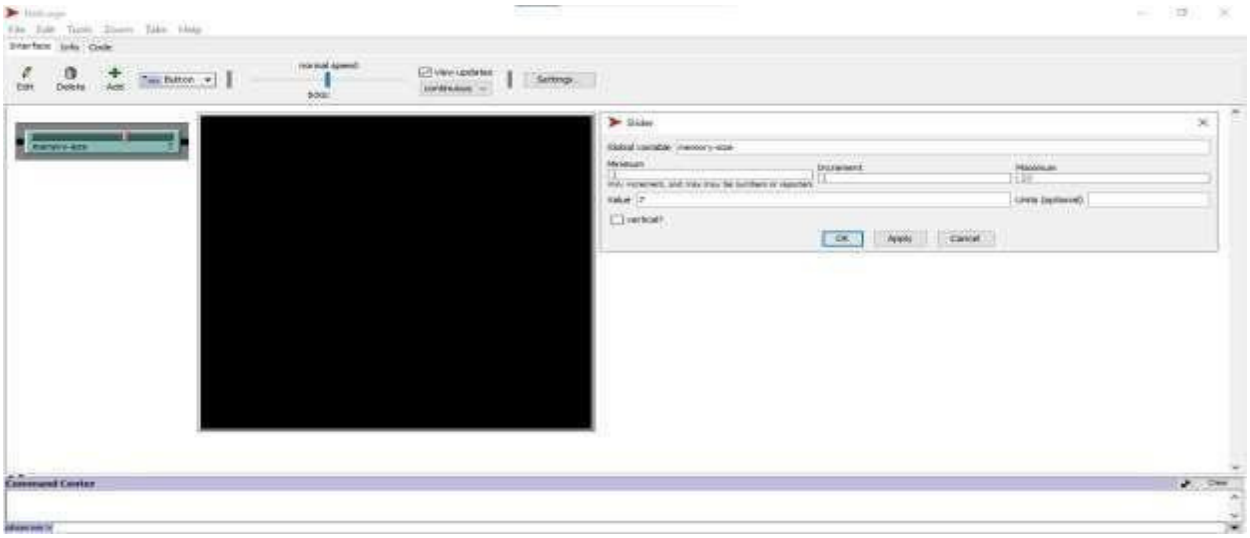
Sliders

Add these sliders in Interface:

1. **memory-size** o

Min: 1 o Max: 10 o

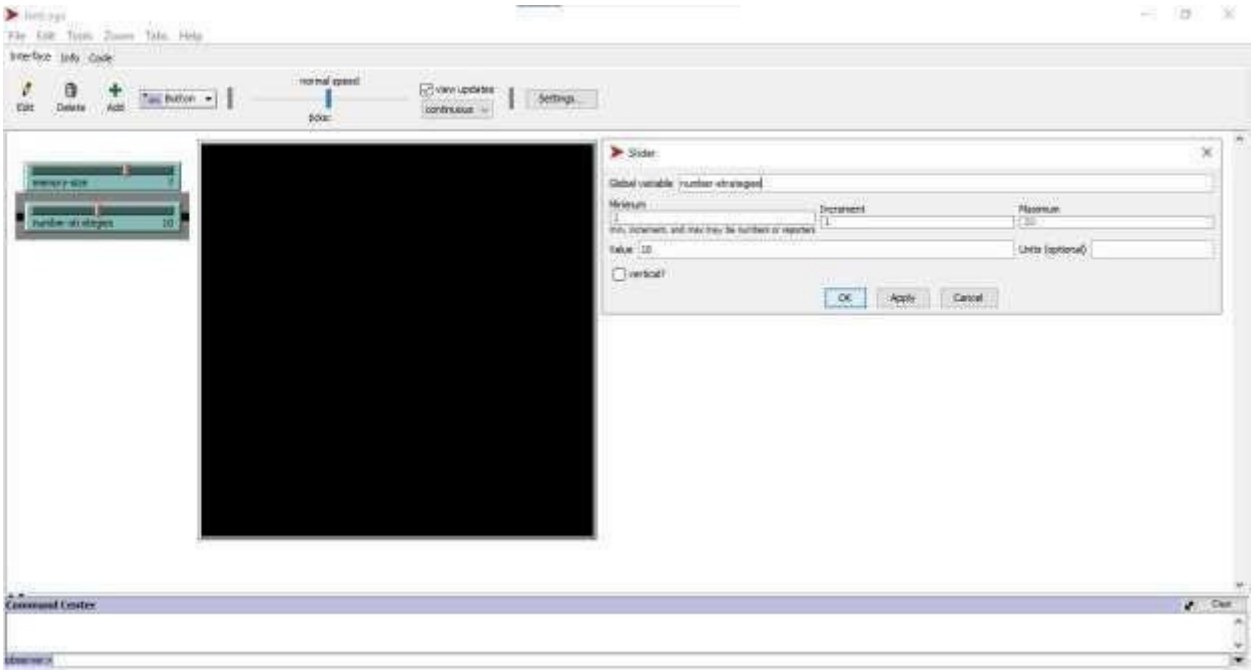
Initial: 7



2. **number-strategies** o Min:

   1 o Max: 20

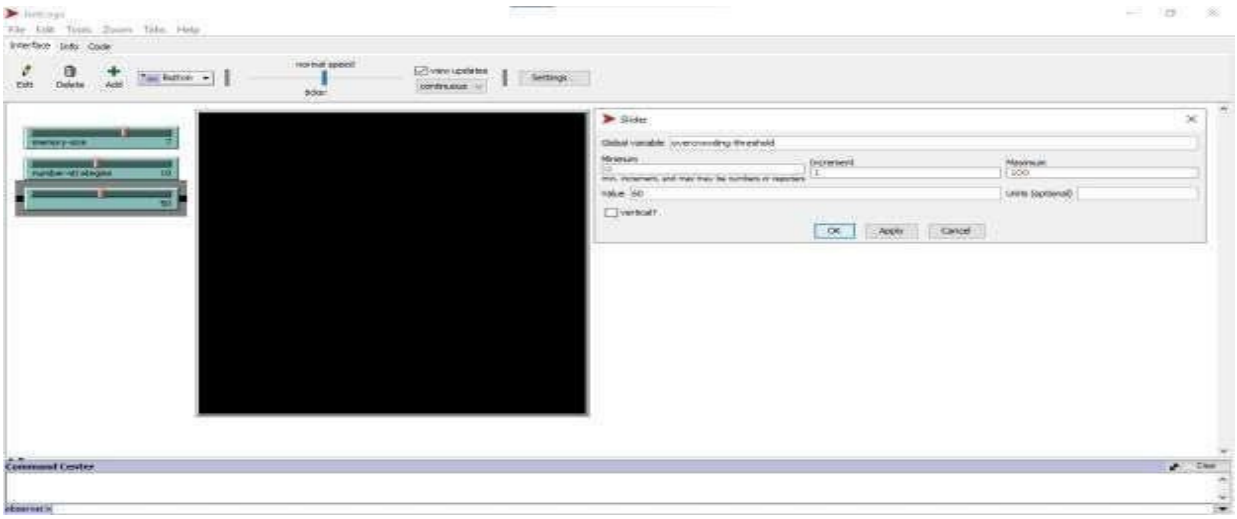
   o Initial: 10



3. **overcrowding-threshold** o Min:

   0 o Max: 100 o

   Initial: 60

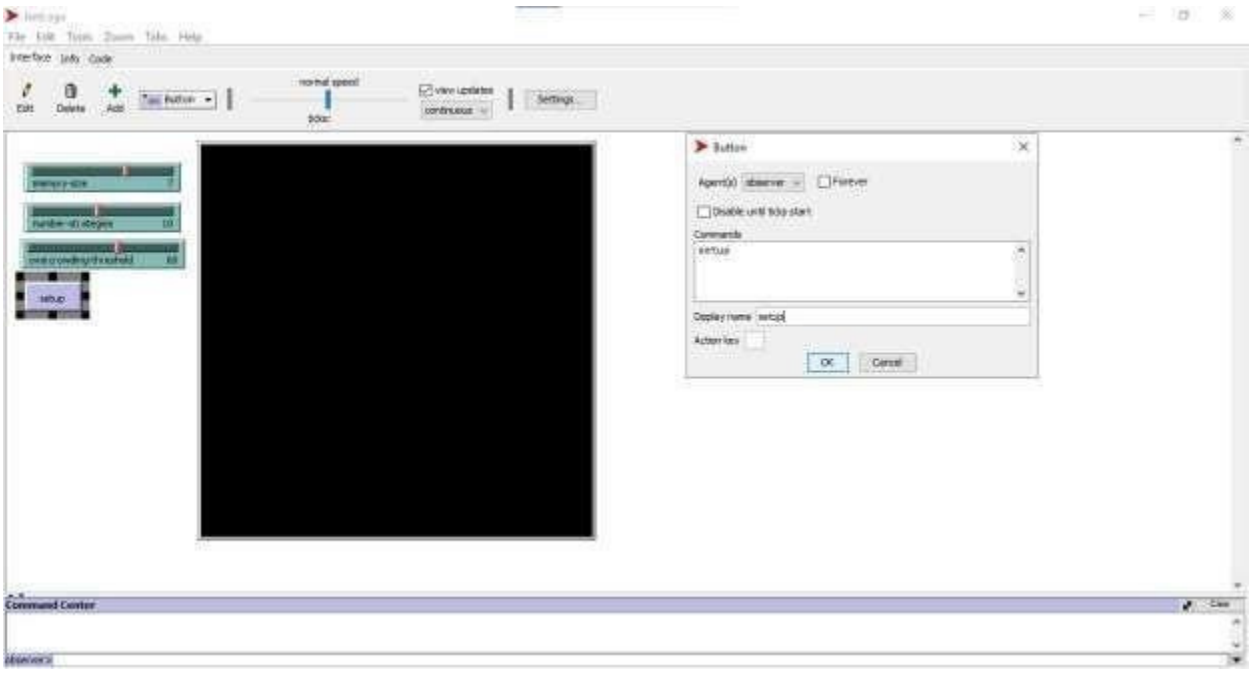


Buttons

1. **setup** o

   Command: setup o

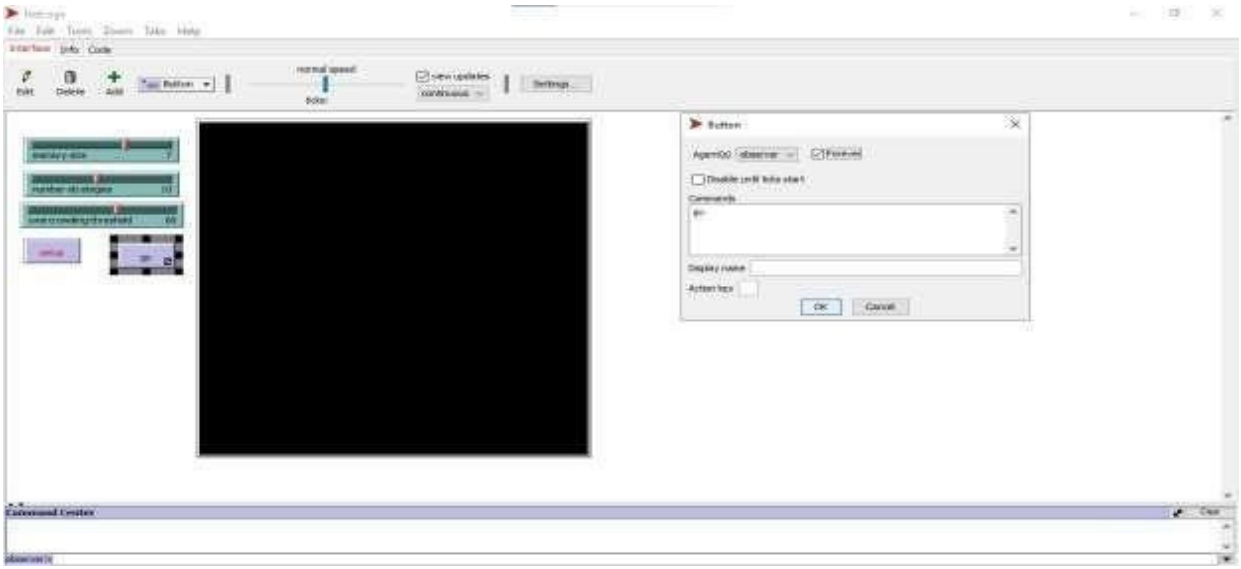
   Display name: setup



2. **go** o Command:

   go

   o Check: **Forever**



Add a Plot

- Right-click → **Plot**
- **Name:** Bar Attendance
- **X axis label:** Time
- **Y axis label:** Attendance
- **X min:** 0
- **X max:** 10
- **Y min:** 0

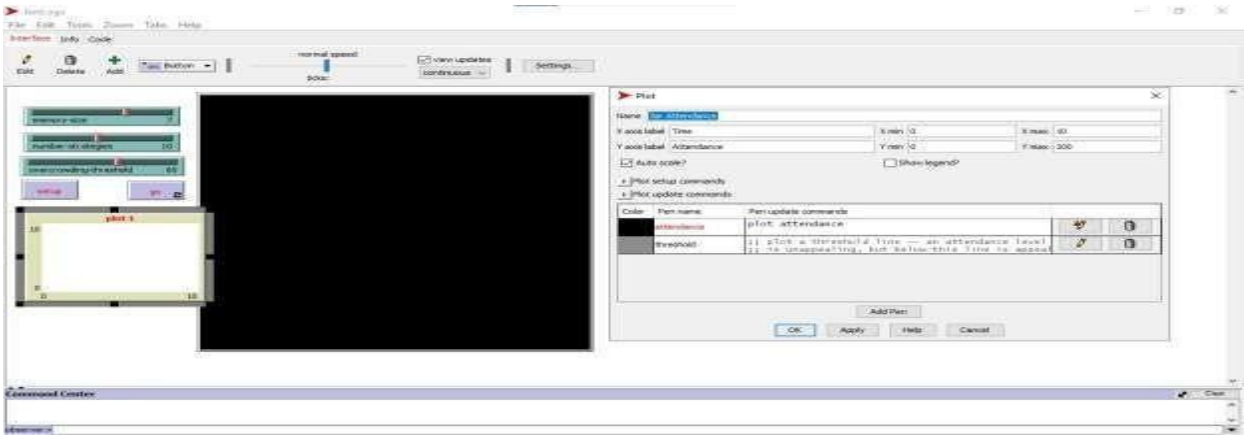
  Y max: 100

1. **Pen 1**
   - **Pen name:** attendance
   - **Pen update commands:** plot attendance
2. **Pen 2**
   - **Pen name:** threshold
   - **Pen update commands:**

;; plot a threshold line -- an attendance level above this line makes the bar

;; is unappealing, but below this line is appealing plot-pen-reset plotxy 0

overcrowding-threshold plotxy plot-x-max overcrowding-threshold



Add a Plot

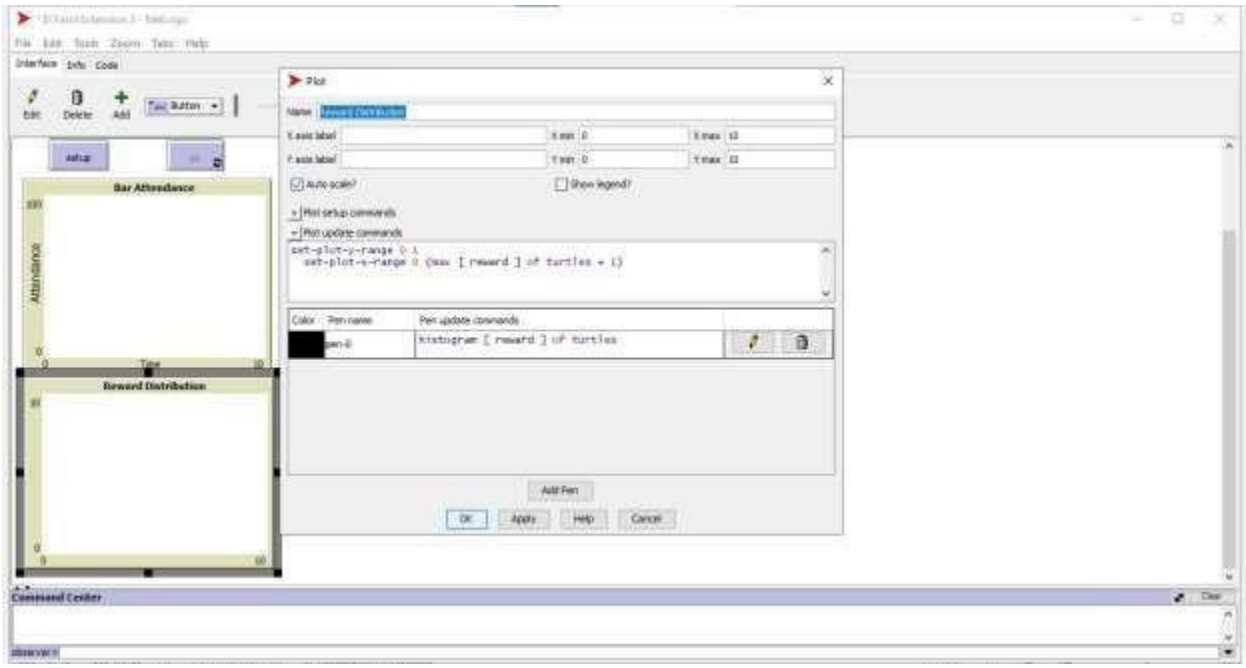
- Right-click → **Plot**
- **Name:** RewardDistribution
- **X min:** 0
- **X max:** 10 • **Y min:** 0
- **Y max:** 10

1. Plot update commands:

set-plot-y-range 0 1 set-plot-x-range 0 (max [reward] of turtles + 1)

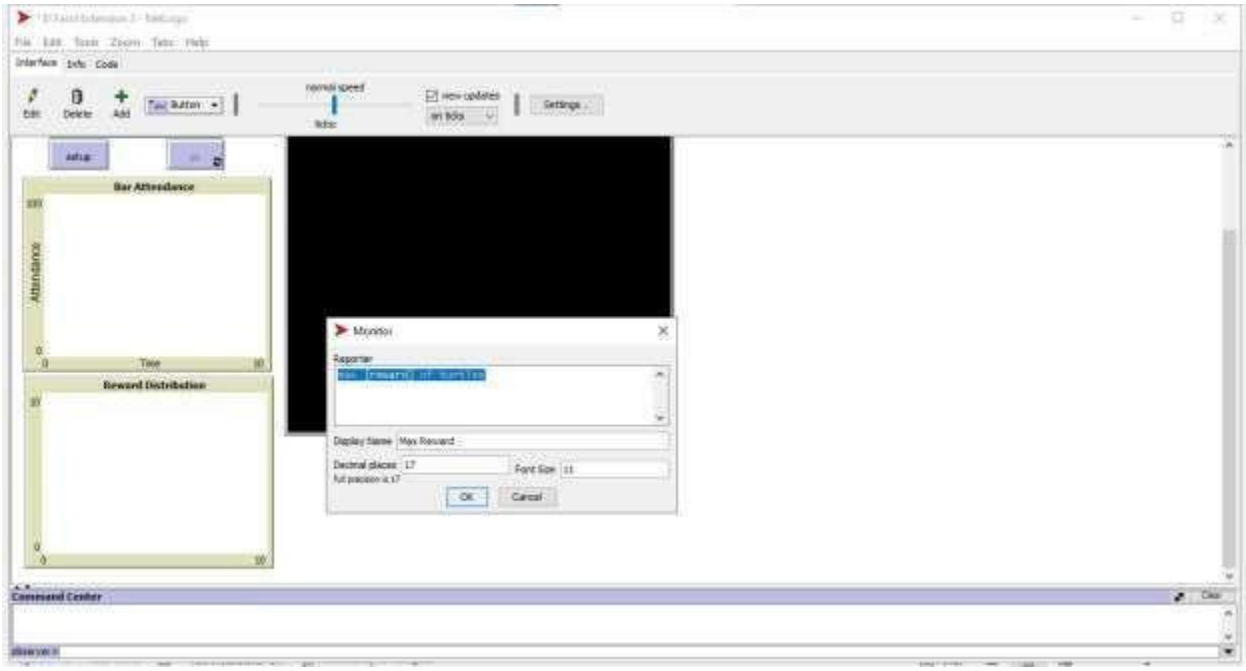
2. Pen

- **Pen name:** pen-0
- **Pen update commands:** histogram [reward] of turtles



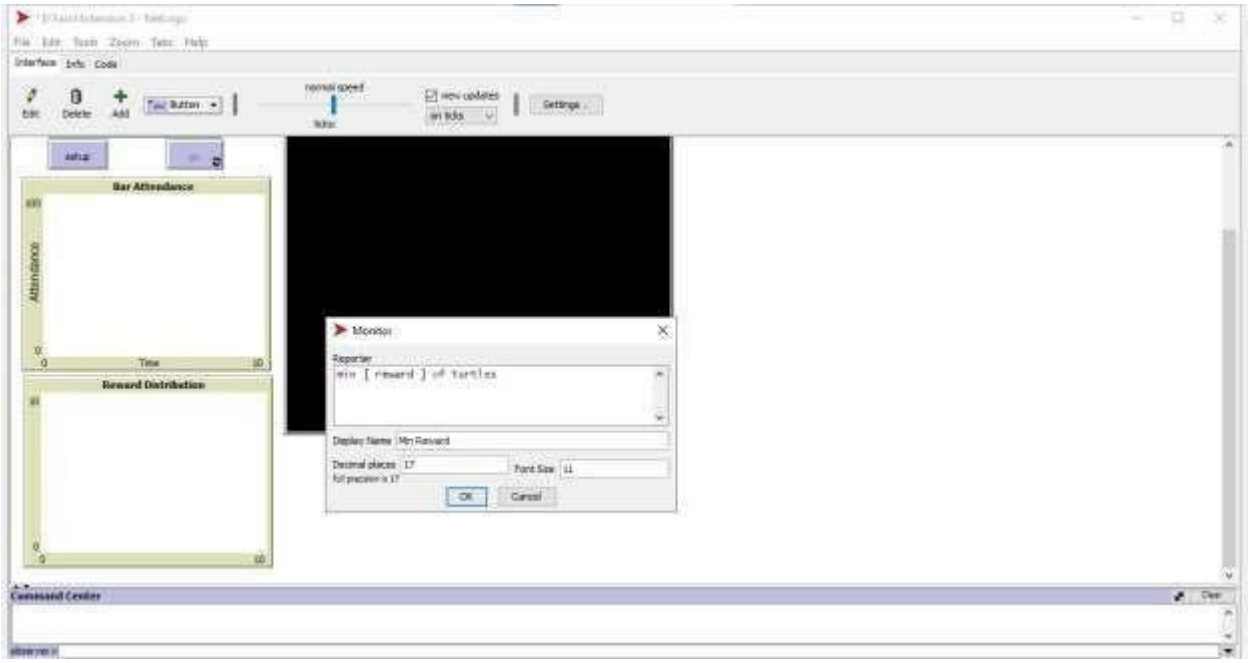
Monitor: Max Reward

- **Type:** Monitor
- **Display Name:** Max Reward
- **Reporter:** max[reward] of turtles
- **Decimal places:** 17
- **Font size:** 11



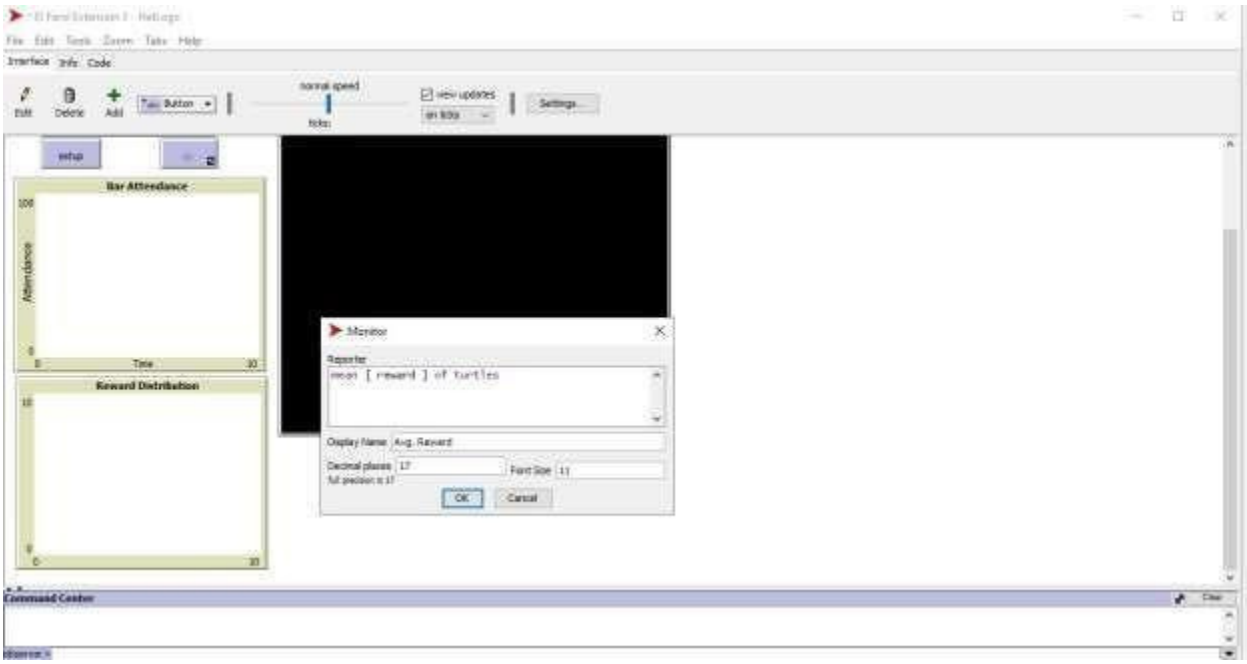
Monitor: Min Reward

- **Type:** Monitor
- **Display Name:** Min Reward
- **Reporter:** max[reward] of turtles
- **Decimal places:** 17
- **Font size:** 11



Monitor: Average Reward

- **Type:** Monitor
- **Display Name:** Avg. Reward
- **Reporter:** max[reward] of turtles
- **Decimal places:** 17
- **Font size:** 11



Step 2: Add the Full Code in the Code Tab

```
globals [attendance ;; the current attendanceat the bar history ;; list ofpast
    values of attendance home-patches ;; agentset of green patches bar-
    patches ;; agentset of bluepatches crowded-patch ;; patch where we
    show the "CROWDED"label
]

turtles-own [
   strategies ;; list of strategies best-strategy ;; index of the current
    best strategy attend?          ;; true if the agent currently plans to
    attend the bar prediction              ;; current prediction of the bar
          attendance reward ;; the amount that each agent has been
                                                          rewarded
]


    to setup     clear-all          set-default-
    shape turtles "person"

 ;; create the 'homes' set home-patches patches with [pycor < 0 or (pxcor < 0
    and pycor >= 0)] ask home-patches [ set pcolor green ]

 ;; create the 'bar' set bar-patches patches with [pxcor > 0 and
    pycor > 0] ask bar-patches [ set pcolor blue ]
```

```
  ;; initialize the previous attendance randomly so the agents have a history
  ;; to work with from the start set history n-values
  (memory-size * 2) [random 100] set attendance first
  history

  ;; use one of the patch labels to visually indicate whether or not the
  ;; bar is "crowded" ask patch (0.75 * max-pxcor)
  (0.5 * max-pycor) [ set crowded-patch self set
  plabel-color yellow
  ]

;; create the agents and give them random strategies create-
      turtles 100 [ set color white move-to-empty-one-of home-
      patches              set strategies n-values number-strategies
      [random-strategy] set     best-strategy     first      strategies
           set reward 0     update-strategies
  ]

  ;; startthe clock reset-ticks
      end


to go
  ;; update the global variables ask crowded-patch
      [ set plabel "" ]
  ;; have each agent predict attendance at the bar and decide whether or not to go
      ask turtles [           set prediction predict-attendance best-strategy sublist history
      0 memory-size set attend? (prediction <= overcrowding-threshold) ;; true or
      false
       ;; scale the turtle's color a shade of red depending on its reward level (white for little reward, black
      for high reward) set color scale-color red reward (max [ reward ] of turtles + 1) 0
  ]

  ;; depending on their decision the turtles go to the bar or stay at home
  ask turtles [ ifelse attend? [ move-to-empty-one-of bar-patches set
  attendance attendance + 1 ]
  [ move-to-empty-one-of home-patches ]
  ]

    ;; if the bar is crowded indicate that on the view
            set attendance count turtles-on bar-patches
```

```
    ifelse attendance > overcrowding-threshold [
    ask crowded-patch [ set plabel "CROWDED"
  ]  ]
  [
   ask turtles with [ attend? ] [
    set reward reward + 1
   ]
  ]

  ;; update the attendance history set history fput
     attendance but-last history
  ;; have the agents decide what the new best strategy is ask turtles
     [ update-strategies ]
   ;; update the plots
  ;; my-update-plots

  ;; advance the clock tick end

;; determines which strategy would have predicted the best results had it been used this round.
;; the best strategy is the one that has the sum of smallest difference between the
;; current attendance and the predicted attendance for each of the preceding
;; weeks (going back MEMORY-SIZE weeks) to update-strategies let best-score memory-size * 100
     + 1 foreach strategies [ the-strategy -> let score 0 let week 1 repeat memory-size
     [ set prediction predict-attendance the-strategy sublist history week (week + memory-size) ;;
     increment the score by the difference between this week's attendance and your prediction for
     this week set score score + abs (item (week - 1) history - prediction)
     set week week + 1
    ]
    if (score <= best-score) [       set
     best-score score     set best- strategy
     the-strategy
    ]
  ]
end

;; thisreports a random strategy. a strategy is just a set of weights from -1.0 to 1.0 which
;; determines how much emphasis is put on each previous time period when making
;; anattendance prediction for the next time period to-report
random-strategy report n-values (memory-size + 1) [1.0 -
random-float 2.0] end

;; This reports an agent's prediction of the current attendance
```

;; using a particular strategy and portion of the attendance history. ;;
More specifically, the strategy is then described by the formula ;; p(t)
= x(t - 1) * a(t - 1) + x(t - 2) * a(t -2) +..
;; ... + x(t - MEMORY-SIZE) * a(t - MEMORY-SIZE) + c * 100,
;; where p(t) is the prediction at time t, x(t) is the attendance of the bar at time t,
;; a(t) is the weight for time t, c is a constant, and MEMORY-SIZE is an external paramete**to-report**
predict-attendance [strategy subhistory]
;; the first element of the strategy is the constant, c, in the prediction formula.
;; one can think of it as the the agent's prediction of the bar's attendance
;; in the absence of any other data
;; then we multiply each week in the history by its respective weight report 100 * first strategy
+ sum (map [[weight week] -> week * weight] butfirst strategy subhistory) **end**

;; In this model it doesn't really matter exactly which patch
;; a turtle is on, only whether the turtle is in the home area
;; or the bar area. Nonetheless, to make a nice visualization ;;
this procedure is used to ensure that we only have one ;;
turtle per patch.
to move-to-empty-one-of [locations] ;; turtle procedure
move-to one-of locations while [any? other turtles-
here] [
move-to one-of locations
]
End

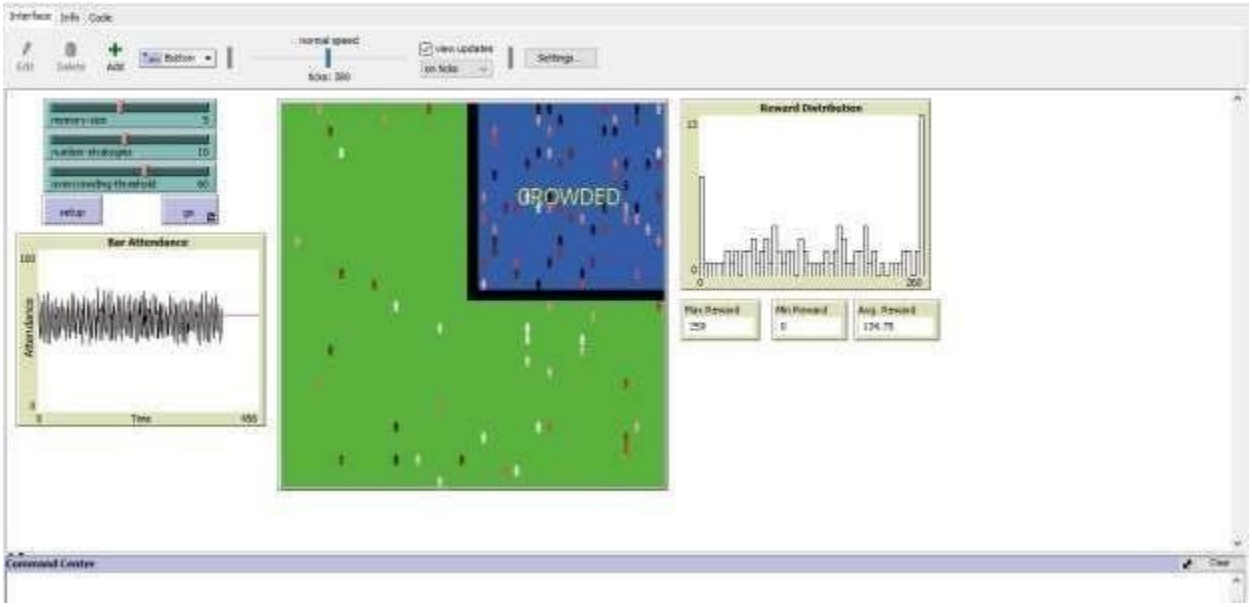

```
  ;; use one of the patch labels to visually indicate whether or not the
  ;; bar is "crowded"
  ask patch (0.75 * max-pxcor) (0.5 * max-pycor) [
    set crowded-patch self
    set plabel-color yellow
  ]

;; create the agents and give them random strategies
  create-turtles 100 [
    set color white
    move-to-empty-one-of home-patches
    set strategies n-values number-strategies [random-strategy]
    set best-strategy first strategies
    set reward 0
    update-strategies
  ]

  ;; start the clock
  reset-ticks
end


to go
  ;; update the global variables
  ask crowded-patch [ set plabel "" ]
  ;; have each agent predict attendance at the bar and decide whether or not to go
  ask turtles [
    set prediction predict-attendance best-strategy sublist history 0 memory-size
    set attend? (prediction <= overcrowding-threshold)  ;; true or false
      ;; scale the turtle's color a shade of red depending on its reward level (white for little reward, black for high reward)
    set color scale-color red reward  (max [ reward ] of turtles + 1) 0
  ]
```

```
  ;; depending on their decision the turtles go to the bar or stay at home
  ask turtles [
    ifelse attend?
      [ move-to-empty-one-of bar-patches
        set attendance attendance + 1 ]
      [ move-to-empty-one-of home-patches ]
  ]

  ;; if the bar is crowded indicate that on the view
  set attendance count turtles-on bar-patches
  ifelse attendance > overcrowding-threshold [
    ask crowded-patch [ set plabel "CROWDED" ]
  ]
  [
    ask turtles with [ attend? ] [
      set reward reward + 1
    ]
  ]

  ;; update the attendance history
  set history fput attendance but-last history
  ;; have the agents decide what the new best strategy is
  ask turtles [ update-strategies ]

  ;; update the plots
  ;; my-update-plots

  ;; advance the clock
  tick
end
```

```
;; determines which strategy would have predicted the best results had it been used this round.
;; the best strategy is the one that has the sum of smallest difference between the
;; current attendance and the predicted attendance for each of the preceding
;; weeks (going back MEMORY-SIZE weeks)
to update-strategies
  let best-score memory-size * 100 + 1
  foreach strategies [ the-strategy ->
    let score 0
    let week 1
    repeat memory-size [
      set prediction predict-attendance the-strategy sublist history week (week + memory-size)
      ;; increment the score by the difference between this week's attendance and your prediction for this week
      set score score + abs (item (week - 1) history - prediction)
      set week week + 1
    ]
    if (score <= best-score) [
      set best-score score
      set best-strategy the-strategy
    ]
  ]
end

;; this reports a random strategy. a strategy is just a set of weights from -1.0 to 1.0 which
;; determines how much emphasis is put on each previous time period when making
;; an attendance prediction for the next time period
to-report random-strategy
  report n-values (memory-size + 1) [1.0 - random-float 2.0]
end
```

```
;; This reports an agent's prediction of the current attendance
;; using a particular strategy and portion of the attendance history.
;; More specifically, the strategy is then described by the formula
;; p(t) = a(t - 1) * a(t - 1) + a(t - 2) * a(t -2) +...
;;      ... + a(t - MEMORY-SIZE) * a(t - MEMORY-SIZE) + c * 100,
;; where p(t) is the prediction at time t, a(t) is the attendance of the bar at time t,
;; a(t) is the weight for time t, c is a constant, and MEMORY-SIZE is an external parameter.
to-report predict-attendance [strategy subhistory]
  ;; the first element of the strategy is the constant, c, in the prediction formula.
  ;; one can think of it as the the agent's prediction of the bar's attendance
  ;; in the absence of any other data
  ;; then we multiply each week in the history by its respective weight
  report 100 * first strategy + sum (map [ [weight week] -> week * weight ] butfirst strategy subhistory)
end

;; In this model it doesn't really matter exactly which patch
;; a turtle is on, only whether the turtle is in the home area
;; or the bar area.  Nonetheless, to make a nice visualization
;; this procedure is used to ensure that we only have one
;; turtle per patch.
to move-to-empty-one-of [locations]  ;; turtle procedure
  move-to one-of locations
  while [any? other turtles-here] [
    move-to one-of locations
  ]
end
```

Output:

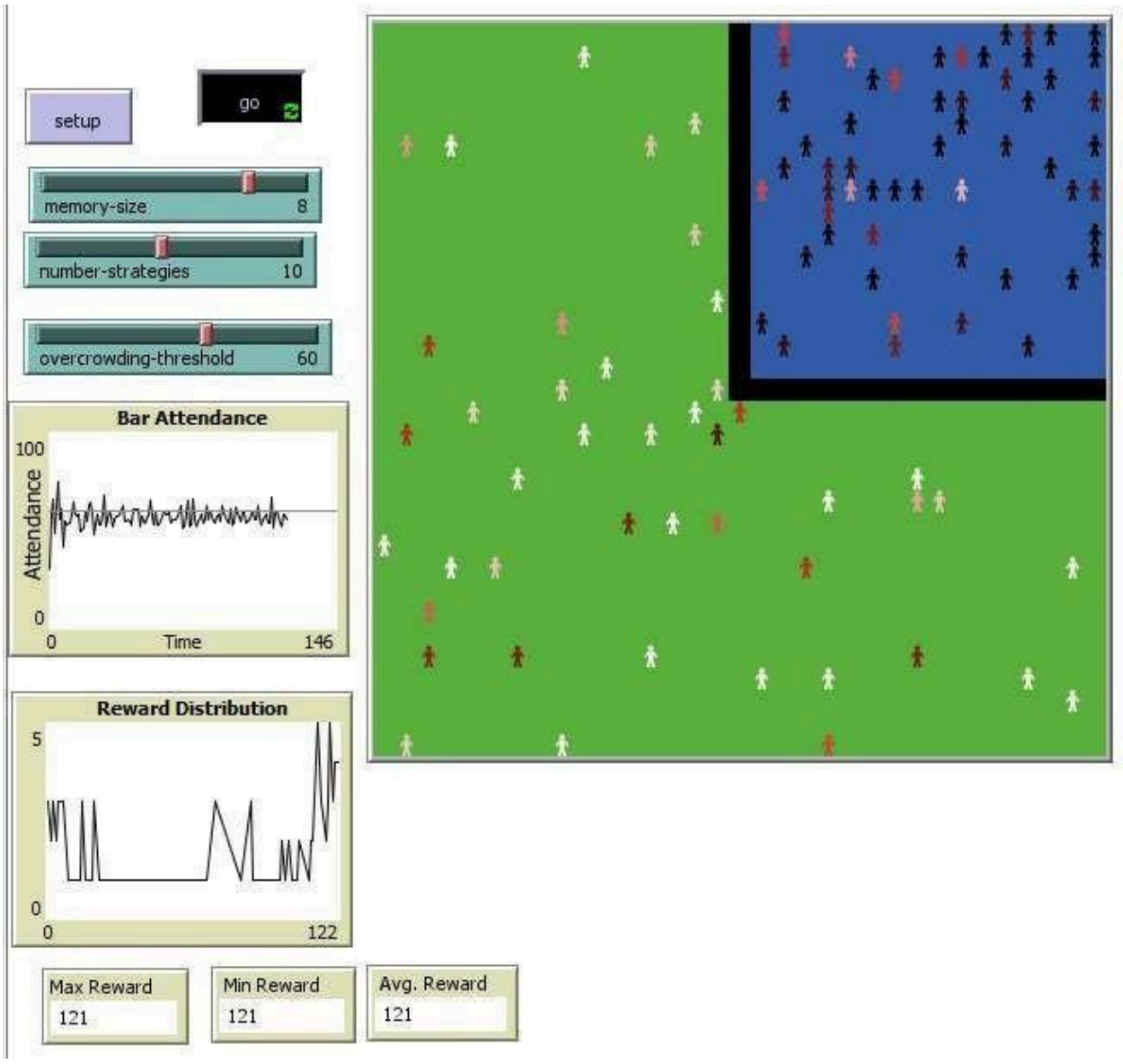


Observations

- Attendance fluctuates rather than stabilizing
  - Agents continuously switch strategies
- Some agents earn consistently higher rewards
- Overcrowding emerges without coordination

STUDENT TASK

1. IncreaseMEMORY-SIZE and observe prediction stability

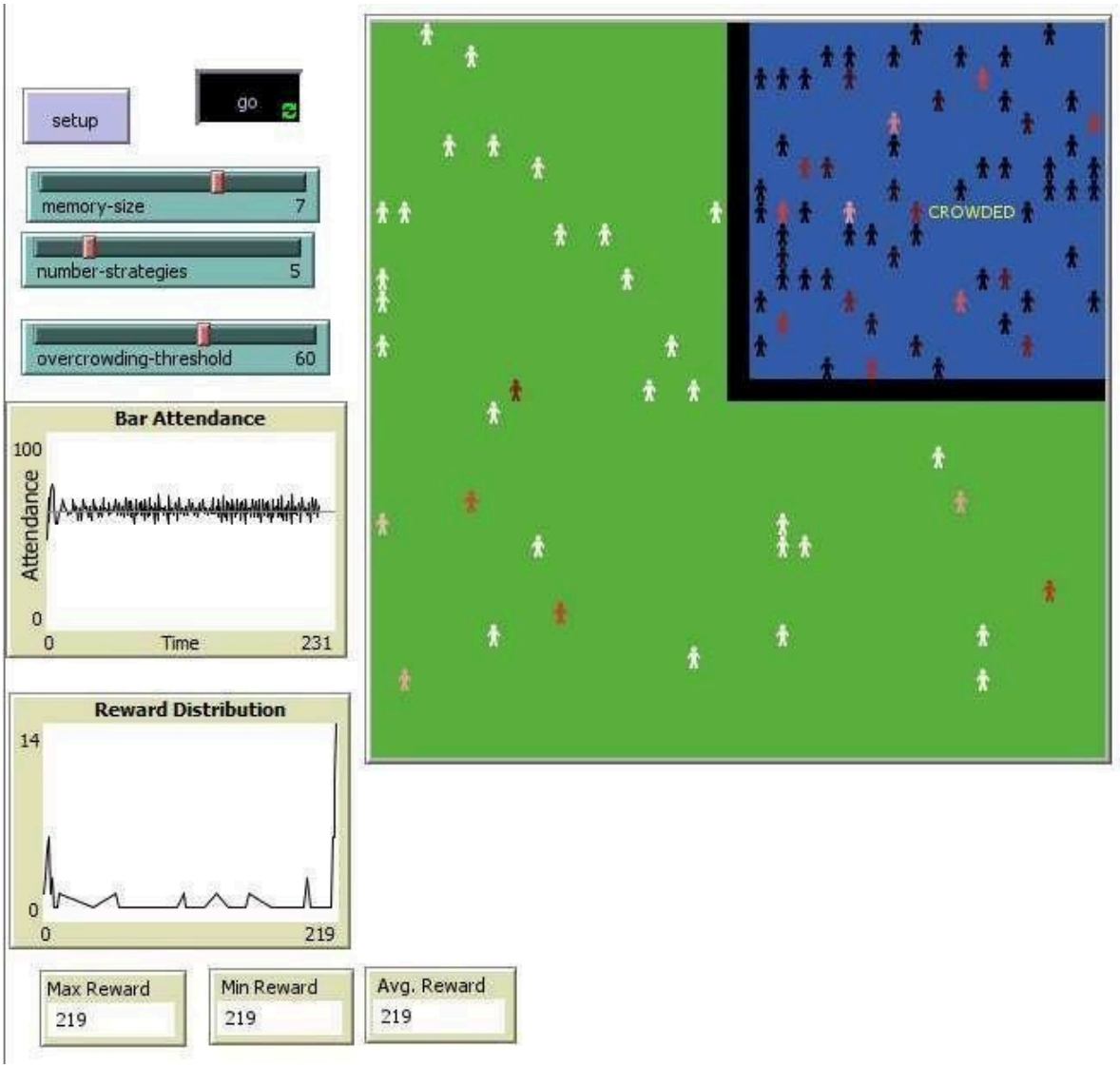


Observation:

With MEMORY-SIZE set to 8, each agent uses the past 8 weeks of attendance data to make predictions. This results in more stable and consistent predictions compared to smaller memory sizes. Fluctuations in predicted attendance from tick to tick are reduced, allowing agents to make more informed decisions about attending the bar, which leads to smoother attendance patterns over time.

2.

ReduceNUMBER-STRATEGIES and compare learning speed



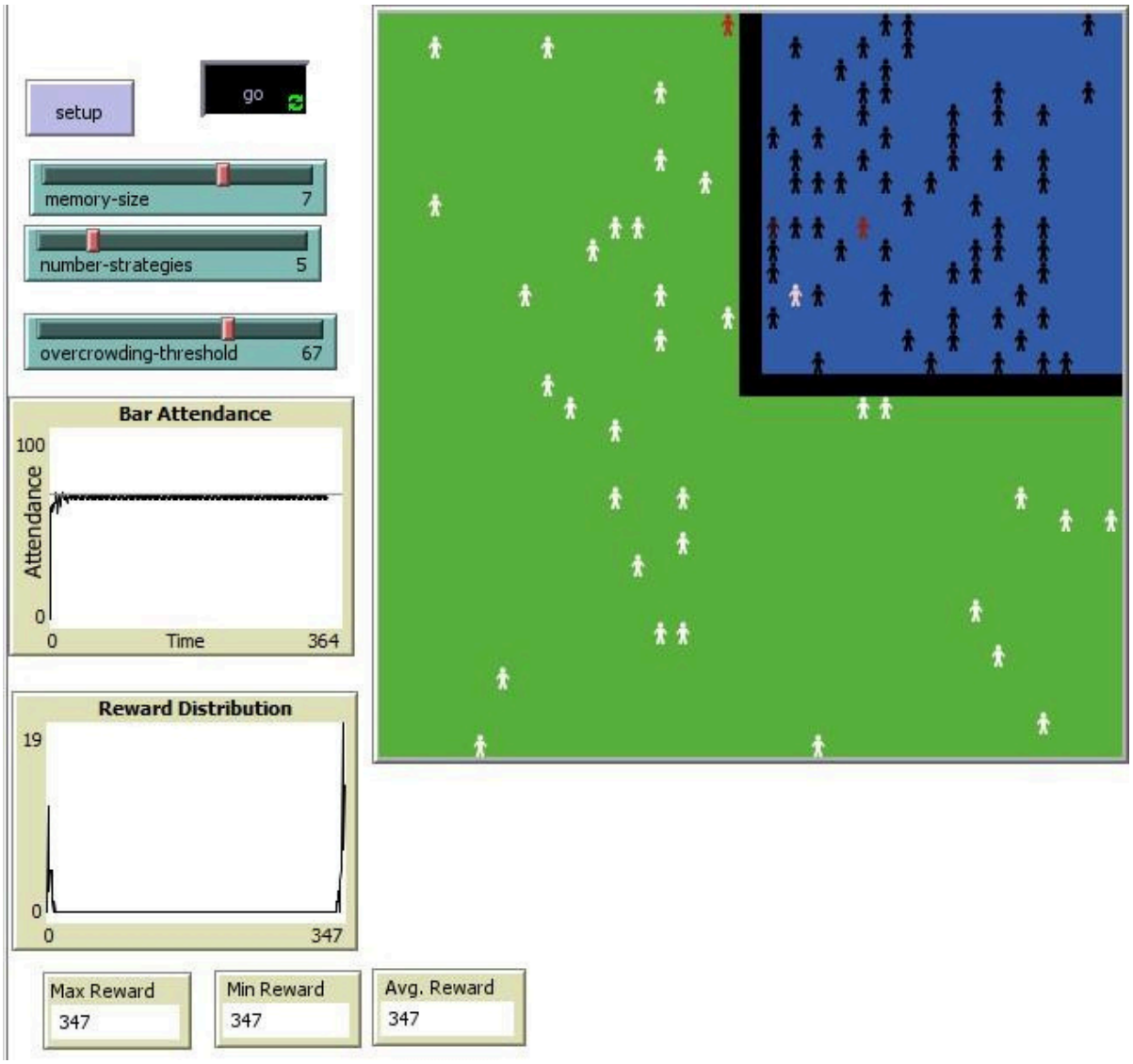
Observation:

With the number of strategies reduced to 5, each agent has fewer options to choose from when predicting attendance. As a result, learning is slower because agents have a smaller pool of

3.

strategiestoidentify the most accurate predictions. This leads to less optimal decisions and slightly morefluctuations in bar attendance compared to having more strategies.

ChangeOVERCROWDING-THRESHOLD and analyze attendance

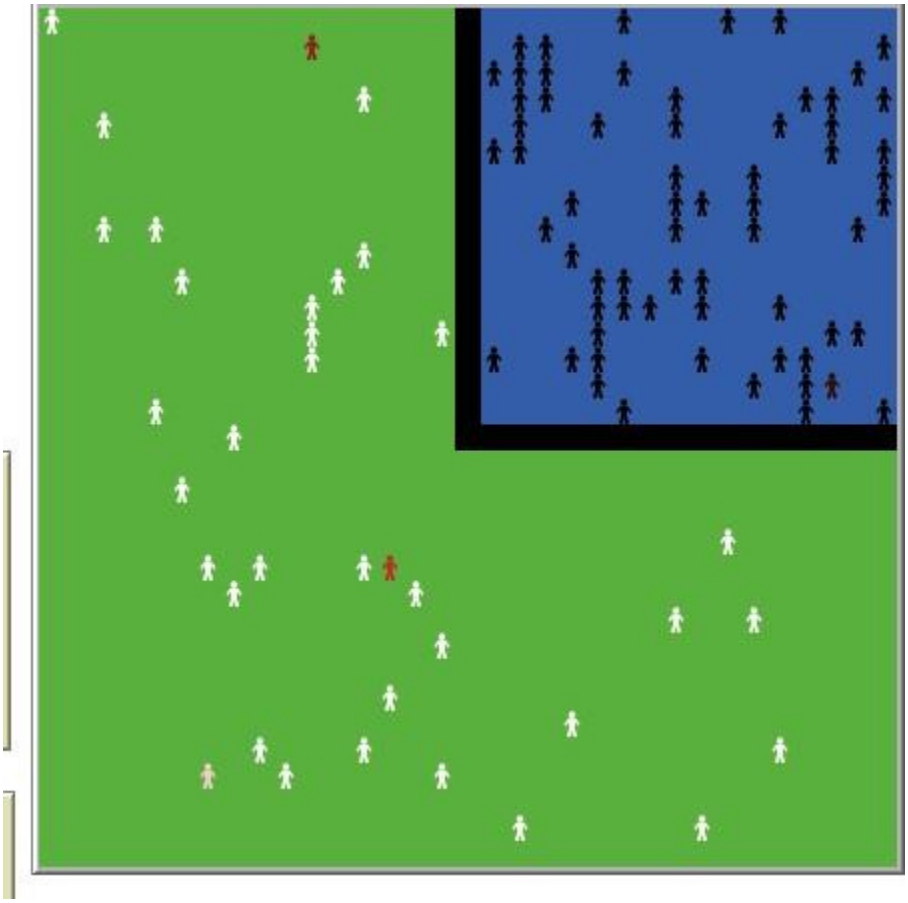


Observation:

4.

With theovercrowding-threshold set to 67, more agents are allowed to attend the bar before it is consideredcrowded. This results in higher overall attendance and fewer agents avoiding the bar. Thebarreaches the ‘CROWDED’ state less frequently, and agents adjust their behavior accordingly,demonstrating how the threshold directly influences attendance patterns and decision-making.

Observereward distribution over long runs



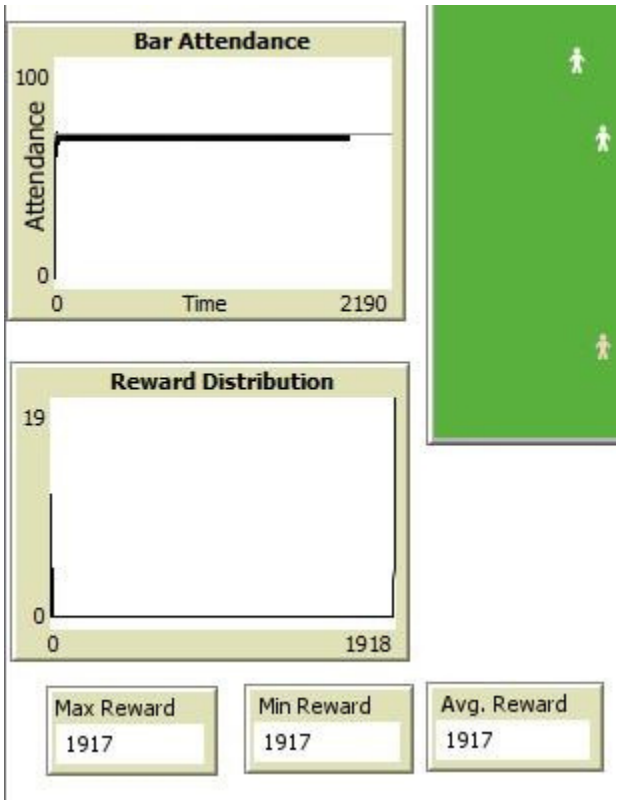
Observation:

Over long runs,agents who consistently attend the bar when it is not crowded accumulate higher rewards,while those who avoid the bar or attend when it is crowded accumulate fewer rewards. The reward distribution shows a clear pattern where successful prediction and

5.

decision-making lead to greater rewards. This is visually represented by turtles' colors, with darker turtles indicating higher accumulated rewards.

Identify periods of frequent overcrowding



Observation:

Periods offrequentovercrowding occur when many agents predict that attendance will be low and decidetogotothe bar simultaneously. These periods can be identified by repeated appearancesofthe'CROWDED' label on the bar patches. This demonstrates how collective decision-makingbased on similar strategies can lead to temporary spikes in attendance, even though agentsaretrying to avoid overcrowding.

Post-Lab Questions

1. Why do agents fail to reach a stable attendance level?
Agents continuously adapt their decisions basedon predictions and past attendance. Since many agents act simultaneously, their collective behavior changesattendance each tick, preventing a perfectly stable level.

2. How does limited memory affect predictions?
Limitedmemoryreduces thehistoricaldataagentscan use, leading to less accurate and more fluctuating predictions of attendance.

3. Why is always attending not the chosen strategy?
Alwaysattendingrisks overcrowding the bar,whichlowers rewards. Agents learn to balance attendance to maximize reward rather than blindly attending every tick.

4. What role does reward play in learning?
Reward signals which strategiesare successful.Agents prefer strategies that have yielded higher rewards, guiding future predictions and improving decision-making over time.

5. How does this model reflect real economic behavior?
The model mimics situations whereindividuals makedecisions based on limited information and adapt to avoid overcrowded resources, similar to real-worldmarkets where participants respond to supply, demand, and collective behavior.

Lab 12: Brian's Brain — A 2D Cellular Automaton

Objective

- To explore the dynamics of a two-dimensional cellular automaton (CA) with three cell states.
- To observe moving and stable patterns called gliders.
- To interactively manipulate cell states and understand how simple rules lead to complexbehaviors.

Tool / Environment

- **Software:** NetLogo 6.4.0
- **Platform**:Windows
- **Environment:** Interface + Code Tab

What is Brian's Brain?

Brian's Brain is a 2D cellular automaton. Unlike simpler CAs (like Conway's Game of Life) that use only two states (live and dead), Brian's Brain uses three states:

Cell State	Color	Description
Firing	White	Active cell that will soon become refractory
Refractory	Red	Just fired, cannot fire again immediately
Dead	Black	Inactive cell, can become firing if conditions are met

This CA is interesting because many patterns **move steadily across the grid**, forming**gliders** and other dynamic structures.

Lab Procedure

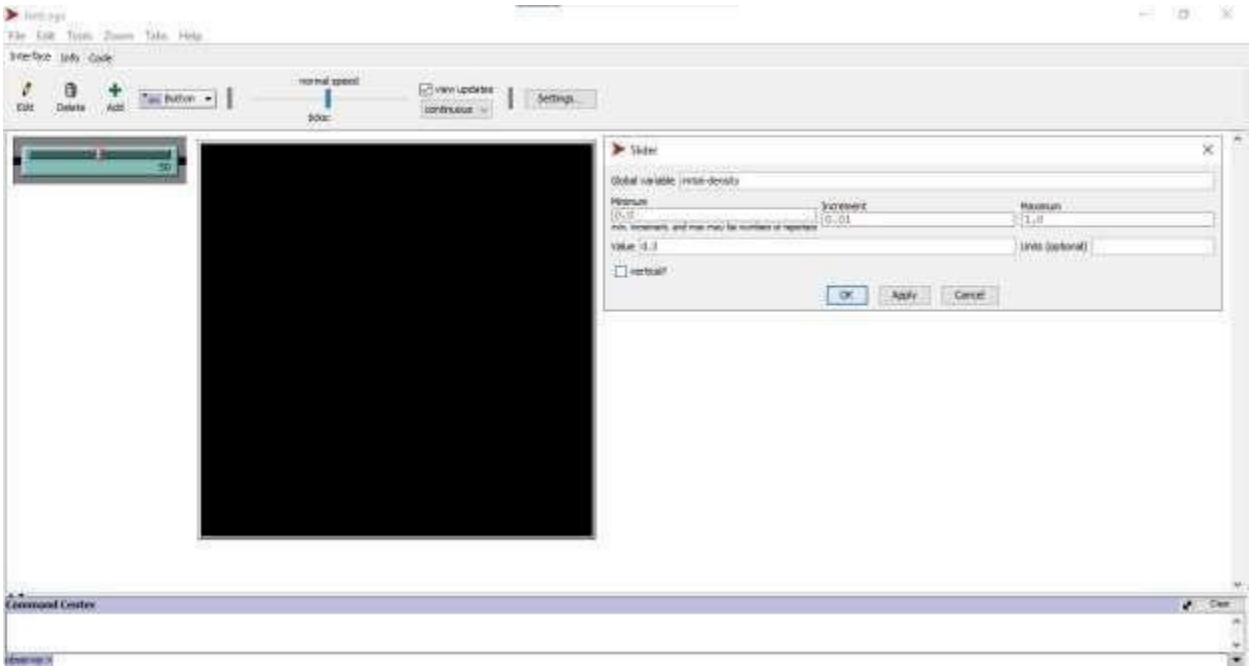
Step 1: Add Required Interface Elements

Sliders

Add these sliders in the Interface tab:

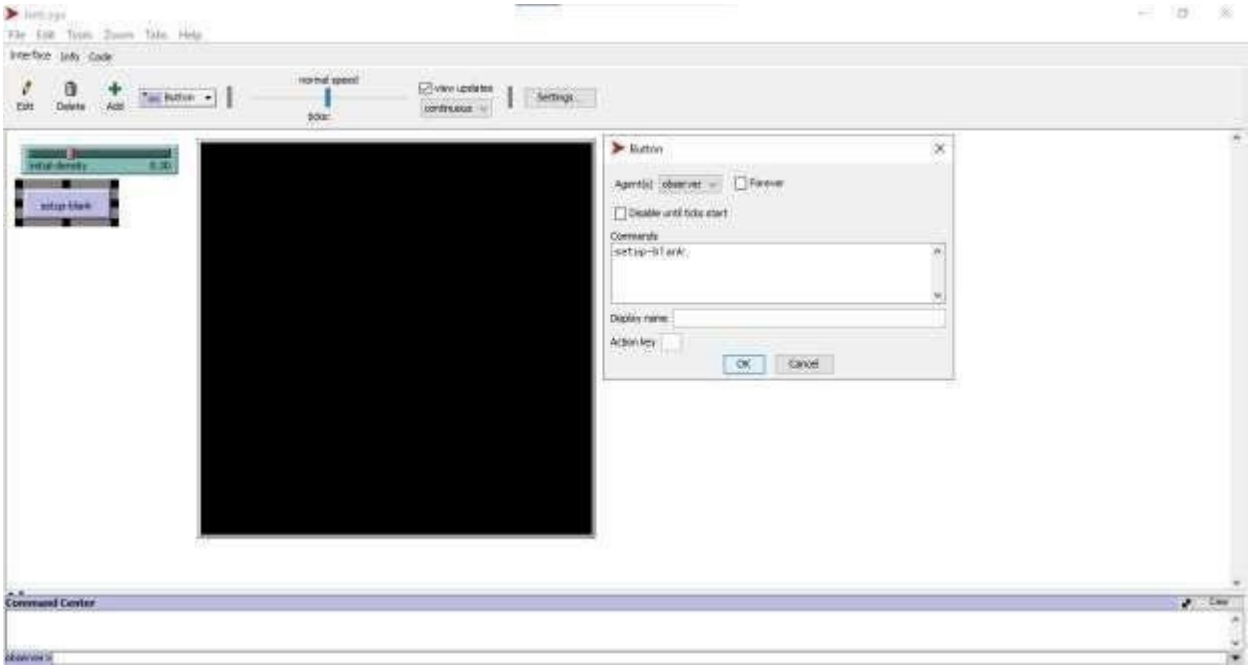
INITIAL-DENSITY

- Min: 0.0 o Max: 1.0o Initial: 0.3
- Increment: 0.01



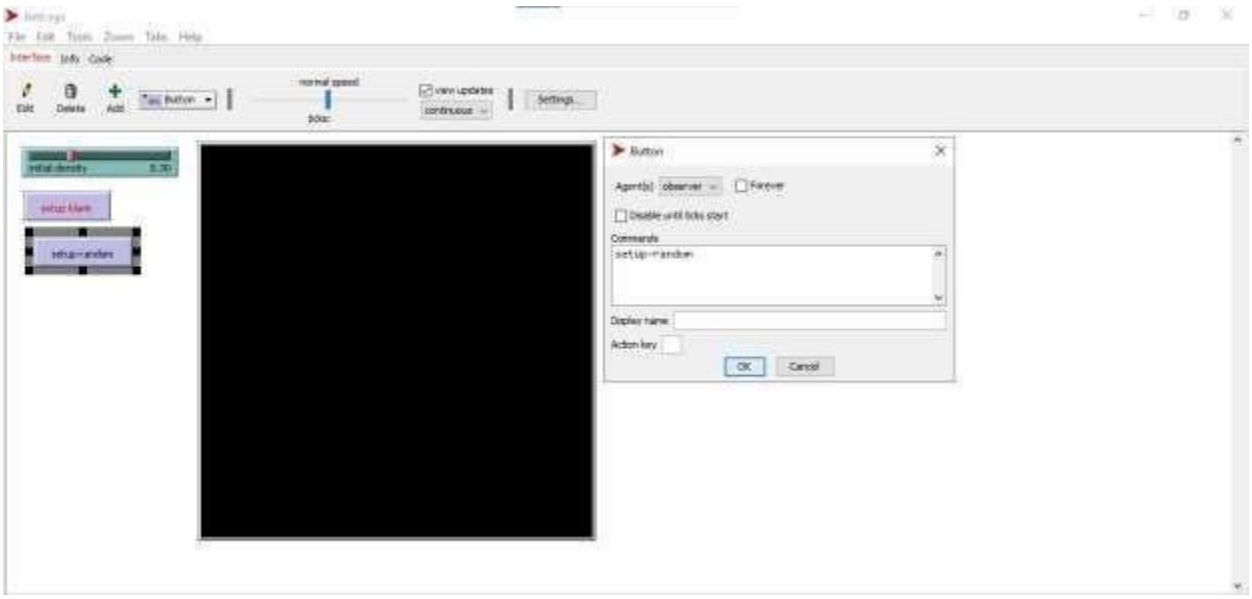
Buttons

1. **Setup Blank** o Command: setup-blank

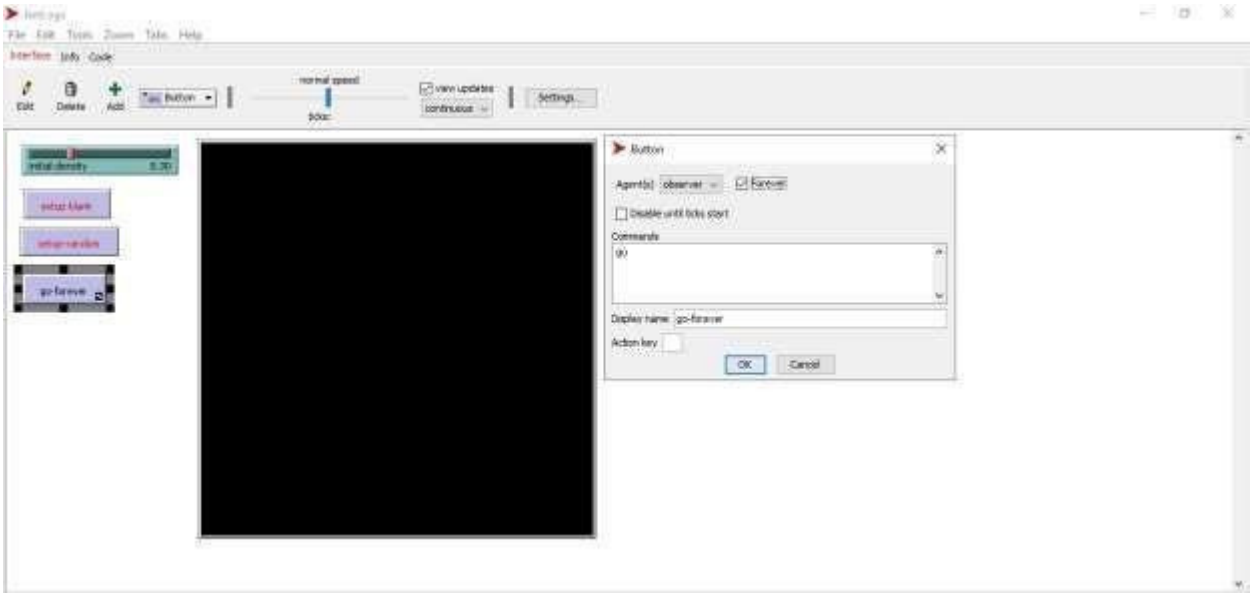


2. **Setup Random**

- Command: setup-random

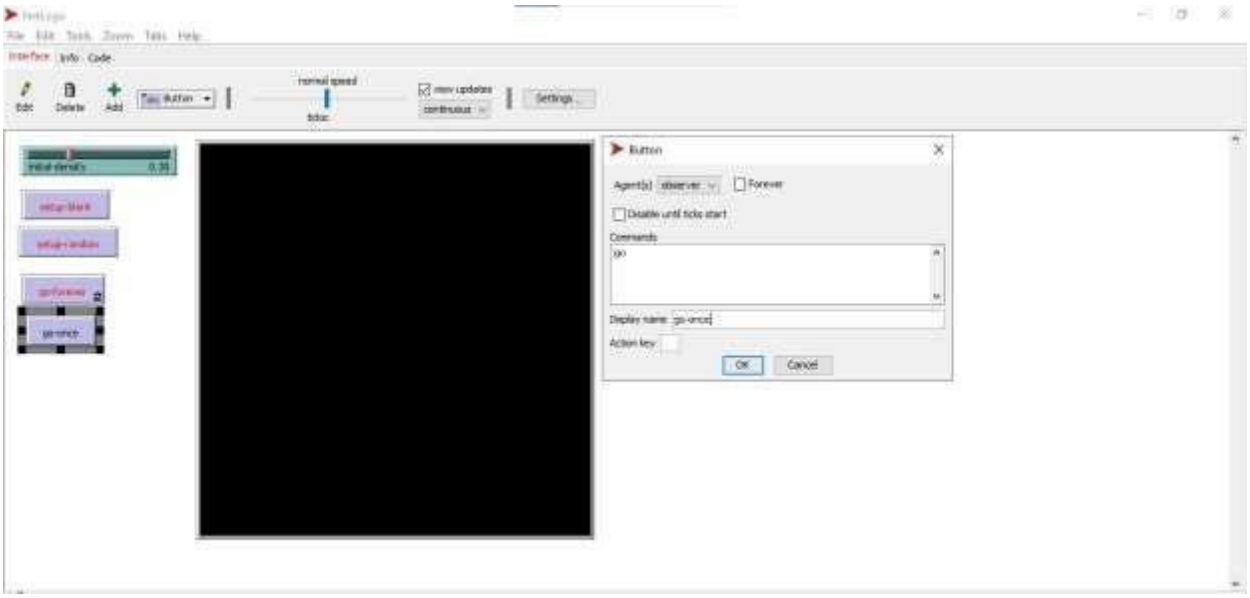


3. **Go Forever**
    - Command: go
    - Display name: Go Forever



4. **Go Once**
    - Command: go
    - Display name: Go

Once



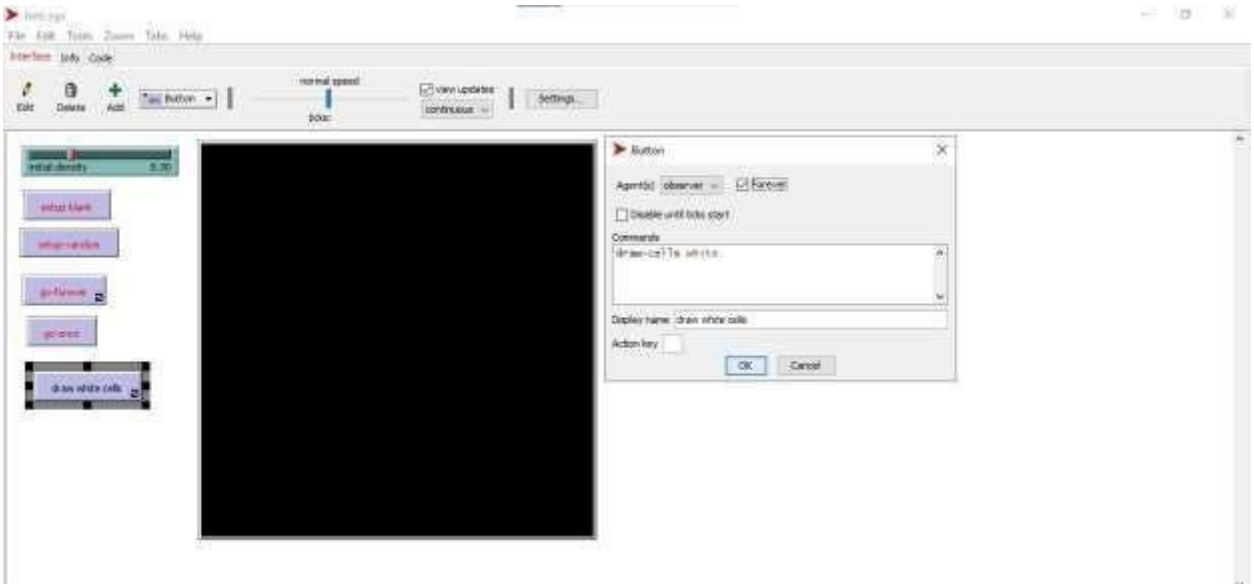
5. **Draw White Cells**

   Command: draw-cells white

   Display name: Draw White

   Cells

   Action: Forever



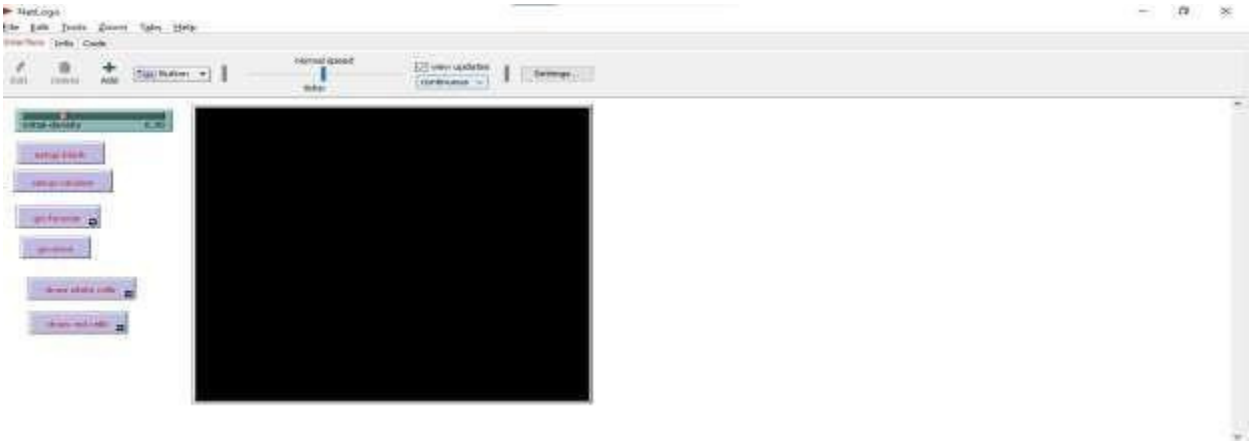
6. **Draw Red Cells**

   Command: draw-cells red

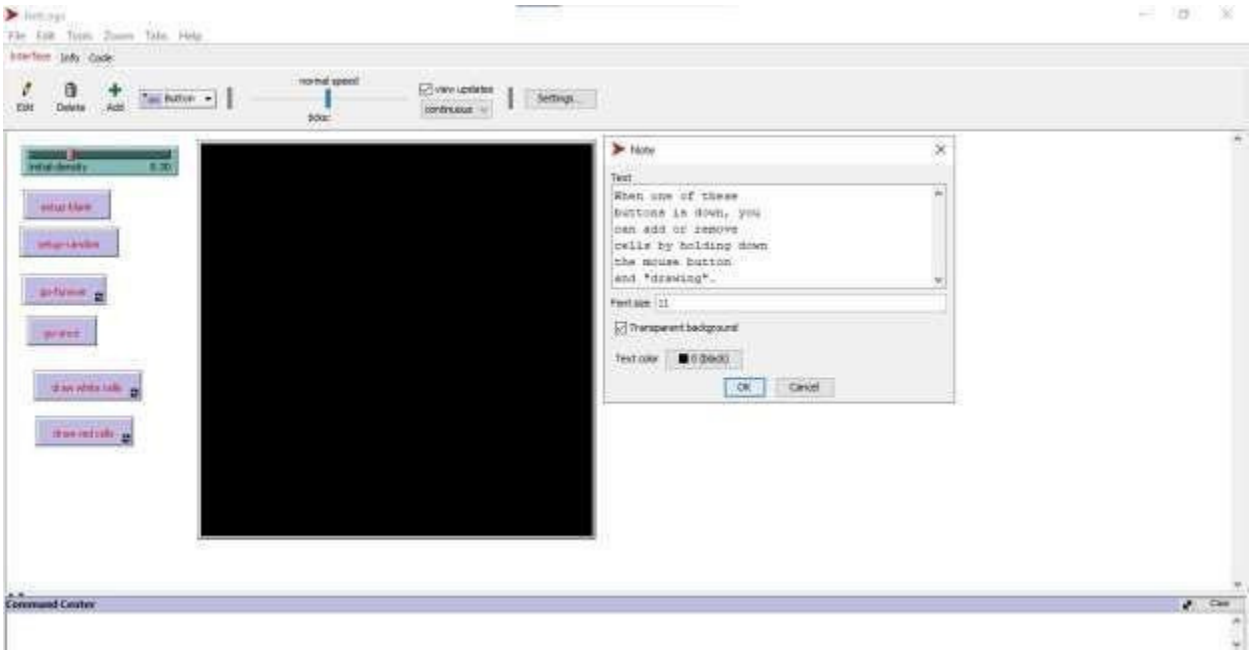
Display name: Draw Red

Cells

Action: Forever



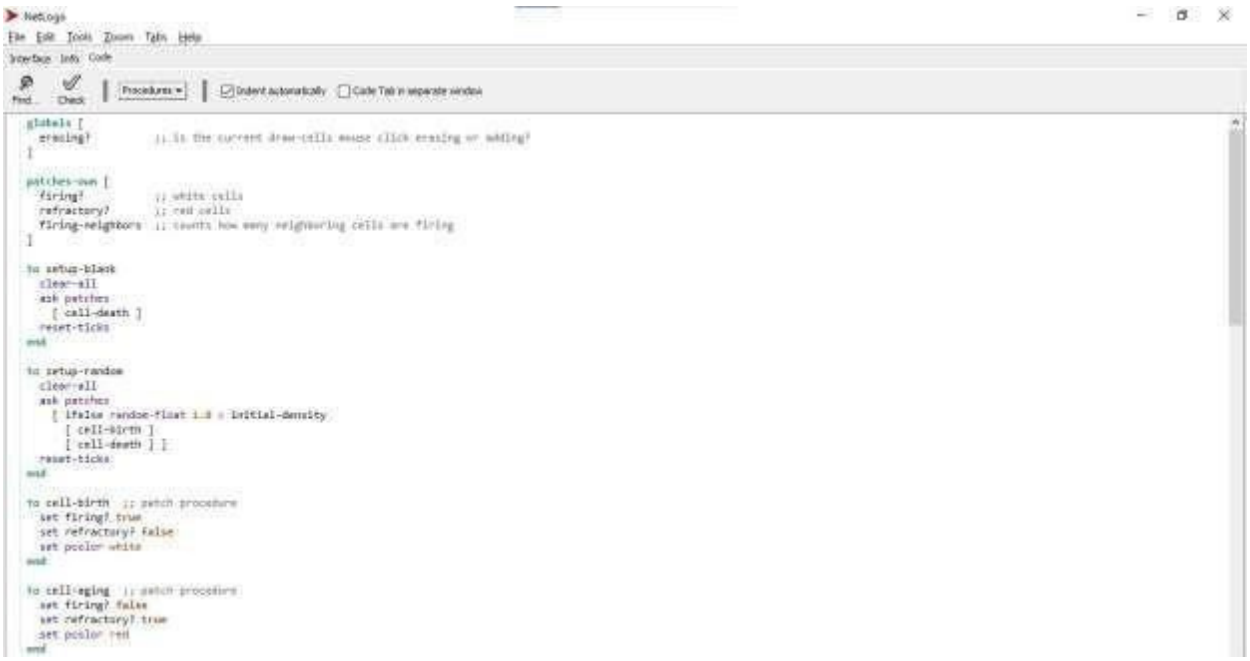
7. **Note** o **Text:** When one of these buttons is down, you can add or remove cells by holding down the mouse button and "drawing".



Step 2: Add the Full Code in the Code Tab

```
globals  [erasing?;;isthecurrentdraw-cells mouse click erasing or
     adding?
]

patches-own [
  firing?          ;; white cells refractory?        ;; red cells firing-
     neighbors ;; counts how many neighboring cells are firing
]

to setup-blank
  clear-all  ask
     patches
     [ cell-death
     ] reset-ticks
     end

to setup-
  random clear-
  all ask patches
    [ ifelse random-float 1.0 < initial-density
      [ cell-birth ]
```

```
    [ cell-death ]
    ] reset-ticks

    end

to cell-birth ;; patch
    procedure set firing?  true
    set refractory?   false  set
    pcolor white end

to cell-aging ;; patch
    procedure set firing?  false
    set  refractory?  true  set
    pcolor red
end
```



```
to cell-death ;; patch procedure
    set firing? false set
    refractory? false set pcolor
    black end

 to go ask patches
    [ set firing-neighbors count neighbors with [firing?] ]
  ;; Starting a new "ask patches" here ensures that all the patches
  ;; finish executing the first ask before any of them start executing
```

```
  ;; the second ask. This keeps all the patches in sync with each other, ;;
  so the births and deaths at each generation all happen in lockstep. ask
  patches [ ifelse firing?
      [ cell-aging ]
      [ ifelse refractory?
        [    cell-death    ]    [    if
        firing-neighbors =
      2[ cell-birth ] ] ]
      ] tick
end
```


```
to draw-cells [target-color] ifelse mouse-down? [      if erasing? = 0 [ set
    erasing? target-color = [pcolor] of patch mouse-xcor mouse- ycor
  ]
  ask patch mouse-xcor mouse-ycor
    [     ifelse erasing? [          cell-
    death
    ] [
      ifelse target-color = white [
    cell-birth
      ] [
    cell-aging
      ]
    ]
  ]
  display
] [
  set erasing? 0
]
```

end

Output

1. Select random

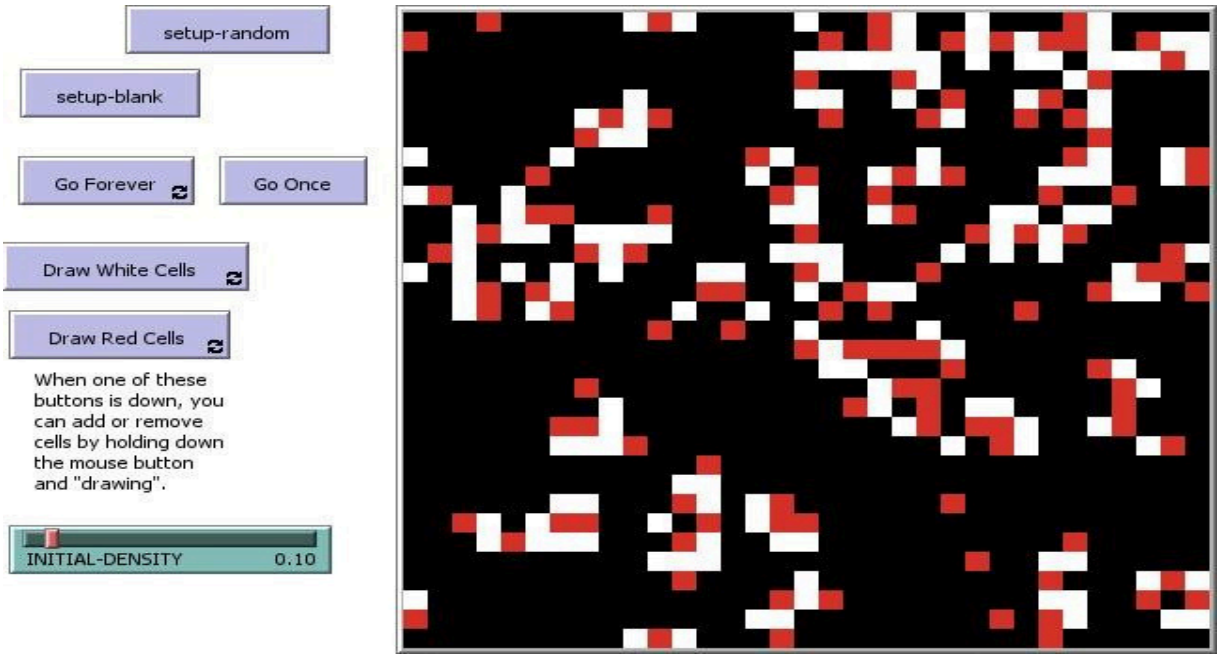
2. Select go-once

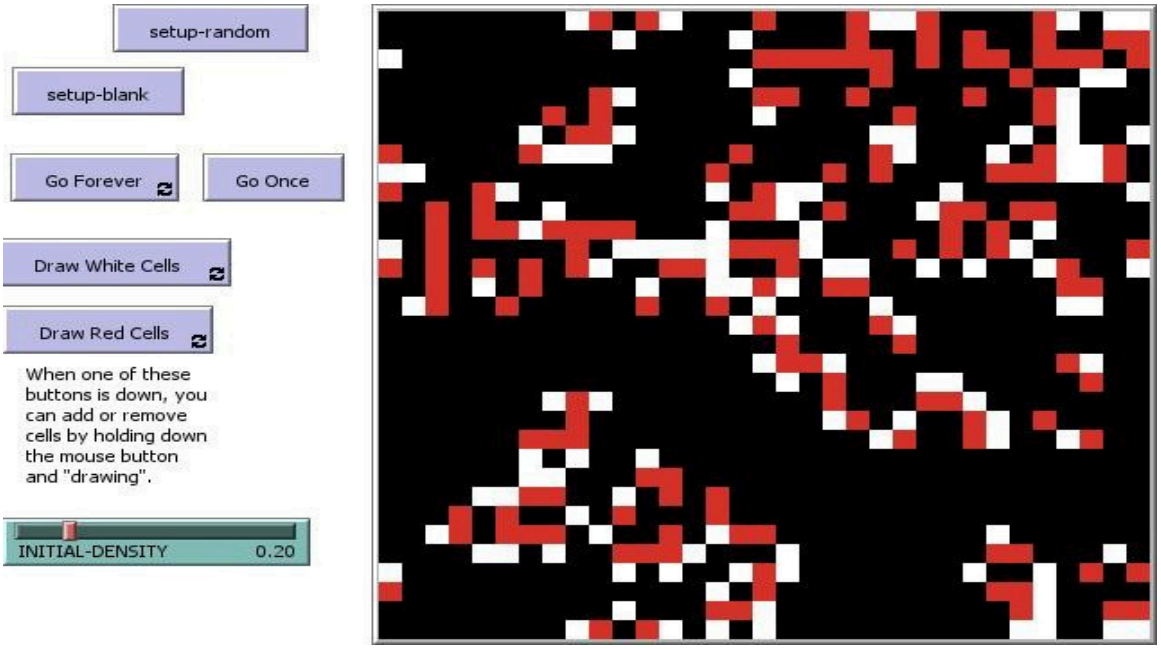
3. Select go-forever

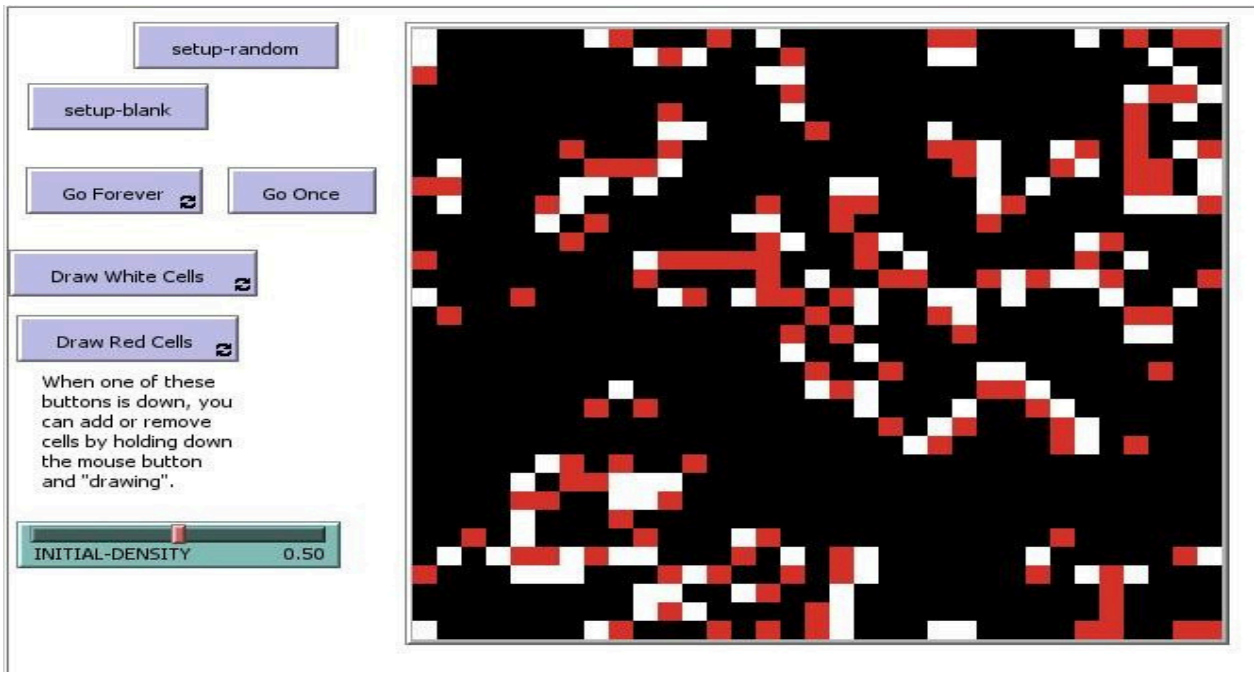
STUDENT TASK

1. Explore Initial Patterns o Use the Setup Random button to initialize the grid with different INITIALDENSITY values (0.1, 0.2, 0.5).

o Observe how the density of firing cells affects the evolution of patterns over time.







OBSERVATION

Density 0.1: Few firing cells; patterns grow slowly, and clusters are small.

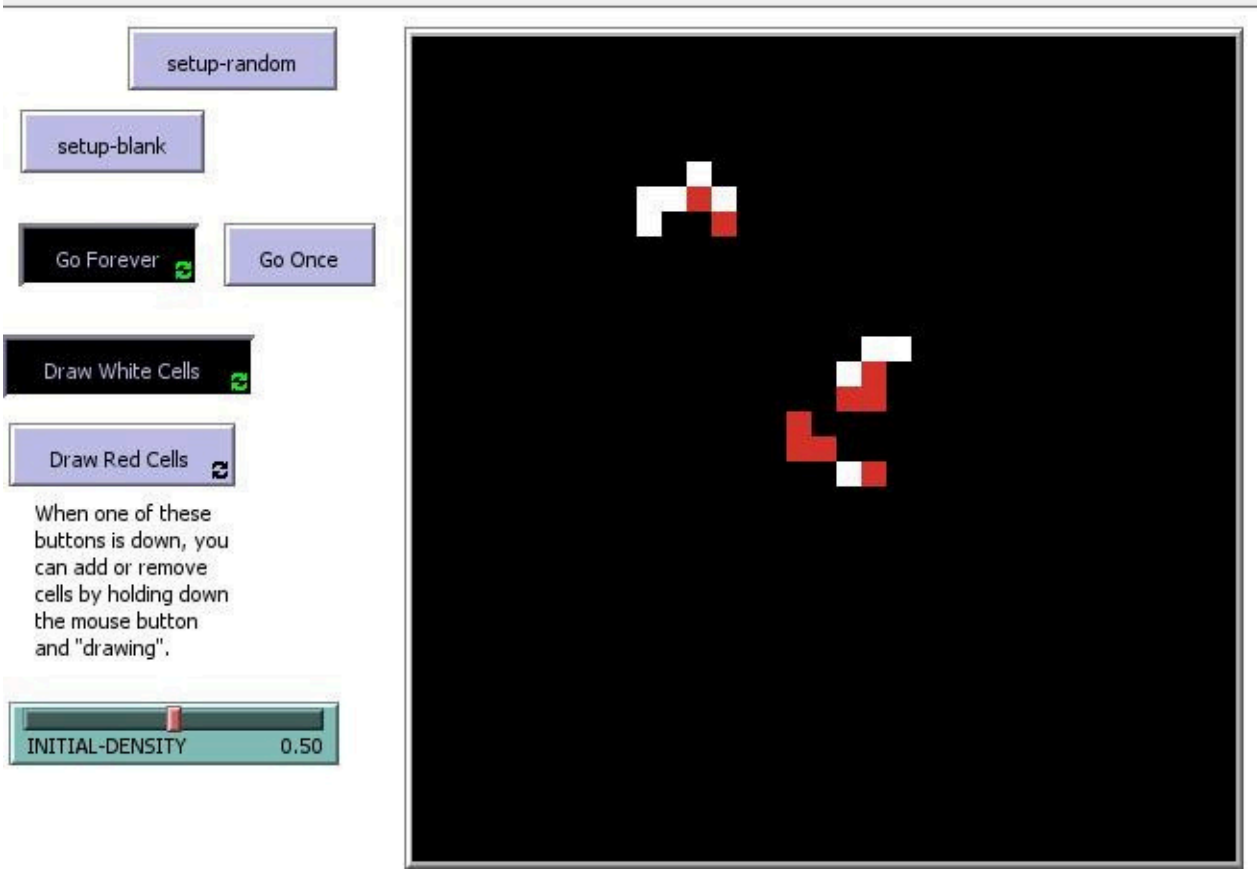
Density 0.2: More firing cells; clusters form faster, and activity spreads more widely.

Density 0.5: Many firing cells; the grid is highly active, and patterns emerge rapidly but may dieoutorstabilize depending on neighbor interactions.

2.

Observe Gliders o Run Go Forever and watch for moving patterns (gliders).

o Try drawing small groups of firing cells manually using Draw White Cells to see if you can create new gliders.

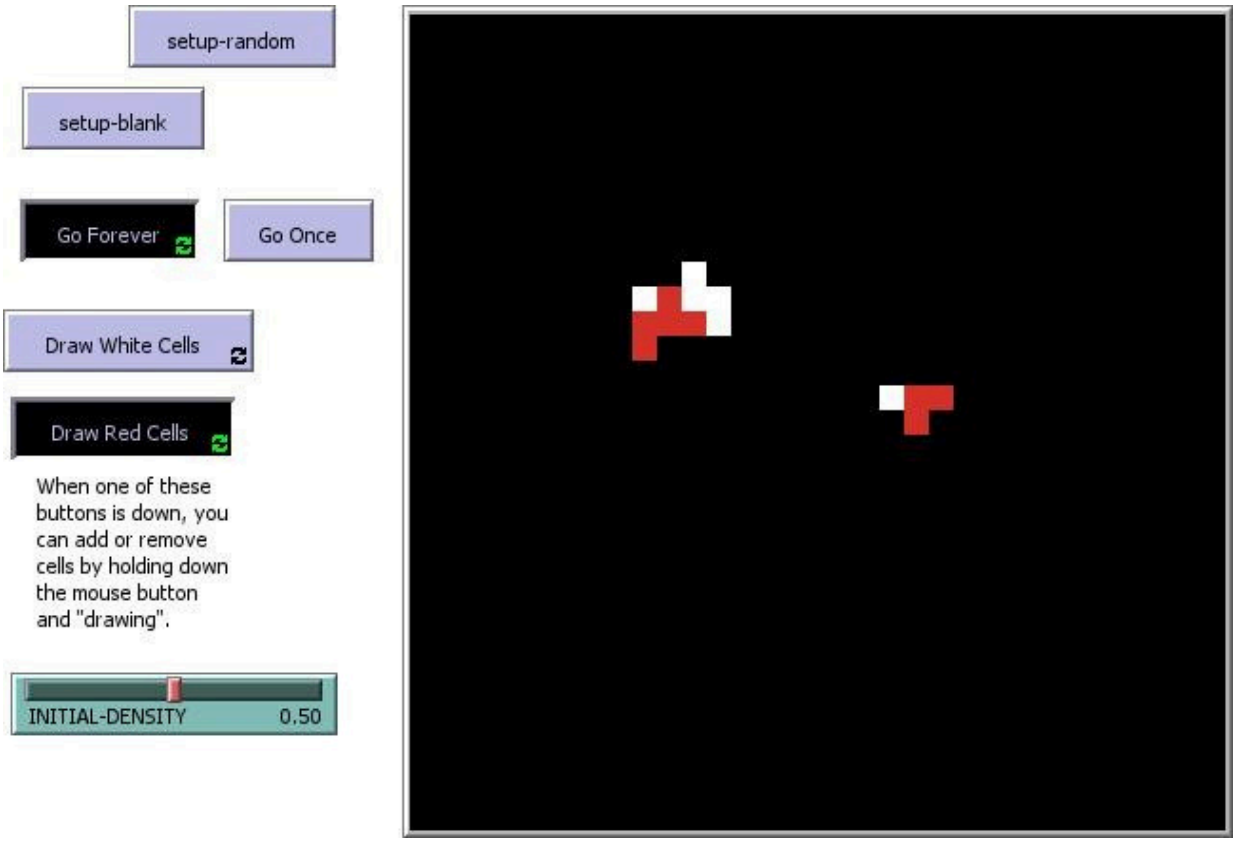


Observation

Gliders are small patterns of firing cells that move diagonally across the grid while maintaining their structure. By manually drawing specific configurations, new gliders can be created. This demonstrates that local arrangements of cells determine the emergence and direction of moving patterns in the cellular automaton.

3.

Manipulate the Grid o Use Draw Red Cells and mouse erasing to modify patternsduring thesimulation. o Observe how adding or removing cells affects glider formation and stability.



Observation

Manipulatingthe grid during the simulation shows that adding or removing cells significantly affects gliderbehavior. Red cells can block or destroy gliders, while erasing cells can create opportunitiesfor new gliders to emerge. This demonstrates that the system is highly sensitive to local changes,and small modifications can dramatically alter pattern stability and evolution.

Experimentwith Cell Rules o Modify the go procedure to require 3 firing neighbors foradead cell to become firing. o Compare the simulation outcome with the original rule.

4.

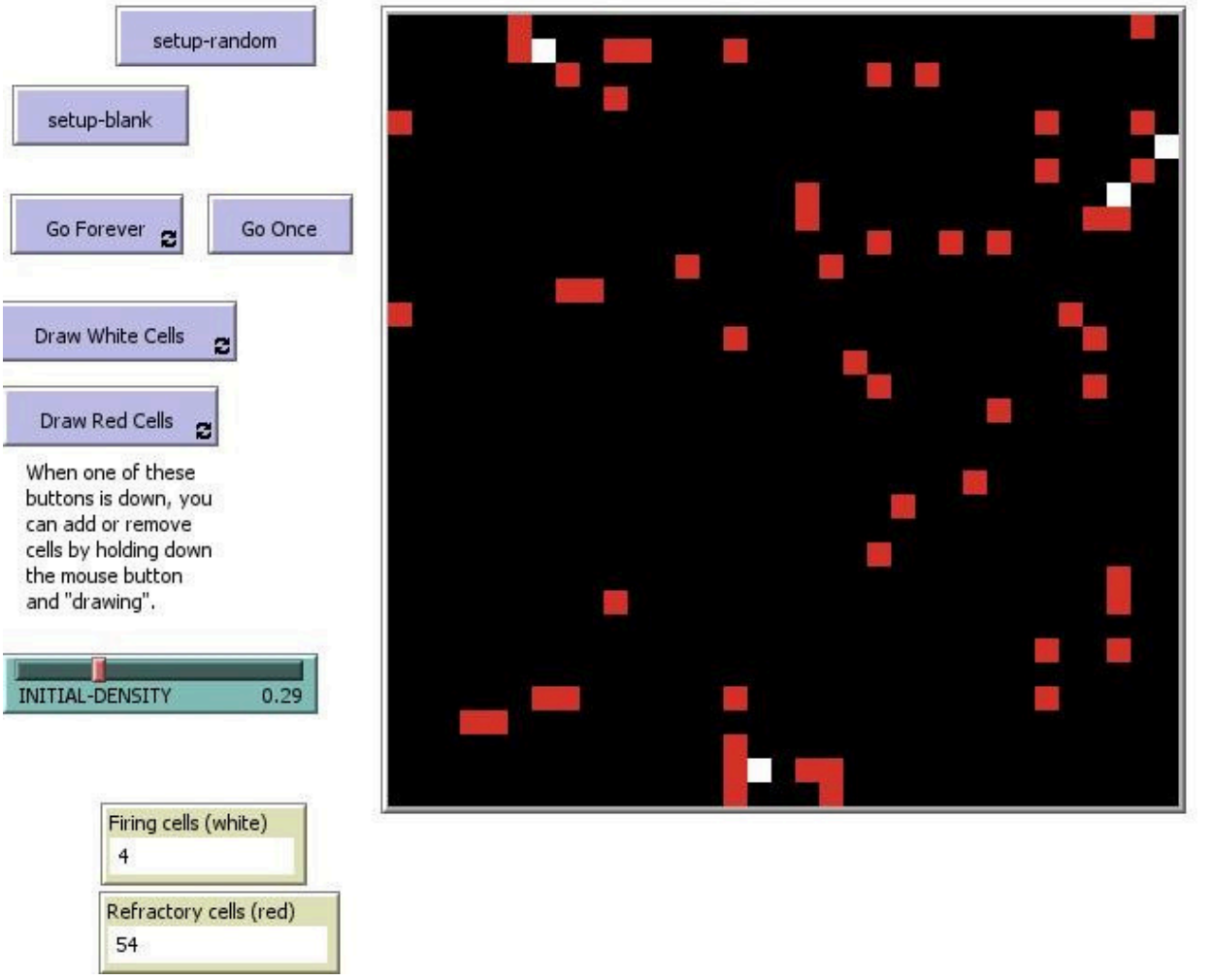
```
to go
ask patches
[ set firing-neighbors count neighbors with [firing?]
;; Starting a new "ask patches" here ensures that all
;; finish executing the first ask before any of them
;; the second ask.  This keeps all the patches in syn
;; so the births and deaths at each generation all ha
ask patches
[ ifelse firing?
[ cell-aging ]
[ ifelse refractory?
[ cell-death ]
[ if firing-neighbors = 3
[ cell-birth ] ] ] ]
tick
end
```

Observation

Changing thebirth rule to require 3 firing neighbors reduces the number of new cells and slows pattern evolution. Compared to the original rule (2 neighbors), fewer gliders and moving patterns emerge. This demonstrates how small changes in local rules can drastically alter the overall behavior and dynamics of the cellular automaton.

Monitor Cell Counts o Observe the monitors for firing and refractory cells. o Record the number of cells in each state at regular intervals (every 10 ticks).

5.

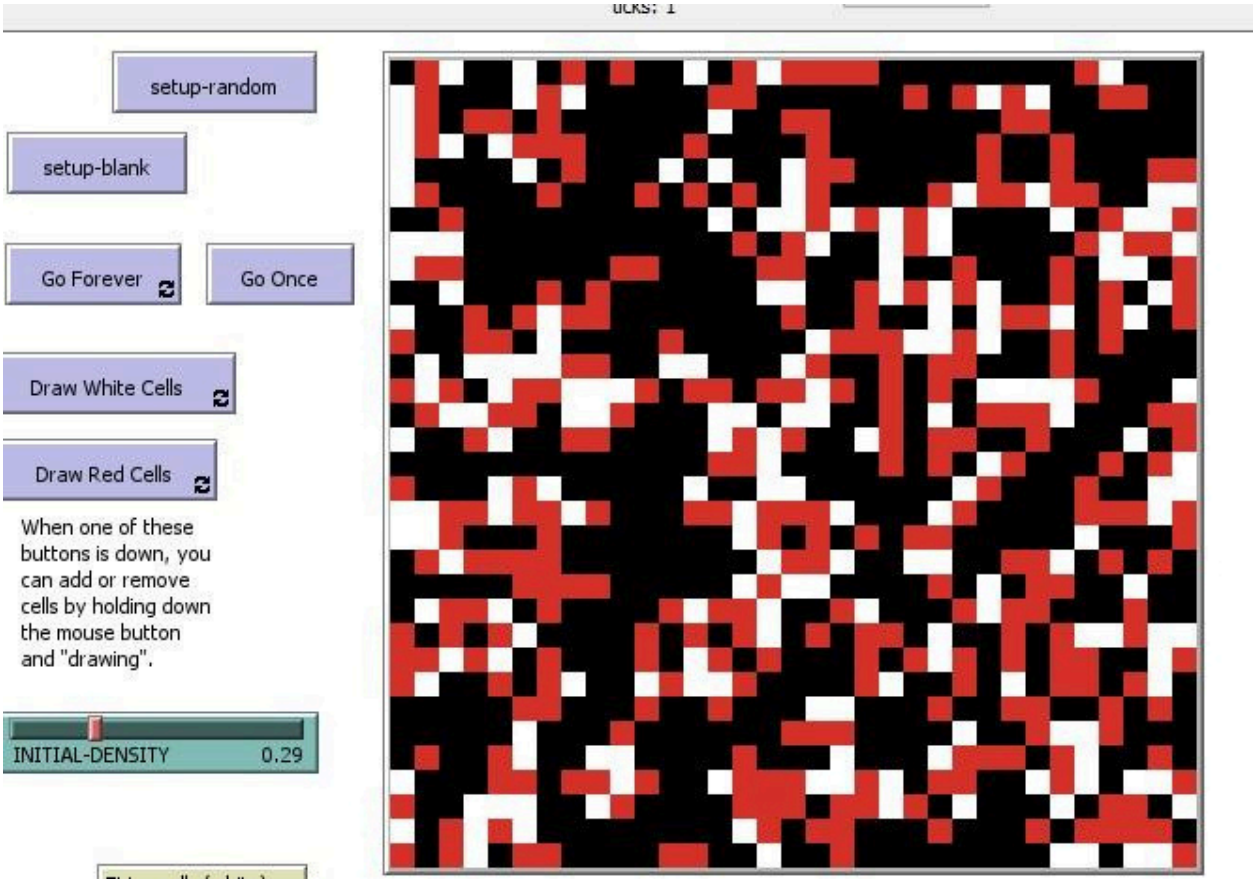


Observation:

The monitorsshow how many cells are in the firing and refractory states at each tick. Firing (white) cellsincrease and then transition to refractory (red), while refractory cells eventually die and becomeblack. Recording the counts every 10 ticks highlights the oscillatory nature of the system andhowcells cycle through active and inactive states over time.

6.

Pattern Analysis o Identify stablepatterns, oscillators, andmoving gliders. o Note the direction, speed, and interaction of gliders.



OBSERVATION

The simulation shows three types of patterns:

- **Stable patterns** like blocks remain unchanged over time.
- **Oscillators** cycle through repeating states without moving.

7.

- **Gliders** are small patterns that move diagonally across the grid. Gliders maintain their shape, move at a steady speed, and interact with other patterns, sometimes merging or disappearing upon collision. Observing these patterns demonstrates how local interactions lead to complex and dynamic behavior in cellular automata.

POST LAB QUESTIONS

1. Understanding the Rules o Explain the three states of cells in Brian's Brain and howeachstatetransitionsto the next.

- **Firing (white):** Active cells that turn refractory in the next tick.
- **Refractory (red):** Recently fired cells that cannot fire immediately; they turn dead next.
- **Dead (black):** Inactive cells that can become firing if exactly 2 neighbors are firing.

o Why does a dead cell only become firing if it has exactly 2 firing neighbors?
This ensures controlled propagation and allows patterns like gliders to form, preventing overcrowding of firing cells.

2. Observation and Analysis o What differences do you notice when running the simulation with different initial densities?

Low density produces slow, sparse patterns; medium density forms clusters and moving gliders; high density results in rapid activity and sometimes chaotic patterns.

o How many distinct types of gliders can you identify in your simulation?

Multiple gliders are observed, usually moving diagonally. Variations in shape may produce differences in speed or interaction.

3. Experimentation o What happens if you change the rule for dead cells from 2 firing neighbors to 3?

Fewer cells are born, pattern evolution slows, and glider formation is often reduced or prevented.

o How does manually adding or removing cells affect the formation and movement of gliders?

Adding or removing cells can block or create gliders, redirect their movement, or destabilize existing patterns, showing sensitivity to local changes.

4. Critical Thinking o Why do gliders move steadily across the grid while other patterns remain static?

Gliders maintain a repeating firing-refractory-dead configuration that shifts position each tick while preserving shape. Other patterns lack this coordinated structure, so they remain static or oscillate in place.

o How does the combination of firing, refractory, and dead states lead to complex behavior despite the simplicity of the rules?

The interplay of the three states with neighbor interactions produces oscillations, gliders, and stable patterns, demonstrating emergent complexity from simple local rules.

5. Extension Question o Propose a modification to the Brian's Brain rules that could create new types of moving patterns or oscillators. Explain your reasoning.

Introduce a "super-firing" state: if a firing cell has 3 or more firing neighbors, it generates two new firing cells in adjacent positions. This could create faster-moving gliders or new oscillators by increasing local propagation while keeping predictable rules.

Lab 13: One-Dimensional (1D) Elementary Cellular Automata

Objective

- To understand the concept of one-dimensional cellular automata (1D CA).
- To study how different elementary CA rules generate different patterns.
- To observehow simple binary rules can produce complex and unpredictable behavior.

Tool / Environment

- **Software:** NetLogo 6.4.0
- **Platform:** Windows
- **Environment:** Interface + Code Tab

Theory

What is a 1D Cellular Automaton?

Thislab demonstrates **one-dimensional cellular automata (1D CA)** using NetLogo. A cellular automaton is a simple computationalsystemthat evolvesover time according to a fixed set of rules. It consists of a collection of cells arranged in a structure, where each cell can exist in one of two states: **ON** or **OFF**.

At the start, the system is initialized with a specific pattern of ON and OFF cells. At each time step (called a **tick**), all cells update their states simultaneously based on predefined rules. These rules determine whether a cell will turn ON or OFF in the next step.

In **one-dimensional cellular automata**, cells are arranged in a single horizontal row. Each cell observes:

- Its own current state
- The state of its immediate left neighbor
- The state of its immediate right neighbor

Using this information, the cell determines the state of the cell directly **below it** in the next row. This process continues row by row until the bottom of the screen is reached, producing a visual pattern over time.

Rule System in 1D Cellular Automata

Since each cell checks three cells (left, center, right) and each cell can be either ON or OFF, there are **8 possible neighborhood combinations**:

III, IIO, IOI, IOO,

OII, OIO, OOI, OOO

Each combination has a rule that decides whether the next cell will be ON or OFF. Because each of these 8 rules can independently be ON or OFF, there are:

2^8 = 256 possible rules

These rules are knownas **Elementary Cellular Automaton rules**.

Rule Representation

In this model:

- **I** represents an ON cell
- **O** represents an OFF cell

Each rule switch corresponds to one neighborhood configuration. For example:

- If the **OOO** switch is ON, then when a cell and both its neighbors are OFF, the next cell will turn ON.
- If the switch is OFF, the next cell will remain OFF.

The overall rule is represented as a **rule number (0–255)**, calculated by converting the ON rule combinations from binary to decimal and adding them.

Example:

If the switches **011** and **001** are ON:

- 011 (binary) = 3
- 001 (binary) = 1
- Rule number = **4**

Importance of Cellular Automata

Cellular automata are simple instructure but capable of producing very complex patterns. In his book **"ANew KindofScience"** , Stephen Wolfram explains that such simple computational systems can model many natural phenomenamore effectively than traditional mathematical equations.

Because of this, cellular automataare used to study:

- Pattern formation
- Complex systems
- Natural processes

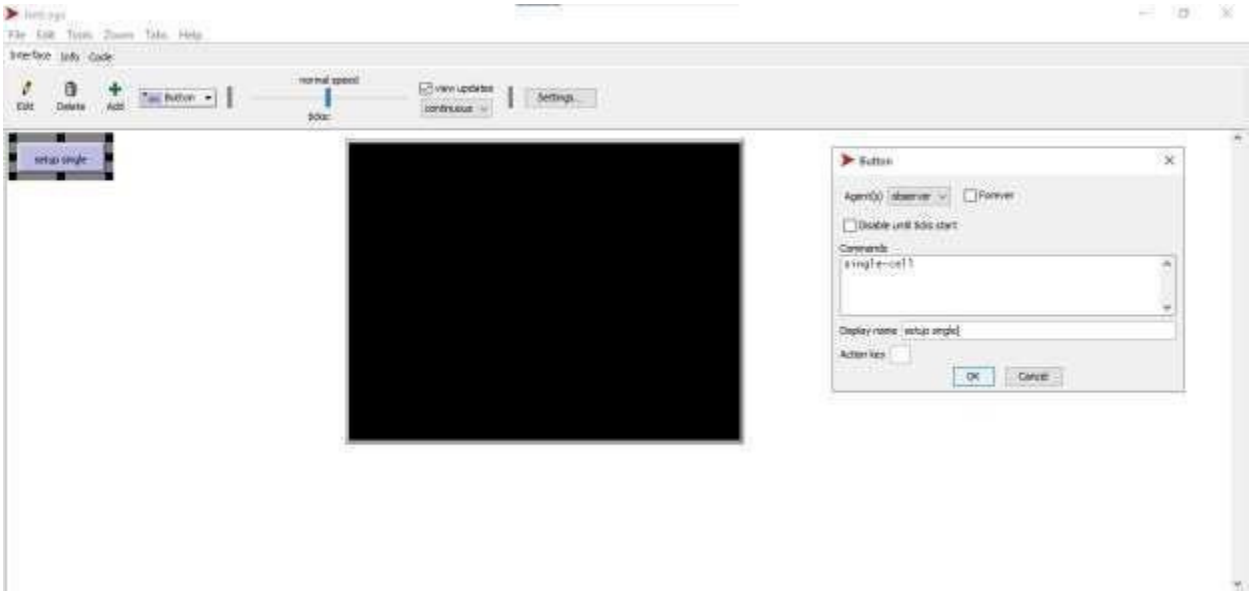
- Computational theory

This model allows users to explore**all256elementary CA rules** and observe how simple rule changes can lead to dramatically different behaviors.

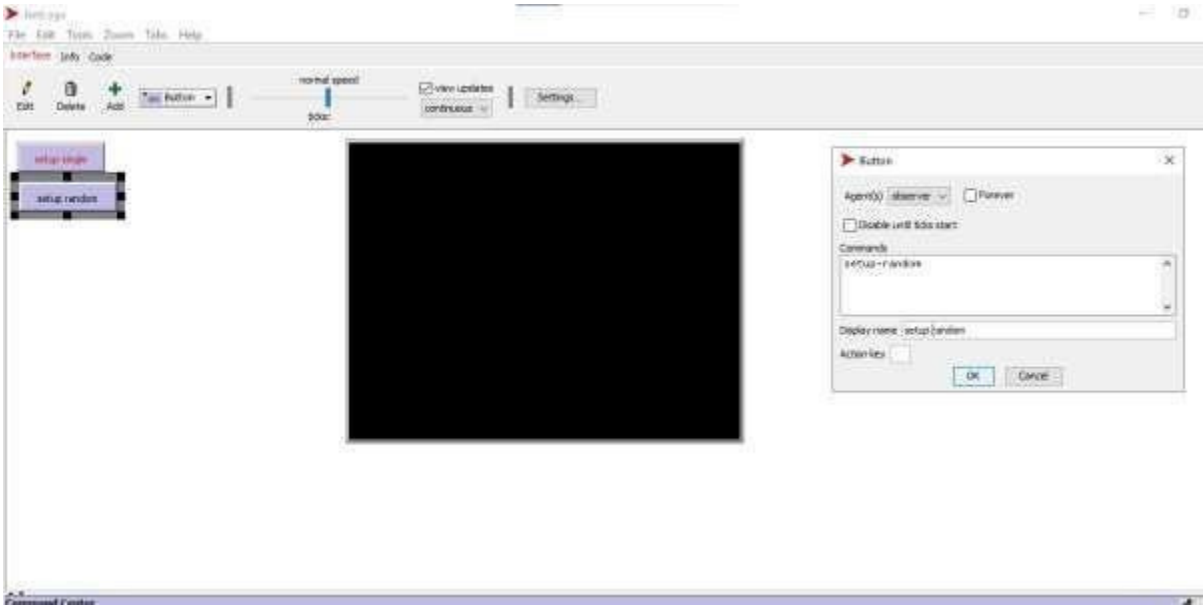
Lab Procedure

Step 1:AddRequiredInterfaceElements Design the Interface exactly as shown:

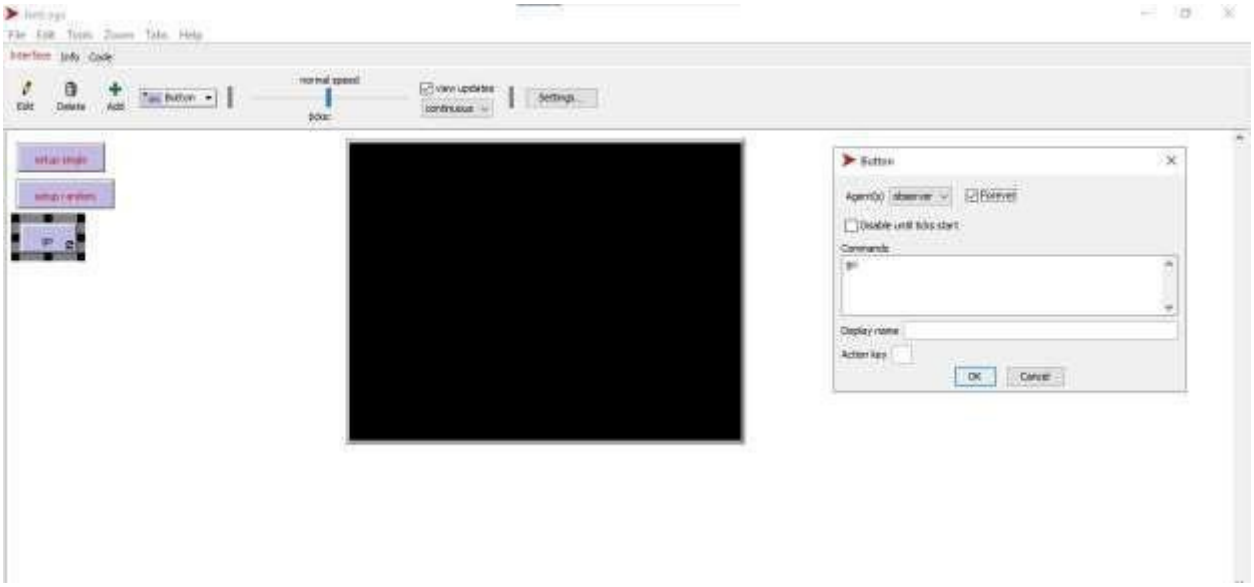
- Buttons:
- Commands: setup-cell
    - Display Name: setup single



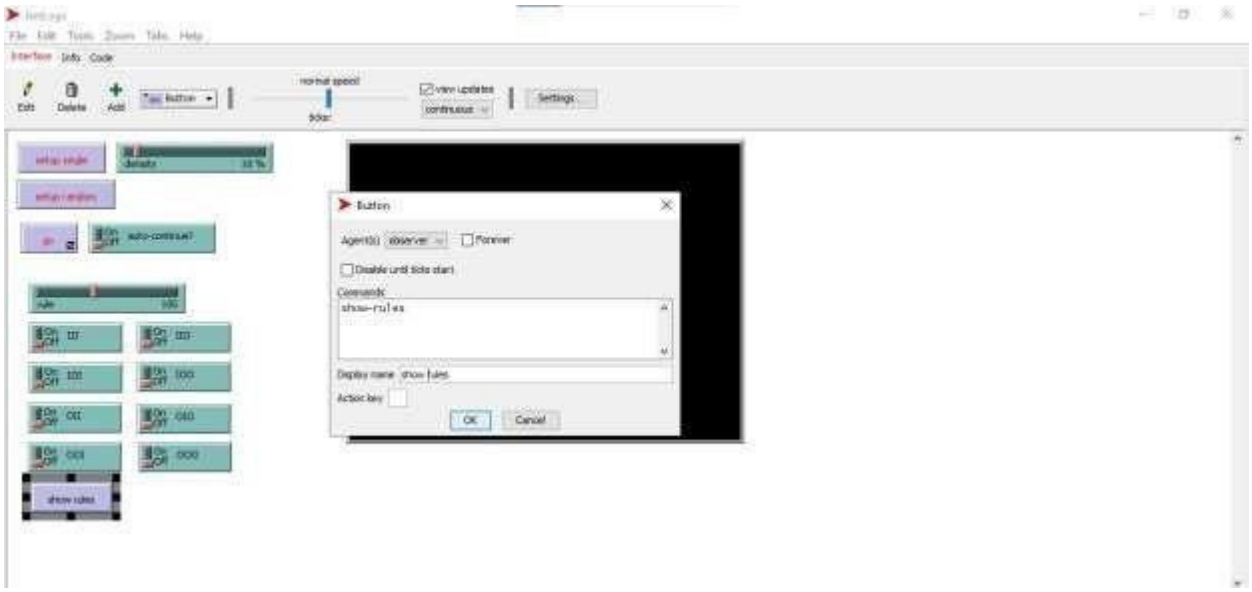
- Commands: setup-random o Display Name: setup random



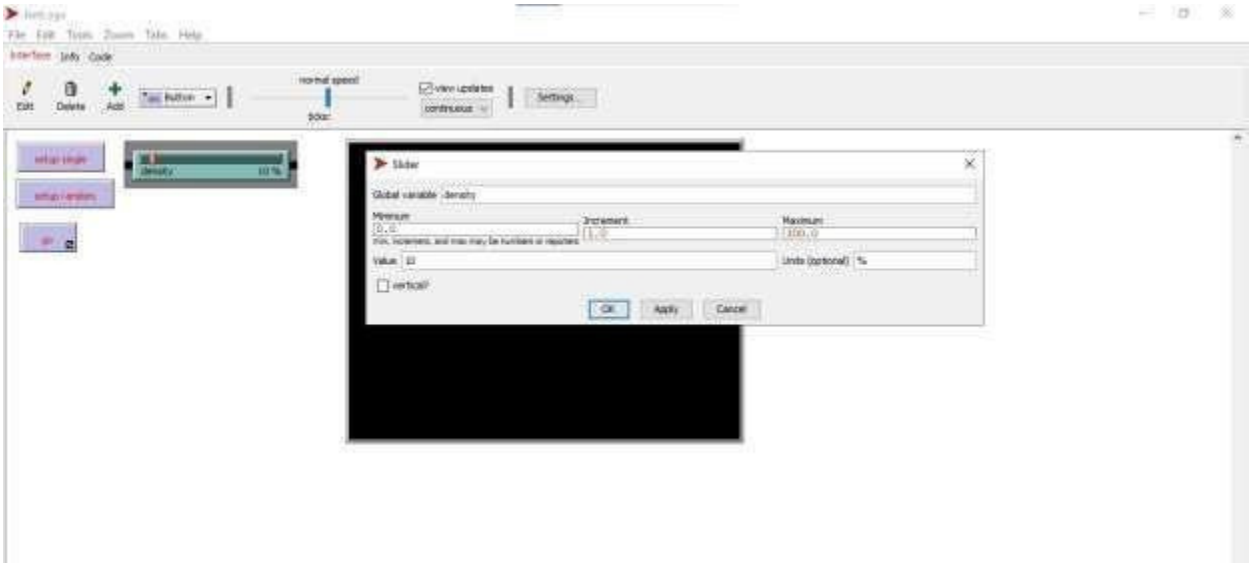
  - go (forever button)



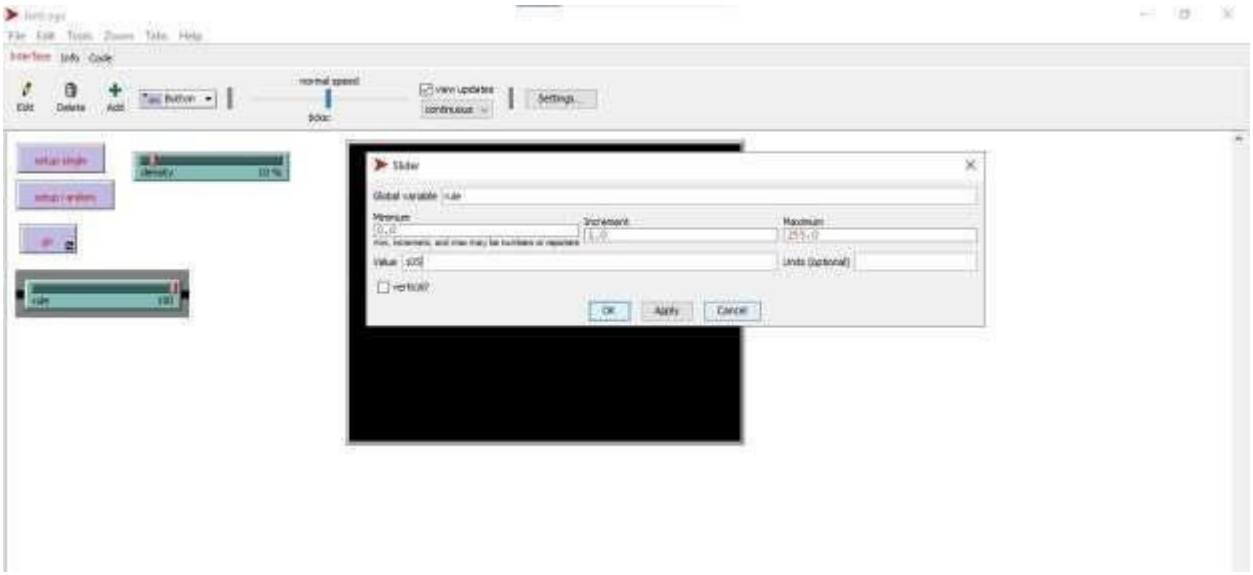
- Commands: show-rules ○ Display Name: show rules



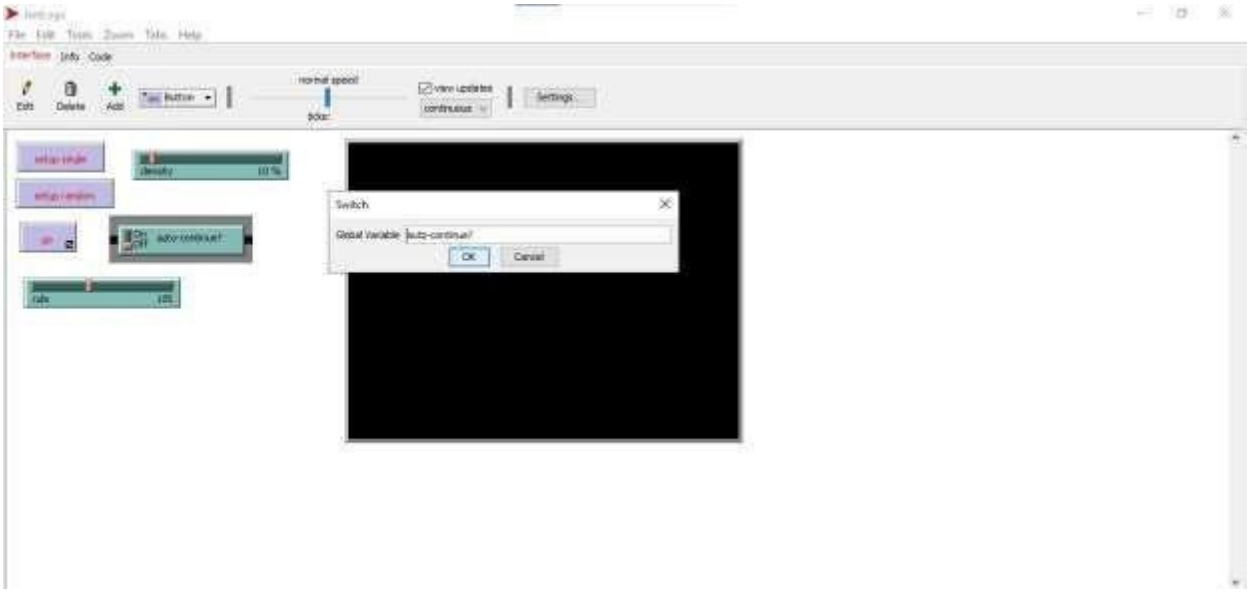
- Slider:
  - density (0.0–100.0, default 10 and increment 1.0)



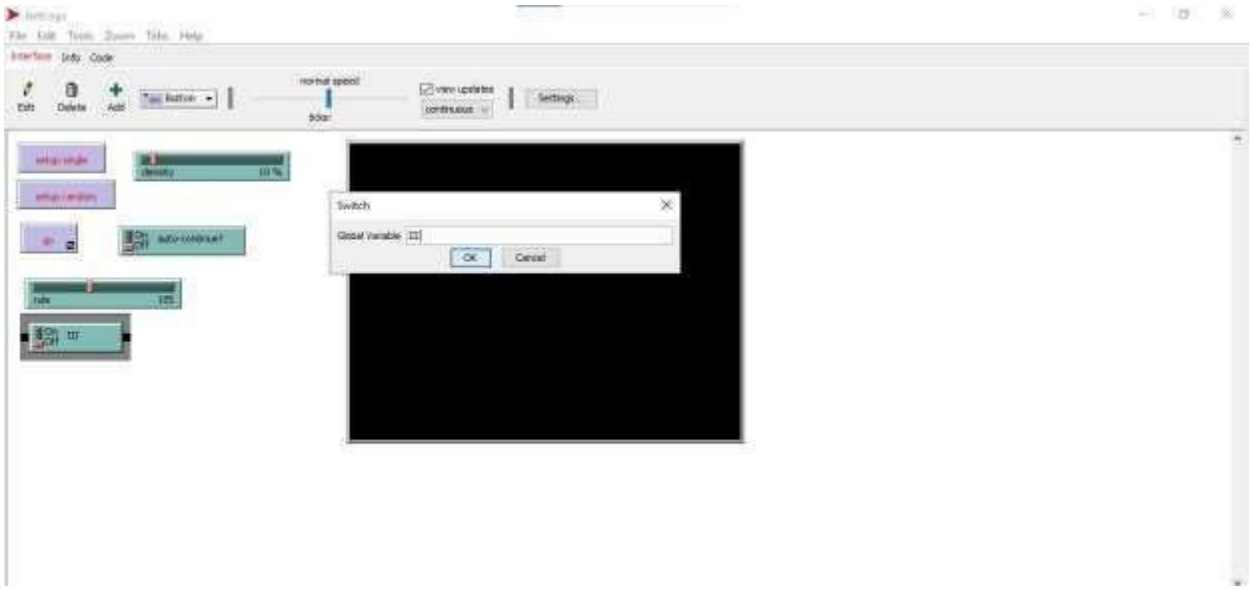
  - rule (0.0–255.0, default 105 and increment 1.0)



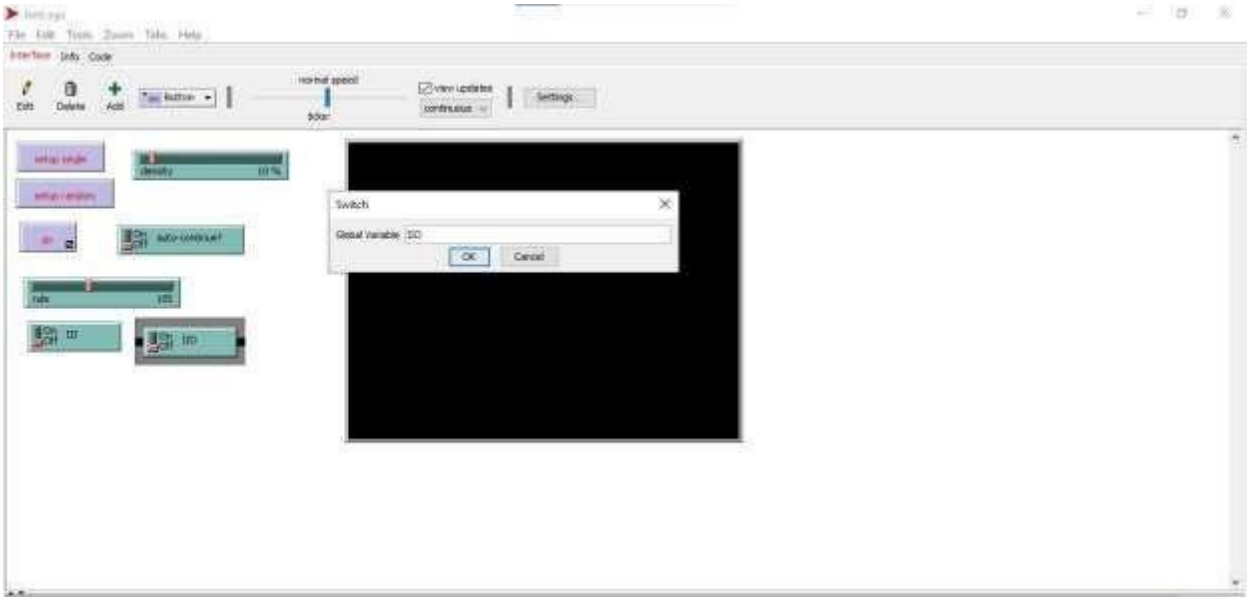
- Switch:
  - auto-continue?



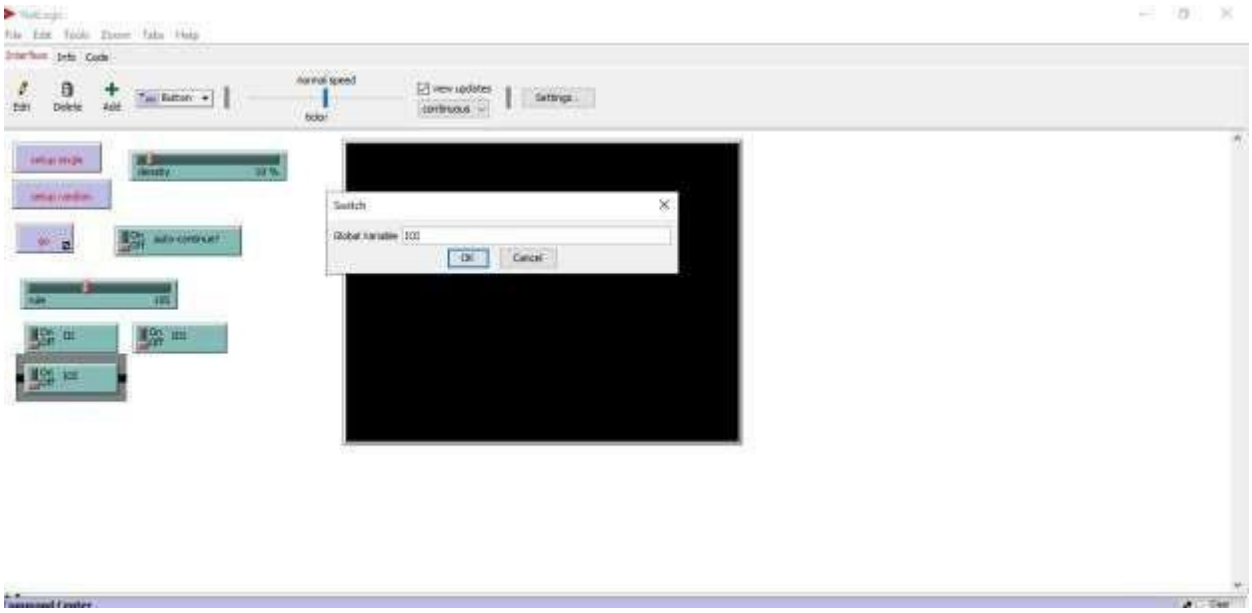
- switches: ○ 111


Switch
Global Variable
OK
Cancel

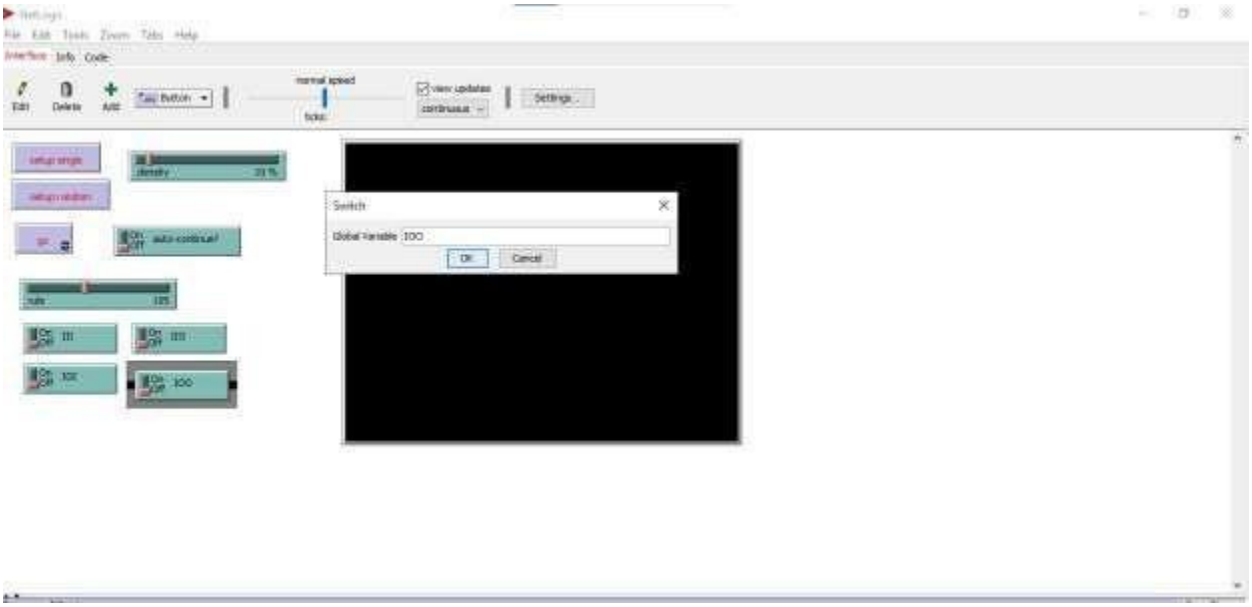
- 110


Switch
Global Variable
OK
Cancel

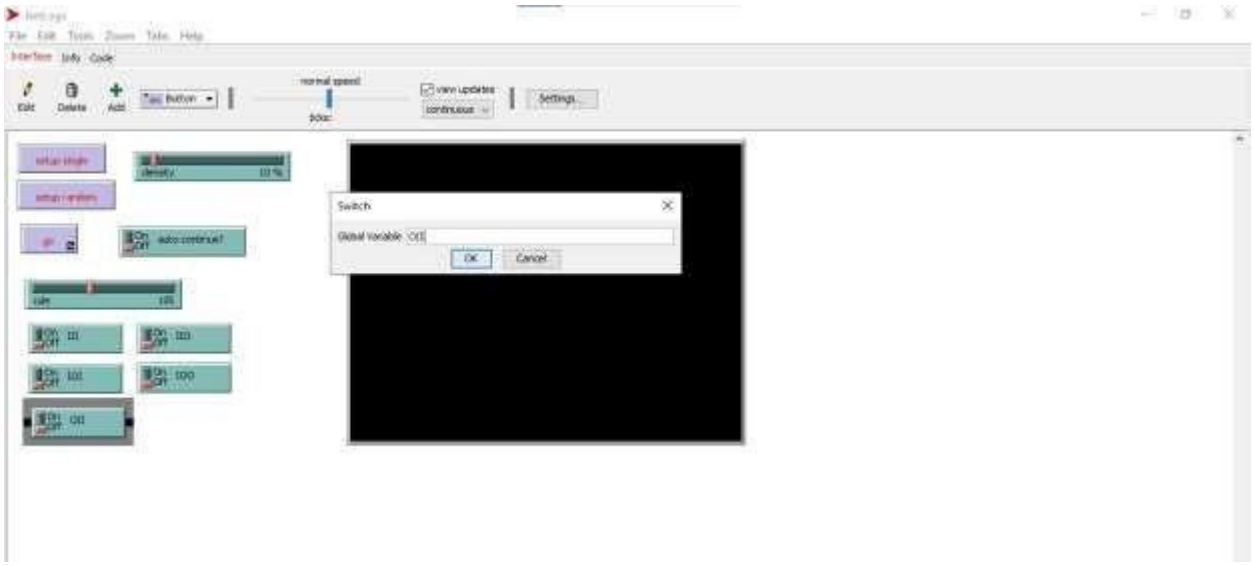
- 101


Switch
Global Variable 100
OK
Cancel

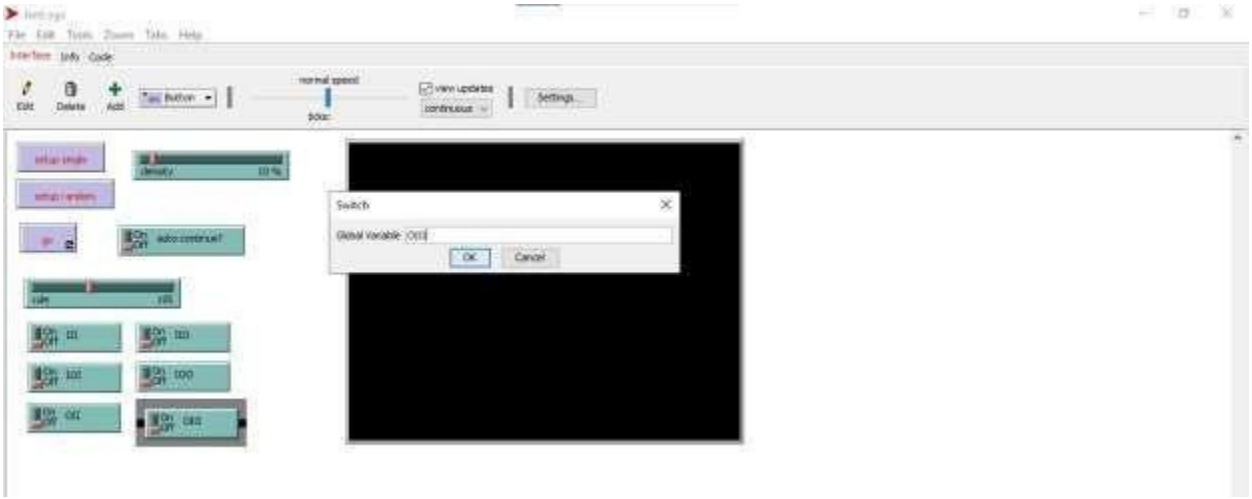
- 100


Switch
Global Variable 100
OK
Cancel

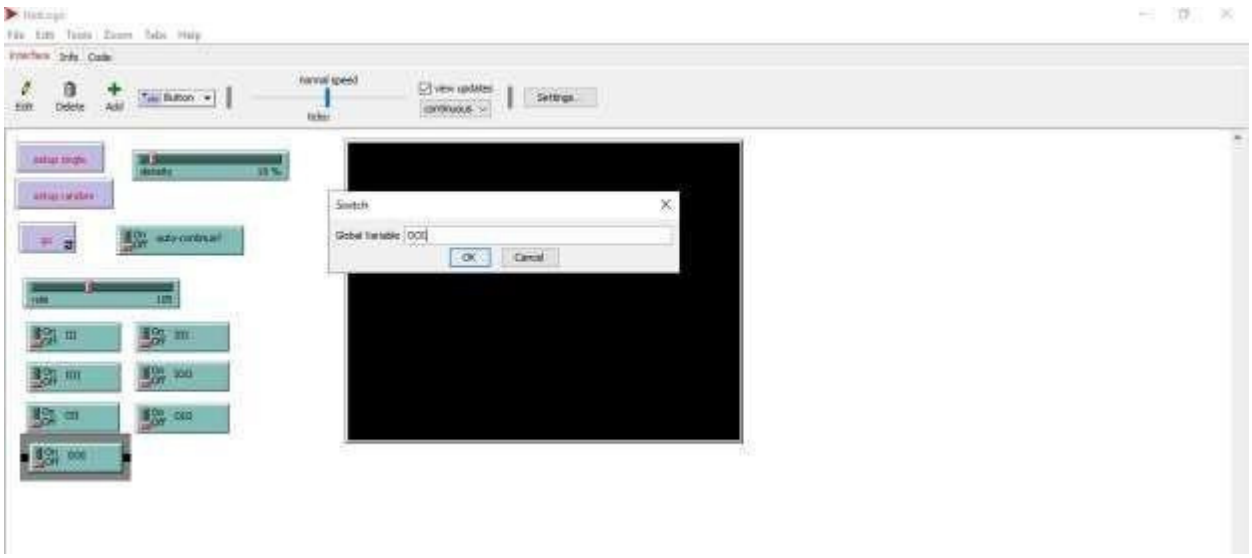
- 011


Switch
Global Variable
OK
Cancel

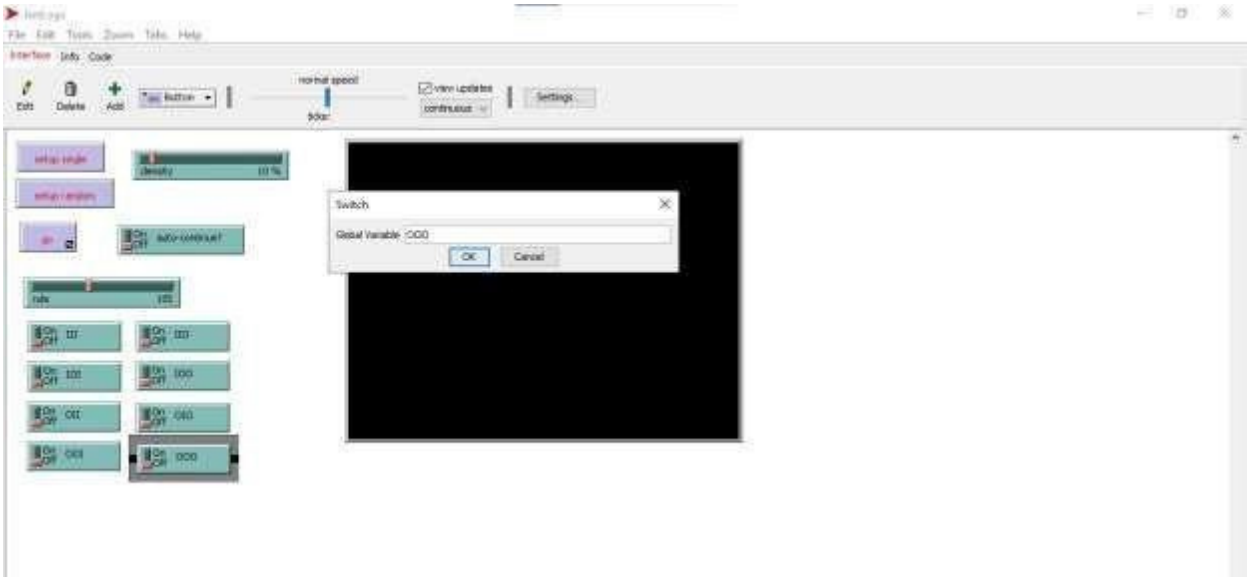
- 010


Switch
Global Variable
OK
Cancel

- 001

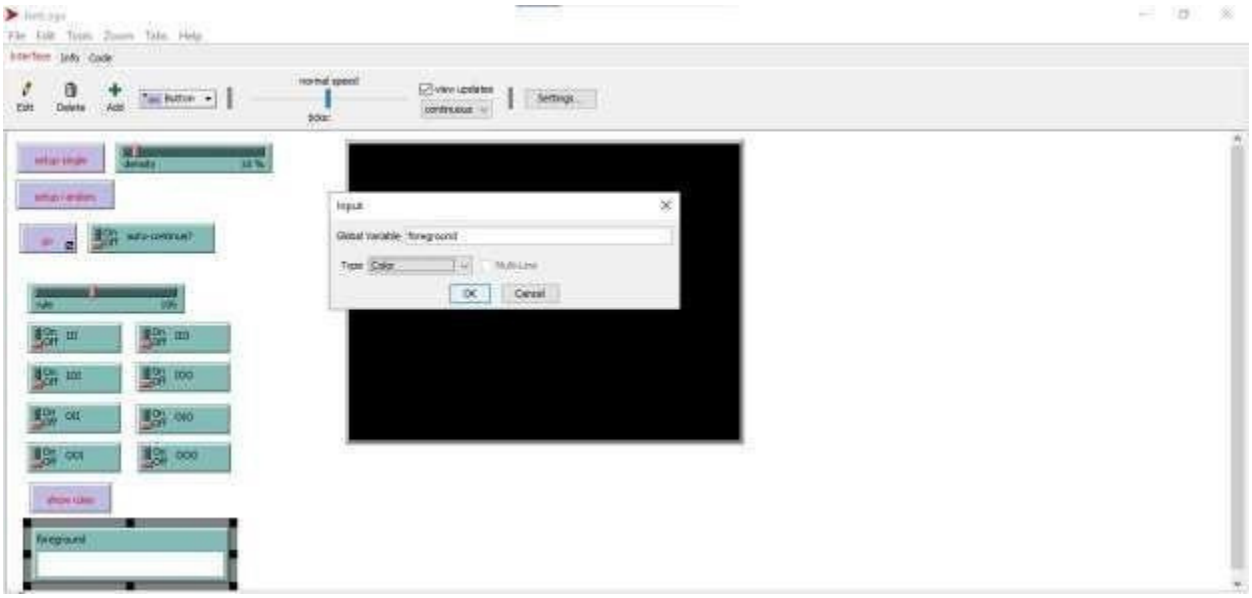


o 000

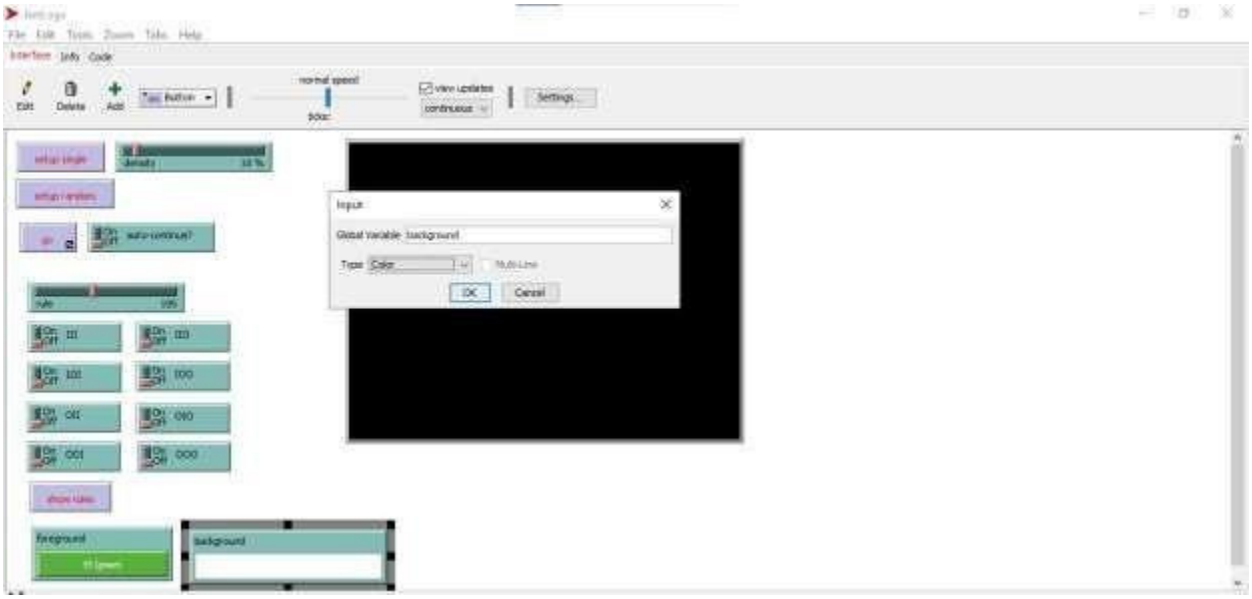


Color choosers: o

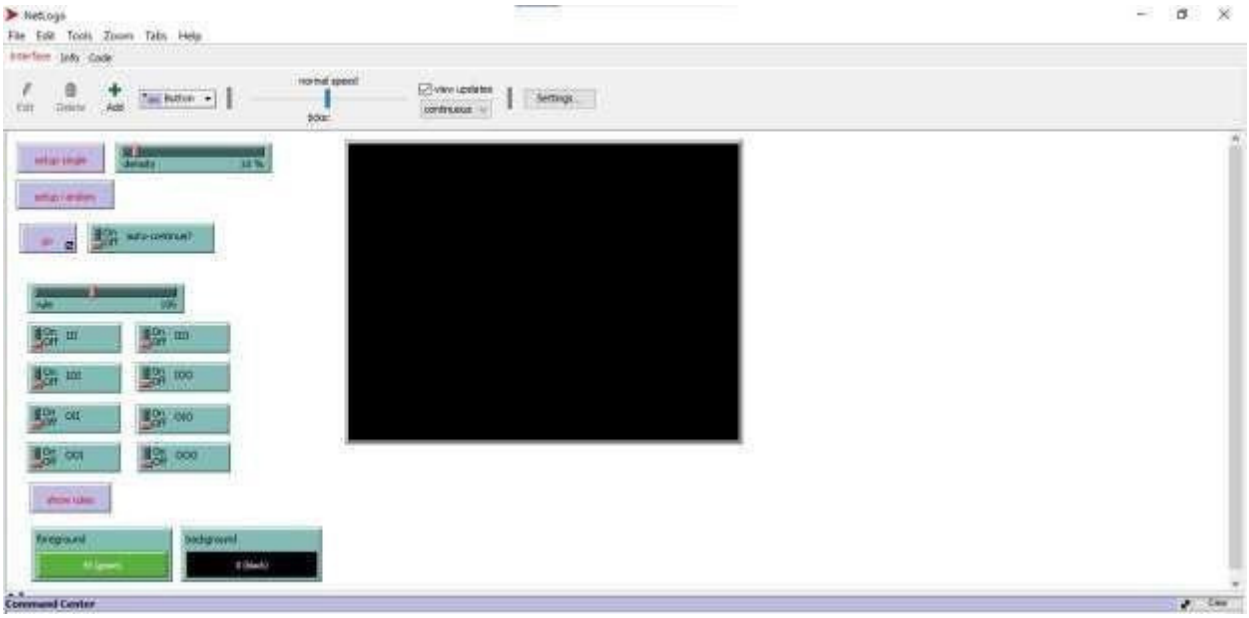
Foreground



- Background



Set the world background to black and foreground to green.



Step 2: Add the Full Code in the Code Tab

```
globals
[ row ;; current row old-rule ;; previous rule rules-shown? ;;
flag to check if rules have been displayed gone? ;; flag to
check if go has already been pressed
]


patches-own
[on?]

to startup ;; initially, nothing has been displayed
set rules-shown? false set gone? false set
old-rule rule end

;;;;;;;;;;;;;;;;;;;;;;;;;
;;; Setup Procedures ;;;
;;;;;;;;;;;;;;;;;;;;;;;;;
to setup-general ;; setup general working environment clear-patches clear-turtles set
    row max-pycor ;; reset current row refresh-rules set gone? false set rules- shown?
    false ;; rules are no longer shown since the view has been cleared end
```

```
to single-cell setup-general reset-ticks ask patches with [pycor = row] [set on? false set pcolor
    background] ;; initialize top row ask patch 0 row [ set pcolor foreground ] ;; setup single
    cell in top center ask patch 0 row [ set on? true ] end

to setup-random setup-general reset-ticks ask patches with [pycor = row] ;; randomly
    place cells across the top of the world
 [
  set on? ((random-float 100) < density)     color-patch
 ]

end


to setup-continue let on?-list [] if not gone? ;; make sure
    go has already been called
  [ stop ]
 ;; copy cell states from the current row to a list set on?-list map [ p -> [
    on? ] of p ] sort patches with [ pycor = row ] setup-general ask
    patches with [ pycor = row ]
 [
  set on? item (pxcor + max-pxcor) on?-list ;; copy states from list to top row      color-patch
 ]

end


;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; GO Procedures      ;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;

to go if (rules-shown?) ;; don't do unless we are properly set up
  [ stop ] if (row = min-pycor) ;; if we reach the end, continue from the top or
    stop
 [
  ifelse auto-continue?
  [
     set gone? true        display ;; ensure everything gets drawn
    before we clear it       setup-continue
  ]
```

```
  [
    ifelse gone?
      [ setup-continue ] ;; a run has already been completed, so continue with another [ set
    gone? true stop ] ;; otherwise just stop
  ]
]
ask patches with [ pycor = row ] ;; apply rule
  [ do-rule ] set row (row - 1) ask patches with [ pycor = row ]
    ;; color in changed cells
  [ color-patch ] tick

end


to do-rule ;; patch procedure let left-on? [on?] of patch-at -1 0 ;; set to true if the
   patch to the left is on let right-on? [on?] of patch-at 1 0 ;; set to true if the
   patch to the right is on

 ;; each of these lines checks the local area and (possibly) ;;
 sets the lower cell according to the corresponding switch
 let new-value
   (iii and left-on? and on? and right-on?)       or
   (iio and left-on?      and on?          and     (not     right-on?))     or
          (ioi and left-on?         and (not on?) and right-on?) or
   (ioo and left-on?     and (not on?) and (not right-on?)) or (oii and (not
     left-on?) and on? and right-on?) or
   (oio and (not left-on?) and on?       and (not right-on?)) or
     (ooi and (not left-on?) and (not on?) and right-on?)         or
     (ooo and (not left-on?) and (not on?) and (not right-on?))
     ask patch-at 0 -1 [ set on? new-value ] end


;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; Utility Procedures ;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;
to color-patch ;;patch procedure ifelse on?
   [ set pcolor foreground ]     [
     set pcolor background ]
     end
```

to-report bindigit [number power-of-two] ifelse
(power-of-two = 0)
[report floor number mod 2]
[report bindigit (floor number / 2) (power-of-two - 1)] **end**

to refresh-rules ;; update either switches or slider depending on which has been changed last ifelse (rule
= old-rule)
[
if (rule != calculate-rule)
[set rule calculate-rule]
]
[extrapolate-switches] set
old-rule rule **end**

to extrapolate-switches
;;set the switches based on the slider set
ooo ((bindigit rule 0) = 1) set ooi
((bindigit rule 1) = 1) set oio ((bindigit
rule 2) = 1) set oii
((bindigit rule 3) = 1) set ioo
((bindigit rule 4) = 1) set ioi
((bindigit rule 5) = 1) set iio
((bindigit rule 6) = 1) set iii
((bindigit rule 7) = 1) **end**

to-report calculate-rule
;;set the slider based on the switches
let result 0 if ooo [set result result +
1] if ooi [set result result + 2] if oio
[set result result + 4] if oii [set
result result + 8] if ioo [set result
result + 16] if ioi [set result result +
32] if iio [set result result
+ 64] if iii [set result result + 128
] report result **end**

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; SHOW-RULES RELATED PROCEDURES ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

```
to show-rules ;; preview cell state transitions
setup-general ask patches with [pycor > max-
pycor - 12]
   [ set pcolor gray - 1 ] let i 0 foreach list-rules [ the-rule ->           let px (min-
     pxcor + i * floor (world-width / 8) + floor (world-width / 16)) - 4         ask
     patch px (max-pycor - 4)
   [
     sprout 1
     [
       set xcor xcor - 3        print-block item 0 the-
     rule ;; left cell       set xcor xcor + 3 print-
     block item 1 the-rule ;; center cell set xcor
     xcor + 3      print-block item 2 the- rule ;; right
     cell           setxy (xcor - 3) (ycor - 3) print-
     block item 3 the-rule ;; next cell state die
     ]    ]    set i
     (i + 1)
 ]
 set rules-shown? true end

;; turtle procedure to print-block [print-on?] ;; draw a 3x3 block with the color
     determined by the state ask patches in-radius 1.5 ;; like "neighbors", but includes the
     patch we're on too
 [
   set on? print-on?
   color-patch
 ]

end
to-report list-rules ;; return a list of state-transition 4-tuples corresponding to the switches report (list
     lput ooo [false false false] lput ooi [false false true ] lput oio


      [false true false]     lput oii [false true true ]       lput ioo [truefalse false ]lput ioi [ true
                    false true ] lput iio[ true true false]       lputiii[  true true
     true ])

end
```


NetLogo
File Edit Tools Zoom Tabs Help
Interface Info Code
Find... Check
Procedures
Indent automatically
Code Tab in separate window
globals
[
row ;; current row
old-rule ;; previous rule
rules-shown? ;; flag to check if rules have been displayed
gone? ;; flag to check if go has already been pressed
]
patches-own
[on?]
to startup ;; initially, nothing has been displayed
set rules-shown? false
set gone? false
set old-rule rule
end
;;;;;;;;;;;;;;;;;;;;;;;;
;;; Setup Procedures ;;;
;;;;;;;;;;;;;;;;;;;;;;;;
to setup-general ;; setup general working environment
clear-patches
clear-turtles
set row max-pycor ;; reset current row
refresh-rules
set gone? false
set rules-shown? false ;; rules are no longer shown since the view has been cleared
end
to single-cell
setup-general
reset-ticks
ask patches with [pycor = row] [set on? false set pcolor background] ;; initialize top row
ask patch 0 row [set pcolor foreground] ;; setup single cell in top center
ask patch 0 row [set on? true]
end

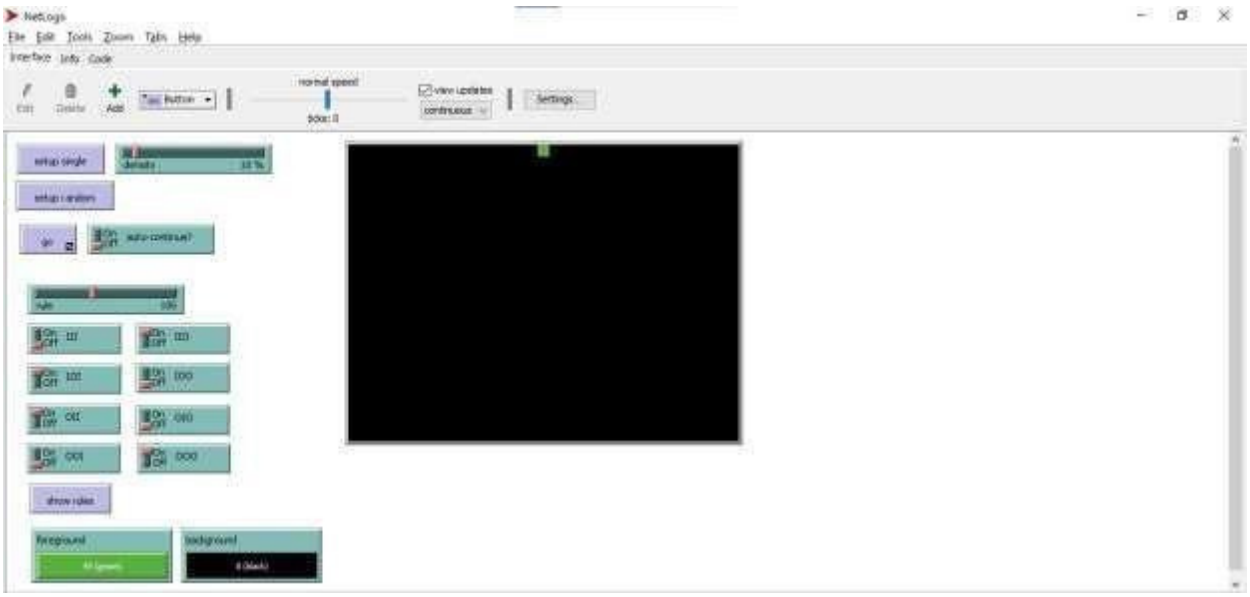

NetLogo
File Edit Tools Zoom Tabs Help
Interface Info Code
Find... Check
Procedures
Indent automatically
Code Tab in separate window
to setup-random
setup-general
reset-ticks
ask patches with [pycor = row] ;; randomly place cells across the top of the world
[
set on? ((random-float 100) < density)
color-patch
]
end
to setup-continue
let on?-list []
if not gone? ;; make sure go has already been called
[stop]
;; copy cell states from the current row to a list
set on?-list map [p -> [on?] of p] sort patches with [pycor = row]
setup-general
ask patches with [pycor = row]
[
set on? item (pxcor + max-pxcor) on?-list ;; copy states from list to top row
color-patch
]
end
;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; GO Procedures ;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;


NetLogo
File Edit Tools Zoom Tabs Help
Interface Info Code
Find... Check
Procedures
Indent automatically
Code Tab in separate window
to do-rule ;; patch procedure
let left-on? [on?] of patch-at -1 0 ;; set to true if the patch to the left is on
let right-on? [on?] of patch-at 1 0 ;; set to true if the patch to the right is on
;; each of these lines checks the local area and (possibly)
;; sets the lower cell according to the corresponding switch
let new-value
(iii and left-on? and on? and right-on?) or
(iio and left-on? and on? and (not right-on?)) or
(ioi and left-on? and (not on?) and right-on?) or
(ioo and left-on? and (not on?) and (not right-on?)) or
(oii and (not left-on?) and on? and right-on?) or
(oio and (not left-on?) and on? and (not right-on?)) or
(ooi and (not left-on?) and (not on?) and right-on?) or
(ooo and (not left-on?) and (not on?) and (not right-on?))
ask patch-at 0 -1 [set on? new-value]
end
;;;;;;;;;;;;;;;;;;;;;;;;;
;;; Utility Procedures ;;;
;;;;;;;;;;;;;;;;;;;;;;;;;
to color-patch ;;patch procedure
ifelse on?
[set pcolor foreground]
[set pcolor background]
end
to-report bindigit [number power-of-two]
ifelse (power-of-two = 0)
[report floor number mod 2]
[report bindigit (floor number / 2) (power-of-two - 1)]
end

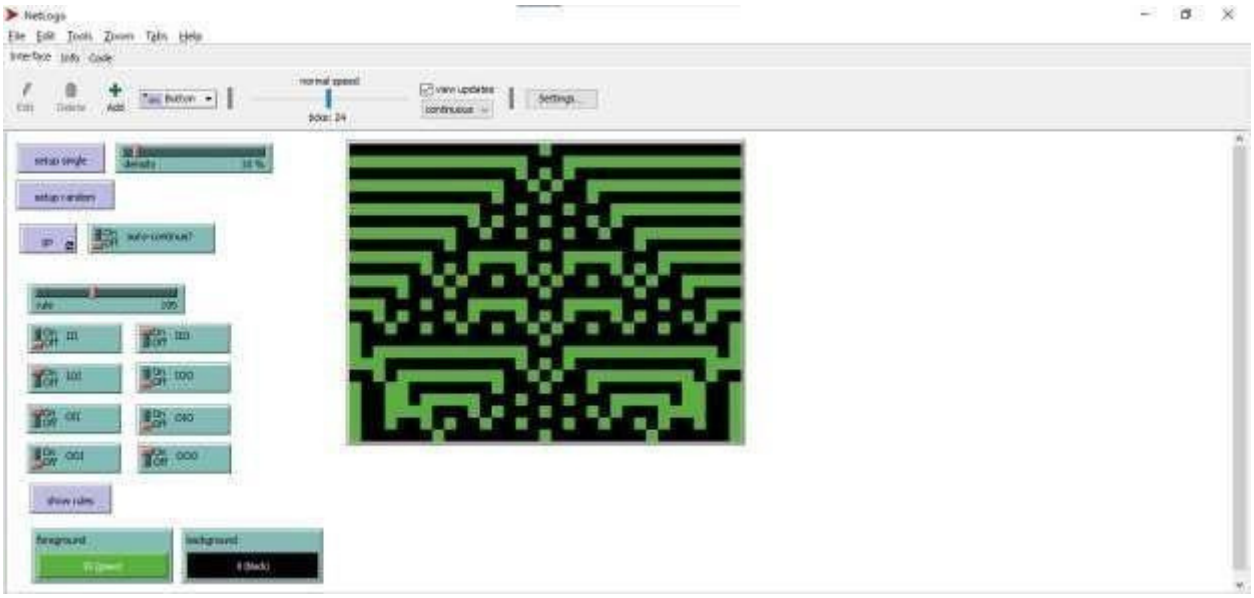

NetLogo
File Edit Tools Zoom Tabs Help
Interface Info Code
Find... Check
Procedures
Indent automatically
Code Tab in separate window
to refresh-rules ;; update either switches or slider depending on which has been changed last
ifelse (rule = old-rule)
[
if (rule != calculate-rule)
[set rule calculate-rule]
]
[extrapolate-switches]
set old-rule rule
end
to extrapolate-switches
;; set the switches based on the slider
set ooo ((bindigit rule 0) = 1)
set ooi ((bindigit rule 1) = 1)
set oio ((bindigit rule 2) = 1)
set oii ((bindigit rule 3) = 1)
set ioo ((bindigit rule 4) = 1)
set ioi ((bindigit rule 5) = 1)
set iio ((bindigit rule 6) = 1)
set iii ((bindigit rule 7) = 1)
end
to-report calculate-rule
;; set the slider based on the switches
let result 0
if ooo [set result result + 1]
if ooi [set result result + 2]
if oio [set result result + 4]
if oii [set result result + 8]
if ioo [set result result + 16]
if ioi [set result result + 32]
if iio [set result result + 64]
if iii [set result result + 128]
report result
end

Output

Select setup single



Select go

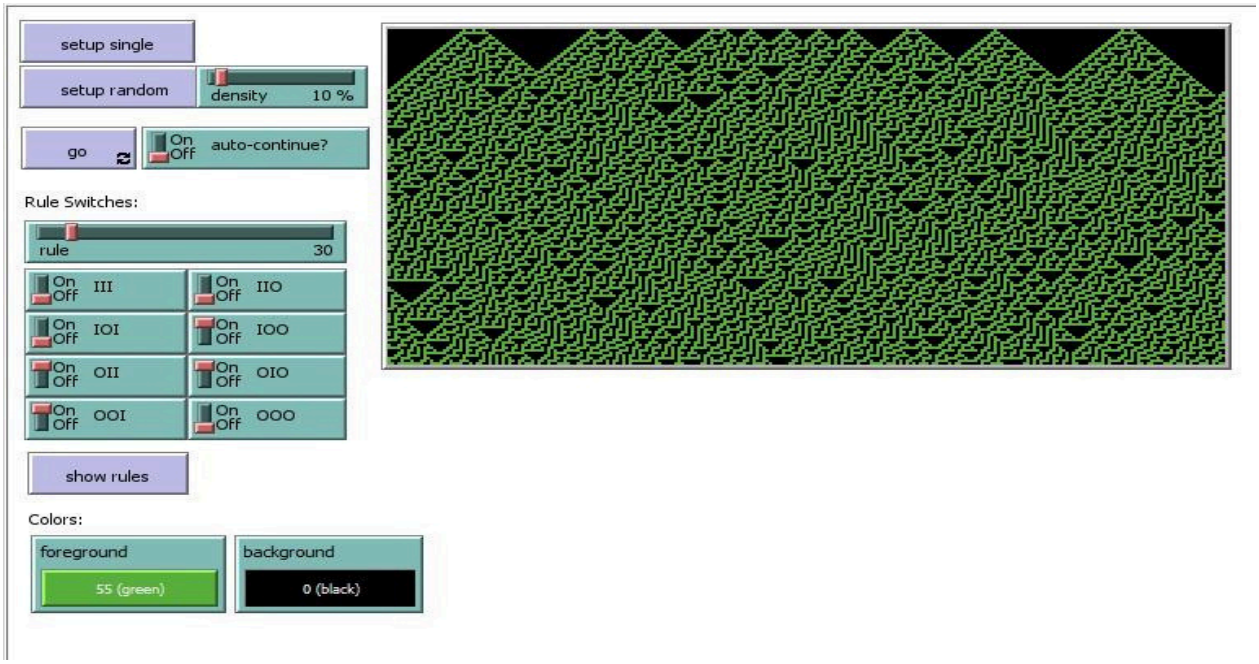


How to Run the Model

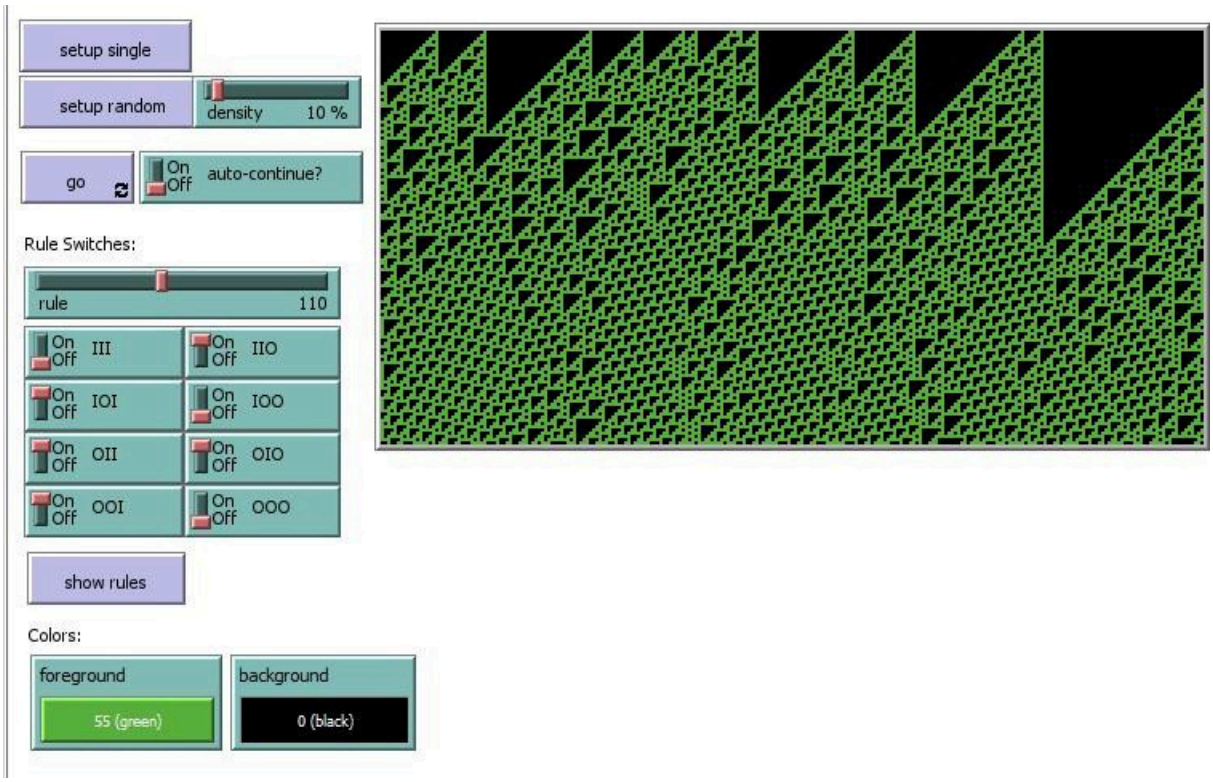
1. Select a rule-number (e.g., 30, 90, 110).
2. Click setuptoinitialize the grid.
3. Click go togenerate the CA pattern over time.
4. Use go-oncetoobserve step-by-step evolution.
5. Observehowchanging the rule changes the overall pattern.

STUDENT TASK

1. Runthesimulationusing Rule 30, Rule 90, and Rule 110.


setup single
setup random
density 10 %
go
On Off auto-continue?
Rule Switches:
rule 30
On Off III
On Off IIO
On Off IOI
On Off IOO
On Off OII
On Off OIO
On Off OOI
On Off OOO
show rules
Colors:
foreground
55 (green)
background
0 (black)


setup single
setup random
density 10 %
go
On Off auto-continue?
Rule Switches:
rule 90
On Off III
On Off IIO
On Off IOI
On Off IOO
On Off OII
On Off OIO
On Off OOI
On Off OOO
show rules
Colors:
foreground
55 (green)
background
0 (black)



Observation:

- **Rule 30:** Produces a chaotic, asymmetric pattern. Small initial changes lead to complex, seeminglyrandom structures.
- **Rule 90:** Generates a regular, symmetric triangular pattern. Glider-like structures appear alongdiagonals.
- **Rule 110:** Produces complex patterns with both periodic and chaotic regions. Supports movingstructures that can interact, showing behavior capable of universal computation.

2. Comparethevisual patterns generated by each rule.

Observation:

- **Rule 30:** Chaotic and asymmetric; no clear repetition; looks random.
- **Rule 90:** Highly regular and symmetric; forms triangular, fractal-like patterns.
- **Rule 110:** Complex mix of periodic and chaotic regions; shows both stable and moving structures.

The differences highlight how simple rule changes in 1D cellular automata can produce drastically different behaviors, from complete order to apparent randomness.

3. Change the initial cell value and observe the difference.

- Starting with a **single cell in the top center** produces the classic patterns for each rule (Rule 30 chaotic, Rule 90 symmetric, Rule 110 complex).
- Starting with **multiple or random cells** at the top row changes the evolution:
  - **Rule 30:** Appears more filled and irregular. o **Rule 90:** Maintains symmetry but shows overlapping triangles. o **Rule 110:** More complex interactions and additional moving structures appear.

  The initial configuration significantly affects the resulting pattern, demonstrating sensitivity to starting conditions in cellular automata.

4. Identify whether the pattern is regular, chaotic, or complex.

- **Rule 30:** Chaotic no repeating pattern, unpredictable behavior.
- **Rule 90:** Regular highly symmetric and repetitive triangular patterns.
- **Rule 110:** Complex —contains both periodic structures and chaotic regions, with moving patterns interactingin non-trivial ways.

5. Stop the simulation afterseveral ticks and analyze symmetry or randomness.

- **Rule 30:** Asymmetricand random-looking; no visible symmetry, patterns appear irregular.
- **Rule 90:** Symmetric;patterns maintain clear triangular repetition across the grid.
- **Rule 110:** Partiallysymmetric; contains both ordered regions and irregular areas, showing a mix of periodic andchaotic behavior.

  Stopping after severalticks highlights the characteristic structure of each rule and shows howinitialconditionsand rule choice affect the evolution of patterns.

POST LAB QUESTIONS

1. What is an elementary cellular automaton?

An elementary cellular automaton (ECA)is a one-dimensional grid of cells where each cell has two possible states (on/off or 1/0). The state of each cell in the next generation depends only on its current state and the state of its immediate neighbors, following a simple deterministic rule.

2. Why are there exactly 256 possible elementary CA rules?

Each cell has 3 neighbors (left, center, right), giving $2^3 = 8$ possible neighborhood patterns. Each pattern can produce either 0 or 1 in the next state. Therefore, the total number of rules is $2^8 = 256$.

3. Describe the behavior of Rule 30 and Rule 90.

- **Rule 30:** Chaotic and asymmetric; producesseemingly random patterns even from a single initial cell.
- **Rule 90:** Regular and symmetric; generatestriangular, fractal-like patterns that repeat predictably.

4. How does changing the initial condition affect the pattern?

The initial configuration of the top row (single cell or multiple/random cells) significantly affects the resulting evolution. Some rules (like Rule 30) amplify randomness, while others (like Rule 90) preserve symmetry but may create overlapping structures.

5. Why is Rule 110 considered important in computational theory?

Rule 110 is capable of universal computation; it can simulate any Turingmachine. Its patterns include both stable and moving structures, demonstrating how simple rules can produce complex, computable behavior.