MINISTRY OF EDUCATION AND TRAINING

HO CHI MINH UNIVERSITY OF SCIENCE

FACULTY OF INFORMANTION TECHNOLOGY _ K20

Artificial Intelligence

Project 1: Searching Solver with NetLogo

Theory lecturer: Nguyễn Ngọc Thảo

Practical lecturer:

Lê Ngọc Thành,

Hồ Thị Thanh Tuyến,

Nguyễn Hải Đăng

Student:

Trần Tiến Hoàng – 20127424

Lâm Kim Nhân – 20127579

Lê Võ Huỳnh Thanh – 20127072

Class: 20CLC09

Table of Contents

Assignment Plan

ID	Name	Job Description	Contribution
20127424	Trần Tiến Hoàng	Do requirement 2 (all 3 levels)	34%
20127579	Lâm Kim Nhân	Do requirement 1 Run test cases Record a video running 5 test cases	33%
20127072	Lê Võ Huỳnh Thanh	Edit and help requirement 1 Create map Run test cases Write project report.	33%

Self-assessment for completion level for each requirement.

No.	Criteria	Complete
1	Requirement 1	✓
2	Requirement 2: Level 1	✓
3	Requirement 2: Level 2	✓
4	Requirement 2: Level 3	✓

5	Generate at least 5 test cases for each level with different attributes. Describe them in the experiment section of your report. Videos to demonstrate.	✓
6	Report your algorithms, experiments with some reflection or comments.	✓

I. Complete

- 100% project

II. What we learn

- Teamwork skill.
- New language NetLogo
- Time management skills
- Assign tasks to meet deadlines
- Discussion skills
- Exchange skills
- The ability to understand

III. Detailed report

1. Requirement 1:

Some information about Netlogo.

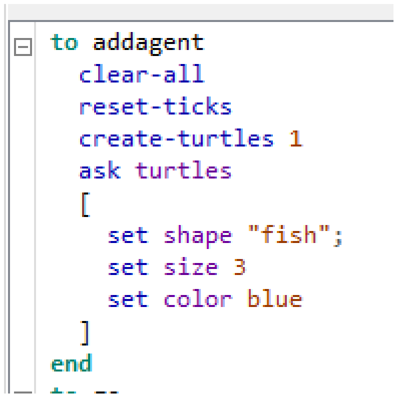
- Uri Wilensky created NetLogo to have "low threshold and no ceiling," in the spirit of the programming language Logo. It uses turtles, patches, links, and the observer as agents to teach programming principles. [1] NetLogo was created with a variety of users in mind, including educators educating children and domain experts without a programming experience who want to mimic relevant phenomena. [2] NetLogo has been used in a number of scholarly papers.
- The NetLogo environment enables exploration of emergent phenomena.
- NetLogo comes with a wealth of documentation and tutorials. It also includes the Models Library, which has a huge number of ready-to-use and modify simulations. These simulations cover topics in biology and medicine, physics and chemistry, mathematics and computer science, economics, and social psychology in the natural and social sciences. Several NetLogo-based model-based inquiry courses are available, with more in the works.
- NetLogo is free and open-source software that is distributed under the terms of the GNU General Public License (GPL). Also available are commercial licenses. It's written in Scala and Java, and it runs on the JVM (JVM). A hybrid interpreter/compiler is at its heart, compiling user code to JVM bytecode in part.

Some features of Netlogo:

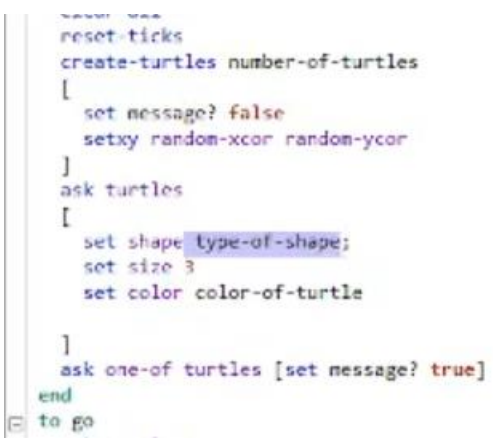
- Free, open source.

- Language is Logo dialect extended to support agents
- Runs are reproducible cross-platform
- Link agents connect turtles to make networks, graphs, and aggregates
- Interface builder w/ buttons, sliders, switches, choosers, monitors, text boxes, notes, output area
- Headless mode allows doing batch runs from the command line
- Controlling API allows embedding NetLogo in a script or application
- Extensions API allows adding new commands and reporters to the NetLogo language; open source example extensions are included
  - Show agents with shapes like turtles, rabbits, humans, etc.

To show the agents with shape, we do this code:



```
to addagent
  clear-all
  reset-ticks
  create-turtles 1
  ask turtles
  [
    set shape "fish";
    set size 3
    set color blue
  ]
end
```



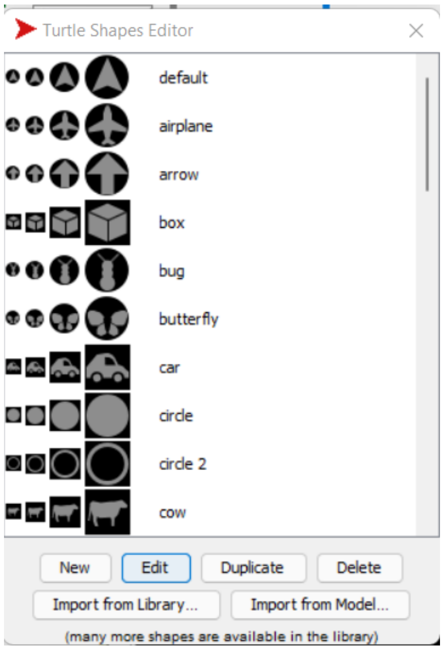
```
  reset-ticks
  create-turtles number-of-turtles
  [
    set message? false
    setxy random-xcor random-ycor
  ]
  ask turtles
  [
    set shape type-of-shape;
    set size 3
    set color color-of-turtle

  ]
  ask one-of turtles [set message? true]
end
to go
```

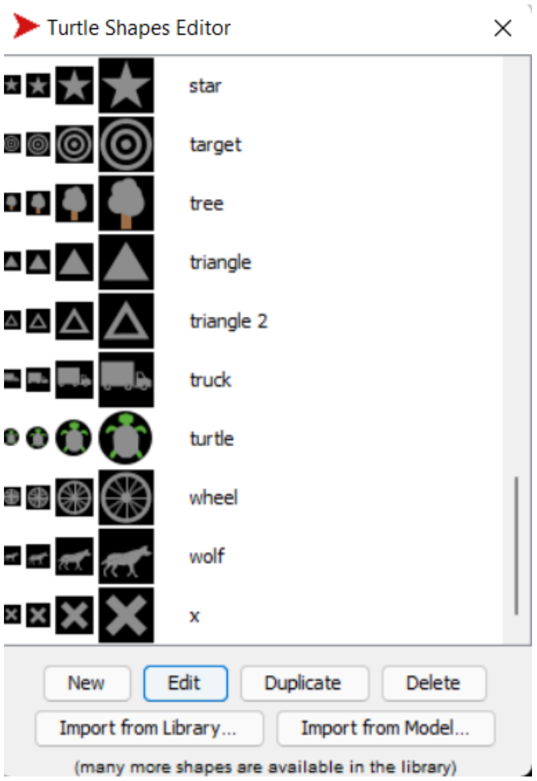
Clear-all: clear all the agent

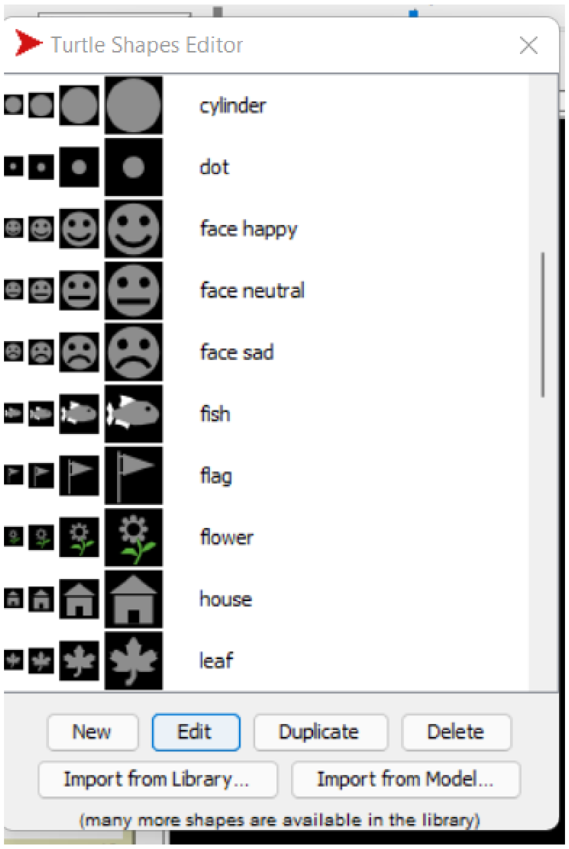
Reset-tick: reset the memory

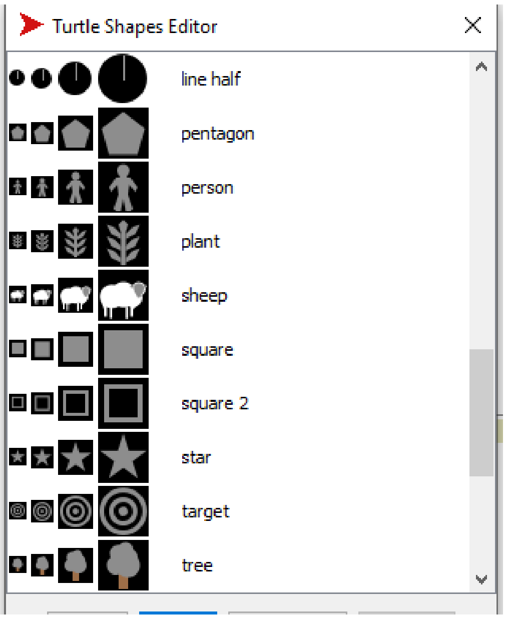
Create-turtles: create agent

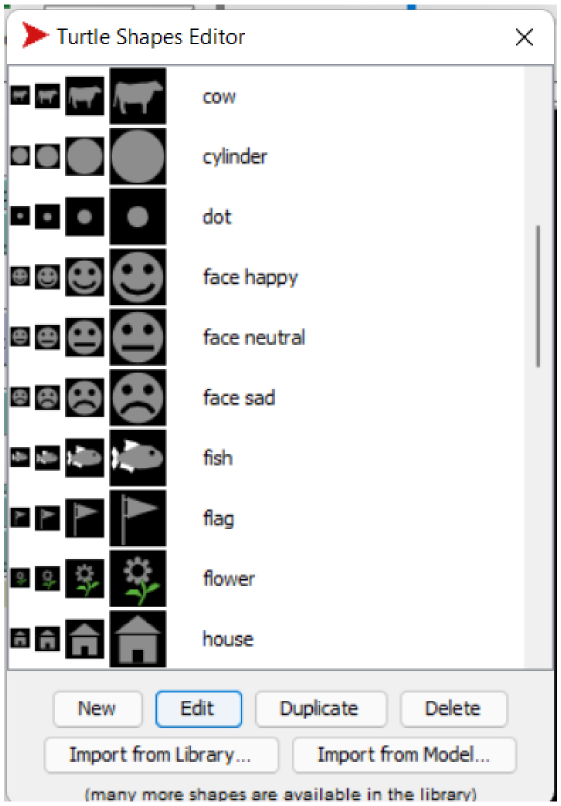
We can change the shape of the agent as we want in Tools => turtle shapes editor(we can use other shapes if we have it's model, we can change it by rename the shape's name by name in following box).





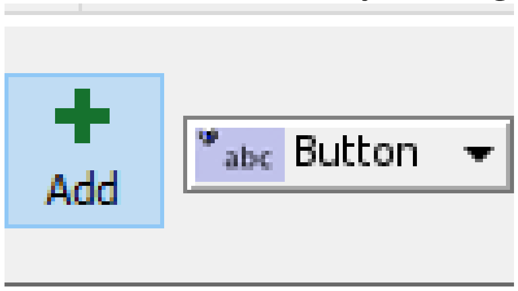

Turtle Shapes Editor
cylinder
dot
face happy
face neutral
face sad
fish
flag
flower
house
leaf
New
Edit
Duplicate
Delete
Import from Library...
Import from Model...
(many more shapes are available in the library)

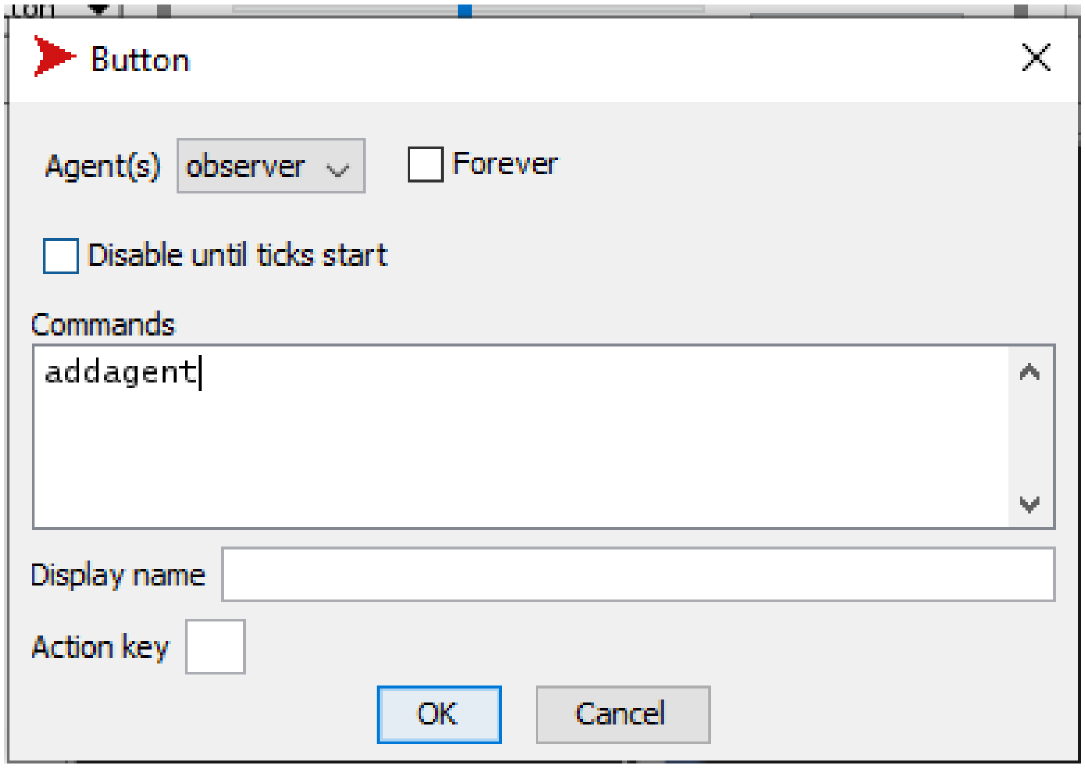

Turtle Shapes Editor
line half
pentagon
person
plant
sheep
square
square 2
star
target
tree



(and more)

- Show buttons to manipulate the simulator environment
  We can add button by clicking add.



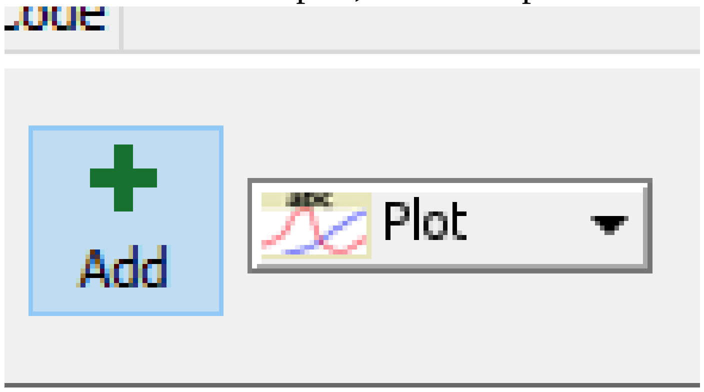


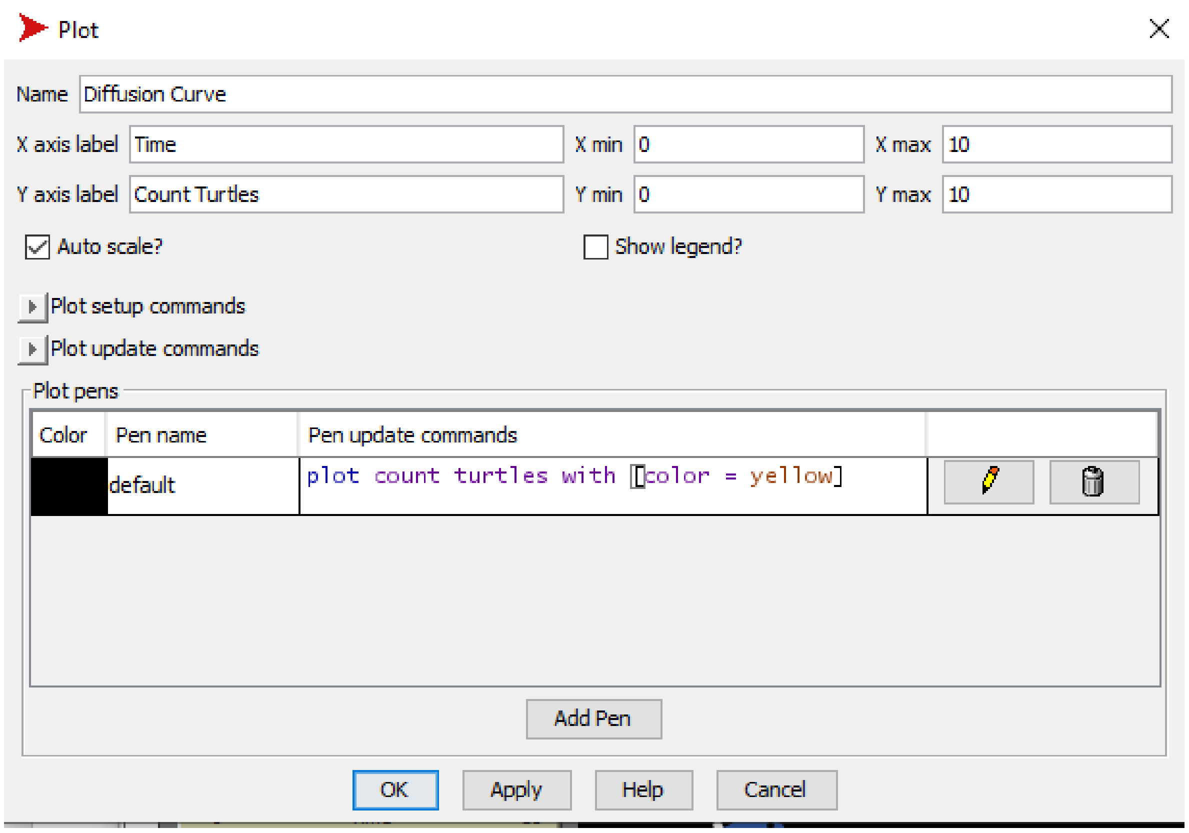
```
addagent
```

In button box, the “Commands” is the code you want to run, the display name to change the name. Then if you want it to run recursively, click forever.

- Showing running parameters of agents through plots

If we want to draw plot, select the plot then add.



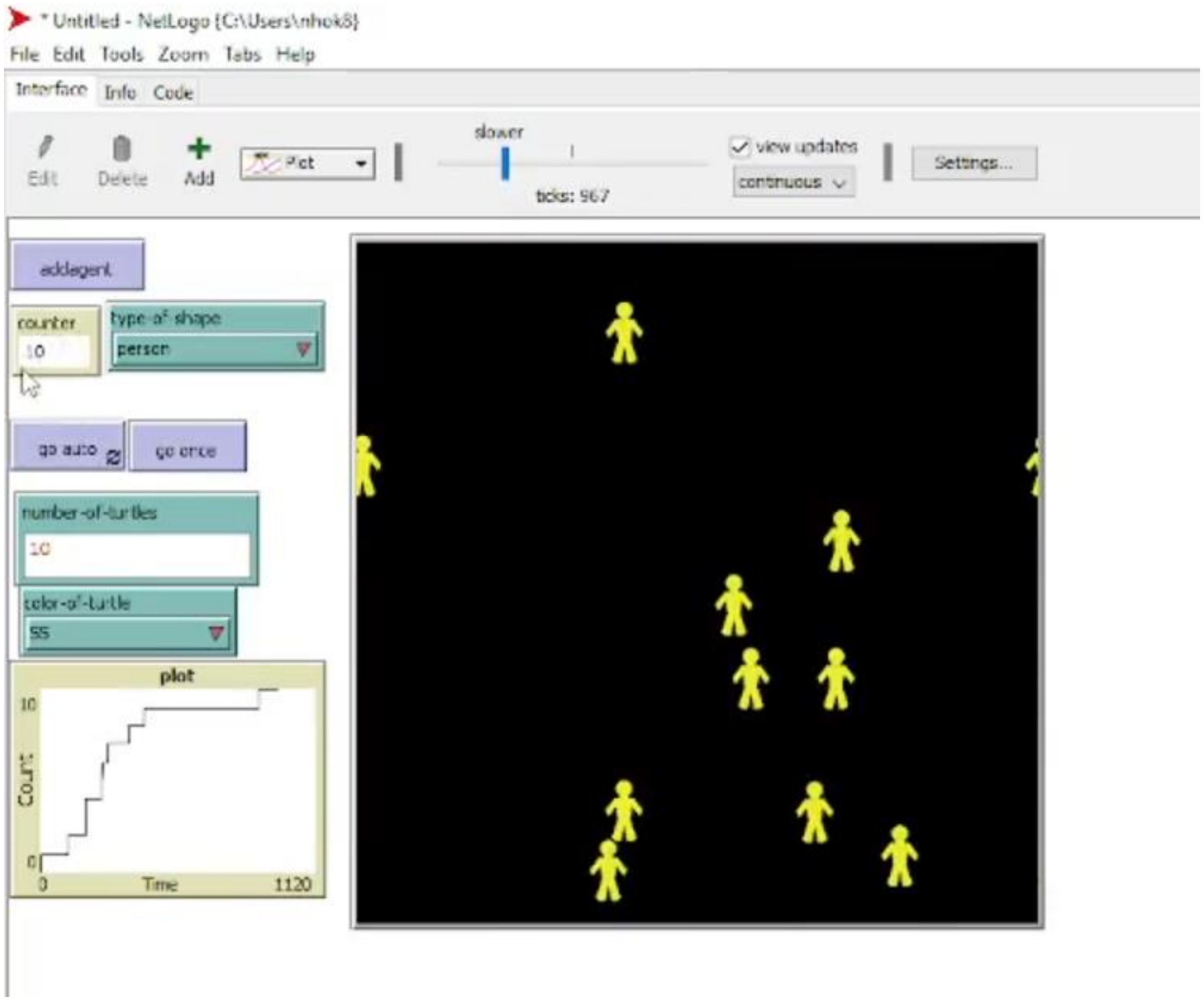


Name: Diffusion Curve
X axis label: Time   X min: 0   X max: 10
Y axis label: Count Turtles   Y min: 0   Y max: 10
[✓] Auto scale?   [ ] Show legend?

Plot pens

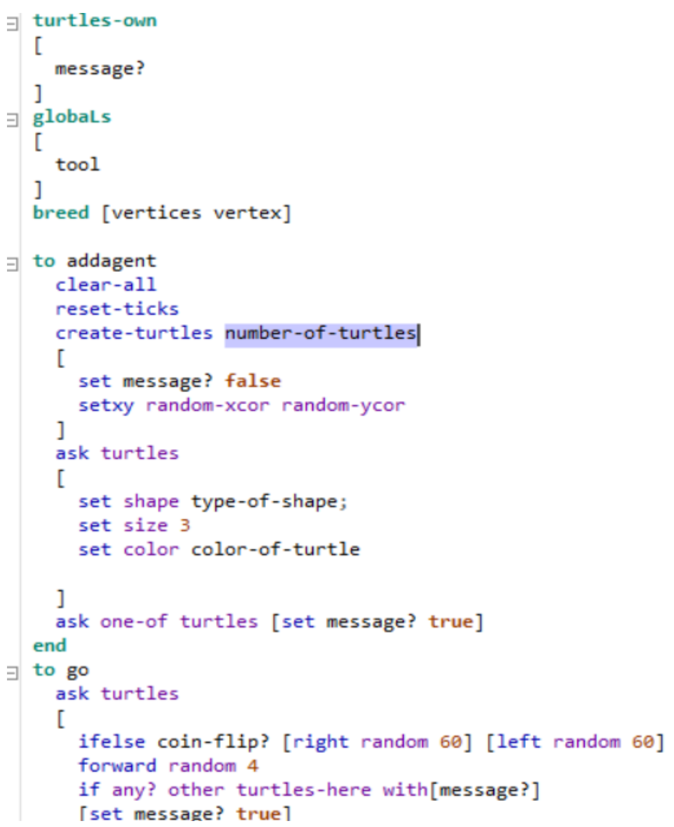
Color	Pen name	Pen update commands
■	default	`plot count turtles with [color = yellow]`

Here is the option we can change in plot. First is the name of the plot, then X is the time run, Y is the turtles with have message.
In the plot pens, color is the color of line of plot, pen name is name of the pen, pen update comma
The image above represents an example for blue objects where there is a random yellow object that is infectious, if the yellow object shares the same x, y degrees as the blue objects. then blue objects will be changed to yellow
Results:



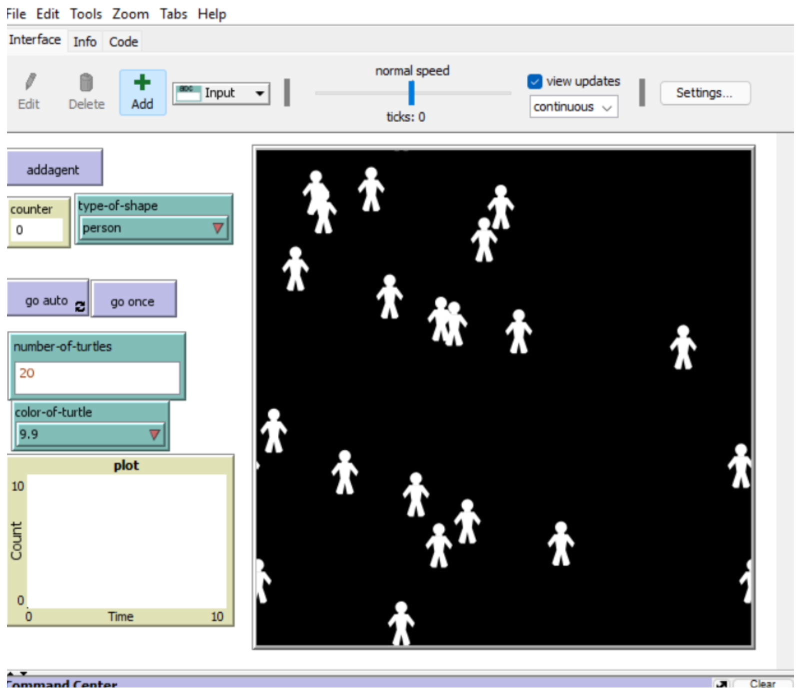
counter: 10
type-of-shape: person
number-of-turtles: 10
color-of-turtle: 55

- Create utilities to change the parameters of the search model, such as heuristic strategies, number of agents, etc.



```
turtles-own
[
  message?
]
globaLs
[
  tool
]
breed [vertices vertex]

to addagent
  clear-all
  reset-ticks
  create-turtles number-of-turtles
  [
    set message? false
    setxy random-xcor random-ycor
  ]
  ask turtles
  [
    set shape type-of-shape;
    set size 3
    set color color-of-turtle

  ]
  ask one-of turtles [set message? true]
end
to go
  ask turtles
  [
    ifelse coin-flip? [right random 60] [left random 60]
    forward random 4
    if any? other turtles-here with[message?]
    [set message? true]
```

We can change number-of-turtles, type-of-shape, color-of-turtle by changing the above code

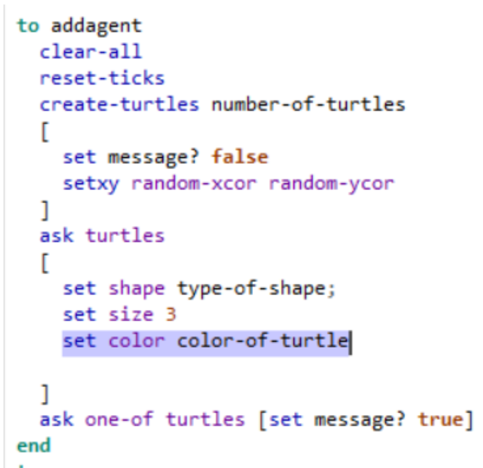


ticks: 0
counter: 0
type-of-shape: person
number-of-turtles: 20
color-of-turtle: 9.9

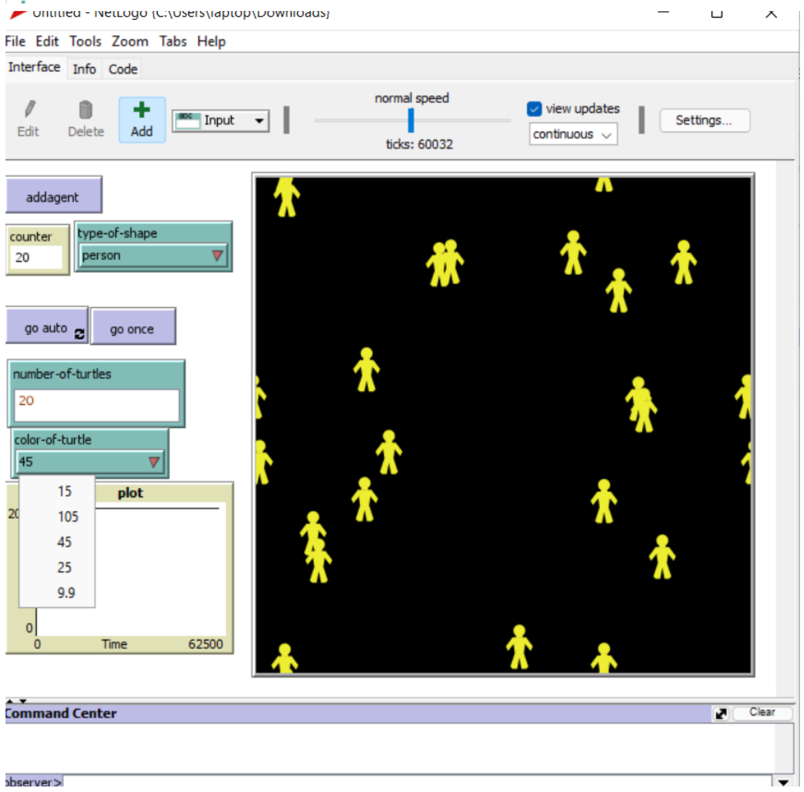
Otherwise we have two ways to change parameter of number-of-turtles, type-of-shape, color-of-turtle is through changing the code or we insert directly through 'input' in the button box

- Color agents and the environment

We can color agents by setting them with colors according to the input of the user, besides, we can also color agents with the options of the button box color palette. , for example red, orange,...



```
to addagent
  clear-all
  reset-ticks
  create-turtles number-of-turtles
  [
    set message? false
    setxy random-xcor random-ycor
  ]
  ask turtles
  [
    set shape type-of-shape;
    set size 3
    set color color-of-turtle

  ]
  ask one-of turtles [set message? true]
end
```

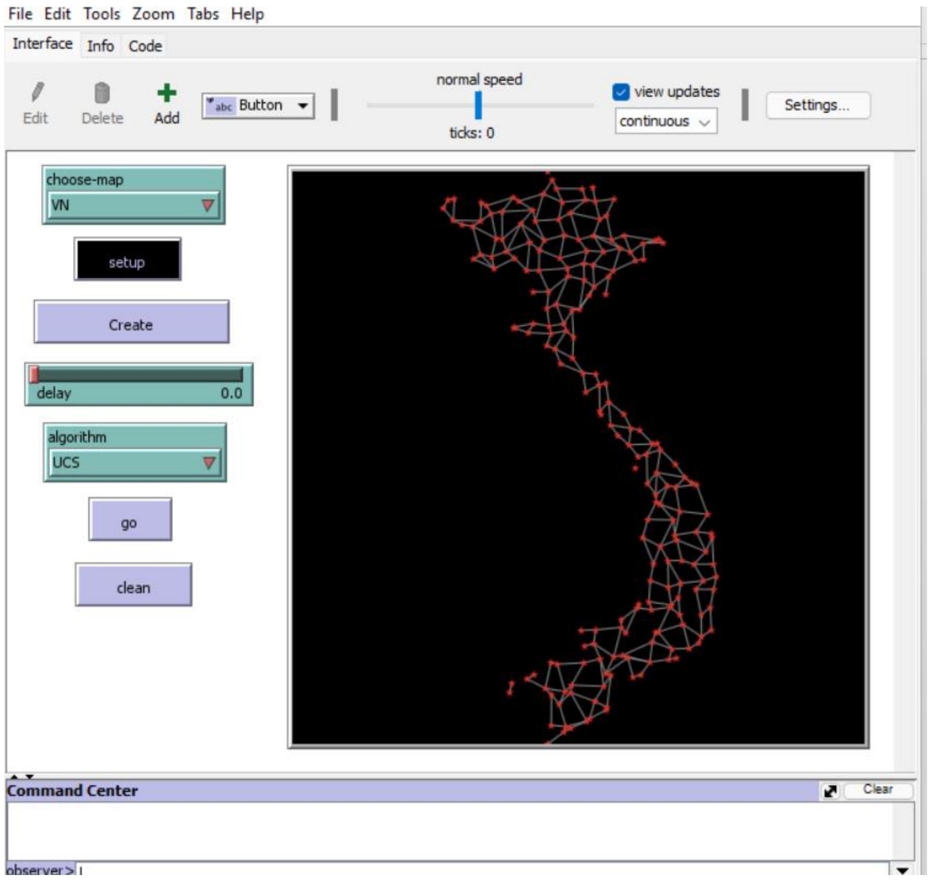


counter 20
type-of-shape person
number-of-turtles 20
color-of-turtle 45

15
105
45
25
9.9

Link videos to show you have programmed in the NetLogo for the search problem: https://drive.google.com/drive/folders/1-rQDPvjOLwKygqjC2ub5YuGjk0sTNYLC?usp=sharing

2. How to use the program written on NetLogo

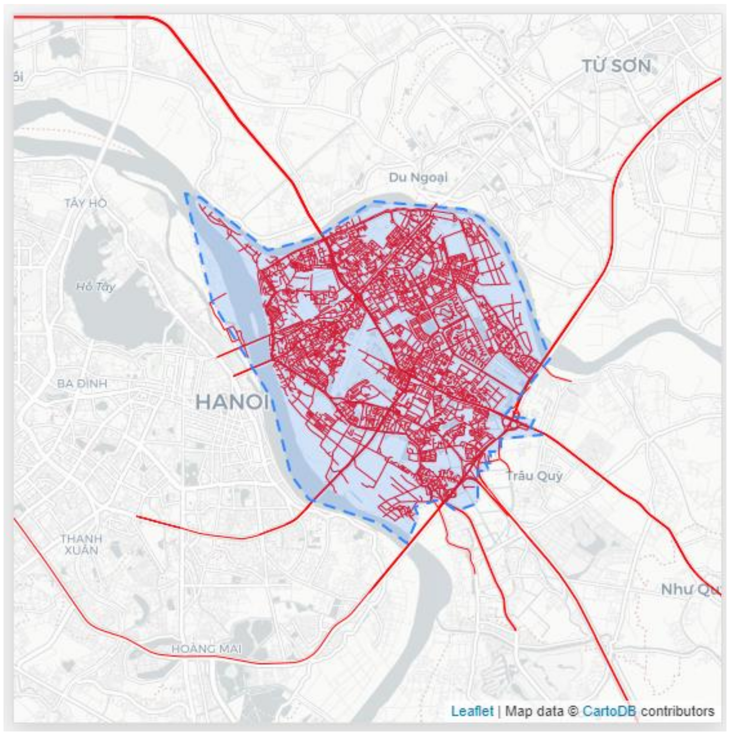


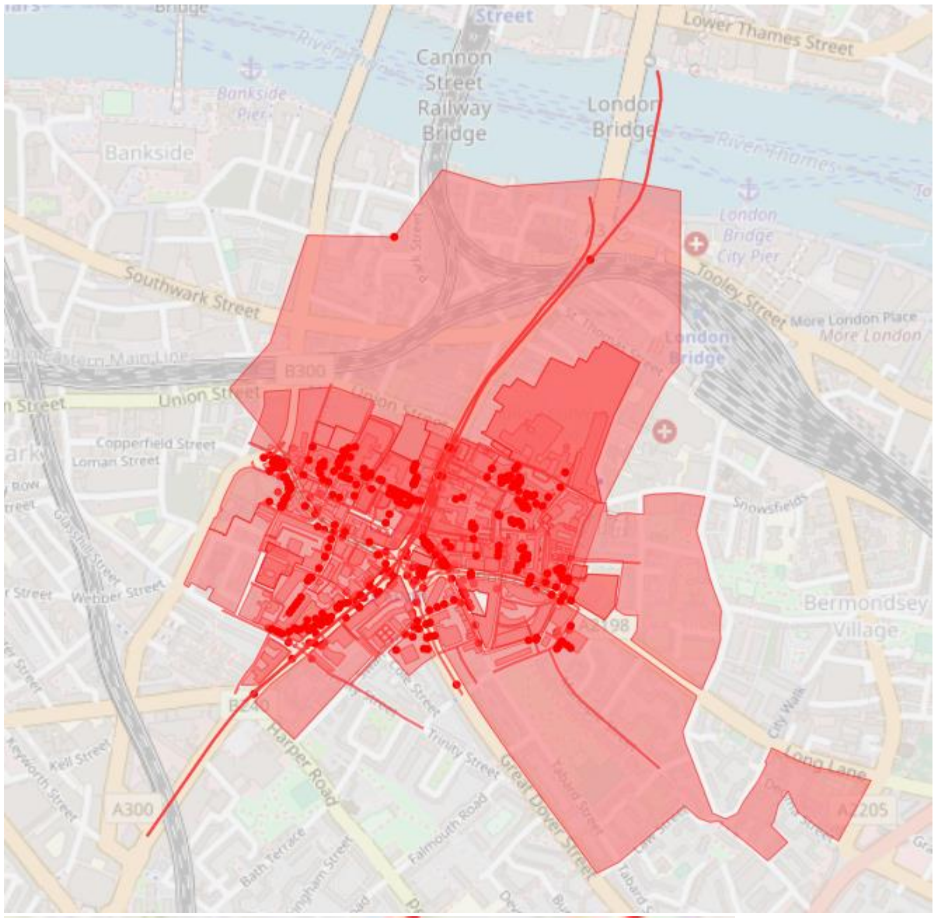
- chosen-map: choose a map to run the program, in this program there are 5 maps corresponding to 5 test cases to run and from there draw conclusions corresponding to each level
- setup: This button has the function of uploading the map and deleting the unnecessary parts, leaving only the path connecting the points.
- Create: create agents, algorithms, number of goals corresponding to each level
- delay: The delay of the algorithm will clearly show how the algorithm works, when we choose to park with a delay of at least 0.1, we will see the algorithm to find the way on the map.

- algorithm: choose between BFS, DFS, UCS, GBFS and A* algorithms
- go: shows the route of the agent using the algorithm to get to the goals
- clean: reset color of all path in map
- Command Center: used to type commands, such as "show [cost-to-des] of src" to show path cost.

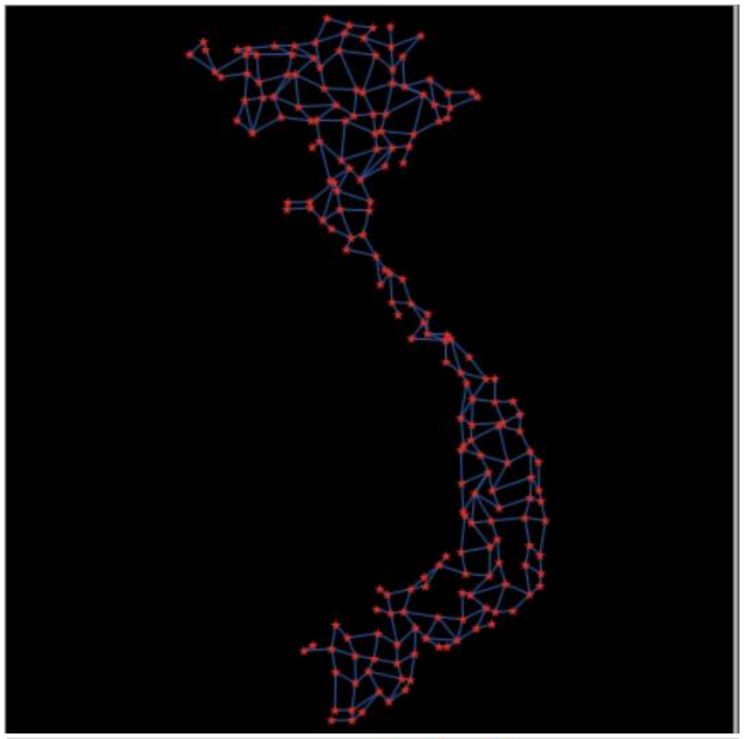
3. Map

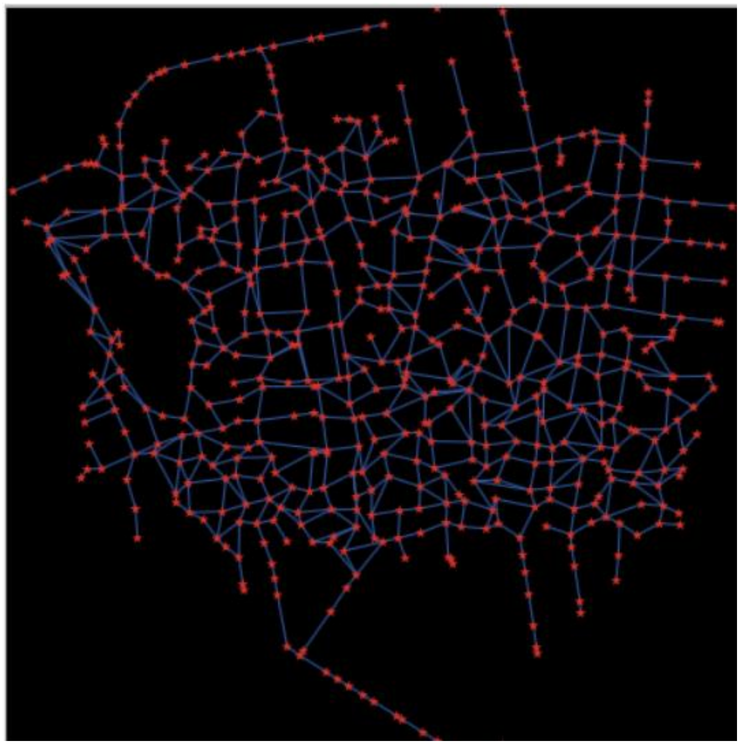
We use maps of countries and locations respectively Vietnam, Da Nang, Hanoi, England, Korea:

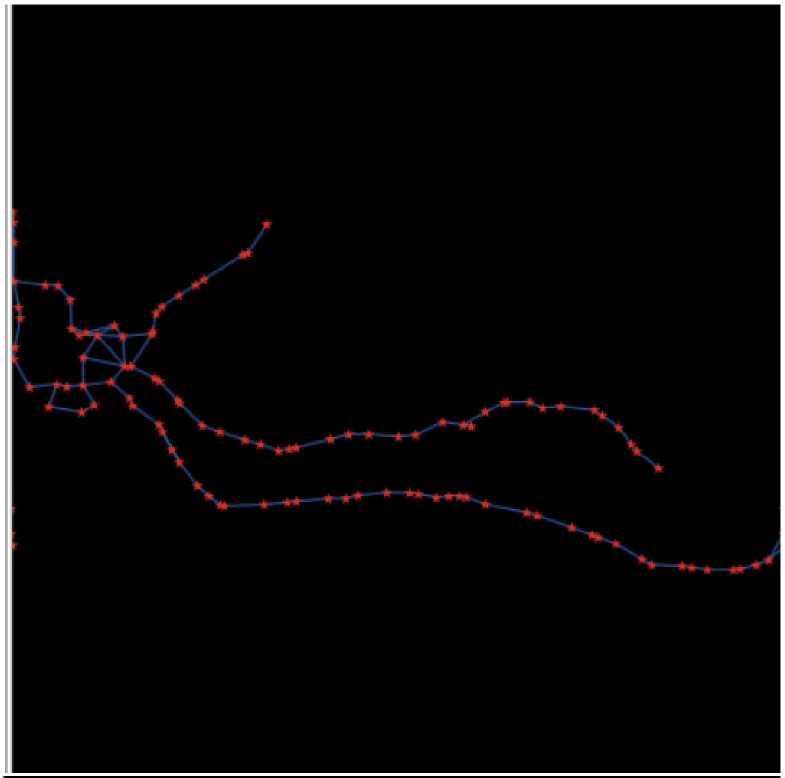

TỪ SƠN
Du Ngoại
TÂY HỒ
Hồ Tây
BA ĐÌNH
HANOI
Trâu Quỳ
THANH XUÂN
HOÀNG MAI
Như Qu
Leaflet | Map data © CartoDB contributors

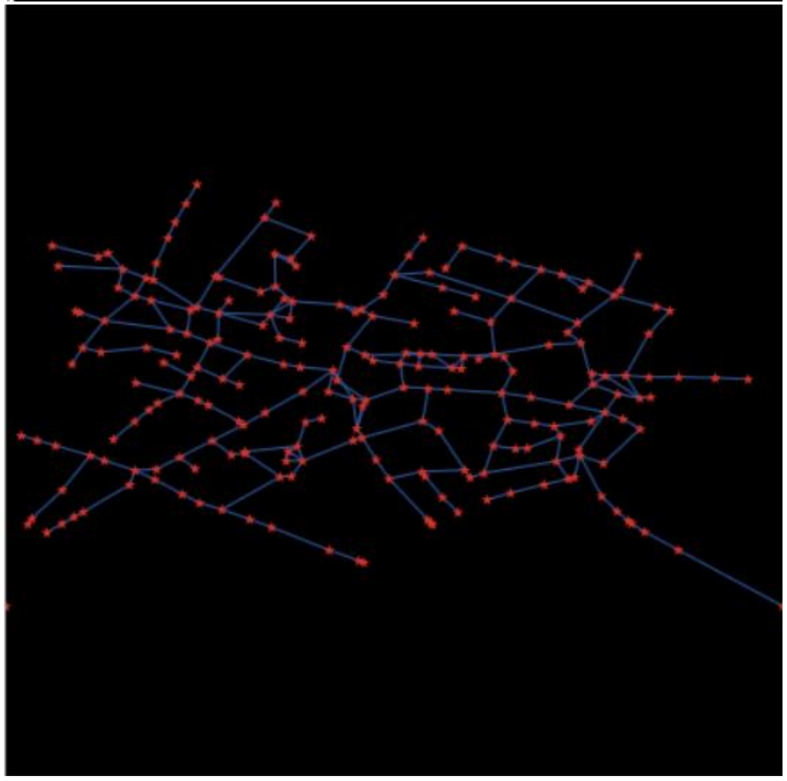


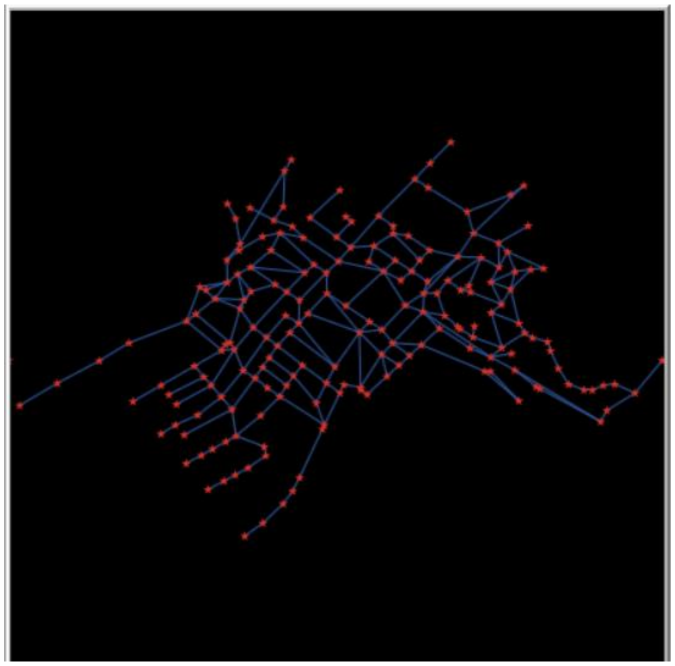
The map after convert is in turn:











The way to have maps for making projects is to use ‘png’ image files then convert to ‘osm’ files and from ‘osm’ files to ‘shp’ files. However, during the transfer process, there are errors or because the conversion tool displays missing data, so the ‘shp’ file image is missing more than the original ‘png’ image.

Five maps are used as 5 test cases for search algorithms from which we draw comparison results through levels.

4. About Algorithms

BFS: Is a search method for finding a node in a tree data structure that meets a set of criteria. It begins at the root of the tree and investigates all nodes at the current depth level before moving on to nodes at the next depth level. To maintain track of the child nodes that have been encountered but not yet inspected, more memory, generally a queue, is required.

DFS: is a tree or graph data structure traversal or search technique. The algorithm starts from the root node (in the case of a graph, any random node can be used as the root node) and examines each branch as far as feasible before backtracking.
UCS: only insert the source, then insert each item one by one as needed. We verify whether the item is already in the priority queue at each stage (using

visited array). If that's the case, we'll use the decrease key, and if that's not the case, we'll use the insert key.

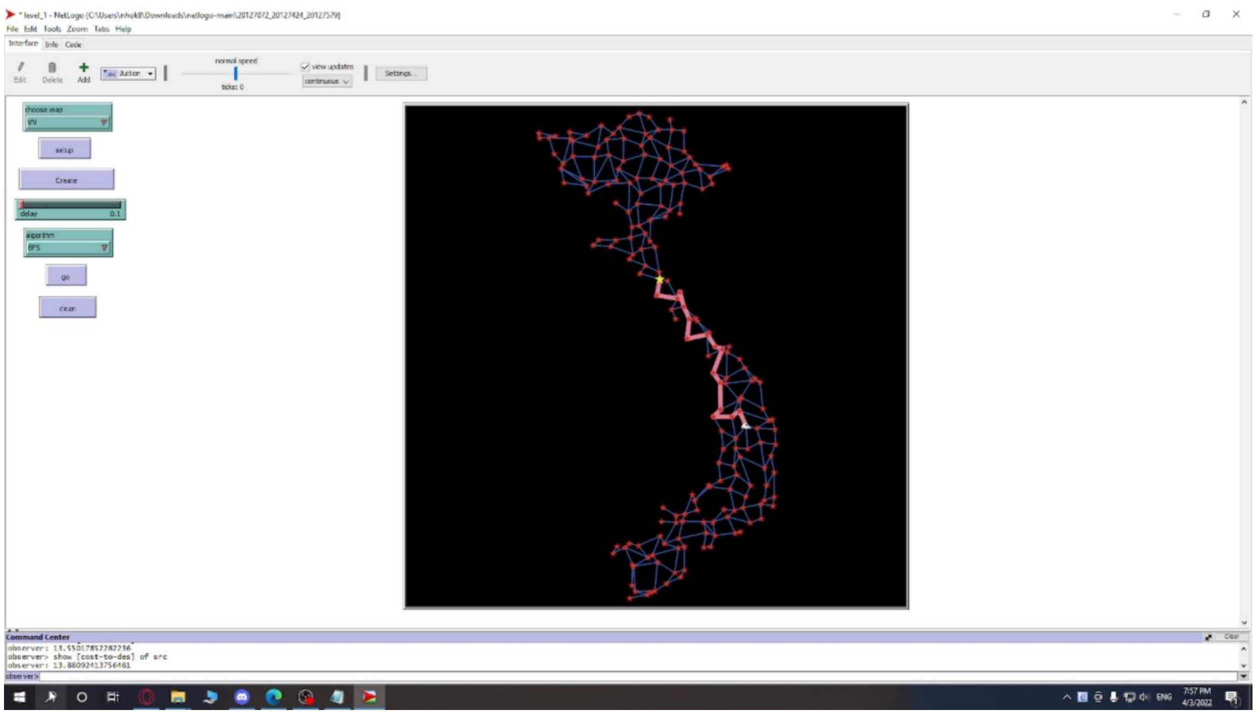
A*: must decide which of its pathways to expand with each iteration of its main loop. It does so based on the path's cost and an estimate of the cost of extending the path to the target. A*, in particular, chooses the path that reduces the amount of time it takes to complete a task.

Our team has added the GBFS algorithm to make the comparison of algorithms clearer, through this we can understand how it works as well as the optimization of each algorithm.

GBFS: refers to a group of search algorithms that explore a network by extending the most promising node based on a set of rules.

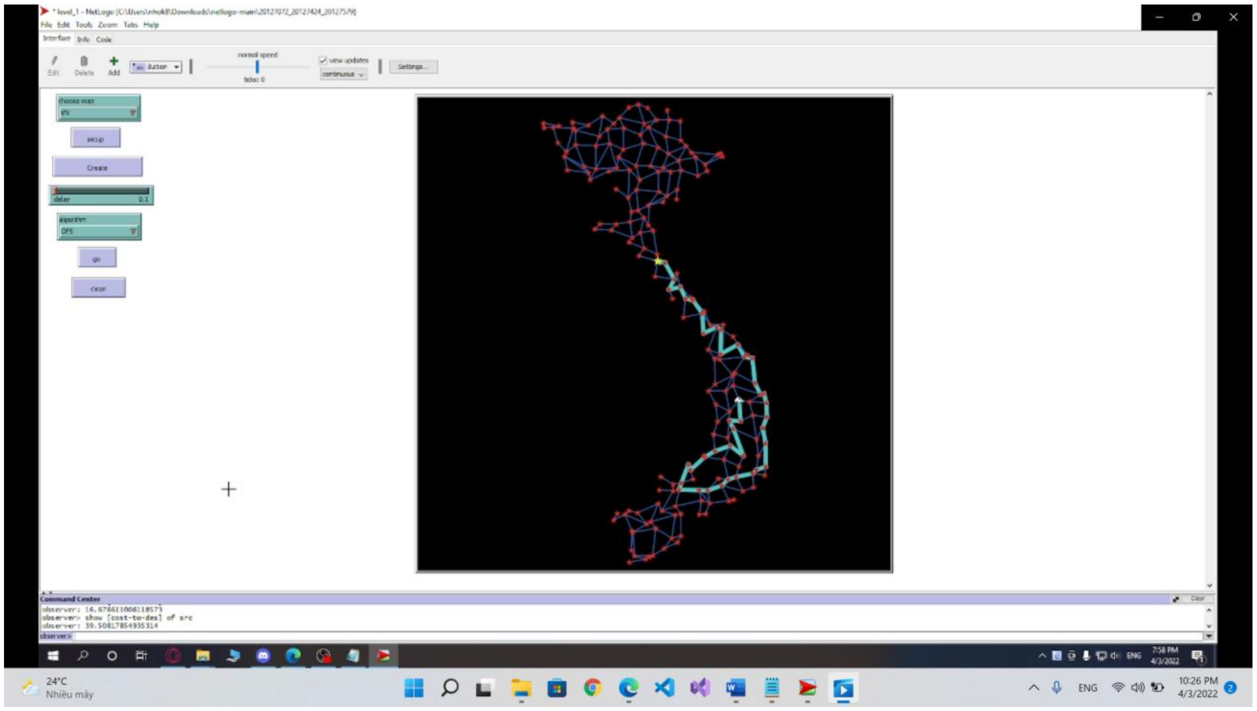
5. Requirement 2: Level 1

Map VN: path cost



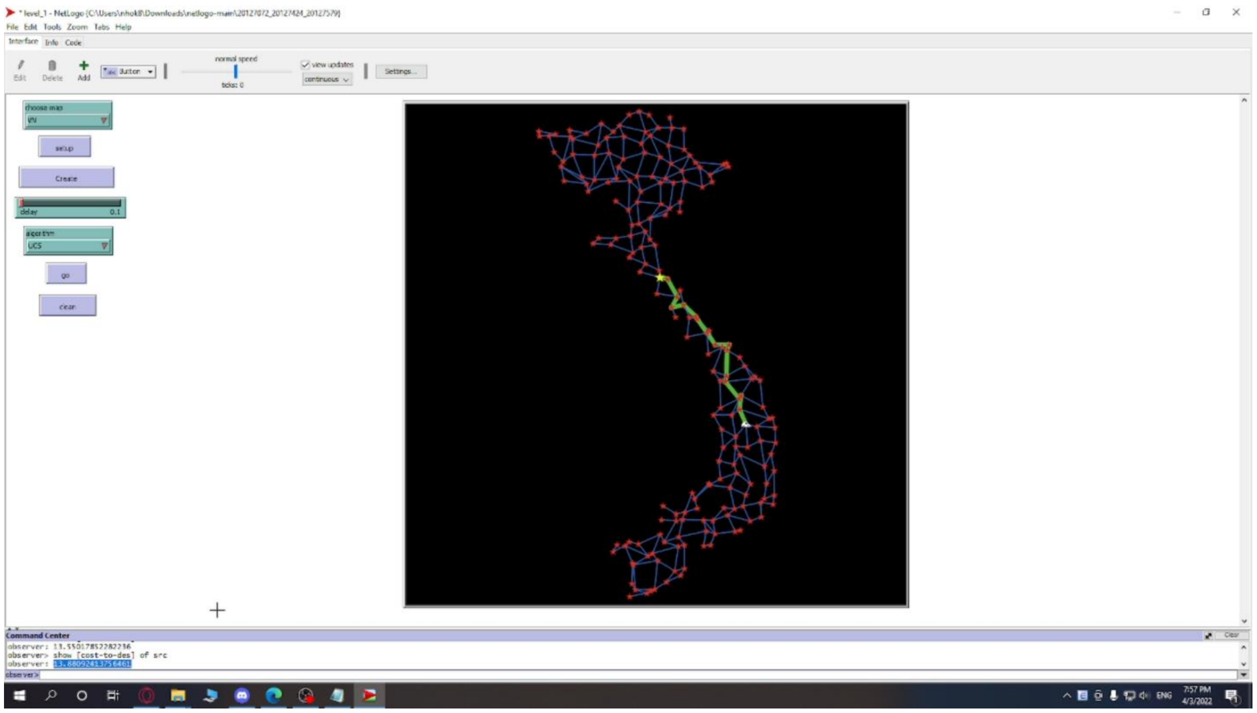
```
observer: 13.55017852282236
observer> show [cost-to-des] of src
observer: 13.88092413756461
observer>
```

- BFS: 16.676611006118573



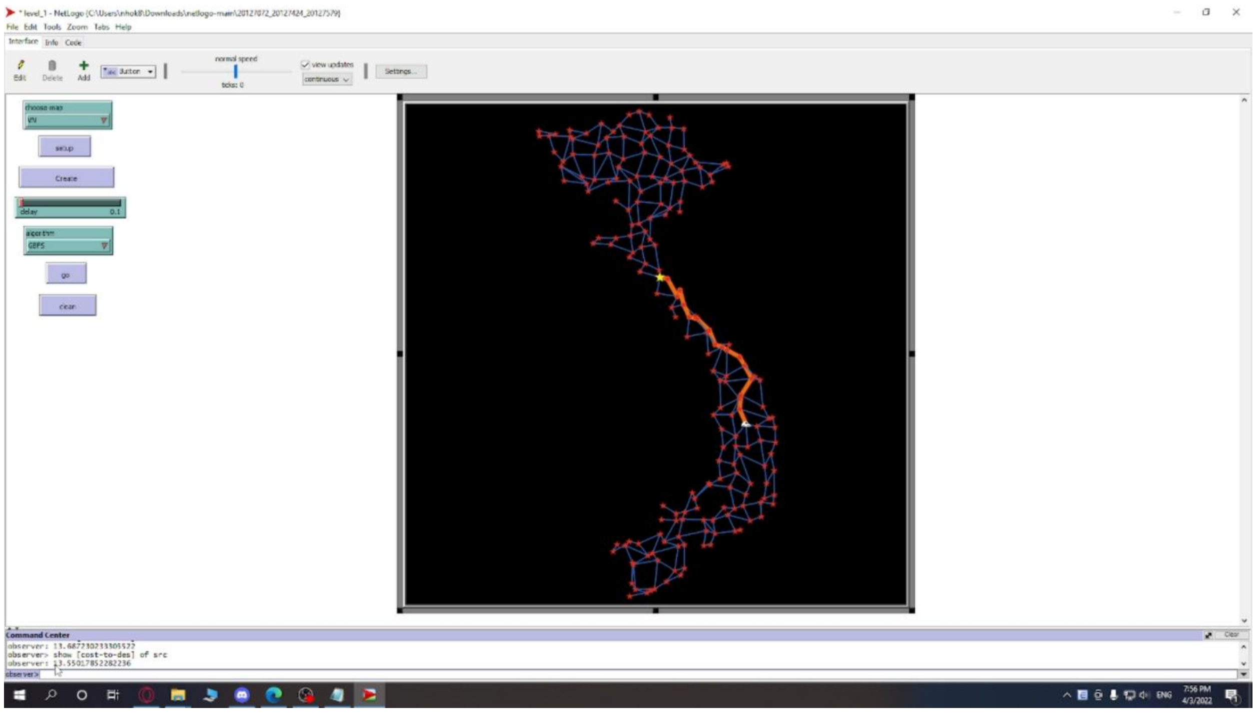
```
Command Center
observer: 16.67861100618573
observer> show [cost-to-des] of src
observer: 39.50817854935314
observer>
```

- DFS: 39.50817854935314



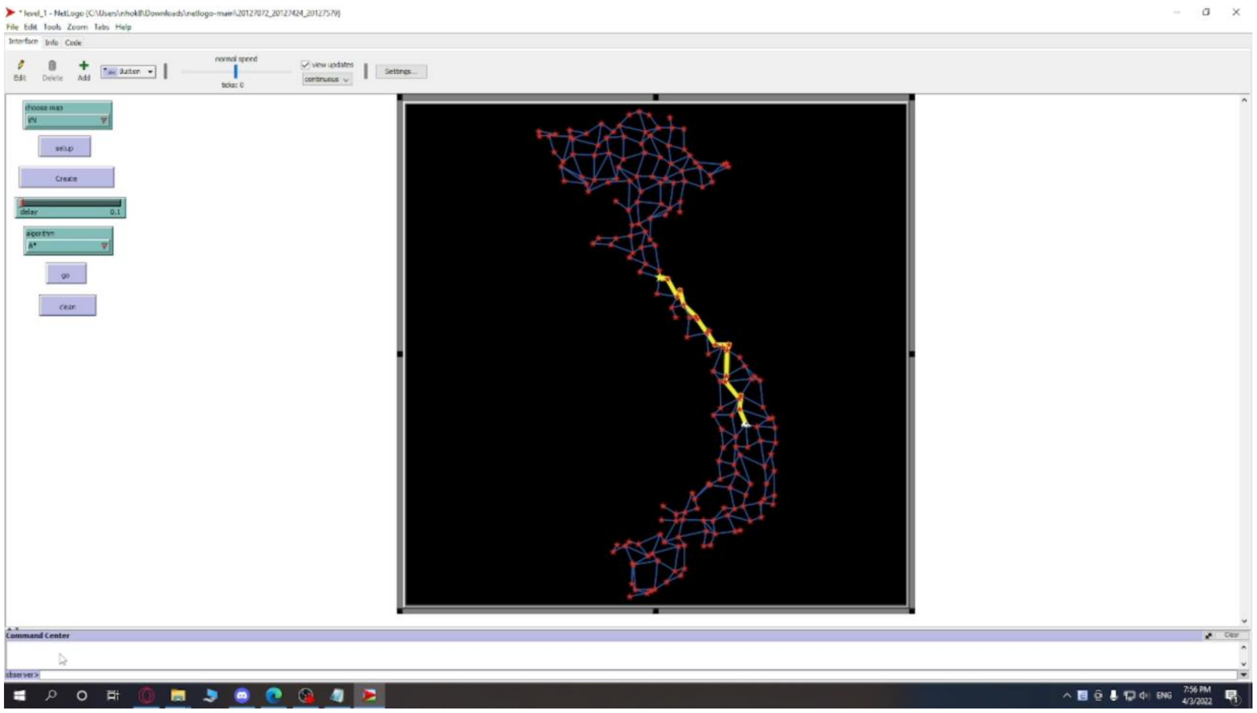
```
Command Center
observer: 13.55017852282236
observer> show [cost-to-des] of src
observer: 13.88092413756461
observer>
```

- UCS: 13.88092413756461



```
Command Center
observer: 13.687230233305522
observer> show [cost-to-des] of src
observer: 13.55017852282236
```

- GBFS: 13.55017852282236



- A*: 13.687230233305522

Similarly we get the path cost of the following maps:

Map DaNang:

- BFS: 4.617604059836681
- DFS: 107.93112725876522
- UCS: 4.617604059836681
- GBFS: 4.61760405983668
- A*: 4.617604059836681

Map England:

- BFS: 20.353383537894220
- DFS: 69.4817977100682
- UCS: 16.85800749499975
- GBFS: 16.85800749499975
- A*: 17.835115611783976

Map Korean:

- BFS: 9.029605155348422
- DFS: 68.92136225330194
- UCS: 9.227846703203804
- GBFS: 9.227846703203804
- A*: 9.227846703203804

Map HaNoi:

- BFS: 6.2223478099770135
- DFS: 6.2223478099770135
- UCS: 6.2223478099770135
- GBFS: 6.2223478099770135
- A*: 6.2223478099770135

Comments:

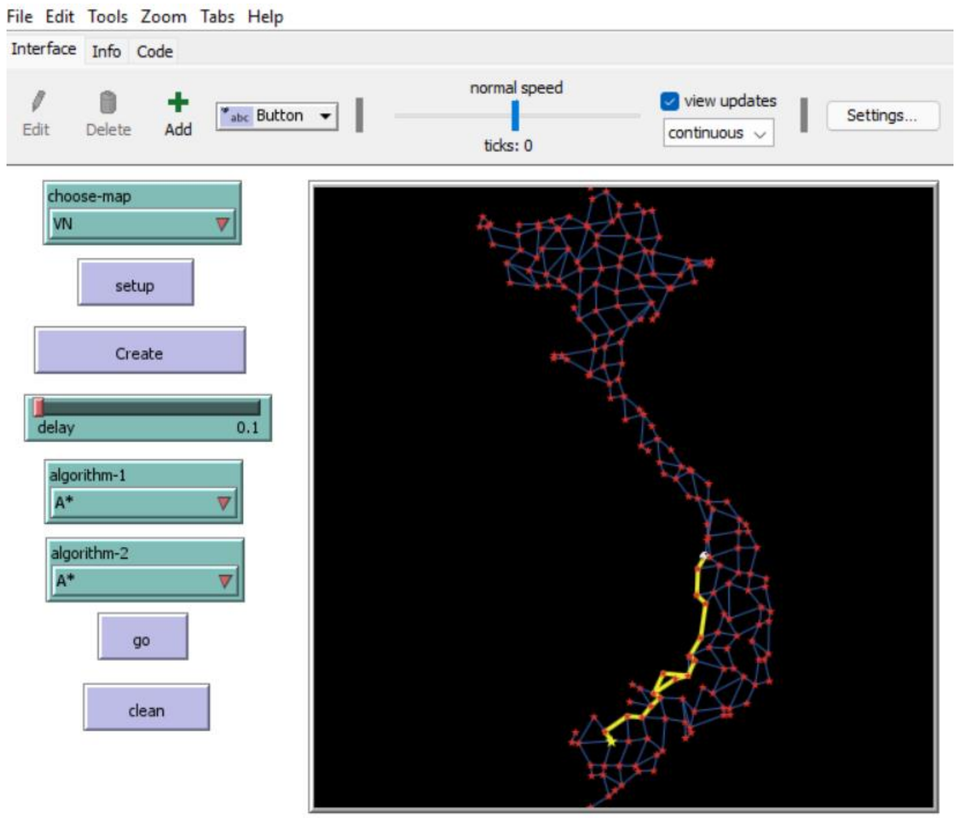
The conclusions are identical to those of the Vietnam map after running the algorithm on the remaining four maps (DaNang, HaNoi, Korean, and England). From this, the following conclusions may be drawn:

- From the above data, we can see that the VN map has the smallest path cost of GBFS (=13.55017852282236), the longest path cost is of DFS (=39.50817854935314). Next, the DaNang map has the longest path cost of DFS (=107.93112725876522), and the rest of the search algorithms have the same path cost (4.617604059836681). Map England has the same minimum path cost of GBFS and UCS (=16.85800749499975), the longest path cost is of DFS (=69.4817977100682). Besides, map Korean has the smallest path cost of BFS (=9.029605155348422), the longest path cost is of DFS (=68.92136225330194). Finally, Map HaNoi has the same path cost of the search algorithms (=6.2223478099770135)
- If comparing all 5 maps together, even though each map has its own complexity but the shortest path cost belongs to GBFS and longest belongs to DFS. So using GBFS in search will help solve the problem at less cost.

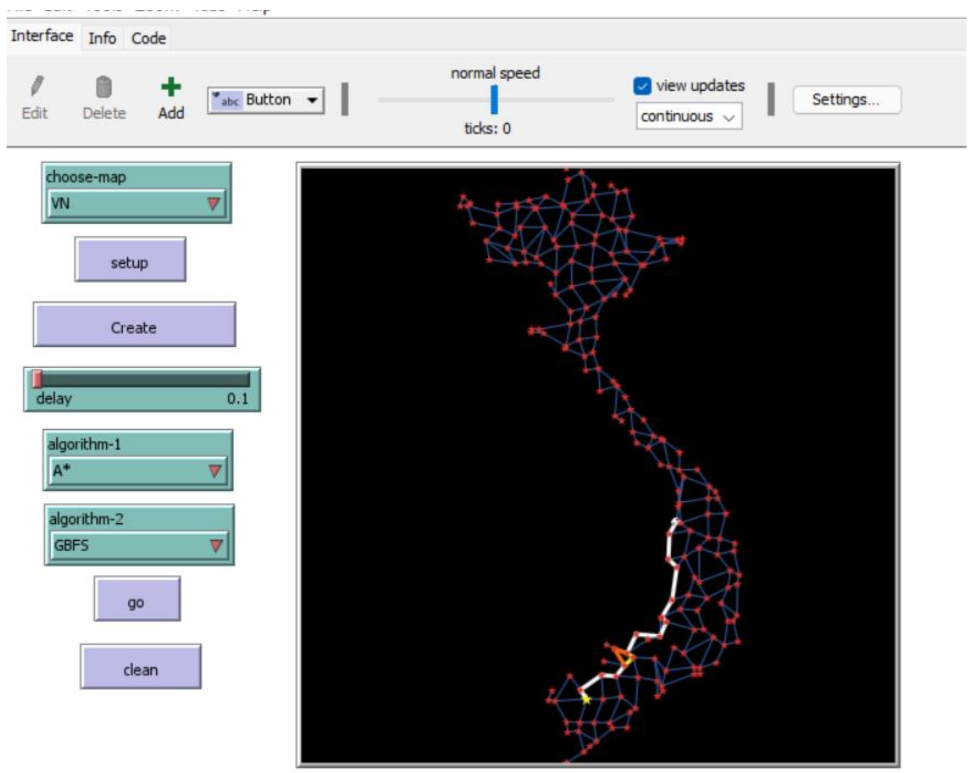
The image above is only drawn from a map of Vietnam if you want to understand better or watch the video:
https://drive.google.com/file/d/1jU66ptRAUAA9yMv718BPI16rpnG4YDFn/view?usp=sharing

6. Requirement 2: Level 2

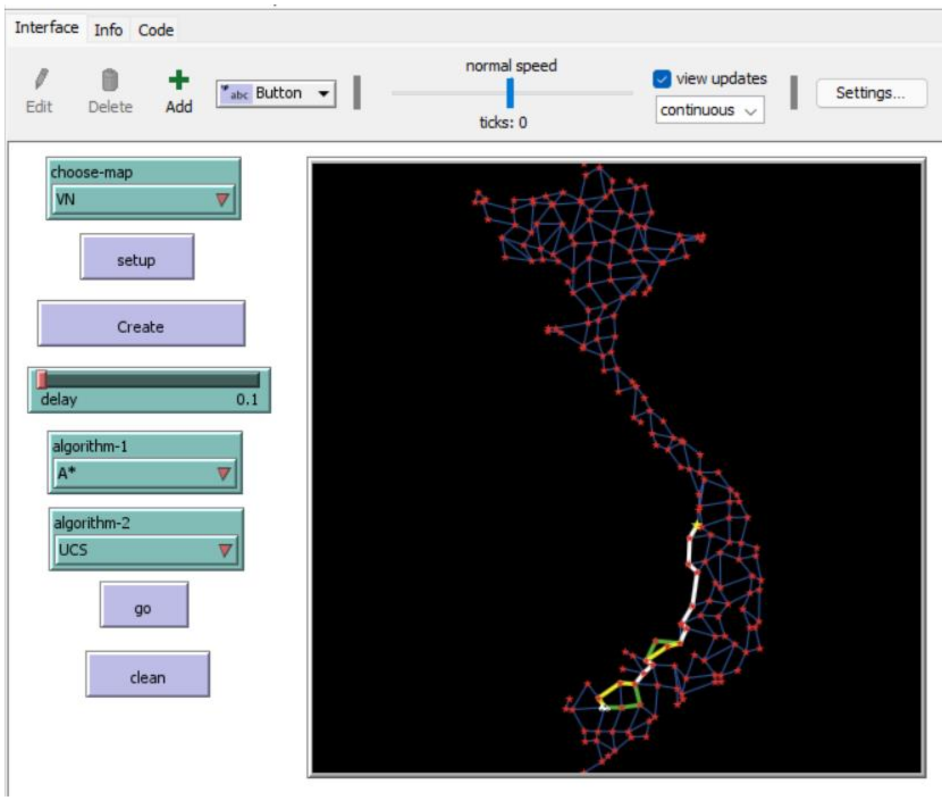
Map VN:



choose-map: VN
delay: 0.1
algorithm-1: A*
algorithm-2: A*



choose-map: VN
delay: 0.1
algorithm-1: A*
algorithm-2: GBFS

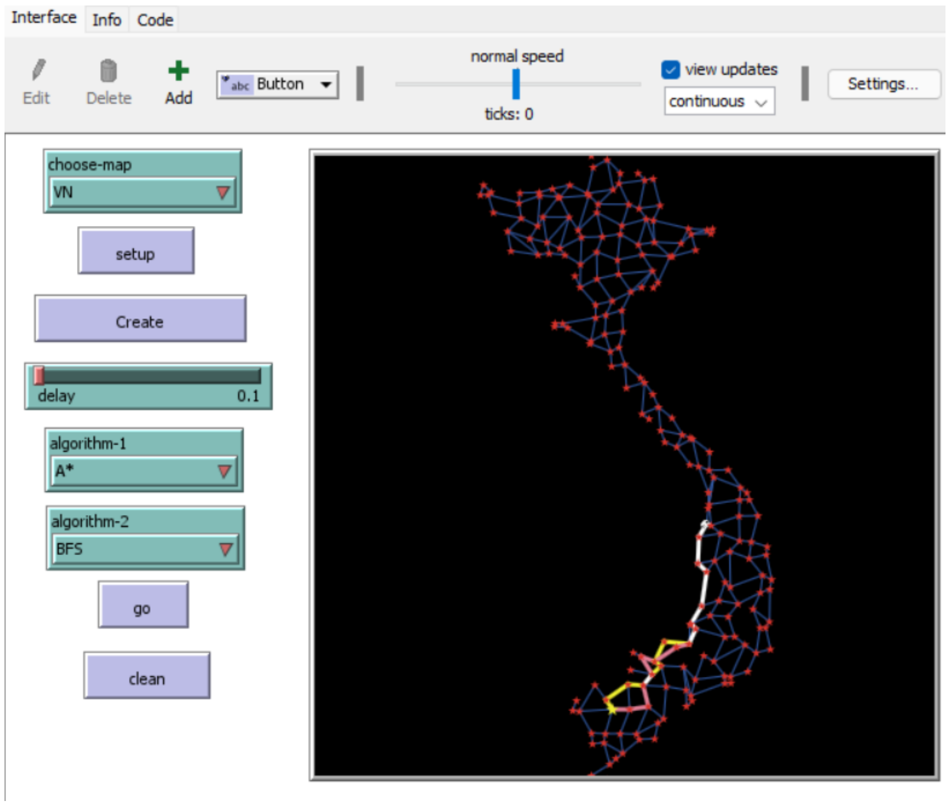


choose-map: VN

delay 0.1

algorithm-1: A*

algorithm-2: UCS

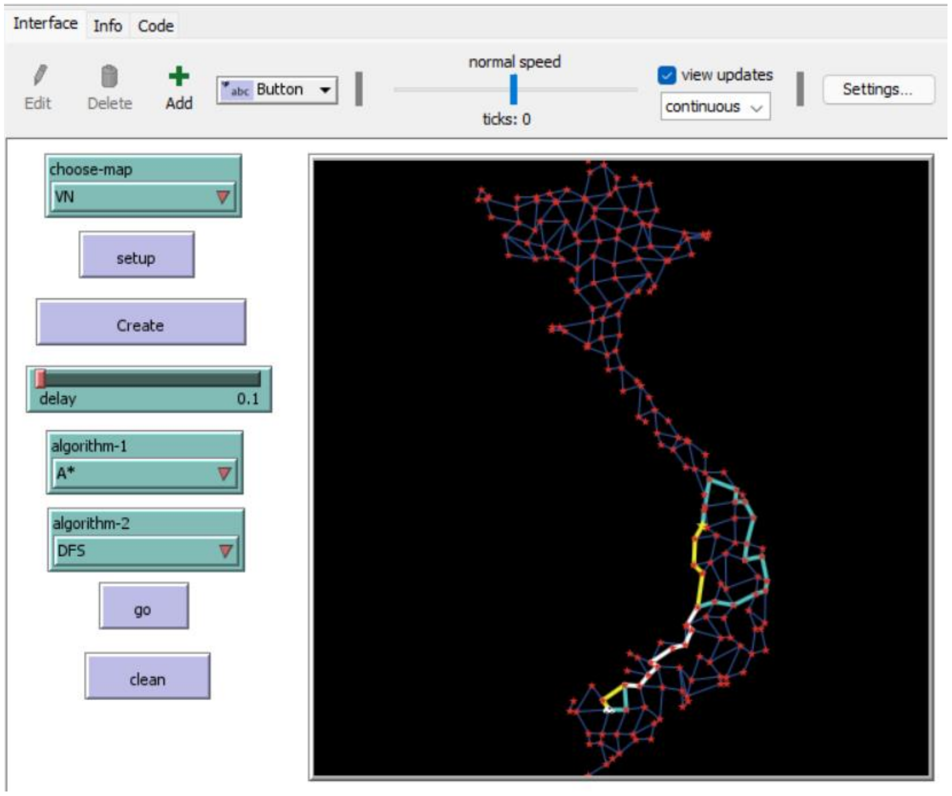


choose-map: VN

delay 0.1

algorithm-1: A*

algorithm-2: BFS



choose-map: VN

delay 0.1

algorithm-1: A*

algorithm-2: DFS

Comment:

For the remaining 4 maps DaNang, HaNoi, Korean, England, after running the algorithm, it also gives equivalent results compared to the map of Vietnam. From this we draw the following conclusions:

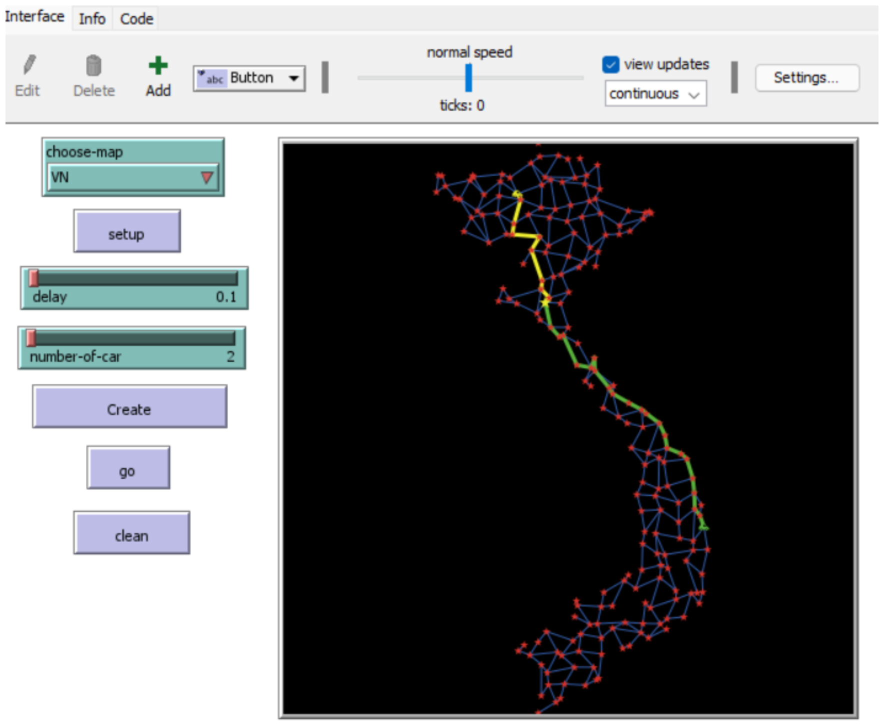
- The agent now reaches the endpoint with algorithm A* and returns to the starting point with algorithms A*, GBFS, UCS, BFS and DFS respectively. The algorithm illustration clearly shows the path of the search algorithm when using the A* algorithm to get to the destination and back using those algorithms. Through this, we can see that the path of A* going and returning using A* or GBFS algorithm will be different but almost equal, the difference is not much for the return path using UCS algorithm. and BFS, but there is a clear difference to the return path using the DFS algorithm. From there, we see that the DFS path to the destination is the longest and A* or GBFS will be the shortest

If you want to see more maps other than Vietnam, check out the video below: https://drive.google.com/file/d/18aUhpMCztiPfFkG8VmY4V_rdkrSAOzi4/view?usp=sharing

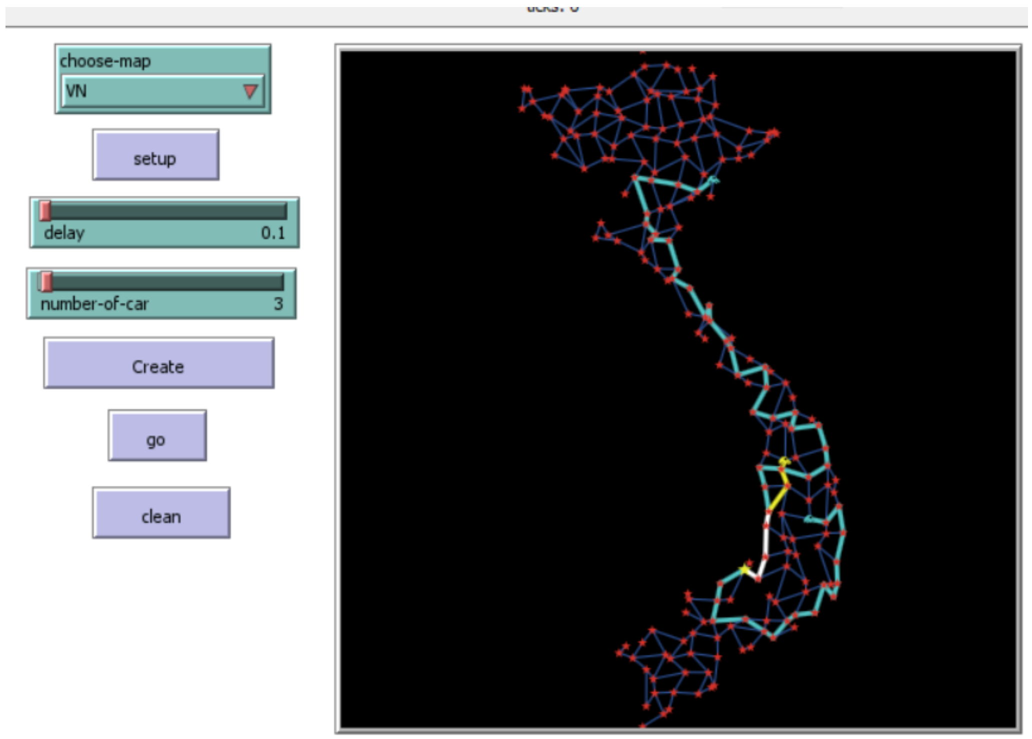
7. Requirement 2: Level 3

Map VN:

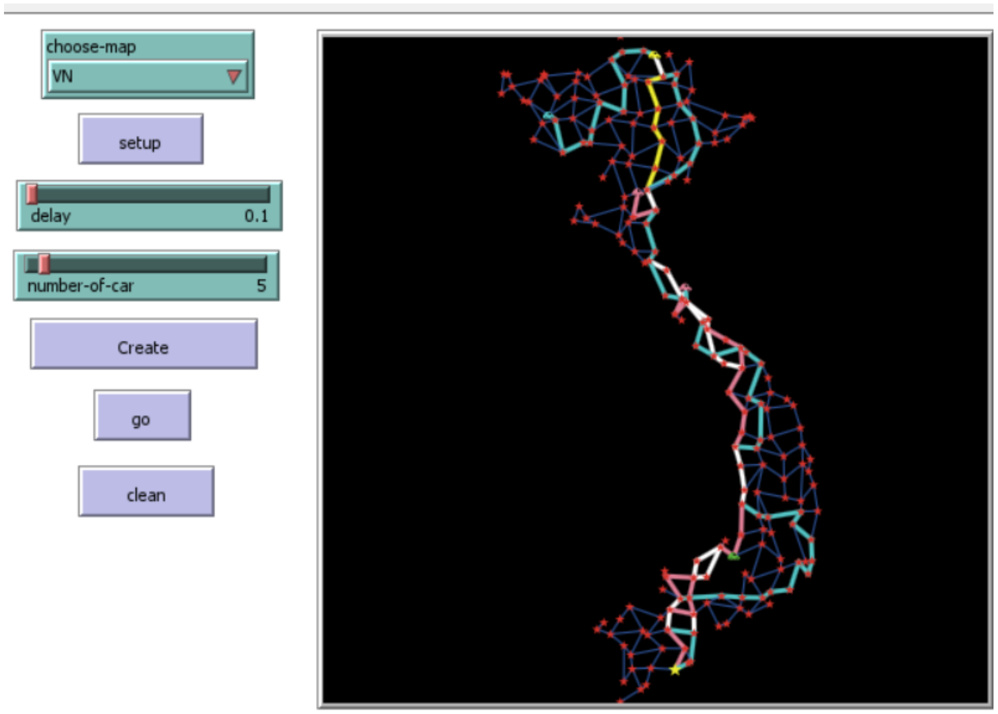
- With 2 car:



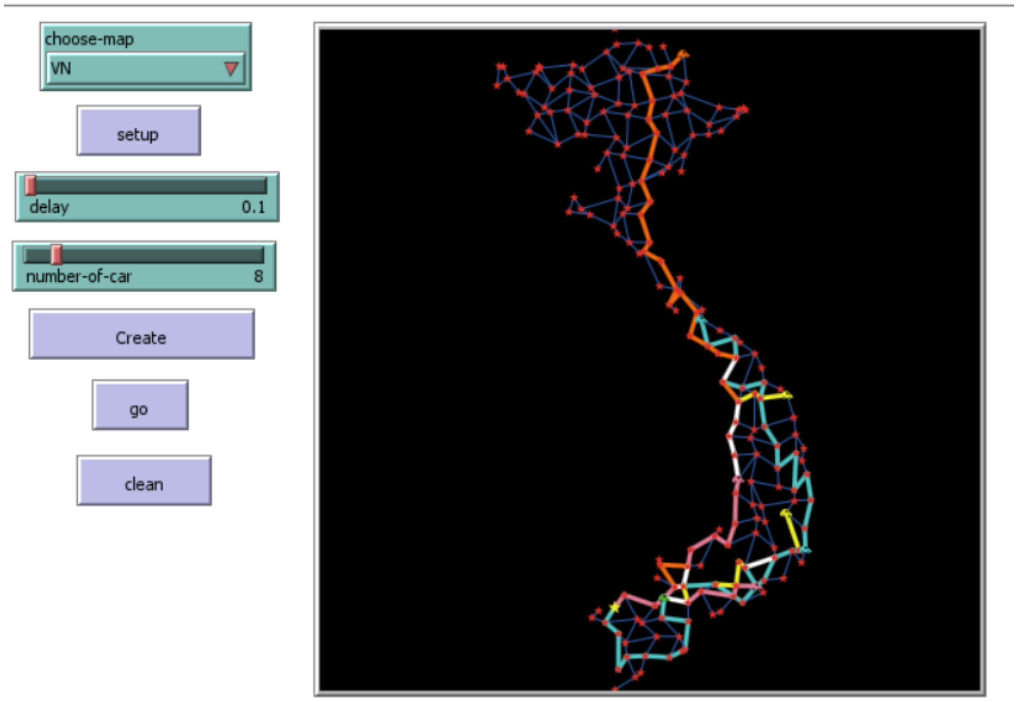
- With 3 car:



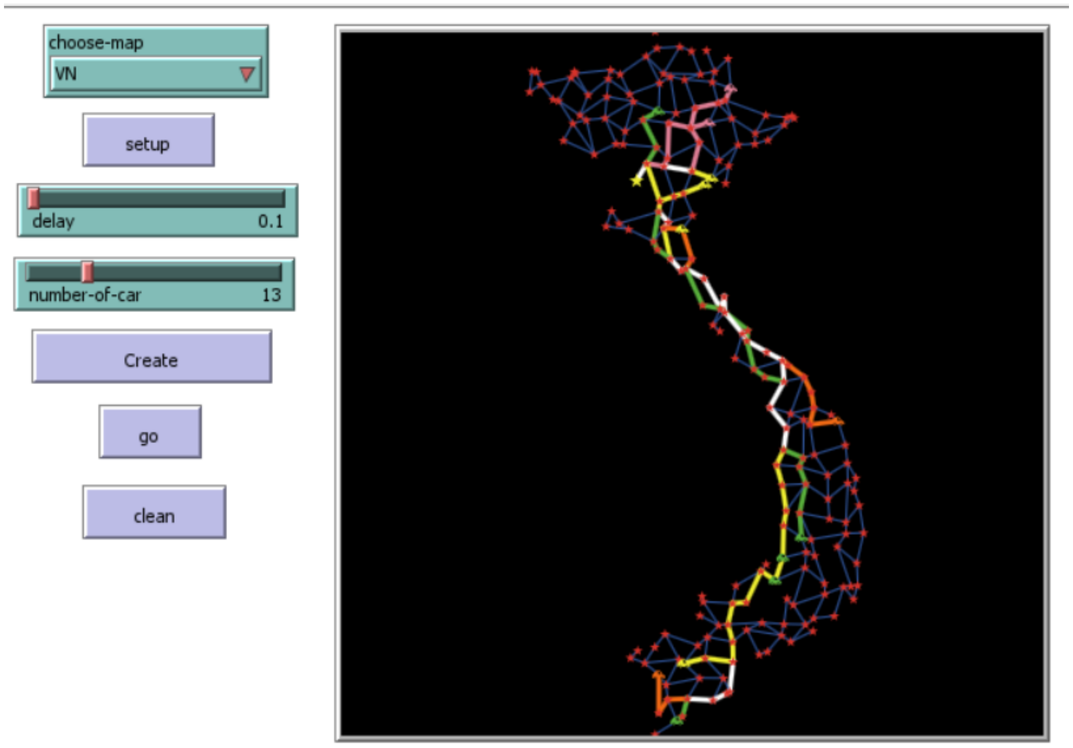
- With 5 car:



- With 8 car:



- With 13 car:



Comment:

After running the algorithm on the remaining four maps (DaNang, HaNoi, Korean, and England), the findings are similar to those of the Vietnam map. The following conclusions may be drawn from this:

- Through the images with the number of vehicles 2, 3, 5, 8, 13 respectively, we can see that the complexity is increasing, the paths are overlapping, creating confusion in seeing the image, the running time with The number of vehicles is also increasing gradually. Although the destination is the same, the starting position of the vehicles is different, so the cost to get to the destination is different, besides, because each vehicle will use an algorithm, it will be different.

The image above is only drawn from a map of Vietnam if you want to understand better check watch the video below for additional maps that aren't related to Vietnam: https://drive.google.com/file/d/1BRLcarsIHvJwkwXAx6emM5__8-D1rAf5/view?usp=sharing

Link videos to show generate at least 5 test cases for each level with different attributes: https://drive.google.com/drive/folders/15T9bbaPXgpGVTjaWKj2mCFDu2pHPrL2T?usp=sharing

VI. References

abmgis/abmgis: Accompanying resources for the book 'Agent-Based Modelling and Geographical Information Systems: A Practical Primer'. (github.com)

nnaaaa/SearchingSolver: 🔍 AI project perform with group of 3 people (github.com)

https://ccl.northwestern.edu/netlogo/docs/

https://mygeodata.cloud/converter/osm-to-shp

https://www.youtube.com/watch?v=NGNCenhcUu4

https://github.com/abmgis/abmgis/tree/master/Chapter08-Networks/Models/GMU-Social

http://www.cs.us.es/~fsancho/?e=131

YangZhouCSS/roads: Path finding model using the A-star algorithm in Netlogo (github.com)

Depth-first search - Wikipedia

Best-first search - Wikipedia

UCS - Wikipedia

Breadth-first search - Wikipedia

A* search algorithm - Wikipedia