LESSON 6 · 12 NOVEMBER 2026

Applied Systems Sciences 1

Watching emergence

Quiz 1, then the traffic model: its ODD, the brake rule and a space-time diagram of the jam

David Maier
Department of Environmental Systems Sciences
University of Graz

Quiz 1

30 minutes

Now **Open Quiz 1 on Moodle.** About 30 minutes, 15 points.

Closed book: no notes, no slides, no AI. Close everything else on your laptop.

When you are done (quietly, others are still writing)

1

Open

Your traffic model in VS Code.

→

2

Run

`uv run python model.py` in its terminal.

→

3

Read

`model.py` next to `ODD.md`: which part of the code belongs to which ODD element?

The traffic model in ODD: overview

1

Purpose and patterns

Can a jam appear with no accident, no road work and no traffic light?

- a jam from nothing
- a jam that travels backwards
- slower traffic with more cars

2

Entities, state variables and scales

- **Car:** position, speed 0 to 5
- **Road:** a ring of 100 cells, the chance to dawdle
- a cell is 7.5 m, a step 1 s, so speed 5 is 135 km/h

3

Process overview and scheduling

One process, driving. In each step every car first gets its new speed from the road as it is. Then all cars move at once.

Open `ODD.md` in your repository next to this slide: it says all of this in more words.

Nagel & Schreckenberg (1992) · Sugiyama et al. (2008)

The traffic model in ODD: design concepts

Concept	In the traffic model
Emergence	The jam. No car is told to stop, and no rule says where the jam goes.
Adaptation	A car adapts its speed to the gap in front of it.
Sensing	A car knows the gap to the next car, nothing else.
Interaction	Indirect: a car that brakes shortens the gap of the car behind it.
Stochasticity	Dawdling: a car slows down by chance, as people do.
Observation	The space-time diagram, the average speed, the standing cars.
Learning, prediction, collectives	Not used.

Emergence is the reason to build an agent-based model: the jam is in no rule.

The traffic model in ODD: details

5 Initialization

The cars evenly spread, all standing. A seed for the random numbers, so a run can be repeated.

6 Input data

None. Nothing comes from outside.

7 Submodels: each car, in this order

Accelerate	`speed = min(speed + 1, 5)`
Brake	`speed = min(speed, gap)`
Dawdle	with chance `p`, if moving: `speed - 1`
Move	`position + speed`, round the ring

> Four rules, one per line. That is the whole model.

From the ODD to the code

ODD	Python
2 Entity and state variables	a dataclass and its fields
2 Scales	`LENGTH = 100`, `V_MAX = 5`
3 Scheduling	`step()`: new speeds, then all move
4 Sensing	`gap()`: the empty cells in front
5 Initialization	`new_road()`
7 Submodels	one function per rule

In model.py

```python
@dataclass
class Car:
    position: int
    speed: int = 0

def accelerate(speed: int) → int:
    return min(speed + 1, V_MAX)
```

A dataclass bundles the state variables of one agent.
A function takes values in and hands one back with `return`.

Together: the brake rule

ODD 7, rule 2

A car never goes further than the empty cells in front of it.

```
speed = min(speed, gap)
```

Now, in model.py

```
def brake(speed: int, gap: int) → int:
    # TODO: the rule of ODD 7.
    # For now the car ignores the gap.
    return speed
```

Replace the TODO with the rule, save, and run `model.py` again.

A space-time diagram, drawn by a coding agent

The brief is already written

> *“the space-time diagram (one row per step), the average speed, the number of standing cars.”*

ODD.md, element 4, Observation

An agent reads `ODD.md` like any other reader. A good ODD is a good brief.

THE PROMPT

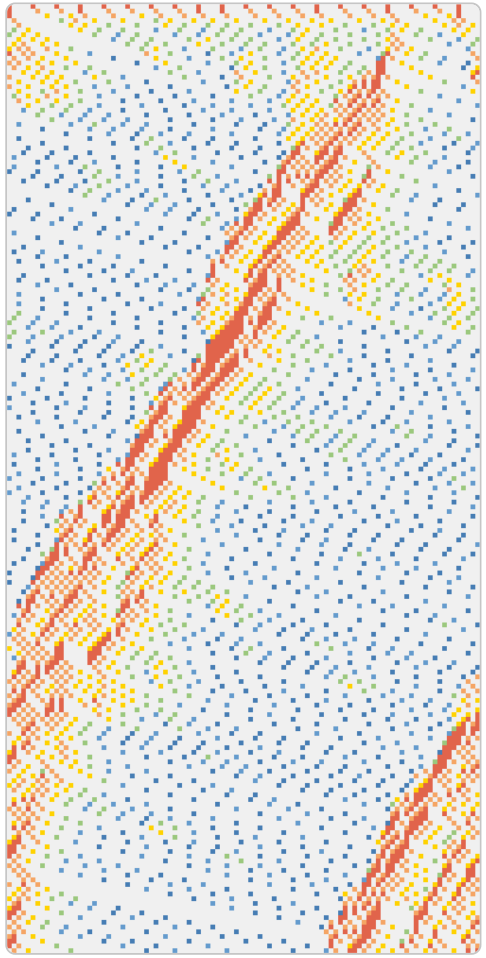
Read ODD.md and model.py. Under Observation the ODD names a space-time diagram. Make it as a picture: write a new file plot.py that runs simulate() from model.py with 20 cars, 30% dawdling, 200 steps and seed 1, and draws one row per step with time going down and one column per cell. An empty cell is white, a car has the colour of its speed. Save the picture as jam.png. Use matplotlib and add it with uv. Do not change model.py. Then run plot.py.

In VS Code: give the prompt to your **coding agent**. Read every change before you keep it.

Live: the agent at work

Live in class: the presenter's screen appears here.

Can we trust the diagram?



standing top speed

One row per step, time going down, the cars drive to the right.

Verification: does the code do what the ODD says? Check the diagram against what you already know.

✓ Same seed, same jam

The last 40 rows are the 40 lines `model.py` prints.

✓ The jam travels backwards

The stripe slants against the cars, as ODD 1 expects.

✓ No dawdling, no jam

With 0% the cars keep 4 cells apart, so the lines never meet.

✓ Always 20 cars

Every line of the text output has 20 digits.

> Code an agent wrote is code you did not write. Check it like any other.

Commit and push your rule

1

Stage

In Source Control: the + next to `model.py`.

→

2

Commit

A message such as “Brake for the car in front”, then Commit.

→

3

Push and check

Sync Changes, then reload your repository on github.com.

“Please tell me who you are”? Tell git your name once, in the terminal:

```
git config --global user.name "Your Name"
git config --global user.email "the e-mail of your GitHub account"
```

No time left in class? Then do it at home, with these steps or the git cheatsheet.

Today, and next week

You can now

- find each ODD element of a model in its code
- write a submodel from its rule in the ODD
- give a coding agent the ODD as its brief, and check what it makes

Before next Thursday

- [ ] Your brake rule is committed and pushed.
- [ ] README task 3: at how many cars does the jam start?
- [ ] If you like: let a coding agent draw the space-time diagram. Then play with the agent and see what happens.

Video **The code of the traffic model, explained.** It replaces the cancelled session of 22 October: watch it at home. The course page links it in that row once it is out.

Before lesson 7: functions and dataclasses

A function

```python
def faster(speed):
    return speed + 1

v = 3
w = faster(v)
print(v, w)   # 3 4
```

`def` gives a block of code a name. `speed` is its parameter, a new label for the value it gets. `return` hands the result back, and `v` stays as it was.

A dataclass

```python
from dataclasses import dataclass

@dataclass
class Car:
    position: int
    speed: int = 0

car = Car(position=3)   # speed 0
car.speed = 2
print(car.position, car.speed)   # 3 2
```

`car.speed` reads or changes one of the car's fields.

Both are in your `model.py`. Lesson 7 opens with Python challenges on them, for participation points (log in with GitHub). More in the Python cheatsheet.

Sources for this lesson

Railsback, S. F. & Grimm, V. (2019). *Agent-Based and Individual-Based Modeling: A Practical Introduction.* 2nd ed.

Nagel, K. & Schreckenberg, M. (1992). A cellular automaton model for freeway traffic. *Journal de Physique I* 2(12).

Sugiyama, Y. et al. (2008). Traffic jams without bottlenecks: experimental evidence for the physical mechanism of the formation of a jam. *New Journal of Physics* 10, 033001.