```
.. _checkpoint_traffic:

Traffic
=======

Aim
^^^

This checkpoint requires you to code a simple cellular automaton which attempts to model
traffic flow. This is an example where an exact theory is not known, so we invent a
theory that has the basic properties that we believe are important. By running
simulations we can look for emergent phenomena, in this case the onset of congestion
(traffic jams).

The model
^^^^^^^^^

The simulation box is a straight line of :math:`N` cells (the road) which can each only
have two values: 1 if a car is present on that section of road, 0 otherwise. The *update
rules* for each iteration are very simple:

    - If the space in front of a car is empty then it moves forward one cell;
    - Otherwise it stays where it is.

If we use :math:`c(j)` to indicate the state of the :math:`j` th cell, and use a
subscript :math:`n` to represent the iteration, we can write down rules that determine
:math:`c_{n+1}(j)` from values at the previous iteration :math:`c_n(j-1)`, :math:`c_n(j)`
and :math:`c_n(j+1)`.

    - if :math:`c_n(j) = 1`

        - if :math:`c_n(j+1) = 1`

            - then :math:`c_{n+1}(j) = 1`

        - else :math:`c_{n+1}(j) = 0`

    - else if :math:`c_n(j) = 0`

        - if :math:`c_n(j-1) = 1`

            - then :math:`c_{n+1}(j) = 1`

        - else :math:`c_{n+1}(j) = 0`


Checkpoint task
^^^^^^^^^^^^^^^

Write an object-oriented Python program to simulate the movement of cars on a one-
dimensional road according to the rules defined above. The road has periodic boundary
conditions, i.e. moving forward from the last cell of the road takes you back to the
first cell.

Your code should:

 1. prompt the user for the number of cells :math:`N` (i.e. the length of the road), the
number of iterations (i.e. the number of timesteps) and the car density (i.e. the
fraction of cells that have a car on them). It should then initialise the simulation by
placing cars in random positions on the road.

  .. collapse:: Initialisation, car density and number of cars

     One way to place cars randomly on the road is to iterate over all of the cells in
the road and determine whether or not to place a car on a cell by generating a random
number and comparing it to the car density (this as the same approach that we used in
Checkpoint 2 to randomly select undecayed nuclei to decay). Because this is a statistical
process, the number of cars on the road, and hence the car density, is likely to vary
slightly each time the simulation is run. In addition, depending on the input values of
```

```
car density and number of cells, it may not be possible to match the input value of the
car density exactly as - self-evidently - the number of cars must be an integer. For
example, a road of 25 cells cannot have a car density of 0.5, as this would mean placing
12.5 cars on the road!

     In this checkpoint, neither of these limitations is problematic, but it does mean
that you will need to determine the actual ('experimental') total number of cars in your
simulation in order to obtain the average speed (see below), rather than calculating this
from input value of car density.

 2. update the road for the given number of timesteps. At each timestep, the code should
compute and print the average speed of the cars.

  .. collapse:: What does 'average speed' mean?

     You should find that the average speed of the cars changes over the first few
timesteps, but that once the movement of the cars reaches steady state the average speed
remains constant, i.e. is the same at each timestep. This is the 'steady state' average
speed.

     At the end of the simulation, your code should graphically represent the positions
of the cars on the road as a function of time. This representation does not need to be
animated, although you may use animation if you wish.

     Finally, use your code to determine the steady state average speed for a range of
car densities between zero, for a completely empty road, to one, for a completely jammed
road. What does this tell you about the onset of congestion (traffic jams)?

  .. collapse:: How to determine the steady state average speed?

     One simple way to determine the steady state average speed is to run the simulation
until it is in equilibrium and then take the average speed for the final timestep. You
may find your graphical representation useful for checking whether the simulation has
reached equilibrium.

     You may use a more sophisticated approach if you wish, but this is not required as
part of the checkpoint code.

     The number of timesteps needed to reach equilibrium depends on the density of cars
and their initial positions on the road. However, the dynamics of traffic cellular
automata are very complex (and beyond this course) and in general do not have exact
solutions.

 3. plot average speed at the steady state as a function of the car density. It is
acceptable to accomplish this task with a separate test code. That is, you may write one
test code that runs the main codebase in a way that takes user input from the terminal to
run with a single configuration and another test code that runs with many configurations
to create the plot. However, to receive full marks, both test codes must run the same
codebase (i.e., they must use the same classes, not two similar versions of them). All
plotting code should be written using ``matplotlib`` and included as part of the
checkpoint submission, although the plotting code is not required to be done in object-
oriented style.

Marking Scheme
^^^^^^^^^^^^^^

    - :ref:`assessment_traffic`

.. admonition:: Note!

   The code you submit for assessment **must be your own work**, not the output of
Generative AI or the work of another person. The course team will use their best
judgement to determine whether this is true. See the discussion in the
:ref:`checkpoint_marking_scheme` and the Generative AI Policy on the course Learn page
for more information.

Optional extras
^^^^^^^^^^^^^^^
```

```
*Optional extras are not marked and don't count towards the assessment for the
checkpoint.*

    1. Is the model sensitive to the initial configuration of the roadway? You might try
to implement very different starting conditions, such as random 'clumps' of cars, rather
than simple random cars. Does this affect the onset of congestion on the road?

    2. The update rules are very simple in this model. Extend your code so that cars move
forward:

        - with a probability of 0.75 if they moved last iteration
        - with a probability of 0.5 if they did not

      Does this affect the flow of traffic at a given density?

    3. Bad driving. If you initiate an 'event' on the road, for example an obstacle that
moves forward with a very low probability or stays fixed for a few iterations before it
disappears, you should be able to see the resulting queue of traffic ripple backwards
along the road like a wave and get longer.

      .. collapse:: Note

          In real life, this 'event' may correspond to someone changing lanes without
indicating causing the driver behind to brake suddenly. Quite often, slow-moving or even
stationary traffic can form from one of these events.


Relevant course sections
^^^^^^^^^^^^^^^^^^^^^^^^

Studying the following course sections will help you complete this checkpoint:

    - :ref:`plotting`

    - :ref:`custom_plotting`

Additional material that you may find useful:

    - :ref:`animation`
```